VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"Jnana Sangama", Belagavi : 590018

A Mini Project report on

GROOVEGURU – MUSIC GENRE CLASSIFICATION AND RECOMMENDATION SYSTEM

Submitted in partial fulfillment of the requirement for the award of the degree of

Bachelor of Engineering
in
Computer Science and Engineering

by

ADITI DAS	**1AY21CS014**
ANJALI K. S.	**1AY21CS029**
BHAVANA B. HALLIKERI	**1AY21CS040**
DEEPA TENDULKAR	**1AY21CS051**

Under the guidance of
Mr. M. GOWTHAM RAJ
Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

ACHARYA INSTITUTE OF TECHNOLOGY

(Affiliated to Visvesvaraya Technological University, Belgaum)

2023-2024

ACHARYA INSTITUTE OF TECHNOLOGY

Acharya Dr. Sarvepalli Radhakrishnan Road, Soladevanahalli, Bangalore – 560107
(Affiliated to Visvesvaraya Technological University, Belagavi)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Certificate

Certified that the Mini Project entitled **"GROOVEGURU–MUSIC GENRE CLASSIFICATION AND RECOMMENDATION SYSTEM"** is a bonafide work carried out by **Aditi Das (1AY21CS014), Anjali K. S. (1AY21CS029), Bhavana B. Hallikeri (1AY21CS040), Deepa Tendulkar (1AY21CS051)** in partial fulfillment for the award of degree of **Bachelor of Engineering in Computer Science & Engineering of Visvesvaraya Technological University**, Belagavi during the year **2023-2024.** It is certified that all corrections/ suggestions indicated for internal assessments have been incorporated in the Report deposited in the departmental library. The project report has been approved as it satisfies the academic requirements in respect of project prescribed for the **Bachelor of Engineering Degree**.

________________	__________________	__________________
Signature of Guide	Signature of HOD	Signature of Principal
Mr. M. Gowtham Raj	**Dr. Ajith Padyana**	**Dr. Rajeswari**
Assistant Professor	**Professor & HOD**	

<u>External Viva</u>

Name of the Examiners	**Signature with Date**
1. ______________________________________	______________________________________
2. ______________________________________	______________________________________

DECLARATION

We , Aditi Das (1AY21CS014), Anjali K. S. (1AY21CS029), Bhavana B. Hallikeri (1AY21CS040) and Deepa Tendulkar (1AY21CS051) students of B.E, Computer Science and Engineering, Acharya Institute of Technology, Bengaluru-560 107, hereby declare that the project entitled " **GrooveGuru – Music Genre Classification and Recommendation System** " is an authentic record of our own work carried out under the supervision and guidance of Mr. M. Gowtham Raj , Assistant Professor, Department of Computer Science and Engineering, Acharya Institute of Technology, Bengaluru. We have not submitted the matter embodied to any other University or Institution for the award of any other degree.

Date: 31/07/2024

Place: Bengaluru

ADITI DAS	1AY21CS014
ANJALI K. S.	1AY21CS029
BHAVANA B. HALLIKERI	1AY21CS040
DEEPA TENDULKAR	1AY21CS051

ACKNOWLEDGEMENT

We express our gratitude to our institution and management for providing us with good infrastructure, laboratory, facilities and inspiring staff, and whose gratitude was of immense help in completion of this report successfully.

We are deeply indebted to **Dr. Rajeswari** Principal, Acharya Institute of Technology, Bangalore, who has been a constant source of enthusiastic inspiration to steer us forward.

We heartily thank **Dr. Ajith Padyana**, Head of the Department, Department of Computer Science and Engineering, Acharya Institute of Technology Bangalore, for his valuable support and for rendering us resources for this project work.

We specially thank **Mr. M. Gowtham Raj,** Assistant Professor, Department of Computer Science and Engineering who guided us with valuable suggestions in completing this project at every stage.

Also, we wish to express deep sense of gratitude for project coordinator **Mr. Prashanth Kumar S. P.** , Assistant Professor, **Mr. M. Gowtham Raj** , Assistant Professor, Department of Computer Science and Engineering, Acharya Institute of Technology for his support and advice during the course of this Mini project.

We would like to express our sincere thanks and heartfelt gratitude to our beloved Parents, Respected Professors, Classmates, Friends, and juniors for their indispensable help at all times.

Last but not the least our respectful thanks to the Almighty.

ADITI DAS	(1AY21CS014)
ANJALI K. S.	(1AY21CS029)
BHAVANA B. HALLIKERI	(1AY21CS040)
DEEPA TENDULKAR	(1AY21CS051)

ABSTRACT

GrooveGuru is an advanced music genre classification and recommendation system utilizing Python libraries for audio processing and machine learning. Audio feature extraction is performed using Librosa, focusing on MFCCs. Machine learning models like Random Forest, Decision Tree, K-Nearest Neighbors, and Support Vector Machine classify music into predefined genres. The system accurately classifies music genres and offers personalized recommendations based on user preferences through a user-friendly interface. GrooveGuru provides an engaging platform for discovering and exploring music genres, enhancing user experience with personalized music recommendations.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1. PROBLEM DEFINITION:

Develop an advanced system, GrooveGuru, for music genre classification and recommendation, enhancing user experience in music discovery and organization.Implement state-of-the-art audio processing using Librosa to extract critical features from music files for analysis. By training SVM models on the extracted audio features, GrooveGuru can accurately classify music into various genres. This classification process not only aids in organizing users' music libraries but also forms the basis for the recommendation system. GrooveGuru's recommendation system operates by analyzing the music provided by the user for genre classification. Once the genre is identified, GrooveGuru utilizes this information to recommend additional tracks within the same genre. This approach ensures that recommendations are closely aligned with the user's current listening preferences, thereby enhancing the overall music discovery experience.

1.2. OVERVIEW OF THE TECHNICAL AREA:

The proposed system will leverage advanced audio processing and machine learning technologies. Key components include:

- **Audio Signal Processing:** Utilizing Librosa, a comprehensive Python library for audio and music analysis, GrooveGuru will extract critical audio features from music files. These features include but are not limited to spectral characteristics, tempo, rhythm patterns, chroma features, mel-frequency cepstral coefficients (MFCCs). These extracted features are essential for analyzing and categorizing music files, providing the foundational data necessary for accurate genre classification.
- **Machine Learning Algorithms:** GrooveGuru will implement robust machine learning models, with a particular focus on Support Vector Machines (SVMs), renowned for their efficacy in classification tasks. The SVM models will be trained on the extracted audio features to recognize and classify music genres with high precision. Additionally, the system may explore other algorithms, such as neural networks and ensemble methods, to further enhance classification accuracy and adaptability. The machine learning pipeline

will include data preprocessing, feature selection, model training, validation, and optimization to ensure the highest level of performance.

- **User Interface Development:** A crucial aspect of GrooveGuru is the design and development of an intuitive user interface. This interface will enable users to seamlessly upload music files, view genre classification results, and receive personalized music recommendations. The interface will be user-friendly, featuring streamlined navigation, responsive design, and visually appealing elements to ensure a positive user experience. It will also provide users with options to interact with their music library, explore recommended tracks, and offer feedback to continuously improve the recommendation system.

1.3. OVERVIEW OF EXISTING SYSTEM:

1.3.1. Existing Systems:

Streaming Platforms: Current streaming platforms, such as Spotify, Apple Music, and YouTube Music, employ sophisticated algorithms to recommend music to users. These algorithms primarily rely on user listening history, song metadata, and collaborative filtering techniques. By analyzing patterns in user behavior and preferences, these platforms suggest music that aligns with the user's past listening habits and similar users' preferences. However, these systems predominantly focus on metadata, such as artist, album, and genre tags, along with user-generated data, which may not always accurately represent the true nature of the music.

Music Libraries: Applications such as iTunes and Windows Media Player provide users with basic genre classification capabilities. These applications categorize music based on metadata embedded within the audio files, such as ID3 tags, which include information like genre, artist, and album. While these systems facilitate basic organization of music libraries, they are limited by the accuracy and completeness of the metadata provided by the music files. Consequently, the genre classification may not always reflect the actual characteristics of the music.

1.3.2. Drawbacks:

Limited Accuracy: Many existing music recommendation and classification systems suffer from limited accuracy due to their heavy reliance on metadata and user behavior. Metadata, which is often manually entered or sourced from various databases, can be inconsistent and subjective, leading to incorrect genre labels. Furthermore, user behavior-based recommendations may reinforce existing preferences rather than introducing users to new and diverse genres, thereby limiting the discovery potential.

Lack of Feature Extraction: A significant limitation of current systems is the absence of advanced audio feature extraction. By not leveraging sophisticated audio analysis techniques, such as those provided by tools like Librosa, these systems miss out on the nuanced and intricate patterns present in the audio signal itself. Advanced feature extraction can provide a more precise and objective basis for genre classification, capturing elements such as tempo, rhythm, timbre, and harmonic content, which are essential for accurate genre identification.

1.4. OVERVIEW OF PROPOSED SYSTEM:

1.4.1. Brief description of the proposed system:

The proposed system, GrooveGuru, is an advanced music genre classification and recommendation platform engineered to significantly enhance the user experience in music discovery and organization. Developed using Python, GrooveGuru harnesses state-of-the-art audio processing and machine learning techniques to perform accurate analysis and categorization of music files. This system is designed to overcome the limitations of existing music platforms by providing more precise genre classification and personalized music recommendations based on in-depth audio content analysis.

1.4.2. Audio Feature Extraction:

GrooveGuru utilizes Librosa, a powerful library for audio and music analysis, to extract critical audio features from music files. This process involves analyzing the actual content of the music rather than relying solely on metadata. Key features extracted include spectral characteristics, tempo, rhythm patterns, chroma features, mel-frequency cepstral coefficients (MFCCs), and zero-crossing rates. These features capture the intricate details and unique elements of each music file, providing a rich dataset for subsequent analysis.

1.4.3. Machine Learning Models:

The extracted audio features are fed into machine learning models that are trained to recognize patterns in the audio data. GrooveGuru employs Support Vector Machine (SVM) algorithms, known for their effectiveness in high-dimensional classification tasks. The SVM models are meticulously trained using labeled datasets to distinguish between various music genres accurately. By identifying distinct audio patterns and characteristics associated with each genre, the models enable precise genre classification.

1.4.4. Personalized Recommendations:

Building on the genre classification, GrooveGuru offers personalized music recommendations. The recommendation engine takes the classified genres and suggests additional tracks that align with the user's identified preferences. This is achieved by analyzing user feedback, listening history, and the popularity of tracks within each genre.

1.4.5. User Interface Development:

GrooveGuru features an intuitive user interface that facilitates easy interaction with the system. Users can upload music files, view classification results, and receive personalized recommendations seamlessly. The interface is designed to be user-friendly, with clear navigation and visually appealing elements to ensure a pleasant user experience.

1.4.6. Backend Infrastructure:

A robust backend infrastructure supports GrooveGuru, enabling efficient data storage, processing, and retrieval. This infrastructure ensures the system can handle large volumes of music files and perform real-time or near-real-time classification and recommendation. Integration with cloud services allows for scalable storage solutions and distributed computing resources, enhancing the system's performance and reliability.

CHAPTER 2
LITERATURE REVIEW

Table 2.1

Literature Survey

S. n	**Paper tittle & Publication Details**	**Name of the authors**	**Technical ideas / algorithms used in the paper & advantages**	**Shortfalls/disadvantages & Solution provided by the proposed system**
1	Enhancing Music Genre Classification through Multi-Algorithm Analysis and User-Friendly Visualization	Navin Kamuni, Dheerendra Panwar	**Naive Bayes:** A probabilistic classifier based on Bayes' Theorem, known for its simplicity and effectiveness. In a music genre classification, where the relationship between features can be complex and non linear, Naïve Bayes might not capture the intricacies of the data as effectively as other models like SVM.	**Inability to Access External Music Files:** The application was limited in its ability to access and process the external music files. **Limited to Linux Environ ment:** The application was initially developed on Linux, which may not be as widely used by the general public compared to other operating systems, potentially limiting the user base.
2	A Cascade – Hybrid Music Recommender System for mobile services	Aristomenis S. Lampropoulos et al	**Personality Diagnosis:** The second level of the system employs a collaborative filtering technique known as personality diagnosis. This approach predicts user preferences based on the	**Cold Start Problem:** While the system aims to address the cold start problem by using a combination of content - based and collaborative filtering techniques, it may still

	based on musical genre classification and personality diagnosis		ratings of other users with similar personality types. **Collaborative Filtering:** The system uses collaborative filtering to predict user preferences by comparing the active user's ratings with those of other users.	struggle with new users that have insufficient data for recommendations. **Dependency on User Ratings:** The personality diagnosis approach relies heavily on user ratings, which may not always be available or reliable. Users may not rate items consistently or may stop providing ratings altogether.
3	Automatic music genre classification based on musical instrument track separation	Aldona Rosner, Bozena Kostek	**Independent Component Analysis (ICA):** The application of ICA in drum beats (instrumental) separation demonstrates the paper's focus on advanced signal processing techniques to improve the classification and analysis of music genres.	**Instrument Influence:** The study explores the influence of specific instruments on genre classification, but it may not fully capture the complexity of musical compositions and the interplay of multiple instruments within a genre. The approach may perform differently across various music genres, particularly those with complex or overlapping sound compone nts. The separation quality might vary depending on the

				musical style and the nature of the audio content.
4	Music genre classification based on fusing audio and lyric information	You Li et al	**Convolutional Neural Networks (CNN):** The document describes the use of CNN for learning higher-order features based on the generated mel-frequency spectrogram in the audio sub model. CNNs are commonly used for analyzing visual imagery but have also been successfully applied to audio data.	**Experimental Settings:** The document mentions that 30 seconds of audio data were directly intercepted for feature extraction. This approach might not capture the full diversity of audio characteristics within a given song, potentially limiting the richness of the audio features used for classification.
5	A Survey of Music Recommendation Systems and Future Perspectives	Yading Song et al	**Hybrid Models:** These combine collaborative filtering and content-based approaches to leverage the strengths of both methods, potentially leading to improved recommendation performance.	**Popularity Bias:** The system tends to recommend popular music more frequently, potentially neglecting lesser-known or niche music.

