ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗ ΒΙΟΙΑΤΡΙΚΗ

Προηγμένη Επεξεργασία Ήχου: Σχεδιασμός Τεχνολογίας για Μουσική Δημιουργία

Κωνστάντιου Γεωργία

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Υπεύθυνος

ΤΣΟΥΚΑΤΟΣ ΚΩΝΣΤΑΝΤΙΝΟΣ

Καθηγητής

Λαμία, 2025

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗ ΒΙΟΙΑΤΡΙΚΗ

Προηγμένη Επεξεργασία Ήχου: Σχεδιασμός Τεχνολογίας για Μουσική Δημιουργία

Κωνστάντιου Γεωργία

Τριμελής Επιτροπή:

ΤΣΟΥΚΑΤΟΣ ΚΩΝΣΤΑΝΤΙΝΟΣ , Καθηγητής (επιβλέπων)

ΔΕΛΗΜΠΑΣΗΣ ΚΩΝΣΤΑΝΤΙΝΟΣ , Καθηγητής

ΣΑΒΕΛΩΝΑΣ ΜΙΧΑΗΛ, Καθηγητής

Με ατομική μου ευθύνη και γνωρίζοντας τις κυρώσεις [(1)], που προβλέπονται από της διατάξεις της παρ. 6 του άρθρου 22 του Ν. 1599/1986, δηλώνω ότι:

1. *Δεν παραθέτω κομμάτια βιβλίων ή άρθρων ή εργασιών άλλων αυτολεξεί* ***χωρίς να τα περικλείω σε εισαγωγικά*** *και χωρίς να αναφέρω το συγγραφέα, τη χρονολογία, τη σελίδα. Η αυτολεξεί παράθεση χωρίς εισαγωγικά χωρίς αναφορά στην πηγή, είναι λογοκλοπή. Πέραν της αυτολεξεί παράθεσης, λογοκλοπή θεωρείται και η παράφραση εδαφίων από έργα άλλων, συμπεριλαμβανομένων και έργων συμφοιτητών μου, καθώς και η παράθεση στοιχείων που άλλοι συνέλεξαν ή επεξεργάσθηκαν, χωρίς αναφορά στην πηγή. Αναφέρω πάντοτε με πληρότητα την πηγή κάτω από τον πίνακα ή σχέδιο, όπως στα παραθέματα.*

2. *Δέχομαι ότι η αυτολεξεί* ***παράθεση χωρίς εισαγωγικά****, ακόμα κι αν συνοδεύεται από αναφορά στην πηγή σε κάποιο άλλο σημείο του κειμένου ή στο τέλος του, είναι αντιγραφή. Η αναφορά στην πηγή στο τέλος π.χ. μιας παραγράφου ή μιας σελίδας, δεν δικαιολογεί συρραφή εδαφίων έργου άλλου συγγραφέα, έστω και παραφρασμένων, και παρουσίασή τους ως δική μου εργασία.*

3. *Δέχομαι ότι υπάρχει επίσης περιορισμός στο μέγεθος και στη συχνότητα των παραθεμάτων που μπορώ να εντάξω στην εργασία μου εντός εισαγωγικών. Κάθε μεγάλο παράθεμα (π.χ. σε πίνακα ή πλαίσιο, κλπ), προϋποθέτει ειδικές ρυθμίσεις, και όταν δημοσιεύεται προϋποθέτει την άδεια του συγγραφέα ή του εκδότη. Το ίδιο και οι πίνακες και τα σχέδια*

4. *Δέχομαι όλες τις συνέπειες σε περίπτωση λογοκλοπής ή αντιγραφής.*

Ημερομηνία: 04/03/2025

Ο – Η Δηλ.

(1) «Όποιος εν γνώσει του δηλώνει ψευδή γεγονότα ή αρνείται ή αποκρύπτει τα αληθινά με έγγραφη υπεύθυνη δήλωση του άρθρου 8 παρ. 4 Ν. 1599/1986 τιμωρείται με φυλάκιση τουλάχιστον τριών μηνών. Εάν ο υπαίτιος αυτών των πράξεων σκόπευε να προσπορίσει στον εαυτόν του ή σε άλλον περιουσιακό όφελος βλάπτοντας τρίτον ή σκόπευε να βλάψει άλλον, τιμωρείται με κάθειρξη μέχρι 10 ετών.

TABLE OF CONTENTS

1. Introduction

Sound engineering is a multifaceted discipline that sits at the crossroads of art and science. [5]

It encompasses the technical processes of capturing, processing, and manipulating audio, and the creative vision required to translate artistic ideas into a polished sonic experience. Whether applied in recording studios, live performances, or post-production settings, sound engineering profoundly influences how we experience music, film, television, and other forms of media. [5]

In today's rapidly evolving audio landscape, the demand for high-quality sound has never been greater. Yet, the processes involved—from signal capture to the final mastering stage—are both intricate and time-consuming. [5] Recognizing these challenges, our project is designed to assist professionals in the sound engineering industry by automating many of the routine procedures. By streamlining tasks that traditionally require painstaking manual adjustments, we aim to free engineers to focus more on creativity and innovation, ultimately enhancing the overall quality of audio productions. [3] [4]

The Role of the Sound Engineer

At the heart of every audio production is the sound engineer—a professional who must skillfully balance technical expertise with artistic intuition. Sound engineering involves:

Definition & Purpose:

- It is both an art and a science, dedicated to capturing, processing, and manipulating sound to achieve the highest possible quality. This is essential not only for the technical accuracy of recordings but also for ensuring that the final output resonates emotionally with its audience.

The Balancing Act:

- The role of the sound engineer extends beyond mere technical execution. It involves translating an artistic vision into a dynamic and engaging auditory experience. This dual responsibility—ensuring technical clarity while nurturing creative expression—is what makes the sound engineer an indispensable part of any production. [5]

Signal Flow & Equipment Setup

Understanding signal flow is fundamental in sound engineering. It is the process by which audio travels from its source—such as a microphone or an instrument—through various stages of amplification, mixing, and digital conversion, ultimately reaching the final output.

- Understanding Signal Flow:

- Sound travels from its point of origin, passes through preamps and mixers, and is then converted from an analog signal into digital data using audio interfaces. This digital data is further processed within a Digital Audio Workstation (DAW), such as Pro Tools, Logic, or Ableton. Mastery of signal flow ensures that every step of this journey maintains clarity and minimizes interference or noise.
- Key Equipment:
    - Microphones: The choice and placement of microphones are critical in capturing the authentic character of the sound source.
    - Mixing Consoles & Audio Interfaces: These devices control and convert the analog signal to digital form, preparing it for further manipulation.
    - Digital Audio Workstations (DAWs): Software platforms where the majority of modern mixing, editing, and processing occur, providing the engineer with a versatile environment for creative work. [5]

Recording Techniques

Recording is the foundation upon which the entire production is built. High-quality recordings ensure that the subtle nuances of the performance are preserved for later stages of processing. [5]

Microphone Placement & Selection:

- Different microphones capture different aspects of a sound source. For example, close miking can capture intimate details, while ambient room miking can infuse the recording with spatial characteristics and depth.

Gain Staging:

- Proper gain staging is crucial for maintaining a clean signal. Setting the correct input levels prevents distortion and ensures that the recorded audio is robust and clear, forming a strong foundation for subsequent processing.

Room Acoustics:

- The physical environment in which recording takes place plays a significant role in the final sound. The acoustics of a room can add unique qualities to the recording. Techniques such as using absorptive panels or diffusers help manage unwanted reflections and ambient noise, allowing the true character of the performance to shine through.

Mixing and Audio Processing

Mixing is the art of combining various audio elements, such as vocals, instruments, and effects. [5] This involves adjusting levels, panning, and frequency ranges so that each component complements the others and contributes to the overall sound.

- Key Tools and Techniques:
    - Equalization (EQ): Shapes the tonal quality of individual tracks, ensuring that frequencies are balanced and clarity is maintained.
    - Compression: Bring quieter sounds forward to create a consistent listening experience.
    - Reverb & Delay: Create a sense of depth and ambiance that can enhance the emotional impact of the music.
    - Automation: Dynamically adjusts various parameters over time, adding movement and expressiveness to the mix.

In addition to the fundamental techniques, creativity in mixing is often enhanced by experimenting with unconventional effects, layering sounds in innovative ways, and applying dynamic processing creatively. Sometimes, breaking traditional rules leads to groundbreaking results, reinforcing that sound engineering is as much about exploration as it is about precision.

[12]

Mastering

Final Polish:

During mastering, the mixed track is refined with subtle adjustments in EQ, compression, and limiting. [5] The goal is to enhance clarity and ensure the track meets professional quality standards.

Purpose:

This final step guarantees that the listener experiences the full spectrum of the production's nuances and dynamic range.

Acoustics and the Recording Environment

The recording environment itself has a profound impact on the quality of the captured audio. Understanding and managing acoustics is essential for optimal sound production. [5]

Room:

- Effective room treatment minimizes unwanted reflections and ambient noise. Techniques such as installing bass traps, diffusers, and absorptive panels can transform a recording space into a controlled environment, free from acoustic anomalies.

Soundproofing:

- In situations where external noise is a concern, soundproofing techniques help isolate the recording environment. This is crucial for ensuring that the integrity of the audio is maintained, regardless of external influences.

Creative Space:

- Sometimes, the natural acoustic characteristics of a room can add a unique flavor to a recording.

A skilled engineer knows when to harness these quirks for artistic effect and when to treat the space to achieve a cleaner sound.

Troubleshooting and Maintenance

The ability to troubleshoot and maintain equipment is a critical skill for any sound engineer. [5]

Engineers must be proficient in diagnosing and resolving issues such as signal interference, electrical noise, and faulty cables. Regular maintenance and a thorough understanding of the equipment help prevent problems that could disrupt the creative process.

Adaptability:

In the fast-paced world of audio production, unexpected technical challenges can arise at any moment. The ability to quickly adapt and resolve these issues ensures that the production process remains smooth and uninterrupted.

Collaboration and Communication

Sound engineering is inherently collaborative. The best results are achieved when engineers, artists, and producers work together seamlessly. [5]

Teamwork:

Effective communication is crucial. Engineers must be able to understand and implement the creative vision of artists and producers, ensuring that every technical decision contributes to a unified final product.

