Source Code –

1. Arduino Source Code –

```
#include <WiFi.h>
#include <HTTPClient.h>
#include <DHT.h>
#include <Wire.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include <LiquidCrystal_I2C.h>
// #include <WiFiManager.h>

#define DHTPIN 4
#define DHTTYPE DHT11
#define SOIL_MOISTURE_PIN 34
#define SOIL_TEMP_PIN 25

DHT dht(DHTPIN, DHTTYPE);

OneWire oneWire(SOIL_TEMP_PIN);
DallasTemperature soilTemperature(&oneWire);

LiquidCrystal_I2C lcd(0x27, 16, 2);

const char* ssid = "realme 12 Pro 5G";
const char* password = "server error";

String serverUrl = "http://Prasanth23.pythonanywhere.com/predict";

String lines[9];
int currentLine = 0;
```

```
String extractStringValue(String json, String key) {
  int keyStart = json.indexOf("\"" + key + "\":");
  if (keyStart == -1) return "";

  int valueStart = json.indexOf("\"", keyStart + key.length() + 3);
  int valueEnd = json.indexOf("\"", valueStart + 1);

  if (valueStart == -1 || valueEnd == -1) return "";

  return json.substring(valueStart + 1, valueEnd);
}

void setup() {
  Serial.begin(115200);
  dht.begin();
  soilTemperature.begin();
  lcd.init();
  lcd.backlight();

  WiFi.begin(ssid, password);
  lcd.setCursor(0, 0);
  lcd.print("Connecting WiFi");

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  lcd.clear();
  lcd.print("WiFi Connected");
  delay(1000);
```

```
}

void loop() {
  float humidity = dht.readHumidity();
  float temperature = dht.readTemperature();
  int soilMoistureRaw = analogRead(SOIL_MOISTURE_PIN);
  float soilMoisture = soilMoistureRaw;
  soilTemperature.requestTemperatures();
  float soilTemp = soilTemperature.getTempCByIndex(0);

  if (isnan(humidity) | | isnan(temperature)) {
    Serial.println("Sensor read error!");
    lcd.clear();
    lcd.print("Sensor Error!");
    delay(2000);
    return;
  }

  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;
    WiFiClient client;

    http.begin(client, serverUrl);
    http.addHeader("Content-Type", "application/json");

    String json = "{\"soil_moisture\":" + String(soilMoisture, 2) +
                  ",\"ambient_temp\":" + String(temperature, 2) +
                  ",\"humidity\":" + String(humidity, 2) +
                  ",\"soil_temp\":" + String(soilTemp, 2) + "}";
    int httpResponseCode = http.POST(json);

    if (httpResponseCode > 0) {
      String response = http.getString();
      Serial.println("Response:");
      Serial.println(response);

      int pH_idx = response.indexOf("\"pH\":");
      int EC_idx = response.indexOf("\"EC\":");
      int soil_idx = response.indexOf("\"soil_type\":\"");
      int crop_idx = response.indexOf("\"suitable_crop\":\"");

      float pH = response.substring(response.indexOf("\"pH\":") + 5, response.indexOf(",", response.indexOf("\"pH\":"))).toFloat();
      float EC = response.substring(response.indexOf("\"EC\":") + 5, response.indexOf(",", response.indexOf("\"EC\":"))).toFloat();
      String soilType = extractStringValue(response, "soil_type");
      String crop = extractStringValue(response, "suitable_crop");

      lines[0] = "Moisture: " + String(soilMoisture, 1);
      lines[1] = "Amb Temp: " + String(temperature, 1) + " C";
      lines[2] = "Humidity: " + String(humidity, 1) + " %";
      lines[3] = "Soil Temp: " + String(soilTemp, 1) + "C";
      lines[4] = "pH: " + String(pH, 2);
      lines[5] = "EC: " + String(EC, 0);
      lines[6] = "Soil: " + soilType;
      lines[7] = "Crop: " + crop;
      lines[8] = "Reading next...";

    } else {
      Serial.print("HTTP Error: ");
      Serial.println(httpResponseCode);
      lines[0] = "Server Error";
    }

    http.end();
  } else {
    Serial.println("WiFi Disconnected");
    lines[0] = "WiFi Lost!";
  }

  for (int i = 0; i < 9; i++) {
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print(lines[i].substring(0, 16));
    delay(2500);
  }

  delay(1000);
}
```

2. Data Preprocessing and Model Training – Python Source Code –

```
import pandas as pd
import joblib
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestClassifier

df = pd.read_csv("Soil_Parameter_ML_Dataset_200.csv")

X = df[['soil_moisture', 'ambient_temp', 'humidity', 'soil_temp']]

# Outputs to predict
y_ph = df['pH']
```

```
y_ec = df['EC']
y_soil_type = df['soil_type']
y_crop = df['suitable_crop']

# Train regression models
ph_model = LinearRegression()
ec_model = LinearRegression()

ph_model.fit(X, y_ph)
ec_model.fit(X, y_ec)

joblib.dump(ph_model, 'pH_model.pkl')
joblib.dump(ec_model, 'EC_model.pkl')

# Train classification models for soil type and crop
soil_type_model = RandomForestClassifier()
crop_model = RandomForestClassifier()

soil_type_model.fit(X, y_soil_type)
crop_model.fit(X, y_crop)

joblib.dump(soil_type_model, 'soil_type_model.pkl')
joblib.dump(crop_model, 'crop_model.pkl')
```

