CLASSIFICATION OF MUSIC GENRE USING ACOUSTIC FEATURES AND LYRICS

BY

UNWANA EKENG ITA

CHISOM CHIBUIKE

AKINPELU IFEOLUWA ENOCH

A DOCUMENTATION FOR A MACHINE LEARNING PROJECT

SUBMITTED

TO

ACME SOFTWARE LAB

JULY 2024

Table of Contents

1. ACKNOWLEDGEMENT…………………………………………………………………...2

1.1 About the Organization……………………………………………………………2

1.2 Project Overview……………………………………………………………. 3

1.3 Objectives……………………………………………………………………….3

1.4 Introduction…………………………………………………………………...4

2. METHODOLOGY………………………………………………………………......5

2.1 Data Collection and Source……………………………………………………...5

2.2 Data Description………………………………………………………………….5

2.3 Data Loading……………………………………………………………………….9

2.4 Data Preprocessing…………………………………………………………………9

2.5 Exploratory Data Analysis…………………………………………………………11

2.6 Feature Engineering……………………………………………………………….16

3. IMPLEMENTATION ……………………………………………………………….17

3.1 Model Building…………………………………………………………………17

3.2 Model Training…………………………………………………………………19

4. RESULTS …………………………………………………………………………21

4.1 Evaluation of Metrics……………………………………………………………..21

4.2 Discussion of Findings……………………………………………………………22

5. CONCLUSION……………………………………………………………………23

6. **REFERENCES**……………………………………………………………………24

1. ACKNOWLEDGMENT

We would like to extend our deepest gratitude to ACME Software Lab for granting us this invaluable opportunity to be a part of the company as interns. This experience has significantly enhanced our knowledge and skills in the field of data science and machine learning. We are especially grateful to our data science leads, **Emurotu Paul Oghenekevwe** and **Hannah Igboke**, whose guidance, expertise, and unwavering support have been instrumental in the successful completion of our project. Additionally, we appreciate the collaboration and dedication of our team members: **Chisom Chibuike**, **Akinpelu Ifeoluwa Enoch**, and myself, **Unwana Ekeng Ita**. This internship has been a remarkable journey of learning and growth, and we are thankful for the opportunity to contribute to and learn from such a prestigious organization.

1.1 **About the Company**

ACME Software Lab offers a comprehensive suite of IT services, including cybersecurity, data analysis, web development, SEO optimization, and app development. They provide customized solutions to enhance digital presence and business success. Additionally, ACME offers educational courses in data science, web development, and UI/UX design. The company aims to empower businesses and individuals through technology, ensuring robust security and actionable insights for informed decision-making. For more information, visit ACME Software Lab**.**

1.2 Project Overview

Project Title: Genre classification using Acoustic Features and Lyrics

Authors: Unwana Ita, Chisom Chibuike, and Akinpelu, Enoch

Date: June – July 2024

Tools and Libraries: Python, pandas, numpy, matplotlib, seaborn, detect, sklearn, and keras

1.3 Objectives

This project has the following objective:

1. Automatically classify songs into genres based on their acoustic features and lyrics.

1.4 Introduction

Classification of music genres is crucial for Music Information Retrieval (MIR). Digital music libraries have experienced exponential growth in recent years, as a result of this, there is an increased need for automatic methods to organize and categorize music tracks. Accurate genre classification can enhance user experience by enabling efficient music recommendation, playlist generation, and music discovery.[1][2]

This project aims to classify songs into different genres based on their acoustic features and lyrics. By leveraging machine learning and deep learning techniques, the project seeks to develop a robust model that can accurately predict the genre of a given song. Integrating acoustic and textual data is expected to improve classification performance by capturing the nuances of music that are often genre-specific.

2. METHODOLOGY

This project employed a computational approach to automatically classify song genres based on the song's acoustic features and lyrics. Here's a breakdown of the key methods used

2.1 Data Collection and Source

The data was collected from the MusicOSet platform. MusicOSet is an enhanced an open and enhanced dataset of musical elements (music, albums, and artists) suitable for music data mining. The attractive features of MusicOSet include the enrichment of existing metadata to which it is linked and the popularity classification of the musical elements present in the dataset.

The dataset was downloaded from the following link: https://rb.gy/c7bsfb

2.2 Data description

Two datasets were used for this project, one contains information about the song acoustic features, while the other contains information about the song lyrics. The following is description of all the features and target variable:

Lyrics Data	
Song_id	The Spotify ID for the song
Lyrics	The music lyrics
Main_genre	The genre of the songs (Target Variable)

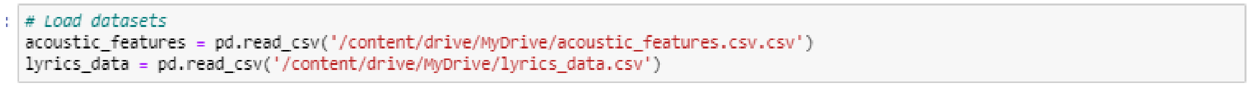
Acoustic Features	
song_id	The Spotify ID for the song.
duration_ms	The duration of the track in milliseconds.
key	The estimated overall key of the track. Integers map to pitches using standard Pitch Class notation. If no key was detected, the value is -1.
mode	Mode indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived. Major is represented by 1 and minor is 0.
time_signature	An estimated overall time signature of a track. The time signature (meter) is a notational convention to specify how many beats are in each bar (or measure).
acousticness	A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic.
danceability	Danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is the least danceable and 1.0 is the most danceable.
energy	Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale. Perceptual features

	contributing to this attribute include dynamic range, perceived loudness, timbre, onset rate, and general entropy.
instrumentalness	Predicts whether a track contains no vocals. “Ooh” and “aah” sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly “vocal”. The closer the instrumentalness value is to 1.0, the greater the likelihood the track contains no vocal content. Values above 0.5 are intended to represent instrumental tracks, but confidence is higher as the value approaches 1.0.
liveness	Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live. A value above 0.8 provides strong likelihood that the track is live.
loudness	The overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing the relative loudness of tracks. Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude). Values typically range between -60 and 0 db.
speechiness	Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audiobook, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values between 0.33 and 0.66 describe tracks

	that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.33 most likely represent music and other non-speech-like tracks.
valence	A measure from 0.0 to 1.0 describes the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).
tempo	The overall estimated tempo of a track is in beats per minute (BPM). In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration.

