Necessary imports

In [2]:
```
import spotipy
import spotipy.util as util
from spotipy.oauth2 import SpotifyClientCredentials
import spotipy.oauth2 as oauth2
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
```

unique client id and client secret

In [3]:
```
clientID = 'd2ccc643900f46b4be5309cf086db36d'
```

In [4]:
```
clientSecret = '38a0accbac3a4d6b928c07757feb106e'
```

create spotify object

In [5]:
```
ccm = SpotifyClientCredentials(client_id=clientID, client_secret=clientSecret
sp = spotipy.Spotify(client_credentials_manager = ccm)
```

function to get list of song uris based on a playlist uri

In [6]:
```
def get_song_URI(mood_uri):
    mood_99 = sp.playlist_tracks(mood_uri)
    mood_songs = mood_99['items']
    mood_song_uri = []
    while mood_99['next']:
        mood_99 = sp.next(mood_99)
        mood_songs.extend(mood_99['items'])
    for x in mood_songs:
        mood_song_uri.append(x['track']['uri'])
    return mood_song_uri
```

Take down playlist uris and use function to get list of song uris for each playlist

In [7]:
```
sad_URI = '37i9dQZF1DWSqBruwoIXkA' #sad hour
happy_URI = '37i9dQZF1DXdPec7aLTmlC' #Happy hits!
angry_URI = '07JztNEdtMFhc6hoLzQLsH' #RAGE! JUST PURE RAGE!

sad_songs_URI = get_song_URI(sad_URI)
happy_songs_URI = get_song_URI(happy_URI)
angry_songs_URI = get_song_URI(angry_URI)
```

function to get features for each song in a list based on the song's uri

In [8]:
```
def get_features(mood_song_uri):
    mood_features = []
    for x in mood_song_uri:
        mood_features.append(sp.audio_features(x)[0])
    return mood_features
```

save list of all features of all songs for each mood

In [9]:
```
sad_features = get_features(sad_songs_URI)

happy_features = get_features(happy_songs_URI)

angry_features = get_features(angry_songs_URI)
```

save lists to dataframes

In [10]:
```
sad_df = pd.DataFrame(sad_features)

happy_df = pd.DataFrame(happy_features)

angry_df = pd.DataFrame(angry_features)
```

add column to each df stating the mood

In [12]:
```
for x in sad_df:
    sad_df['genre'] = 'sad'

for x in happy_df:
    happy_df['genre'] = 'happy'

for x in angry_df:
    angry_df['genre'] = 'angry'
```

add extra playlists

In [13]:
```
sad_2_uri = '37i9dQZF1DWW2hj3ZtMbuO' #sad girl starter pack
sad_3_uri = '37i9dQZF1DWVV27DiNWxkR' #sad indie
sad_4_uri = '37i9dQZF1DX7qK8ma5wgG1' #Sad Songs

sad_songs_2_uri = get_song_URI(sad_2_uri)
sad_songs_3_uri = get_song_URI(sad_3_uri)
sad_songs_4_uri = get_song_URI(sad_4_uri)
```

In [14]:
```
happy_2_uri = '37i9dQZF1DX0UrRvztWcAU' #Wake Up Happy
happy_3_uri = '37i9dQZF1DX84kJlLdo9vT' #Happy Days
happy_4_uri = '37i9dQZF1DWZKuerrwoAGz' #Happy favorites

happy_songs_2_uri = get_song_URI(happy_2_uri)
happy_songs_3_uri = get_song_URI(happy_3_uri)
happy_songs_4_uri = get_song_URI(happy_4_uri)
```

In [15]:
```
angry_2_uri = '1hmtcQfAflMKwcbN0qqj4U'  #angry women playlist

angry_songs_2_uri = get_song_URI(angry_2_uri)


angry_2_features = get_features(angry_songs_2_uri)

angry_2_df = pd.DataFrame(angry_2_features)

for x in angry_2_df:
    angry_2_df['genre'] = 'angry'
```

In [16]:
```
sad_2_features = get_features(sad_songs_2_uri)
sad_3_features = get_features(sad_songs_3_uri)
sad_4_features = get_features(sad_songs_4_uri)
happy_2_features = get_features(happy_songs_2_uri)
happy_3_features = get_features(happy_songs_3_uri)
happy_4_features = get_features(happy_songs_4_uri)
```