Listening Skills:

A key part of the process is listening, not only to the music itself but also to feedback and subtle nuances.

The Significance of Sound Engineering

Sound engineering is not merely a technical endeavor; it is a critical component of the artistic process that influences how we experience audio across various media. [5]

Quality of Sound:

The clarity, balance, and precision of a recording can significantly impact the listener's experience. High-quality sound is essential for conveying the full emotional and artistic intent of the production.

Creative Expression:

Beyond its technical aspects, sound engineering offers a canvas for creative expression. Through techniques like EQ, compression, and innovative effects processing, engineers can shape audio in ways that evoke specific moods and enhance storytelling.

Live Sound Management:

In live settings, sound engineers are crucial for managing acoustics and ensuring that every audience member experiences the performance as intended, regardless of the venue's size or design.

Technical Innovation:

The field of sound engineering is continually evolving, with new technologies and methodologies emerging to further enhance audio production. This constant innovation drives the industry forward, enabling sophisticated and high-quality productions.

Automating the Future of Sound Engineering

In the modern era, sound engineering professionals are often challenged by the sheer volume of tasks that require precise and consistent execution. [3] [4] Many of these tasks are repetitive yet crucial for the overall quality of the audio.

Our project is designed to meet this challenge head-on by automating many of these procedures. By integrating advanced technologies and intelligent automation tools, we aim to:

Streamline Routine Tasks:

Automation can handle repetitive tasks with high accuracy, reducing the risk of human error and freeing up valuable time for creative work.

Enhance Workflow Efficiency:

By automating critical processes, sound engineers can focus on the artistic aspects of production, leading to a more efficient and productive workflow.

Improve Consistency and Quality:

Automated processes ensure that every step—from recording to final mastering—is executed with precision, resulting in consistently high-quality audio outputs across all projects.

Foster Innovation:

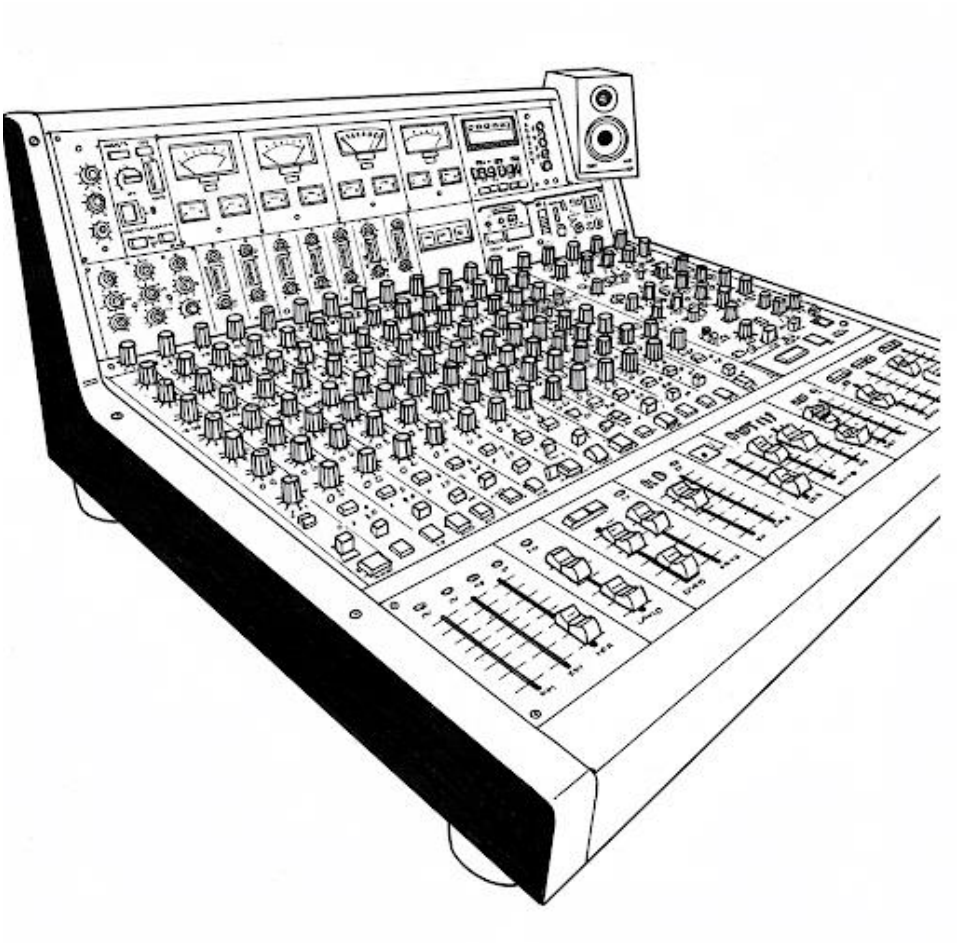
With routine technical tasks managed by automation, engineers are empowered to experiment with new creative techniques and push the boundaries of what is sonically possible.

By embracing automation, our project not only simplifies the technical side of sound engineering but also paves the way for a future where creative expression and technical excellence coexist seamlessly.

Sound engineering is a fast-moving field that blends technical skills with creativity. Key tasks include recording, mixing, mastering, and troubleshooting. Each part is vital in creating high-quality audio. A sound engineer's skill in handling equipment, managing sound environments, and working well with others brings audio to life and creates an engaging listening experience.

Our project focuses on helping sound engineering professionals by automating many routine technical tasks that can slow down creativity. By making these tasks easier, we aim to boost efficiency, reduce mistakes, and let engineers concentrate on the creative side of their work.

This overview introduces the basic ideas of sound engineering and encourages the use of new technology. As we progress, combining human creativity with smart automation will raise the quality of audio production and make every listening experience more engaging and professional.



[12]

2. A General Overview of Music

Music is a universal form of expression that has been intricately woven into the fabric of human life since ancient times. [5] At its core, music is the art of organizing sounds and silences over time, creating compositions that can evoke a wide range of emotions, tell compelling stories, or entertain listeners. Throughout history, as diverse cultures flourished and new technologies emerged, the form and function of music evolved significantly. Today, we recognize an extensive array of musical styles or genres, each characterized by its unique elements.

Historical Development and Separation of Genres

Early Music:

In the earliest periods of human existence, music was primarily linked to rituals, religious ceremonies, celebrations, and storytelling traditions that fostered community bonding. [5] During these times, there was little need to classify music into strict genres since the musical practices were predominantly similar and decentralized within small groups or tribes. Instruments were often handcrafted from available natural materials, producing sounds that were integral to cultural identity.

The Emergence of Distinct Styles:

As human societies grew increasingly complex, the diversification of musical styles began to take shape. [1] In ancient civilizations such as Greece, Egypt, and China, a variety of instruments and vocal techniques were developed, each serving particular purposes during religious ceremonies, court entertainment, or folk traditions. This early division laid the foundation for more defined musical categories, allowing for the expression of regional styles and cultural narratives.

The Middle Ages to the Modern Era:

The advent of musical notation during the Middle Ages marked a transformative period in music history that enabled composers and musicians to document and share their artistic ideas. With the later invention of the printing press, these ideas gained further reach. In Europe, for instance, the evolution from Medieval to Renaissance, and then into the Baroque, Classical, Romantic, and Modern periods, introduced increasingly distinct musical styles. Each period developed its own aesthetic guidelines regarding harmony, rhythm, and form, leading to the emergence of well-known composers who defined their respective eras. [1]

20th Century and Beyond:

The 20th century witnessed an explosion of musical genres catalyzed by advancements in recording technology, radio broadcasts, television, and ultimately the internet. Today, genres such as classical, jazz, rock, pop, hip-hop, electronic, and folk coexist and flourish. These genres are distinguished by various factors, including the types of instruments used, the structural elements of the music, production techniques, and the cultural influences or social contexts from which they arise. [1] [4]

Basic Differences Between Musical Genres

Instrumentation:

Classical music typically employs orchestral instruments such as violins, pianos, and woodwinds, creating complex arrangements suited for orchestras and chamber ensembles. Conversely, rock music centers around electric guitars, bass, and drums, often emphasizing a strong rhythmic drive. Electronic music, on the other hand, is characterized by its reliance on synthesizers, drum machines, and digital effects to create immersive soundscapes. [1]

Rhythm and Beat:

Some genres, notably pop and hip-hop, place a premium on a strong, steady beat that encourages listeners to dance. This rhythmic predictability contributes to mainstream appeal. In contrast, genres like classical and jazz often feature more varied rhythms, syncopation, and complex timing, inviting deeper listening and engagement. [1]

Harmony and Melody:

Classical and jazz music often delve into rich harmonies and improvisational melodies, creating intricate musical narratives. Popular songs, meanwhile, frequently prioritize simple and catchy tunes paired with straightforward chord progressions that resonate with a broad audience. [1]

Structure and Form:

The organization of a piece of music, known as its form, varies across genres. Classical symphonies may consist of multiple movements that contrast in tempo and mood, showcasing the composer's creativity. In contrast, a pop song typically adheres to a verse-chorus-verse structure that enhances its catchiness and memorability. [1]

Introduction to Musical Waves

At the foundation of all music is sound, which is a form of energy that travels in waves. Understanding these musical waves is crucial for grasping how music functions from a physics perspective.

What Are Sound Waves?

Sound waves are generated by vibrations that propagate through various media, such as air, water, or solid objects. When a musical instrument is played or a voice is heard, it creates these vibrations that our ears detect and interpret as sound. [5]

Key Properties of Sound Waves:

Frequency:

Frequency, measured in Hertz (Hz), is the number of vibrations occurring per second. It plays a crucial role in determining the pitch of a sound—higher frequencies are perceived as high-pitched sounds, while lower frequencies are associated with low-pitched sounds.

Amplitude:

Amplitude indicates the height of a sound wave and determines its loudness. A wave with a larger amplitude produces a louder sound, while a smaller amplitude results in softer sounds.

Wavelength:

The wavelength refers to the physical distance between two consecutive points in the wave. Higher frequencies correspond to shorter wavelengths, while lower frequencies have longer wavelengths.

