SONG POPULARITY PREDICTION

A Project Report
Submitted in Partial fulfilment
of the Degree of
Master of Computer Applications

Supervisor's Name – Dr. Lokesh Jain

Submitted By – Himanshu
Enrollment No - 03022201
Semester - 3rd Semester

Jagan Nath University
Bahadurgarh (NCR)

(2022-24)

PROJECT CERTIFICATE

This is to certify that the project report entitled **Song Popularity Prediction** submitted to **JaganNath University, Bahadurgarh** in partial fulfillment of the requirement for the award of the degree of **Master of Computer Applications (MCA),** is the original work carried out by **Himanshu Kaushik, MCA 3rd semester, Session 2022-24,** under the guidance of **Dr. Lokesh Jain.** The matter embodied in this Project is a genuine work done by the student and has not been submitted whether to this University or to any other University / Institute for the fulfillment of the requirement of any course of study.

Name of the student: Himanshu Kaushik

Name of the Guide Dr. Lokesh Jain

Signature of the Student

Signature of the Guide

Enrolment No.: 030222016

Date: 23rd December 2022

ACKNOWLEDGEMENT

I offer my sincere thanks and humble regards to JAGANNATH UNIVERSITY, BAHADURGARH for imparting us very valuable professional course MCA. I pay my gratitude and sincere regards to Dr. Lokesh Jain, my project guide for giving me the cream of his knowledge. I am thankful to him as he has been a constant source of advice, motivation, and inspiration. My sincere thanks go to Mohit Mathur Sir, our HOD of IT Department for this coordination in extending every support for the completion of the project. I am also thankful to him for giving his suggestions and encouragement throughout the project work. I take the opportunity to express my gratitude and thanks to our library staff for allowing me to utilize their resources to complete the project. I am also thankful to my family and friends for constantly motivating me to complete the project and providing me with an environment, which enhanced my knowledge.

Name: Himanshu Kaushik

Course: MCA 3rd Semester

Enroll. No: 030222016

TABLE OF CONTENT

1. INTRODUCTION

In the rapidly evolving landscape of the 21st century, the integration of artificial intelligence (AI) has become pivotal in reshaping various facets of our lives. From personalized recommendations to autonomous vehicles, AI has permeated diverse sectors, revolutionizing the way we interact with technology. One particularly intriguing application of AI is in predicting the popularity of songs, a task that not only aligns with the current technological zeitgeist but also holds immense potential in the music industry.

As we navigate through the era of AI, where data-driven insights and predictive analytics are paramount, the ability to forecast the popularity of songs has emerged as a compelling avenue for exploration. This project delves into the intersection of machine learning and music, aiming to develop a predictive model that anticipates the popularity of songs based on various features. By harnessing the power of algorithms and data, this endeavor seeks to offer a glimpse into the future of music trends, providing valuable insights for both artists and music enthusiasts alike.

The ubiquitous nature of music in our lives makes understanding and predicting its popularity a task of considerable importance. In an era where digital platforms and streaming services have become the primary means of music consumption, the ability to forecast which songs are likely to capture the public's attention has profound implications. Artists can benefit from tailored insights to inform their creative process, while record labels and streaming platforms can optimize their promotional strategies.

Moreover, from the perspective of consumers, a predictive model for song popularity serves as a curated guide in the vast ocean of available music. Personalized recommendations that align with individual preferences enhance the overall music discovery experience, creating a symbiotic relationship between creators and consumers. AI, particularly machine learning, plays a central role in the realization of predictive models for song popularity. By analyzing vast datasets encompassing various musical attributes, user behaviors, and historical trends, machine learning algorithms can discern patterns and relationships that elude traditional methods. This project harnesses the computational power of AI to not only predict the popularity of songs but also to unravel the intricate dynamics that influence musical trends.

2. OBJECTIVES

The primary goal of this project is to develop an interactive and user-friendly system that assists individuals in discovering top-rated songs tailored to their specific preferences and constraints. By combining data analytics, machine learning algorithms, the system aims to enhance the music experience for users by offering relevant and popularity score suggestions.

The objectives of the Song Popularity Prediction model project are multifaceted, encompassing both technical and user-centric goals:

2.1 Dataset Exploration and Preparation

Conduct a comprehensive exploration of the dataset, encompassing a diverse collection of songs attributed to various artists. This involves understanding the distribution of features, identifying outliers, and addressing missing data. The goal is to curate a high-quality dataset that captures the nuances of different music genres, artist styles, and temporal trends.

2.2 Model Development and Evaluation

Implement and fine-tune machine learning models to predict song popularity scores based on a set of relevant features. This includes selecting appropriate algorithms, optimizing hyperparameters, and evaluating model performance using metrics such as Mean Absolute Error (MAE) or R-squared. The objective is to create a robust predictive model that effectively captures the complexities of factors influencing song popularity.

2.3 Interpretability and User Interaction:

Enhance the user experience by implementing a user-friendly interface for interacting with the predictive model. Prioritize features that allow users to input song details and receive interpretable predictions. Additionally, focus on making the model's decision-making process transparent, enabling users, including artists and industry professionals, to understand the factors influencing predicted popularity scores.

2.4 Continuous Improvement and Future Exploration:

Lay the groundwork for ongoing enhancements and exploration. This involves documenting the model architecture, key decisions, and data sources for future reference. Consider avenues for potential collaborations with artists and industry experts to refine the model based on real-world insights. Explore possibilities for extending the project's capabilities, such as incorporating new data sources or adapting the model to evolving music trends.

3. TOOLS & ENVIRONMENT

The software is designed to be light-weighted so that it doesn't be a burden on the machine running it. This system is being build keeping in mind the general availability of hardware and software compatibility. Here are the minimum hardware and software requirement for face recognition system for attendance.

3.1 HARDWARE REQUIREMENT:

Processor	: Intel Pentium 4 or above
Hard Disk Utilization	: 1 GB or above
Input Devices	: Keyboard or Mouse
RAM	: 2 GB or above

3.2 SOFTWARE REQUIREMENT:

Operating System	: Window 7,8, 8.1, 10, 11, and above

3.3 TECHNOLOGY USED:

Code Compiler	: Jupyter Notebook

4. ANALYSIS DOCUMENT

4.1 Problem statement:

In the rapidly evolving landscape of the music industry, artists and stakeholders are confronted with the daunting task of navigating an overwhelming volume of music content. The surge in digital platforms and streaming services has democratized music creation and consumption, leading to an unprecedented diversity of musical offerings. However, this abundance also poses a significant challenge – the difficulty in predicting and understanding the factors that contribute to the success or popularity of a song.

Artists invest substantial time, effort, and resources in creating music, yet the inherent uncertainty surrounding audience reception remains a critical hurdle. Without a reliable means of anticipating which songs are likely to resonate with listeners, artists face challenges in optimizing their creative processes, marketing strategies, and overall career trajectories.

Moreover, the burgeoning digital age has redefined how consumers discover and engage with music. Traditional methods of promotion and distribution are being eclipsed by algorithm-driven recommendations and personalized playlists. This shift demands a nuanced understanding of the intricate interplay of factors influencing song popularity in a data-driven and algorithmically guided environment.

4.2 Proposed solution:

To address the challenges outlined in the problem statement and empower stakeholders in the music industry, we propose the development and implementation of a robust Song Popularity Prediction model. This solution involves a multi-faceted approach that integrates advanced machine learning techniques, comprehensive data analysis, and user-friendly interfaces. The primary components of the proposed solution are as follows:

4.3 Data-driven Predictive Modeling

Objective: Develop a machine learning model capable of predicting song popularity based on a diverse set of features, including but not limited to musical attributes, genre, and temporal trends.

Methodology: Employ regression or ensemble learning algorithms to discern patterns and relationships within a curated dataset of songs. This involves training the model on historical data to understand the factors influencing song popularity.

4.4 Feature Engineering and Selection

Objective: Identify and extract relevant features that significantly contribute to predicting song popularity.

Methodology: Conduct thorough feature engineering, considering the impact of musical attributes, artist characteristics, and broader contextual factors. Utilize techniques such as correlation analysis and feature importance ranking to select the most influential features.