2.3 Data Loading

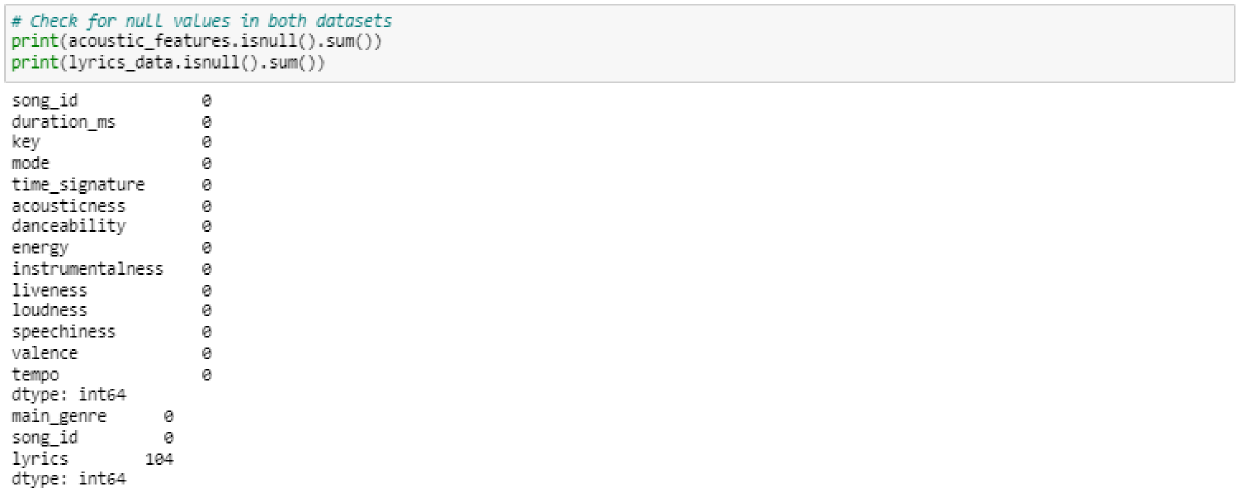
The datasets are loaded using Python’s pandas library, which allows for efficient manipulation and analysis of the data



```
# Load datasets
acoustic_features = pd.read_csv('/content/drive/MyDrive/acoustic_features.csv.csv')
lyrics_data = pd.read_csv('/content/drive/MyDrive/lyrics_data.csv')
```

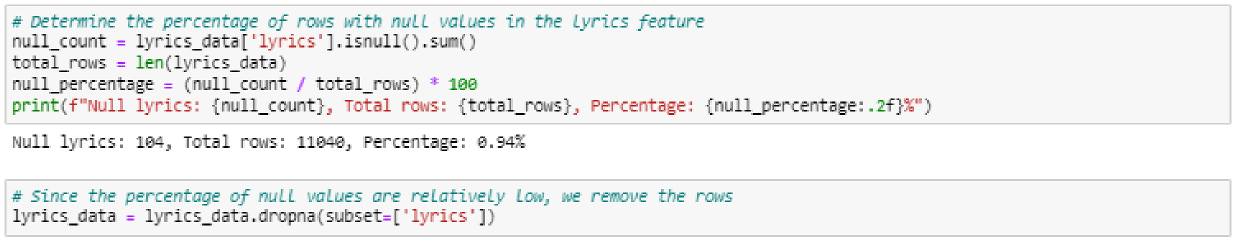
2.4 Data Preprocessing

The first step in preprocessing is to clean the data. This involves handling missing data, removing duplicates, and ensuring the data is in a suitable format for further processing. The following codes snippets shows how some of this cleaning steps was achieved.



```
# Check for null values in both datasets
print(acoustic_features.isnull().sum())
print(lyrics_data.isnull().sum())
```

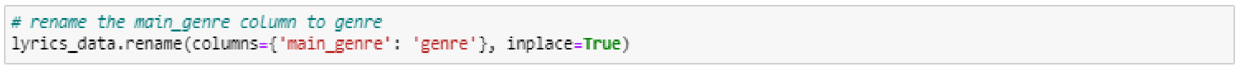
```
song_id             0
duration_ms         0
key                 0
mode                0
time_signature      0
acousticness        0
danceability        0
energy              0
instrumentalness    0
liveness            0
loudness            0
speechiness         0
valence             0
tempo               0
dtype: int64
main_genre      0
song_id         0
lyrics        104
dtype: int64
```



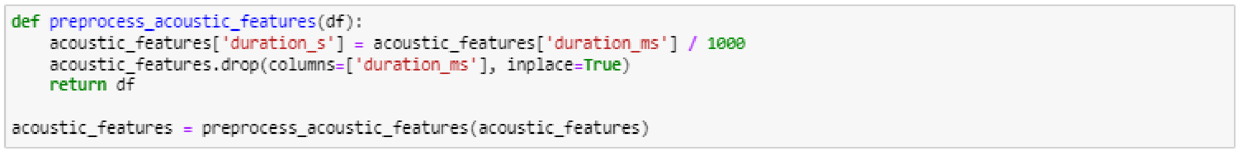
```
# Determine the percentage of rows with null values in the lyrics feature
null_count = lyrics_data['lyrics'].isnull().sum()
total_rows = len(lyrics_data)
null_percentage = (null_count / total_rows) * 100
print(f"Null lyrics: {null_count}, Total rows: {total_rows}, Percentage: {null_percentage:.2f}%")
```

```
Null lyrics: 104, Total rows: 11040, Percentage: 0.94%
```

```
# Since the percentage of null values are relatively low, we remove the rows
lyrics_data = lyrics_data.dropna(subset=['lyrics'])
```



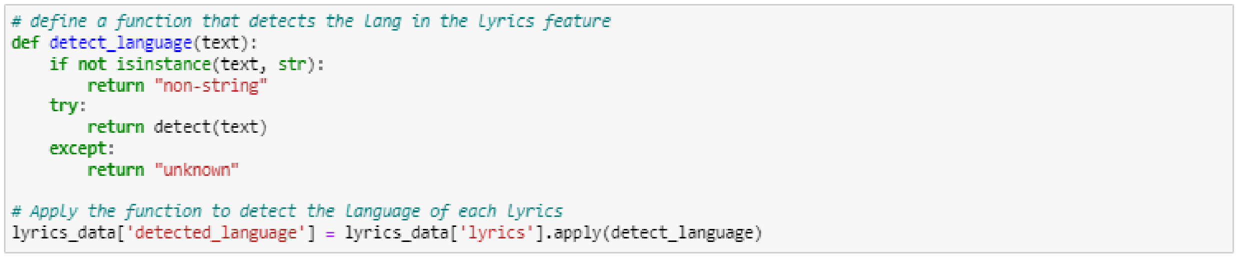
```
# rename the main_genre column to genre
lyrics_data.rename(columns={'main_genre': 'genre'}, inplace=True)
```