In [17]:
```
sad_2_df = pd.DataFrame(sad_2_features)
sad_3_df = pd.DataFrame(sad_3_features)
sad_4_df = pd.DataFrame(sad_4_features)
happy_2_df = pd.DataFrame(happy_2_features)
happy_3_df = pd.DataFrame(happy_3_features)
happy_4_df = pd.DataFrame(happy_4_features)
```

In [18]:
```
for x in sad_2_df:
    sad_2_df['genre'] = 'sad'

for x in sad_3_df:
    sad_3_df['genre'] = 'sad'

for x in sad_4_df:
    sad_4_df['genre'] = 'sad'

for x in happy_2_df:
    happy_2_df['genre'] = 'happy'

for x in happy_3_df:
    happy_3_df['genre'] = 'happy'

for x in happy_4_df:
    happy_4_df['genre'] = 'happy'
```

consolodate all dataframes into one dataframe

In [19]:
```
emotion_df = happy_df.append(angry_df.append(sad_df))
```

```
C:\Users\manda\AppData\Local\Temp\ipykernel_288\348178859.py:1: FutureWarni
ng: The frame.append method is deprecated and will be removed from pandas i
n a future version. Use pandas.concat instead.
  emotion_df = happy_df.append(angry_df.append(sad_df))
```

In [20]:
```
emotion_plus_df = emotion_df.append(sad_2_df.append(sad_3_df.append(sad_4_df.
```

```
C:\Users\manda\AppData\Local\Temp\ipykernel_288\1195311430.py:1: FutureWarn
ing: The frame.append method is deprecated and will be removed from pandas
in a future version. Use pandas.concat instead.
  emotion_plus_df = emotion_df.append(sad_2_df.append(sad_3_df.append(sad_4
_df.append(happy_2_df.append(happy_3_df.append(happy_4_df.append(angry_2_d
f)))))))
C:\Users\manda\AppData\Local\Temp\ipykernel_288\1195311430.py:1: FutureWarn
ing: The frame.append method is deprecated and will be removed from pandas
in a future version. Use pandas.concat instead.
  emotion_plus_df = emotion_df.append(sad_2_df.append(sad_3_df.append(sad_4
_df.append(happy_2_df.append(happy_3_df.append(happy_4_df.append(angry_2_d
f)))))))
C:\Users\manda\AppData\Local\Temp\ipykernel_288\1195311430.py:1: FutureWarn
ing: The frame.append method is deprecated and will be removed from pandas
in a future version. Use pandas.concat instead.
  emotion_plus_df = emotion_df.append(sad_2_df.append(sad_3_df.append(sad_4
_df.append(happy_2_df.append(happy_3_df.append(happy_4_df.append(angry_2_d
f)))))))
C:\Users\manda\AppData\Local\Temp\ipykernel_288\1195311430.py:1: FutureWarn
ing: The frame.append method is deprecated and will be removed from pandas
in a future version. Use pandas.concat instead.
  emotion_plus_df = emotion_df.append(sad_2_df.append(sad_3_df.append(sad_4
_df.append(happy_2_df.append(happy_3_df.append(happy_4_df.append(angry_2_d
f)))))))
C:\Users\manda\AppData\Local\Temp\ipykernel_288\1195311430.py:1: FutureWarn
ing: The frame.append method is deprecated and will be removed from pandas
in a future version. Use pandas.concat instead.
  emotion_plus_df = emotion_df.append(sad_2_df.append(sad_3_df.append(sad_4
_df.append(happy_2_df.append(happy_3_df.append(happy_4_df.append(angry_2_d
f)))))))
C:\Users\manda\AppData\Local\Temp\ipykernel_288\1195311430.py:1: FutureWarn
ing: The frame.append method is deprecated and will be removed from pandas
in a future version. Use pandas.concat instead.
  emotion_plus_df = emotion_df.append(sad_2_df.append(sad_3_df.append(sad_4
_df.append(happy_2_df.append(happy_3_df.append(happy_4_df.append(angry_2_d
f)))))))
```

