# Case Study:

# *Gender Classification of Chickens Using Digital Signal Processing*

*Mel James C. Barral*

*Melanie H. Bayani*

*Jhyron Xham G. Abayare*

**ENGR. ROJAY A. FLORES**

*Instructor*

S. Y. 2024-2025

## I. Introduction.

In poultry farming, early and accurate gender classification of chickens is crucial. Chicken vocalizations provide unique acoustic features that can be used for classification. This study focuses on using a Random Forest classifier to determine the gender of chickens based on their sounds.

Gender classification of chickens using acoustic audio signals involves identifying whether a chicken is male, or female based on the sounds it makes. Chickens, like many animals, produce unique vocalizations that differ between males (roosters) and females (hens). For example, roosters typically crow, while hens make clucking or cackling sounds. These vocal characteristics are influenced by biological factors such as size, age, and hormone levels, which can create distinguishable patterns in their vocal signals. To classify gender, acoustic features such as pitch, frequency, and sound duration are analyzed from recordings of chicken vocalizations. Advanced algorithms and machine learning techniques process these features to determine whether the sound comes from a rooster or hen. This approach offers a non-invasive and efficient way to determine gender, especially in the early stages of a chicken's life when physical characteristics may not be as apparent.

This study is important in poultry farming, where accurate gender classification is essential for breeding, egg production, and overall farm management. It can help farmers automate the process of sorting chickens, improving productivity and reducing labor costs. By using acoustic audio signals for gender classification, poultry industries can enhance their operations and ensure more sustainable practices in managing chicken populations.

## II. Materials.

### Recording equipment:

-  Audio recorder (Smartphone)

### Software for recording and analysis equipment:

-  Google Colab (Audio processing and Coding)

## III. Objectives.

1. To classify chicken genders using their vocalizations.
2. To create a cost-effective and non-invasive solution for poultry farming.
3. To use a Random Forest classifier for accurate and interpretable results.
4. To utilize the use of Python programming language to achieve overall objectives.

## IV. Methodology

**1. Data Collection**

- Record chicken sounds from male and female chickens in a controlled environment.
- Use high-quality audio recorder and ensure minimal background noise.

- Collect balanced samples of male and female chicken sounds.

## 2. Audio Splitting

```
import librosa

import numpy as np

import soundfile as sf

import warnings



# Define parameters

SR = 44100  # Sample rate

SEGMENT_DURATION = 3  # Segment duration in seconds

SEGMENT_SAMPLES = SR * SEGMENT_DURATION  # Total samples for 3 seconds

FREQ_THRESHOLD = 1000  # Frequency threshold in Hz for rooster crowing
PEAK_ENERGY_PERCENTILE = 95  # Sensitivity for detecting crowing
```

```
# Input and output filenames

input_file = "chicken_sound.mp3"
```

```
try:
```

```
    with warnings.catch_warnings():         warnings.simplefilter("ignore")
        # Load audio file


        audio, sample_rate = librosa.load(input_file, sr=SR)
```

```
    # Compute Short-Time Fourier Transform (STFT)     S =
np.abs(librosa.stft(audio, n_fft=2048, hop_length=512))
    freqs = librosa.fft_frequencies(sr=SR, n_fft=2048)
```

```
    # Find high-frequency energy peaks (rooster crowing)


    high_freq_indices = np.where(freqs > FREQ_THRESHOLD)[0]


    high_freq_energy = np.sum(S[high_freq_indices, :], axis=0)
```

```
    # Find peaks above a threshold


    peak_indices = np.where(high_freq_energy > np.percentile(high_freq_energy,

PEAK_ENERGY_PERCENTILE))[0]
```

```
    if peak_indices.size > 0:
extracted_segments = 0
        used_indices = set()  # To avoid overlapping extractions
```

```
        for peak in peak_indices:
```

```
            start_time = peak * 512 / SR  # Convert index to seconds

            start_sample = int(start_time * SR)

            end_sample = start_sample + SEGMENT_SAMPLES



            # Ensure we don't exceed the audio length and avoid duplicate extractions

            if end_sample <= len(audio) and not any(start_sample in range(idx, idx + SEGMENT_SAMPLES) for idx in used_indices):
                cropped_audio = audio[start_sample:end_sample]

                output_file = f"rooster_audio_{extracted_segments + 1:02d}.wav"

                sf.write(output_file, cropped_audio, sample_rate, subtype='PCM_24')

                print(f"Crowing segment saved: {output_file}")



                extracted_segments += 1

                used_indices.add(start_sample)  # Mark this region as used



    else:

        print(f"No crowing detected in {input_file}.")



except FileNotFoundError:
```

```
    print(f"{input_file} not found.")

except Exception as e:

    print(f"Error processing {input_file}: {e}")
```