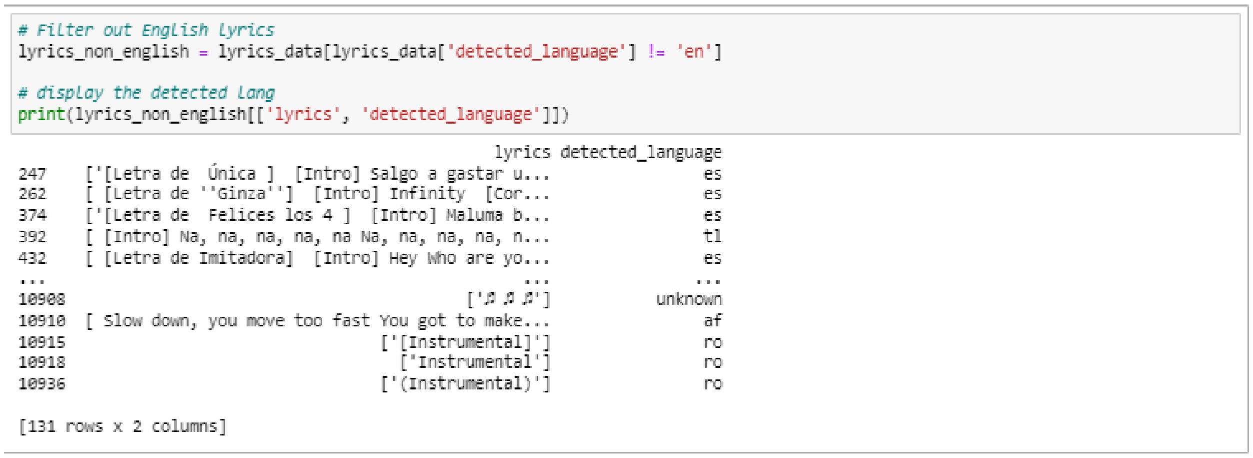
```python
def preprocess_acoustic_features(df):
    acoustic_features['duration_s'] = acoustic_features['duration_ms'] / 1000
    acoustic_features.drop(columns=['duration_ms'], inplace=True)
    return df

acoustic_features = preprocess_acoustic_features(acoustic_features)
```

The above steps were used to clean the data since it was observed that the data has no duplicate, outliers or any inconsistencies.

The next preprocessing step involved the text feature in our dataset. We started with the lyrics feature. First, we ensured that all the lyrics for the songs were in English language. The 'langdetect' library was used.

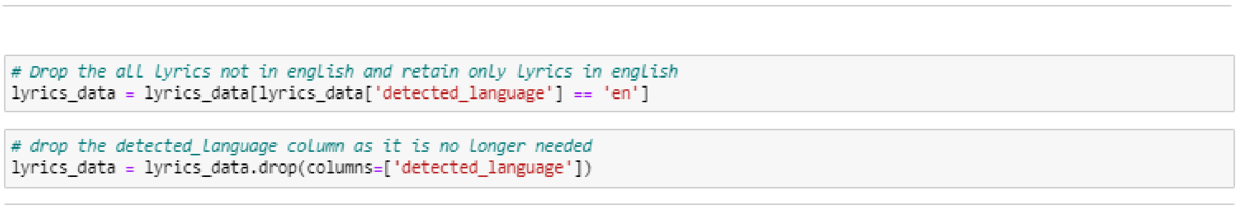


```python
# define a function that detects the lang in the lyrics feature
def detect_language(text):
    if not isinstance(text, str):
        return "non-string"
    try:
        return detect(text)
    except:
        return "unknown"

# Apply the function to detect the language of each lyrics
lyrics_data['detected_language'] = lyrics_data['lyrics'].apply(detect_language)
```



```python
# Filter out English lyrics
lyrics_non_english = lyrics_data[lyrics_data['detected_language'] != 'en']

# display the detected lang
print(lyrics_non_english[['lyrics', 'detected_language']])
```

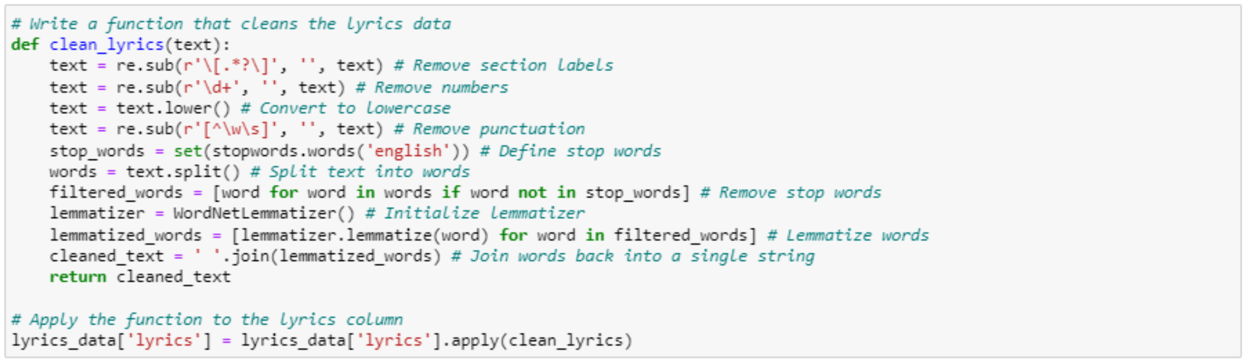
```
                                                  lyrics detected_language
247    ['[Letra de  Única ]  [Intro] Salgo a gastar u...                es
262    [ [Letra de ''Ginza'']  [Intro] Infinity  [Cor...                es
374    ['[Letra de  Felices los 4 ]  [Intro] Maluma b...                es
392    [ [Intro] Na, na, na, na, na Na, na, na, na, n...                tl
432    [ [Letra de Imitadora]  [Intro] Hey Who are yo...                es
...                                                  ...               ...
10908                                        ['♬ ♬ ♬']           unknown
10910  [ Slow down, you move too fast You got to make...                af
10915                               ['[Instrumental]']                ro
10918                                 ['Instrumental']                ro
10936                               ['(Instrumental)']                ro

[131 rows x 2 columns]
```



```python
# Drop the all lyrics not in english and retain only lyrics in english
lyrics_data = lyrics_data[lyrics_data['detected_language'] == 'en']
```

```python
# drop the detected_language column as it is no longer needed
lyrics_data = lyrics_data.drop(columns=['detected_language'])
```