Harmonics and Overtones:

Most musical sounds are not pure tones; rather, they consist of a fundamental frequency accompanied by additional frequencies known as harmonics or overtones. These extra frequencies contribute to the unique tone or timbre of different instruments. [5]

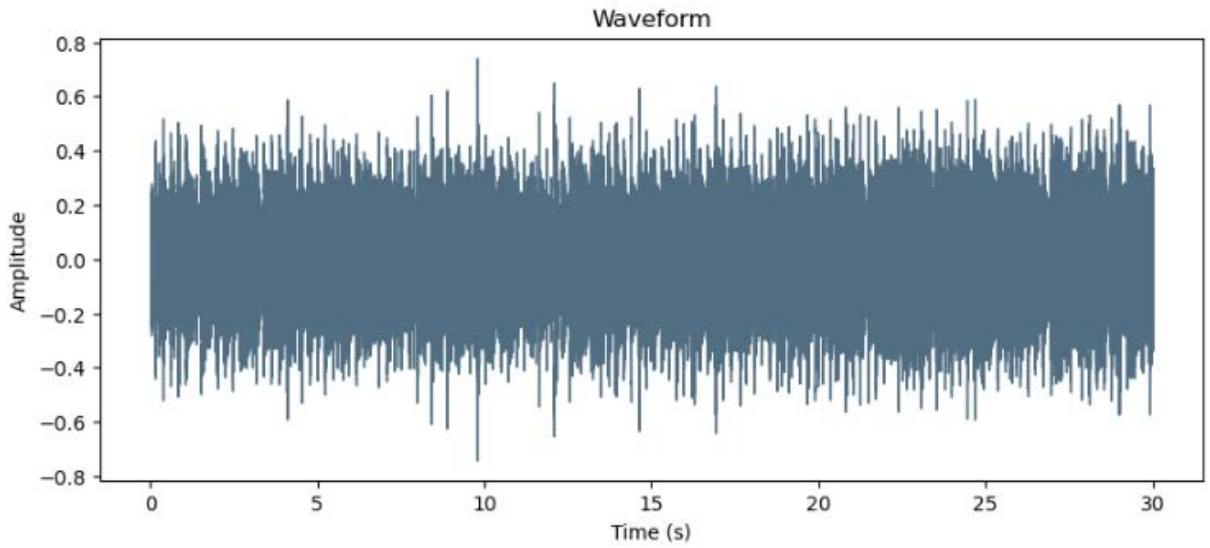
Graphical Representations of Musical Waves

To effectively study and analyze music, engineers and scientists employ graphical representations to visualize sound waves. These graphs facilitate the analysis of distinct characteristics inherent in different musical genres from a physics perspective. [5]

Common Graphs and Their Uses

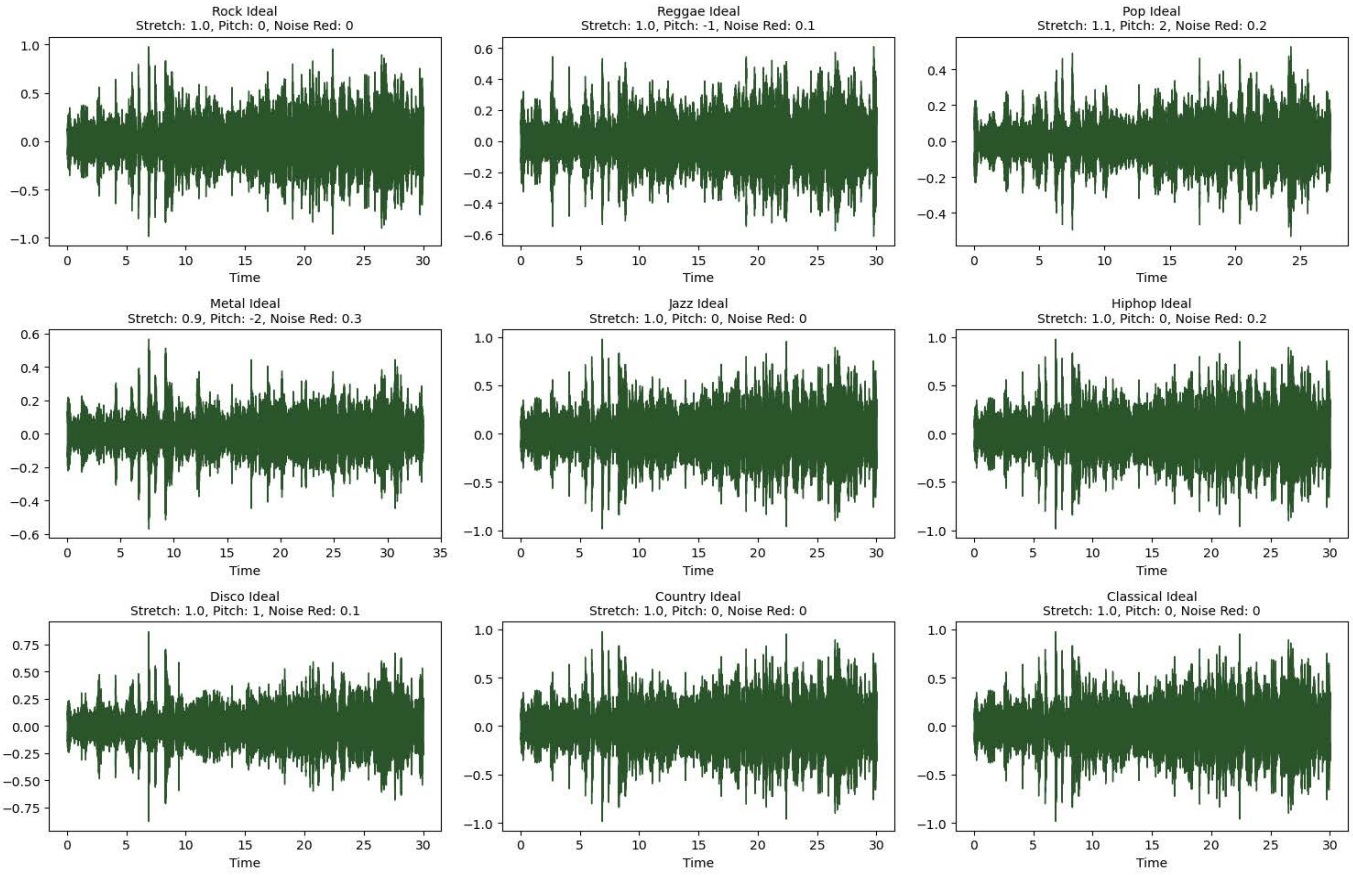
- Waveform Graphs:

A waveform graph illustrates how the amplitude of a sound wave fluctuates over time. By analyzing a waveform, one can discern the attack, sustain, and decay phases of a sound. For instance, sharp peaks might signify percussive instruments such as drums, frequently found in rock or hip-hop. Conversely, smoother curves may be typical in classical or ambient music, which often relies on softer dynamics.



The waveform reveals important phases of a sound:

- Attack: The initial phase where the sound rapidly rises in amplitude. For example, the sharp, quick onset of a drum hit in rock or hip-hop often appears as a pronounced spike.
- Sustain: The period during which the sound maintains a relatively steady amplitude. This can be seen in longer-held notes or chords in classical or ambient music.
- Decay/Release: The phase where the sound gradually fades away.



In our project, functions such as librosa.display.waveshow() are used to display these waveforms. By visually comparing the original and modified waveforms, users can immediately see how effects alter the signal's dynamics.

The characteristics of a waveform can also hint at the musical genre. Aggressive, percussive genres like rock or hip-hop typically exhibit abrupt changes, whereas genres like classical or ambient music often feature smoother, more flowing dynamics.

- Frequency Spectrum (Fourier Transform):

The Fourier Transform is a mathematical tool that changes a signal from the time domain to the frequency domain. When used with audio, it breaks down a complex sound wave into its individual frequencies.

Frequency Components:

The graph displays the strength / intensity of different frequency bands. High peaks show strong frequencies, while a wide spectrum suggests a mix of sounds.

Harmonics and Overtones:

Most musical sounds are not just single notes; they have a main frequency along with other harmonics that give each instrument its unique sound.

In our code, we use the Fourier Transform when we extract features like Mel-Frequency Cepstral Coefficients (MFCCs) through the function librosa.feature.mfcc(). These coefficients capture the sound's qualities and help classify music genres.

We also use the Short-Time Fourier Transform (STFT) to create spectrograms, which show how frequency components change over time.

- Spectrograms

A spectrogram shows sound in three dimensions: frequency, time, and amplitude. It looks like a heat map where:

- The horizontal axis shows time.

- The vertical axis shows frequency.

- The brightness or color shows the amplitude at each frequency.

Spectrograms help us see how a sound's frequency changes over time. They are particularly useful for:

- Tracking changes in music: For example, a jazz solo may show quick and detailed changes in sounds.

- Detecting sudden sounds: Quick bursts of energy, like drum hits, are easy to see in a spectrogram.

In our project, we create the spectrogram using librosa.display.specshow(). We use it for visual analysis and to compare how different effects. This allows users to adjust settings easily through our user interface. [10]

3. Foundations of Artificial Intelligence and Machine Learning

Artificial intelligence (AI) and machine learning (ML) are changing many fields, including healthcare, finance, and the arts. Recently, these technologies have made significant strides in music. They help researchers and musicians classify music genres, improve audio quality, and even create new music. This blend of technology and art deepens our understanding of music and provides new ways to express creativity and drive innovation.

In this thesis, we look into the basics of AI and ML, how these methods are used to analyze audio, and their practical applications in music processing. We emphasize techniques for feature extraction, model training and processing in digital audio workstations (DAWs), focusing on both theory and practice in this field. [9]

Artificial Intelligence (AI)

Artificial intelligence aims to create systems that can perform tasks requiring human intelligence, such as reasoning, understanding language, seeing, and making decisions. AI uses a variety of methods, including rule-based systems and data-driven approaches that rely on statistical techniques. [9]

Machine Learning (ML)

Machine learning is a part of AI that concentrates on algorithms that learn from data and make predictions without needing specific programming for every task. The main steps in the ML process include collecting data, extracting features, training models, evaluating them, and deploying them.

Important concepts include:

- Feature Extraction: In audio processing, this step converts raw audio into meaningful descriptors. Common features are Mel-Frequency Cepstral Coefficients (MFCCs), Zero Crossing Rate (ZCR), and Spectral Centroid. These features capture essential information about sound to help distinguish different music styles.

