**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:**

* Matlab
* 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.

1. **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 Method

**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**

Remove any clips shorter than 3 seconds **Segments**

Export each segment in WAV format with sequential

**Save as WAV** filenames

# 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))

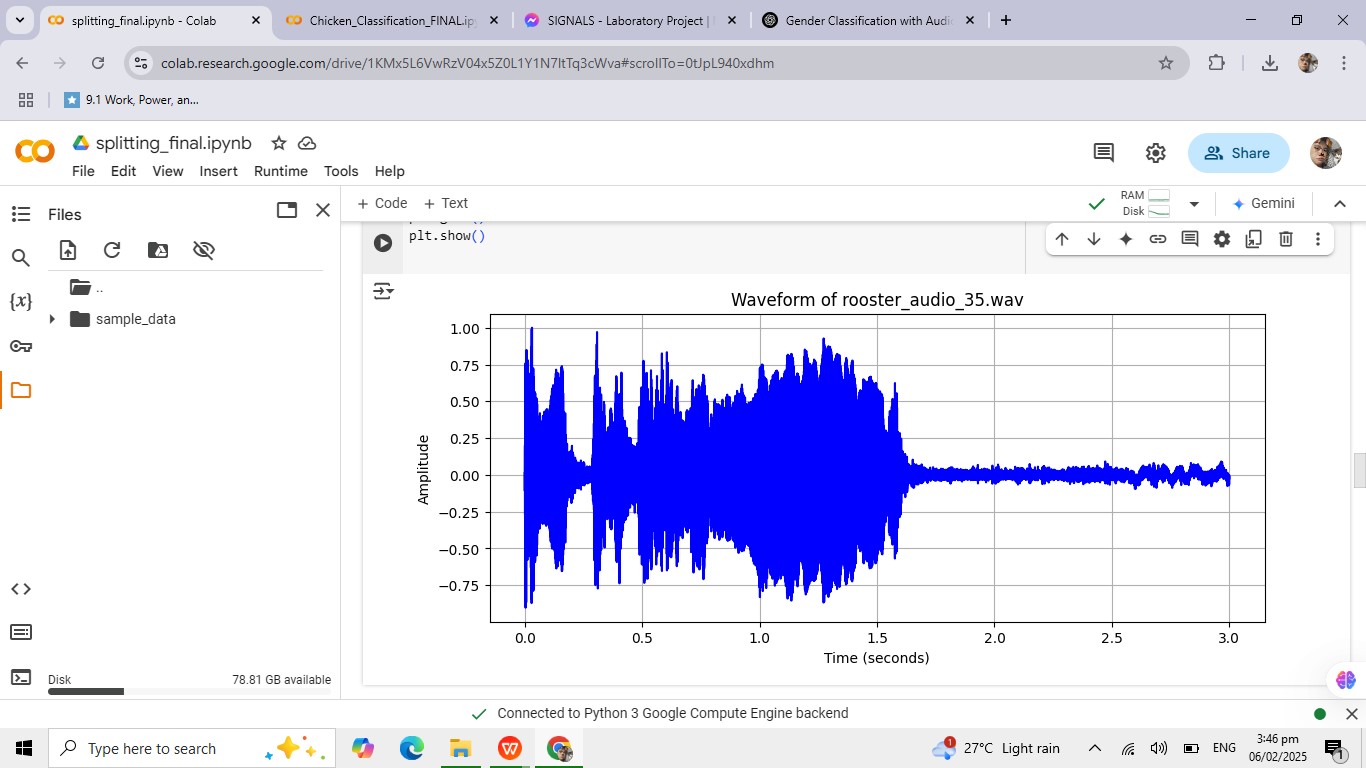
plt.plot(time\_axis, audio\_data, color='b')