Further steps were taken to preprocess the text data. A function was created to perform this task. The function performed several crucial preprocessing steps: it removed section labels, numbers, and punctuation, converted text to lowercase, and eliminated stop words. Additionally, it lemmatized the remaining words to ensure that different forms of a word are treated as the same token. This preprocessing is essential as it reduces noise and standardizes the text data, making it more suitable for the model. Clean and consistent text data enhances the model's ability to learn meaningful patterns, ultimately leading to better performance and more accurate genre classification. This function was then applied to the lyrics column of the dataset, ensuring all text data undergoes the same cleaning process.



```python
# Write a function that cleans the lyrics data
def clean_lyrics(text):
    text = re.sub(r'\[.*?\]', '', text) # Remove section labels
    text = re.sub(r'\d+', '', text) # Remove numbers
    text = text.lower() # Convert to lowercase
    text = re.sub(r'[^\w\s]', '', text) # Remove punctuation
    stop_words = set(stopwords.words('english')) # Define stop words
    words = text.split() # Split text into words
    filtered_words = [word for word in words if word not in stop_words] # Remove stop words
    lemmatizer = WordNetLemmatizer() # Initialize lemmatizer
    lemmatized_words = [lemmatizer.lemmatize(word) for word in filtered_words] # Lemmatize words
    cleaned_text = ' '.join(lemmatized_words) # Join words back into a single string
    return cleaned_text

# Apply the function to the lyrics column
lyrics_data['lyrics'] = lyrics_data['lyrics'].apply(clean_lyrics)
```

2.5 Exploratory Data Analysis

The next crucial step is the exploratory data analysis (EDA). EDA is like examining the ingredients before you cook. It helps us to understand what you are working on within your machine learning project. By analyzing data distributions, uncovering relationships, and spotting outliers, EDA ensures your model is built on a solid foundation, leading to more accurate and interpretable results.

The following code snippets shows all EDA steps that were carried out in this project:

```
# Plot distributions of numerical features
acoustic_features.hist(bins=30, figsize=(15, 10))
plt.tight_layout()
plt.show()
```

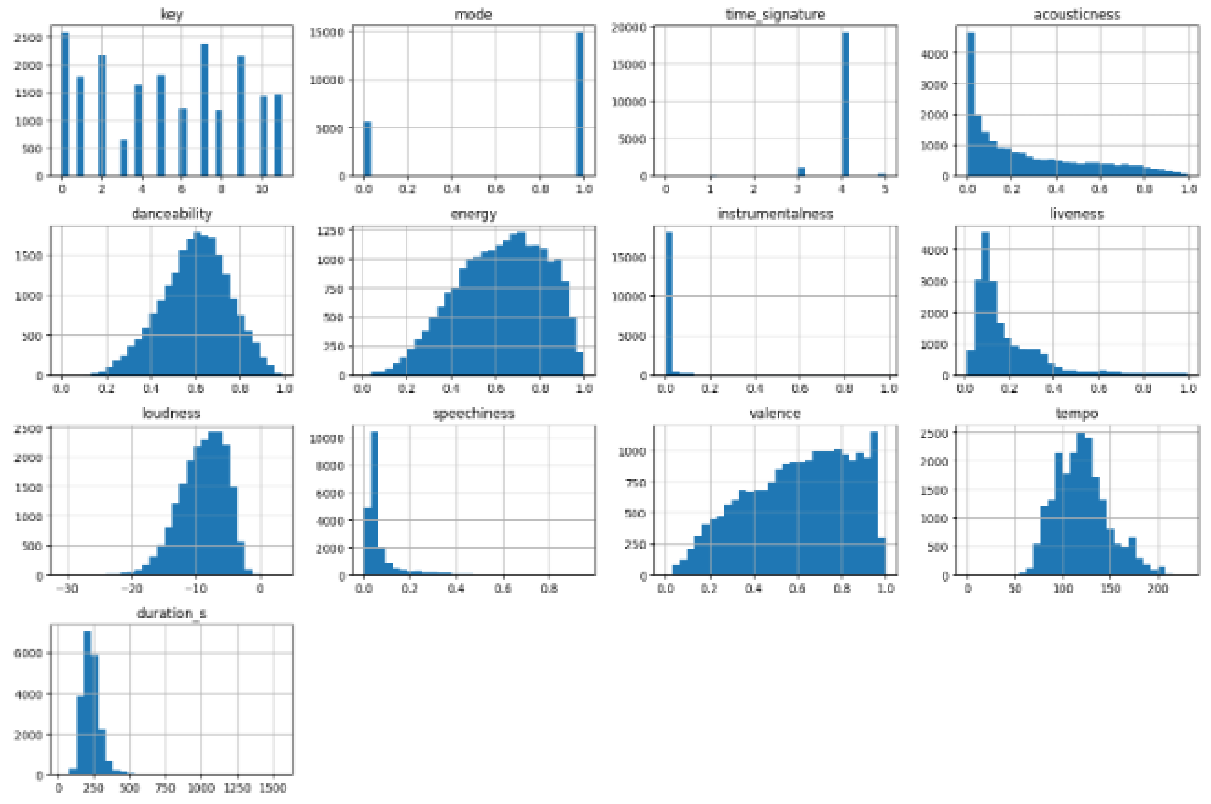


Figure I

```
# Display the distribution of genres
genre_distribution = lyrics_data['genre'].value_counts()

# Plot the distribution of genres
plt.figure(figsize=(35, 10))
sns.barplot(x=genre_distribution.index, y=genre_distribution.values)
plt.title('Distribution of Genres')
plt.xlabel('Genre')
plt.ylabel('Count')
plt.xticks(rotation=90)
plt.show()
```