CHAPTER 3

REQUIREMENT SPECIFICATION

3.1. FUNCTIONAL REQUIREMENTS:

3.1.1. Music File Upload and Processing:

- The system exclusively handles WAV files and provides a user-friendly interface for uploads through drag-and-drop or file picker options.
- Upon upload, the files are validated to ensure they meet the WAV format and size requirements, with immediate feedback given for any issues.
- After validation, the files undergo processing to prepare for feature extraction. This includes normalization to adjust the audio signal to a standard volume level, resampling to ensure a uniform sample rate, and preprocessing such as conversions or trimming to ensure the audio data is in optimal condition for analysis.
- This thorough processing ensures the WAV files are ready for accurate feature extraction, setting the stage for precise genre classification.

3.1.2. Audio Feature Extraction:

- After processing the music files, the system extracts detailed audio features essential for genre classification by breaking the audio into smaller segments for analysis.
- Key spectral features include Mel-frequency cepstral coefficients (MFCCs) that capture timbral qualities, along with spectral centroid and spectral bandwidth for insights into the sound’s brightness and texture.
- Temporal features such as root mean square (RMS) energy measure loudness, while the zero-crossing rate indicates the signal’s noisiness.
- Rhythmic features like tempo, beat, and onset information help understand the music’s rhythmic structure.
- Techniques like the short-time Fourier transform (STFT) are used to analyze the frequency content of the audio signal.
- The extracted features are then organized into a dataset for machine learning models, ensuring an accurate representation of the music’s characteristics for genre classification.

3.1.3. Machine Learning Model Implementation Importance:

In a music genre classification system, different machine learning models play distinct roles:

- **Support Vector Machines (SVM)**: Primarily used for classifying music genres, SVM excels at finding the optimal hyperplane to separate genres based on audio features like rhythm and pitch. This allows SVM to handle complex and high-dimensional data, ensuring accurate and reliable classification.
- **Decision Trees**: These are used to understand and visualize the contribution of various features to genre classification. By splitting the feature space into regions based on feature values, Decision Trees provide clear rules for classification and easy interpretation of how different features influence the results. However, they can be prone to overfitting.
- **Random Forests**: Enhance Decision Trees by aggregating multiple trees to improve classification accuracy and reduce overfitting. They assess feature importance more comprehensively by evaluating the impact of each feature across many trees, offering a stable measure of feature relevance. This helps refine the feature set and improve model performance.

3.1.4. Personalized Recommendation System Importance:

- A personalized recommendation system enhances the user experience in music applications by tailoring suggestions to individual tastes.
- By analyzing listening history, favorite genres, and liked artists, the system provides recommendations that match each user's preferences, boosting engagement and satisfaction.
- Users enjoy a more convenient music discovery process, finding new tracks effortlessly. The system also introduces them to new artists and genres that complement their tastes, enriching their music experience.
- Personalized recommendations foster user retention and loyalty. When users feel understood and catered to, they are more likely to remain active and engaged, leading to greater satisfaction and long-term use of the application.

3.1.5. User Interface Design Importance:

- A well-designed, responsive, and intuitive UI ensures seamless user interaction, enhancing the overall experience.

- Ease of navigation is a key benefit. By logically organizing features, users can easily access playlists, search for songs, and adjust settings, minimizing frustration and creating a smoother experience.
- Effective result visualization is another critical aspect. Clear and visually appealing presentations of music recommendations and search results help users quickly interpret and engage with the content.
- An intuitive UI prominently showcases personalized music recommendations, making it easy for users to discover new music that matches their preferences.
- A responsive UI adapts to various devices and screen sizes, ensuring a consistent user experience across smartphones, tablets, and desktops. This adaptability makes the application more accessible and convenient, enhancing the overall user experience regardless of the device being used.

3.2. NON-FUNCTIONAL REQUIREMENTS:

Non-functional requirements are critical to the successful implementation and operation of GrooveGuru, ensuring the system meets performance, security, usability, and scalability expectations beyond basic functional capabilities.

Overview of the importance of all the non-functional requirements:

3.2.1. Performance:

- Performance is critical for any music application, ensuring smooth and efficient operation. Effective performance guarantees prompt responses to user interactions, efficient processing of large audio files, and real-time genre classifications and recommendations.
- A high-performing system processes user inputs, such as play, pause, or search commands, quickly, providing a seamless user experience. Minimizing delays or lags helps maintain user engagement and satisfaction.
- Efficient processing of large audio files is essential. The system must handle substantial files without compromising performance, ensuring smooth playback and minimal load times, even for high-resolution files.
- Real-time genre classification and recommendations are crucial for user satisfaction. The system should deliver accurate results and personalized suggestions promptly, enhancing the user experience with relevant and engaging content without delays.

3.2.2. Usability:

- Usability is crucial for enhancing the user experience in a music application by ensuring the interface is intuitive, easy to navigate, and accessible.
- An intuitive interface uses familiar design elements and clear instructions, guiding users seamlessly through tasks. This reduces the learning curve and makes interactions more natural and enjoyable.
- Clear navigation is essential for efficient task completion. A well-organized menu structure with logical categories and easily identifiable buttons helps users effortlessly upload music, view results, and explore recommendations.
- Accessibility features ensure the application is usable by individuals with diverse needs. Options such as adjustable text sizes, screen reader compatibility, and keyboard navigation support make the application more inclusive, allowing users with different abilities to interact with the system effectively.

3.2.3. Scalability:

- Scalability is essential for a music application, ensuring it can handle growth and increasing demands without compromising performance.
- Handling rising user traffic requires the system to manage more simultaneous interactions and requests while remaining responsive. This involves optimizing server resources, load balancing, and using efficient algorithms.
- Managing large volumes of music data is crucial as the application deals with extensive audio files and metadata. Scalable systems use robust database management, efficient storage solutions, and scalable cloud infrastructure to handle growing data needs.
- Meeting additional computational demands involves expanding resources for tasks like real-time genre classification and recommendation generation, ensuring consistent performance and reliability as the application evolves.

3.2.4. Reliability:

- Reliability is crucial for a music application, ensuring continuous operation, graceful error handling, and data consistency.
- A reliable system operates seamlessly without interruptions by using redundant systems, failover mechanisms, and regular maintenance to prevent downtime and ensure accessibility.

- Effective error handling minimizes the impact of issues by detecting, logging, and addressing errors with informative messages and prompt resolution, maintaining user trust.
- Maintaining data consistency and integrity involves accurate and synchronized data across user interactions.

3.3. SOFTWARE REQUIREMENTS:

3.3.1. Google Collab:

Google Collab is a powerful tool for machine learning and data science, offering collaborative coding, efficient model prototyping, and cloud-based computation.

- **Collaborative Coding**: Colab enables real-time collaboration, allowing multiple users to work on the same notebook simultaneously and integrates with Google Drive for easy file management.

- **Prototyping Models**: Its interactive environment is ideal for developing and experimenting with machine learning models, allowing users to write code, test algorithms, and visualize results directly in the notebook.

- **Cloud-Based Computation**: Colab provides access to Google's cloud infrastructure, including GPUs and TPUs, enhancing the speed and efficiency of training and evaluating models without relying on local hardware.

3.3.2. Python:

Python is a key programming language used for backend functionalities, machine learning, and audio processing.

- **Backend Functionalities**: Python's versatility and extensive libraries make it ideal for server-side operations, database interactions, and application logic.

- **Machine Learning Algorithms**: Python is prominent in machine learning, supported by libraries like TensorFlow, Keras, and scikit-learn, which aid in developing, training, and evaluating models.

- **Audio Processing**: Python supports audio processing with libraries such as Librosa and PyDub, essential for tasks like genre classification and feature extraction.

3.3.3. Interface:

Django is a powerful framework for developing the frontend UI of applications, handling tasks like music uploads, result viewing, and recommendations.

- **Designing the Frontend UI**: Django's tools and templates create user-friendly, responsive interfaces, facilitating easy music file uploads.
- **User Interaction**: Django integrates with HTML, CSS, and JavaScript for dynamic content rendering and navigation, allowing seamless viewing of results and recommendations.
- **Receiving Recommendations**: It connects backend algorithms with the frontend, delivering personalized recommendations and real-time data based on user interactions.

3.3.4. Librosa:

Librosa is a Python library essential for music and audio analysis, facilitating audio file handling and feature extraction.

- **Loading Audio Files**: Librosa simplifies importing audio files and converting them into arrays for easy processing.
- **Extracting Key Features**: Librosa excels in extracting features crucial for genre classification:
  - **MFCCs**: Capture timbral qualities and spectral characteristics of audio.
  - **Chroma Features**: Analyze harmonic content and pitch.
  - **Mel-Spectrograms**: Provide a detailed time-frequency representation of the audio.

3.4. HARDWARE REQUIREMENTS:

3.4.1. Processor (CPU):

The Central Processing Unit (CPU) is essential for a music application's computational tasks, including data processing, machine learning model training, and real-time inference.

- **Data Processing**: The CPU manages the computations needed to process audio data, including loading, preprocessing, and transforming files for analysis.
- **Machine Learning Model Training**: It executes complex algorithms to train and optimize models, crucial for accurate genre classification.
- **Real-Time Inference**: The CPU applies trained models to new audio samples for genre classification and recommendations, affecting the speed and responsiveness of these tasks.

3.4.2. RAM (Memory):

RAM (Random Access Memory) is essential for the smooth operation of a music application, impacting tasks like data handling and multitasking.

- **Basic Tasks**: 4GB of RAM supports tasks such as loading and processing audio files, running machine learning models, and managing routine operations for small to moderate datasets.
- **Performance Enhancement**: Increasing RAM improves performance by allowing the system to handle larger datasets and complex operations more efficiently.
- **Simultaneous Execution**: Adequate RAM enables smooth execution of multiple processes, such as audio processing and model inference, without significant performance degradation.

3.4.3. Graphics Processing Unit (GPU):

The Graphics Processing Unit (GPU) is crucial for accelerating computational tasks in a music application, especially for machine learning and data analysis.

- **Accelerating Computation-Intensive Tasks**: GPUs excel at parallel processing, speeding up complex tasks like training neural networks by handling numerous calculations simultaneously.

- **Enhancing Speed and Performance**: The GPU’s parallel architecture significantly speeds up model training and data processing, improving performance and enabling faster iterations.

- **Beneficial for Large-Scale Data Analysis**: GPUs efficiently process large datasets and analyze audio features, enhancing overall performance and responsiveness in music classification tasks.

3.4.4. Speaker:

The speaker is essential for providing auditory feedback in a music application, allowing users to listen to classified tracks immediately.

- **Auditory Feedback for Verification**: The speaker enables users to hear and verify the genre classification of music tracks in real-time, ensuring the results meet their expectations.
- **Enhancing User Experience**: By allowing users to enjoy and explore classified tracks directly, the speaker makes the interaction more engaging and immersive.

- **Direct Interaction with Results**: The speaker creates a seamless connection between classification results and the user's auditory experience, enhancing the overall enjoyment and intuition of the application.

3.5. TECHNOLOGIES USED:

3.5.1. Python:

Python is the primary language for GrooveGuru, selected for its extensive libraries and frameworks essential for machine learning, data processing, and audio analysis. It supports advanced tools like TensorFlow, Keras, and scikit-learn for model development, and Librosa for audio feature extraction. Python's simplicity and readability streamline development and maintenance. Its versatility and strong support in the data science community ensure GrooveGuru remains adaptable and capable of advanced analytics.

3.5.2. Librosa:

Librosa is a Python library integral to GrooveGuru for audio and music analysis. It extracts essential features for genre classification, such as Mel-Frequency Cepstral Coefficients (MFCCs), chroma features, and mel-spectrograms. These tools facilitate effective signal processing and feature extraction, crucial for accurate music classification. By streamlining audio analysis, Librosa enhances the application's functionality and supports precise genre identification.

3.5.3. Scikit-learn:

Scikit-learn is a key Python library for machine learning in GrooveGuru. It offers efficient implementations of algorithms like Support Vector Machines (SVM) for genre classification. The library provides a user-friendly interface for training, validating, and predicting with models, simplifying the development process. Its comprehensive tools and features support accurate and scalable genre classification.

3.5.4. NumPy:

NumPy is a crucial Python library for numerical computations in GrooveGuru, handling essential array and matrix operations for audio processing and machine learning. Its efficient performance and ease of use enable complex calculations and data manipulation, ensuring swift and accurate handling of large datasets and enhancing the overall functionality and performance of the application.

