

**Project Title:**

Pop-ify: Song Popularity Prediction using Decision Tree  
Regression

**Members:** Lance Chrysler Cabana, Edrison Camarillo, Miguelle Edrienne Dingcon,  
Ricki Reinelle Macapagal, Nina Erika Patricia Satur, John Justin Verendia

## Table of Contents

<b>Title Page.....</b>	<b>1</b>
<b>Table of Contents.....</b>	<b>2</b>
<b>I. Introduction.....</b>	<b>4</b>
Background:.....	4
Objectives:.....	5
<b>II. Literature Review.....</b>	<b>6</b>
Prediction in Song Popularity.....	6
Prediction using Decision Tree Regression.....	7
<b>III. Data Source.....</b>	<b>10</b>
<b>IV. Implementation to Python.....</b>	<b>12</b>
File 1: Data Pre-processing.....	12
File 2: Exploratory Data Analysis.....	14
Genre Analysis (Pie Chart).....	15
Outlier Analysis (Boxplot).....	16
Correlation Matrix (Heatmap).....	17
File 3: Feature Selection.....	18
File 4: Decision Tree Regressor.....	21
Model Creation.....	22
Model Evaluation.....	24
Data Visualization.....	25
Export Model.....	25
<b>V. Features of Software.....</b>	<b>26</b>
A. Song Popularity Prediction.....	27
B. Model Training.....	35
<b>VI. Conclusion.....</b>	<b>45</b>
Findings.....	45
Conclusion.....	46

## **I. Introduction**

Section I is about the introduction of the project. This consists of the background of the area of interest, as well as the objectives needed to be accomplished.

### **Background:**

Music production is a career that feeds the entertainment industry, as technological advancements throughout the years made music widely available. Take Spotify as an example; it is accessible in 183 countries, where 80+ were just added in 2020, not to mention its massive audience of over 456 million active users worldwide at the start of 2022, which is 16.44% more than just the previous year (Shepherd, 2023).

Unfortunately, only a few of the hundreds of annually produced songs make it to the charts (Essa et al., 2022). With the big competition in the music industry, considering the extensive data readily available on the internet, knowing trends using approaches in data mining could gain a competitive advantage. There now exists music analysis that paved the way to accessing a song's metadata, knowing relevant features such as artists, genres, beats per minute, and more (Essa et al., 2022).

Song popularity prediction is especially important in preserving businesses competitive within the growing music industry (Kyauk et al., 2015). Thus, this project would cover the possibility of predicting song popularity through identifiable audio features and exploring which predictor has more influence on song popularity.

**Objectives:**

This project aimed to implement a data mining algorithm in song popularity prediction, specifically, the decision tree regression. Additionally, the project aimed to perform Exploratory Data Analysis (EDA) on the Spotify Songs dataset and to create a simple system integrating the prediction model suitable for dynamic uses. Lastly, the project aimed to test the performance of the model in predicting the popularity of the song using Mean Square Error (MSE) and Root Mean Square Error (RMSE).

## II. Literature Review

Section II covers the discussion of the related literature of the project. A total of 4 related studies are presented, separated by the application of decision tree regressor in song popularity prediction and other areas.

### Prediction in Song Popularity

Numerous studies focus on song popularity prediction and regression models. The following related literature supports the methodology and insights of this project.

The study by Essa et al. (2022) entitled *Predicting the Song Popularity Using Machine Learning Algorithm* proved the possibility of predicting if a song is popular by means of certain attributes when trained with machine learning. Specifically, the authors performed Exploratory Data Analysis and compared models using classification, regression, and ensemble learning. They made sure to observe features of audio signal-related data such as acousticness, energy, liveliness, danceability, valence, count, instrumentalness, loudness, tempo, mode, and speechiness of the music from a generated dataset from the Spotify web Application Programming Interface (API). The Kaggle dataset consists of 198,000 songs. The result showed excellent performance of Decision Trees algorithms — binary trees — in regression and classification. The models were tuned with the optimal hyperparameter values: Regression has a maximum depth of 10, minimum samples to split of 6, and minimum samples required at the leaf of 7, while classification has 10, 12, and 1, respectively. The Decision Tree

regressor yielded RMSE values of 7.88407 and 8.30137 for the train and test data, respectively.

Another study in the field of song popularity prediction was made by Kyauk et al. (2015) entitled *Predicting Song Popularity*. Regression and classification algorithms were applied with the popular dataset in this area of study — music data from The Million Song Dataset with songs up to 2011. This particular study used Echo Nest's social field known as “song hottnesss” for the popularity metric. After feature selection, the authors saw that the following features were consistent among the dataset: artist familiarity, timbre\_mean, loudness 2, and loudness. Other features included in the study were: start\_of\_fade\_out, tempo2, duration2, tempo, tatum\_var, tatum\_mean, and year. Although the MSE and RMSE values of the compared regressors – baseline, full model, forward stepwise selection, backward stepwise selection, and lasso regression — had very little disparity, the backward stepwise selection had the best results of 0.0176 and 0.1327, respectively. The authors also mentioned the possibility of using the number of downloads on iTunes or the number of plays on Spotify as other popularity metrics.

### **Prediction using Decision Tree Regression**

Other studies also made use of a Decision Tree regressor as a predictive algorithm in varying fields. The following studies serve as related literature on the usage of similar algorithms for the same purpose:

The work of Brunda et al. (2020), entitled *Crop Price Prediction Using Random Forest and Decision Tree Regression* made use of decision tree regression. The main problem that the authors encountered was the instability of agricultural commodity prices, which could have a negative impact on the economy. Farmers can plant crops in accordance with the projected costs with the help of successfully predicting crop prices. The data was collected from Open Government Data (OGD) India, which consists of 330 different crops from 2011 to 2018. There are a total of 8,68,966 entries with four attributes: index, item name, date, and price. 80% of the data was used for training, and the remaining 20% was for testing. With a training period that lasted a total of 1.67 seconds, the Decision Tree Regressor was able to achieve an accuracy of 99.08%. Decision trees seem to fit well with the data; however, they are sensitive to the particular data on which they were trained. The model also required a lot of hyperparameter tuning, which may lead to overfitting.

Another study that utilized decision tree regression is *Stock Market Analysis Using Linear Regression and Decision Tree Regression* by Karim et al. in 2021. The goal of the study is to make use of two supervised regression machine learning algorithms in order to make better decisions, along with a statistical formula to provide better accuracy in predicting stock prices. They made use of a recent one-year dataset from Amazon that was then separated into three parts: one year (100%), six months (50%), and three months (25%). The prepared data set was separated by 80% for training and 20% for testing. The model resulted in an accuracy of 0.9936, or 99.4%, for the one-year dataset. The dataset was then divided into two for the six-month data,

which resulted in 0.9088 or 90.9% accuracy in stock price prediction. In the three-month dataset, which is half of the previous dataset, the accuracy of the prediction is only 0.8353, or 83.5%. The result of the experiment suggests that the accuracy of the prediction using decision tree regression lowers as the size of the dataset decreases.



### III. Data Source

Section III is dedicated to discussing the dataset used in the project. This section contains information on the data source, data gathering methods, and the dataset's metadata.

The dataset used in this project is from the “Spotify Past Decades Songs Attributes” dataset on Kaggle. The creation of this data repository consisting of seven (7) datasets is from the study of Leonardo Henrique. Song data from the dataset mirrors Spotify's “All Out” playlists per decade, from 1950s up to 2010s — inclusive of popular songs for that period. The original contributors scraped the data from the “Organize Your Music” website with a total of 667 records and 15 attributes.

**Table 1. Song Metadata**

Field Name	Data Type	Description
number	int	ID of the track
title	varchar	Title of track
artist	varchar	Name of singer or band
top genre	varchar	Genre of the track
year	int	Release year (or rerelease)
bpm	int	Beats per minute (tempo)
nrgy	int	Energy of a song - higher value corresponds to more energetic song
dnce	int	Danceability; higher value corresponds to easiness to dance to the song
dB	int	Loudness; higher value corresponds to louder song

live	int	Liveness; higher value corresponds to the likelihood of song being a live recording
val	int	Valence; higher value corresponds to more positive mood of the song
dur	int	Duration; the length of the song
acous	int	Acousticness; higher value corresponds to the likelihood of the song being acoustic
spch	int	Speechiness; higher value corresponds to more spoken word in the song
pop	int	Popularity; higher value corresponds to the likelihood of the song being more popular

Table 1 shows the metadata of the tracks from Kaggle’s Spotify Past Decades Songs Attributes dataset. Aside from number, title, artist, top genre, and year, the rest of the features are audio signal-related data already extracted through proprietary algorithms.

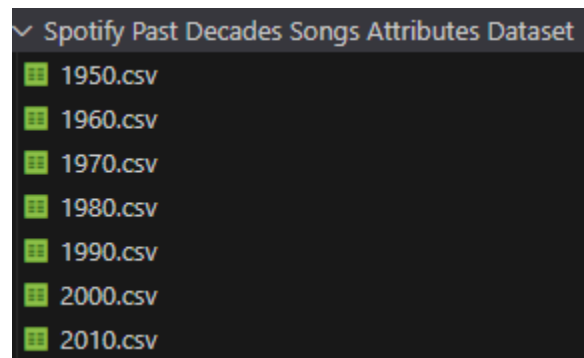
As part of the demonstration of the final system, there is a need to collect data using the same methods as Henrique’s. Web scraping of song information was performed, targeting the Spotify playlist “All Out 2020s.” A total of 127 songs were collected.

## IV. Implementation to Python

Section IV consists of screenshots of the implementation of the project using Python. There are a total of four files with their own respective uses. The first file — Data Pre-processing — is focused on aggregating the files from the data source, as well as handling missing data values. The second file — Exploratory Data Analysis — examines the dataset, which includes genre analysis, outlier analysis, and correlation analysis. The third file — Feature Selection — made use of another model to identify the most relevant features with respect to the target value. The last file — Decision Tree Regressor — consists of the model creation, model evaluation, and model export to be used by the software — "Pop-ify."

