**AIDI 2003 FINAL PROJECT REPORT (GROUP 14)**

**TEAM MEMBERS:**

**OLUWASEUN OBAFEMI**

**BABAFEMI ADEOYE**

**ASHWINI SARJARE**

**INSTRUMENT RECOGNITION APPLICATION**

This project aims to bridge the domains of audio recognition and text sentiment analysis, offering an innovative approach to engage users with the world of musical instruments. By combining advanced AI models and interactive web interfaces, the project enables users to input musical instrument names and receive predictions about both the instrument's audio representation and the sentiment associated with it.

# FRONTEND

**Imports**:

* **streamlit**: This library is used to create interactive web applications with Python.
* **requests**: This library is used for making HTTP requests to the prediction server.
* **pygame**: This library is used to handle audio playback.
* **os**: This library provides functions for interacting with the operating system.

**Streamlit Setup**:

* The Streamlit application title is set to "Instrument Recognition AI with Audio."
* A brief description is displayed using the **st.write()** function.

**User Input**:

* A text input field is provided using **st.text\_input()** where the user can enter the name of a musical instrument.

**Recognition and Audio Playback**:

* A button labeled "Recognize" is displayed.
* When the "Recognize" button is clicked and there is user input, an HTTP POST request is made to a prediction server running at **http://localhost:5000/predict**, sending the entered instrument name as JSON data.
* The response from the prediction server is parsed as JSON, containing the predicted instrument name and a confidence score.
* The predicted instrument and confidence score are displayed using **st.write()**.

**Audio Playback**:

* The audio filename corresponding to the predicted instrument is constructed.
* The audio file's path is generated using **os.path.join()** to ensure cross-platform compatibility.
* The **pygame.mixer** module is initialized for audio playback.
* The audio file is loaded using **pygame.mixer.Sound()** and played using **sound.play()**.

Below are the things we considered:

* **Dependencies**: This code requires the following libraries to be installed: **streamlit**, **requests**, and **pygame**.
* **Audio Files**: The code expects audio files named after the predicted instrument
* **Input Validation**: The code doesn't currently include extensive input validation or error handling for scenarios like empty input, invalid instrument names, or server unavailability.

# BACKEND

**Imports:**

* Flask: This library is used to create a Flask web application.
* request: This module provides access to the incoming request data.
* jsonify: This function is used to convert Python dictionaries into JSON format.
* pipeline: This function from the transformers library sets up a text classification pipeline.
* pygame: This library is imported but not used in this code snippet.

**Flask Setup:**

* An instance of the Flask application is created with app = Flask(\_\_name\_\_).

**Language Model Setup:**

* The Hugging Face pipeline() function is used to load a pre-trained language model for text classification.
* The model is configured for text classification using the "nlptown/bert-base-multilingual-uncased-sentiment" model.

**Prediction Endpoint (/predict):**

* This code defines a route /predict with the HTTP method POST.
* The endpoint receives JSON data containing an instrument\_name field from the request.
* If the instrument\_name field is present, the text classification model is used to predict the sentiment label and confidence score for the input text.
* The predicted sentiment label and confidence score are extracted from the model's output.
* The result is returned as a JSON response using jsonify().

**Main Application Execution:**

* The block if \_\_name\_\_ == "\_\_main\_\_": ensures that the Flask app is only run when the script is executed directly, not when it's imported as a module.
* The app.run(debug=True) statement starts the Flask development server with debugging enabled.

Below are the things we considered while writing this code;

* **Hugging Face Model:** The code utilizes a pre-trained BERT-based model ("nlptown/bert-base-multilingual-uncased-sentiment") from the Hugging Face Transformers library. This model is designed for text classification tasks.
* **Input Validation:** The code doesn't currently include extensive input validation. It assumes that the incoming JSON data contains a valid instrument\_name field.
* **Deployment:** This Flask app can be deployed to a web server for use as a text classification service.
* **Text Classification Use Case:** This code is specifically designed for text sentiment classification.

# ADDITIONAL PROJECT DETAILS

**Technical Explanation:**

The code implements a Flask-based web service for text classification using the Hugging Face Transformers library. The service takes an instrument name as input, leverages a pre-trained BERT-based sentiment classification model, predicts the sentiment label and confidence score, and returns the result in JSON format.

**Evaluation Metrics:**

For sentiment classification, the model's performance can be evaluated using metrics such as accuracy, precision, recall, F1-score, and confusion matrix. The was conducted on a labeled dataset with true sentiment labels to gauge how well the model's predictions align with the actual sentiments.

**Limitations:**

* Input Dependency: The code assumes the input data contains a valid instrument\_name field. It lacks explicit validation for the presence of this field and for potential data types.
* Language Model Limitations: The model's performance heavily depends on the training data. If the dataset used to train the model doesn't cover musical instrument names or specific language nuances, the predictions might be inaccurate.
* Single Model: The code uses a single sentiment classification model. It may not perform well on instruments with unconventional or less common names.
* Generalization: The model might not generalize well to musical instrument descriptions that are more complex or include multiple sentiments.

**Ethical Considerations:**

* Bias: The sentiment classification model might inherit biases from its training data, leading to unfair predictions.
* Data Privacy: The model requires user input data. We ensured that user data is handled securely and ethically, abiding by privacy regulations.
* Misuse: If the service is misused to classify inappropriate or harmful content, ethical implications may arise.

**Future Improvements:**

* Input Validation
* Ensemble Models
* Multi-Label Classification
* Interactive Feedback

**Language Model Selection and Justification:**

* Selection: The chosen language model is "nlptown/bert-base-multilingual-uncased-sentiment" from the Hugging Face Transformers library.
* Justification: This model is selected because it's pre-trained for sentiment classification tasks and is multilingual, allowing it to handle various languages. However, it's worth noting that the model's primary intention seems to be sentiment analysis, not musical instrument recognition.