3.5.5. Pandas:

Pandas is a key Python library for data manipulation and analysis in GrooveGuru. It handles tasks like managing, cleaning, and preprocessing datasets, which are essential for preparing data for machine learning models. With its powerful data structures like DataFrames and Series, Pandas enables efficient data manipulation and organization. This ensures that data is well-structured and ready for accurate analysis and classification.

3.5.6. Matplotlib and Seaborn:

Matplotlib and Seaborn are essential Python libraries for data visualization in GrooveGuru. Matplotlib offers a flexible framework for creating various plots and charts, while Seaborn enhances these with more aesthetically pleasing visualizations. These tools are crucial for visualizing audio features, model performance, and data insights, aiding in understanding patterns, assessing accuracy, and identifying improvements. By presenting data clearly, they support effective decision-making and optimization in the project.

3.5.7. Django:

Django is a key web framework for GrooveGuru, providing the infrastructure to develop its web interface. It facilitates user interaction with the music classification and recommendation system, offering a secure and maintainable platform. Django’s built-in tools for authentication, database management, and URL routing streamline development, ensuring high performance and security. This allows the team to focus on enhancing functionality and user experience while benefiting from Django’s stability and scalability.

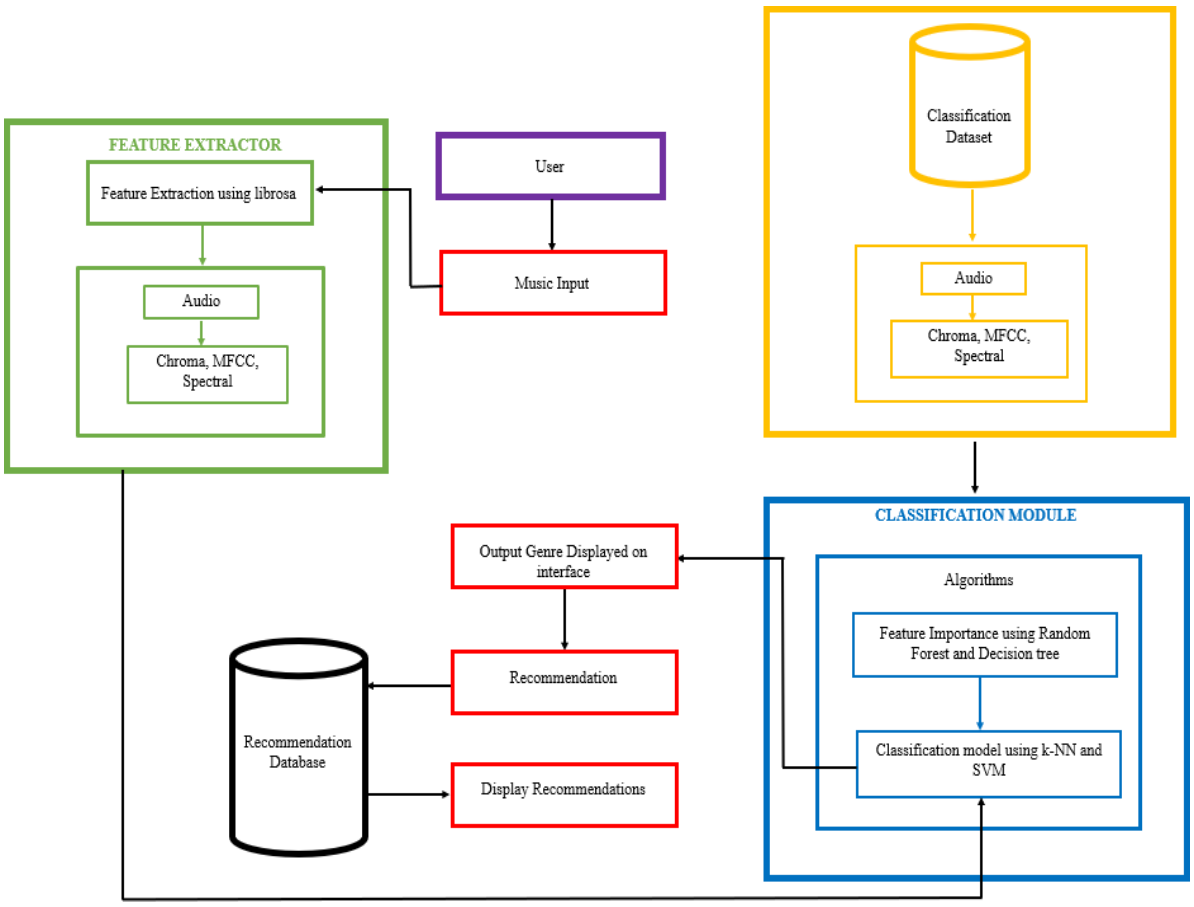
3.5.8. Git and GitHub:

Git is essential for version control in the GrooveGuru project, enabling efficient collaboration, code tracking, and management of project versions. GitHub complements Git by hosting repositories and providing tools for collaborative development, issue tracking, and code review. This integration enhances team coordination, streamlines development, and ensures effective management of code changes.

CHAPTER 4

SYSTEM DESIGN

4.1 SYSTEM ARCHITECTURE:



Legend:

🟢 Feature Extractor	🟡 Classification Dataset	🔵 Classification Module
🔴 Website Interface	🟣 User	⚫ Recommendation Database

Fig 4.1 System Architecture

The provided diagram outlines a system designed for music genre classification and recommendation. The system is divided into three main components: the Feature Extractor, the Classification Module, and the Recommendation Module, each responsible for a specific set of tasks to ultimately provide music genre classification and relevant recommendations to the user.

4.2 DATA FLOW DIAGRAM

The flowchart provided depicts a system for classifying music genres and recommending songs based on these classifications. The process involves several key stages, from data preparation to user interaction, using various machine learning techniques and data processing methods.

1. Data Collection and Preparation:

The first step involves collecting a dataset consisting of audio files and their corresponding features. These features could include various audio characteristics like mfcc, chroma_sftt and spectral. The dataset serves as the foundation for training the model.

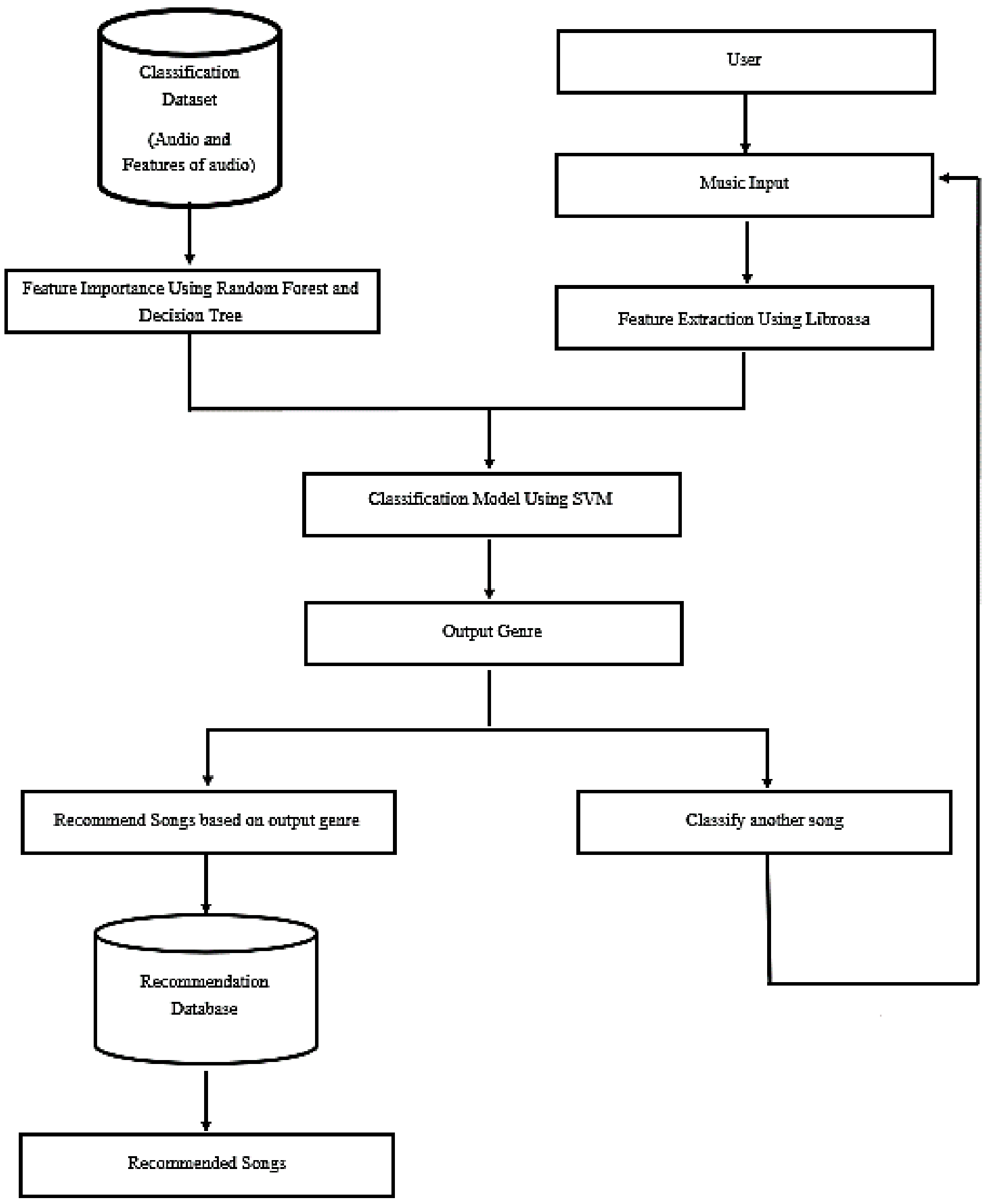

Classification Dataset (Audio and Features of audio)
User
Music Input
Feature Importance Using Random Forest and Decision Tree
Feature Extraction Using Libroasa
Classification Model Using SVM
Output Genre
Recommend Songs based on output genre
Classify another song
Recommendation Database
Recommended Songs

Fig 4.2 Data Flow Diagram

2. Feature Importance Extraction:

Before feeding the data into a machine learning model, it's essential to determine the most relevant features that contribute to genre classification. This is achieved using algorithms like Random Forest and Decision Trees. These algorithms help in identifying which features are most important in distinguishing between different genres, allowing for more efficient and accurate model training.

3. User Interaction and Music Input:

The system includes a user interface where users can input a music track they want to classify. This input can be an audio file or a snippet of a song. The system then processes this input to extract the necessary features for classification.

4. Feature Extraction Using Librosa:

For the input music, features are extracted using the Librosa library, a popular Python package for music and audio analysis. Librosa helps in extracting various features like Mel-frequency cepstral coefficients (MFCCs), chroma features, spectral contrast, and more. These features are crucial for understanding the characteristics of the music, which are then used for genre classification.

5. Classification Model Using SVM:

The extracted features are then fed into a Support Vector Machine (SVM) model, which has been trained on the pre-collected dataset. SVM is a robust classification algorithm that finds the optimal hyperplane to separate different classes, in this case, different music genres. The SVM model processes the features and predicts the genre of the input music.

6. Output Genre:

Once the SVM model has processed the input features, it outputs the predicted genre. This genre represents the classification result, indicating the most likely category the input music belongs to, based on the features extracted and the model's training.

7. Recommendation System:

The predicted genre is then used to query a recommendation database. This database contains a curated list of songs categorized by genre. The system retrieves songs from this database that match the predicted genre, providing the user with a list of recommendations. The system retrieves songs from this database that match the predicted genre, providing the user with a list of recommendations. These recommendations are then presented to the user through an intuitive and user-friendly interface, enhancing their music discovery experience.

8. Recommendation Database:

The recommendation database is a critical component, storing songs along with their genre information. It enables the system to provide relevant song suggestions based on the output genre, enhancing the user's experience by offering similar music.

9. Recommended Songs:

The system then presents the recommended songs to the user, allowing them to explore new music that matches their genre preferences. This feature not only enhances user engagement but also helps users discover new songs and artists.

10. Option to Classify Another Song:

The system offers the user an option to classify another song. This loop allows users to continually interact with the system, inputting new songs for classification and receiving genre-based recommendations, thereby making the system more interactive and user-friendly.

In summary, this system provides an integrated approach to music genre classification and song recommendation using machine learning and data processing techniques. It offers a seamless user experience from music input to receiving genre-based song recommendations.

4.3. MODULE DESCRIPTIONS AND DESIGN

4.3.1 Feature Extraction Module

The Feature Extractor is a crucial component in the music genre classification and recommendation system. It serves as the initial stage in the data processing pipeline, where raw audio input from the user is analyzed and transformed into a set of representative features. These features encapsulate essential characteristics of the audio signal, enabling the subsequent classification and recommendation processes.

Purpose:

The primary purpose of the Feature Extractor is to convert the raw audio data into a structured format that can be effectively used for machine learning tasks. By extracting relevant features, this module helps in capturing the unique attributes of different music genres, such as rhythm, harmony, and timbre. These features are instrumental in distinguishing one genre from another and are essential for building an accurate classification model. Additionally, the Feature Extractor aims to reduce the dimensionality of the audio data, making the machine learning process more efficient. It also ensures consistency in feature representation, which is crucial for the model's performance and generalization across diverse audio samples.