In [ ]:
```
emotion_plus_df.to_csv('csv4.csv')
```

select only relevant features

In [21]:
```
trim_emotion_plus_df = emotion_plus_df[['danceability', 'energy', 'key', 'lou
```

shuffle songs

In [22]:
```
df_trim_plus_shuffle = trim_emotion_plus_df.sample(frac=1)
```

In [ ]:
```
df_trim_plus_shuffle.to_csv('full_songs_calmless.csv')
```

scale the data

In [23]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
numeric = df_trim_plus_shuffle.select_dtypes(exclude=['object'])
df_trim_plus_shuffle[numeric.columns] = scaler.fit_transform(numeric)
df_trim_plus_shuffle
```

Out[23]:

	danceability	energy	key	loudness	mode	speechiness	acousticness	instr
19	-0.833150	-0.871082	-0.817425	-1.695880	0.630202	-0.336436	0.819070	
23	-1.431959	-0.335272	-1.373531	-1.133017	0.630202	-0.421490	1.764491	
92	-0.107927	0.373516	-0.261318	0.474237	-1.586794	-0.567785	-0.339535	
70	-1.032753	1.208873	1.685054	0.495418	0.630202	3.937849	-0.768991	
126	0.650563	0.643530	0.016735	1.159671	0.630202	-0.148181	-0.962926	
...	...	...	...	...	...	...	...	
2	0.637256	-1.145315	1.128947	-0.879968	0.630202	-0.435099	0.877772	
41	-1.179128	-1.900512	1.128947	-3.067595	0.630202	-0.430563	1.965316	
68	1.149570	0.141472	-1.095478	-0.822919	0.630202	-0.009825	-0.733461	
12	-0.766616	0.040217	0.016735	0.046370	-1.586794	-0.590466	-0.675067	
90	-0.966219	0.031779	-1.095478	0.515470	0.630202	-0.316022	0.927206	

1080 rows × 14 columns

In [ ]:
```
df_trim_plus_shuffle.to_csv('full_songs_no_calm.csv')
```

split data into training and testing set

In [24]:
```
from sklearn.model_selection import train_test_split
labels = df_trim_plus_shuffle[['genre']]
features = df_trim_plus_shuffle.loc[:, df_trim_plus_shuffle.columns != 'genre
X_train, X_test, y_train, y_test = train_test_split(features, labels, random_
```

import various classifiers

In [25]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.linear_model import RidgeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import BaggingClassifier
```

beginning of random forest grid search

In [26]:
```
param_grid = {'max_depth':[2, 10, 17, 25, 32],
             'n_estimators':[2, 20, 50, 100, 200],
             'min_samples_split':[2, 7, 11, 16, 20]}
grid_search_rf = GridSearchCV(RandomForestClassifier(random_state = 42),
                              param_grid, verbose = 1,
                              cv = 3)
grid_search_rf.fit(X_train, y_train['genre'])
print(grid_search_rf.best_params_)
```

```
Fitting 3 folds for each of 125 candidates, totalling 375 fits
{'max_depth': 10, 'min_samples_split': 2, 'n_estimators': 200}
```

In [27]:
```
param_grid = {'max_depth':[2,5,8,11,14,17],
             'n_estimators':[100,140,180,220,260,300],
             'min_samples_split':[2,3,4,5,6,7]}
grid_search_rf = GridSearchCV(RandomForestClassifier(random_state = 42),
                              param_grid, verbose = 1,
                              cv = 3)
grid_search_rf.fit(X_train, y_train['genre'])
print(grid_search_rf.best_params_)
```

```
Fitting 3 folds for each of 216 candidates, totalling 648 fits
{'max_depth': 8, 'min_samples_split': 6, 'n_estimators': 140}
```

In [28]:
```
param_grid = {'max_depth':[5,6,7,8,9,10,11],
             'n_estimators':[100,116,132,148,164,180],
             'min_samples_split':[6]}
grid_search_rf = GridSearchCV(RandomForestClassifier(random_state = 42),
                              param_grid, verbose = 1,
                              cv = 3)
grid_search_rf.fit(X_train, y_train['genre'])
print(grid_search_rf.best_params_)
```

```
Fitting 3 folds for each of 42 candidates, totalling 126 fits
{'max_depth': 8, 'min_samples_split': 6, 'n_estimators': 148}
```