### File 1: Data Pre-processing

Datasets



\*Kaggle dataset provided a total of 7 .csv files

Import Library

```
import pandas as pd
```

## Data Integration

```
# store csv filenames in list
decades = '1950 1960 1970 1980 1990 2000 2010'.split()

# initial list to read datasets
datasets = [[] for i in range(len(decades))]

# read the datasets
for i in range(len(decades)):
    datasets[i] = pd.read_csv(f"Spotify Past Decades Songs Attributes Dataset/{decades[i]}.csv")
```

```
# transform dataset into DataFrame
df_songs = pd.concat(datasets)
```

## Handling Missing Values

```
# check for missing values
df_songs.isna().sum()
```

\*`isna()` function is used to check for empty cells and `.sum()` aggregates the number of empty cells per column

### Output:

```
Number      0
title       0
artist      0
top genre   16
year        0
bpm         0
nrgy        0
dnce        0
dB          0
live        0
val         0
dur         0
acous       0
spch        0
pop         0
dtype: int64
```

```
# drop rows with missing values
df_songs.dropna(inplace=True)
```

\*`dropna()` function drop the rows containing missing data

## Export Dataset

```
df_songs.to_csv('dataset.csv', index=False)
```

## File 2: Exploratory Data Analysis

### Import Libraries

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
dataset = pd.read_csv('../Model/dataset.csv')
```

```
dataset.info()
```

\*info() function displays descriptive details about the dataframe

### Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 651 entries, 0 to 650
Data columns (total 15 columns):
 #   Column      Non-Null Count  Dtype
---  -
 0   Number     651 non-null   int64
 1   title       651 non-null   object
 2   artist      651 non-null   object
 3   top genre   651 non-null   object
 4   year        651 non-null   int64
 5   bpm         651 non-null   int64
 6   nrgy        651 non-null   int64
 7   dnce        651 non-null   int64
 8   dB          651 non-null   int64
 9   live        651 non-null   int64
10   val         651 non-null   int64
11   dur         651 non-null   int64
12   acous       651 non-null   int64
13   spch        651 non-null   int64
14   pop         651 non-null   int64
dtypes: int64(12), object(3)
memory usage: 76.4+ KB
```

```
print(dataset['top genre'].nunique())
```

Output: 115

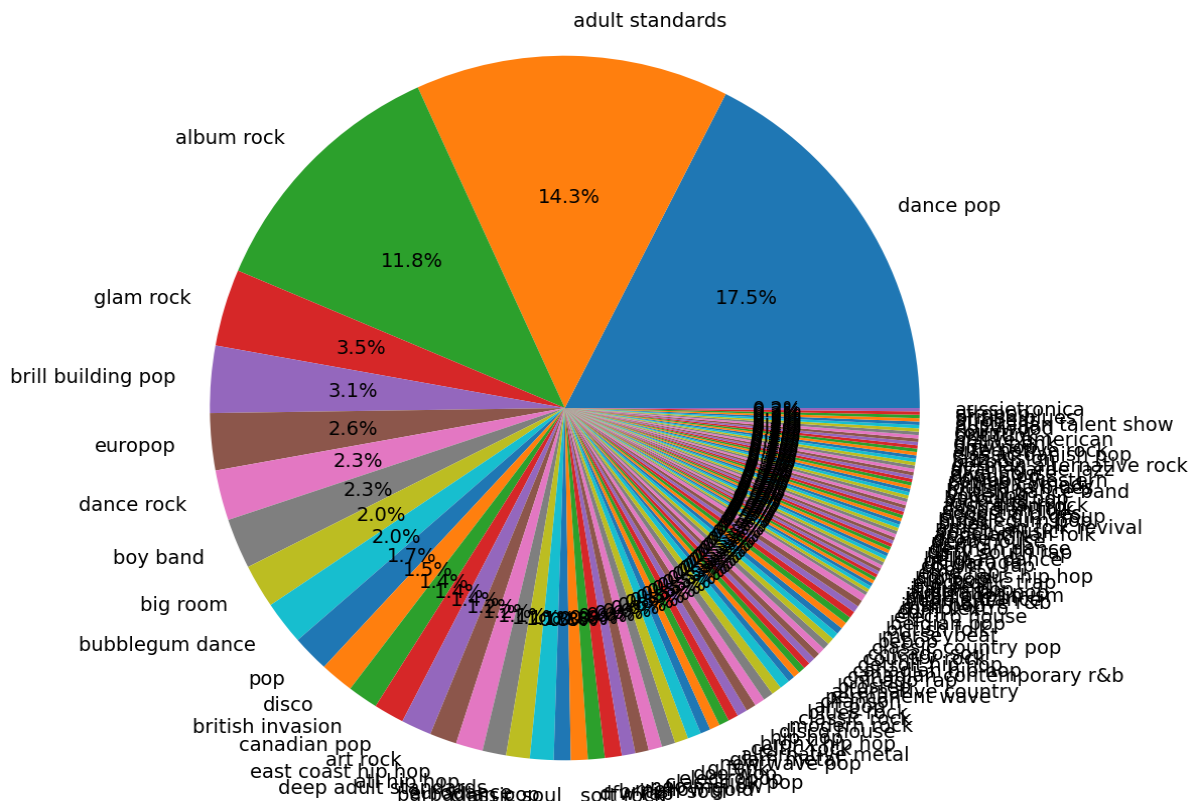
```
dataset.drop(['Number', 'artist', 'title', 'year'], axis=1, inplace=True)
```

## Genre Analysis (Pie Chart)

```
# Find percent of each genre
dataset_genre = dataset['top genre'].value_counts() / len(dataset)
sizes = dataset_genre.values.tolist()
labels = dataset_genre.index.values.tolist()

# Pie chart for genre
fig1, ax1 = plt.subplots(figsize=(10,10))
ax1.pie(sizes, labels=labels, autopct='%1.1f%%', shadow=False, textprops={'fontsize': 14})
ax1.axis('equal')
plt.show()
```

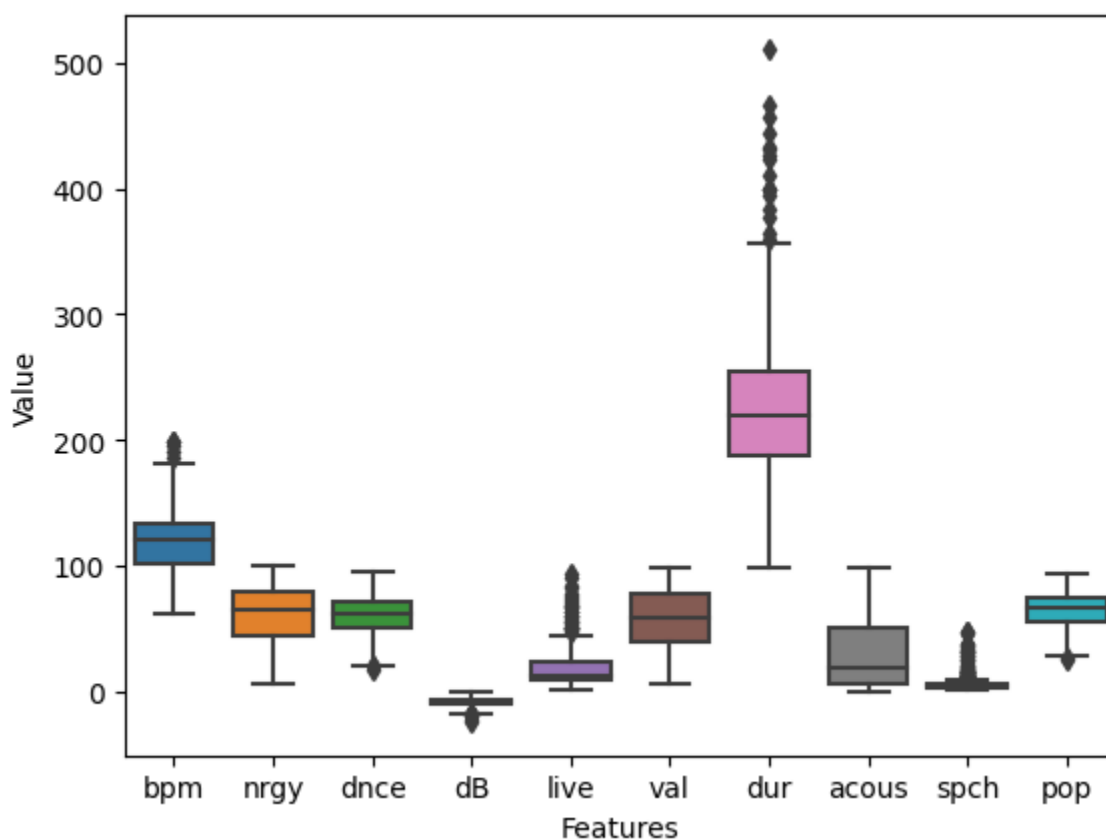
Output:



## Outlier Analysis (Boxplot)

```
# Plot boxplot (variables only)
sns.boxplot(data=dataset.drop(['top genre'], axis=1))
plt.xlabel('Features')
plt.ylabel('Value')
plt.show()
```

Output:



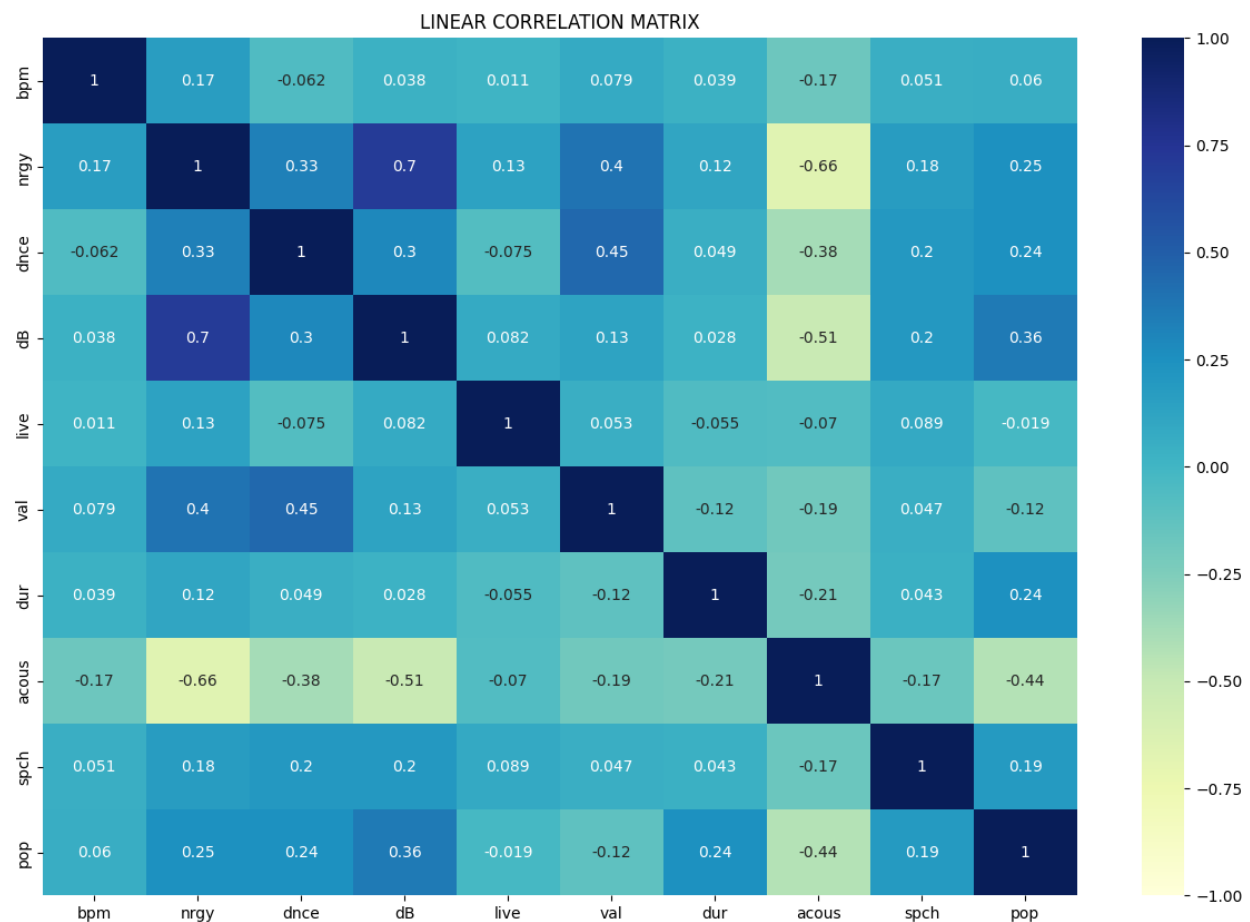
In performing the outlier analysis, the outliers can be identified and analyzed to determine if they are erroneous in nature. Technically, the audio signal-related data were generated through the website's proprietary algorithms, implying a low possibility of erroneousness among the data; hence, they are not removed from the dataset. The

boxplot graph provides an indication of how spread out the values in the dataset are and highlights the presence of outliers.