Flowchart

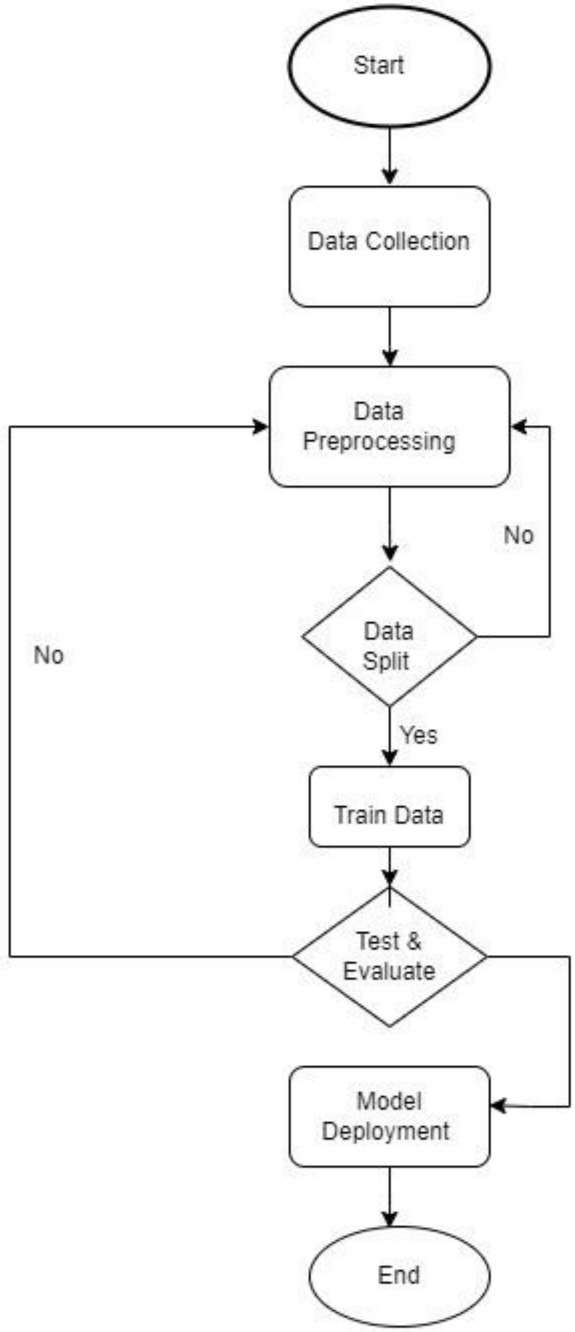

Figure 4.1 Flowchart Diagram

Use case Diagram

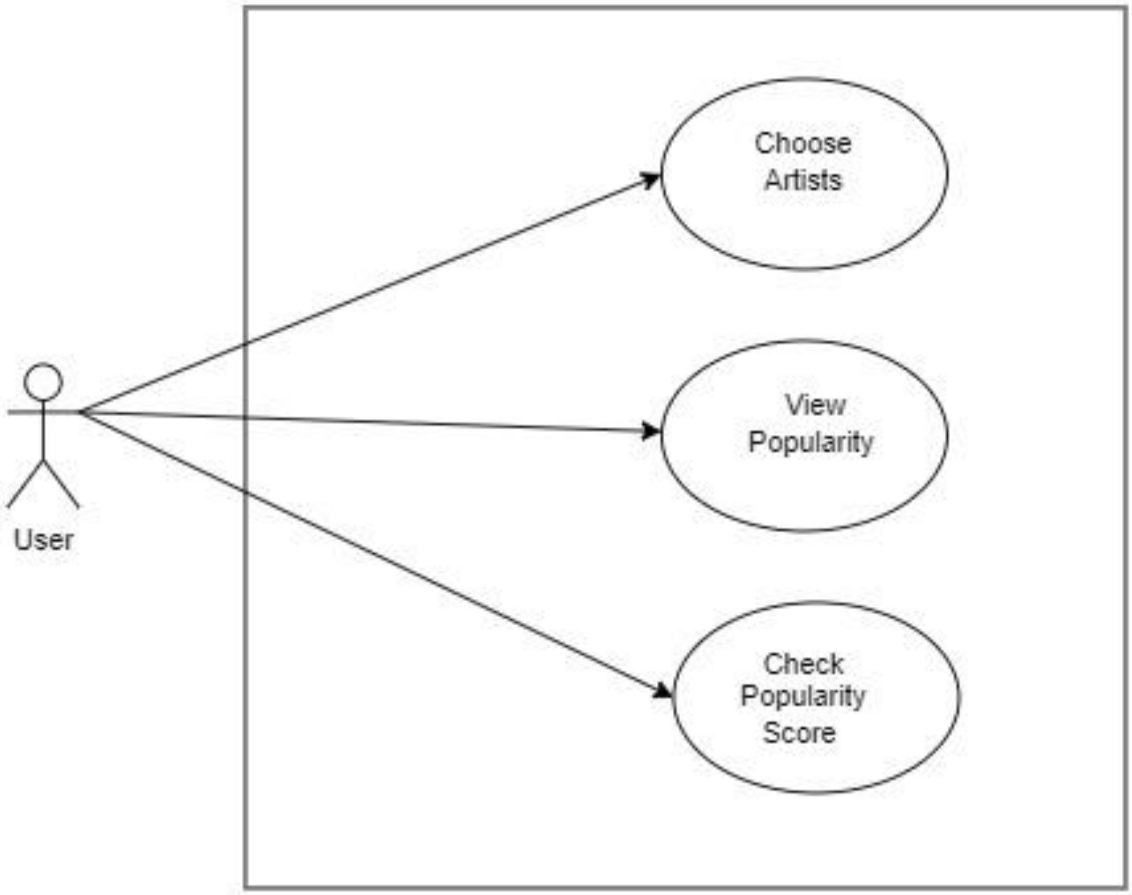

Figure 4.2 Use case Diagram

DFD – 0 Level

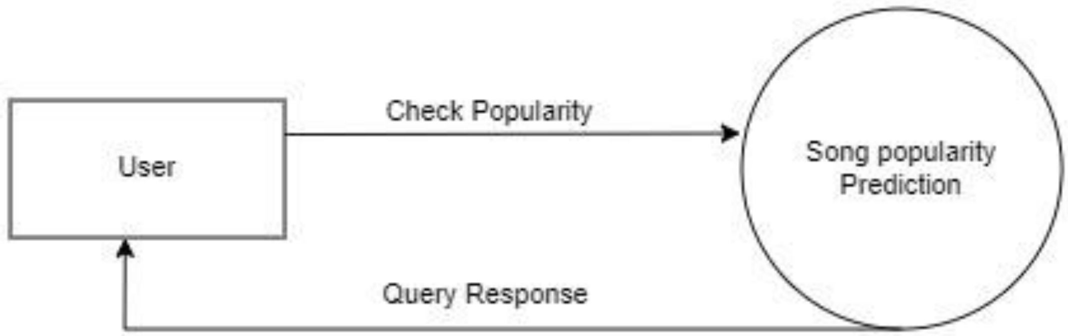

Figure 4.3 DFD – 0 Level

Data Dictionary:

ATTRIBUTE	DATATYPE	DESCRIPTION
key	int	Unique key of each song
pop_rating	object	Column created by us on the basis of popularity score
acousticness	float64	Value of song’s acousticnes
danceability	float64	The score of danceability of song
duration_ms	Int64	Total time of the song in miliseconds
energy	float64	Energetic score of the song
Instrumentalness	float64	The score of instrumentalness used in the song
Loudness	float64	The loudness of song in integer value
popularity	Int64	The popularity score of song in dataset
Speechiness	float64	The speech related context in the song available
valence	float64	The positivity score of the song in dataset

5. DESIGN DOCUMENT

5.1 Data analysis: - Initial analysis process is performed on data so as to discover patterns, spot irregularities, test hypothesis and check assumptions with the help of summary statistics and graphical representations. Multiple regression multivariate analysis on data fields were performed to see if there is a significant statistical relationship between multiplevariables

Based on the analysis, there are fields which has nothing to do with decision making and also errors in dataset entries that need to be removed for better performance of the model.

5.2 Sampling techniques: - sampling data for training and test purpose, this research divide the whole data set into test and training as well as validation data. Of the total n dataset n(25/100) is allocated for test data, and the remaining n(75/100) are allocated for training data while n is the total number of datasets. Sampling was done on datasets processed after multivariate data analysis. For decision making purpose, the datasets are even divided into validation and test data to see the performance of the system.

Application of machine learning algorithms: - Machine learning algorithms are applied to a system to figure out how the system behaves and how accurate the decision-making process made.

The dataset is composed of different tables, which consists of many information about users, restaurants and foods. For the purpose of providing the best recommendation. There are ratings of foods and restaurants in this dataset and are used accordingly to recommend to the users. Data preprocessing and feature extraction was done on this dataset to filter out only relevant attributes for recommendations so that unnecessary field is removed from the datasets.

Selecting the appropriate programming language for the implementation is also done. Python was chosen as a platform to develop the system because of wide and rich libraries crucial for this project, such as sklearn, pandas, matplot, keras, tensorflow and others. Anaconda Navigator includes all these libraries into one package. Specific environments were created for the implementation and installed all the necessary packages listed above. Jupyter Notebook is used to write scripts and simulation of the results in browsers.

To evaluate how well a classifier is performing, we have to test the model on invisible data. For the purpose of this research, before building a model, the data is split into two parts: a training set and a test set. The training set is used to train and evaluate the model during the development stage. You then use the trained model to make predictions on the unseen test set. This approach gives you a sense of the model's performance and robustness. Luckily, sklearn have a function called train_test_split () which divides data into sets. The function randomly splits the data using the test size parameter. For the purpose of this research, the test size represents 25% of the original dataset. The remaining make up the training data.

The value of performance evaluation for Accuracy, Loss and model evaluation was mentioned between 0 and 1. For Accuracy measurement, the more the value is closer to 1, the more acceptable and accurate the system is. For the loss measurement, the more the value approaches 0, the more accurate the system.

5.3 External APIs Integration

API Selection: Identify and integrate external APIs for geocoding and mapping. Ensure they align with the system's requirements. Distance Calculation: Design the system to calculate distances between locations using the geocoding and mapping APIs.