In [29]:
```
param_grid = {'max_depth':[8],
              'n_estimators':[132,133,134,135,136,137,138,139,140,141,142,143,
              'min_samples_split':[6]}
grid_search_rf = GridSearchCV(RandomForestClassifier(random_state = 42),
                              param_grid, verbose = 1,
                              cv = 3)
grid_search_rf.fit(X_train, y_train['genre'])
print(grid_search_rf.best_params_)
```

```
Fitting 3 folds for each of 33 candidates, totalling 99 fits
{'max_depth': 8, 'min_samples_split': 6, 'n_estimators': 154}
```

max depth: 8 min_samples_split: 6 n_estimators: 154

In [30]:
```
from sklearn.metrics import accuracy_score
```

In [31]:
```
op_rf = RandomForestClassifier(max_depth = 8, n_estimators = 154,
                               min_samples_split = 6, random_state = 42)
op_rf.fit(X_train, y_train['genre'])
```

Out[31]:
```
RandomForestClassifier
RandomForestClassifier(max_depth=8, min_samples_split=6, n_estimators=154,
                       random_state=42)
```

In [32]:
```
rf_test_acc = accuracy_score(y_test, op_rf.predict(X_test))
rf_test_acc
```

Out[32]: 0.8472222222222222

beginning of gradient boosting classifier grid search

In [33]:
```
param_grid = {'max_depth': [2, 10, 17, 24, 32],
              'n_estimators': [2,41,81,121,160,200],
              'learning_rate':[.01, .2, .4, .6, .8, 1]}
grid_search_gb = GridSearchCV(GradientBoostingClassifier(random_state = 42),
                              param_grid, verbose = 1,
                              cv = 3)
grid_search_gb.fit(X_train, y_train['genre'])
print(grid_search_gb.best_params_)
```

```
Fitting 3 folds for each of 180 candidates, totalling 540 fits
{'learning_rate': 0.2, 'max_depth': 2, 'n_estimators': 81}
```

In [34]:
```
param_grid = {'max_depth': [2, 5,8,11,14,17],
              'n_estimators': [41,57,73,89,105,121],
              'learning_rate':[.01,.1,.16,.24,.32,.4]}
grid_search_gb = GridSearchCV(GradientBoostingClassifier(random_state = 42),
                              param_grid, verbose = 1,
                              cv = 3)
grid_search_gb.fit(X_train, y_train['genre'])
print(grid_search_gb.best_params_)
```

```
Fitting 3 folds for each of 216 candidates, totalling 648 fits
{'learning_rate': 0.4, 'max_depth': 5, 'n_estimators': 41}
```

In [35]:
```
param_grid = {'max_depth': [2,3,4,5,6,7,8],
              'n_estimators': [41,46,52,57],
              'learning_rate':[.32,.33,.34,.35,.36,.37,.38,.39,.4]}
grid_search_gb = GridSearchCV(GradientBoostingClassifier(random_state = 42),
                              param_grid, verbose = 1,
                              cv = 3)
grid_search_gb.fit(X_train, y_train['genre'])
print(grid_search_gb.best_params_) #taking too long, come back
```

```
Fitting 3 folds for each of 252 candidates, totalling 756 fits
{'learning_rate': 0.36, 'max_depth': 7, 'n_estimators': 57}
```

In [36]:
```
param_grid = {'max_depth': [7],
              'n_estimators': [52,53,54,55,56,57],
              'learning_rate':[.36]}
grid_search_gb = GridSearchCV(GradientBoostingClassifier(random_state = 42),
                              param_grid, verbose = 1,
                              cv = 3)
grid_search_gb.fit(X_train, y_train['genre'])
print(grid_search_gb.best_params_)
```

```
Fitting 3 folds for each of 6 candidates, totalling 18 fits
{'learning_rate': 0.36, 'max_depth': 7, 'n_estimators': 57}
```

learning rate: .36 max_depth: 7 n_estimators: 57

In [38]:
```
op_gbc = GradientBoostingClassifier(max_depth = 7, n_estimators = 57,
                               learning_rate = .36, random_state = 42)
op_gbc.fit(X_train, y_train['genre'])
```

Out[38]:
```
GradientBoostingClassifier
GradientBoostingClassifier(learning_rate=0.36, max_depth=7, n_estimators=5
7,
                           random_state=42)
```