Tools: Librosa

Librosa is an open-source Python library for analyzing and processing audio signals. It is widely used in the field of music information retrieval (MIR) due to its robust feature extraction capabilities. In this project, librosa is employed to extract various audio features, including:

- **Mel-frequency cepstral coefficients (MFCCs):** MFCCs are a representation of the short-term power spectrum of a sound, capturing the timbral texture of the audio. They are derived from the Fourier Transform of the audio signal and are widely used in audio processing tasks because they closely resemble the human auditory system's perception of sound.
- **Chroma Features:** These features capture the harmonic and melodic characteristics of music by mapping the energy distribution across the 12 different pitch classes (Chroma) of the musical scale. They are useful for identifying musical chords and harmonic content.
- **Spectral Features:** Spectral features include various attributes like spectral centroid, bandwidth, contrast, and roll-off, which describe the shape and nature of the audio spectrum. They provide information about the brightness, noisiness, and tonal quality of the audio signal.

The process of feature extraction using librosa involves several steps:

- **Loading the Audio File:** The raw audio file is loaded into the system, converting it into a waveform representation.
- **Preprocessing:** The audio signal may undergo preprocessing steps, such as resampling, normalization, and noise reduction, to enhance the quality of the extracted features.
- **Feature Extraction:** Using librosa, specific features like MFCCs, chroma, and spectral properties are computed from the audio signal. These features are typically extracted over short frames of the audio, capturing both time-varying and static aspects of the signal.
- **Aggregation:** The frame-wise features are aggregated (e.g., by calculating the mean and standard deviation) to produce a fixed-size feature vector representing the entire audio clip.

4.3.2 Classification Module

The Classification Module is a critical component of your music genre classification and recommendation system. It leverages various machine learning algorithms to analyze the extracted features and classify the music into predefined genres. This module is responsible for both identifying the most relevant features for classification and using those features to accurately categorize the music.

Purpose

The primary purpose of the Classification Module is to classify the genre of a given music piece based on the extracted audio features. This involves two main processes:

- **Feature Importance Assessment**: Determining which features are most influential in distinguishing different genres.
- **Genre Classification**: Using machine learning models to predict the genre based on the significant features.

Algorithms and Techniques

1. Random Forest

- Random Forest is an ensemble learning method that operates by constructing multiple decision trees during training and outputting the mode of the classes for classification tasks. It is particularly useful for assessing feature importance, as it provides a measure of the significance of each feature in the prediction process.
- In the context of genre classification, Random Forest can help identify which audio features (e.g., MFCCs, chroma, spectral features) are most critical for differentiating genres. This is done by measuring the decrease in node impurity (e.g., Gini impurity or entropy) caused by each feature.

2. Decision Tree

- Decision Tree is a supervised learning algorithm that is used for both classification and regression tasks. It works by splitting the dataset into smaller subsets based on the value of input features, creating a tree-like model of decisions. Each internal node represents a decision based on a feature, and each leaf node represents an outcome.
- In the context of genre classification, Decision Trees can help identify the most significant audio features (e.g., MFCCs, chroma, spectral features) by measuring the decrease in impurity (such as Gini impurity or entropy) that results from each split. This impurity reduction indicates how well a feature separates the data into different genres.

The features that lead to the most significant reductions in impurity are considered the most important for classification.

3. Support Vector Machine (SVM)

- SVM is a powerful classification algorithm that finds the hyperplane that best separates the data into different classes. It is chosen for its effectiveness in high-dimensional spaces and its ability to handle situations where the data is not linearly separable by using kernel functions.
- SVM is implemented by transforming the input features into a higher-dimensional space using a kernel function (e.g., linear, polynomial, radial basis function). It then finds the optimal hyperplane that maximizes the margin between different genre classes. The use of SVM in genre classification ensures robust and accurate predictions, especially when the data has complex patterns.

4. k-Nearest Neighbours (k-NN)

- k-Nearest Neighbours (k-NN) is a simple, non-parametric classification algorithm that classifies a data point based on how its neighbours are classified. It is particularly useful in scenarios where the decision boundary is irregular and can effectively handle multi-class classification problems. k-NN is chosen for its simplicity and effectiveness in certain cases where a clear, geometric understanding of the data is beneficial.
- The k-NN algorithm classifies an input data point by identifying the ‘k’ closest training examples in the feature space. The algorithm calculates the distance between the input data point and all other points in the training set using a distance metric (commonly Euclidean distance). It then selects the ‘k’ nearest data points and assigns the class label that is most common among these neighbours to the input point. The choice of 'k' (the number of neighbours) and the distance metric are critical in determining the performance of the k-NN classifier. Additionally, k-NN does not make any assumptions about the underlying data distribution, making it a versatile tool for classification.

4.3.3 Recommendation Module

Purpose

The Recommendation Module's primary purpose is to generate a list of music recommendations for the user based on the genre classified by the Classification Module.

Processes:

1. **Receive Classified Genre**:
    - **Input**: The classified genre from the Classification Module.

- **Process**: The module receives the classified genre and uses this information to start the recommendation process.

2. **Query Recommendation Database**:
   - **Purpose**: To find songs within the same genre that are similar to the input song.
   - **Process**: Songs with similar genres are fetched from Recommendation Database.
3. **Output Recommendations**:
   - **Purpose**: To prepare the final list of recommendations for display on the user interface.
   - **Output**: A formatted list of recommended songs ready to be displayed to the user along with youtube links of songs.

4.3.4 User Interface:

The User Interface (UI) of a music genre classification and recommendation system plays a crucial role in providing users with a seamless and intuitive experience. It serves as the bridge between the users and the system's underlying functionalities, allowing users to interact with the system easily and efficiently.

Purpose

The primary purpose of the User Interface is to:

1. **Facilitate User Interaction**: Enable users to interact with the system by uploading music files, viewing genre classifications, and receiving music recommendations.
2. **Enhance Usability**: Provide a user-friendly and intuitive layout that simplifies the navigation and usage of the system's features.
3. **Deliver Feedback**: Display classification results, feature importance, and other relevant information to the users in a clear and concise manner.

Design Considerations

1. **User-Centric Design**: The UI should be designed with the user in mind, focusing on ease of use, accessibility, and a visually appealing layout. The design should cater to both technical and non-technical users, ensuring that the interface is intuitive and requires minimal learning.
2. **Responsive Design**: The interface should be responsive and adaptable to different devices and screen sizes, ensuring a consistent user experience across desktops, tablets, and mobile devices.

3. **Tools and Technologies**:
   - **Django**: A high-level Python web framework used for developing the web application. Django simplifies the development process with its built-in features for handling database interactions, user authentication, and template rendering.
   - **HTML/CSS**: Standard web technologies for structuring and styling the user interface.
   - **Bootstrap**: A popular front-end framework used to design responsive and mobile-first web pages.

4.4 UML DIAGRAMS

4.4.1 UML CLASS DIAGRAM:

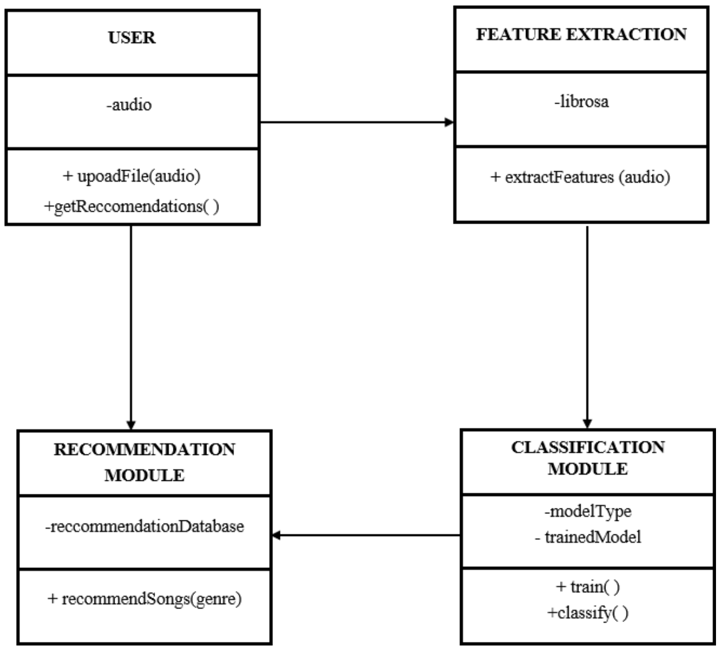


Fig 4.3 UML Class Diagram

This diagram illustrates the flow and interaction between various components in a music recommendation system. It starts with the USER module, which allows users to upload an audio file using the uploadFile(audio) method. Once the audio file is uploaded, the user can request song recommendations through the getRecommendations() method. The uploaded audio is then processed by the feature extraction module, which uses the Librosa library to extract audio features. The method extractFeatures(audio) is responsible for analyzing the audio file and extracting relevant features necessary for classification.

These extracted features are then passed to the classification module, which includes a trained model and information about the model type. The classification module provides methods for training (train()) and classifying (classify()) the audio features. Once the audio is classified, the results are sent to the recommendation module, which maintains a recommendation database. The recommendation module uses the recommendSongs(genre) method to provide song recommendations based on the genre or other classification results. This integrated process enables the system to deliver personalized music recommendations to the user based on the uploaded audio file's characteristics.

4.4.2. UML Sequence Diagram

The sequence diagram illustrates the interaction between the user and the system components: Feature Extractor, Classification Module, and Recommendation System. The process begins with the user uploading an audio file through the uploadFile(audio) method, which is sent to the Feature Extractor. The Feature Extractor then processes the audio to extract relevant features using the extractFeatures(audio) method. These extracted features are passed to the Classification Module, which determines the genre of the music by invoking the classify() method.

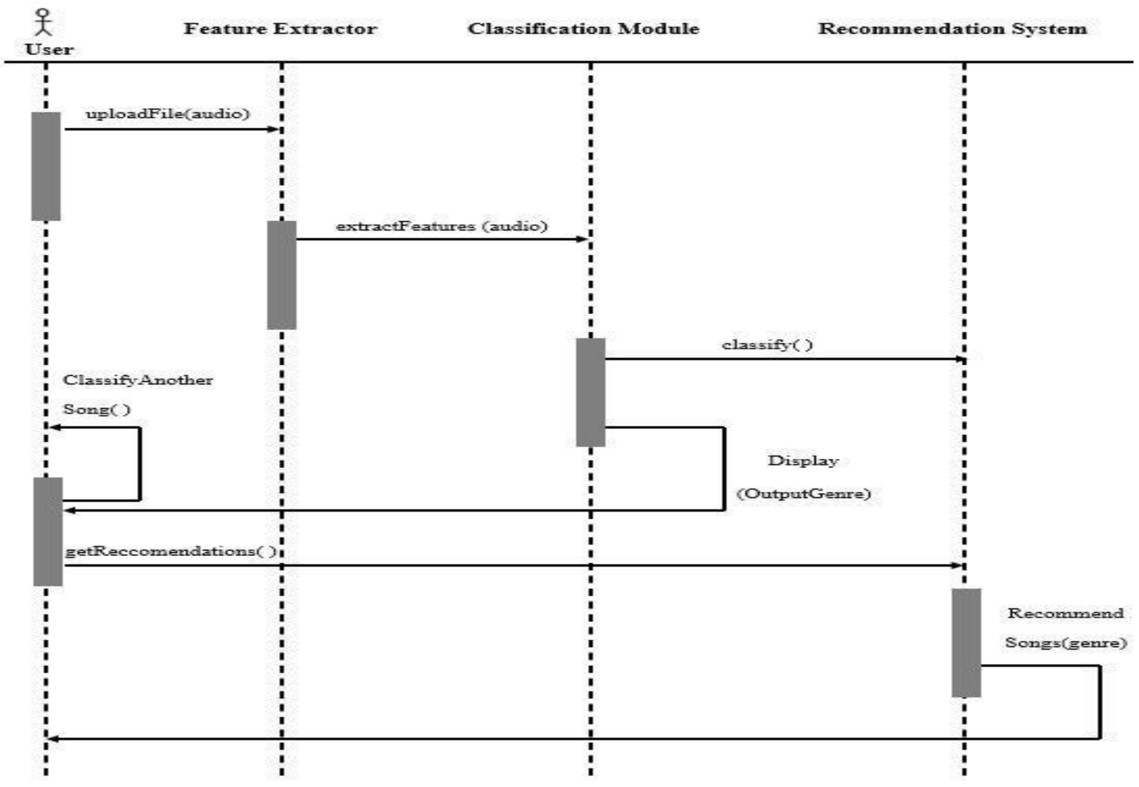


Fig 4.4 UML Sequence Diagram

4.4.3. UML Use Case Diagram

The diagram represents a system designed to classify music and provide recommendations based on the user's uploaded audio files. The process begins with the user uploading a .wav audio file, from which the system extracts features using the librosa library, a popular Python package for music and audio analysis. These features are crucial for analyzing the content and characteristics of the music. Once the features are extracted, the system classifies the music into a specific genre, which is then displayed to the user. This classification helps the user understand the type of music they have uploaded and lays the foundation for the recommendation process.