5. PROGRAM CODE

```
#importing data and other libraries
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set_style('darkgrid')

#importing the dataset
spotify = pd.read_csv('SpotifyFeatures0419.csv')

# Create the sorted dataframe, drop zeros, convert any categoricals
sort_data = spotify.sort_values('popularity', ascending=False).reset_index()
spotify_ordered = sort_data.drop(['index', 'track_id'], axis=1)
spotify_ordered.index = spotify_ordered.index + 1

spotify_ordered = spotify_ordered[spotify_ordered.popularity > 0]

spotify_ordered[['mode', 'key', 'time_signature']] = \
    spotify_ordered[['mode', 'key', 'time_signature']].astype('category')

spotify_ordered.shape
spotify_ordered.head()

classified = spotify_ordered.copy()
classified['pop_rating'] = ''

for i, row in classified.iterrows():
    score = 'unpopular'
    if (row.popularity > 50) & (row.popularity < 75):
```

```
        score = 'medium'
    elif row.popularity >= 75:
        score = 'popular'
    classified.at[i, 'pop_rating'] = score

# Inspect the new column
classified[['track_name', 'popularity', 'pop_rating']].head(3)

fig, ax = plt.subplots(1,1, figsize=(8,5))
_ = spotify_ordered['popularity'].plot(kind='hist', bins=50)
_ = plt.xlabel('Popularity')
_ = plt.title('Popularity Distribution', fontsize=14)

spotify_ordered[['popularity']].describe()
#looking how other features correlate with the popularity column
spotify_ordered.corr()

fig, ax = plt.subplots(1,1, figsize=(10,6))
_ = sns.heatmap(spotify.corr(), square=True, cmap='YlOrRd')
_ = plt.title('Correlation heat map', fontsize=14)
_ = plt.xticks(fontsize=12)
_ = plt.yticks(fontsize=12)

def scat_plot(x, y, hue=None, xlab='', ylab='', titl=''):
    '''Plots a scatterplot using given inputs'''
    fig, ax = plt.subplots(figsize=(10,6))
    _ = sns.scatterplot(x, y, hue=hue, s=12)
    _ = plt.xlabel(xlab, fontsize=12)
    _ = plt.ylabel(ylab, fontsize=12)
    _ = plt.title(titl, fontsize=14)
    _ = plt.legend(fontsize=12)
    plt.show()
```

```
def regress_plot(x='', y='', data=None, xlab='', ylab='', titl=''):
    '''Plots a scatterplot with a regression line
    using given inputs'''
    fig, ax = plt.subplots(figsize=(10,6))
    _ = sns.regplot(x, y, data=data, scatter_kws={"s": 10}, line_kws={'color':'r'})
    _ = plt.xlabel(xlab, fontsize=12)
    _ = plt.ylabel(ylab, fontsize=12)
    _ = plt.title(titl, fontsize=14)
    _ = plt.ylim(-3, 103)
    plt.show()

s = spotify_ordered

scat_plot(s.popularity, s.loudness, hue=classified.pop_rating, xlab='Popularity',\
        ylab='Loudness', titl='Loudness with popularity on X-axis')

regress_plot('loudness', 'popularity', data=s, xlab='Loudness',\
        ylab='Popularity', titl='Popularity vs. loudness')

scat_plot(s.popularity, s.instrumentalness, hue=classified.pop_rating, xlab='Popularity',\
        ylab='Instrumentalness', titl='Instrumentalness with popularity on X-axis')

regress_plot('instrumentalness', 'popularity', data=s, xlab='Instrumentalness',\
        ylab='Popularity', titl='Popularity vs. instrumentalness')



fig = plt.figure(figsize=(12,6))

ax1 = plt.subplot(1,2,1)
_ = sns.regplot(s.energy, s.popularity, scatter_kws={"s": 5}, line_kws={'color':'r'})
_ = plt.title('Pop vs. energy')

ax2 = plt.subplot(1,2,2)
```

```
_ = sns.regplot(s.danceability, s.popularity, scatter_kws={"s": 5}, line_kws={'color':'r'})
_ = plt.title('Pop vs. danceability')

artists = ['Shakira', 'Jonas Brothers', 'Tame Impala', 'The Weeknd']

# Create a list of indices corresponding to the artists above
# The first comprehension creates a list of lists, the second flattens it into one
to_drop = [classified[classified.artist_name == name].index.tolist() for name in artists]
to_drop = [ind for sub in to_drop for ind in sub]

# Gather the test cases
df_x = classified.copy()
cases = df_x[df_x.index.isin(to_drop)]

# Remove the test cases from data
classified.drop(to_drop, inplace=True)
spotify_ordered.drop(to_drop, inplace=True)

print(classified.shape)
print(spotify_ordered.shape)

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

# Set random state
state=25

# here we Shuffle the data
reg_data = spotify_ordered.sample(frac=1, random_state=state).reset_index(drop=True)

# First, try without categoricals
X = reg_data.select_dtypes(include='number').drop('popularity', axis=1)
y = reg_data.popularity
```

```
# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=state)

def custom_loss(prediction, actual):
    paired = zip(prediction, actual)
    listed = list(paired)
    diffs = {'Under five': 0, 'Five to ten': 0, 'Over ten': 0, 'Average error': 0}
    sum = 0

    for pair in listed:
        sum += abs(pair[0] - pair[1])
        if abs(pair[0] - pair[1]) < 5:
            diffs['Under five'] += 1
        elif 5 <= abs(pair[0] - pair[1]) < 10:
            diffs['Five to ten'] += 1
        else:
            diffs['Over ten'] += 1

    diffs['Average error'] = sum / len(listed)
    return diffs

%%time

linreg = LinearRegression()
linreg.fit(X_train, y_train)

lin_pred = linreg.predict(X_test)

print(linreg.score(X_test, y_test))custom_loss(lin_pred, y_test)

no_artist = reg_data.drop(['artist_name', 'track_name'], axis=1)
df_encoded = pd.get_dummies(no_artist)
```

```
df_encoded.columns

XX = df_encoded.drop('popularity', axis=1)
yy = df_encoded.popularity

X_train, X_test, y_train, y_test = train_test_split(XX, yy, test_size=0.25,
random_state=state)

%%time
lr = LinearRegression()
lr.fit(X_train, y_train)

lr_pred = lr.predict(X_test)

print(lr.score(X_test, y_test))custom_loss(lr_pred, y_test)

from sklearn.model_selection import cross_val_score

# Use the regressor from above to set up the cross_val
cvals = cross_val_score(lr, XX, yy, cv=6)

# check the results
print(cvals)
print('The mean cross-validatoin score is: {num:.{dig}f}'.format\
    (num=np.mean(cvals), dig=4))

%%time
from sklearn.linear_model import Ridge

ridge = Ridge(alpha=0.5, normalize=True)
ridge.fit(X_train, y_train)

print(ridge.score(X_test, y_test))
```

```
custom_loss(ridge.predict(X_test), y_test)

from sklearn.model_selection import GridSearchCV

alphas = {'alpha': [0.0005, 0.0006, 0.00075, 0.0009, 0.001]}

rg = Ridge(normalize=True)

rg_cv = GridSearchCV(rg, alphas, cv=6)

rg_cv.fit(XX, yy)

print(rg_cv.best_params_)
print(rg_cv.best_score_)

fig = plt.subplots(figsize=(9,6))
_ = plt.plot(list(rg_cv.predict(X_test))[:500], label='Predicted')
_ = plt.plot(list(y_test)[:500], c='r', alpha=0.3, label='Actual')
_ = plt.legend(loc='upper right')
_ = plt.ylabel('Popularity', fontsize=12)
_ = plt.title('Actual popularity vs. predicted', fontsize=14)

fig, ax = plt.subplots(1,1, figsize=(8,5))
_ = sns.countplot(x='pop_rating', data=classified)
_ = plt.xlabel('Ratings', fontsize=14)
_ = plt.title('Counts', fontsize=14)

df = classified.drop(['artist_name', 'track_name'], axis=1)
df = pd.get_dummies(df, columns=['key', 'mode', 'time_signature'])
df.shape

from sklearn.model_selection import train_test_split
```

```
df_pop = df[df.pop_rating == 'popular']

df_med = df[df.pop_rating == 'medium']

df_unpop = df[df.pop_rating == 'unpopular']

# Set random seed
state=25

X_tr_p, X_ts_p, y_tr_p, y_ts_p = train_test_split(df_pop.drop(['popularity', 'pop_rating'],
axis=1),\
                                    df_pop.pop_rating, test_size=0.15, random_state=state)

X_tr_m, X_ts_m, y_tr_m, y_ts_m = train_test_split(df_med.drop(['popularity', 'pop_rating'],
axis=1),\
                                    df_med.pop_rating, test_size=0.15, random_state=state)

X_tr_up, X_ts_up, y_tr_up, y_ts_up = train_test_split(df_unpop.drop(['popularity',
'pop_rating'], axis=1),\
                                    df_unpop.pop_rating, test_size=0.15, random_state=state)

pop_train = pd.concat([X_tr_p, y_tr_p], axis=1)
med_train = pd.concat([X_tr_m, y_tr_m], axis=1)
unpop_train = pd.concat([X_tr_up, y_tr_up], axis=1)

training = pd.concat([pop_train, med_train, unpop_train], axis=0)

training = training.sample(frac=1, random_state=state).reset_index(drop=True)

# Popularity has been removed, so only 30 columns
training.shape
```

```
pop_test = pd.concat([X_ts_p, y_ts_p], axis=1)
med_test = pd.concat([X_ts_m, y_ts_m], axis=1)
unpop_test = pd.concat([X_ts_up, y_ts_up], axis=1)

final_test = pd.concat([pop_test, med_test, unpop_test], axis=0)

final_test = final_test.sample(frac=1, random_state=state).reset_index(drop=True)

final_test.shape

X_class = training.drop('pop_rating', axis=1)
y_class = training.pop_rating

X_train, X_test, y_train, y_test = train_test_split(X_class, y_class, test_size=0.25,
random_state=state)

%%time

from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Basic decision tree
dt = DecisionTreeClassifier(max_depth=20, random_state=state)
dt.fit(X_train, y_train)

pred = dt.predict(X_test)

print(accuracy_score(pred, y_test))

params = {'max_depth': [2, 10, 20, 40, 50],
        'min_samples_leaf': np.arange(1,10,2),}

dt = DecisionTreeClassifier(random_state=state)
```

```
dt_cv = GridSearchCV(dt, params, cv=6)

dt_cv.fit(X_class, y_class)

print(dt_cv.best_params_)
print('The average runtime is: ', np.mean(dt_cv.cv_results_['mean_fit_time']))
print('The best score is: ', dt_cv.best_score_)

p = dt_cv.best_estimator_.predict(X_test)

print(accuracy_score(p, y_test))

%%time

from sklearn.ensemble import BaggingClassifier

tree = DecisionTreeClassifier(max_depth=2, random_state=state)
