CLASSIFICATION PROBLEM

CS958- Group 46

Ayesha Michael Gaikwad- 202363500

Santosh Eknath Bangera- 202366764

Krish Hitesh Dasani- 202388555

Aishwarya Sanjay Kurhe- 202382549

Obaretin Abieyuwa- 202394985

INTRODUCTION

The project focuses on music genre classification using Machine Learning, targetting the prediction of song genres from Spotify dataset, which consists of many attributes like tempo, loudness, danceability etc. We conducted thorough parameter tuning, training and evaluation to achieve a high precision model. Our study proved to be benefecial for music recommender systems which heavily rely on accurate classification of songs based on their genre, audio indexing and music/song composers.

In [1]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
import seaborn as sns
from sklearn.model_selection import GridSearchCV
```

DATA OVERVIEW

The Spotify dataset used in this project is a mix of factors that are used in predicting the genre of the song. Each song in the dataset possesses a set of attributes like the tempo (bpm), loudness (dB), danceability(dnce), song's title(title), year, energy(nrgy), live, value (val), speech(spch), pop, top genre, and duration(dur). These are used to uniquely identify a song. Out of the total attributes present in the dataset we have selected 7 factors that best suit our model and those which give optimal results.

In [2]:

```
# Load the datasets
train_df = pd.read_csv('CS98XClassificationTrain.csv')
test_df = pd.read_csv('CS98XClassificationTest.csv')
train_df.head()
```

Out[2]:

	Id	title	artist	year	bpm	nrgy	dnce	dB	live	val	dur	acous	spch	pop	top genre
0	1	My Happiness	Connie Francis	1996	107	31	45	-8	13	28	150	75	3	44	adult standards
1	2	Unchained Melody	The Teddy Bears	2011	114	44	53	-8	13	47	139	49	3	37	NaN
2	3	How Deep Is Your Love	Bee Gees	1979	105	36	63	-9	13	67	245	11	3	77	adult standards
			Barbra					-							adult

	Id	title	artist	year	bpm	nrgy	dnce	dB	live	val	dur	acous	spch	pop	top genre
3	4	Woman in Love	Barbra Streisand	1980	170	28	47	-16	13	33	232	25	3	67	adult standards
4	5	Goodbye Yellow Brick Road - Remastered 2014	Elton John	1973	121	47	56	-8	15	40	193	45	3	63	glam rock

Pre-processing

The data underwent pre-processing and cleaning where we converted the text labels into unique numeric values. The artist's name is also assigned a numeric value which is mapped to a particular artist. This mapping is then applied to both the training and the test datasets. If the model bumps into a new artist it will assign a default value to this new artist in order to avoid processing errors.

The below snippet also selects a set of factors/ attributes which represent accurate features to our machine learning model. There are seven ('dnce','dB','dur','acous','nrgy','pop','artist_encoded') features which represent the song's characteristics.

In [3]:

```
# Preprocess the training dataset
genre_encoder = LabelEncoder()
train_df['top genre encoded'] = genre_encoder.fit_transform(train_df['top genre'])

# Creating a mapping for artist encoding that includes a default for unseen artists
unique_artists = train_df['artist'].unique()
artist_mapping = {artist: i for i, artist in enumerate(unique_artists)}
default_artist_value = len(unique_artists)  # Assign a default value for unseen artists

# Apply the mapping to train and test datasets
train_df['artist_encoded'] = train_df['artist'].map(artist_mapping).astype(int)
test_df['artist_encoded'] = test_df['artist'].map(artist_mapping).fillna(default_artist_
value).astype(int)

# Prepare features and target for training
features = ['dnce', 'dB', 'dur', 'acous', 'nrgy', 'pop', 'artist_encoded']
X_train = train_df[features]
y_train = train_df['top genre encoded']