### Correlation Matrix (Heatmap)

```
# Plot linear correlation matrix
fig, ax = plt.subplots(figsize=(15,10))
sns.heatmap(dataset.drop('top genre', axis=1).corr(), annot=True, cmap='YlGnBu', vmin=-1, vmax=1, center=0, ax=ax)
plt.title('LINEAR CORRELATION MATRIX')
plt.show()
```

Output:



The correlation matrix shows the behavior of two variables in relation to each other. Negative correlation value means if one variable increases, the other decreases.



### File 3: Feature Selection

#### Import Libraries

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error as mse
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.metrics import accuracy_score as acc
import matplotlib.pyplot as plt
```

```
dataset = pd.read_csv('dataset.csv')
```

#### Pre-processing

```
# drop unnecessary features
drops = ['Number', 'title', 'artist', 'top genre']

for drop in drops:
    dataset = dataset.drop(drop, axis=1)
```

#### Data Splitting

```
# X for features and y for target
X = dataset.drop('pop', axis=1)
y = dataset['pop']

# split into testing and training datasets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

#### Model Training

```
# Feature selection by ExtraTreesRegressor(model based)
regressor= ExtraTreesRegressor()

regressor.fit(X_train, y_train)
```

**\*ExtraTreesRegressor()** implements Extremely Randomized Trees ensemble method — extension of the Random Forest algorithm

```
regressor.feature_importances_
```

`feature_importances_` property accesses the feature importances of the model and returns an array of feature importance scores.

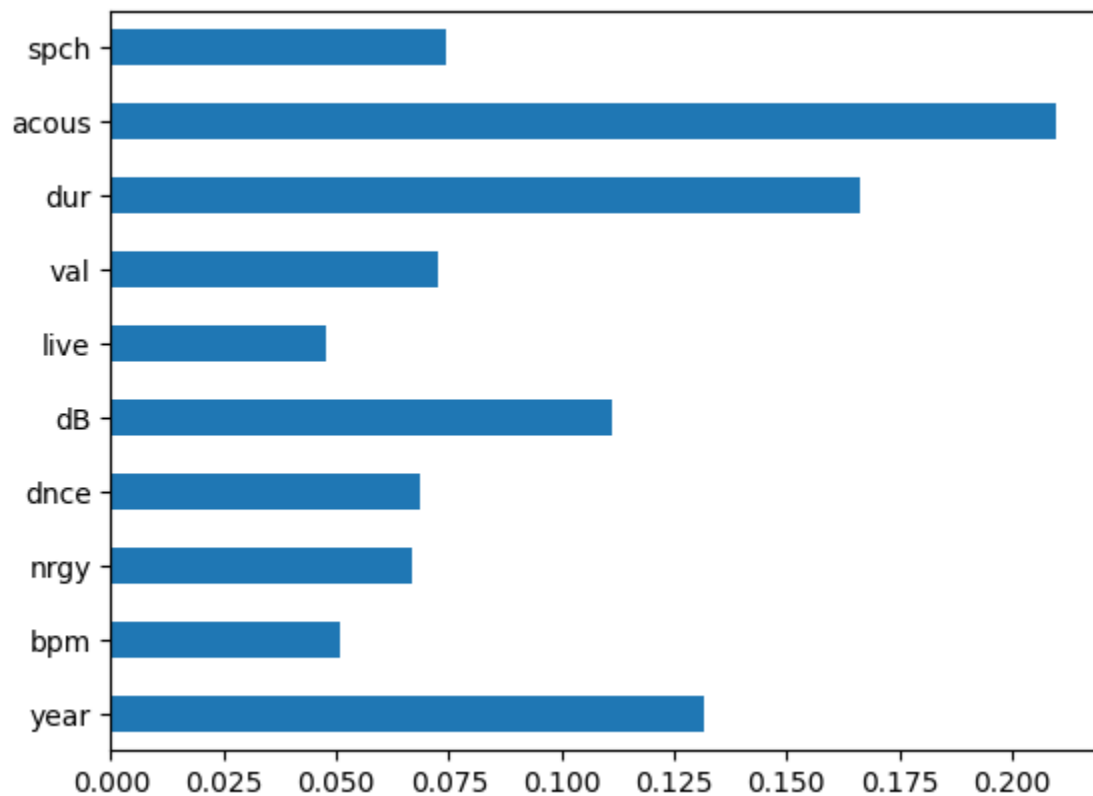
Output:

```
array([0.1315101 , 0.05100388, 0.06679683, 0.0688105 , 0.11131056,  
       0.04766263, 0.07259804, 0.16620681, 0.20966236, 0.07443828])
```

Plotting - For All Features

```
feat_importances = pd.Series(regressor.feature_importances_, index=X_train.columns)  
feat_importances.plot(kind='barh')  
plt.show()
```

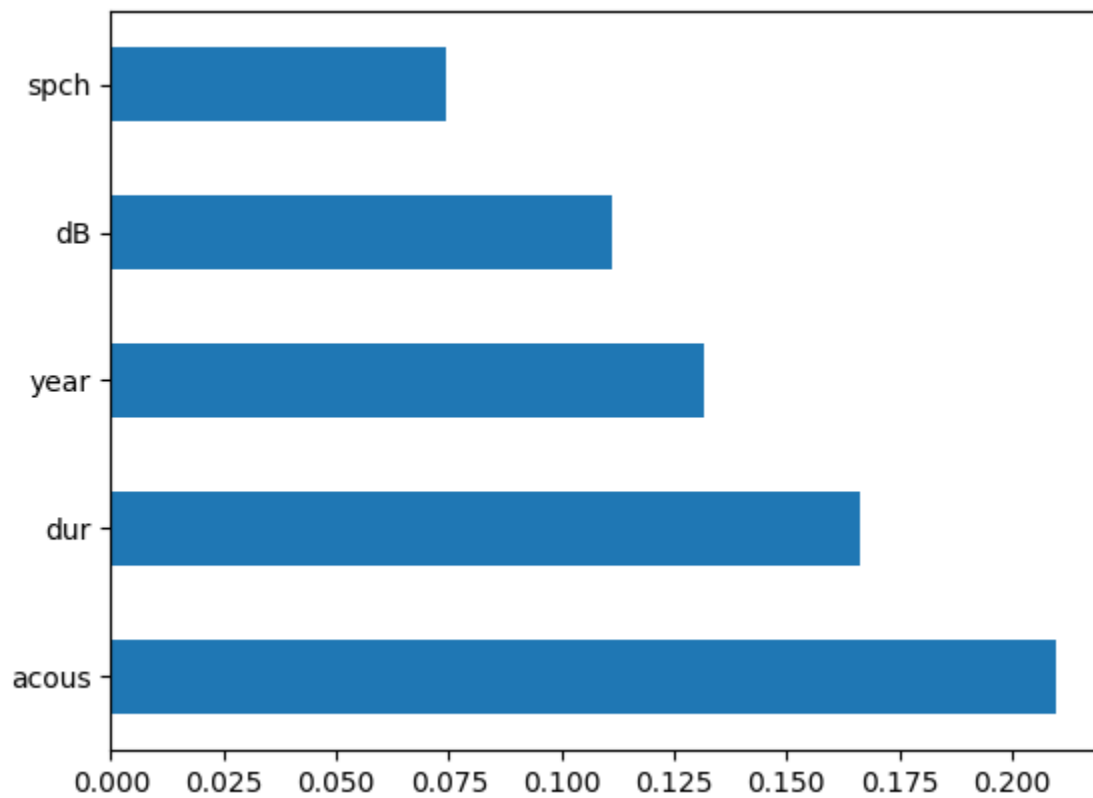
Output:



## Plotting - For Top 5 Features

```
feat_importances.nlargest(5).plot(kind='barh')  
plt.show()
```

Output:



## File 4: Decision Tree Regressor

### Import Libraries

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV, cross_val_score
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
from sklearn import tree
import joblib
```

Python

```
dataset = pd.read_csv("dataset.csv")
```

### Pre-processing

```
# drop unnecessary features
drops = ['Number', 'title', 'artist']

for drop in drops:
    dataset = dataset.drop(drop, axis=1)
```

### Data Transformation

```
# list of categorical columns
categorical_cols = ['top genre']

# initialize the OneHotEncoder
encoder = OneHotEncoder(sparse_output=False)

# fit and transform the selected categorical columns
encoded_features = encoder.fit_transform(dataset[categorical_cols])
```

\*One Hot Encoder was used to transform categorical columns into numerical,  
the resulting dataset now consists of 126 columns.

```
# create a new DataFrame with the encoded features
encoded_df = pd.DataFrame(encoded_features, columns=encoder.get_feature_names_out(categorical_cols))
```

Python

```
# remove the original categorical columns from the original DataFrame
dataset.drop(columns=categorical_cols, inplace=True)

# Concatenate the encoded DataFrame with the original DataFrame
dataset = pd.concat([dataset, encoded_df], axis=1)
```

## Data Selection

```
# X for features and y for target
X = dataset.drop('pop', axis=1)
y = dataset['pop']
```

## Feature Normalization

```
# perform feature normalization using min-max scaling method
# Create an instance of MinMaxScaler
scaler = MinMaxScaler()

# Fit the scaler to the data and transform the features
X_scaled = scaler.fit_transform(X)
```

## Data Splitting

```
# split into testing and training datasets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

## Model Creation

### Model Training

```
# initialize the Decision Tree Regressor
regressor = DecisionTreeRegressor()
```

## Hyperparameter Tuning

```
param_grid={"splitter":["best","random"],
            "max_depth" : [1,3,5,7,9,11,12],
            "min_samples_leaf":[1,2,3,4,5,6,7,8,9,10],
            "min_weight_fraction_leaf":[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9],
            "max_features":["auto","log2","sqrt",None],
            "max_leaf_nodes":[None,10,20,30,40,50,60,70,80,90],
            "random_state": [5]
}

# perform grid search
grid_search = GridSearchCV(regressor, param_grid, scoring='neg_mean_squared_error', cv=5, verbose=3)
grid_search.fit(X_train, y_train)
```

Python

\*Grid Search CV algorithm was used to find the optimal set of parameters,  
the search takes 3-5 mins as cross-validation is set to 5.

```
# display best parameters
best_params = grid_search.best_params_
print("Best Parameters:", best_params)
```