- Model Training and Validation: Supervised learning models, like logistic regression and support vector machines (SVM), are often used for tasks such as genre classification. During training, the model learns how input features relate to labels (like music genres). We validate the model's performance with separate test data to ensure it generalizes well. [9]

Algorithms and Their Mathematical Foundations:

- Logistic Regression: This algorithm models the probability that a given input belongs to a specific class using the logistic (sigmoid) function. [11] It is particularly effective when the relationship between features and labels is approximately linear. In our code, a logistic regression model is trained using scikit-learn's LogisticRegression class.

- Support Vector Machines (SVM): SVMs find the optimal hyperplane that separates classes in the feature space, maximizing the margin between different classes, to capture non-linear relationships in the data. [11]

These algorithms are conceptually straightforward but deliver strong classification results when features are carefully chosen and prepared.

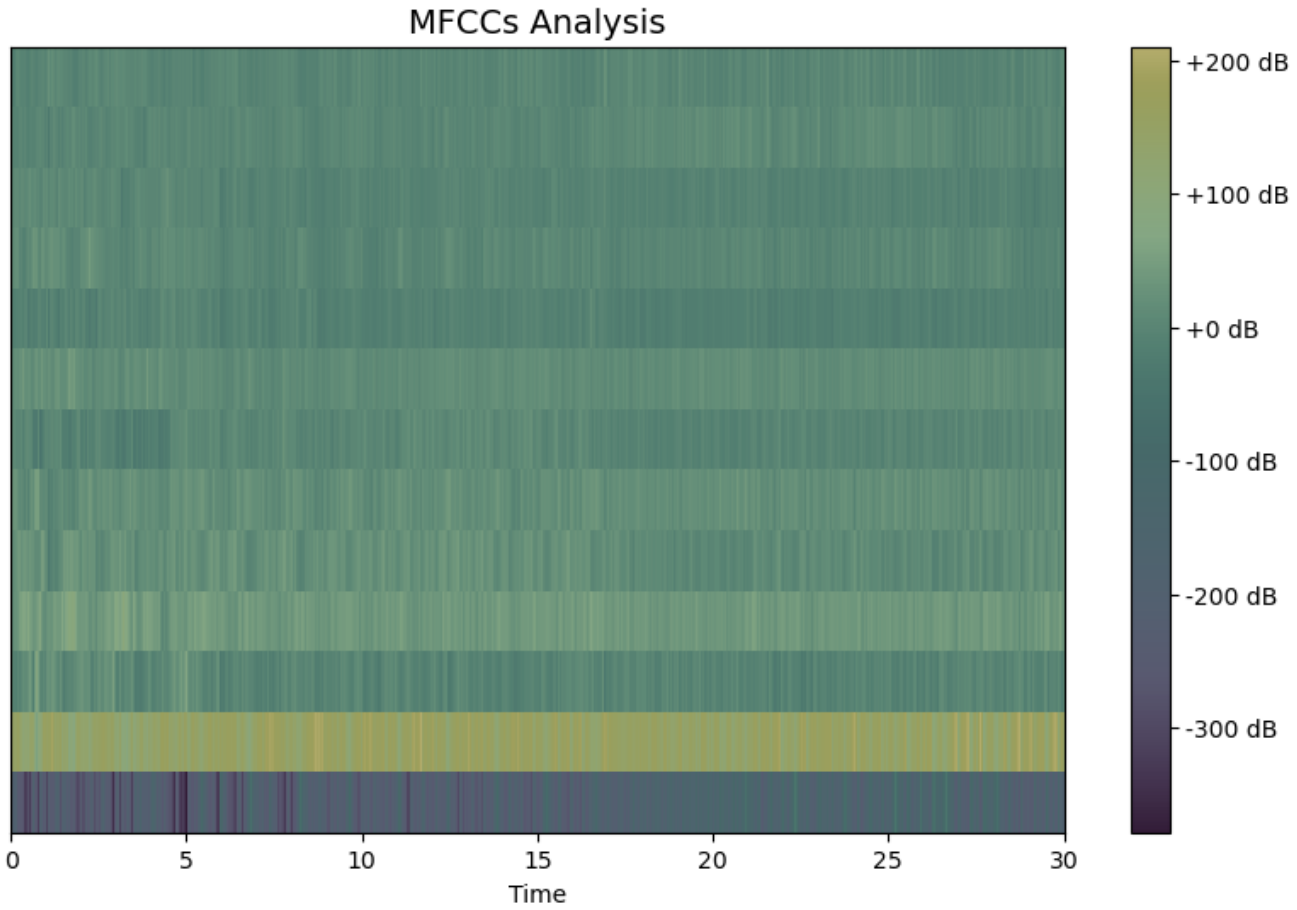
Machine Learning in Music

The key to any ML application in music is extracting relevant features from audio signals. These signals are complex, containing both time-varying amplitude (shown by waveforms) and frequency-related information (analyzed with spectral techniques). In our work, we use libraries like Librosa [10] to extract features such as MFCCs, ZCR, and Spectral Centroid.

- Mel-Frequency Cepstral Coefficients (MFCCs):

MFCCs are commonly used in audio processing because they represent the short-term power spectrum of sound, designed based on how humans hear.

We compute them by mapping the power spectrum to the mel scale and applying a discrete cosine transform (DCT). In our code, we use the function librosa.feature.mfcc() to extract these coefficients for genre classification.



This image shows a heatmap of Mel-Frequency Cepstral Coefficients (MFCCs) for a 30-second audio track. Here's a simple breakdown:

- Horizontal Axis (Time): This axis shows the time, from 0 to 30 seconds, and how the audio changes during that time.

- Vertical Axis (MFCC Index): Each row represents one MFCC. These coefficients highlight important qualities of the sound.

- Color Scale (dB): Warmer colors mean higher coefficient values in decibels, showing stronger energy in certain frequencies. Cooler colors mean lower coefficient values. [10]

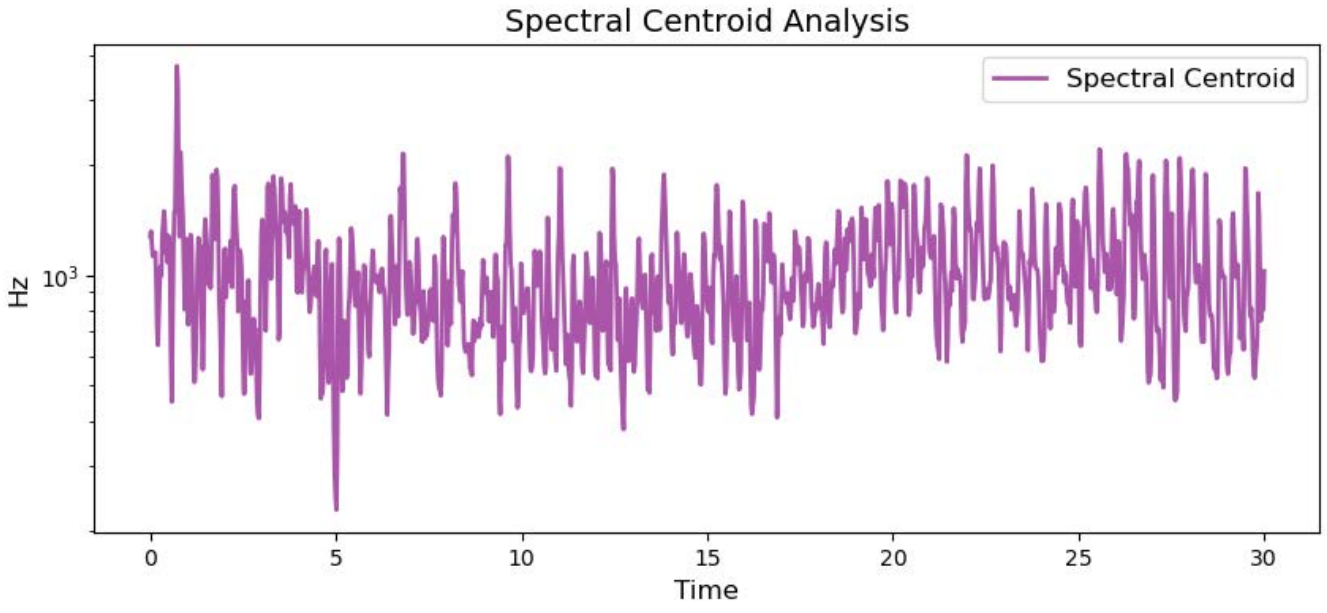
In short, this heatmap shows how the “color” or “character” of the sound changes every second. Bright areas indicate a strong emphasis on certain frequency shapes, while darker areas show less emphasis.

- Zero Crossing Rate (ZCR):

The ZCR measures how often the signal changes from positive to negative. This feature helps differentiate between percussive and tonal sounds since percussive elements usually have a higher zero-crossing rate.

- Spectral Centroid:

This feature indicates the "center of mass" of the spectrum and relates to the perceived brightness of a sound.



This graph shows how the Spectral Centroid of the audio changes over the 30-second track. In simple terms:

- Spectral Centroid = “Center of mass” of the frequencies.

Higher values (toward the top) mean the sound is “brighter” (more high-frequency energy).

Lower values mean the sound is “darker” (more low-frequency content).

- Horizontal axis (Time):

From 0 to 30 seconds, indicating how the track evolves over time.

- Vertical axis (Hz):

Shows the specific frequency (in Hertz) where the bulk of the energy is centered for each moment.