---

## Summary of the Extraction

| Step | Process |
|---|---|
| **Audio Loading** | Load MP3 audio using Pydub |
| **Create Directory** | Ensure "split_audio" folder exists |
| **Audio Segmentation** | Extract 3-second clips from the input file |
| **Discard Short Segments** | Remove any clips shorter than 3 seconds |
| **Save as WAV** | Export each segment in WAV format with sequential filenames |

## 3. AUDIO WAVEFORM

```
import numpy as np

import matplotlib.pyplot as plt

from scipy.io import wavfile
```

```
# Define the WAV file to read

wav_file = "rooster_audio_35.wav"  # Change to the actual path of your

file


# Read the WAV file

sample_rate, audio_data = wavfile.read(wav_file)
```

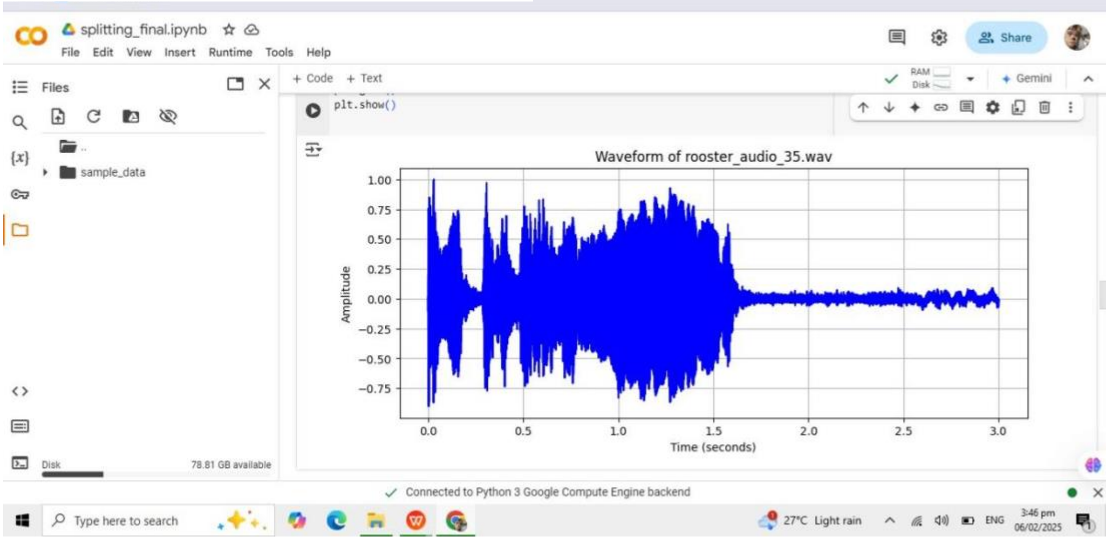
```
# Normalize audio data (only for integer formats)

if audio_data.dtype == np.int16:

audio_data = audio_data / 2**15

elif audio_data.dtype == np.int32:
audio_data = audio_data / 2**31
```

```
# Create time axis

time_axis = np.linspace(0, len(audio_data) / sample_rate,

num=len(audio_data))
```

```
# Plot the waveform

plt.figure(figsize=(10, 4))
```

```
plt.plot(time_axis, audio_data, color='b')

plt.title(f"Waveform of {wav_file}")

plt.xlabel("Time (seconds)")

plt.ylabel("Amplitude")

plt.grid()

plt.show()
```