Output: Best Parameters: {'max\_depth': 5, 'max\_features': 'auto',  
'max\_leaf\_nodes': None, 'min\_samples\_leaf': 1, 'min\_weight\_fraction\_leaf':  
0.1, 'random\_state': 5, 'splitter': 'best'}

```
# initialize tuned regressor
tuned_regressor = grid_search.best_estimator_
```

```
# fit the training datasets
tuned_regressor.fit(X_train, y_train)
```

Output:

```
DecisionTreeRegressor
DecisionTreeRegressor(max_depth=5, max_features='auto',
                      min_weight_fraction_leaf=0.1, random_state=5)
```

## Model Evaluation

```
#final_regressor.score(X_test, y_test)

# perform prediction using the test data
y_predict = tuned_regressor.predict(X_test).astype(int)
y_predict
```

Output:

```
array([65, 68, 62, 68, 50, 68, 58, 62, 65, 71, 71, 68, 68, 50, 62, 68, 65,
       72, 50, 50, 71, 58, 72, 50, 62, 68, 65, 65, 71, 50, 50, 50, 68, 68,
       72, 58, 58, 68, 58, 72, 58, 62, 72, 58, 65, 50, 71, 62, 72, 68, 68,
       50, 72, 68, 50, 71, 58, 62, 50, 68, 58, 65, 58, 71, 72, 71, 65, 72,
       58, 58, 58, 68, 50, 62, 65, 50, 68, 58, 50, 68, 72, 62, 68, 65, 72,
       58, 50, 65, 71, 72, 50, 50, 50, 65, 65, 50, 68, 62, 65, 58, 62, 50,
       58, 50, 50, 58, 65, 50, 68, 65, 50, 71, 58, 62, 71, 68, 58, 50, 62,
       62, 65, 71, 72, 58, 68, 72, 65, 72, 62, 50, 62])
```

```
# model score in training dataset
tuned_regressor.score(X_train, y_train)
```

Output: 0.3249045925048163

```
# model score in testing dataset
tuned_regressor.score(X_test, y_test)
```

Output: 0.341217059030823

```
# compute for the Mean Square Error
mse = mean_squared_error(y_test, y_predict)

# compute for the Root Mean Square Error
rmse = np.sqrt(mse)

# display results
print(f"MSE: {mse}")
print(f"RMSE: {rmse}")
```

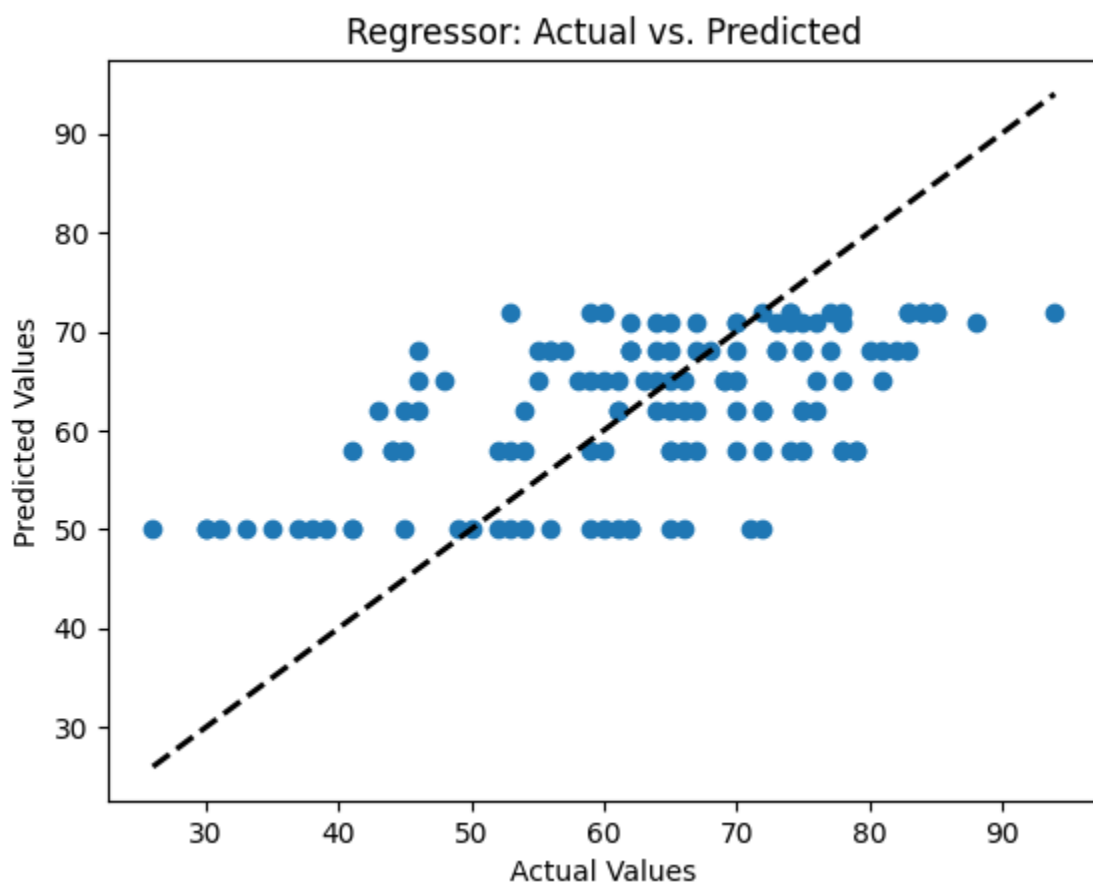
Output: MSE: 128.5801526717557

RMSE: 11.339318880415865

## Data Visualization

```
# plt the predicted values against the actual values
plt.scatter(y_test, y_predict)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2) # diagonal line
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Regressor: Actual vs. Predicted')
plt.show()
```

Output:



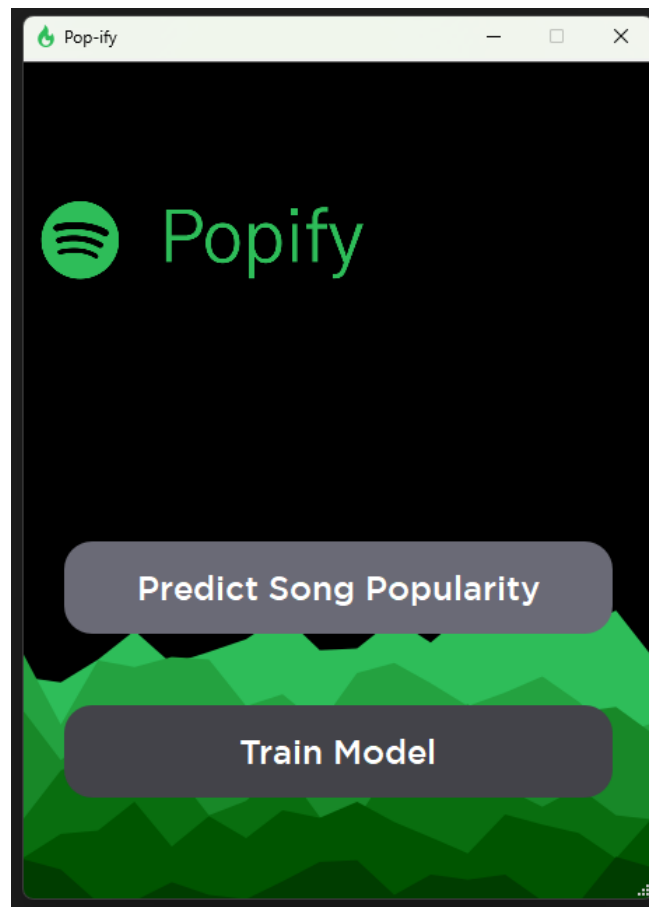
## Export Model

```
joblib.dump(tuned_regressor, "../Decision_Tree_Regressor")
```

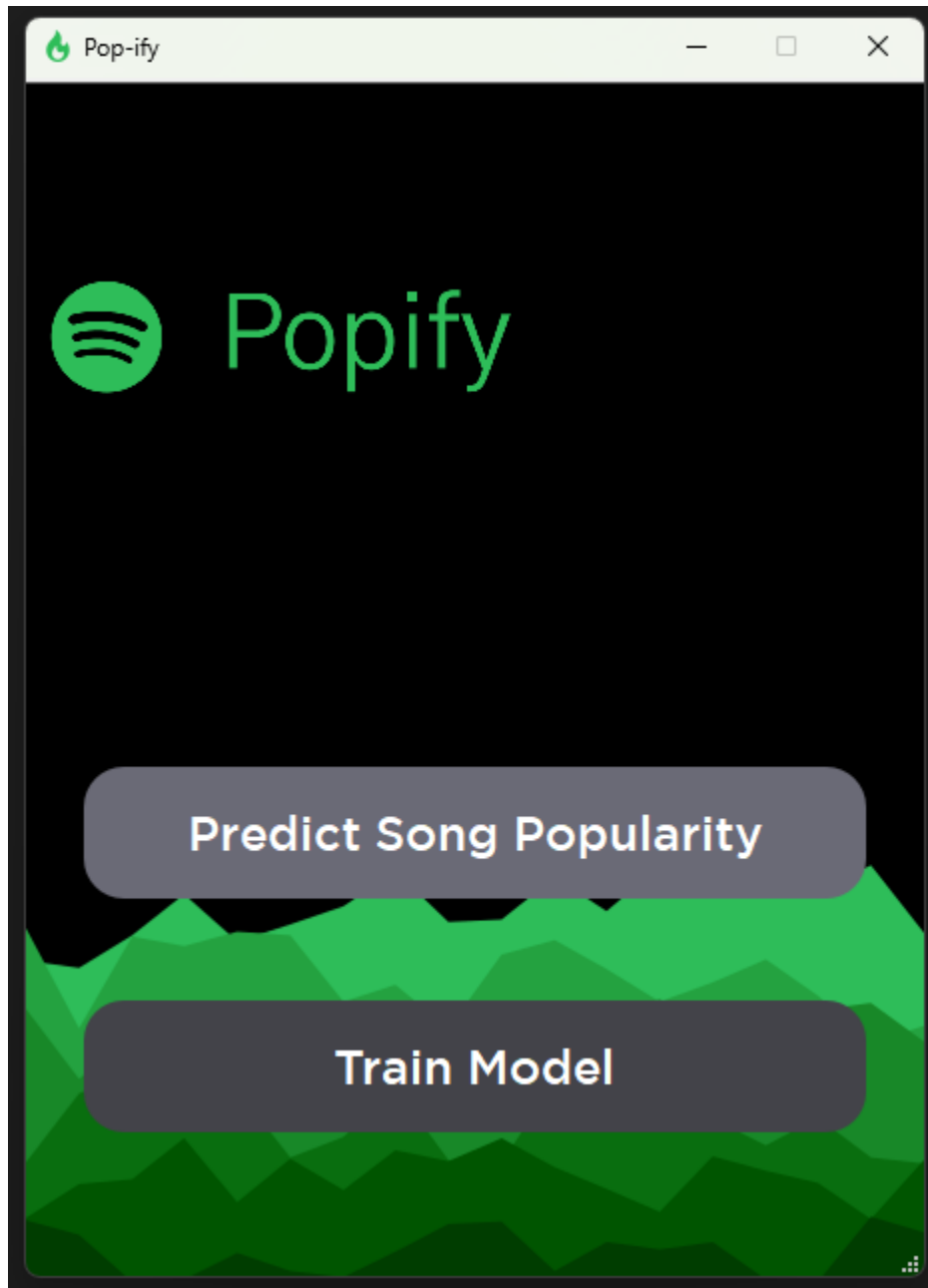


## V. Features of Software

Section V consists of screenshots of the developed software entitled “Pop-ify.” This demonstration also serves as a step-by-step user guide to the software. There are two features implemented in the software: prediction of song popularity and retraining the model. Song popularity prediction asks for user input — predictors — which are features necessary to find the target value — popularity. Model retraining asks for a dataset that is used to retrain the model; additionally, the updated MSE and RMSE value is displayed for the retrained model.