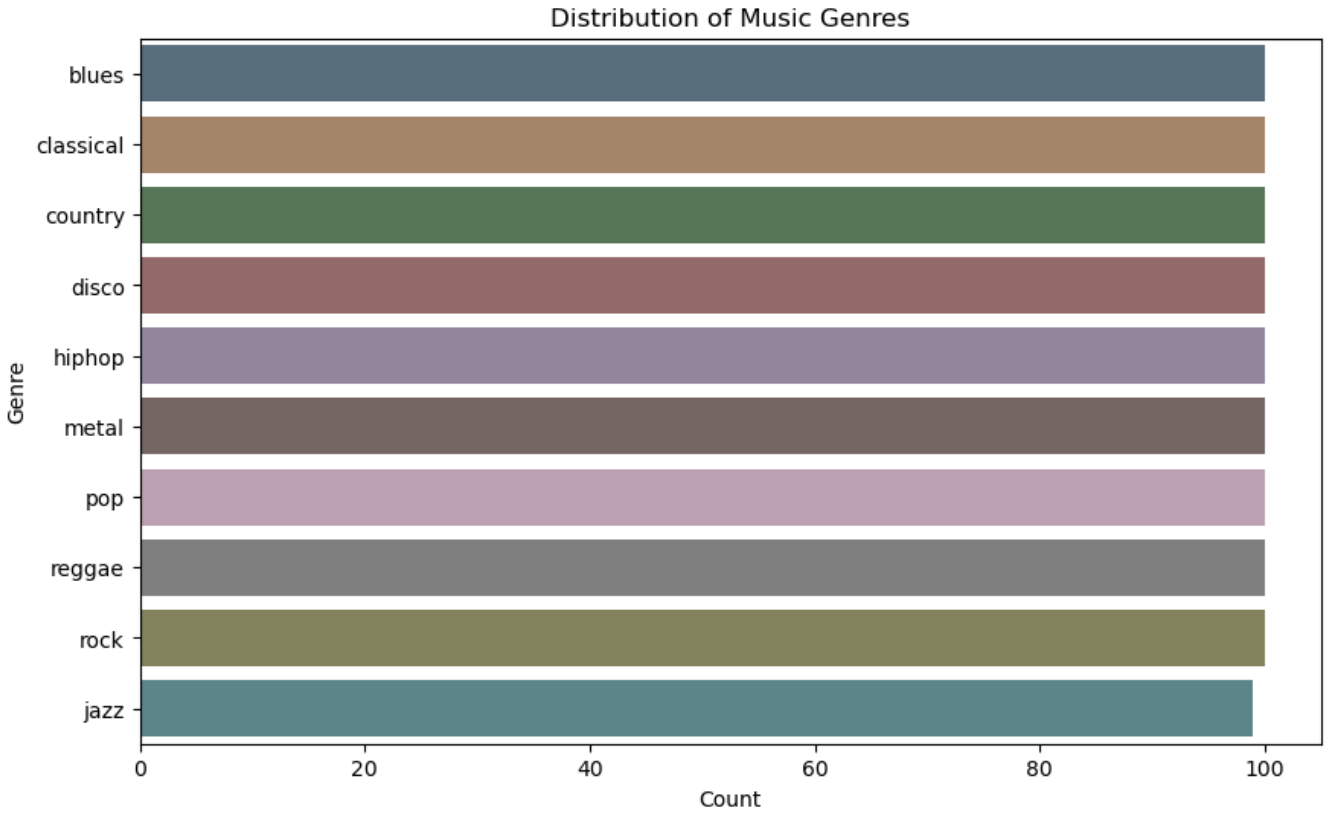
So, whenever the pink line spikes upward, it means the audio has stronger high frequencies at that instant. When it dips, the audio leans more toward lower frequencies. Essentially, this line gives us a quick snapshot of how “bright” or “dark” the audio is at each point in time. [10]

4. The project

Automatic Music Genre Classification and Audio Enhancement Using Machine Learning

This project uses machine learning to process music. We obtained a dataset from Kaggle [6] that includes many short audio tracks from different music genres. This variety and number of tracks are perfect for training supervised machine learning models to classify music by genre and for testing ways to enhance audio quality.

The dataset used in this study was obtained from the GTZAN dataset [6], which contains audio tracks across various musical genres. The dataset is organized into separate folders, each corresponding to a specific genre. For training and evaluation purposes, the entire dataset was split using an 80/20 ratio, where 80% of the samples were used for training and the remaining 20% were reserved for testing. For example, if the dataset comprised 1000 audio files, approximately 800 would form the training set and 200 would be used as the test set.



The dataset used in this study was divided into training and testing sets using an 80/20 split, as implemented in our code via scikit-learn's train_test_split function with test_size=0.2 and random_state=42. This means that 80% of the audio samples were used for training the models, while the remaining 20% were reserved for evaluating their performance.

The code we developed does feature extraction and trains models using techniques like logistic regression and support vector machines. [11]

Additionally, we created a graphical user interface (GUI) using Tkinter. This interface simulates a digital audio workstation (DAW) environment.

It allows users to load, record, process, visualize, and save audio files. The GUI provides real-time feedback on genre predictions and suggests audio transformations.

Detailed Code Analysis

Import Libraries

```
# Import libraries
import os
import numpy as np
import librosa
import librosa.display
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import X_train, X_test, y_train, y_test = train_te
import soundfile as sf
from tkinter import *
from tkinter import filedialog, messagebox
import seaborn as sns
import pandas as pd
```

These import statements load the necessary Python libraries:

- os, numpy: These libraries facilitate file handling and numerical operations.
- librosa and librosa.display: These are used for audio processing and for visualizing audio features, such as waveforms and spectrograms. [10]
- matplotlib.pyplot: This library is used for plotting graphs.
- scikit-learn modules: These modules assist with data splitting, label encoding, scaling, and training machine learning models, including logistic regression and support vector machines (SVM). [11]
- soundfile: This library is used for reading and writing audio files.
- Tkinter: This toolkit is employed for building the graphical user interface (GUI) and includes additional widgets such as filedialog and messagebox.

- seaborn and pandas: These libraries are utilized for data visualization (e.g., genre distribution) and for handling tabular data.

Additional Imports for DAW UI and Audio Recording

```
import simpleaudio as sa
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import sounddevice as sd
```

- simpleaudio: Enables playback of audio files.
- FigureCanvasTkAgg: Integrates Matplotlib figures within the Tkinter GUI.
- sounddevice: Provides audio recording functionality.

Tooltip Class

```
class CreateToolTip(object):
    """
    Creates a tooltip for a widget.
    Used to help the user understand the functionality of each element.
    """
    def __init__(self, widget, text='widget info'):
        self.widget = widget
        self.text = text
        self.tipwindow = None
        self.id = None
        self.x = self.y = 0
        widget.bind("<Enter>", self.enter)
        widget.bind("<Leave>", self.leave)
    def enter(self, event=None):
        self.schedule()
    def leave(self, event=None):
        self.unschedule()
        self.hidetip()
```

```
def schedule(self):
    self.unschedule()
    self.id = self.widget.after(500, self.showtip)
def unschedule(self):
    id = self.id
    self.id = None
    if id:
        self.widget.after_cancel(id)
def showtip(self, event=None):
    x, y, cx, cy = self.widget.bbox("insert")
    x = x + self.widget.winfo_rootx() + 25
    y = y + cy + self.widget.winfo_rooty() + 25
    self.tipwindow = tw = Toplevel(self.widget)
    tw.wm_overrideredirect(True)
    tw.wm_geometry("+%d+%d" % (x, y))
    label = Label(tw, text=self.text, justify=LEFT,
                  background="#ffffe0", relief=SOLID, borderwidth=1,
                  font=("tahoma", "8", "normal"))
    label.pack(ipadx=1)
def hidetip(self):
    tw = self.tipwindow
    self.tipwindow = None
    if tw:
        tw.destroy()
```

This class creates small pop-up help boxes (tooltips) that appear when a user hovers over a widget. Each widget can be given a short description to assist beginners.

Global Variables and Data Path

```
data_path = 'C:\\Users\\georg\\Downloads\\Data\\genres_original'

current_waveform = None
original_waveform = None  # Stores the original (pre-modification) waveform
current_sr = None
current_file = None

recording = False
recorded_frames = []
recording_stream = None

selected_mode = None  # "record" or "load"
```

These global variables keep track of important state information:

- data_path: This is the location of the dataset.
- current_waveform, original_waveform, current_sr: These variables hold the audio signal and its sample rate.
- recording and recorded_frames: These manage the audio recording process.
- selected_mode: This indicates whether you will record new audio or load a file you have saved.

Ideal Parameters for Each Genre

```
ideal_parameters = {
    "rock": {"stretch": 1.0, "pitch": 0, "genre": 0, "noise_reduction": 0},
    "reggae": {"stretch": 1.0, "pitch": -1, "genre": 1, "noise_reduction": 0.1},
    "pop": {"stretch": 1.1, "pitch": 2, "genre": 2, "noise_reduction": 0.2},
    "metal": {"stretch": 0.9, "pitch": -2, "genre": 3, "noise_reduction": 0.3},
    "jazz": {"stretch": 1.0, "pitch": 0, "genre": 4, "noise_reduction": 0},
    "hiphop": {"stretch": 1.0, "pitch": 0, "genre": 5, "noise_reduction": 0.2},
    "disco": {"stretch": 1.0, "pitch": 1, "genre": 6, "noise_reduction": 0.1},
    "country": {"stretch": 1.0, "pitch": 0, "genre": 7, "noise_reduction": 0},
    "classical": {"stretch": 1.0, "pitch": 0, "genre": 8, "noise_reduction": 0}
}
```

This dictionary defines “ideal” audio processing parameters (like time stretch and pitch shift) for each genre. These values are later used to suggest or apply modifications that match the characteristics of each musical style.

Functions for Safe Audio Loading and Feature Extraction

safe_load_audio

```
def safe_load_audio(audio_file, sr=22050):
    try:
        waveform, sr = librosa.load(audio_file, sr=sr)
        return waveform, sr
    except Exception as e:
        print(f"Error loading audio file {audio_file}: {e}")
        return None, None
```

This function safely loads an audio file using Librosa. If there is an error (e.g. the file is corrupt or missing), it catches the exception and returns None values.

extract_features_and_visualize

```
def extract_features_and_visualize(audio_file):
    try:
        waveform, sr = safe_load_audio(audio_file)
        if waveform is None:
            return None
        plt.figure(figsize=(10, 4))
        librosa.display.waveshow(waveform, sr=sr)
        plt.title("Waveform")
        plt.xlabel("Time (s)")
        plt.ylabel("Amplitude")
        plt.show()
        mfccs = librosa.feature.mfcc(y=waveform, sr=sr, n_mfcc=13)
        plt.figure(figsize=(10, 6))
        librosa.display.specshow(mfccs, sr=sr, x_axis='time')
        plt.colorbar(format='%+2.0f dB')
        plt.title("MFCCs")
        plt.tight_layout()
        plt.show()
        zcr = librosa.feature.zero_crossing_rate(y=waveform)
        spectral_centroid = librosa.feature.spectral_centroid(y=waveform, sr=sr)
        features = np.hstack((np.mean(mfccs, axis=1),
                              np.mean(zcr, axis=1),
                              np.mean(spectral_centroid, axis=1)))
        return features
    except Exception as e:
        print(f"Error processing {audio_file}: {e}")
        return None
```

- Loads the audio.
- Plots the waveform.
- Extracts MFCCs and displays their spectrogram. [10]
- Computes additional features (MFCC means, ZCR, and spectral centroid).
- Returns a concatenated feature vector.