bc = BaggingClassifier(base_estimator=tree, n_estimators=100, random_state=state)

bag_cv = cross_val_score(bc, X_class, y_class, cv=6)

print(bag_cv)
print('The mean cross-validatoin score is: {num:.{dig}f}'.format\
    (num=np.mean(bag_cv), dig=4))

%%time

from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=100, random_state=state)
rf.fit(X_train, y_train)
```

```
rf_pred = rf.predict(X_test)

print(accuracy_score(rf_pred, y_test))

rf_params = {'n_estimators': [100, 300, 350],
            'max_depth': [2, 5, 100],
            'min_samples_leaf': [1, 2]}

rf_new = RandomForestClassifier(random_state=state)

rf_cv = GridSearchCV(rf_new, rf_params, n_jobs=-1, cv=6)

rf_cv.fit(X_train, y_train)

print(rf_cv.best_params_)

%%time

rf = RandomForestClassifier(max_depth=100, n_estimators=300, min_samples_leaf=1,
random_state=state)

rf.fit(X_train, y_train)

rf_pred = rf.predict(X_test)

print(accuracy_score(rf_pred, y_test))

important = pd.Series(data=rf.feature_importances_, index=X_train.columns).sort_values()

fig = plt.subplots(figsize=(10,8))
_ = important.plot(kind='barh')
_ = plt.title('RF feature importances', fontsize=14)
```

```
%%time

from sklearn.ensemble import AdaBoostClassifier

dt_b = DecisionTreeClassifier(max_depth=1, random_state=state)

adb = AdaBoostClassifier(base_estimator=dt_b, n_estimators=200)

adb.fit(X_train, y_train)

adb_pred = adb.predict(X_test)

print(accuracy_score(adb_pred, y_test))

%%time

from sklearn.ensemble import GradientBoostingClassifier

gb = GradientBoostingClassifier(max_depth=3, n_estimators=200, random_state=state)

gb.fit(X_train, y_train)

gb_pred = gb.predict(X_test)

print(accuracy_score(gb_pred, y_test))

from sklearn.metrics import confusion_matrix

bc.fit(X_train, y_train)

confusion_matrix(y_test, bc.predict(X_test))

confusion_matrix(y_test, rf_pred)
```

```
confusion_matrix(y_test, adb_pred)

dt_bal = DecisionTreeClassifier(max_depth=2, class_weight='balanced',
random_state=state)
bc_bal = BaggingClassifier(base_estimator=dt_bal, n_estimators=200)

bc_bal.fit(X_train, y_train)

bcb_p = bc_bal.predict(X_test)

print(accuracy_score(bcb_p, y_test))

confusion_matrix(y_test, bcb_p)

from sklearn.metrics import classification_report

print(classification_report(y_test, bcb_p))

rf_b = RandomForestClassifier(max_depth=100, n_estimators=300, min_samples_leaf=1,\
                    class_weight={'medium': 2.475, 'popular': 100, 'unpopular': 1},
random_state=state)

rf_b.fit(X_train, y_train)

rfb_p = rf_b.predict(X_test)

print(accuracy_score(rf_pred, y_test))

print(classification_report(y_test, rfb_p))

dt_bal = DecisionTreeClassifier(max_depth=2,\
                        class_weight={'medium': 2.475, 'popular': 150, 'unpopular': 1},\
```

```
                          random_state=state)

ad_bal = AdaBoostClassifier(base_estimator=dt_bal, n_estimators=200)

ad_bal.fit(X_train, y_train)

ad_bal_p = ad_bal.predict(X_test)

print(accuracy_score(ad_bal_p, y_test))

print(classification_report(y_test, ad_bal_p))

X_final = final_test.drop('pop_rating', axis=1)
y_final = final_test.pop_rating

holdout_p = bc_bal.predict(X_final)

print(accuracy_score(holdout_p, y_final))

confusion_matrix(y_final, holdout_p)

print(classification_report(y_final, holdout_p))

rf_ho_p = rf_b.predict(X_final)

print(accuracy_score(rf_ho_p, y_final))

print(classification_report(y_final, rf_ho_p))

adb_ho_p = ad_bal.predict(X_final)

print(accuracy_scoare(adb_ho_p, y_final))
```

```
confusion_matrix(y_final, adb_ho_p)

print(classification_report(y_final, adb_ho_p, np.unique(y_final)))

cases_mix = cases.sample(frac=1.0, random_state=state).reset_index(drop=True)

cases_drop = cases_mix.drop(['artist_name', 'track_name', 'popularity'], axis=1)
cases_enc = pd.get_dummies(cases_drop, columns=['key', 'mode', 'time_signature'])

cases_X = cases_enc.drop(['pop_rating'], axis=1)
cases_y = cases_mix[['pop_rating']]

cases_pred = pd.DataFrame(bc_bal.predict(cases_X), columns=['predicted_pop'])

results = pd.concat([cases_mix, cases_pred], axis=1)
results.iloc[:,[0,1,15,16,17]]

import tkinter as tk
from tkinter import ttk, Text

class SongPopularityPredictionGUI:
    def __init__(self, master):
        self.master = master
        master.title("Song Popularity Prediction")

        self.artist_label = ttk.Label(master, text="Artist:")
        self.artist_label.grid(row=0, column=0)

        # Use a Combobox for the artist dropdown
        artist_values = df['artists'].unique().tolist()
        self.artist_combobox = ttk.Combobox(master, values=artist_values)
        self.artist_combobox.grid(row=0, column=1)
```

```
        self.song_label = ttk.Label(master, text="Song:")
        self.song_label.grid(row=1, column=0)

        # Use a Combobox for the song dropdown
        song_values = df['name'].unique().tolist()
        self.song_combobox = ttk.Combobox(master, values=song_values)
        self.song_combobox.grid(row=1, column=1)

        self.predict_button = ttk.Button(master, text="Predict Popularity",
command=self.predict_popularity)
        self.predict_button.grid(row=2, column=0, columnspan=2)

        # Add a Text widget to display the results
        self.result_text = Text(master, height=5, width=50, state=tk.DISABLED)
        self.result_text.grid(row=3, column=0, columnspan=2)

    def predict_popularity(self):
        artist = self.artist_combobox.get()
        song = self.song_combobox.get()

        if not artist or not song:
            print("Please select both artist and song.")
            return

        result = songPopularityPrediction(artist, song)

        if result is not None:
            pred, actual = result
            result_text = f"Predicted popularity: {pred[0]:.2f}\nActual popularity: {actual[0]}"

            # Clear previous results
            self.result_text.config(state=tk.NORMAL)
            self.result_text.delete("1.0", tk.END)
```

```
            # Display new results
            self.result_text.insert(tk.END, result_text)
            self.result_text.config(state=tk.DISABLED)

    # ... other methods ...

# Assuming 'df', 'features', and 'rfr' are defined before calling this function

root = tk.Tk()
app = SongPopularityPredictionGUI(root)
root.mainloop()



import tkinter as tk
from tkinter import ttk
import pandas as pd

# Function to update the treeview based on the selected name
def update_treeview(*args):
    selected_name = name_var.get()
    if not selected_name:
        return
    selected_row = df[df['name'] == selected_name]
    treeview.delete(*treeview.get_children())  # Clear previous entries

    # Insert data into treeview
    for attribute in df.columns:
        value = selected_row[attribute].values[0]
        treeview.insert("", "end", values=(attribute, value))

# Create the main window
root = tk.Tk()
```

```
root.title("DataFrame GUI")

# Dropdown for selecting a name
name_var = tk.StringVar()
name_label = ttk.Label(root, text="Select Name:")
name_dropdown = ttk.Combobox(root, textvariable=name_var, values=df['name'].tolist())
name_dropdown.bind('<<ComboboxSelected>>', update_treeview)

# Treeview for displaying attributes
treeview = ttk.Treeview(root, columns=("Attribute", "Value"), show="headings", height=20)
treeview.heading("Attribute", text="Attribute")
treeview.heading("Value", text="Value")

# Layout
name_label.grid(row=0, column=0, padx=10, pady=10, sticky='w')
name_dropdown.grid(row=0, column=1, padx=10, pady=10, sticky='w')
treeview.grid(row=1, column=0, columnspan=2, padx=10, pady=5, sticky='w')

# Run the Tkinter main loop
root.mainloop()

import tkinter as tk
from tkinter import ttk

class SongPopularityPredictionGUI:
    def __init__(self, master):
        self.master = master
        master.title("Song Popularity Prediction")

        self.artist_label = ttk.Label(master, text="Artist:")
        self.artist_label.grid(row=0, column=0)

        # Use a Combobox for the artist dropdown
```

```
        artist_values = df['artists'].unique().tolist()
        self.artist_combobox = ttk.Combobox(master, values=artist_values)
        self.artist_combobox.grid(row=0, column=1)

        self.song_label = ttk.Label(master, text="Song:")
        self.song_label.grid(row=1, column=0)

        # Use a Combobox for the song dropdown
        song_values = df['name'].unique().tolist()
        self.song_combobox = ttk.Combobox(master, values=song_values)
        self.song_combobox.grid(row=1, column=1)

        self.predict_button = ttk.Button(master, text="Predict Popularity",
command=self.predict_popularity)
        self.predict_button.grid(row=2, column=0, columnspan=2)

    def predict_popularity(self):
        artist = self.artist_combobox.get()
        song = self.song_combobox.get()

        if not artist or not song:
            print("Please select both artist and song.")
            return

        result = songPopularityPrediction(artist, song)

        if result is not None:
            pred, actual = result
            result_text = f"Predicted popularity: {pred}\nActual popularity: {actual}"
            self.show_result(result_text)

    def show_result(self, result_text):
        result_label = ttk.Label(self.master, text=result_text)
```

```
    result_label.grid(row=3, column=0, columnspan=2)

# Assuming 'df', 'features', and 'rfr' are defined before calling this function

root = tk.Tk()
app = SongPopularityPredictionGUI(root)
root.mainloop()
```