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 Method

**Step Process**

**Audio**

Load audio file, resample to 44.1 kHz

**Preprocessing**

**Feature Extraction** Extract 13 MFCC coefficients per frame

**Flattening &** Convert MFCC matrix to a 1D vector with 1300

**Padding** 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)

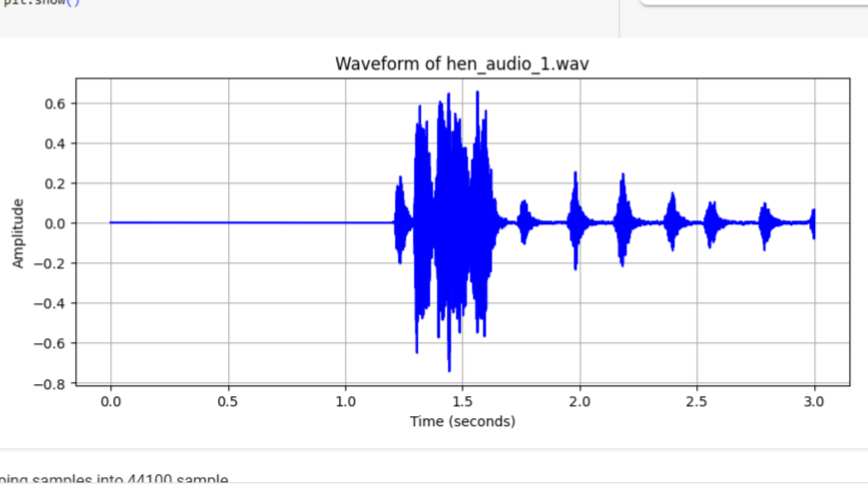
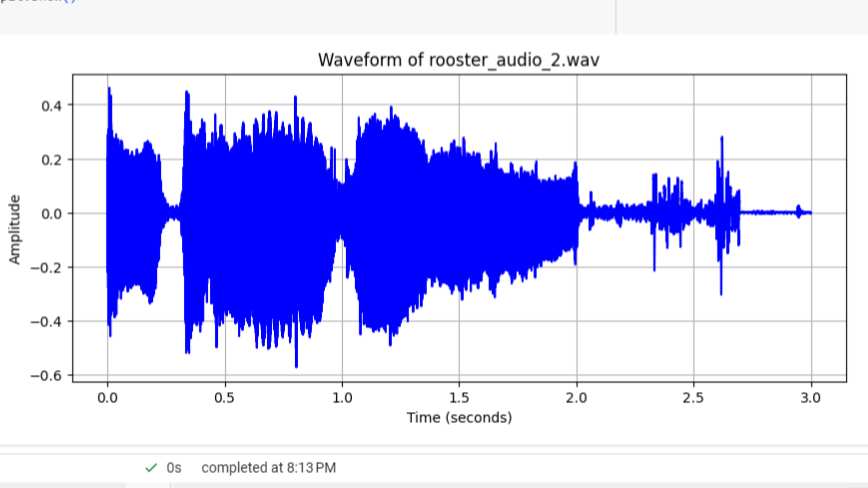
return predicted\_class[0] # Return the class name