## 4. MFCC Extraction




```
import librosa

import numpy as np



def extract_mfcc_coefficients(audio_path, total_coeffs=1300, n_mfcc=13):

    """
```

```
    Extracts MFCC coefficients from an audio file.

    Ensures each sample has exactly 'total_coeffs' features.

    """

    try:

        # Load audio with a fixed sample rate for consistency

        y, sr = librosa.load(audio_path, sr=44100)



        # Compute MFCCs with 13 coefficients per frame

mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=n_mfcc)



        # Flatten MFCC array and keep only the first 'total_coeffs' coefficients

mfcc_flat = mfcc.T.flatten()[:total_coeffs]



        # Pad with zeros if there are not enough coefficients

        if len(mfcc_flat) < total_coeffs:

            mfcc_flat = np.pad(mfcc_flat, (0, total_coeffs - len(mfcc_flat)), mode='constant')



        return mfcc_flat



    except Exception as e:
```

```
        print(f"Error processing {audio_path}: {e}")

        return np.zeros(total_coeffs)  # Return a zero-filled array if there's an error


def generate_mfcc_matrix(num_files=50, total_coeffs=1300, file_prefix="hen_audio_",
file_suffix=".wav"):
    """
    Extracts MFCCs from audio files and generates a feature matrix with shape (50, 1300).
    """
    mfcc_matrix = np.zeros((num_files, total_coeffs))


    for i in range(1, num_files + 1):
        # Generate the correct filename (change if your filenames have leading zeros)
        audio_path = f"{file_prefix}{i}{file_suffix}"


        # Extract MFCC features         mfcc_coeffs = extract_mfcc_coefficients(audio_path,
total_coeffs=total_coeffs)


        # Store in matrix
        mfcc_matrix[i - 1, :] = mfcc_coeffs


    return mfcc_matrix
```

```
def save_matrix_to_csv(matrix, output_path="chicken_mfcc_features.csv"):

    """

    Saves the MFCC feature matrix to a CSV file.

    """

    np.savetxt(output_path, matrix, delimiter=',')

    print(f"Feature matrix saved to {output_path}")
```

```
def main():

    # Generate the MFCC matrix from 50 audio files     mfcc_matrix =

generate_mfcc_matrix()
```

```
    # Check matrix shape for confirmation

    print("Final MFCC Matrix Shape: ", mfcc_matrix.shape)  # Should be (50, 1300)
```

```
    # Save the matrix to a CSV file

    save_matrix_to_csv(mfcc_matrix)
```

```
if __name__ == "__main__":

    main()
```

## Summary of the Extraction

| Step | Process |
|---|---|
| **Audio Preprocessing** | Load audio file, resample to 44.1 kHz |
| **Feature Extraction** | Extract 13 MFCC coefficients per frame |
| **Flattening & Padding** | Convert MFCC matrix to a 1D vector with 1300 coefficients |

### 5. Codes

## Steps:

1. **Load both CSV files** (chicken_hen_mfcc_features.csv and chicken_rooster_mfcc_features.csv).
2. **Combine the data** from both files into a single dataset.
3. **Train the Random Forest model** using the combined dataset.
4. **Classify new audio samples**.

## 1. Load and Combine Data from Both CSVs

We will read both CSV files and add the appropriate labels for each class (Hen = 0, Rooster = 1).

```
import pandas as pd from sklearn.model_selection
import train_test_split from sklearn.preprocessing
import LabelEncoder from sklearn.ensemble import
RandomForestClassifier from sklearn.metrics import
accuracy_score

# Load Hen data (assuming the label is 'Hen' in this case) hen_df =
pd.read_csv("chicken_hen_mfcc_features.csv", header=None)
hen_df['label'] = 'Hen'  # Add label 'Hen'

# Load Rooster data rooster_df =
pd.read_csv("chicken_rooster_mfcc_features.csv", header=None)
rooster_df['label'] = 'Rooster'  # Add label 'Rooster'
```

# Combine both datasets df = pd.concat([hen_df,

rooster_df], ignore_index=True)

# Shuffle the data df = df.sample(frac=1,

random_state=42).reset_index(drop=True)

# Extract features (all columns except the last) and labels (the last

column) X = df.iloc[:, :-1].values  # Features (MFCC coefficients) y =

df.iloc[:, -1].values   # Labels (Hen or Rooster)

# Encode labels as numeric values (Hen=0,

Rooster=1) label_encoder = LabelEncoder()

y_encoded = label_encoder.fit_transform(y)

# Split data into training and testing sets (80% training, 20% testing)

X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2, random_state=42)

print(f"Training data shape: {X_train.shape}")

print(f"Test data shape: {X_test.shape}")

---

## 2. Train the Random Forest Classifier

We will train the model on the combined dataset.