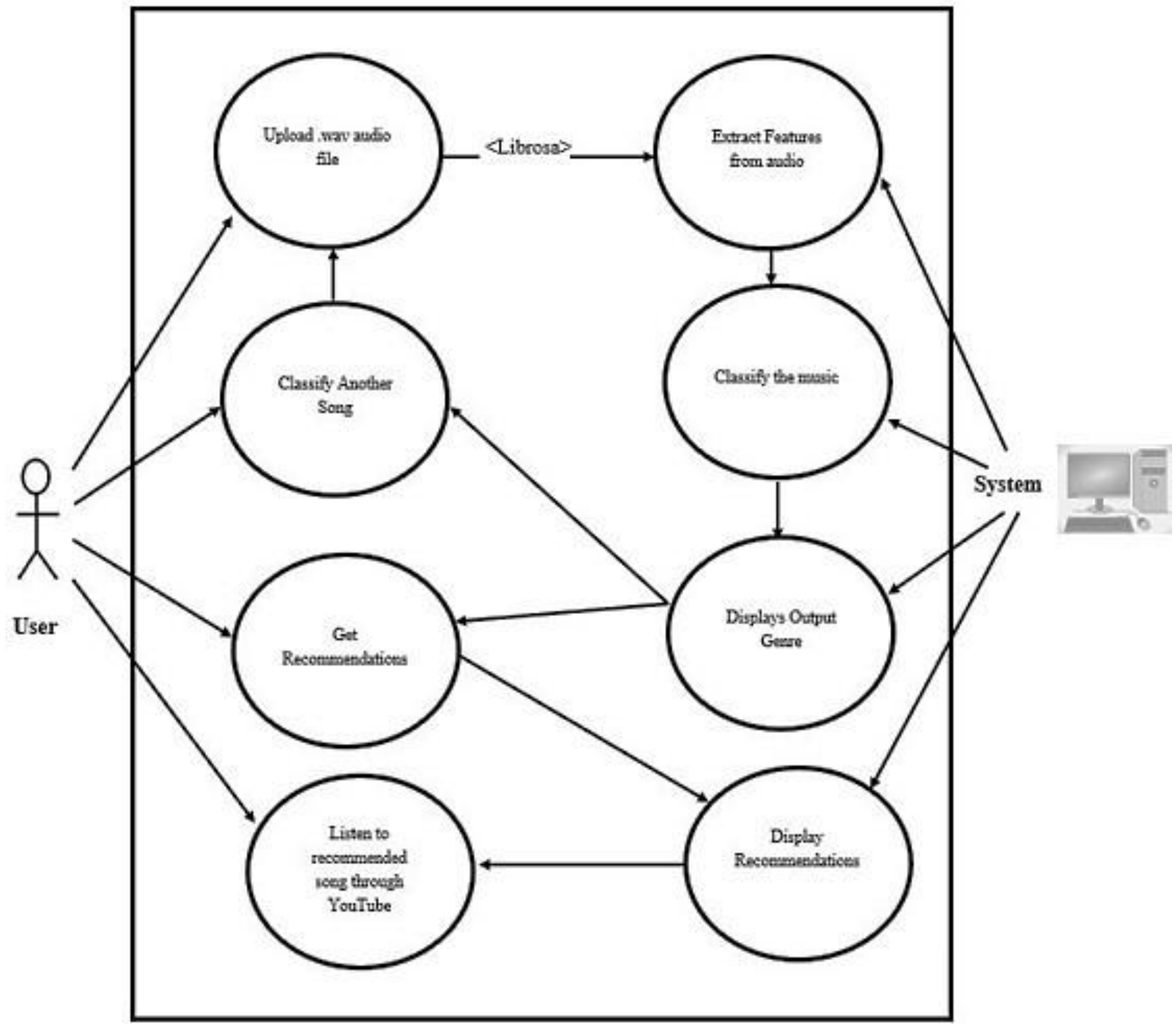


Fig 4.5 UML Use Case Diagram

Following the classification, users are presented with several engaging options: they can classify another song, request personalized music recommendations, or listen to the recommended tracks. If they opt for recommendations, the system utilizes the identified genre and detailed features extracted from the music to generate a curated list of songs that align with their preferences. This involves analyzing not just the genre, but also other musical elements like tempo, rhythm, and timbre, providing a more nuanced set of suggestions.

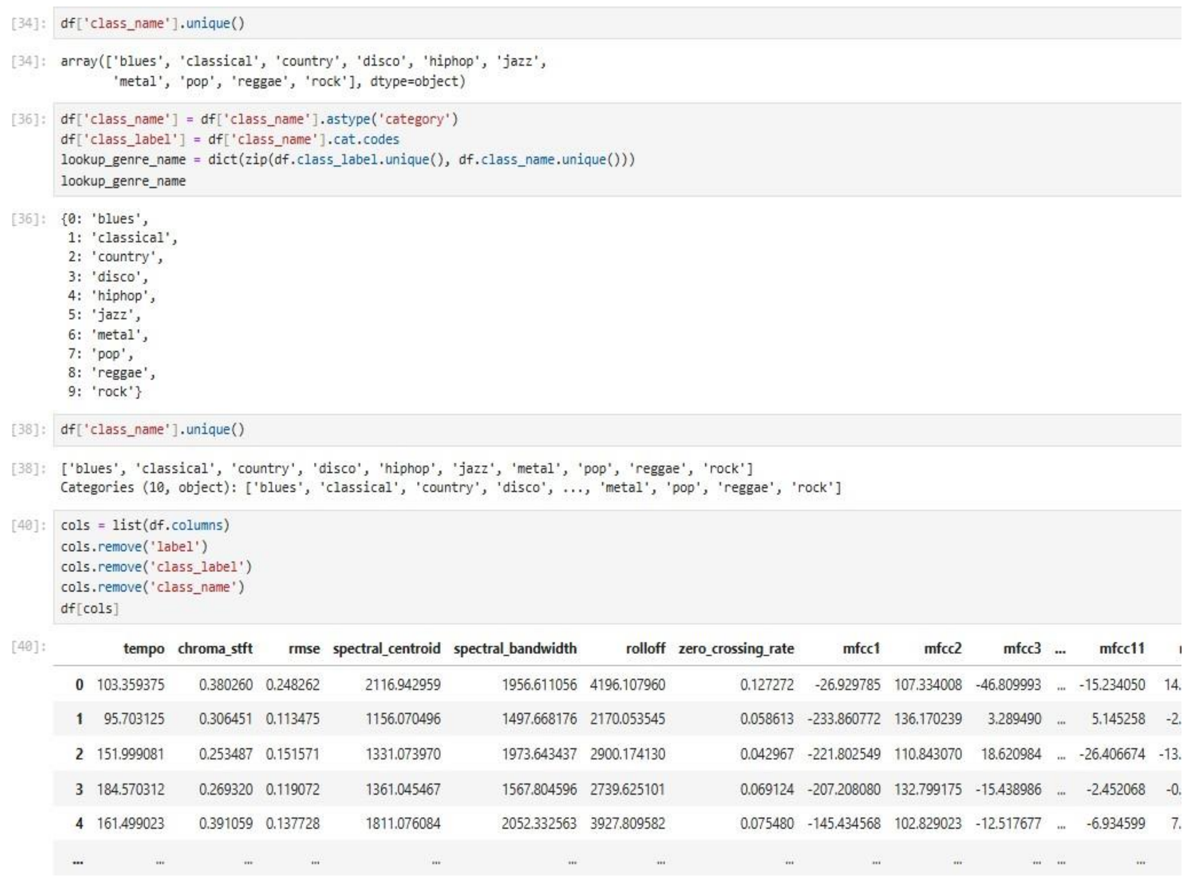
Once the recommendations are prepared, they are displayed through an intuitive and visually appealing interface. Users can then explore these new tracks and artists, with options to listen to them directly through integrated platforms like YouTube. The system may include direct playback options or provide easy access links, ensuring a smooth transition from the application to the music content.

Additionally, the recommendation system continuously evolves based on user interactions and feedback. As users engage with the suggested music, the system refines its algorithms and updates its recommendations to better match individual tastes and emerging trends. This adaptive approach not only keeps the music experience fresh but also enhances user satisfaction by aligning more closely with their evolving preferences.

The entire user journey, from initial classification to exploring recommendations, is illustrated in a comprehensive diagram. This visual representation highlights the seamless integration of user actions with system functionalities, providing a clear overview of how the application supports an engaging and intuitive music discovery experience. The diagram underscores the system’s capability to deliver personalized, relevant content and maintain a high level of user engagement and satisfaction.

CHAPTER 5
PROJECT IMPLEMENTATION

5.1 DATA PREPROCESSING:



```
[34]: df['class_name'].unique()
```

```
[34]: array(['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz',
             'metal', 'pop', 'reggae', 'rock'], dtype=object)
```

```
[36]: df['class_name'] = df['class_name'].astype('category')
      df['class_label'] = df['class_name'].cat.codes
      lookup_genre_name = dict(zip(df.class_label.unique(), df.class_name.unique()))
      lookup_genre_name
```

```
[36]: {0: 'blues',
       1: 'classical',
       2: 'country',
       3: 'disco',
       4: 'hiphop',
       5: 'jazz',
       6: 'metal',
       7: 'pop',
       8: 'reggae',
       9: 'rock'}
```

```
[38]: df['class_name'].unique()
```

```
[38]: ['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', 'metal', 'pop', 'reggae', 'rock']
      Categories (10, object): ['blues', 'classical', 'country', 'disco', ..., 'metal', 'pop', 'reggae', 'rock']
```

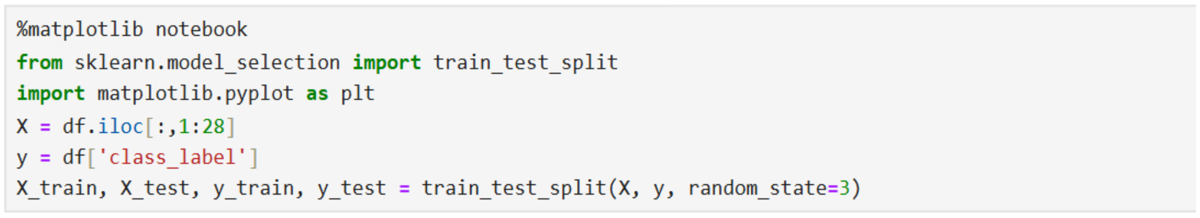
```
[40]: cols = list(df.columns)
      cols.remove('label')
      cols.remove('class_label')
      cols.remove('class_name')
      df[cols]
```

[40]:

	tempo	chroma_stft	rmse	spectral_centroid	spectral_bandwidth	rolloff	zero_crossing_rate	mfcc1	mfcc2	mfcc3	...	mfcc11	ı
0	103.359375	0.380260	0.248262	2116.942959	1956.611056	4196.107960	0.127272	-26.929785	107.334008	-46.809993	...	-15.234050	14.
1	95.703125	0.306451	0.113475	1156.070496	1497.668176	2170.053545	0.058613	-233.860772	136.170239	3.289490	...	5.145258	-2.
2	151.999081	0.253487	0.151571	1331.073970	1973.643437	2900.174130	0.042967	-221.802549	110.843070	18.620984	...	-26.406674	-13.
3	184.570312	0.269320	0.119072	1361.045467	1567.804596	2739.625101	0.069124	-207.208080	132.799175	-15.438986	...	-2.452068	-0.
4	161.499023	0.391059	0.137728	1811.076084	2052.332563	3927.809582	0.075480	-145.434568	102.829023	-12.517677	...	-6.934599	7.
...	...	...	...	...	...	...	...	...	...	...	...	...	

The code snippet preprocesses a music classification dataset for machine learning. It reads data from a CSV file into a pandas DataFrame and drops the irrelevant 'beats' column. A lookup dictionary is created to map these numerical labels back to genre names. The code then extracts the column names into a list, removes non-feature columns (label, class_label, class_name), and subsets the DataFrame to include only the feature columns. These steps prepare the dataset for training and evaluating genre classification models.

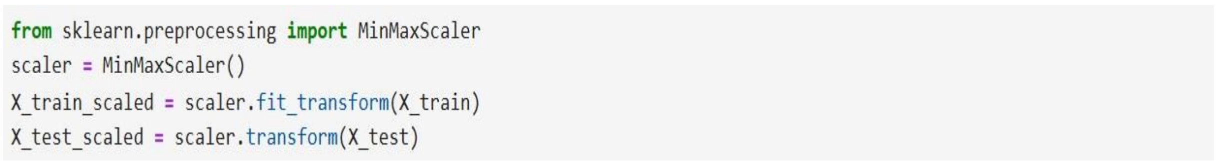
5.2 DATA SPLITTING FOR TRAINING AND TESTING:



```
%matplotlib notebook
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
X = df.iloc[:,1:28]
y = df['class_label']
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=3)
```

The code snippet prepares the dataset for training a machine learning model. It imports necessary libraries and splits the dataset into features (X) and target labels (y). Using train_test_split, it divides the data into training and testing sets with a random state of 3 for reproducibility. This setup is crucial for training and evaluating the model's performance on unseen data.