## A. Song Popularity Prediction



\*Click "Predict Song Popularity" Button

Song Popularity Prediction

 **Popify**

**Upload File**

**Title**

**Top Genre**

**Year**

**BPM**

**Energy**

**Danceability**

**Loudness (dB)**

**Liveliness**

**Valence**

**Duration**

**Acousticness**

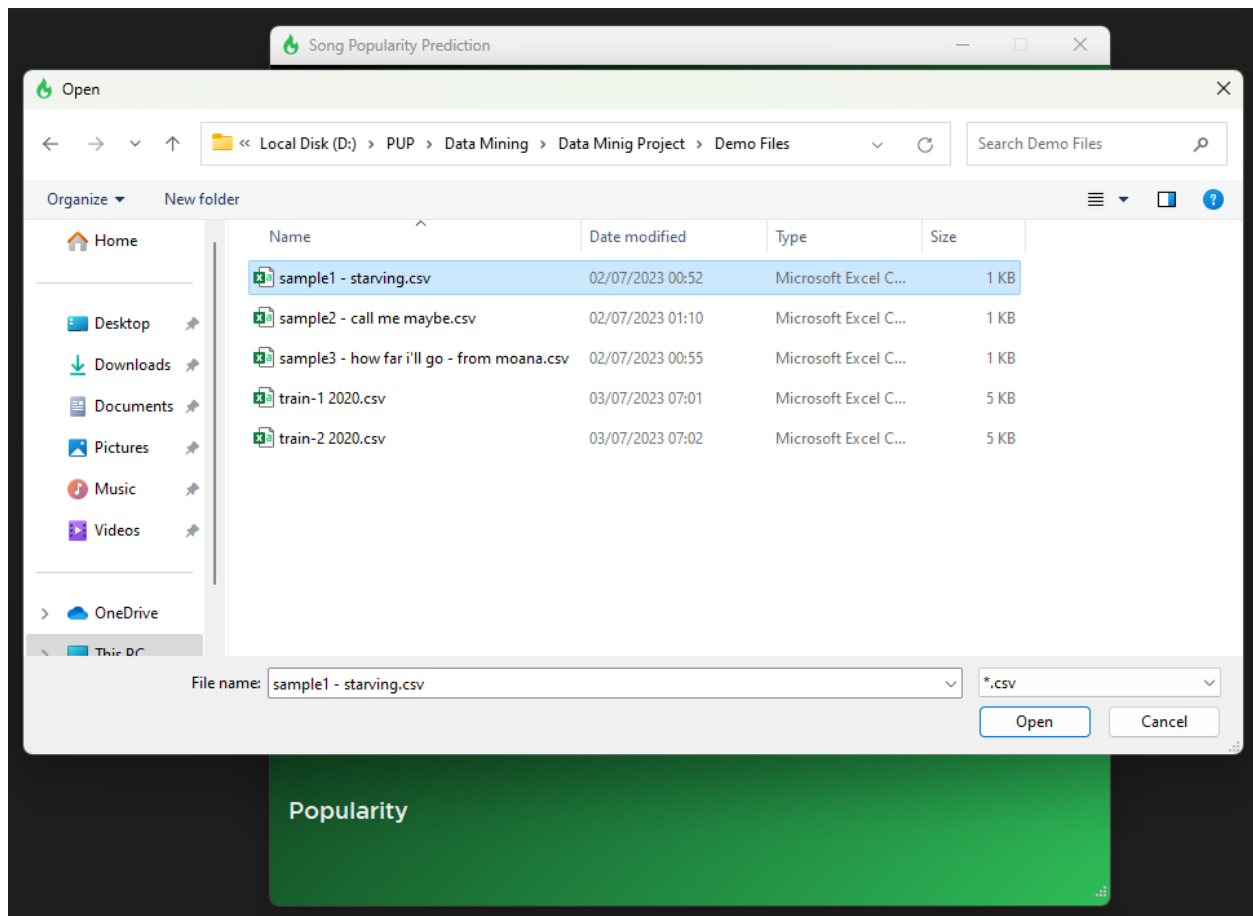
**Speechiness**

**Clear** **Predict**

**Popularity**


\*Upload .csv file containing necessary song information of 1 song

Alternatively, User can input data in the input fields manually.



\*Select desired .csv file then click “Open” Button

Song Popularity Prediction

 Popify


Upload File	D:/PUP/Data Mining/Data Minig Project/Demo Files/sample1 - starving.csv
Title	Starving
Top Genre	pop
Year	2017
BPM	100
Energy	61
Danceability	73
Loudness (dB)	-4
Liveliness	10
Valence	53
Duration	182
Acousticness	37
Speechiness	6

Clear Predict

Popularity

\*Click "Predict" Button

Song Popularity Prediction

 **Popify**

Upload File

Title

Top Genre

Year

BPM

Energy

Danceability

Loudness (dB)

Liveliness

Valence

Duration


Acousticness

Speechiness

**Popularity** **68**

\*Prediction output is displayed next to “Popularity” Label

Song Popularity Prediction

 Popify

Upload File

Title

Top Genre

Year

BPM

Energy

Danceability

Loudness (dB)

Liveliness

Valence

Duration


Acousticness

Speechiness

Popularity **68**

\*Click "Clear" Button to clear fields

Song Popularity Prediction

 Popify

Upload File

Title

Top Genre

Year

BPM

Energy

Danceability

Loudness (dB)