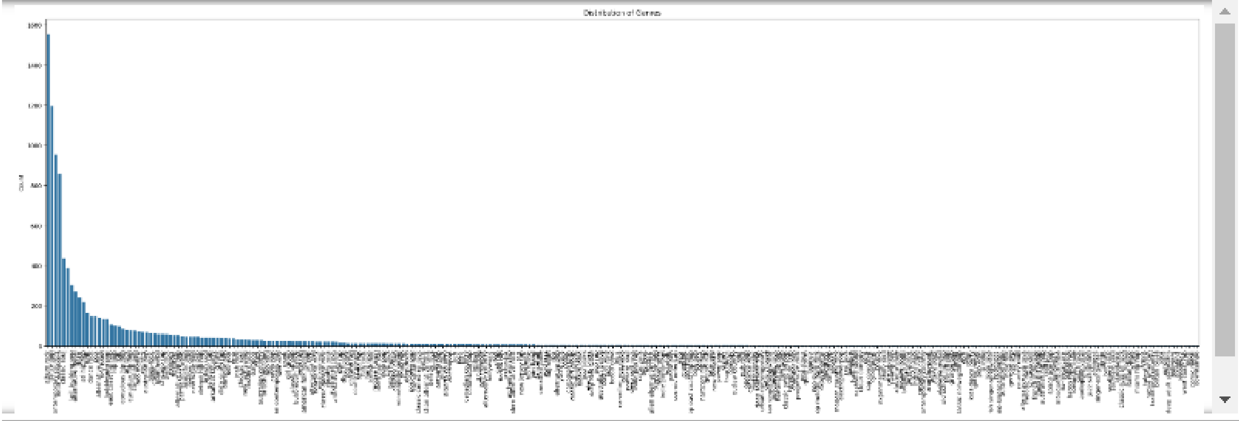


Figure II

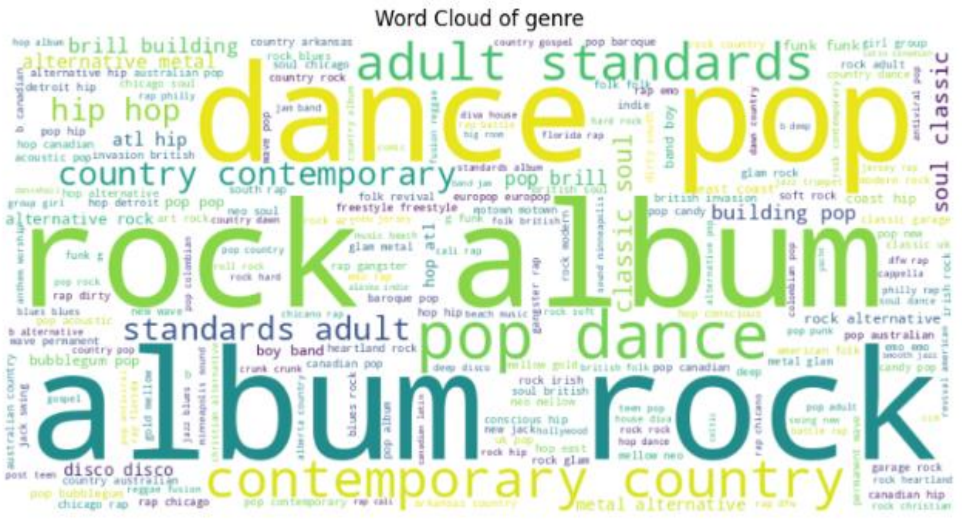
```
# Calculate Class Distribution
genre_counts = lyrics_data['genre'].value_counts()
print(genre_counts)
```

```
genre
album rock              1554
dance pop               1194
contemporary country     956
adult standards          856
classic soul             436
                        ...
broadway                   1
west coast rap             1
bboy                       1
comic metal                1
downtempo                  1
Name: count, Length: 292, dtype: int64
```

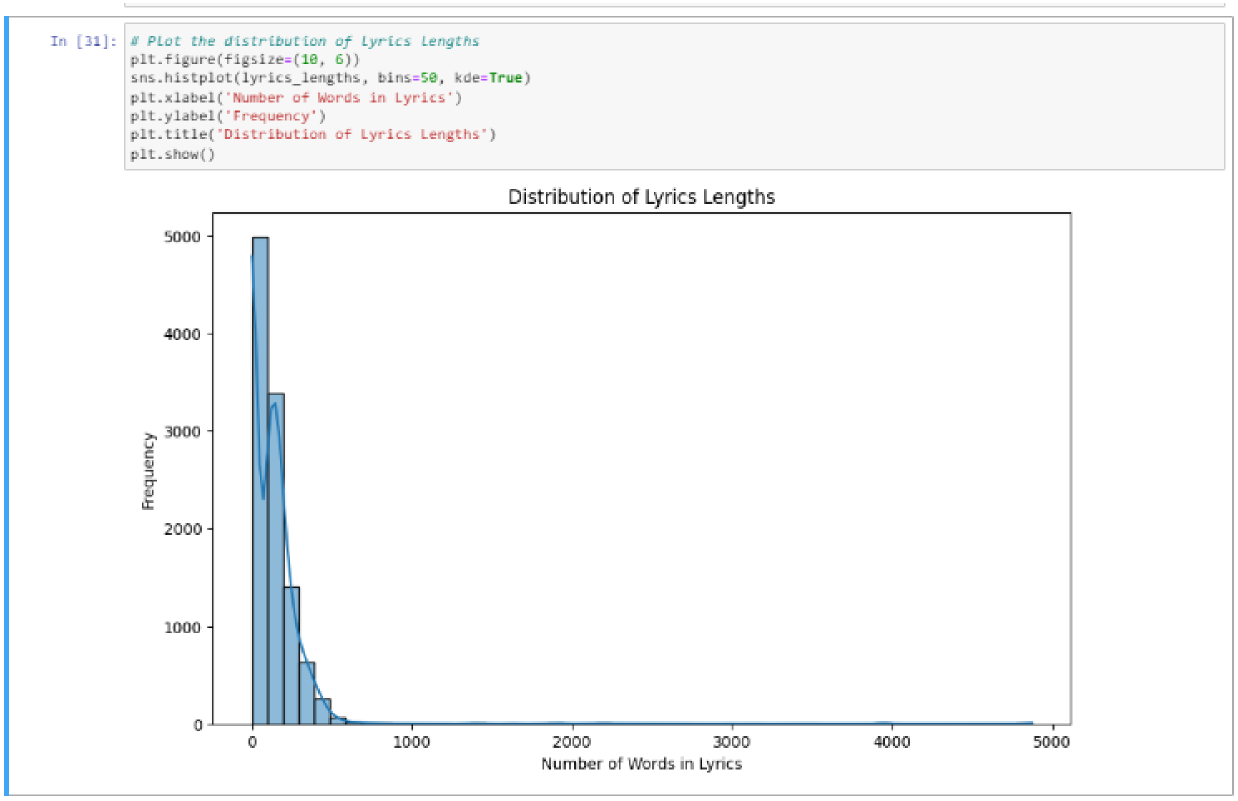
In [27]:
```
# Assuming genre_counts is your Series containing genre counts
total_songs = len(lyrics_data)

genre_percentages = (genre_counts / total_songs) * 100
genre_percentages_df = pd.DataFrame({'Count': genre_counts, 'Percentage (%)': genre_percentages})

print(genre_percentages_df.to_string())
```

```
                             Count  Percentage (%)
genre
album rock                    1554       14.382230
dance pop                     1194       11.050440
contemporary country           956        8.847756
adult standards                856        7.922258
classic soul                   436        4.035169
-                              391        3.618695
brill building pop             304        2.813512
alternative metal              271        2.508098
disco                          240        2.221194
atl hip hop                    219        2.026839
funk                           166        1.536326
dance rock                     148        1.369736
pop                            147        1.360481
alternative rock               142        1.314206
boy band                       134        1.240167
bubblegum pop                  134        1.240167
east coast hip hop             107        0.990282
```

```python
# Generate a word cloud for the most frequent genre
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(' '.join(lyrics_data['genre']))

# Display the word cloud
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud of genre')
plt.show()
```