7. TESTING

Testing is very vital for any system to be successfully implemented. The common view is that it is performed to prove that there are no errors in a program, Therefore the most useful and practical approach is with the explicit intention of finding the errors. The system is tested experimentally to ensure that the software does not fail. The system is run according to its specifications and in the way the user expects. Following testing practices are used. The system will process as normal input preparation of test-sample data.

TEST CASES

input	Expected result	Actual Output	Pass/Fail
Artist Name – Jonas Brothers Pop_rating - medium	Popular	Popular	Pass
Artist Name – The Weekend Pop_rating – popular	Popular	Popular	Pass
Artist Name – Shakira Pop_rating – popular	Popular	Popular	Pass
Artist Name – tame Impala Pop_rating – Popular	Medium	Medium	Pass

Artist Name – Chase Rice Song Name – I am with you	Actual Popularity – 25.94 Predicted Popularity – 25.94	Actual Popularity – 25.94 Predicted Popularity – 26	Fail
Artist Name – Justin Bieber Song Name – Love me like you do	Actual Popularity – 85 Predicted Popularity – 85	Actual Popularity – 85 Predicted Popularity – 84	Fail
Artist Name – Chase Rice Song Name – Thunder with us	Actual Popularity – 65.94 Predicted Popularity – 65.94	Actual Popularity – 25.94 Predicted Popularity – 23	Fail
Artist Name – Justin Bieber Song Name – The Chainsmokers	Actual Popularity – 85.90 Predicted Popularity – 85.90	Actual Popularity – 95.90 Predicted Popularity – 74	Fail
Artist Name – Chase Rice Song Name – Those who were with us	Actual Popularity – 76.94 Predicted Popularity – 76.94	Actual Popularity – 76.94 Predicted Popularity – 56	Fail

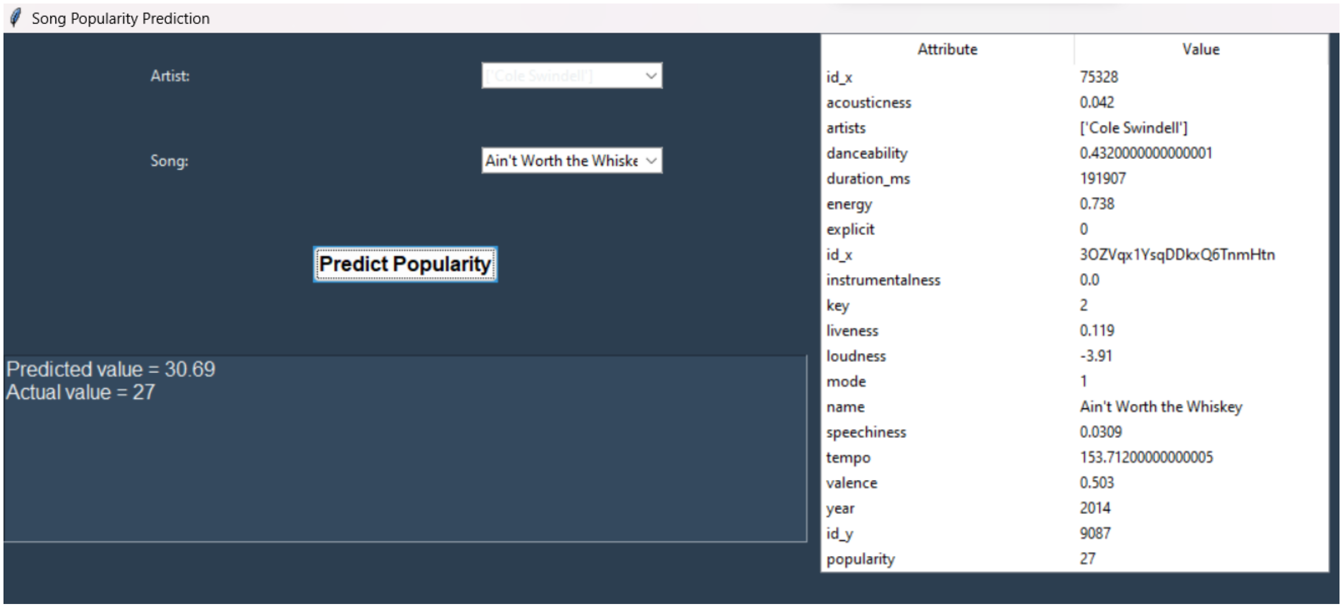
Song Popularity Prediction
Artist:
Song:
Ain't Worth the Whiske
Predict Popularity
Predicted value = 30.69
Actual value = 27
Attribute
Value
id_x
75328
acousticness
0.042
artists
['Cole Swindell']
danceability
0.4320000000000001
duration_ms
191907
energy
0.738
explicit
0
id_x
3OZVqx1YsqDDkxQ6TnmHtn
instrumentalness
0.0
key
2
liveness
0.119
loudness
-3.91
mode
1
name
Ain't Worth the Whiskey
speechiness
0.0309
tempo
153.71200000000005
valence
0.503
year
2014
id_y
9087
popularity
27

Fig 7.1

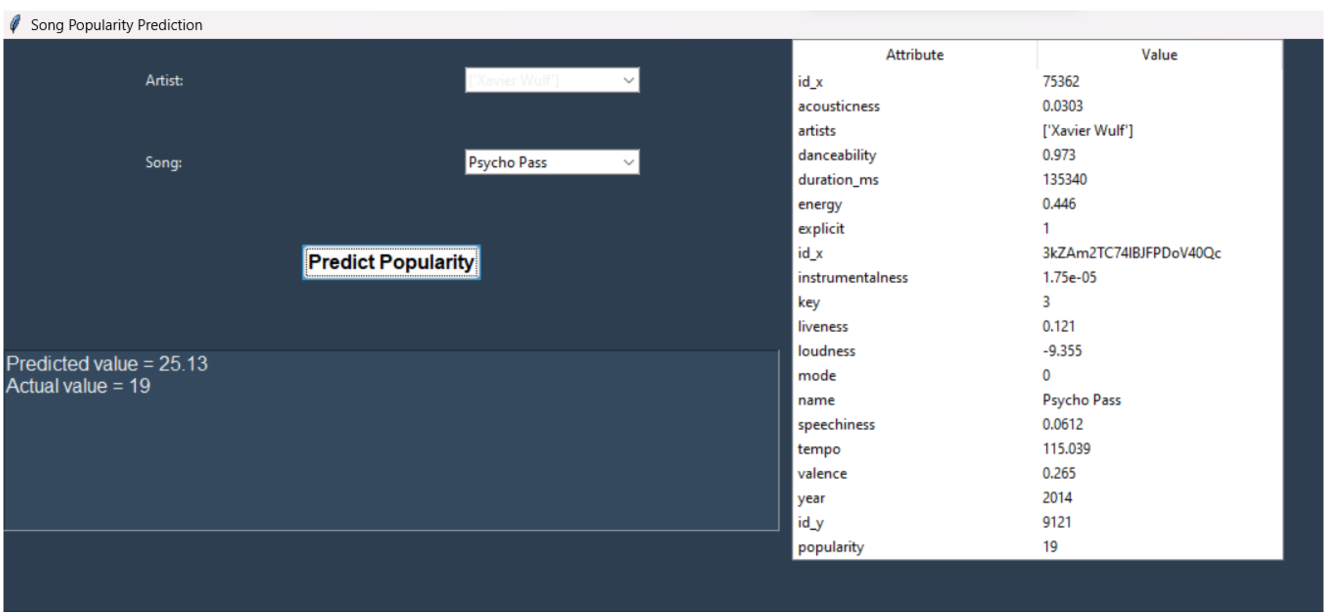
Song Popularity Prediction
Artist:
Song:
Psycho Pass
Predict Popularity
Predicted value = 25.13
Actual value = 19
Attribute
Value
id_x
75362
acousticness
0.0303
artists
['Xavier Wulf']
danceability
0.973
duration_ms
135340
energy
0.446
explicit
1
id_x
3kZAm2TC74IBJFPDoV40Qc
instrumentalness
1.75e-05
key
3
liveness
0.121
loudness
-9.355
mode
0
name
Psycho Pass
speechiness
0.0612
tempo
115.039
valence
0.265
year
2014
id_y
9121
popularity
19

Fig 7.2

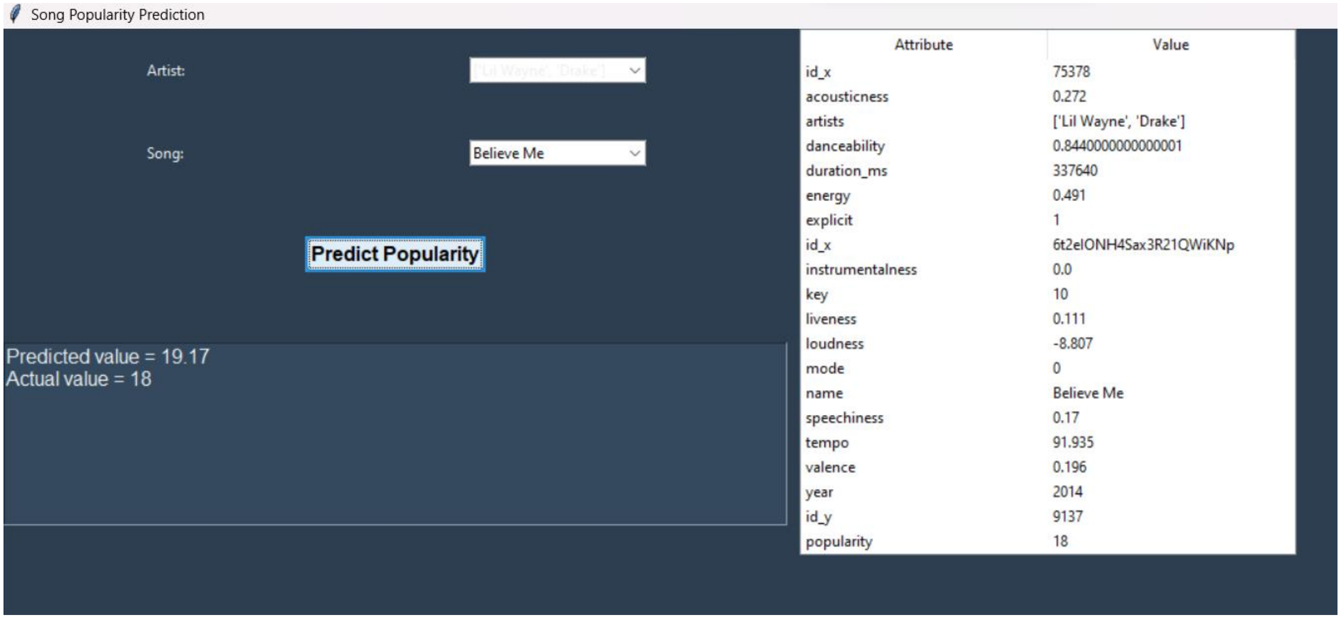
Song Popularity Prediction
Artist:
Song:
Believe Me
Predict Popularity
Predicted value = 19.17
Actual value = 18
Attribute
Value
id_x
75378
acousticness
0.272
artists
['Lil Wayne', 'Drake']
danceability
0.8440000000000001
duration_ms
337640
energy
0.491
explicit
1
id_x
6t2elONH4Sax3R21QWiKNp
instrumentalness
0.0
key
10
liveness
0.111
loudness
-8.807
mode
0
name
Believe Me
speechiness
0.17
tempo
91.935
valence
0.196
year
2014
id_y
9137
popularity
18