In [39]:
```
gbc_test_acc = accuracy_score(y_test, op_gbc.predict(X_test))
gbc_test_acc
```

Out[39]:
```
0.8611111111111112
```

beginning of knn classifier grid search

In [40]:
```
param_grid = {'weights' : ['uniform', 'distance'],
              'metric' : ['euclidean', 'manhattan', 'minkowski'],
              'n_neighbors' : [1, 5, 9, 13, 17, 21]}
grid_search_knn = GridSearchCV(KNeighborsClassifier(),
                               param_grid, verbose = 1,
                               cv = 3)
grid_search_knn.fit(X_train, y_train['genre'])
print(grid_search_knn.best_params_)
```

```
Fitting 3 folds for each of 36 candidates, totalling 108 fits
{'metric': 'manhattan', 'n_neighbors': 9, 'weights': 'distance'}
```

In [41]:
```
param_grid = {'weights' : ['distance'],
              'metric' : ['manhattan'],
              'n_neighbors' : [5,7,9,11,13]}
grid_search_knn = GridSearchCV(KNeighborsClassifier(),
                               param_grid, verbose = 1,
                               cv = 3)
grid_search_knn.fit(X_train, y_train['genre'])
print(grid_search_knn.best_params_)
```

```
Fitting 3 folds for each of 5 candidates, totalling 15 fits
{'metric': 'manhattan', 'n_neighbors': 11, 'weights': 'distance'}
```

metric: manhattan weights: distance n_neighbors: 11

In [42]:
```
op_knn = KNeighborsClassifier(metric = 'manhattan', weights = 'distance',
                              n_neighbors = 11)
op_knn.fit(X_train, y_train['genre'])
```

Out[42]:
```
KNeighborsClassifier
KNeighborsClassifier(metric='manhattan', n_neighbors=11, weights='distanc
e')
```

In [43]:
```
knn_test_acc = accuracy_score(y_test, op_knn.predict(X_test))
knn_test_acc
```

Out[43]:
```
0.8148148148148148
```

beginning of ridge classifier grid search

In [44]:
```
param_grid = {'alpha':[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]}
grid_search_rc = GridSearchCV(RidgeClassifier(random_state = 42),
                              param_grid, verbose = 1,
                              cv = 3)
grid_search_rc.fit(X_train, y_train['genre'])
print(grid_search_rc.best_params_)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
{'alpha': 0.1}
```

In [46]:
```
op_rc = RidgeClassifier(alpha = .1)
op_rc.fit(X_train, y_train['genre'])
```

Out[46]:
```
RidgeClassifier
RidgeClassifier(alpha=0.1)
```

In [47]:
```
rc_test_acc = accuracy_score(y_test, op_rc.predict(X_test))
rc_test_acc
```

Out[47]:
```
0.8425925925925926
```

beginning of svc grid search

In [48]:
```
param_grid = {'kernel' : ['poly', 'rbf', 'sigmoid'],
             'C' : [50, 10, 1.0, 0.1, 0.01]}
grid_search_svc = GridSearchCV(SVC(random_state = 42),
                              param_grid, verbose = 1,
                              cv = 3)
grid_search_svc.fit(X_train, y_train['genre'])
print(grid_search_svc.best_params_)
```

```
Fitting 3 folds for each of 15 candidates, totalling 45 fits
{'C': 1.0, 'kernel': 'rbf'}
```

In [49]:
```
param_grid = {'kernel' : ['rbf'],
             'C' : [.1,.8,1.5,2.2,2.9,3.6,4.3,5.1,5.8,6.5,7.1,7.9,8.6,9.3,10]
grid_search_svc = GridSearchCV(SVC(random_state = 42),
                              param_grid, verbose = 1,
                              cv = 3)
grid_search_svc.fit(X_train, y_train['genre'])
print(grid_search_svc.best_params_)
```

```
Fitting 3 folds for each of 15 candidates, totalling 45 fits
{'C': 1.5, 'kernel': 'rbf'}
```