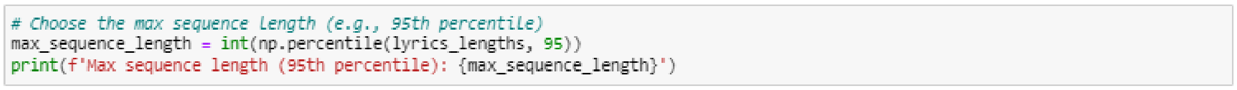


```python
# Calculate the length of each lyrics sequence
# Extract the lyrics column
lyrics = lyrics_data['lyrics']

# Split the lyrics into individual words
lyrics_lengths = lyrics.apply(lambda x: len(x.split()))
```



```
In [31]: # Plot the distribution of Lyrics Lengths
plt.figure(figsize=(10, 6))
sns.histplot(lyrics_lengths, bins=50, kde=True)
plt.xlabel('Number of Words in Lyrics')
plt.ylabel('Frequency')
plt.title('Distribution of Lyrics Lengths')
plt.show()
```

Figure III



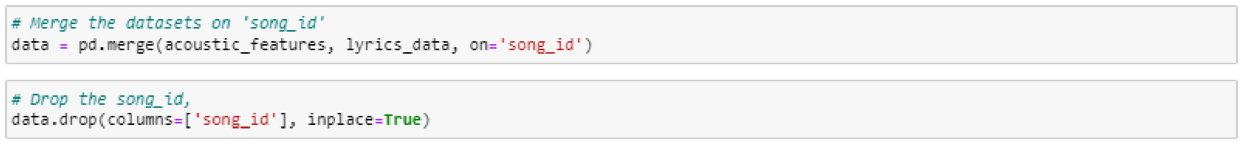
```
# Choose the max sequence length (e.g., 95th percentile)
max_sequence_length = int(np.percentile(lyrics_lengths, 95))
print(f'Max sequence length (95th percentile): {max_sequence_length}')
```

From the results of our exploratory data analysis, we had few observations. Some of the acoustics features are not normally distributed. There are 292 different genres from the dataset. The maximum lyrics length for the 95th percentile was observed to be 359. This means that 95% of the maximum lyrics' length is below 359. Also observed, was unequal class distribution indicating a significant difference in the number of songs belonging to each genre. 'Album rock' genre has 1554 songs while genres like 'west coast rap', 'comic metal', and 'downtempo' have 1 song each. A few genres represent a much larger portion of the data compared to others. Many genres have very few occurrences forming a long tail distribution as we noticed in Figure II above. While there is no strict definition for imbalance, a common rule of thumb is that a class representing 10-20% of the data can be considered imbalanced. In this case, many genres fall below this threshold. This imbalanced nature of the target variable is handled later in this project.

2.6 Feature Engineering

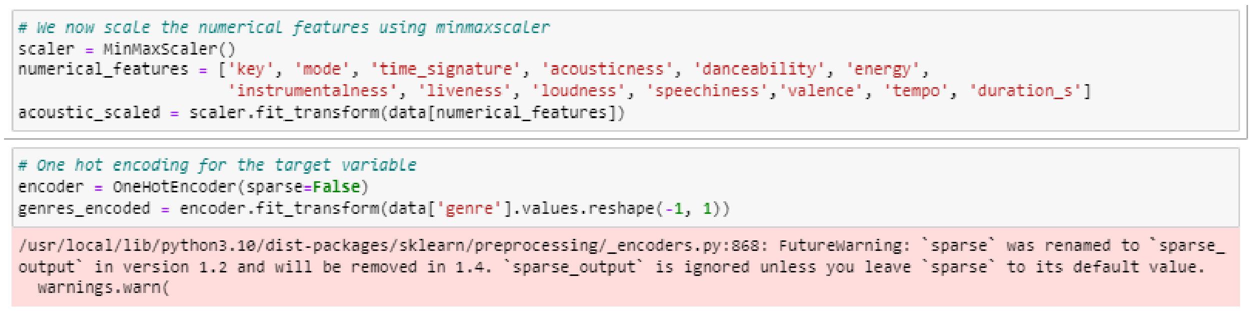
Feature engineering involves preparing the input data into features that the machine learning algorithm can understand. It also involves transforming variables into features that will help the machine learning algorithm achieve better performance and yield optimal results. Though they are used interchangeably, there is also a difference between feature selection and feature extraction. In the feature selection step, redundant or unused features are removed and as a result, a subset of the original features are obtained. Feature extraction on the other hand reduces the dimensions of the dataset by creating new features.[3]

The following code snippets show how some of these steps were achieved in this project:



```
# Merge the datasets on 'song_id'
data = pd.merge(acoustic_features, lyrics_data, on='song_id')
```

```
# Drop the song_id,
data.drop(columns=['song_id'], inplace=True)
```



```
# We now scale the numerical features using minmaxscaler
scaler = MinMaxScaler()
numerical_features = ['key', 'mode', 'time_signature', 'acousticness', 'danceability', 'energy',
                      'instrumentalness', 'liveness', 'loudness', 'speechiness','valence', 'tempo', 'duration_s']
acoustic_scaled = scaler.fit_transform(data[numerical_features])
```

```
# One hot encoding for the target variable
encoder = OneHotEncoder(sparse=False)
genres_encoded = encoder.fit_transform(data['genre'].values.reshape(-1, 1))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_encoders.py:868: FutureWarning: `sparse` was renamed to `sparse_
output` in version 1.2 and will be removed in 1.4. `sparse_output` is ignored unless you leave `sparse` to its default value.
  warnings.warn(
```

Feature scaling is the process of transforming the values of the numerical features to a common scale. Notice above that we normalized the numerical features using min-max-scaling. This rescales the data to a fixed range typically 0 and 1. Normalization is useful when you need the data to fit within a specific range. One hot encoder was also used to encode the genre feature. We used this technique because the genre data a nominal in nature and one hot shines in this type of data as it will not assume any order for the categories.

To encode the lyrics data (text data), we used word embeddings combined with sequence encoding techniques (like tokenization and padding) for encoding the lyrics feature. This approach captures the semantic meaning and relationships between words, which is crucial for understanding the context of the lyrics.