train_df.head()
```

Out[3]:

	Id	title	artist	year	bpm	nrgy	dnce	dB	live	val	dur	acous	spch	pop	top genre	top genre encoded	artist_encoc
0	1	My Happiness	Connie Francis	1996	107	31	45	-8	13	28	150	75	3	44	adult standards	1	
1	2	Unchained Melody	The Teddy Bears	2011	114	44	53	-8	13	47	139	49	3	37	NaN	86	
2	3	How Deep Is Your Love	Bee Gees	1979	105	36	63	-9	13	67	245	11	3	77	adult standards	1	
3	4	Woman in Love	Barbra Streisand	1980	170	28	47	-16	13	33	232	25	3	67	adult standards	1	
4	5	Goodbye Yellow Brick Road - Remastered 2014	Elton John	1973	121	47	56	-8	15	40	193	45	3	63	glam rock	68	

HEATMAP PLOTTING

A heatmap is plotted using specific columns of the dataset by calculating the correlation between the features

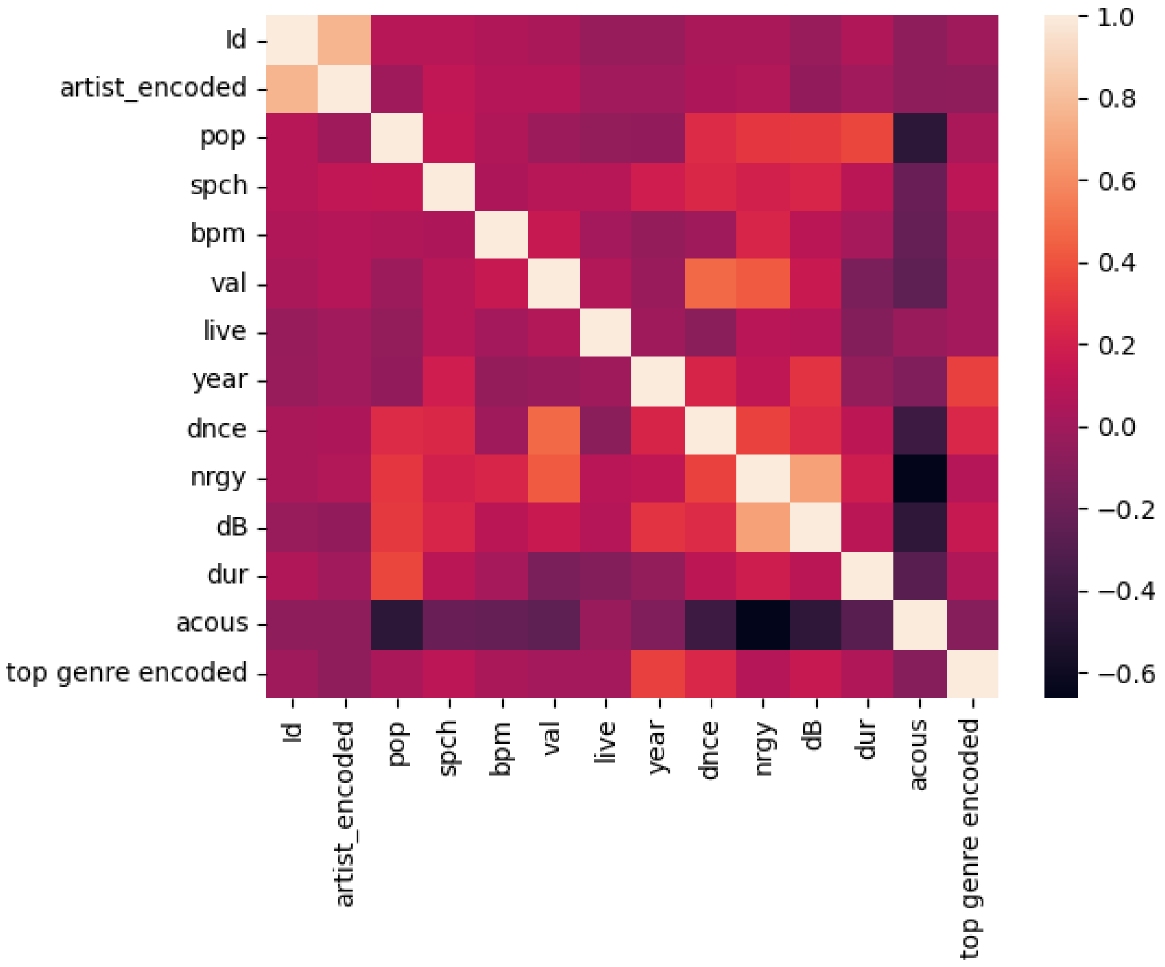
A heatmap is plotted using specific columns of the dataset by calculating the correlation between the features including factors like tempo, energy, danceability, etc against artist's information and genre. Correlation measures how strongly pairs are related with values -1 indicates perfect negative correlation, +1 indicates perfect positive correlation and 0 indicates no linear relationship. In the heatmap, colour intensity indicates the strenght of correlation.

In [4]:

```
df= train_df[['Id','artist_encoded','pop','spch','bpm','val','live','year','dnce','nrgy'
,'dB','dur','acous','top genre encoded']]
corr = df.corr()
sns.heatmap(corr)
```

Out[4]:

```
<Axes: >
```

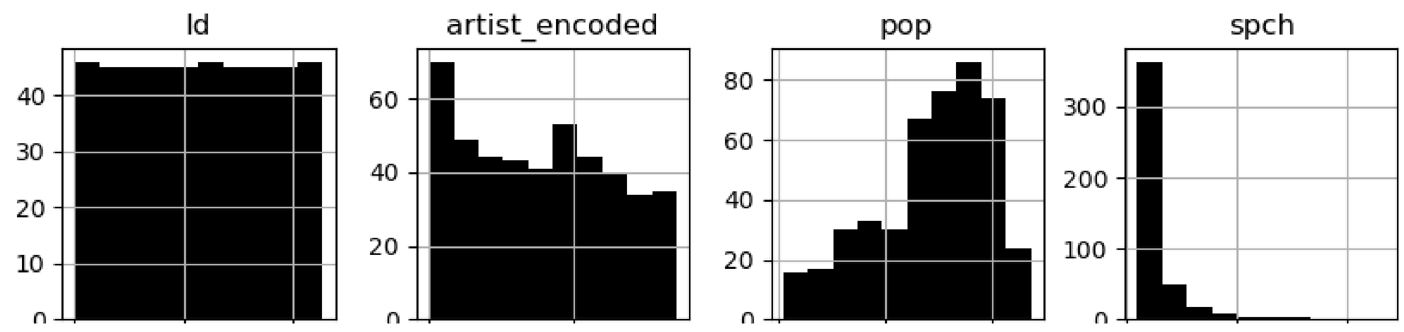


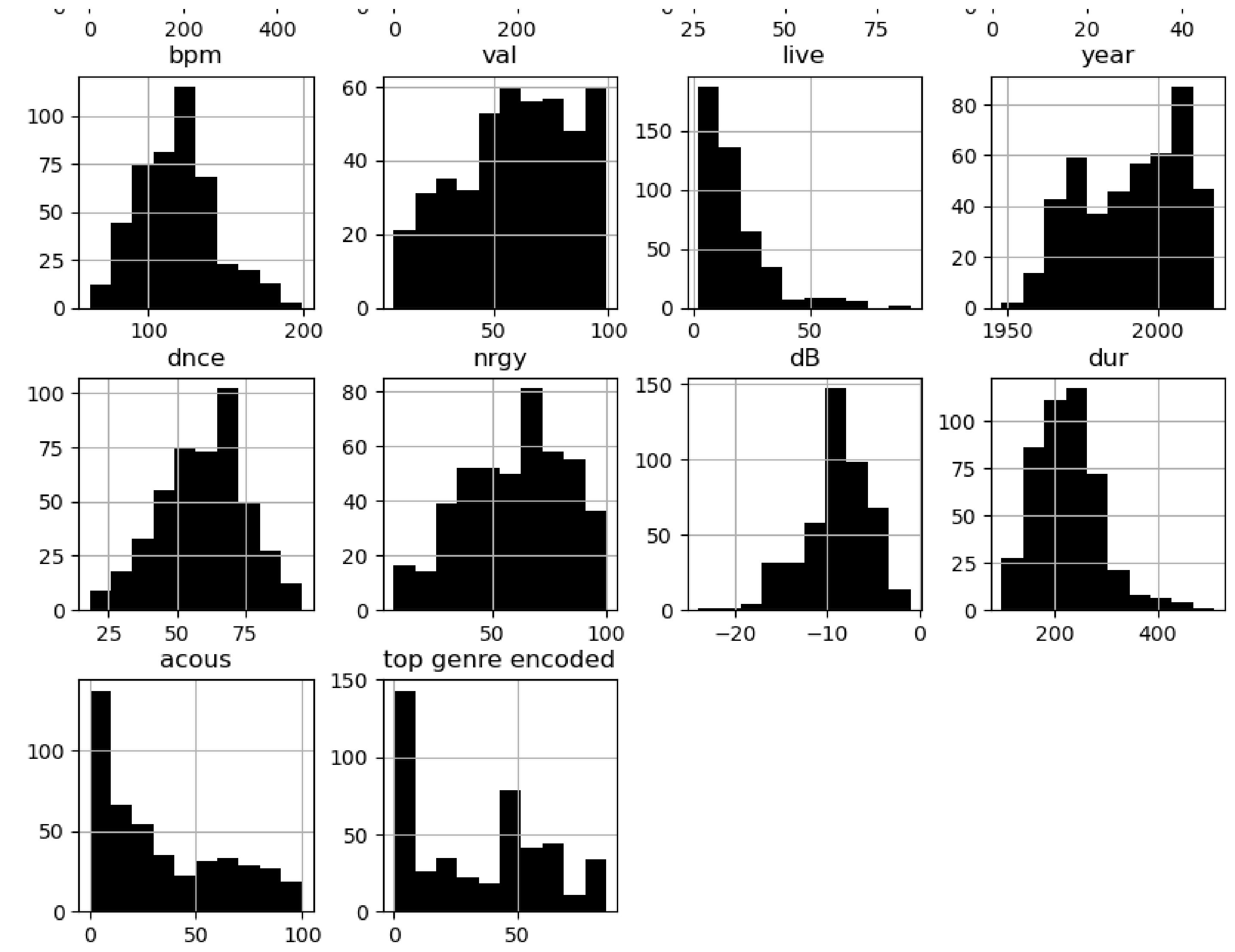
HISTOGRAM PLOTTING

Histograms are important for understanding the features in the Spotify dataset. Each graph plots the frequency of songs across different values of a atribute, helping us understand how it differs across the music collection.

In [5]:

```
numerical_columns = list(df.dtypes[df.dtypes != 'object'].index.values)
df.loc[:,numerical_columns].hist(figsize=(10,10),color='black');
```





SPLITTING DATA

80% of the data is used in training while the remaining 20% is used for the validation set. This step ensures that the model is evaluated fairly.

In [6]:

```
# Split the dataset into training and validation sets
X_train_split, X_val, y_train_split, y_val = train_test_split(X_train, y_train, test_siz
e=0.2, random_state=42)

# Scaling features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_split)
X_val_scaled = scaler.transform(X_val)
```

RANDOM FOREST CLASSIFIER

The below code performs optimization process for the selected algorithm (Random Forest) using GridSearchCV which helped in selecting the best hyperparameter. By implementing the 5 fold cross validation, the performance of the model is evaluated for each combination.

In [7]:

```
# Define the parameter grid for RandomForestClassifier
param_grid_rf = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Setup GridSearchCV
rf_model = RandomForestClassifier(random_state=42)
grid_search_rf = GridSearchCV(rf_model, param_grid_rf, cv=5, scoring='accuracy', n_jobs=
-1)

# Fit GridSearchCV
grid_search_rf.fit(X_train_scaled, y_train_split)

# Best parameters and score
print("Random Forest Best Parameters:", grid_search_rf.best_params_)
print("Random Forest Best Score:", grid_search_rf.best_score_)
```

```
C:\Users\HOME\anaconda3\Lib\site-packages\sklearn\model_selection\_split.py:725: UserWarn
ing: The least populated class in y has only 1 members, which is less than n_splits=5.
  warnings.warn(
```

```
Random Forest Best Parameters: {'max_depth': None, 'min_samples_leaf': 4, 'min_samples_sp
lit': 2, 'n_estimators': 100}
Random Forest Best Score: 0.31499238964992393
```

TRAINING THE RF CLASSIFIER

With the song's genre as the target and scaled features as input, this code sample trains the Rnadom Forest classifier on a dataset of songs. The fixed random state is set to a fixed number so that it reproduces the same results everytime we run the code. After training, the classifier is now ready to predict the genre of new songs based on what it has learned before.

In [8]:

```
# Training the Random Forest classifier
rf_classifier = RandomForestClassifier(random_state=42)
rf_classifier.fit(X_train_scaled, y_train_split)
```

Out[8]:

```
▾         RandomForestClassifier
RandomForestClassifier(random_state=42)
```

In [9]:

```
# Validate the model
y_val_pred = rf_classifier.predict(X_val_scaled)
val_accuracy = accuracy_score(y_val, y_val_pred)
print("Validation Accuracy:", val_accuracy)
```

```
Validation Accuracy: 0.38461538461538464
```

```
Validation Accuracy: 0.38461538461538464
```

In [10]:

```
# Validate Random Forest model
y_val_pred_rf = rf_classifier.predict(X_val_scaled)
val_accuracy_rf = accuracy_score(y_val, y_val_pred_rf)
print("Random Forest Validation Accuracy:", val_accuracy_rf)
```