Fig 7.3

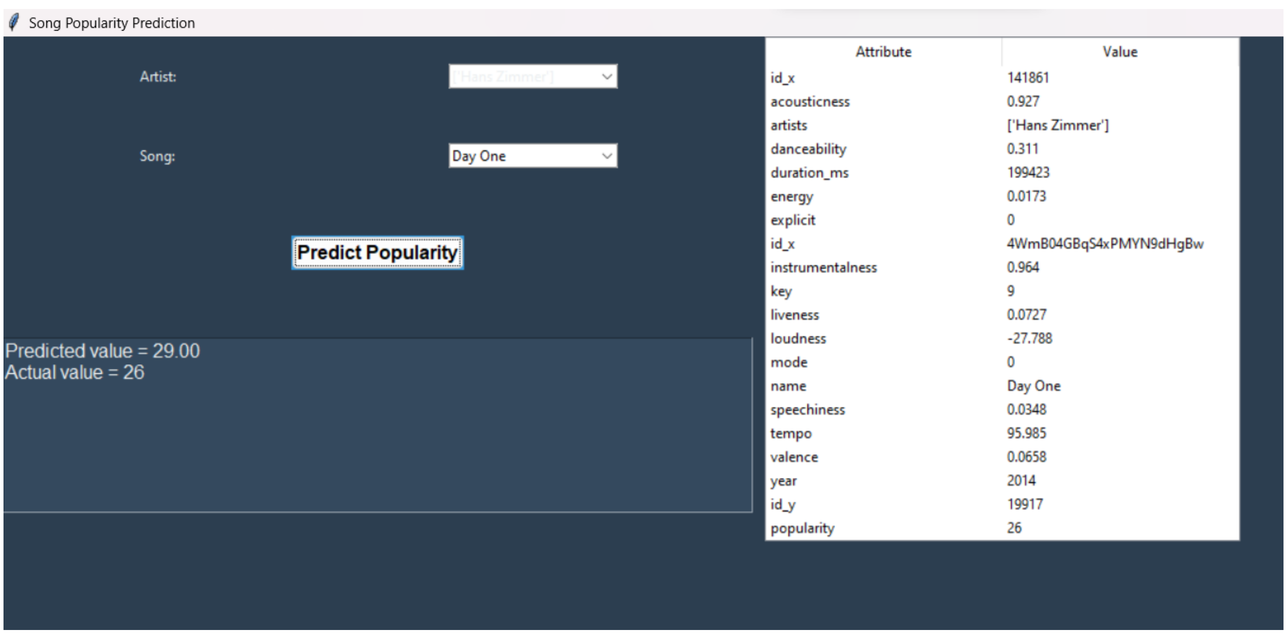
Song Popularity Prediction
Artist:
Song:
Day One
Predict Popularity
Predicted value = 29.00
Actual value = 26
Attribute
Value
id_x
141861
acousticness
0.927
artists
['Hans Zimmer']
danceability
0.311
duration_ms
199423
energy
0.0173
explicit
0
id_x
4WmB04GBqS4xPMYN9dHgBw
instrumentalness
0.964
key
9
liveness
0.0727
loudness
-27.788
mode
0
name
Day One
speechiness
0.0348
tempo
95.985
valence
0.0658
year
2014
id_y
19917
popularity
26

Fig 7.4

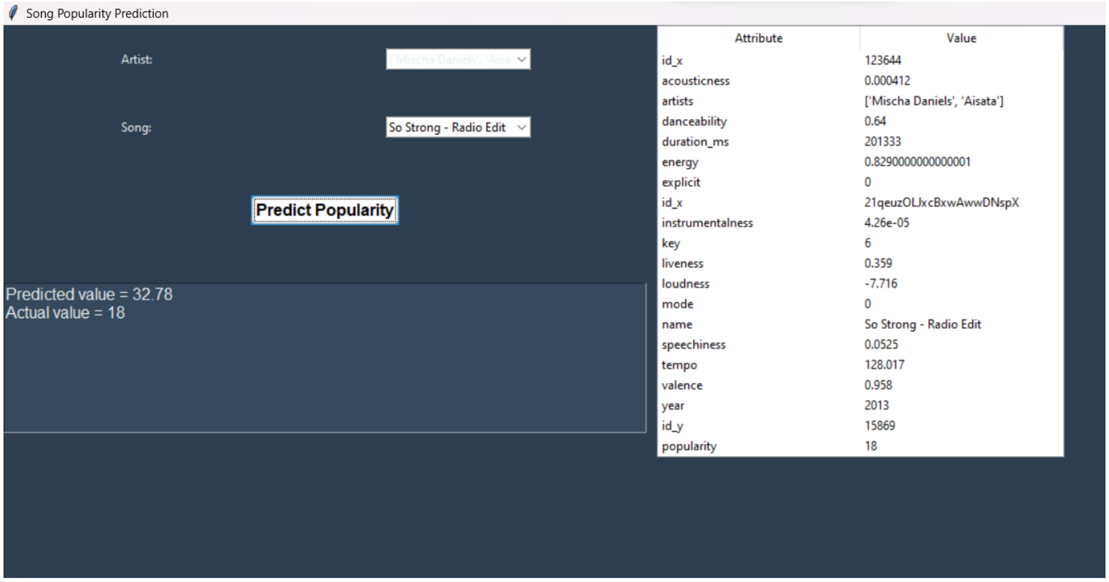
Song Popularity Prediction
Artist:
Song:
So Strong - Radio Edit
Predict Popularity
Predicted value = 32.78
Actual value = 18
Attribute
Value
id_x
123644
acousticness
0.000412
artists
['Mischa Daniels', 'Aisata']
danceability
0.64
duration_ms
201333
energy
0.8290000000000001
explicit
0
id_x
21qeuzOLJxcBxwAwwDNspX
instrumentalness
4.26e-05
key
6
liveness
0.359
loudness
-7.716
mode
0
name
So Strong - Radio Edit
speechiness
0.0525
tempo
128.017
valence
0.958
year
2013
id_y
15869
popularity
18

Fig 7.5

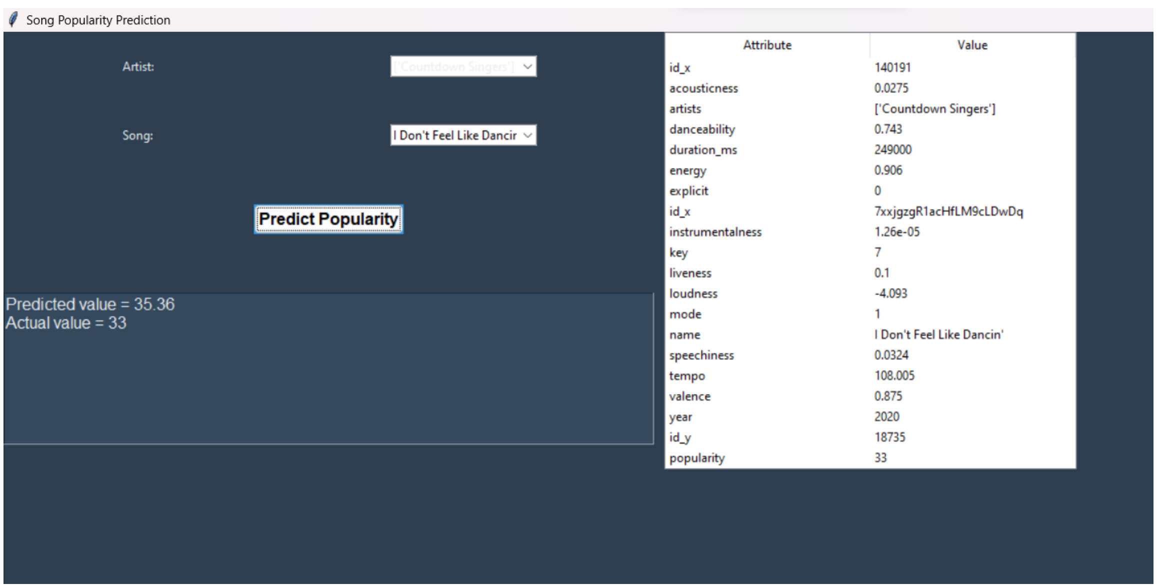
Song Popularity Prediction
Artist:
Song:
I Don't Feel Like Dancir
Predict Popularity
Predicted value = 35.36
Actual value = 33
Attribute
Value
id_x
140191
acousticness
0.0275
artists
['Countdown Singers']
danceability
0.743
duration_ms
249000
energy
0.906
explicit
0
id_x
7xxjgzgR1acHfLM9cLDwDq
instrumentalness
1.26e-05
key
7
liveness
0.1
loudness
-4.093
mode
1
name
I Don't Feel Like Dancin'
speechiness
0.0324
tempo
108.005
valence
0.875
year
2020
id_y
18735
popularity
33