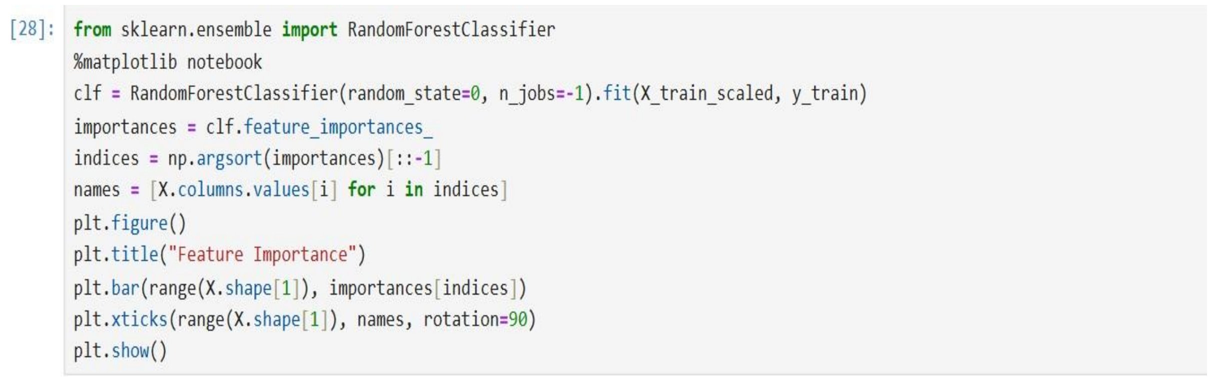
5.3 MIN-MAX NORMALIZATION:



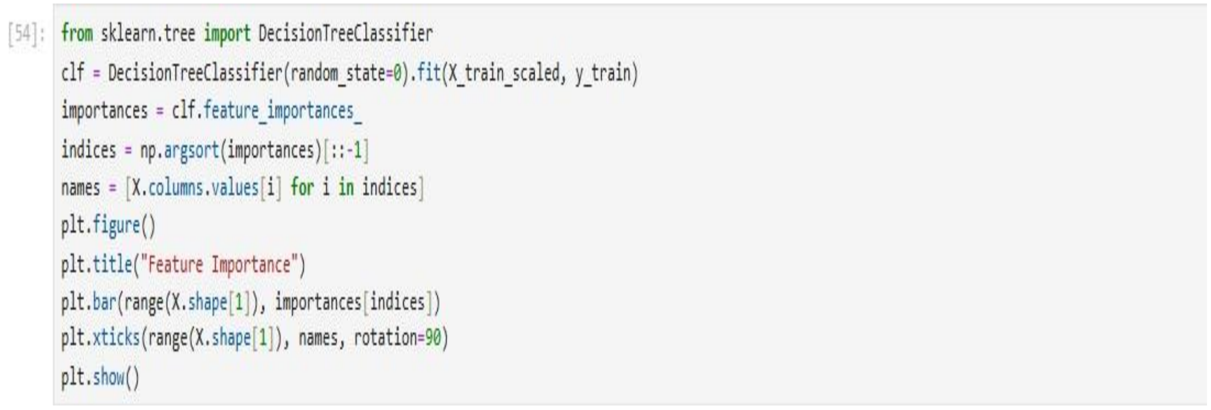
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

The code snippet scales the data using MinMaxScaler. It fits the scaler to X_train and transforms it into X_train_scaled. The same scaler is applied to X_test, resulting in X_test_scaled. This process normalizes feature values for better model performance.

5.4 FEATURE IMPORTANCE:



```
[28]: from sklearn.ensemble import RandomForestClassifier
      %matplotlib notebook
      clf = RandomForestClassifier(random_state=0, n_jobs=-1).fit(X_train_scaled, y_train)
      importances = clf.feature_importances_
      indices = np.argsort(importances)[::-1]
      names = [X.columns.values[i] for i in indices]
      plt.figure()
      plt.title("Feature Importance")
      plt.bar(range(X.shape[1]), importances[indices])
      plt.xticks(range(X.shape[1]), names, rotation=90)
      plt.show()
```



```
[54]: from sklearn.tree import DecisionTreeClassifier
      clf = DecisionTreeClassifier(random_state=0).fit(X_train_scaled, y_train)
      importances = clf.feature_importances_
      indices = np.argsort(importances)[::-1]
      names = [X.columns.values[i] for i in indices]
      plt.figure()
      plt.title("Feature Importance")
      plt.bar(range(X.shape[1]), importances[indices])
      plt.xticks(range(X.shape[1]), names, rotation=90)
      plt.show()
```

The code trains a RandomForestClassifier and DecisionTreeClassifier and visualizes feature importances. It fits the classifier using the scaled training data, computes feature importances, and sorts them. A bar chart is then plotted to display the sorted feature importances with feature names on the x-axis, highlighting the most influential features for model decisions.

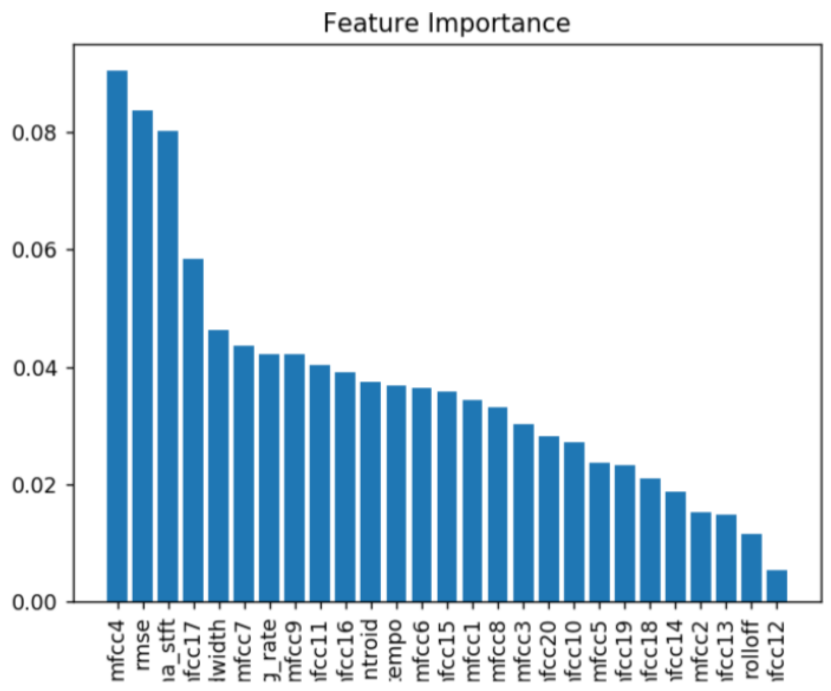


Fig 5.1 Feature Importance using Random Forest

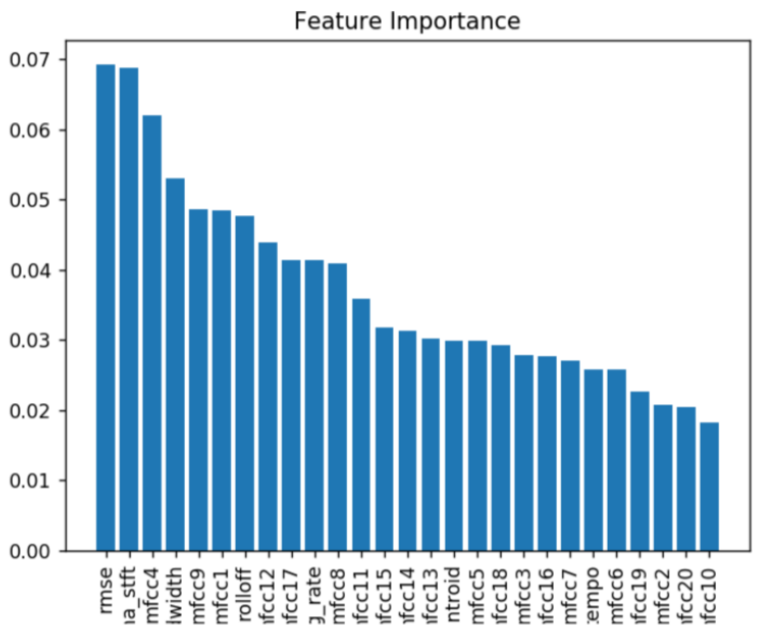
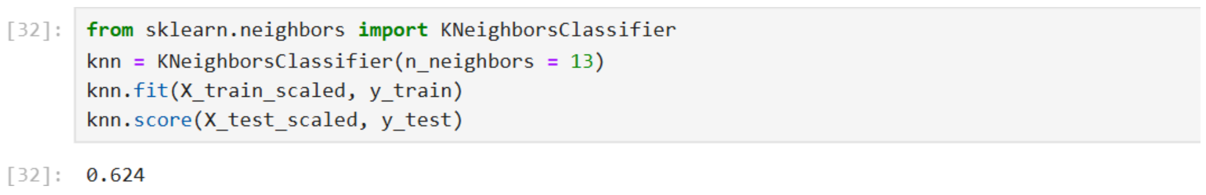


Fig 5.2 Feature Importance using Decision Tree

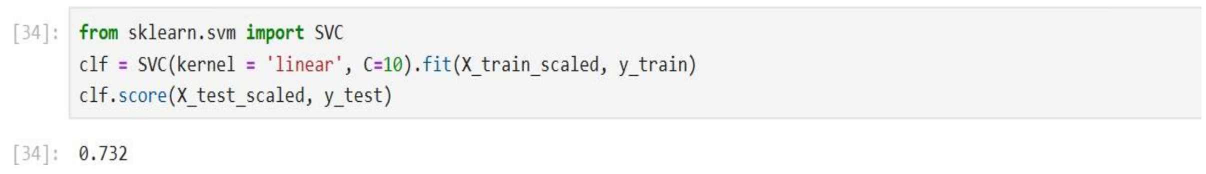
5.5 PREDICTING USING KNN:



```
[32]: from sklearn.neighbors import KNeighborsClassifier
      knn = KNeighborsClassifier(n_neighbors = 13)
      knn.fit(X_train_scaled, y_train)
      knn.score(X_test_scaled, y_test)

[32]: 0.624
```

The code trains a KNeighborsClassifier with 13 neighbors on the scaled training data and evaluates its performance on the scaled test data. It calculates the accuracy of the model on the test set using knn.score(X_test_scaled, y_test).

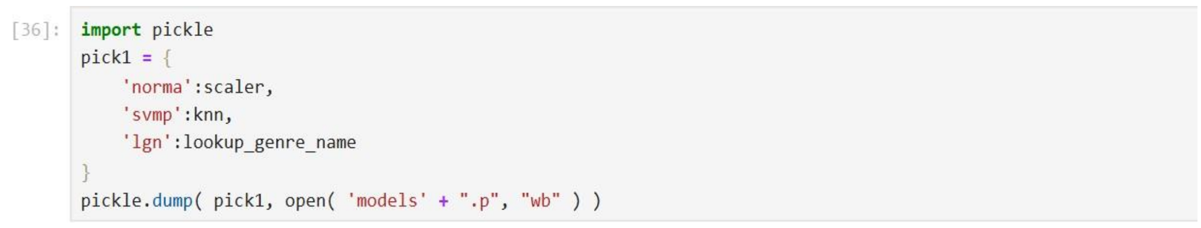
5.6 PREDICTING USING SVM:



```
[34]: from sklearn.svm import SVC
      clf = SVC(kernel = 'linear', C=10).fit(X_train_scaled, y_train)
      clf.score(X_test_scaled, y_test)

[34]: 0.732
```

The code trains an SVC (Support Vector Classifier) with a linear kernel and regularization parameter C=10C = 10C=10 on the scaled training data. It then evaluates the classifier's performance on the scaled test data, returning the accuracy score using clf.score(X_test_scaled, y_test).

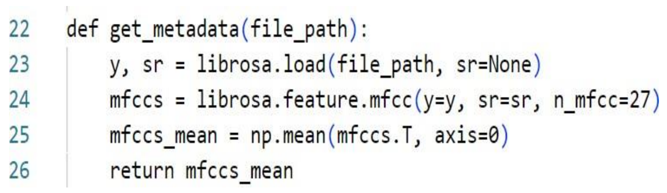
5.7 MODEL:



```
[36]: import pickle
      pick1 = {
          'norma':scaler,
          'svmp':knn,
          'lgn':lookup_genre_name
      }
      pickle.dump( pick1, open( 'models' + ".p", "wb" ) )
```

The code snippet creates a dictionary pick1 containing a scaler, a KNeighborsClassifier model, and a lookup dictionary for genre names. It then serializes this dictionary into a file named 'models.p' using pickle.dump. This allows for the saved models and preprocessing steps to be loaded and reused later.