The following code snippets shows how this was achieved:

```
tokenizer = Tokenizer(num_words=10000)  # Limit to top 10,000 words
tokenizer.fit_on_texts(lyrics_data['lyrics'])
sequences = tokenizer.texts_to_sequences(lyrics_data['lyrics'])
max_sequence_length = 500  # Define max sequence length based on your EDA
lyrics_padded = pad_sequences(sequences, maxlen=max_sequence_length)
```

```
# Load pre-trained GloVe embeddings
glove_path = '/content/drive/MyDrive/glove.6B.100d.txt'
embedding_index = {}
with open(glove_path, encoding="utf8") as f:
    for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embedding_index[word] = coefs
```

```
# Create an embedding matrix
embedding_dim = 100
word_index = tokenizer.word_index
embedding_matrix = np.zeros((len(word_index) + 1, embedding_dim))
for word, i in word_index.items():
    embedding_vector = embedding_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

Next, we handle the imbalanced genre data as follows:

```
# Combine all features and handle imbalance using SMOTE
combined_features = np.hstack((acoustic_scaled, lyrics_padded))
```

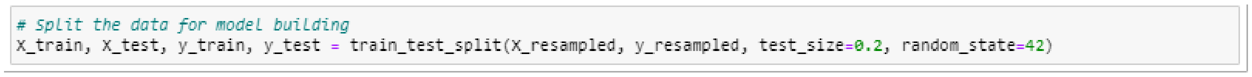
```
from imblearn.over_sampling import SMOTE
# Manually duplicate samples for small classes
min_samples_required = 6  # Set a minimum required samples, for example, 6

for idx, count in enumerate(genre_counts):
    if count < min_samples_required:
        diff = int(min_samples_required - count)
        class_samples = combined_features[genres_encoded[:, idx] == 1]
        class_labels = genres_encoded[genres_encoded[:, idx] == 1]
        combined_features = np.vstack((combined_features, np.tile(class_samples, (diff, 1))))
        genres_encoded = np.vstack((genres_encoded, np.tile(class_labels, (diff, 1))))

# Apply SMOTE with adjusted parameters
smote = SMOTE(random_state=42, k_neighbors=5)
X_resampled, y_resampled = smote.fit_resample(combined_features, genres_encoded)
```

3. IMPLEMENTATION

3.1 Model Building

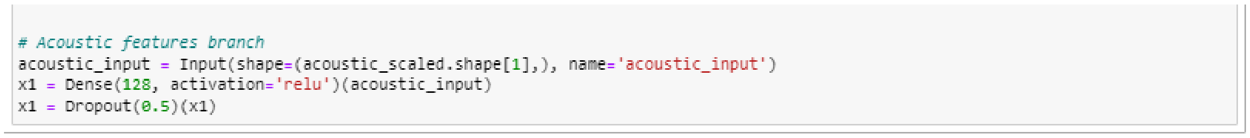
A machine learning model is essentially a computer program trained to make predictions or classifications based on the data. It is an automated system that learns from data to perform a specific task. Various machine learning and deep learning models can be used for genre classification. In this case, we are using deep learning model.

The following code snippet splits the dataset into training and testing sets, which is a crucial step in any machine-learning project. Using the train_test_split function from sklearn.model_selection, the data is divided such that 80% is used for training and 20% for testing, ensuring the model can be evaluated on unseen data. The random_state=42 ensures the reproducibility of results. This split is essential to prevent overfitting and to assess the model's generalization capability on new data.



```
# Split the data for model building
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)
```

Then the model's branch dedicated to acoustic features is defined. The Input layer specifies the shape of the acoustic features, followed by a Dense layer with 128 neurons and ReLU activation, which introduces non-linearity. A Dropout layer is added to reduce overfitting by randomly setting 50% of the neurons to zero during training. This branch processes the acoustic features independently before combining them with the lyrics branch.



```
# Acoustic features branch
acoustic_input = Input(shape=(acoustic_scaled.shape[1],), name='acoustic_input')
x1 = Dense(128, activation='relu')(acoustic_input)
x1 = Dropout(0.5)(x1)
```

The following code defines the lyrics processing branch of the model. The Input layer specifies the shape of the input sequences. The Embedding layer converts the input tokens into dense vectors of fixed size using a pre-trained embedding matrix, which is not trainable. This is followed by an LSTM (Long Short-Term Memory) layer with 128 units to capture temporal dependencies in the lyrics. A Dropout layer is added to prevent overfitting by randomly deactivating 50% of the neurons during training. This branch handles the text data independently before merging with the acoustic branch.

```python
# Lyrics branch
lyrics_input = Input(shape=(max_sequence_length,), name='lyrics_input')
embedding_layer = Embedding(input_dim=len(word_index) + 1,
                            output_dim=embedding_dim,
                            weights=[embedding_matrix],
                            input_length=max_sequence_length,
                            trainable=False)(lyrics_input)
x2 = LSTM(128, return_sequences=False)(embedding_layer)
x2 = Dropout(0.5)(x2)
```

The outputs of the acoustic and lyrics branches are concatenated to form a single tensor. The concatenated output is then passed through a Dense layer with a softmax activation function, which produces the final genre classification probabilities. The model is defined using the Model class, specifying the input layers for acoustic features and lyrics, and the output layer for genre classification. This architecture allows the model to simultaneously learn from both types of data, leveraging their combined predictive power.

```python
# Concatenate both branches
concatenated = Concatenate()([x1, x2])
output = Dense(genres_encoded.shape[1], activation='softmax')(concatenated)
```

```python
# Define the model
model = Model(inputs=[acoustic_input, lyrics_input], outputs=output)
```

We now compile the model, specifying the Adam optimizer with a learning rate of 0.001. The loss function used is categorical_crossentropy, which is appropriate for multi-class classification problems. The model's performance will be evaluated using the accuracy metric. Compiling the model configures it for training by linking the optimizer and loss function.

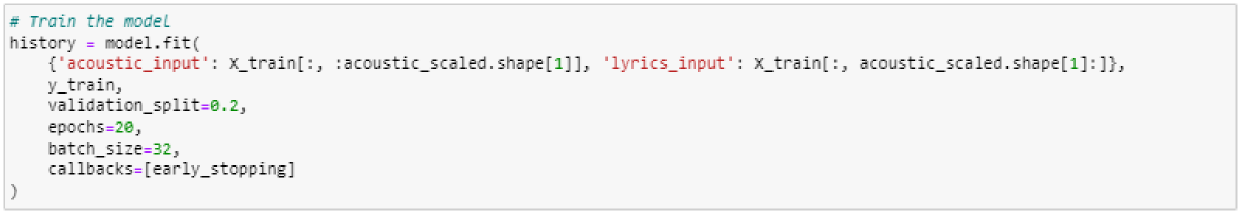
```python
# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
```

Before training the model, we set up early stopping, a regularization technique that helps prevent overfitting. The EarlyStopping callback monitors the validation loss and stops training if it does not improve for three consecutive epochs (patience=3). Additionally, it restores the best weights observed during training. This ensures the model does not overfit the training data and performs well on unseen data.