# 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:**

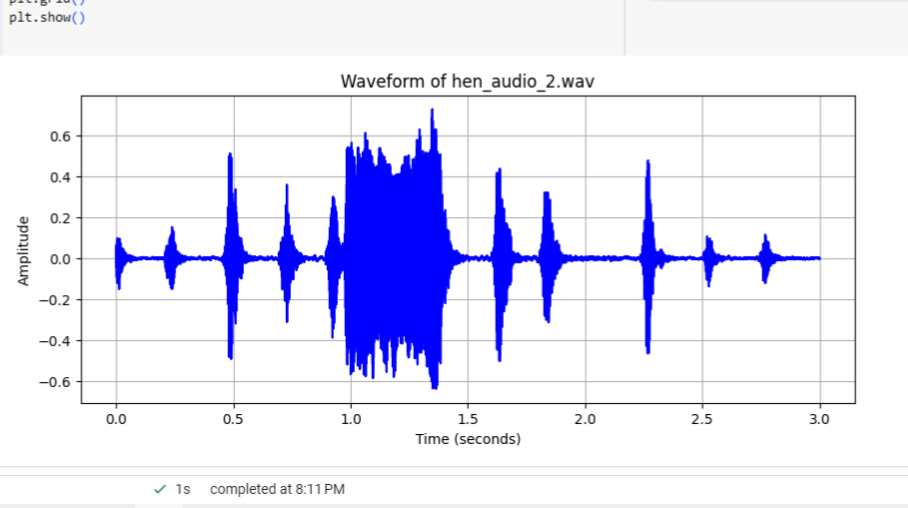
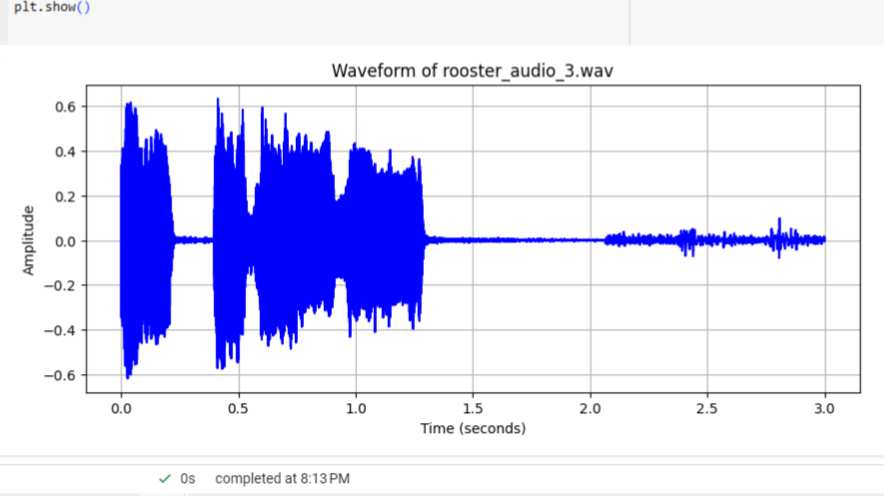
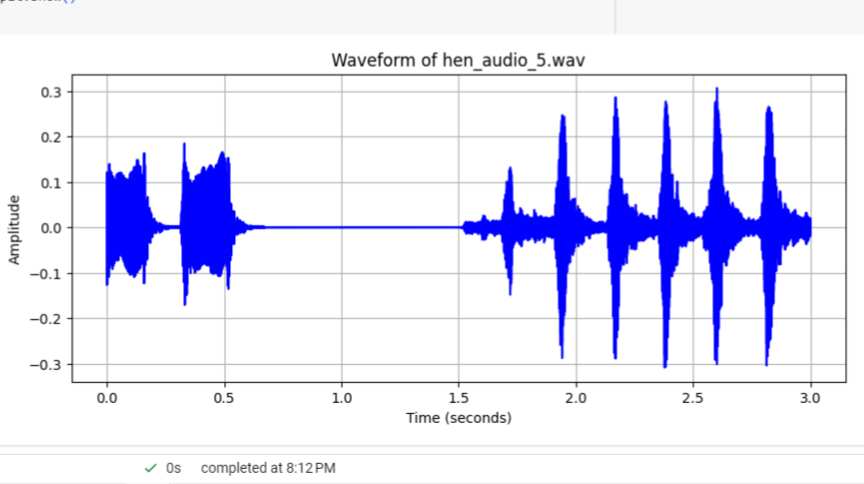
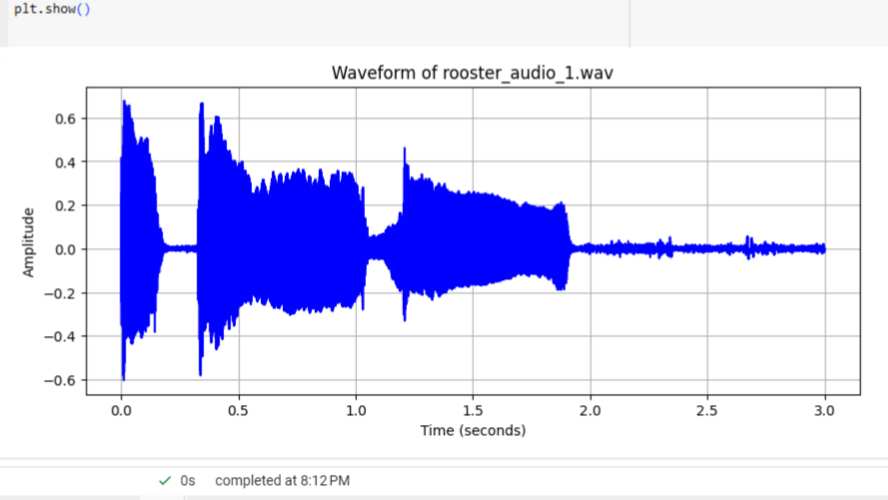
1. **Load and combine both CSV files** (chicken\_hen\_mfcc\_features.csv and chicken\_rooster\_mfcc\_features.csv).
2. **Train the Random Forest model** using the combined data.
3. **Classify new audio** and display whether it’s from a Hen or Rooster.