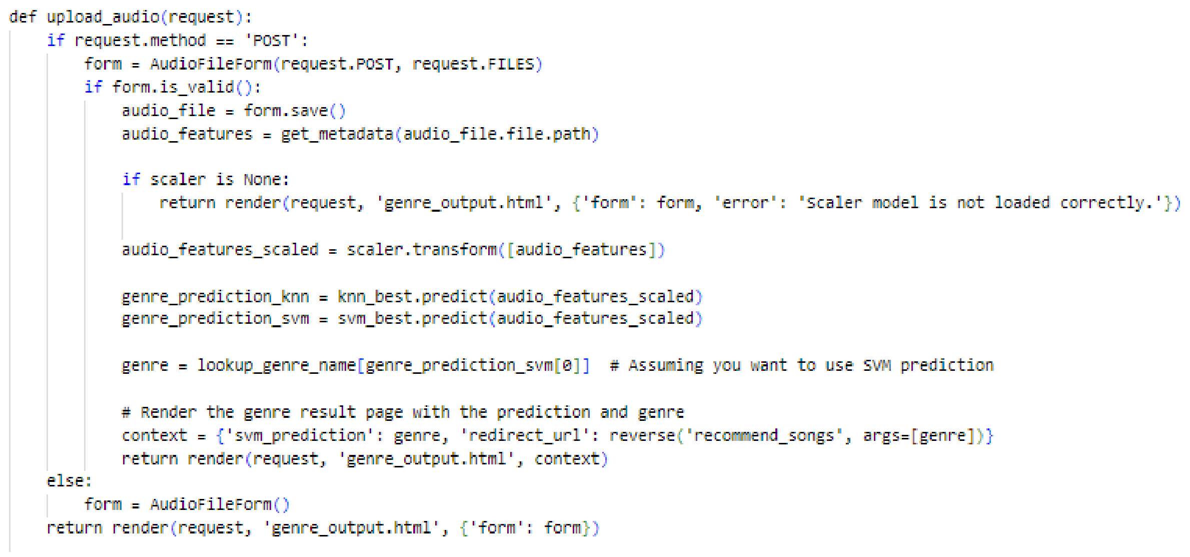
5.8 FEATURE EXTRACTION USING LIBROSA:



```
def get_metadata(file_path):
    y, sr = librosa.load(file_path, sr=None)
    mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=27)
    mfccs_mean = np.mean(mfccs.T, axis=0)
    return mfccs_mean
```

The get_metadata function extracts audio features from a given file path. It uses librosa to load the audio file and compute the Mel-frequency cepstral coefficients (MFCCs) with n_mfcc=27. The function calculates the mean of the MFCCs across time frames and returns this mean vector, which serves as a summary of the audio's spectral features.

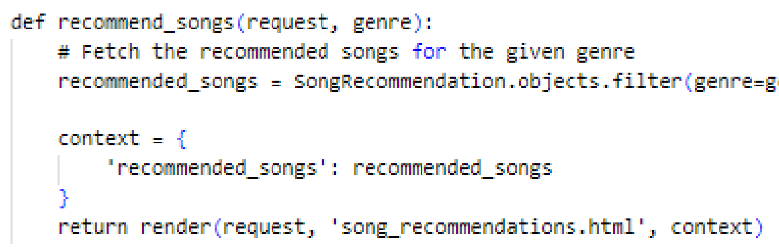
5.9 MUSIC INPUT AND GENRE OUTPUT:



```
def upload_audio(request):
    if request.method == 'POST':
        form = AudioFileForm(request.POST, request.FILES)
        if form.is_valid():
            audio_file = form.save()
            audio_features = get_metadata(audio_file.file.path)

            if scaler is None:
                return render(request, 'genre_output.html', {'form': form, 'error': 'Scaler model is not loaded correctly.'})

            audio_features_scaled = scaler.transform([audio_features])

            genre_prediction_knn = knn_best.predict(audio_features_scaled)
            genre_prediction_svm = svm_best.predict(audio_features_scaled)

            genre = lookup_genre_name[genre_prediction_svm[0]]  # Assuming you want to use SVM prediction

            # Render the genre result page with the prediction and genre
            context = {'svm_prediction': genre, 'redirect_url': reverse('recommend_songs', args=[genre])}
            return render(request, 'genre_output.html', context)
    else:
        form = AudioFileForm()
    return render(request, 'genre_output.html', {'form': form})
```

This function handles the uploading of an audio file, processes it to predict its genre using machine learning models, and then renders a template to display the result along with a recommendation link.

5.10 RECOMMENDATION:



```
def recommend_songs(request, genre):
    # Fetch the recommended songs for the given genre
    recommended_songs = SongRecommendation.objects.filter(genre=genre)[:5]

    context = {
        'recommended_songs': recommended_songs
    }
    return render(request, 'song_recommendations.html', context)
```

This function is designed to show a list of recommended songs based on the genre provided. It queries the database for song recommendations, limits the results to the top 5, and then renders them in a template for display.

CHAPTER 6

TESTING

Table 6.1

Testing Table

Test Case ID	Test Name	Description	Input	Expected Output	Actual Output	Status
1.1	Unit Testing	Validate correct file format (.wav)	Audio file with .wav extension	True	True	Pass
1.2		Validate incorrect file format (non-.wav)	Audio file with non-.wav extension	False	False	Pass
1.3		Validate successful feature extraction	Valid .wav audio file	True	True	Pass
1.4		Validate unsuccessful feature extraction	Invalid or corrupted audio file	False	False	Pass
1.5		Validate successful genre recommendation	Valid audio file	True	True	Pass
1.6		Validate unsuccessful genre recommendation	Invalid or unclassifiable audio file	False	False	Pass
2.1		User successfully uploads a	.wav file of a song	File uploaded successfully	File uploaded	Pass

		valid audio file			successfully	
2.2	Acceptance Testing	User attempts to upload an invalid file type	.mp3 file	System rejects file, displays error message about invalid format	System rejects file, displays error message about invalid format	Pass
2.3		System correctly classifies a clear genre	.wav file of a rock song	Genre classified as “Rock”	Genre classified as “Rock”	Pass
2.4		User receives relevant genre-based recommendations	Classified jazz song	List of recommended jazz artists or songs	List of recommended jazz artists or songs	Pass
2.5		User navigates through entire classification process	User actions from file upload to viewing recommendations	Smooth navigation, clear instructions, responsive interface	Smooth navigation, clear instructions, responsive interface	Pass
3.1	Integration Testing	Verify that uploaded file is correctly passed to feature extraction module	.wav audio file	Features (MFCC, chroma, spectral) successfully extracted from the uploaded file	Features (MFCC, chroma, spectral) successfully extracted from the uploaded file	Pass
3.2		Ensure extracted features are correctly input into classification model	Extracted audio features	Classification model receives correct feature set and produces a genre label	Classification model receives correct feature set and produces a genre label	Pass

3.3		Check if classified genre correctly triggers recommendation system	Classified genre label	Recommendation system generates a list based on the classified genre	Recommendation system generates a list based on the classified genre	Pass
3.4		Test if user actions in UI correctly trigger backend processes	User initiates classification through UI	Backend processes start and complete without errors	Backend processes start and complete without errors	Pass
4.1	System Testing	Verify complete workflow from file upload to genre recommendation	.wav audio file of a song	Successful upload, classification, and relevant genre recommendations	successful upload, classification, and relevant genre recommendations	Pass
4.2		Evaluate overall accuracy of genre classification	Set of 100 pre-classified songs across various genres	At least 90% classification accuracy	At least 90% classification accuracy	Pass
4.3		Assess the quality of genre-based recommendations	Classify songs from diverse genres	Recommendations are relevant and varied for each genre	Recommendations are relevant and varied for each genre	Pass

6.1 UNIT TESTING

6.1.1 Objective

To ensure that the genre classification and recommendation system is functioning correctly and efficiently, providing accurate results based on the implemented machine learning models.

6.1.2 Scope

- Testing the genre classification accuracy.
- Testing the recommendation system's relevance.
- Ensuring the correctness of the feature extraction process.
- Validating the handling of different types of audio files.

6.2 ACCEPTANCE TESTING

6.2.1. Objective

To verify that the genre classification and recommendation system meets all specified requirements and performs accurately and efficiently in a real-world scenario.

6.2.2. Scope

- End-to-end testing of the entire system.
- Validation against user requirements.

6.2.3 Acceptance Criteria

- System correctly classifies genres.
- System provides relevant genre recommendations.
- System accepts .wav audio files only.

6.3 INTEGRATION TESTING

6.3.1 Objective

To verify that the integrated components of the genre classification and recommendation system interact correctly and provide accurate and efficient results.

6.3.2 Scope

- Integration between feature extraction and genre classification
- Integration between genre classification and recommendation system
- Handling of different types of audio files

6.4 SYSTEM TESTING

6.4.1 Objective

- To validate that the system meets its functional requirements.
- To ensure the system's performance, security, and usability.
- To verify that the system works as expected in a real-world environment.
- To identify and resolve any defects before the system is deployed to production.

6.4.2 Scope

- **Functional Testing:** Verify that the system correctly extracts audio features, classifies genres accurately, and provides relevant genre recommendations.
- **Performance Testing**: Assess the system's response times, scalability, and behavior under different load conditions, ensuring it can handle audio files efficiently.

Unit testing validates the accuracy of genre classification, the relevance of recommendations, and the correctness of the feature extraction process. It also ensures the system can handle different types of audio files effectively.

Acceptance testing confirms that the system meets all specified requirements and performs accurately in real-world scenarios. It ensures correct genre classification, relevant recommendations, and acceptance of .wav audio files.

Integration testing verifies that the integrated components interact correctly and provide accurate results. It also assesses the system's ability to handle different audio file types seamlessly.

System testing ensures the system meets its functional requirements and operates as expected in a real-world environment. It checks for performance, security, usability, and the resolution of any defects before deployment.

CHAPTER 7

RESULTS AND DISCUSSION

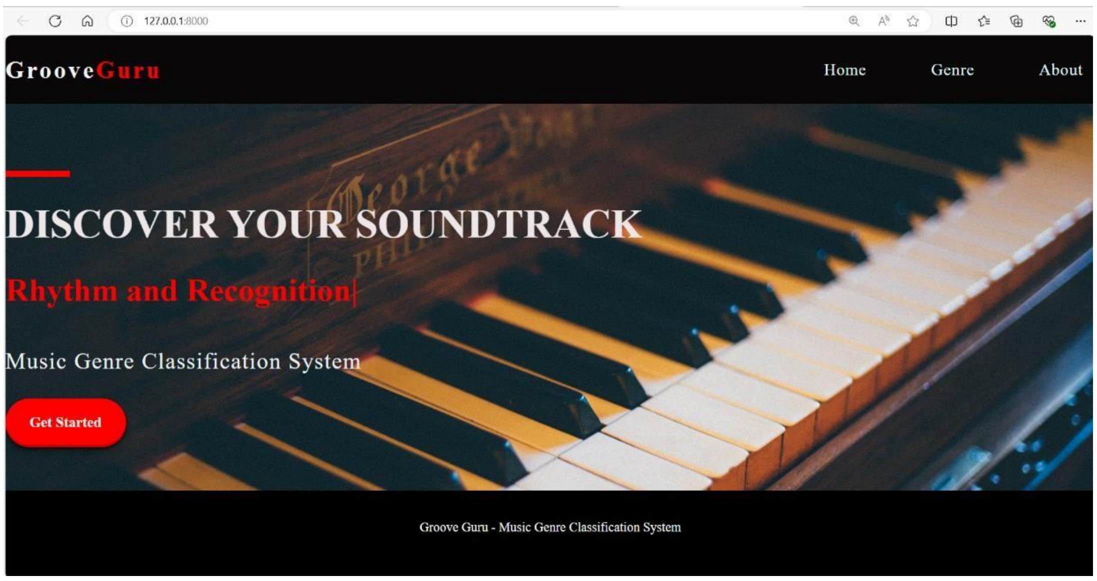
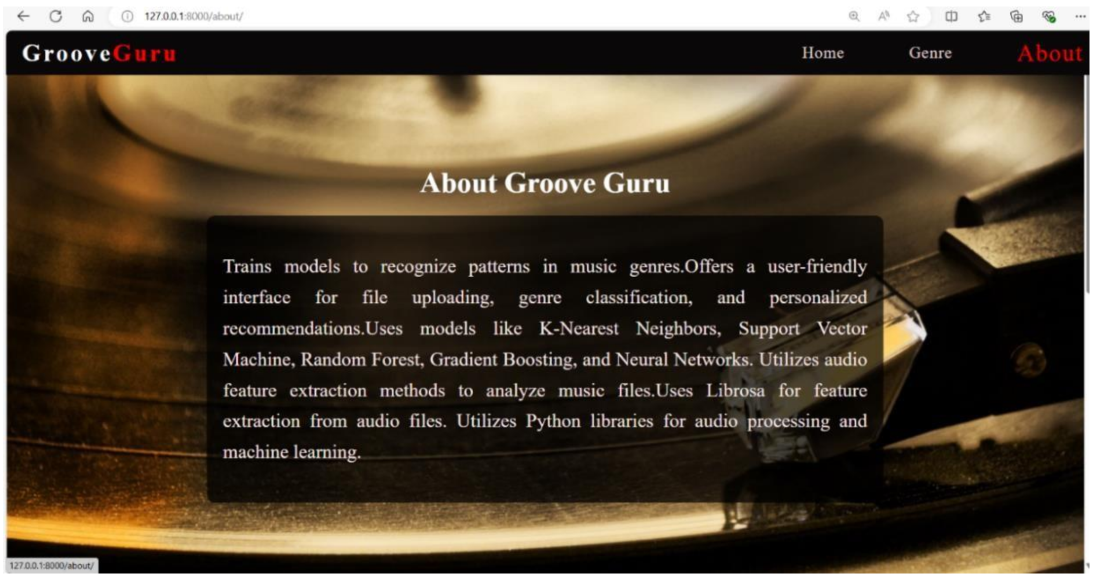


Fig 7.1 Homepage

The Groove Guru homepage features a captivating piano image, prominently showcasing the name and tagline to clearly communicate its value proposition. The "Get Started" button serves as a strong call-to-action, inviting users to embark on a personalized music discovery journey.