```python
# Define early stopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
```

3.2 Model Training

The training process involves fitting the model on the training data and validating their performance on the test data. The model was trained using the fit method. The training data is split into acoustic and lyrics inputs, and a validation split of 20% is used to monitor the model's performance on unseen data during training. The model is trained for up to 20 epochs with a batch size of 32. The early_stopping callback is included to halt training early if the validation loss does not improve, ensuring the model generalizes well. This training process allows the model to learn and optimize its weights to perform accurate genre classification.



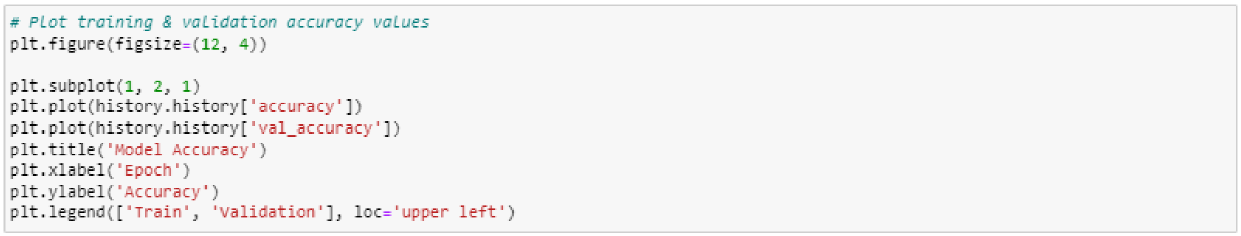
```
# Train the model
history = model.fit(
    {'acoustic_input': X_train[:, :acoustic_scaled.shape[1]], 'lyrics_input': X_train[:, acoustic_scaled.shape[1]:]},
    y_train,
    validation_split=0.2,
    epochs=20,
    batch_size=32,
    callbacks=[early_stopping]
)
```

4. RESULTS

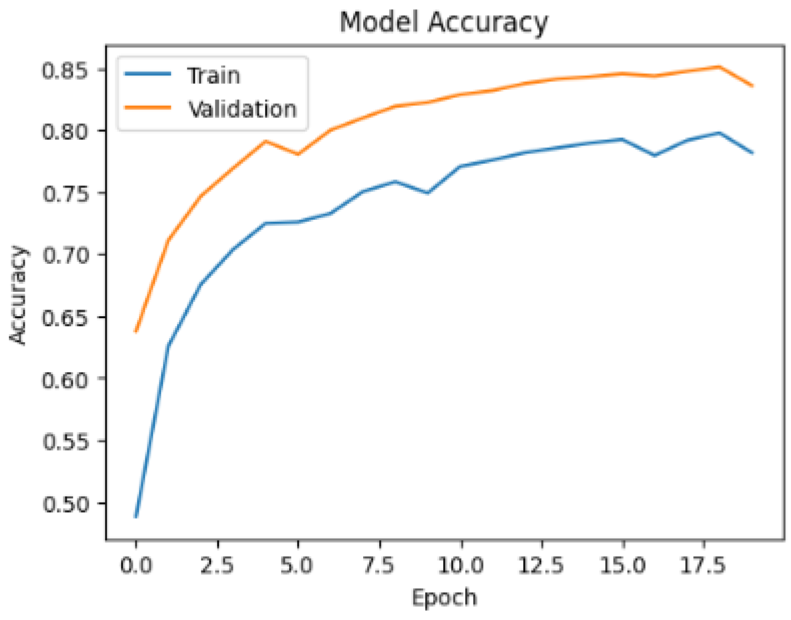
4.1 Evaluation of Metrics

The evaluation of the model's performance over 20 epochs shows a significant improvement in both training and validation metrics. The model's accuracy on the training data improved from 48.81% in the first epoch to 78.17% in the final epoch, while the validation accuracy increased from 63.79% to 83.60%. Concurrently, the training loss decreased from 2.4285 to 0.8723, and the validation loss decreased from 1.6318 to 0.6466, indicating that the model was learning effectively without overfitting significantly. After training, the model was evaluated on the test set, achieving a loss of 0.6424 and an accuracy of 83.75%. This demonstrates the model's ability to generalize well to unseen data, validating the efficacy of the preprocessing steps and the architecture combining acoustic features and lyrics. The use of early stopping ensured the model did not overfit the training data, as indicated by the stable validation loss and accuracy metrics.

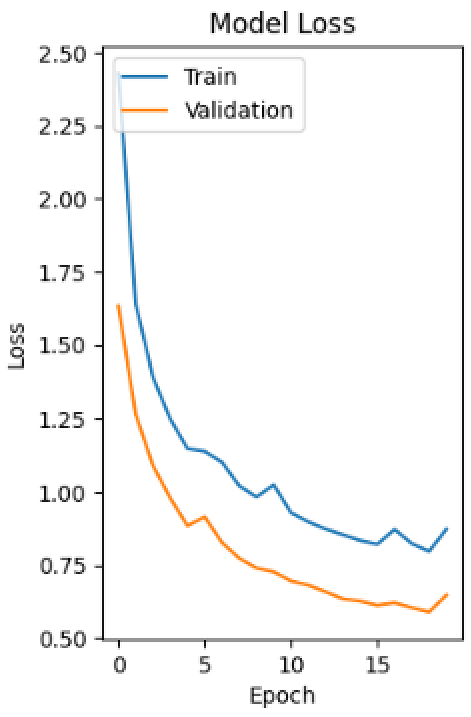
After plotting the training versus validation accuracy values, and training versus validation loss values, we obtained the following:



```python
# Plot training & validation accuracy values
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'], loc='upper left')
```



```
# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'], loc='upper left')

plt.show()
```



4.2 Evaluation of Results

The evaluation of the model's performance across 20 epochs demonstrates significant improvement in both training and validation metrics. The model's training accuracy increased from 48.81% to 78.17%, while validation accuracy improved from 63.79% to 83.60%, indicating effective learning without substantial overfitting. The final test set evaluation yielded a loss of 0.6424 and an accuracy of 83.75%, closely aligning with validation performance and confirming the model's generalization capability. These results validate the preprocessing steps and the combined acoustic-lyrics model architecture, showing that the model effectively learned to classify genres using the integrated features, with early stopping ensuring optimal performance by preventing overfitting.

5. CONCLUSION

In this project, we developed a deep-learning model for genre classification by integrating acoustic features and lyrics. Through extensive preprocessing, including text cleaning and feature scaling, we ensured high-quality input data. The model, combining dense and LSTM layers, effectively learned from both types of data, achieving a test accuracy of 83.75%. The use of early stopping helped maintain generalization by preventing overfitting. The results demonstrate that leveraging both acoustic features and lyrics significantly enhances genre classification performance, highlighting the potential for advanced multi-modal approaches in music classification tasks.

REFERENCES

[1] Oramas, S., Barbieri, F., Nieto Caballero, O., & Serra, X. (2018). Multimodal deep learning for music genre classification. *Transactions of the International Society for Music Information Retrieval. 2018; 1 (1): 4-21.*

[2] Bahuleyan, H. (2018). Music genre classification using machine learning techniques. *arXiv preprint arXiv:1804.01149*.

[3] Verdonck, T., Baesens, B., Óskarsdóttir, M., & vanden Broucke, S. (2021). Special issue on feature engineering editorial. *Machine learning*, 1-12.