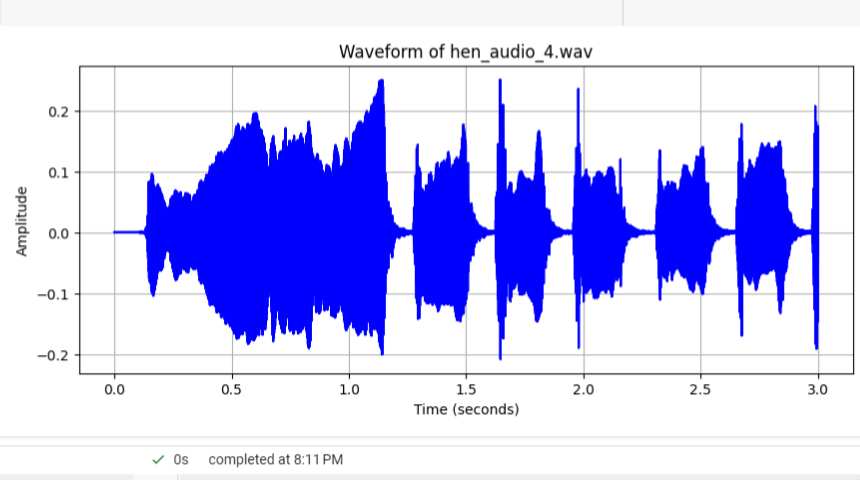
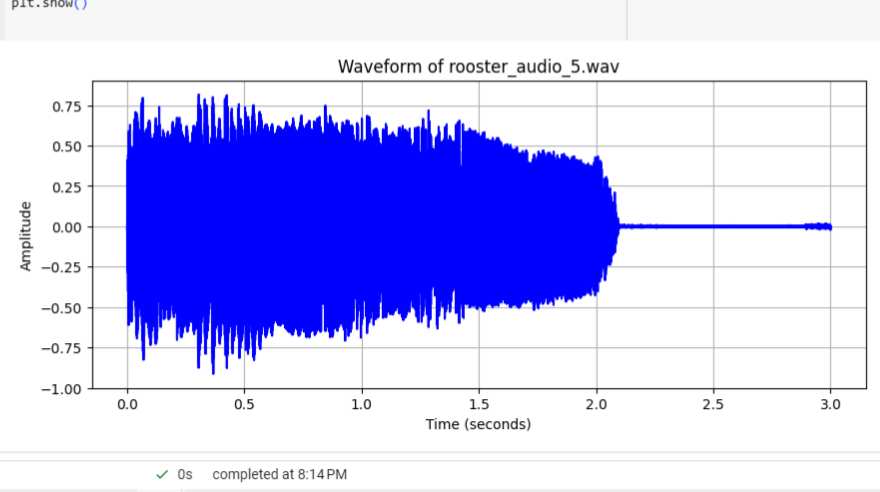
**V. Waveforms**.