Fig 7.6

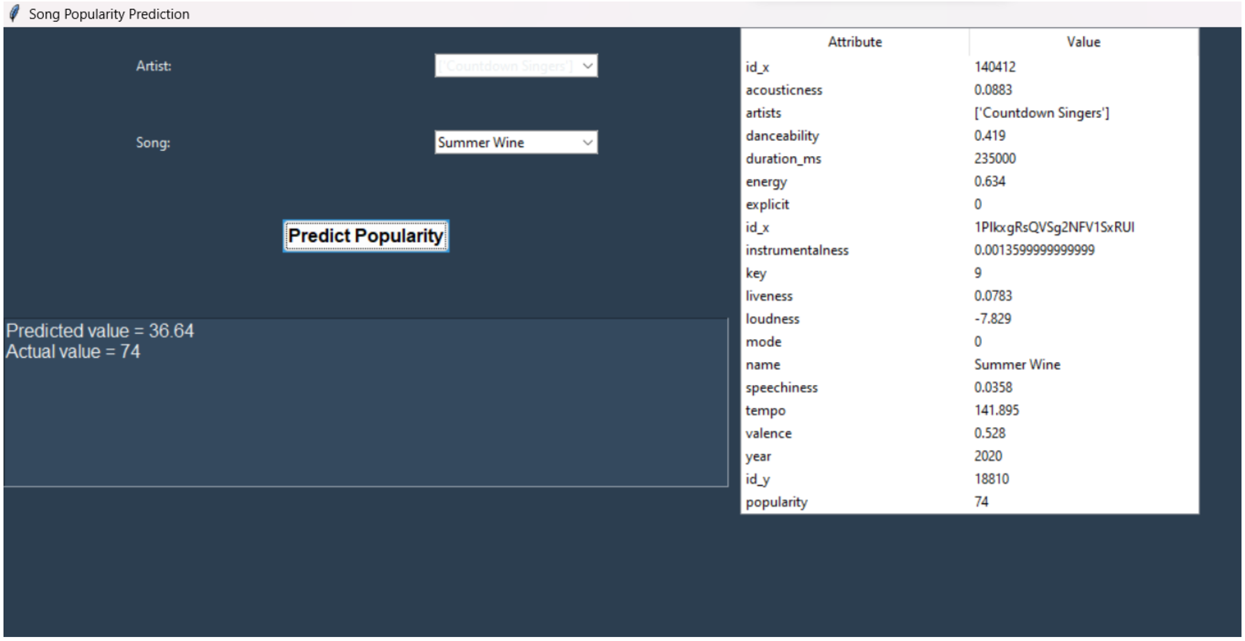
Song Popularity Prediction
Artist:
Song:
Summer Wine
Predict Popularity
Predicted value = 36.64
Actual value = 74
Attribute
Value
id_x 140412
acousticness 0.0883
artists ['Countdown Singers']
danceability 0.419
duration_ms 235000
energy 0.634
explicit 0
id_x 1PlkxgRsQVSg2NFV1SxRUI
instrumentalness 0.0013599999999999
key 9
liveness 0.0783
loudness -7.829
mode 0
name Summer Wine
speechiness 0.0358
tempo 141.895
valence 0.528
year 2020
id_y 18810
popularity 74

Fig 7.7

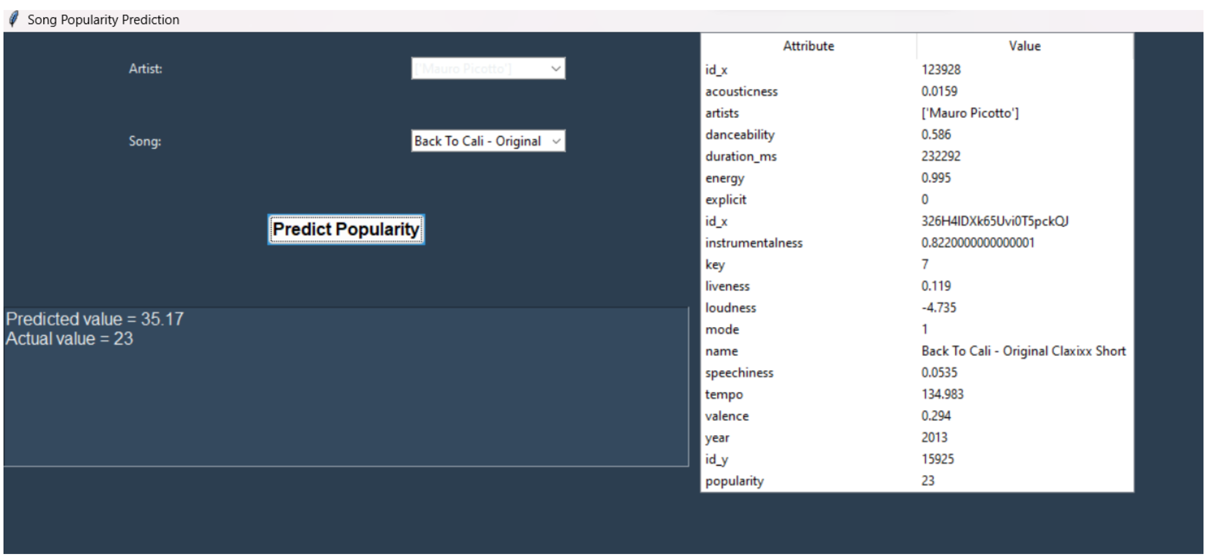
Song Popularity Prediction
Artist:
Song:
Back To Cali - Original
Predict Popularity
Predicted value = 35.17
Actual value = 23
Attribute
Value
id_x 123928
acousticness 0.0159
artists ['Mauro Picotto']
danceability 0.586
duration_ms 232292
energy 0.995
explicit 0
id_x 326H4IDXk65Uvi0T5pckQJ
instrumentalness 0.8220000000000001
key 7
liveness 0.119
loudness -4.735
mode 1
name Back To Cali - Original Claxixx Short
speechiness 0.0535
tempo 134.983
valence 0.294
year 2013
id_y 15925
popularity 23