In [50]:
```
param_grid = {'kernel' : ['rbf'],
              'C' : [.8,.9,1,1.1,1.2,1.3,1.4,1.5,1.6,1.7,1.8,1.9,2.0,2.1,2.2]}
grid_search_svc = GridSearchCV(SVC(random_state = 42),
                               param_grid, verbose = 1,
                               cv = 3)
grid_search_svc.fit(X_train, y_train['genre'])
print(grid_search_svc.best_params_)
```

```
Fitting 3 folds for each of 15 candidates, totalling 45 fits
{'C': 1, 'kernel': 'rbf'}
```

In [51]:
```
op_svc = SVC(kernel = 'rbf', C = 1 )
op_svc.fit(X_train, y_train['genre'])
```

Out[51]:
```
SVC
SVC(C=1)
```

In [52]:
```
svc_test_acc = accuracy_score(y_test, op_svc.predict(X_test))
svc_test_acc
```

Out[52]:
```
0.8379629629629629
```

beginning of bagging classifier grid search

In [53]:
```
param_grid = {'n_estimators' : [2,22,41,61,80,100],
              'max_samples': [.5,.6,.7,.8,.9,1.0],
              'max_features' : [.1,.2,.3,.4,.5,.6,.7,.8,.9,1]}
grid_search_bbc = GridSearchCV(BaggingClassifier(random_state = 42),
                               param_grid, verbose = 1,
                               cv = 3)
grid_search_bbc.fit(X_train, y_train['genre'])
print(grid_search_bbc.best_params_)
```

```
Fitting 3 folds for each of 360 candidates, totalling 1080 fits
{'max_features': 0.8, 'max_samples': 1.0, 'n_estimators': 100}
```

In [54]:
```
param_grid = {'n_estimators' : [80,84,87,91,95,98,102,105,109,113,116,120],
              'max_samples': [1.0],
              'max_features' : [.8]}
grid_search_bbc = GridSearchCV(BaggingClassifier(random_state = 42),
                               param_grid, verbose = 1,
                               cv = 3)
grid_search_bbc.fit(X_train, y_train['genre'])
print(grid_search_bbc.best_params_)
```

```
Fitting 3 folds for each of 12 candidates, totalling 36 fits
{'max_features': 0.8, 'max_samples': 1.0, 'n_estimators': 87}
```

In [55]:
```
param_grid = {'n_estimators' : [84,85,86,86,87,88,89,90,91],
              'max_samples': [1.0],
              'max_features' : [.8]}
grid_search_bbc = GridSearchCV(BaggingClassifier(random_state = 42),
                               param_grid, verbose = 1,
                               cv = 3)
grid_search_bbc.fit(X_train, y_train['genre'])
print(grid_search_bbc.best_params_)
```

```
Fitting 3 folds for each of 9 candidates, totalling 27 fits
{'max_features': 0.8, 'max_samples': 1.0, 'n_estimators': 90}
```

In [56]:
```
op_bbc = BaggingClassifier(n_estimators = 90,
                          max_samples = 1.0,
                          max_features = .8,
                          random_state = 42)
op_bbc.fit(X_train, y_train['genre'])
bbc_test_acc = accuracy_score(y_test, op_bbc.predict(X_test))
bbc_test_acc
```

Out[56]:
```
0.8703703703703703
```

get song uri based on artist and title

In [57]:
```
def searchSong(artist, title):
    query = f"artist:%{artist} track:%{title}"
    results = sp.search(query, type="track", limit=1)
    return results['tracks']['items'][0]['uri']
```

feed into prediction algorithm

In [58]:
```
def predictMood(uri):
    song_features = sp.audio_features(uri)[0]
    features_column = pd.DataFrame.from_dict(song_features, orient = 'index')
    features_row = features_column.T
    trim_row = features_row[['danceability', 'energy', 'key', 'loudness', 'mo
    #only_numeric= trim_row.select_dtypes(exclude=['object'])
    trim_row = scaler.transform(trim_row)
    return op_bbc.predict(trim_row) # this will be changed to model_in.predic
```

In [59]:
```
def moodPredictor(artist, title):
    try:
        x = (predictMood(searchSong(artist, title)))[0]
        print(f'It looks like this song is {x}!')
    except: print("Oops! I couldn't find that song. Did you spell everything
```

In [60]:
```
from joblib import dump, load
```

In [ ]:
```
dump(op_bbc, 'improved_model.joblib')
```

In [ ]:
```
dump(scaler, 'improved_scaler_scaler.joblib')
```