Liveliness

Valence

Duration

Acousticness

Speechiness

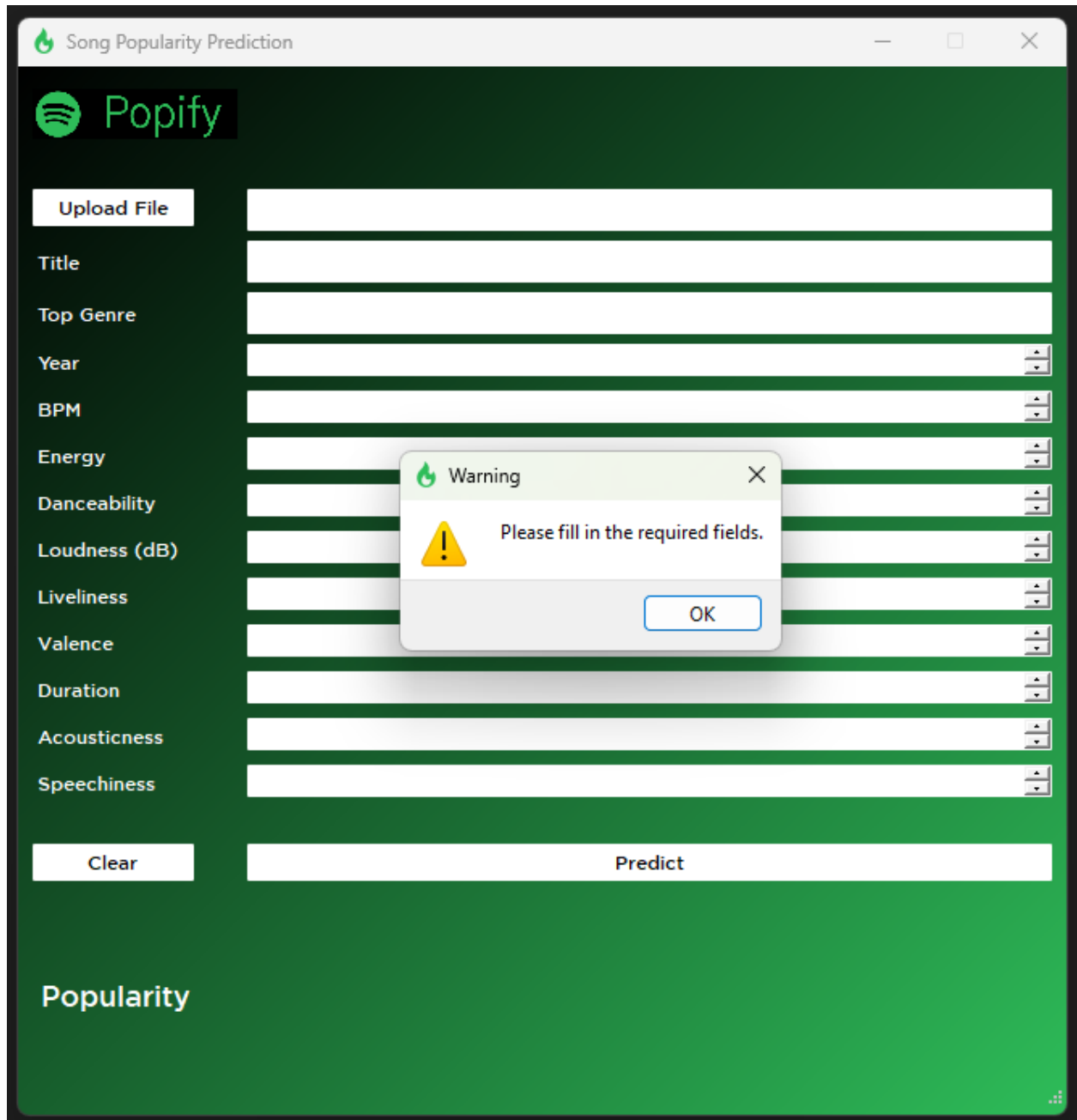
Clear

Predict

Popularity

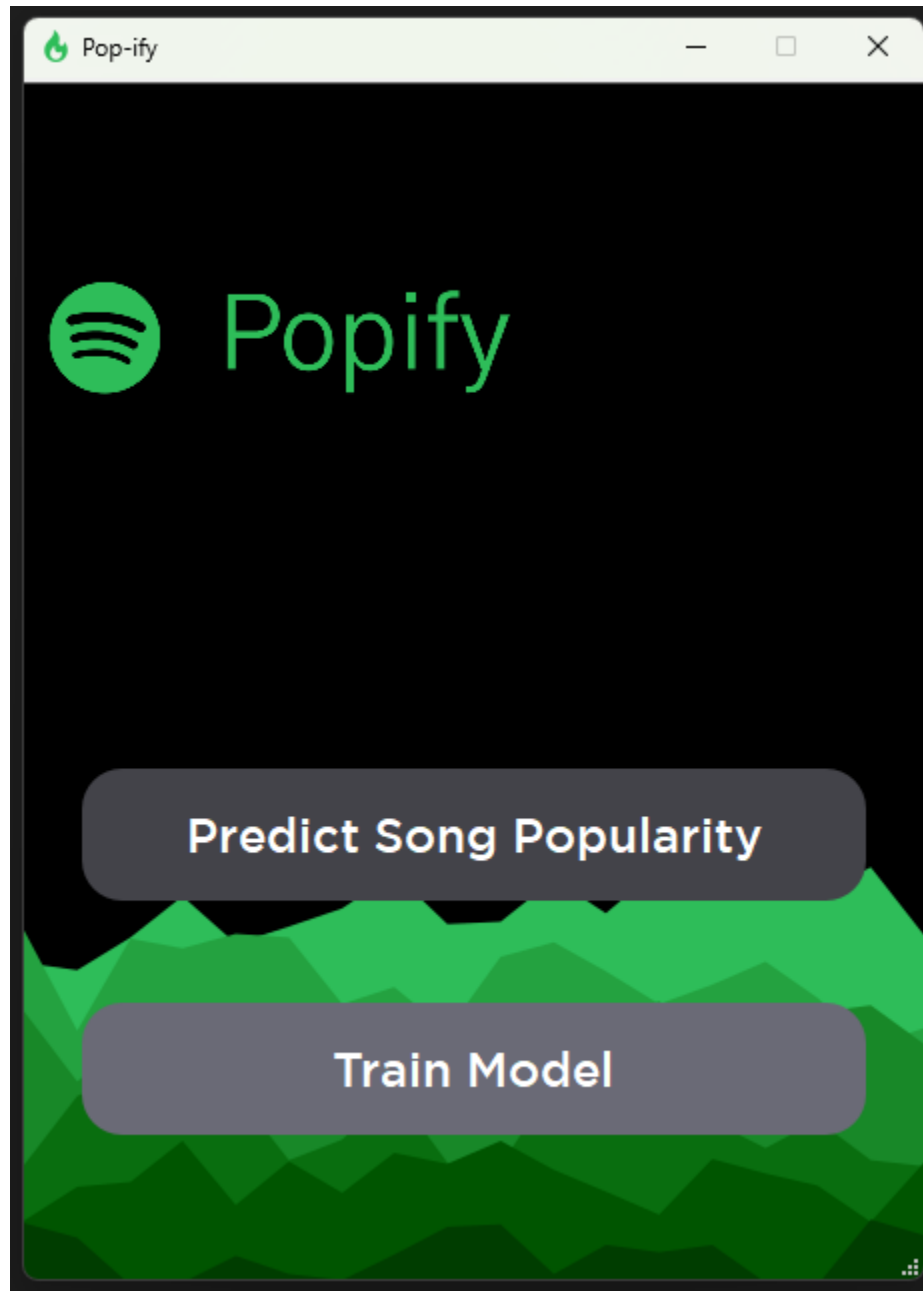
\*Clicking the “Predict” Button without filling the fields will prompt a warning message



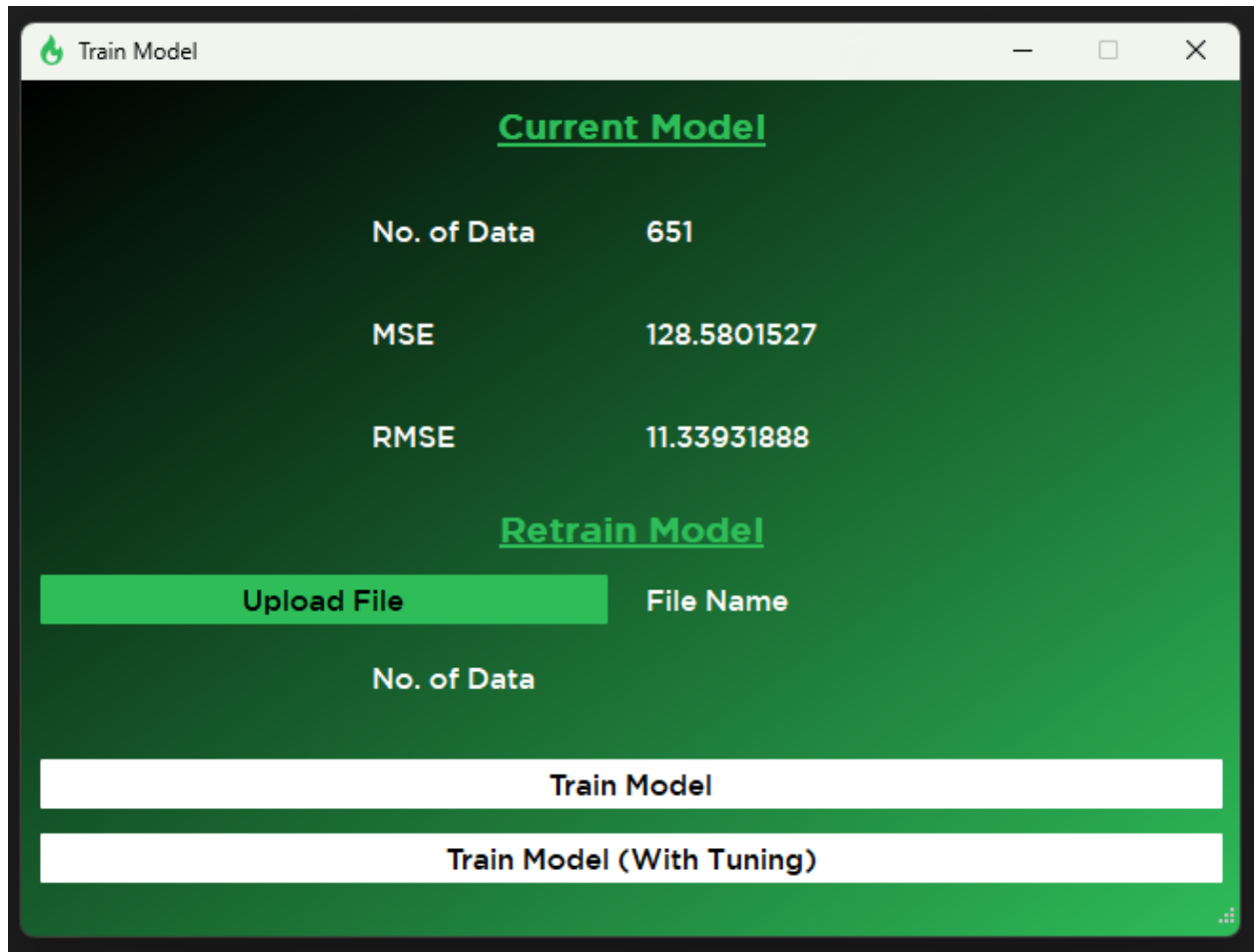


A warning message is displayed.