Fig 7.8

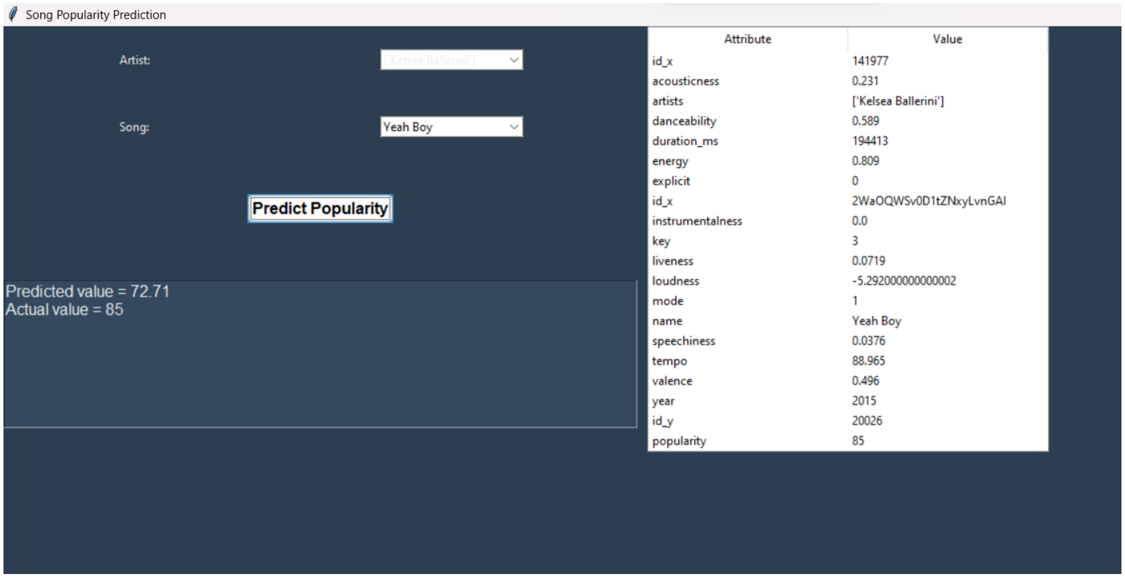

Fig 7.9

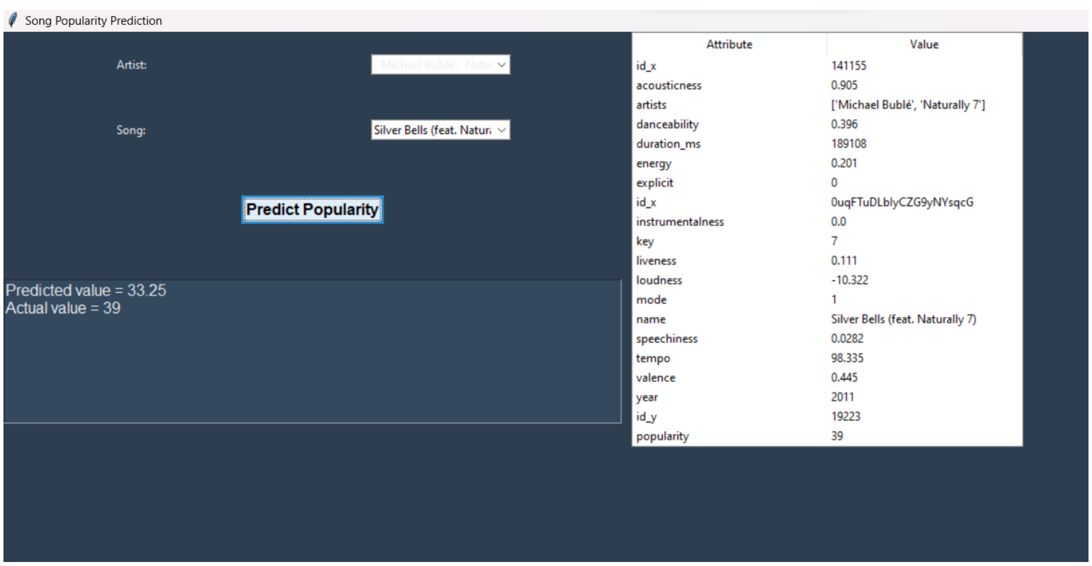

Fig 7.10

8. OUTPUT

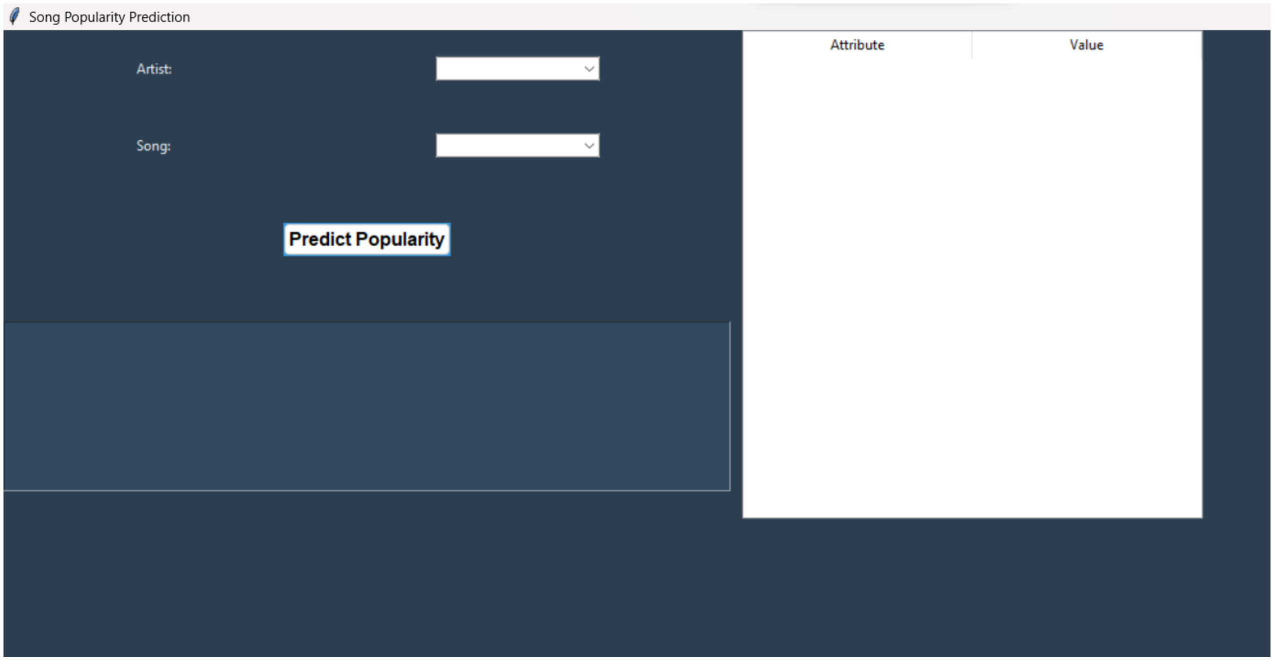

Fig 8.1

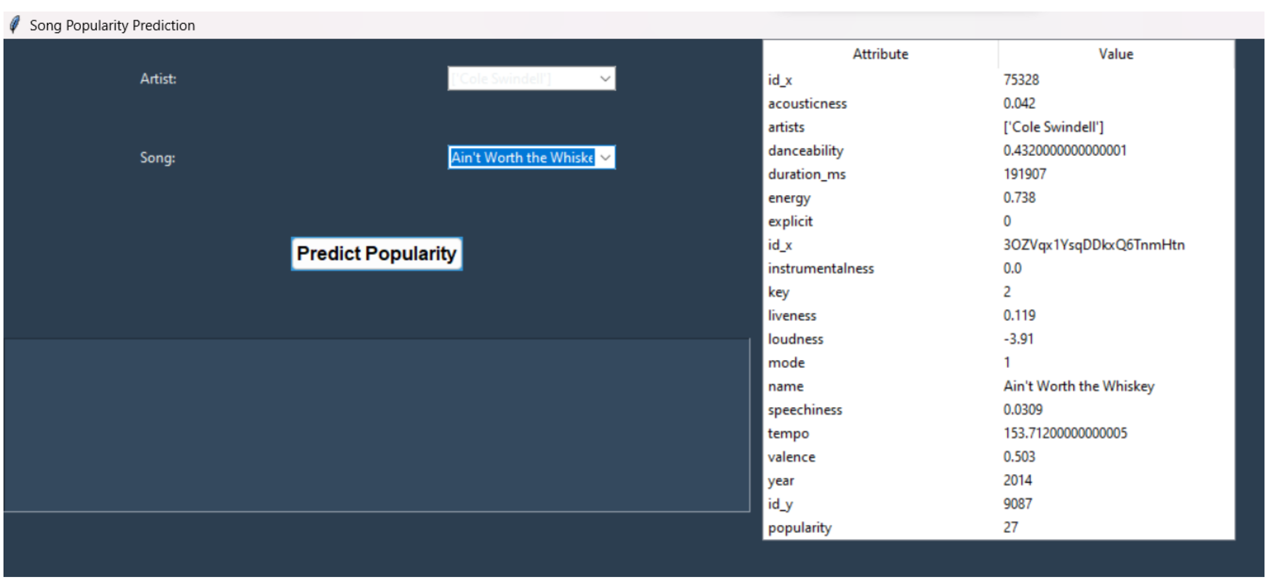

Fig 8.2

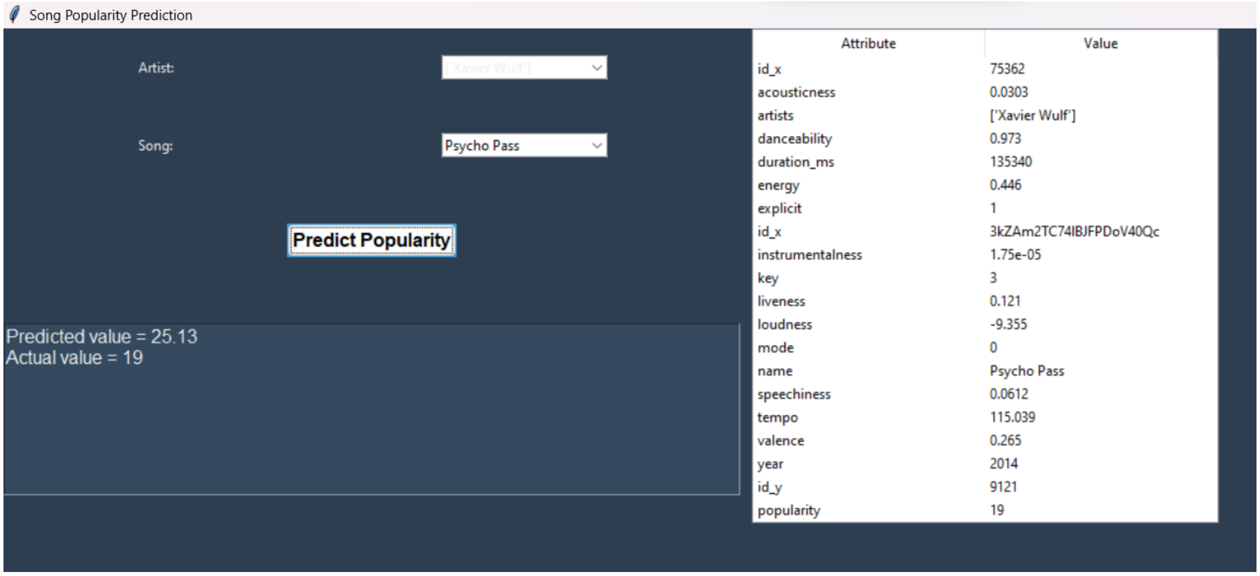
Song Popularity Prediction
Artist:
['Xavier Wulf']
Song:
Psycho Pass
Predict Popularity
Predicted value = 25.13
Actual value = 19
Attribute
Value
id_x
75362
acousticness
0.0303
artists
['Xavier Wulf']
danceability
0.973
duration_ms
135340
energy
0.446
explicit
1
id_x
3kZAm2TC74IBJFPDoV40Qc
instrumentalness
1.75e-05
key
3
liveness
0.121
loudness
-9.355
mode
0
name
Psycho Pass
speechiness
0.0612
tempo
115.039
valence
0.265
year
2014
id_y
9121
popularity
19

Fig 8.3

9. LIMITATIONS

While the proposed Song Popularity Prediction project offers valuable insights and solutions, it's important to acknowledge its limitations. Here are some potential constraints and challenges associated with the project

9.1 Data Bias and Representativeness

The predictive model heavily relies on the quality and representativeness of the training data. If the dataset is biased towards certain genres, time periods, or cultural contexts, the model may exhibit limitations in accurately predicting the popularity of songs outside these biases.

9.2 Evolving Music Trends

The music industry is dynamic, with trends evolving rapidly. The model's training data may become outdated, impacting its ability to predict the popularity of songs in response to emerging trends or shifts in consumer preferences.

9.3 Complexity of Musical Appeal

The model may struggle to capture the nuanced and subjective aspects of musical appeal. Elements such as lyrical content, emotional resonance, and cultural significance may not be fully represented by quantitative features, limiting the model's ability to predict the popularity of songs with these nuanced qualities.

10. FUTURE APPLICATIONS

10.1 Trend Analysis for Music Industry Reports

Use the model to analyze trends in music popularity for the creation of industry reports. Provide insights into genre shifts, emerging artists, and market dynamics, supporting industry professionals in strategic decision-making.

10.2 Targeted Licensing for Advertisements

Collaborate with advertising agencies to predict the popularity of songs for use in commercials and advertisements. Optimize the licensing process by selecting music that aligns with the target audience and enhances brand messaging.

10.3 Event Planning for Music Venues

Assist music venues in planning events and concerts by predicting the popularity of potential performers. Optimize bookings to attract larger audiences and enhance the venue's reputation.

10.4 Playlist Generation for Radio Stations

Collaborate with radio stations to use the model in generating playlists. Predict the popularity of songs to ensure a dynamic and engaging selection for listeners.

10.5 Cross-industry Collaboration

Explore collaborations with industries beyond music, such as fashion, gaming, or technology, to leverage the predictive model for enhancing product launches, user experiences, or brand collaborations.

BIBLIOGRAPHY

[1] Keiji Yanai, Takuma Maruyama and Yoshiyuki Kawano, A Cooking Recipe Recommendation System with Visual Recognition of Food Ingredients, The University of Electro-Communications, Tokyo, Japan2. APreference-Based Restaurant Recommendation System for Individuals and Groups, page 28, 2014

[2] Taehee Lee, Stefano Soatto, Learning and Matching Multiscale Template Descriptors for Real- Time Detection, Localization and Tracking, Computer Science Department University of California,
Los Angeles, CA 90095, page 1457

[3] Souvik Debnath Machine Learning Based Recommendation System, Department of Computer Science and Engineering Indian Institute of Technology, , 2008

[4] Gediminas Adomavicius and Alexander Tuzhilin, Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions, Member, IEEE, page 734, 2005