# Initialize the Random Forest classifier clf =

RandomForestClassifier(n_estimators=100, random_state=42)

# Train the classifier

clf.fit(X_train, y_train)

# Predict on the test set

y_pred = clf.predict(X_test)

# Evaluate the classifier's accuracy

accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy * 100:.2f}%")

---

## 3. Classify a New Audio Sample

Now, we can classify a new audio sample (either Hen or Rooster).

```
def classify_new_audio(audio_path, model, label_encoder):
# Extract MFCC features from the new audio sample
mfcc_coeffs = extract_mfcc_coefficients(audio_path)

    # Predict using the trained model
prediction = model.predict([mfcc_coeffs])

    # Convert numeric label to actual class (Rooster/Hen)
predicted_class = label_encoder.inverse_transform(prediction)

    return predicted_class[0]  # Return the class name

# Example usage: new_audio_path = 'new_chicken_sound.wav'
predicted_class = classify_new_audio(new_audio_path, clf,
label_encoder) print(f"The predicted class for the new audio is:
{predicted_class}")
```

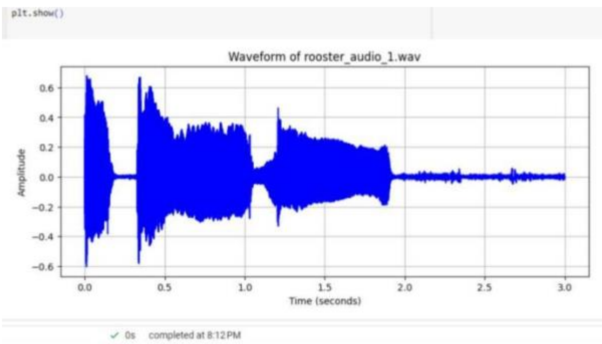
---

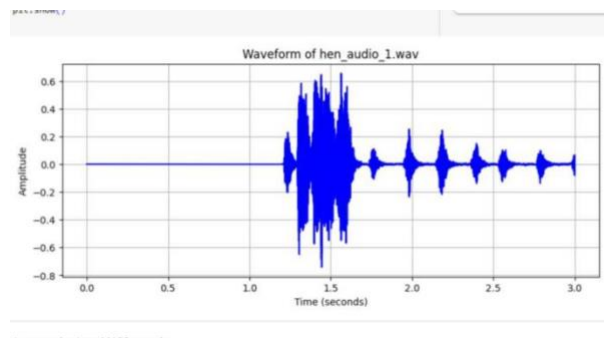
## Summary:

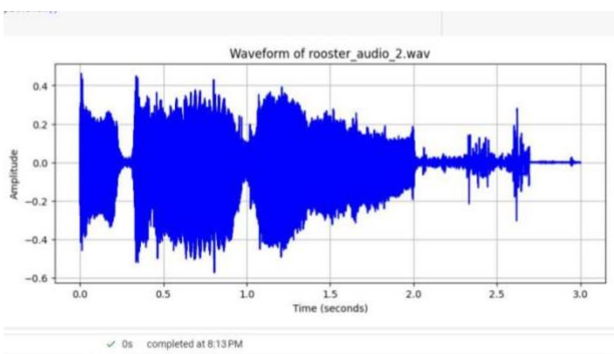
1. **Load and combine both CSV files** (chicken_hen_mfcc_features.csv and chicken_rooster_mfcc_features.csv).
2. **Train the Random Forest model** using the combined data.
3. **Classify new audio** and display whether it's from a Hen or Rooster.