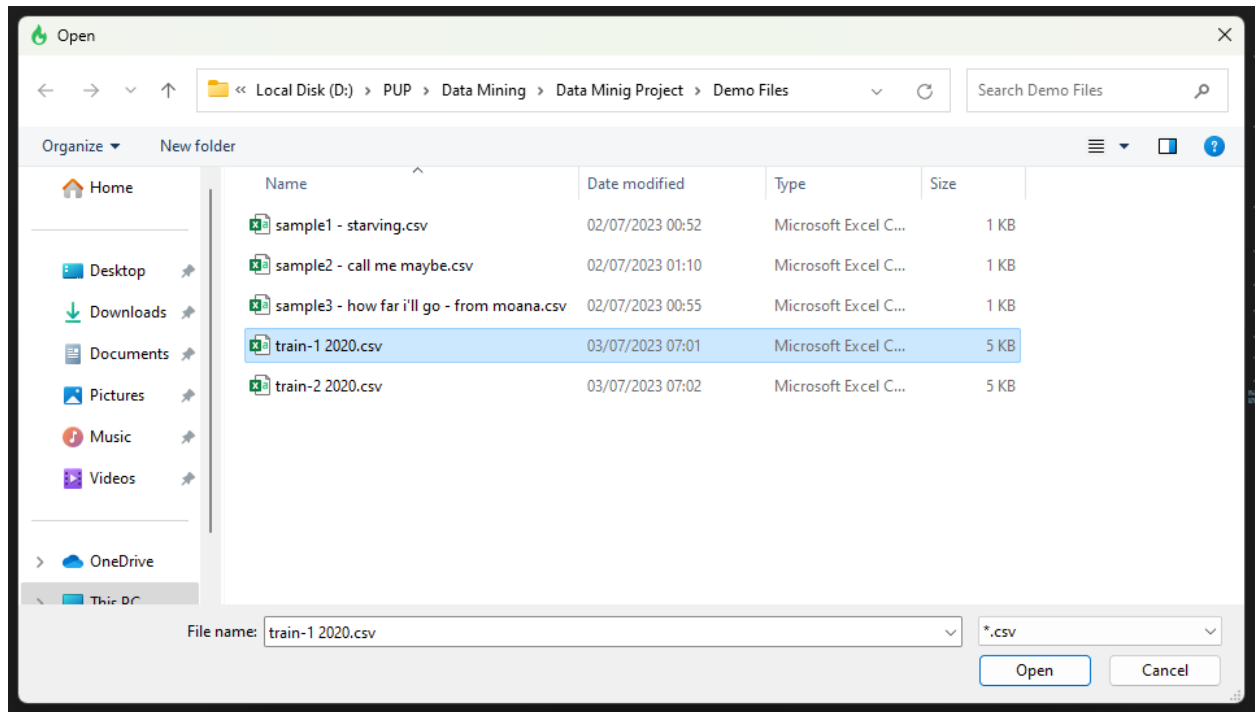
## B. Model Training



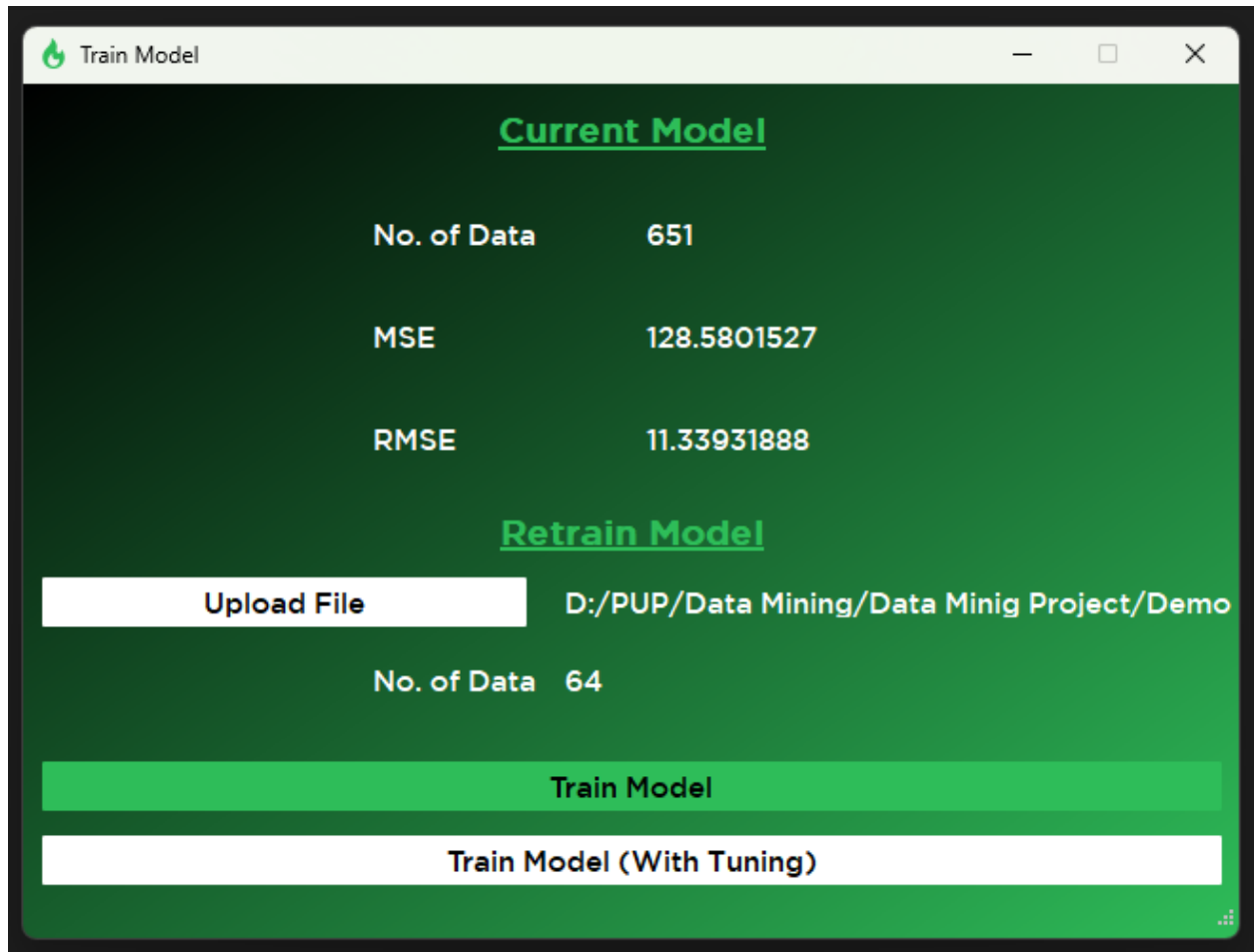
\*Click "Train Model" Button



\*Click “Upload File” Button

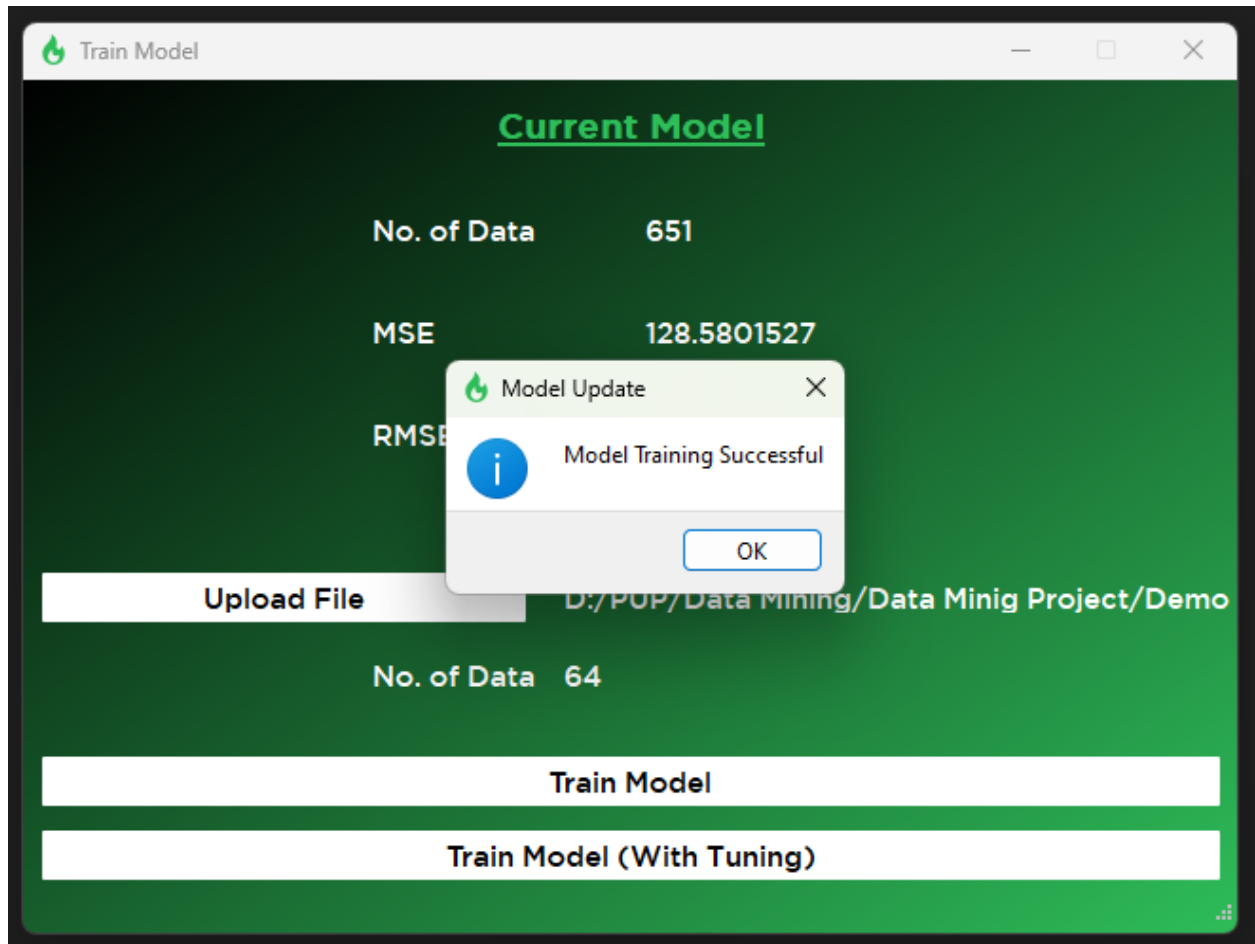


\*Select the desired .csv file and click “Open” Button



User can choose to train the model in two modes, a fixed model and a model with hyperparameter tuning (will take 3-5 mins).

\*Click "Train Model" Button



A success message is displayed.

Train Model

— □ ×

Current Model

No. of Data715MSE133.72027972027973RMSE11.56374851509145

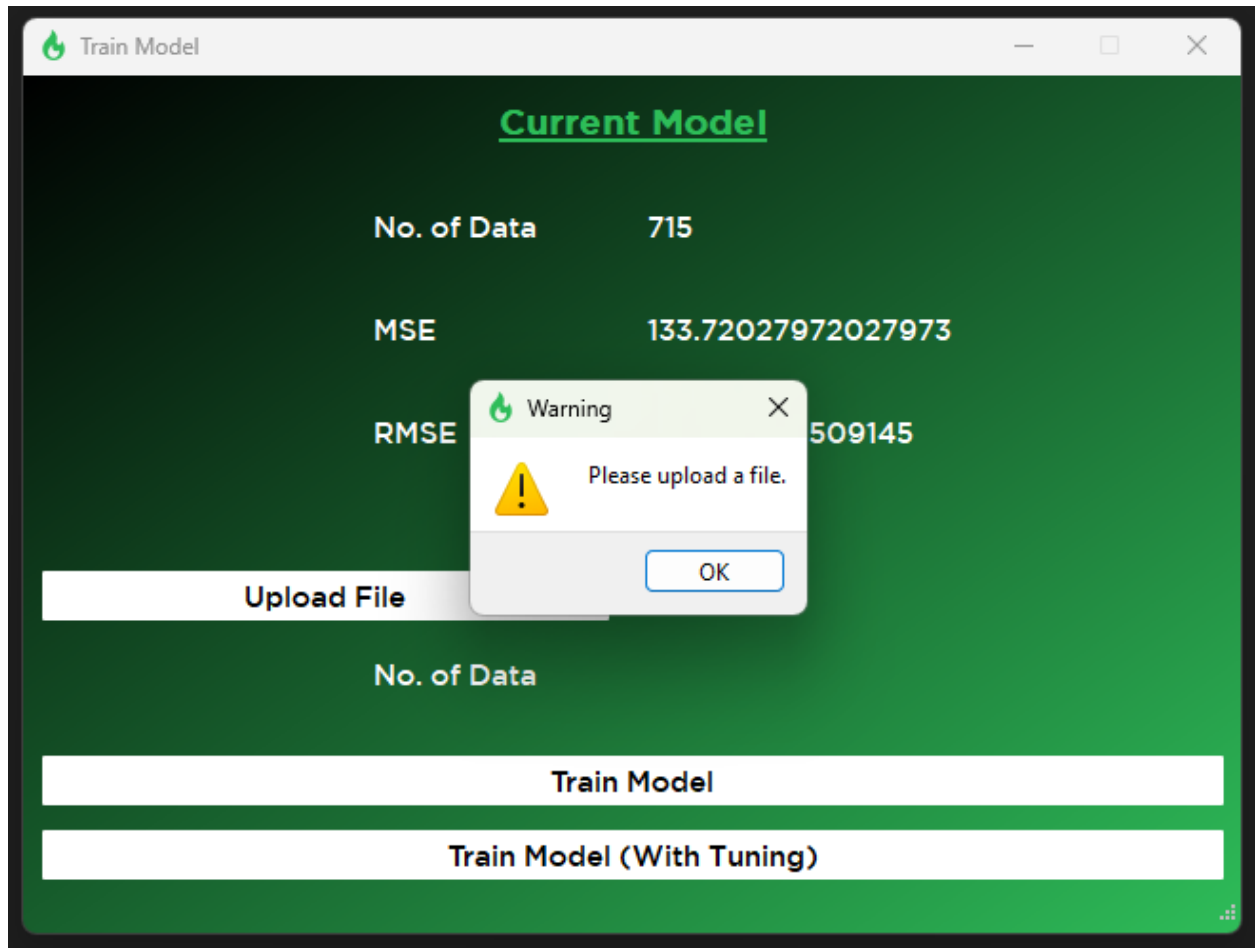
Retrain Model

Upload FileFile NameNo. of Data

Train Model

Train Model (With Tuning)