```
Random Forest Validation Accuracy: 0.38461538461538464
```

```
Random Forest Validation Accuracy: 0.38461538461538464
```

This part of the code tests the Random Forest model to check how accurate it is at classifying the song genres depending upon what it learned during training, using the validation set (new songs set) for the test.

In [11]:

```
# Prepare the test dataset
X_test = test_df[features]
X_test_scaled = scaler.transform(X_test)
```

PREDICTING THE "TOP GENRE"

The new song set is then used by the model to predict the genre of the song, converts the numeric values back to the genre labels and displays that as the output.

In [12]:

```
# Predict the 'top genre' for the test dataset
test_predictions_encoded = rf_classifier.predict(X_test_scaled)
test_predictions = genre_encoder.inverse_transform(test_predictions_encoded)

# Save the predictions to a CSV file
output_df = pd.DataFrame({'Id': test_df['Id'], 'Top Genre': test_predictions})
output_csv_path = 'predicted_top_genres_with_artist_rf.csv'
output_df.to_csv(output_csv_path, index=False)

print("Output saved to:", output_csv_path)
```

```
Output saved to: predicted_top_genres_with_artist_rf.csv
```

SPACE VECTOR MODEL

The code used for SVM describes how to maximize, train, evaluate the Space Vector Machine (SVM) classifier to predict the genre from the given dataset.

The model works by using GridSearchCV with 5-fold-cross-validation to identify the best combination for optimal results. Scaled training data is used to train the SVM model after determining the ideal parameters. After that, the model's performance is evaluated using the test and validation dataset to make sure the model predicts the song's genre successfully.

Once the model is evaluated, the top genre is predicted by applying the trained SVM classifier to the scaled test data. The numerical values are decoded back to the genre labels.

In [13]:

```
# Define the parameter grid for SVM
param_grid_svc = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf', 'poly'],
    'gamma': ['scale', 'auto']
}

# Setup GridSearchCV
svm_model = SVC(random_state=42)
grid_search_svc = GridSearchCV(svm_model, param_grid_svc, cv=5, scoring='accuracy', n_jo
bs=-1)

# Fit GridSearchCV
grid_search_svc.fit(X_train_scaled, y_train_split)

# Best parameters and score
print("SVC Best Parameters:", grid_search_svc.best_params_)
print("SVC Best Score:", grid_search_svc.best_score_)
```

```
C:\Users\HOME\anaconda3\Lib\site-packages\sklearn\model_selection\_split.py:725: UserWarn
ing: The least populated class in y has only 1 members, which is less than n_splits=5.
  warnings.warn(
```

```
SVC Best Parameters: {'C': 0.1, 'gamma': 'scale', 'kernel': 'linear'}
SVC Best Score: 0.2873287671232877
```

In [14]:

```
# Training the SVM classifier

svm_classifier = SVC(random_state=42)
svm_classifier.fit(X_train_scaled, y_train_split)  # Ensure to use y_train_split if you'r
e training with a split dataset
```

Out[14]:

```
▾         SVC
SVC(random_state=42)
```

In [15]:

```
# Validate the SVM model
y_val_pred_svm = svm_classifier.predict(X_val_scaled)
val_accuracy_svm = accuracy_score(y_val, y_val_pred_svm)
print("SVM Validation Accuracy:", val_accuracy_svm)
```

```
SVM Validation Accuracy: 0.3626373626373626
```

```
SVM Validation Accuracy: 0.3626373626373626
```

In [16]:

```
# Validate SVM training set
y_train_pred_svm = svm_classifier.predict(X_train_scaled)
train_accuracy_svm = accuracy_score(y_train_split, y_train_pred_svm)
val_accuracy_svm = accuracy_score(y_val, y_val_pred_svm)
print(f"SVM Training Accuracy: {train_accuracy_svm}")
```

```
SVM Training Accuracy: 0.3674033149171271
```

```
SVM Training Accuracy: 0.3674033149171271
```

In [17]:

```
# Prepare the test dataset
X_test = test_df[features]
X_test_scaled = scaler.transform(X_test)
```

In [18]:

```
# Predict 'top genre' for the test dataset using SVM
test_predictions_encoded_svm = svm_classifier.predict(X_test_scaled)
test_predictions_svm = genre_encoder.inverse_transform(test_predictions_encoded_svm)