Enhance_audio

```
def enhance_audio(audio_file, rate=1.2):
    try:
        waveform, sr = safe_load_audio(audio_file)
        enhanced_waveform = librosa.effects.time_stretch(waveform, rate=rate)
        enhanced_file = "enhanced_audio.wav"
        sf.write(enhanced_file, enhanced_waveform, sr)
        return enhanced_file
    except Exception as e:
        print(f"Error enhancing audio: {e}")
        return None
```

This function applies a time-stretch effect (modifying the playback speed) to an audio file and saves the result. It is an example of audio enhancement using Librosa's effects module. [10]

prepare_data

```
def prepare_data(data_path):
    data = []
    labels = []
    genres = os.listdir(data_path)
    for genre in genres:
        genre_path = os.path.join(data_path, genre)
        for file in os.listdir(genre_path):
            file_path = os.path.join(genre_path, file)
            features = extract_features_and_visualize(file_path)
            if features is not None:
                data.append(features)
                labels.append(genre)
    return np.array(data), np.array(labels)
```

This function iterates through the dataset directory, processes each audio file using the feature extraction function, and accumulates the feature vectors and corresponding genre labels into NumPy arrays. This prepares the dataset for training the ML models. [6]

visualize_data_distribution

```
def visualize_data_distribution(labels):
    plt.figure(figsize=(10, 6))
    sns.countplot(y=labels, order=pd.Series(labels).value_counts().index)
    plt.title("Distribution of Music Genres")
    plt.xlabel("Count")
    plt.ylabel("Genre")
    plt.show()
```

This function uses Seaborn to plot a count graph of the music genres, providing an overview of the dataset balance across genres.

open_file_and_classify

```
def open_file_and_classify():
    file_path = filedialog.askopenfilename(filetypes=[("Audio Files", "*.wav")])
    if file_path:
        features = extract_features_and_visualize(file_path)
        if features is not None:
            features_scaled = scaler.transform([features])
            genre_prediction = svm_model.predict(features_scaled)
            predicted_genre = encoder.inverse_transform(genre_prediction)[0]
            messagebox.showinfo("Prediction", f"Predicted Genre: {predicted_genre}")
            enhanced_audio = enhance_audio(file_path)
            if enhanced_audio:
                messagebox.showinfo("Enhancement", f"Enhanced audio saved as: {enhanced_audio}")
```

This function opens a file dialog to allow the user to select an audio file. It then:

- Extracts and visualizes features.
- Scales the features.
- Uses a pre-trained SVM model to predict the genre. [11]
- Displays the prediction and, optionally, enhances the audio.

Simple UI for Classification

```
def create_ui():
    root = Tk()
    root.title("Music Classification & Enhancement")
    root.geometry("500x300")
    Label(root, text="Music Genre Classifier", font=("Helvetica", 16)).pack(pady=10)
    instructions = """
    1. Click 'Open File' to load an audio file (.wav).
    2. The system will predict the genre.
    3. Optionally, it will enhance the audio file.
    """
    Label(root, text=instructions, justify=LEFT, wraplength=400).pack(pady=10)
    Button(root, text="Open File", command=open_file_and_classify, font=("Helvetica",
12)).pack(pady=20)
    Button(root, text="Exit", command=root.quit, font=("Helvetica", 12)).pack(pady=10)
    root.mainloop()
```

This function creates a basic Tkinter GUI that allows the user to open an audio file for classification and enhancement. It demonstrates a minimal UI design.

Common Feature Extraction Function

```
def extract_features_from_signal(signal, sr):
    # This function applies the same feature extraction logic used in other scenarios.
    mfccs = librosa.feature.mfcc(y=signal, sr=sr, n_mfcc=13)
    zcr = librosa.feature.zero_crossing_rate(y=signal)
    spectral_centroid = librosa.feature.spectral_centroid(y=signal, sr=sr)
    features = np.hstack((np.mean(mfccs, axis=1),
                          np.mean(zcr, axis=1),
                          np.mean(spectral_centroid, axis=1)))
    return features

def extract_features(audio_file):
    waveform, sr = safe_load_audio(audio_file)
    if waveform is None:
        return None
    return extract_features_from_signal(waveform, sr)
```

These functions simplify how we extract features so they can be reused easily. We use them in both preparing the data and in the user interface parts of the code.

Functions for the DAW-Style UI

Dynamic Knob Suggestions

```
def dynamic_knob_suggestions(features, predicted_genre):
    mean_mfcc = np.mean(features[:13])
    zcr_val = features[13]
    spectral_centroid_val = features[14]
    if spectral_centroid_val > 1500:
        stretch = 0.95
    else:
        stretch = 1.05
    pitch = int(np.round((mean_mfcc - 20) / 5))
    noise_reduction = max(0, 0.5 - zcr_val)
    genre_mapping = {"rock": 0, "reggae": 1, "pop": 2, "metal": 3, "jazz": 4, "hiphop": 5, "disco": 6,
"country": 7, "classical": 8}
    genre_knob = genre_mapping.get(predicted_genre, 0)
    return {"stretch": stretch, "pitch": pitch, "genre": genre_knob, "noise_reduction":
noise_reduction}
```

This function computes suggested UI knob values (for time stretch, pitch shift, etc.) based on the extracted features and the predicted genre. [1]

These suggestions help the user adjust the audio processing parameters to match the style of the music.

Update Knob Positions

```
def update_knob_positions(predicted_genre, features=None):
    if features is not None:
        dynamic_values = dynamic_knob_suggestions(features, predicted_genre)
    else:
        dynamic_values = {"stretch": 1.0, "pitch": 0, "genre": 0, "noise_reduction": 0}
    ideal = ideal_parameters.get(predicted_genre, {"stretch": 1.0, "pitch": 0, "genre": 0,
"noise_reduction": 0})
    if abs(dynamic_values["stretch"] - ideal["stretch"]) > 0.05:
        dynamic_values["stretch"] = ideal["stretch"]
    if abs(dynamic_values["pitch"] - ideal["pitch"]) > 1:
        dynamic_values["pitch"] = ideal["pitch"]
    if abs(dynamic_values["noise_reduction"] - ideal["noise_reduction"]) > 0.05:
        dynamic_values["noise_reduction"] = ideal["noise_reduction"]
    stretch_scale.set(dynamic_values["stretch"])
    pitch_scale.set(dynamic_values["pitch"])
    genre_scale.set(dynamic_values["genre"])
    noise_reduction_scale.set(dynamic_values["noise_reduction"])
```

This function updates the UI controls (knobs/sliders) with either dynamically suggested values or fixed "ideal" values for the predicted genre. It ensures that the interface reflects parameters that are consistent with the genre characteristics.

Update Feature Comparison Diagram

```
def update_feature_comparison_diagram():
    if current_file is None:
        messagebox.showwarning("Warning", "No audio file loaded for feature comparison")
        return
    waveform, sr = safe_load_audio(current_file)
    if waveform is None:
        messagebox.showerror("Error", "Could not load audio for comparison")
        return
    mfccs = librosa.feature.mfcc(y=waveform, sr=sr, n_mfcc=13)
    zcr = librosa.feature.zero_crossing_rate(y=waveform)
    spectral_centroid = librosa.feature.spectral_centroid(y=waveform, sr=sr)
    avg_mfcc = np.mean(mfccs, axis=1)
    avg_zcr = np.mean(zcr)
    avg_sc = np.mean(spectral_centroid)

    mfcc_threshold = mfcc_sensitivity_scale.get()
    zcr_threshold = zcr_sensitivity_scale.get()
    sc_threshold = sc_sensitivity_scale.get()

    fig, axs = plt.subplots(3, 1, figsize=(10, 12))
```

```
    axs[0].bar(range(1, 14), avg_mfcc, color='skyblue', label='MFCCs')
    axs[0].axhline(y=mfcc_threshold, color='red', linestyle='--', linewidth=2, label=f'Threshold:
{mfcc_threshold}')
    axs[0].set_title("MFCC Comparison", fontsize=14)
    axs[0].set_xlabel("MFCC Coefficient", fontsize=12)
    axs[0].set_ylabel("Value", fontsize=12)
    axs[0].legend(fontsize=12)
    axs[0].grid(True)

    axs[1].bar(["ZCR"], [avg_zcr], color='lightgreen', label=f'Avg ZCR: {avg_zcr:.3f}')
    axs[1].axhline(y=zcr_threshold, color='red', linestyle='--', linewidth=2, label=f'Threshold:
{zcr_threshold}')
    axs[1].set_title("Zero Crossing Rate Comparison", fontsize=14)
    axs[1].set_ylabel("ZCR Value", fontsize=12)
    axs[1].legend(fontsize=12)
    axs[1].grid(True)

    axs[2].bar(["Spectral Centroid"], [avg_sc], color='plum', label=f'Avg SC: {avg_sc:.1f} Hz')
    axs[2].axhline(y=sc_threshold, color='red', linestyle='--', linewidth=2, label=f'Threshold:
{sc_threshold} Hz')
    axs[2].set_title("Spectral Centroid Comparison", fontsize=14)
    axs[2].set_ylabel("Frequency (Hz)", fontsize=12)
    axs[2].legend(fontsize=12)
    axs[2].grid(True)

    plt.tight_layout()
    plt.show()
```

This function computes average feature values from the currently loaded audio file and compares them against threshold values (adjustable via UI sliders), helping the user visually assess the audio's characteristics.