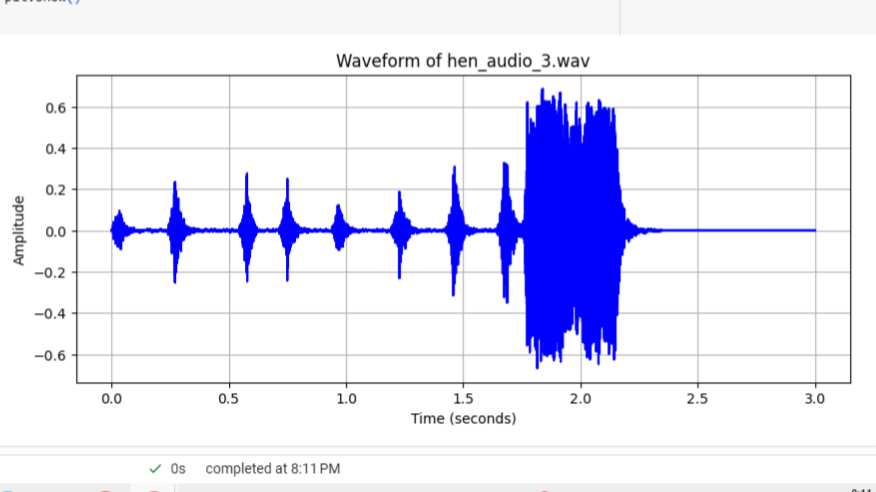
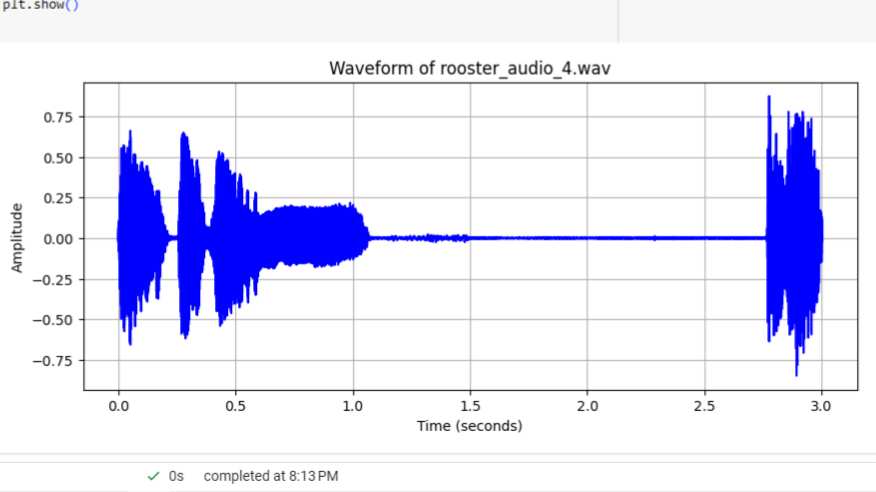


**Hen**

**Rooster**

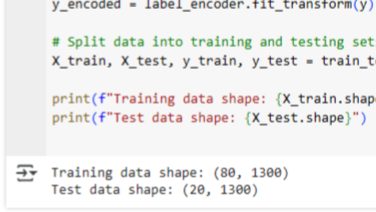
 





**VI. Results.**

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



* 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.

## 475451310_655657933483701_1381296841790867883_nFrom “Train the Random Forest Classifier

## 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.

## 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.