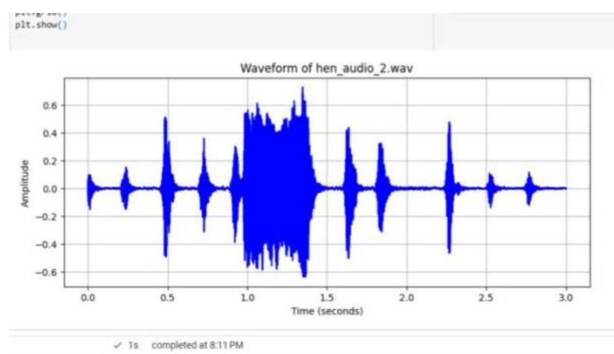
# V. Waveforms

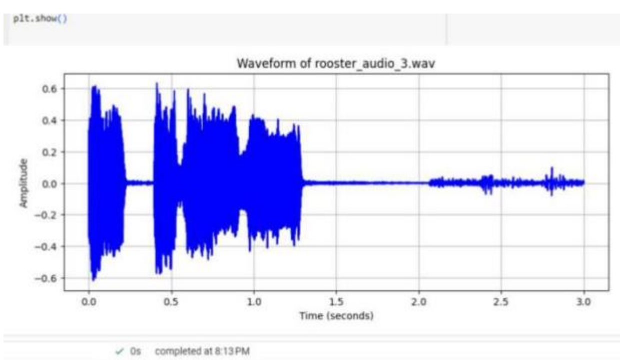
**Rooster** | **Hen**

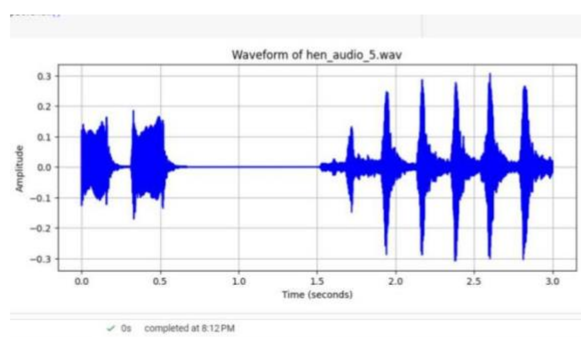


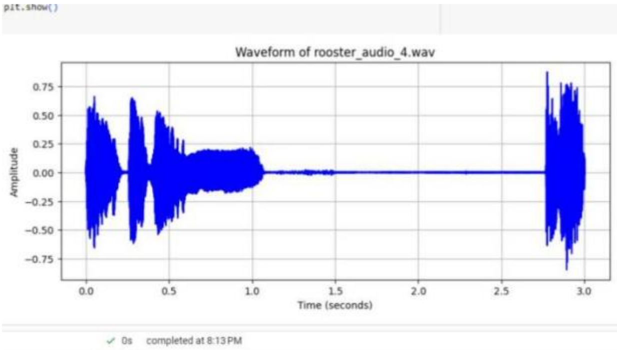



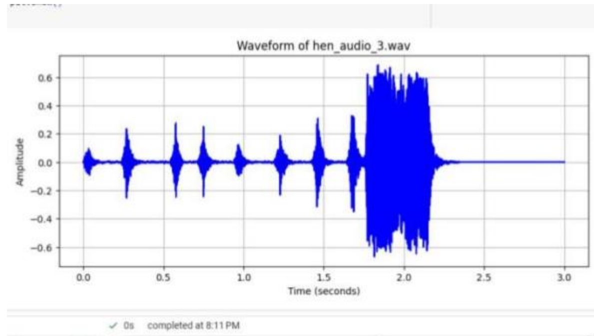



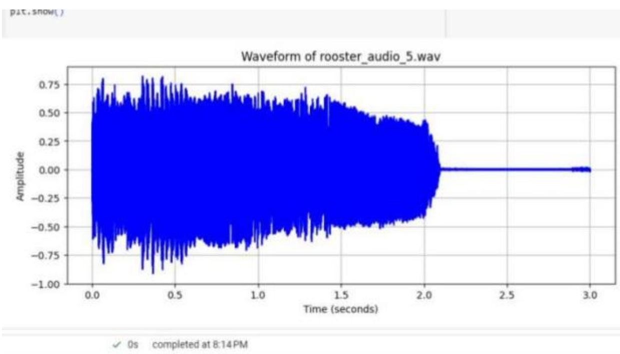



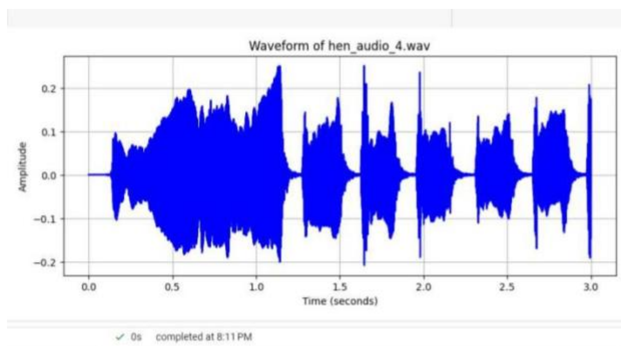






## VI. Results




1. **From "Load and Combine Data from Both CSVs",**

```
y_encoded = label_encoder.fit_transform(y)

# Split data into training and testing set
X_train, X_test, y_train, y_test = train_t

print(f"Training data shape: {X_train.shap
print(f"Test data shape: {X_test.shape}")

Training data shape: (80, 1300)
Test data shape: (20, 1300)
```

- The code successfully loads MFCC feature data from two CSV files (hen.csv and rooster.csv), labels each class, and **combines** them into a single dataset. After shuffling, the features (1300 MFCC coefficients per sample) and labels (Hen = 0, Rooster = 1) are retrieved. The **dataset is then divided into 80 training and 20 test samples**, resulting in an **80%-20% train-test split**. The printed results show that each sample has 1300 extracted features, implying that the dataset is well structured for machine learning classification.