Analyze Features

```
def analyze_features():
    if current_file is None:
        messagebox.showwarning("Warning", "No audio file loaded for analysis")
        return
    features = extract_features(current_file)
    if features is None:
        messagebox.showerror("Error", "Could not extract features")
        return
    waveform, sr = safe_load_audio(current_file)
    mfccs = librosa.feature.mfcc(y=waveform, sr=sr, n_mfcc=13)
    plt.figure(figsize=(10, 6))
    librosa.display.specshow(mfccs, sr=sr, x_axis='time', cmap='viridis')
    plt.colorbar(format='%+2.0f dB')
    plt.title("MFCCs Analysis", fontsize=14)
    plt.show()
    spectral_centroid = librosa.feature.spectral_centroid(y=waveform, sr=sr)
    frames = range(len(spectral_centroid[0]))
    t = librosa.frames_to_time(frames, sr=sr)
    plt.figure(figsize=(10,4))
    plt.semilogy(t, spectral_centroid[0], label='Spectral Centroid', color='magenta', linewidth=2)
    plt.ylabel('Hz', fontsize=12)
    plt.xlabel('Time', fontsize=12)
    plt.title("Spectral Centroid Analysis", fontsize=14)
    plt.legend(fontsize=12)
    plt.show()
```

This function analyzes the features of the currently loaded audio file by plotting:

- An MFCC spectrogram.
- A time-series plot of the spectral centroid.

These visualizations help in understanding the audio signal's structure and timbral content.

Save Audio

```
def save_audio():
    global current_waveform, current_sr
    if current_waveform is not None and current_sr is not None:
        file_path = filedialog.asksaveasfilename(defaultextension=".wav", filetypes=[("Wave Files",
"*.wav")])
        if file_path:
            sf.write(file_path, current_waveform, current_sr)
            messagebox.showinfo("Saved", f"Audio saved to: {file_path}")
    else:
        messagebox.showwarning("Warning", "No audio loaded")
```

This function saves the currently processed audio waveform to a WAV file using a file save dialog.

It uses the soundfile library to write the audio data.

Start and Stop Recording

```
def start_recording():
    global recording, recorded_frames, recording_stream, current_sr
    recording = True
    recorded_frames.clear()
    current_sr = 22050
    def callback(indata, frames, time, status):
        if status:
            print(status)
        recorded_frames.append(indata.copy())
    recording_stream = sd.InputStream(samplerate=current_sr, channels=1, callback=callback)
    recording_stream.start()
    messagebox.showinfo("Recording", "Recording started...")

def stop_recording():
    global recording, recorded_frames, recording_stream, current_waveform, current_sr,
original_waveform, update_waveform_plot
    if recording_stream is not None:
        recording_stream.stop()
        recording_stream.close()
    recording = False
    if recorded_frames:
        audio_data = np.concatenate(recorded_frames, axis=0).flatten()
        current_waveform = audio_data
        if original_waveform is None:
            original_waveform = current_waveform.copy()
        update_waveform_plot()  # This function updates the waveform display in the UI
        try:
            # Use common feature extraction function on recorded audio
            features = extract_features_from_signal(current_waveform, current_sr)
            if features is not None:
                features_scaled = scaler.transform([features])
                genre_prediction = svm_model.predict(features_scaled)
                predicted_genre = encoder.inverse_transform(genre_prediction)[0]
                pred_genre_label.config(text=f"Predicted Genre: {predicted_genre}")
                update_knob_positions(predicted_genre, features)
        except Exception as e:
            print("Error during feature extraction on recorded audio:", e)
        messagebox.showinfo("Recording", "Recording stopped and processed.")
```

- start_recording:

Initializes audio recording using the sounddevice library. It sets up a callback to capture incoming audio frames.

- stop_recording:

Stops the recording, concatenates the recorded frames into a single waveform, updates the waveform display, extracts features from the recorded audio, and predicts the genre using the pre-trained ML model.

Plot Ideal Waveforms for Each Genre

```
def plot_ideal_waveforms():
    if current_waveform is None:
        messagebox.showwarning("Warning", "No audio loaded for processing.")
        return
    genres = list(ideal_parameters.keys())
    num_genres = len(genres)
    nrows = int(np.ceil(np.sqrt(num_genres)))
    ncols = int(np.ceil(num_genres / nrows))
    fig, axs = plt.subplots(nrows, ncols, figsize=(15, 10))
    axs = np.atleast_2d(axs)
    for idx, genre in enumerate(genres):
        params = ideal_parameters[genre]
        try:
            processed = current_waveform.copy()
            processed = librosa.effects.time_stretch(processed, rate=params["stretch"])
            if params["pitch"] != 0:
                processed = librosa.effects.pitch_shift(processed, sr=current_sr,
n_steps=params["pitch"])
        except Exception as e:
            print(f"Error processing for genre {genre}: {e}")
            processed = current_waveform
        r = idx // ncols
        c = idx % ncols
        ax = axs[r, c]
        librosa.display.waveshow(processed, sr=current_sr, ax=ax, color='green')
        ax.set_title(f"{genre.capitalize()} Ideal\nStretch: {params['stretch']}, Pitch: {params['pitch']},
Noise Red: {params['noise_reduction']}", fontsize=10)
    for i in range(idx+1, nrows*ncols):
        r = i // ncols
        c = i % ncols
        axs[r, c].axis('off')
    plt.tight_layout()
    plt.show()
```

This function applies the “ideal” transformation parameters for each genre (time stretching and pitch shifting) to the current waveform. It then plots these idealized waveforms in a grid, allowing the user to visually compare how the processed audio would look for different genres.

DAW-Style UI Creation

```
def create_daw_ui():
    global current_waveform, current_sr, current_file, original_waveform
    global stretch_scale, pitch_scale, genre_scale, pred_genre_label, canvas, ax
    global mfcc_sensitivity_scale, zcr_sensitivity_scale, sc_sensitivity_scale, noise_reduction_scale,
update_waveform_plot

    genres_list = ["rock", "reggae", "pop", "metal", "jazz", "hiphop", "disco", "country", "classical"]

…
```

This function builds an advanced DAW-style user interface using Tkinter. The UI is divided into multiple columns:

- Column 0: Basic action buttons for loading audio, recording, stopping, analyzing, playing, applying effects, saving, and displaying the predicted genre.
- Column 1: Sliders to set thresholds for MFCC, ZCR, and spectral centroid comparisons.
- Column 2: Sliders for adjusting audio processing parameters such as time stretch, pitch shift, genre transformation, and noise reduction.
- Column 3: Buttons to display ideal waveforms and to navigate back to the main splash screen.

The UI also embeds a Matplotlib canvas to display waveform comparisons between the original and modified audio.

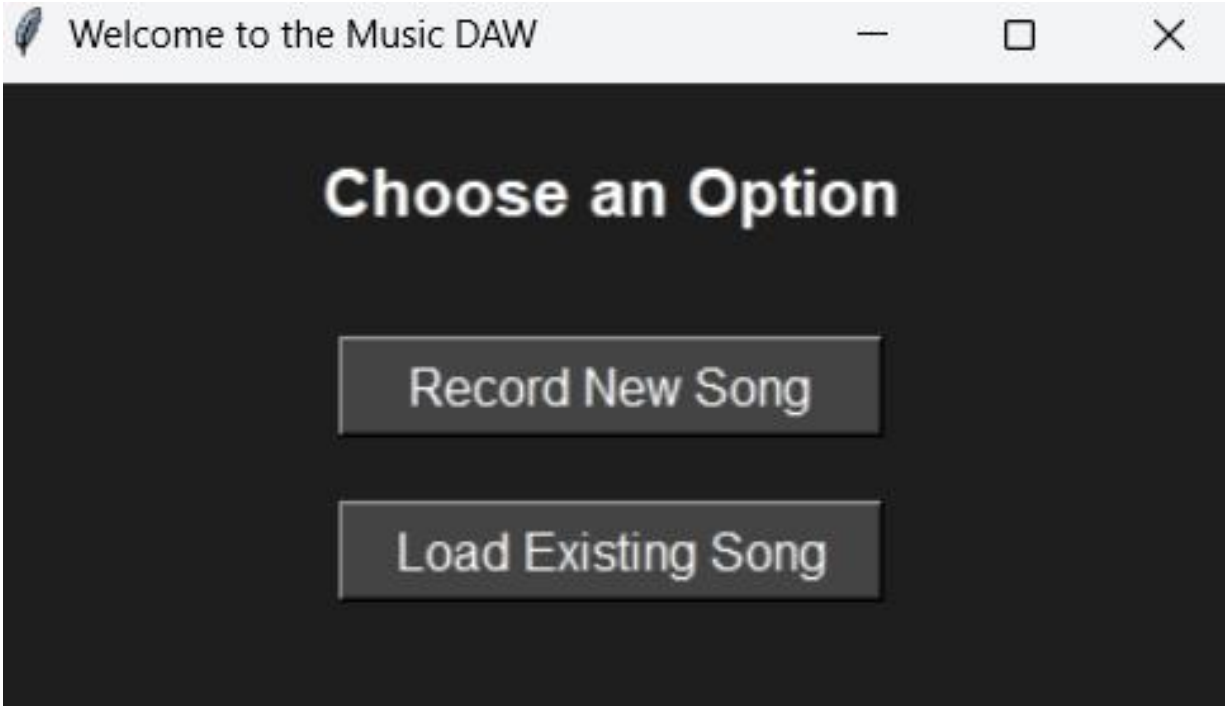
LOADING AUDIO UI:

RECORDING AUDIO UI:

Splash Screen

```
def create_splash_screen():
    global selected_mode
    splash = Tk()
    splash.title("Welcome to the Music DAW")
    splash.geometry("400x300")
    splash.configure(bg="#1e1e1e")

    lbl = Label(splash, text="Choose an Option", font=("Helvetica", 16, "bold"), bg="#1e1e1e",
fg="white")
    lbl.pack(pady=20)
    CreateToolTip(lbl, "Select whether you want to record a new track or load an existing song.")

    def choose_mode(mode):
        global selected_mode
        selected_mode = mode
        splash.destroy()
        create_daw_ui()

    btn_record_new = Button(splash, text="Record New Song", command=lambda:
choose_mode("record"), font=("Helvetica", 12), bg="#444444", fg="white", width=18)
    btn_record_new.pack(pady=10)
    CreateToolTip(btn_record_new, "Choose to record a new track via the microphone.")

    btn_load_existing = Button(splash, text="Load Existing Song", command=lambda:
choose_mode("load"), font=("Helvetica", 12), bg="#444444", fg="white", width=18)
    btn_load_existing.pack(pady=10)
    CreateToolTip(btn_load_existing, "Choose to load an existing song.")

    splash.mainloop()
```

This function creates a splash screen that allows the user to choose between recording a new song or loading an existing one. The selected mode is stored in a global variable and then used to configure the DAW UI accordingly.



