PYTHON SCRIPT

```
import serial
import time
import numpy as np
import pickle

# Load the trained models
with open('C:/Users/RITHIGA/Documents/aqi/models/RandomForest_regressor.pkl', 'rb') as f:
    rf_regressor = pickle.load(f)

with open('C:/Users/RITHIGA/Documents/aqi/models/RandomForest_classifier.pkl', 'rb') as f:
    rf_classifier = pickle.load(f)

with open('C:/Users/RITHIGA/Documents/aqi/models/label_encoder.pkl', 'rb') as f:
    encoder = pickle.load(f)

# Set up serial communication with Maixduino (adjust 'COM15' to your port)
ser = serial.Serial('COM15', 115200, timeout=1)  # Replace COM15 with your actual port
time.sleep(2)  # Allow time for serial connection

def get_aqi_category(aqi_class):
    return encoder.inverse_transform([aqi_class])[0]  # Convert class number back to label

while True:
    if ser.in_waiting > 0:
        data = ser.readline().decode().strip()

        if data.startswith("DATA:"):
            try:
                # Parse the sensor values (assuming the order is PM2.5, PM10, NO2, SO2, CO, O3)
                values = list(map(float, data[5:].split(',')))  # Exclude 'DATA:' prefix

                if len(values) == 6:  # Ensure we have 6 values
                    # Convert to NumPy array and reshape for the model
                    input_data = np.array(values).reshape(1, -1)

                    # Make AQI prediction
                    aqi_prediction = rf_regressor.predict(input_data)
                    aqi = int(aqi_prediction[0])  # Get the predicted AQI

                    # Predict AQI category
                    aqi_class = rf_classifier.predict(input_data)[0]
                    category = get_aqi_category(aqi_class)
```

```
            # Send back the AQI and category to Maixduino
            response = f"{aqi},{category}\n"
            ser.write(response.encode())

            # Print for debugging
            print(f"Sent to Maixduino: {response}")

        else:
            print(f"Received data does not contain 6 values: {data}")

    except Exception as e:
        print(f"Error processing data: {e}")

time.sleep(2)
```

Run code on platformIO IDE

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// Initialize LCD (I2C address 0x27 for a 16x2 LCD)
LiquidCrystal_I2C lcd(0x27, 16, 2);

// Pins for MQ sensors
const int mq7Pin = A0;     // MQ-7 CO sensor
const int mq135Pin = A1;   // MQ-135 Air Quality sensor
const int mq136Pin = A2;   // MQ-136 H2S sensor
const int methanePin = A3; // Methane sensor (MQ-4)
const int ozonePin = A4;   // Ozone sensor (MQ-131)

// Use hardware Serial2 for Nova PM sensor
#define pmsSerial Serial2

// Declare function prototype
bool readPMData(int &pm1_0, int &pm2_5, int &pm10);

void setup() {
  Serial.begin(115200);  // For Serial Monitor
  pmsSerial.begin(9600); // PMS sensor baud rate

  // Initialize LCD
  lcd.init();
  lcd.backlight();
```

```
  // Start message
  lcd.setCursor(0, 0);
  lcd.print("AQI Prediction");
  delay(2000);  // Display initial message for 2 seconds
  lcd.clear();
}

void loop() {
  // Reading analog values from MQ sensors
  int mq7Value = analogRead(mq7Pin);       // CO level
  int mq135Value = analogRead(mq135Pin);   // Air quality
  int mq136Value = analogRead(mq136Pin);   // H2S level
  int methaneValue = analogRead(methanePin); // Methane (CH4) level

  // Read values from the Nova PM sensor (PMS5003)
  int pm1_0, pm2_5, pm10;
  if (readPMData(pm1_0, pm2_5, pm10)) {
    Serial.print("PM 1.0: ");
    Serial.println(pm1_0);
    Serial.print("PM 2.5: ");
    Serial.println(pm2_5);
    Serial.print("PM 10: ");
    Serial.println(pm10);
  }

  // Display readings on Serial Monitor
  Serial.println("------------ Sensor Values ------------");
  Serial.print("MQ-7 CO: "); Serial.println(mq7Value);
  Serial.print("MQ-135 : "); Serial.println(mq135Value);
  Serial.print("MQ-136 H2S: "); Serial.println(mq136Value);
  Serial.print("Methane (CH4): "); Serial.println(methaneValue);

  // Send data format for the model:
"DATA:<mq7Value>,<mq135Value>,<mq136Value>,<pm2_5>,<pm10>,<methaneValue>;"
  Serial.print("DATA:");
  Serial.print(mq7Value); Serial.print(",");      // MQ-7 CO
  Serial.print(mq135Value); Serial.print(",");    // MQ-135 Air Quality
  Serial.print(mq136Value); Serial.print(",");    // MQ-136 H2S
  Serial.print(pm2_5); Serial.print(",");         // PM2.5
  Serial.print(pm10); Serial.print(",");          // PM10
  Serial.print(methaneValue); Serial.println(";"); // Methane (CH4)

  delay(2000);  // Delay before updating
```

```
  // Read prediction from Python script
  if (Serial.available() > 0) {
    String prediction = Serial.readStringUntil('\n');
    int separator = prediction.indexOf(',');
    String aqiValue = prediction.substring(0, separator);
    String aqiCategory = prediction.substring(separator + 1);

    // Display prediction on LCD
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("AQI: ");
    lcd.print(aqiValue);
    lcd.setCursor(0, 1);
    lcd.print("Category: ");
    lcd.print(aqiCategory);
  }
}

// Function to read data from Nova PM sensor
bool readPMData(int &pm1_0, int &pm2_5, int &pm10) {
  if (pmsSerial.available() >= 32) {
    uint8_t buffer[32];
    pmsSerial.readBytes(buffer, 32);

    if (buffer[0] == 0x42 && buffer[1] == 0x4D) {
      pm1_0 = (buffer[10] << 8) | buffer[11];  // PM 1.0
      pm2_5 = (buffer[12] << 8) | buffer[13];  // PM 2.5
      pm10  = (buffer[14] << 8) | buffer[15];  // PM 10
      return true;
    }
  }


  return false;
}
```