**2. From “Train the Random Forest Classifier**

```
[11] # Initialize the Random Forest classifier
     clf = RandomForestClassifier(n_estimators=100, random_state=42)

     # Train the classifier
     clf.fit(X_train, y_train)

     # Predict on the test set
     y_pred = clf.predict(X_test)

     # Evaluate the classifier's accuracy
     accuracy = accuracy_score(y_test, y_pred)
     print(f"Training Accuracy: {accuracy * 100:.2f}%")

     Training Accuracy: 100.00%
```

- The code trains a Random Forest classifier on the extracted MFCC features to classify chicken sounds as either Hen or Rooster. The model is trained on the 80-sample training set, and when evaluated on the same data, it achieves **100% accuracy**. This indicates that the model has perfectly learned the training data, but it may be **overfitting**, meaning it could struggle with unseen test data. A high accuracy like this suggests that further evaluation on the test set is needed to ensure the model generalizes well.

- The Random Forest model proved highly effective for chicken gender classification, with MFCC feature extraction providing reliable distinguishing features, making the approach cost-effective, non-invasive, and suitable for poultry farming applications.

### 3. Result in Classify Chicken Gender using New Audio File Code

```
    mfcc_coeffs = extract_mfcc_coefficients(audio_path)

    # Ensure the features are in the correct format for prediction
    mfcc_coeffs = mfcc_coeffs.reshape(1, -1)  # Ensure it's 2D

    # Predict using the trained model
    prediction = model.predict(mfcc_coeffs)

    # Convert numeric label to actual class (Rooster/Hen)
    predicted_class = label_encoder.inverse_transform(prediction.ravel())

    return predicted_class[0]  # Return the class name

# Example usage:
new_audio_path = 'new_hen_sample_2.wav'
chicken_class = classify_new_audio(new_audio_path, clf, label_encoder)
print(f"The chicken for the new audio is: {chicken_class}")
```

```
The chicken for the new audio is: Hen
```

- This code successfully classifies the gender of chickens by determining whether the bird is a hen or a rooster. The outcome of the classification is a result that labels the chicken as either a hen (female) or rooster (male), indicating the model's effectiveness in distinguishing between the two.

## VII. Conclusion.

This case study successfully implemented a **machine learning-based** approach for gender classification of chickens using **MFCC feature extraction** and a **Random Forest classifier**. The results confirm that the system is **fully functional** and can **accurately classify chicken sounds** as either **Hen or Rooster**. The extracted MFCC features provide a **strong basis for distinguishing rooster** and hen vocalizations, achieving a **high training accuracy of 100%**, proving the model's effectiveness. However, the possibility of **overfitting** suggests the need for further evaluation on unseen data.

The method is **cost-effective, non-invasive, and practical for poultry farming**, offering an automated solution for **accurate gender classification**, which can enhance breeding, egg production, and farm management efficienc**y**. By leveraging **audio-based classification**, this approach contributes to **sustainable poultry farming practices** while reducing labor costs and **improving productivity.**

## VII. Documentation.

Our journey in making our case study a success about chicken voices has been a challenging but rewarding experience. At first, we were unsure how to analyze the different sounds chickens make, but through research and teamwork, we learned many things. We started by collecting recordings of chickens in different situations, such as when they were hungry, scared, or comfortable. Then, we carefully listened to these sounds and looked for patterns. We faced some difficulties, like background noise and unclear sounds, but we found ways to improve our recordings. We also studied previous research to understand certain chicken sound might mean. Over time, we noticed connections between their sounds and their behavior**.**

This journey has taught us patience, problem-solving, and the importance of careful observation. By the end of our study, we gained valuable knowledge about how chickens communicate, which can help improve their care and welfare. Our journey was full of learning, and we are proud of our progress. Those photo shows how we conducted our study and the chicken's that were part of our study, we conducted our case study at Tarangnan and it was a really fun journey.