Model Training and Data Visualization

```
X, y = prepare_data(data_path)
visualize_data_distribution(y)

encoder = LabelEncoder()
y_encoded = encoder.fit_transform(y)

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

classification_report, X_test, y_train, y_test = train_test_split(X_scaled, y_encoded, test_size=0.2,
random_state=42, stratify=y_encoded)

lr_model = LogisticRegression(max_iter=500)
lr_model.fit(X_train, y_train)

svm_model = SVC(kernel='rbf', probability=True)
svm_model.fit(X_train, y_train)

print("Logistic Regression Performance:")
print(classification_report(y_test, lr_model.predict(X_test), target_names=encoder.classes_))

print("SVM Performance:")
print(classification_report(y_test, svm_model.predict(X_test), target_names=encoder.classes_))
```

- Data Preparation:

The prepare_data function is called to extract features from all audio files in the dataset. The distribution of genres is then visualized.

- Preprocessing:

The genre labels are encoded numerically, and the features are scaled.

- Data Splitting:

The dataset is divided into training and testing sets.

- Model Training:

Both logistic regression and SVM models are trained on the training data.

- Evaluation:

  The performance of each model is printed using classification metrics (precision, recall, F1-score), which provide insight into how well the models generalize to unseen data.

After we encode labels and scale features, we train models like Logistic Regression and SVM to classify audio into different genres. We can also consider other methods such as deep learning, ensemble techniques, transfer learning, and tuning model settings.

After training and testing our models, we obtained classification reports for both Logistic Regression and SVM:

```
Logistic Regression Performance:
              precision    recall  f1-score   support

       blues       0.39      0.43      0.41        21
   classical       0.80      1.00      0.89        12
     country       0.42      0.33      0.37        24
       disco       0.63      0.55      0.59        22
     hiphop        0.44      0.53      0.48        15
        jazz       0.62      0.48      0.54        27
       metal       0.68      0.94      0.79        18
         pop       0.67      0.84      0.74        19
      reggae       0.42      0.36      0.39        22
        rock       0.35      0.30      0.32        20

    accuracy                           0.55       200
   macro avg       0.54      0.58      0.55       200
weighted avg       0.53      0.55      0.53       200
```

Logistic Regression Results

- Accuracy: The overall accuracy of Logistic Regression is 55%.
- Per-Class Performance:
  - *Classical* music achieved very high recall (1.00) and a strong F1-score (0.89), indicating that nearly all classical samples were correctly identified.
  - In contrast, genres like *rock* and *blues* show low precision (around 0.35 and 0.39, respectively) and low recall (0.30 and 0.43), meaning these genres are often misclassified.
- Interpretation: These results suggest that while the Logistic Regression model works well for some genres (e.g., classical), it struggles with others where the audio features might overlap more.

SVM Performance:

	precision	recall	f1-score	support
blues	0.60	0.57	0.59	21
classical	0.65	0.92	0.76	12
country	0.67	0.42	0.51	24
disco	0.57	0.59	0.58	22
hiphop	0.44	0.47	0.45	15
jazz	0.71	0.56	0.63	27
metal	0.64	0.78	0.70	18
pop	0.73	0.84	0.78	19
reggae	0.50	0.59	0.54	22
rock	0.33	0.30	0.32	20
accuracy			0.58	200
macro avg	0.58	0.60	0.58	200
weighted avg	0.59	0.58	0.58	200

SVM Results

- Accuracy: The SVM model achieves a slightly higher overall accuracy of 58%.
- Per-Class Performance:
    - *Pop* music shows improvement with a precision of 0.73 and recall of 0.84, leading to an F1-score of 0.78.
    - *Blues* and *jazz* also see better performance under SVM compared to Logistic Regression.
    - However, *rock* remains challenging with a precision of 0.33 and recall of 0.30.
- Interpretation: The SVM model generally improves classification for several genres, suggesting that its ability to find an optimal hyperplane in feature space is beneficial. Nevertheless, certain genres like rock are still problematic, perhaps due to similar audio characteristics shared with other genres.

✓ SVM outperforms Logistic Regression overall (58% vs. 55% accuracy) and improves performance for genres like pop, blues, and jazz.
✓ Despite these improvements, both models struggle with certain genres (e.g., rock), indicating that further feature refinement or model tuning may be necessary.

- stratify=y_encoded : ensures that the training and testing sets will have the same proportion of samples for each genre as in the full dataset.

Display Genre Spectra Table

```
def display_genre_spectra_table():
    feature_columns = [f'MFCC_{i+1}' for i in range(13)] + ['ZCR', 'SC']
    df = pd.DataFrame(X, columns=feature_columns)
    df['Genre'] = y
    mean_table = df.groupby('Genre').mean()
    std_table = df.groupby('Genre').std()
    print("\n==== Mean Feature Values by Genre ====")
    print(mean_table)
    print("\n==== Standard Deviations of Features by Genre ====")
    print(std_table)

display_genre_spectra_table()
```

This function creates a Pandas DataFrame from the extracted features, groups the data by genre, and prints out the mean and standard deviation of the features. This provides statistical insight into the characteristics of each genre.

Start the Application

```
create_splash_screen()
```

This line launches the splash screen, which in turn starts the entire DAW UI process. From here, the user can choose the mode (record or load) and interact with the full application.

5. Future Work

There are several promising areas for future research and development based on this work, each offering unique opportunities for advancement in audio processing technologies:

Real-Time Plugin Development:

The next step involves creating an innovative real-time plugin that is fully compatible with leading Digital Audio Workstation (DAW) software such as Ableton Live, Pro Tools, and Logic Pro. This plugin would serve as an essential tool for sound engineers and music producers, enabling functionalities like real-time genre classification, automated audio enhancement, and intelligent parameter adjustments seamlessly integrated within their production workflows. To achieve this, the development team will focus on optimizing the underlying algorithms for low latency performance, ensuring that they can process audio in milliseconds without noticeable delay. Additionally, efforts will be made to ensure compatibility with a wide range of DAWs and their plugin standards, such as VST, AU, and AAX formats, significantly enhancing the audio processing efficiency and creativity in music production environments.

Adaptive and Context-Aware Systems:

Another exciting area to explore is the development of adaptive systems that intelligently adjust processing parameters based on the specific contexts of live performances or recording sessions. These systems could implement advanced feedback loops, where continuous real-time analysis of audio input allows for dynamic recalibration of processing settings. For instance, during a live concert, the system could measure audience noise levels and energy, automatically fine-tuning reverb, equalization, and compression settings to ensure optimal audio quality in various acoustic environments. This capability would not only enhance the listening experience but also empower sound engineers to focus on creative aspects rather than technical adjustments.

Broader Genre Coverage and Larger Datasets:

Expanding the dataset utilized for training the models is crucial for improving performance across a wider array of musical genres. Future projects could endeavor to compile a more extensive database featuring diverse musical styles, such as jazz, hip-hop, classical, and traditional world music, along with a larger variety of samples to develop more robust and adaptable models. Additionally, investigating multilingual and cross-cultural musical datasets will not only enhance the system's adaptability but also enable it to cater to global music trends. By encompassing a richer array of cultural contexts and musical structures, the models can be designed to generalize better to real-world audio scenarios, thus increasing their versatility and effectiveness in music classification and processing tasks.

6. Conclusion

In this work, we showed how to combine machine learning with audio processing to create a system for classifying music genres and improving audio quality. We used a dataset of short audio tracks from Kaggle that covers multiple genres. Our feature extraction process captured important audio characteristics such as Mel-Frequency Cepstral Coefficients (MFCCs), Zero Crossing Rate (ZCR), and Spectral Centroid. We then used these features to train and test machine learning models like logistic regression and support vector machines (SVM), which gave us good results for classification.

We went further by integrating these machine learning features into an interactive user interface similar to a Digital Audio Workstation (DAW). This interface lets users record or load audio, view waveforms and spectrograms, and apply real-time audio effects based on the predicted genre. This system not only allows for quick testing and analysis but also acts as a proof-of-concept for advanced audio processing tools that can help sound engineers and producers.

Overall, this project shows how using data-driven methods can change music production. By combining strong signal processing techniques with smart automation, we create new chances for enhanced creativity and improved workflow in audio engineering.

7. References

1. Tzanetakis, G., & Cook, P. (2002). Musical Genre Classification of Audio Signals. *IEEE Transactions on Speech and Audio Processing*.

2. Aptly Lab, University of York, Department of Electrical Engineering and Computer Science. Retrieved from https://bil.eecs.yorku.ca/aptly-lab/

3. Music Genre Detection with Deep Learning. Retrieved from https://towardsdatascience.com/music-genre-detection-with-deep-learning-cf89e4cb2ecc

4. How to Start Implementing Machine Learning to Music. Retrieved from https://medium.com/towards-data-science/how-to-start-implementing-machine-learning-to-music-4bd2edccce1f

5. Music Production Glossary. (n.d.). What is Sound Engineering? Retrieved from https://musicproductionglossary.com/what-is-sound-engineering/

6. Kaggle. (n.d.). GTZAN Dataset for Music Genre Classification. Retrieved from https://www.kaggle.com/datasets/andradaolteanu/gtzan-dataset-music-genre-classification/data

7. Sigtia, S., Benetos, E., & Dixon, S. (2016). Automatic Music Transcription Using Deep Learning: Challenges and Future Directions. *IEEE Signal Processing Magazine*.

8. OpenAI. (2023). ChatGPT [Large Language Model]. Retrieved from https://openai.com/chatgpt

9. Chollet, F. (2018). Deep Learning with Python. Manning Publications.

10. McFee, B., Raffel, C., Liang, D., Ellis, D. P. W., McVicar, M., Battenberg, E., & Nieto, O. (2015). librosa: Audio and Music Signal Analysis in Python. In *Proceedings of the 14th Python in Science Conference* (pp. 18–24).

11. scikit-learn. (n.d.). Retrieved from https://scikit-learn.org/stable/index.html

12. Google. (n.d.). Gemini [Large Language Model]. Retrieved from https://gemini.google.com/app?hl=el