# Save SVM predictions to a CSV file
output_df_svm = pd.DataFrame({'Id': test_df['Id'], 'Top Genre': test_predictions_svm})
output_csv_path_svm = 'predicted_top_genres_full_svm.csv'
output_df_svm.to_csv(output_csv_path_svm, index=False)
print("SVM Output saved to:", output_csv_path_svm)
y_val_pred_svm = svm_classifier.predict(X_val_scaled)
train_accuracy_svm = accuracy_score(y_train_split, y_train_pred_svm)
```

```
SVM Output saved to: predicted_top_genres_full_svm.csv
```

K Nearest Neighbor

The process is initiated by specifying a set of parameters(n_neighbors, weights etc) which is processed by GridSearchCV. The 5-fold-cross-validation is then employed to check the accuracy achieved by combining these parameters with the training data used.

Once the right parameters are identfiied, the entire training dataset is trained with the new KNN classifier. This trained model is then gauged on both train and test dataset calculating the precision of the model to predict genres. Following this, the model is applied to a set of new songs. The predictions are then converted back to their original labels are are compiled along with their corresponding IDs.

In [19]:

```
# Define the parameter grid for KNeighborsClassifier
param_grid_knn = {
    'n_neighbors': [3, 5, 7, 9],
    'weights': ['uniform', 'distance'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute']
}

# Setup GridSearchCV
knn_model = KNeighborsClassifier()
grid_search_knn = GridSearchCV(knn_model, param_grid_knn, cv=5, scoring='accuracy', n_jo
bs=-1)

# Fit GridSearchCV
grid_search_knn.fit(X_train_scaled, y_train_split)

# Best parameters and score
print("KNN Best Parameters:", grid_search_knn.best_params_)
print("KNN Best Score:", grid_search_knn.best_score_)
```

```
C:\Users\HOME\anaconda3\Lib\site-packages\sklearn\model_selection\_split.py:725: UserWarn
ing: The least populated class in y has only 1 members, which is less than n_splits=5.
  warnings.warn(
```

```
KNN Best Parameters: {'algorithm': 'auto', 'n_neighbors': 9, 'weights': 'uniform'}
KNN Best Score: 0.25962709284627095
```

In [20]:

```
# Train KNN classifier
knn_classifier = KNeighborsClassifier()
knn_classifier.fit(X_train_scaled, y_train_split)
```

Out[20]:

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

In [21]:

```
# Validate KNN model
y_val_pred_knn = knn_classifier.predict(X_val_scaled)
val_accuracy_knn = accuracy_score(y_val, y_val_pred_knn)
print("KNN Validation Accuracy:", val_accuracy_knn)
```

```
KNN Validation Accuracy: 0.3076923076923077
```

```
KNN Validation Accuracy: 0.3076923076923077
```

In [ ]:

```
# Validate KNN training set
y_train_pred_knn = knn_classifier.predict(X_train_scaled)
train_accuracy_knn = accuracy_score(y_train_split, y_train_pred_knn)
val_accuracy_knn = accuracy_score(y_val, y_val_pred_knn)
print(f"KNN Training Accuracy: {train_accuracy_knn}")
```

```
Running cells with 'c:\Program Files\Python36\python.exe' requires the ipykernel package.

Run the following command to install 'ipykernel' into the Python environment.

Command: '"c:/Program Files/Python36/python.exe" -m pip install ipykernel -U --user --for
```

```
ce-reinstall'
```

In [23]:

```
# Prepare the test dataset
X_test = test_df[features]
X_test_scaled = scaler.transform(X_test)
```

In [24]:

```
# Predict 'top genre' for the test dataset using KNN
test_predictions_encoded_knn = knn_classifier.predict(X_test_scaled)
test_predictions_knn = genre_encoder.inverse_transform(test_predictions_encoded_knn)

# Save KNN predictions to a CSV file
output_df_knn = pd.DataFrame({'Id': test_df['Id'], 'Top Genre': test_predictions_knn})
output_csv_path_knn = 'predicted_top_genres_full_knn.csv'
output_df_knn.to_csv(output_csv_path_knn, index=False)
print("KNN Output saved to:", output_csv_path_knn)
```

```
KNN Output saved to: predicted_top_genres_full_knn.csv
```

CONCLUSION

In our quest to find a model which predicts the song genres accurately, the results attained using Random Forest were the most promising. Random Forest holds many advantages over Space Vector Machine (SVM) and K-neartest neighbors (KNN) such as it's ability to handle complex data and robustness to overfitting and imbalance. This is shown by the scores achieved by the model, supporting its selection as the most effective model for genre prediction in this project.