3. Flask Server Backend - Python Source Code –

```
from flask import Flask, request, jsonify
from flask_cors import CORS
import joblib
import numpy as np
import pandas as pd
import os
```

```
# Start Flask app
app = Flask(__name__)
CORS(app)

# Load trained models
ph_model = joblib.load('pH_model.pkl')
ec_model = joblib.load('EC_model.pkl')
soil_type_model = joblib.load('soil_type_model.pkl')
crop_model = joblib.load('crop_model.pkl')

# Global dictionary to store last prediction
latest_data = {}

# Path to live log CSV file
LOG_FILE = "live_data_log.csv"

@app.route('/predict', methods=['POST'])
def predict():
   global latest_data
   data = request.get_json()

   try:
      # Extract input sensor values
      soil_moisture = float(data['soil_moisture'])
      ambient_temp = float(data['ambient_temp'])
      humidity = float(data['humidity'])
      soil_temp = float(data['soil_temp'])

      input_data = [[soil_moisture, ambient_temp, humidity, soil_temp]]
```

```
        # Predict
        pH = ph_model.predict(input_data)[0]
        EC = ec_model.predict(input_data)[0]
        soil_type = soil_type_model.predict(input_data)[0]
        crop = crop_model.predict(input_data)[0]

        latest_data = {
            'soil_moisture': soil_moisture,
            'ambient_temp': ambient_temp,
            'humidity': humidity,
            'soil_temp': soil_temp,
            'pH': round(pH, 2),
            'EC': round(EC, 2),
            'soil_type': soil_type,
            'suitable_crop': crop
        }

        # ---- Logging part ----
        df_log = pd.DataFrame([latest_data])
        if os.path.exists(LOG_FILE):
            df_log.to_csv(LOG_FILE, mode='a', header=False, index=False)
        else:
            df_log.to_csv(LOG_FILE, index=False)

        return jsonify(latest_data)

    except Exception as e:
        return jsonify({'error': str(e)})

@app.route('/latest', methods=['GET'])
def get_latest():
    if latest_data:
        return jsonify(latest_data)
    else:
        return jsonify({'message': 'No data received yet'}), 404

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

4. Frontend (Web Interface) –

HTML Source Code –

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Monitoring Dashboard</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="container">
        <h1>Monitoring Dashboard</h1>
        <div class="card">
            <h2>Sensor Readings</h2>
            <p><strong>Soil Moisture :</strong> <span id="moisture">--</span></p>
            <p><strong>Ambient Temperature :</strong> <span id="temperature">--</span></p>
            <p><strong>Humidity :</strong> <span id="humidity">--</span></p>
            <p><strong>Soil Temperature :</strong> <span id="soil_temp">--</span></p>
        </div>

        <div class="card">
            <h2>Predicted Parameters</h2>
```

```
      <p><strong>pH :</strong> <span id="ph">--</span></p>
      <p><strong>EC :</strong> <span id="ec">--</span></p>
      <p><strong>Soil Type :</strong> <span id="soil_type">--</span></p>
      <p><strong>Suitable Crop :</strong> <span id="crop">--</span></p>
    </div>
  </div>

  <script src="script.js"></script>
</body>
</html>
```

CSS Source Code –

```
body {
 font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
 background: #f0f4f7;
 margin: 0;
 padding: 0;
 display: flex;
 justify-content: center;
 align-items: flex-start;
 min-height: 100vh;
}

.container {
 width: 90%;
 max-width: 600px;
 margin-top: 30px;
 background: #ffffff;
 padding: 20px 30px;
 border-radius: 10px;
 box-shadow: 0 6px 12px rgba(0, 0, 0, 0.1);
```

```css
}

h1 {
  text-align: center;
  margin-bottom: 25px;
  color: #2e7d32;
}

.card {
  background: #e8f5e9;
  padding: 15px 20px;
  margin-bottom: 20px;
  border-radius: 8px;
  box-shadow: inset 0 0 5px rgba(0, 0, 0, 0.05);
}

.card h2 {
  margin-top: 0;
  color: #1b5e20;
}

.card p {
  font-size: 16px;
  margin: 8px 0;
}
```

JavaScript Source Code –

```javascript
async function fetchData() {
  try {
    const response = await fetch("https://Prasanth23.pythonanywhere.com/latest");
    const data = await response.json();
    document.getElementById("moisture").innerText = data.soil_moisture;

    document.getElementById("temperature").innerText = data.ambient_temp + " °C";

    document.getElementById("humidity").innerText = data.humidity + " %";

    document.getElementById("soil_temp").innerText = data.soil_temp + " °C";


    document.getElementById("ph").innerText = data.pH;

    document.getElementById("ec").innerText = data.EC;

    document.getElementById("soil_type").innerText = data.soil_type;

    document.getElementById("crop").innerText = data.suitable_crop;

  } catch (err) {

    console.error("Error fetching data:", err);

  }

}


setInterval(fetchData, 10000);

fetchData();
```