About Groove Guru

Trains models to recognize patterns in music genres.Offers a user-friendly interface for file uploading, genre classification, and personalized recommendations.Uses models like K-Nearest Neighbors, Support Vector Machine, Random Forest, Gradient Boosting, and Neural Networks. Utilizes audio feature extraction methods to analyze music files.Uses Librosa for feature extraction from audio files. Utilizes Python libraries for audio processing and machine learning.

Fig 7.2 About Page

The page emphasizes Groove Guru's core features: training models to recognize music genre patterns, offering an intuitive interface for file uploads, genre classification, and personalized recommendations. It highlights the use of various machine learning models and audio feature extraction techniques for precise genre identification.

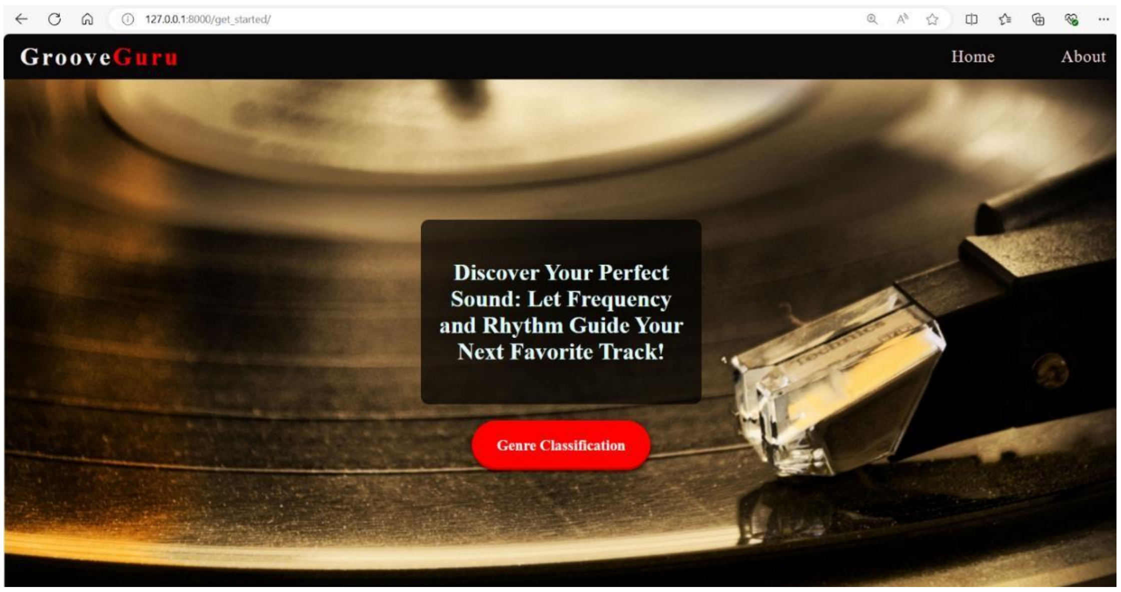

127.0.0.1:8000/get_started/
GrooveGuru
Home
About
Discover Your Perfect Sound: Let Frequency and Rhythm Guide Your Next Favorite Track!
Genre Classification

Fig 7.3 Get Started Page

The “Get Started” page offers a straightforward introduction to Groove Guru's music genre classification features, guiding users through the steps of uploading audio files and starting the classification process for an effortless music discovery experience.

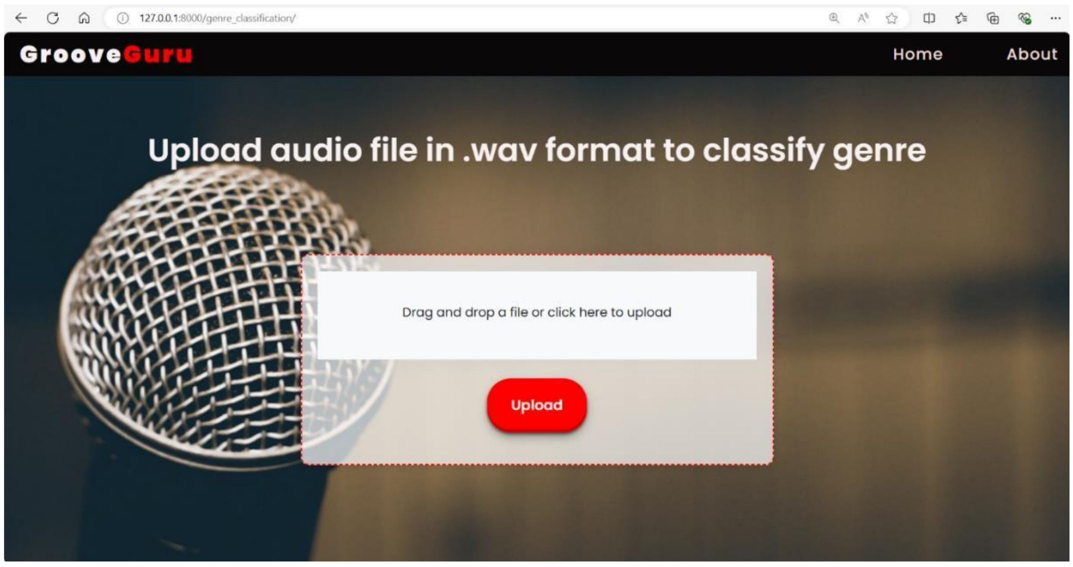

127.0.0.1:8000/genre_classification/
GrooveGuru
Home
About
Upload audio file in .wav format to classify genre
Drag and drop a file or click here to upload
Upload

Fig 7.4 Upload File

The image depicts the upload page of Groove Guru, a music genre classification platform. Users are instructed to upload an audio file in .wav format to initiate the classification process. The user-friendly interface allows for easy file uploading via drag-and-drop or manual selection.

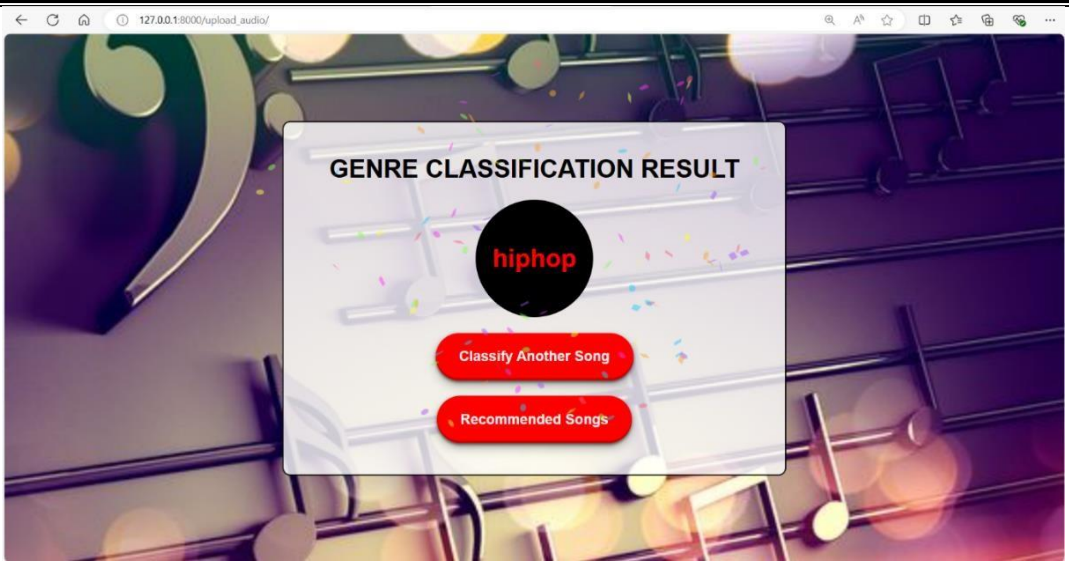
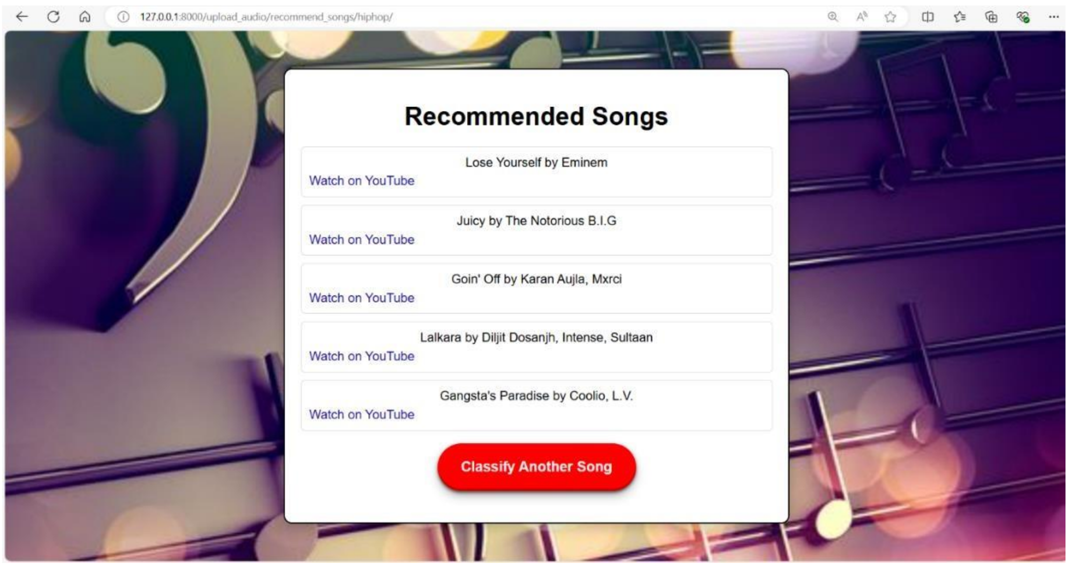


Fig 7.5 Classified Genre Result

The image depicts the genre classification result page of Groove Guru. The classified genre, “hiphop”, is displayed prominently in a circle. Users have the option to classify another song or explore recommended songs within the same genre, providing a seamless and engaging music discovery experience.



127.0.0.1:8000/upload_audio/recommend_songs/hiphop/

Recommended Songs

Lose Yourself by Eminem
Watch on YouTube

Juicy by The Notorious B.I.G
Watch on YouTube

Goin' Off by Karan Aujla, Mxrci
Watch on YouTube

Lalkara by Diljit Dosanjh, Intense, Sultaan
Watch on YouTube

Gangsta's Paradise by Coolio, L.V.
Watch on YouTube

Fig 7.6 Song Recommendation Page

The page displays a list of recommended songs within the classified genre, likely “hiphop” based on the title of previous images in the conversation. Users can easily access these songs by clicking the “Watch on YouTube” links, facilitating further music exploration.

CHAPTER 8
CONCLUSION & FUTURE SCOPE

8.1. CONCLUSION:

GrooveGuru stands to significantly improve how users discover and organize their music collections through accurate genre classification and intelligent recommendations. By utilizing advanced audio feature extraction techniques and robust machine learning models, GrooveGuru offers a sophisticated solution that caters to modern music enthusiasts' needs, ensuring an enriched and personalized musical experience.

GrooveGuru aims to continually advance its capabilities to redefine music discovery and user interaction. Planned enhancements include integrating deep learning models such as convolutional and recurrent neural networks for more accurate genre classification and feature extraction. Real-time audio processing will enable instantaneous classification and recommendations during music playback.

8.2. FUTURE SCOPE:

Enhanced metadata integration, intuitive user interfaces, and cross-platform support will further enrich the user experience, while social features and personalized playlists will foster community engagement and tailored music experiences. Integration with streaming services and support for multiple languages and regions will ensure GrooveGuru remains versatile and accessible globally. Continuous feedback mechanisms will drive ongoing improvements, making GrooveGuru a pivotal tool for music enthusiasts worldwide.

REFERENCES

[1] N. Kamuni and D. Panwar, “Enhancing Music Genre Classification through Multi-Algorithm Analysis and User-Friendly Visualization”, Apr. 2024, doi: 10.52783/jes.3178.

[2] A. S. Lampropoulos, P. S. Lampropoulouand G. A. Tsihrintzis, “A Cascade-Hybrid Music Recommender System for mobile services based on musical genre classification and personality diagnosis”, vol. 59, no. 1, Jul. 2012, doi: 10.1007/S11042-011-0742-0.

[3] A. Rosner and B. Kostek, “Automatic music genre classification based on musical instrument track separation”, vol. 50, no. 2, Apr. 2018, doi: 10.1007/S10844-017-0464-5.

[4] Y. Li, Z. Zhang, H. Dingand L. Chang, “Music genre classification based on fusing audio and lyric information”, vol. 82, no. 13, Dec. 2022, doi: 10.1007/s11042-022-14252-6.

[5] Y. Song, S. Dixonand M. Pearce, “A Survey of Music Recommendation Systems and Future Perspectives”, Jun. 2012.

WEB REFERENCES

[1] “librosa-librosa 0.10.2 documentation”, Jul. 10,2024. http://librosa.org/doc/latest/index.html (accessed Jul. 25, 2024).

[2] “Documentation scikit-learn: machine learning in Python – scikit-learn 0.21.3 documentation”. http://scikit-learn.org/0.21/documenation.html (accessed Jul. 25,2024)