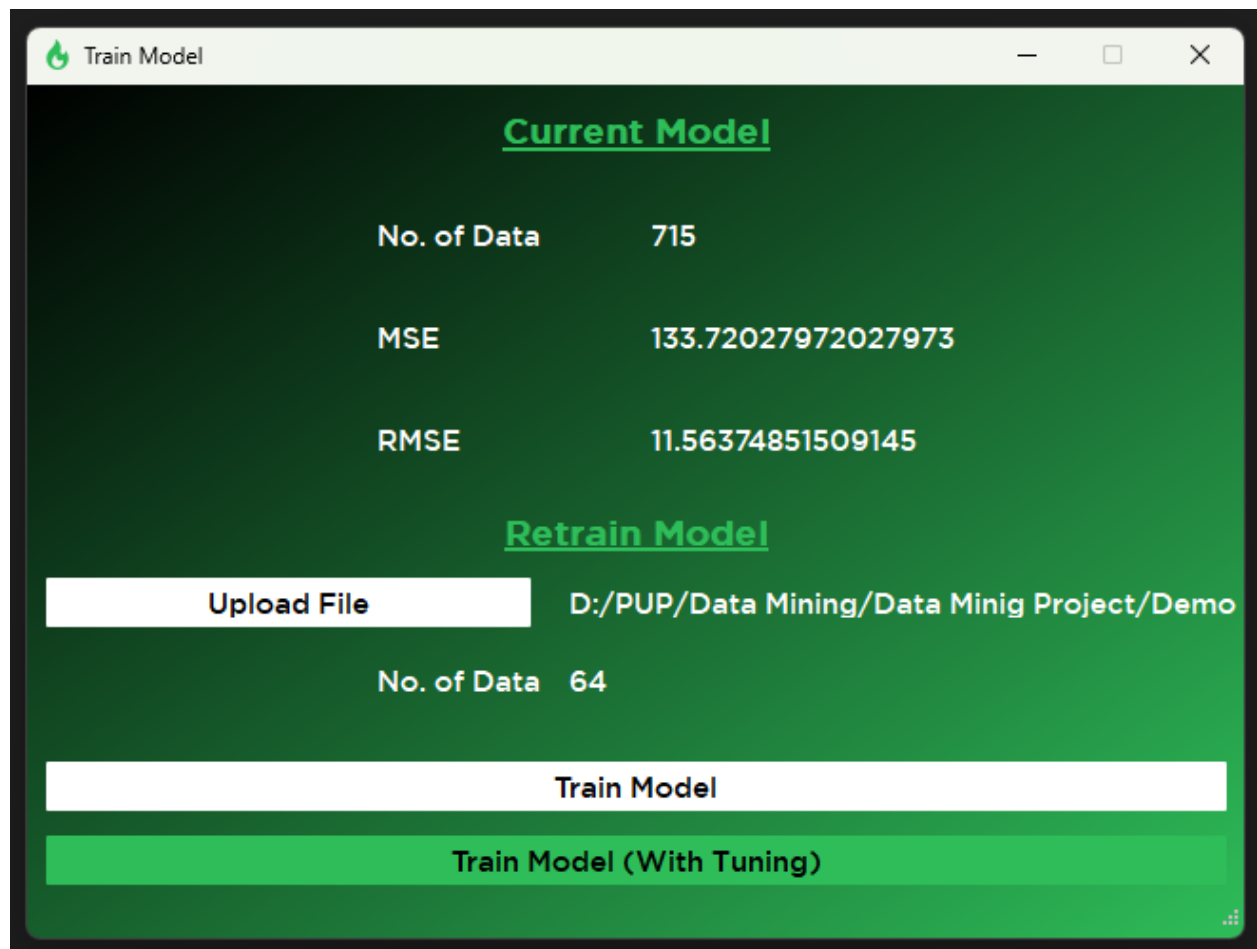
Model information is updated.



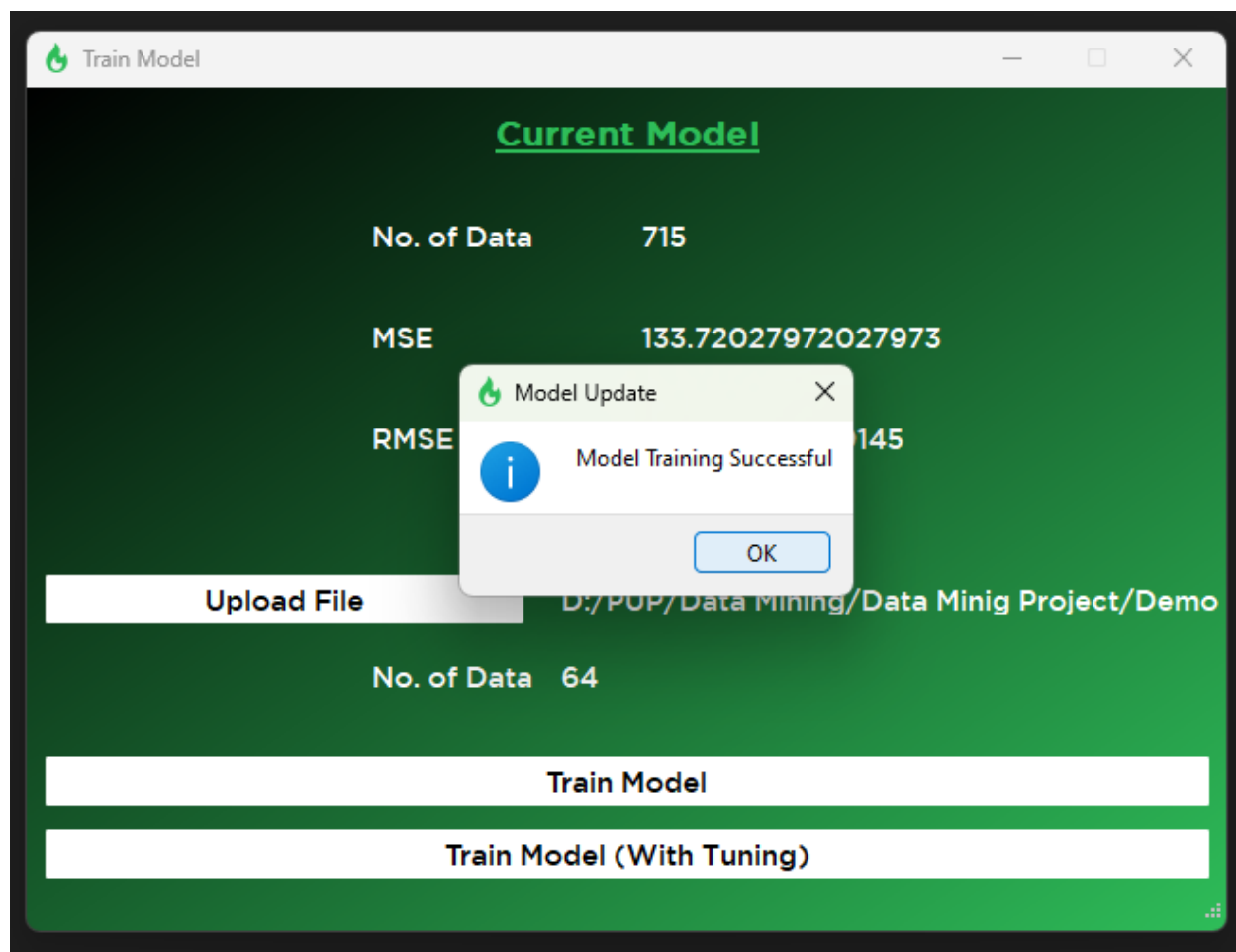
\*Upload Another File

Clicking the "Train Model" and "Train Model (With Tuning)" Buttons without uploading a file will prompt a warning message.





\*Click "Train Model (with Tuning)" Button



After 3 minutes, the model is updated.

[illegible]

Model information is updated.

## **VI. Conclusion**

Section VI covers the conclusion of the project, from dataset EDA up to evaluation of trained model — decision tree regressor. This section is divided into two parts, the findings and the conclusion.

### **Findings**

1. The top three genres present in the dataset are “dance pop,” “adult standards,” and “album rock,” with 17.5%, 14.3%, and 11.8% records, respectively. The remaining 112 top genres build up 56.4% of the dataset, where the majority do not exceed 2% or 13 occurrences.
2. The “duration” attribute had the most outliers, followed by “liveness” and “speechiness.”
3. The “energy” feature yielded the highest correlation value of 0.7 when compared to the “loudness” feature and has the highest correlation value compared to other features. On the other hand, the “acousticness” feature all have negative correlation values with respect to other features.
4. The target attribute “popularity” has the greatest correlation value of 0.36 with the “loudness” attribute. And has a negative correlation with “liveness” and “acousticness.”
5. The top three features that directly influence the “popularity” of the song are: “year,” “duration,” and “acousticness.”
6. In ascending order, the trained model’s predicted values are 50, 58, 62, 65, 68, 71, and 72.

7. The decision tree regressor model has an MSE of 128.58 and an RMSE of 11.34.  
The training and testing scores are 32.49% and 34.12%, respectively.

## **Conclusion**

The EDA performed on the dataset revealed information that explains the high MSE and RMSE values of the decision tree regressor. The application of one-hot encoding to the categorical “top genre” feature added 115 binary columns, and only “top genre\_dance pop,” “top genre\_adult standards,” and “top genre\_album rock” exceeded 13 occurrences. This means there are insufficient records for most features present in the dataset, compromising the model's performance in training. Additionally, linear correlation among features revealed that the target variable “popularity” has the most negligible correlation value with the rest, which explains the difficulty in predicting the actual values. And although “year,” “duration,” and “acousticness” features were discovered to be relevant to “popularity,” removing other features from the dataset still produced a close MSE and RMSE values.

To fulfill the other objective of the project, the dynamic nature of updating and retraining the model during runtime, hyperparameter tuning is necessary. The training and test scores also revealed that the model is underfitting, which explains the high MSE and RMSE values. Although the resulting 128.58 MSE and 11.34 RMSE values are high, the project still accomplished its objectives and was able to perform the necessary data mining procedures.

## References

- Brunda, S., Namish, L., Chiranthan, S., & Khan, A. (2020). Crop Price prediction using Random Forest and Decision Tree Regression. *International Research Journal of Engineering and Technology (IRJET)*, 07(09).  
<https://www.irjet.net/archives/V7/i9/IRJET-V7I938.pdf>
- Carbone, Nicolas. (2020). Spotify Past Decades Songs Attributes, Version 9. Retrieved May 20, 2023, from <https://www.kaggle.com/datasets/cnic92/spotify-past-decades-songs-50s10s>
- Essa, Y., Usman, A., Garg, T., & Singh, M. K. (2022). Predicting the Song Popularity Using Machine Learning Algorithm. *International Journal of Scientific Research & Engineering Trends*, 8(2), 1054–1062.  
[https://ijsret.com/wp-content/uploads/2022/05/IJSRET\\_V8\\_issue2\\_281.pdf](https://ijsret.com/wp-content/uploads/2022/05/IJSRET_V8_issue2_281.pdf)
- Karim, R., Alam, K., Hossain, R., & Hossaim, R. (2021). Stock Market Analysis Using Linear Regression and Decision Tree Regression. *2021 1st International Conference on Emerging Smart Technologies and Applications (eSmarTA)*.  
<https://doi.org/10.1109/esmarta52612.2021.9515762>
- Kyauk, E., Park, E., & Pham, J. (2015). Predicting Song Popularity.  
[http://cs229.stanford.edu/proj2015/140\\_report.pdf](http://cs229.stanford.edu/proj2015/140_report.pdf)
- Shepherd, J. (2023, May 15). *23 Essential Spotify Statistics You Need to Know in 2023*. The Social Shepherd. Retrieved May 20, 2023, from <https://shorturl.at/flDG4>