Table of Contents

Question 1

1.0 Design Rationale: Strengths and Weaknesses of the Final IoT System Design

The GrowSync system represents a comprehensive and well-structured IoT solution tailored for precision agriculture. Its design aims to enhance agricultural productivity by monitoring environmental conditions and automating key farm operations through an integrated sensor-actuator-cloud architecture. While the system showcases strong technical competencies in wireless communication, system integration, and user functionality, it also encounters several limitations, particularly in power management, bandwidth usage, and operational flexibility. The following section critically evaluates three main strengths and three weaknesses of the system, based on wireless capability, functionality, interoperability, power management, bandwidth, and device handling.

1.1 Strengths

1.1.1 Wireless Capability

The GrowSync system effectively leverages a multi-layered wireless communication architecture, incorporating ZigBee, LoRaWAN, and Wi-Fi protocols for short, medium, and long-distance data transmission respectively. This diversified communication strategy allows the system to operate efficiently across different spatial configurations, from small greenhouses to large open-field farms. ZigBee is used for high-frequency, low-power sensor readings within close proximity up to 100 meters, while LoRaWAN extends coverage to remote zones, supporting data transmission over several kilometers with extremely low energy usage. (Singh et al., 2020)

This tiered wireless approach not only increases scalability but also ensures data resilience in low-connectivity environments. The Dragino LSPH01 pH sensor, for instance, supports LoRaWAN and can transmit data for up to five years using an 8500mAh Li-$SOCl_2$ battery, demonstrating the system's capability to sustain long-term operations with minimal maintenance. (Dragino, 2025) Moreover, Wi-Fi enables real-time uplinks to cloud platforms via the ESP32 microcontroller, completing the end-to-end data flow from field sensors to the user's mobile application.

1.1.2 Functionality

Functionality is another key strength of the GrowSync system. The design integrates a wide array of sensors, including DHT22 for temperature and humidity, MQ135 for air quality, LSPH01 for soil pH, PIR sensors for pest detection, and GPS for spatial awareness, which collectively ensure a holistic understanding of the environmental conditions. These sensors are complemented by actuators such as misting pumps, ventilation fans, solenoid valves, and pH regulation pumps to perform automated responses based on real-time data inputs.

This closed-loop feedback mechanism supports intelligent automation. For example, when the air quality sensor, MQ135, detects harmful gases like ammonia or CO_2 beyond safe thresholds (Quartz, 2022), the system activates the ventilation fan and sends alerts via the mobile app. Similarly, if the soil pH deviates from optimal ranges, the appropriate dosing pump is activated to correct the soil chemistry. These features reduce the need for manual intervention and ensure optimal crop health, positioning GrowSync as a practical tool for modern precision agriculture.

1.1.3 Interoperability

Interoperability is at the heart of GrowSync's seamless integration across devices, platforms, and interfaces. The system supports widely accepted communication protocols, including MQTT for device-cloud communication and HTTPS for cloud-app communication (Dizdarević et al., 2019), both encrypted using Transport Layer Security (TLS) to ensure secure and reliable data exchange. The use of MQTT over Wi-Fi allows for lightweight, low-bandwidth messaging with publish-subscribe architecture, which suits IoT environments with frequent sensor updates. (Hsu, 2024)

Furthermore, GrowSync integrates with Amazon Web Services (AWS) for cloud data management and Agroview for advanced AI-based analytics and crop health monitoring. AWS components such as IoT Core, Timestream, RDS, and S3 are used to handle device authentication, time-series data, relational data, and log storage respectively. This high level of interoperability ensures scalability, enables predictive insights, and enhances the system's potential to interconnect with third-party tools and platforms in the future. (AWS, 2016)

1.2 Weaknesses

1.2.1 Power Management Limitations

Despite incorporating low-power components, the overall energy demand of GrowSync remains a concern, especially in off-grid scenarios. Components including the ESP32 microcontroller and high-power actuators, for example, 12V DC ventilation fans and submersible pumps, require continuous power supply to function optimally. While individual sensors such as the LSPH01 offer ultra-low-power operation, the combination of Wi-Fi-based gateways and electromechanical devices significantly increases the system's average power consumption.

In remote agricultural settings without a direct power source, reliance on large batteries or solar panels becomes necessary, which adds to installation and maintenance costs. Moreover, frequent actuator switching such as misting systems during hot hours, can lead to high energy draw, making the system less suitable for small-scale or energy-constrained farms without supplementary energy optimization strategies. (Fidelis & Idim, 2020)

1.2.2 Bandwidth Constraints and Latency

Another major limitation is the trade-offs between range, bandwidth, and latency. While LoRaWAN is noted for its energy efficiency and large range, it is limited by data rates, usually ranging from 0.3 to 50 kbps, accompanied by a latency of around 10 seconds. This makes the technology suitable only for intermittent transmission of low-volume sensor data (Jawad et al., 2017). These limitations by nature restrains the system's scalability for usage where high-throughput or near-real-time feedback is required, such as rapid pest detection alerts or immediate environmental control actions.

Even though Wi-Fi handles greater bandwidth operations at the gateway level, the whole communication channel is subject to variable latencies, especially if packet retransmission happens due to signal interference or blockage. Such a situation might seriously impact actuator control timing precision or user notification periods in emergency scenarios, for example, during pest invasions or gas leaks.

1.2.3 Device Categorization and Boot Resilience

A further challenge in the GrowSync system is that the architecture uses a static classification approach for devices, classifying modules into different layers such as perception, gateway, actuation, and application. While the structure allows for a complete architectural understanding, it also introduces a level of stiffness. For instance, the addition of new sensors in different other agricultural areas requires manual coding adjustments and consequent redeployment, thus limiting scalability and flexibility to on-ground changes. Besides, the lack of dynamic reconfigurability or plug-and-play device support constrains timely deployment and efficient troubleshooting during real-world field operations. (Gubbi et al., 2013)

Additionally, the dependency of the GrowSync system on network availability and cloud integration adversely affects the resilience of the system at booting. While instant booting is evident in systems such as the Arduino Mega and the ESP32, the overall system is not operational until a proper reconnection with AWS cloud services is established. Eventually, in the case of power or network outages, the farm automation process may be paused or delayed until all connections are restored, posing potential risks to crops in sensitive environments. (Rejeb et al., 2019)

1.3 Summary

In conclusion, the final design of GrowSync presents a variety of benefits within the domain of wireless, intelligent, and automated farms. It is unique in its wireless capacity, feature-richness, and connectivity to capable cloud platforms. However, performance and scaling may be compromised due to the challenges of power consumption, bandwidth, and the limited adaptability of devices for more complicated or resource-limited environments. As mentioned, and for further iterations of the technology to develop the full potential of IoT in agriculture, the ability to mitigate these weaknesses through energy harvesting, dynamic reconfiguration, or local fallback will be important. Therefore, addressing these limitations can improve system performance, ultimately making the system more applicable for broader use cases and agricultural contexts. In the long term, with more refinement, GrowSync could become a scalable model for future smart farming technology.

2.0 Alternative Designs Ideas

In the development of GrowSync, a total of three alternative design strategies were considered. Each offered potential advantages but was ultimately rejected due to limitations that conflicted with the project's vision, scalability goals, or operational constraints.

2.1 Direct Sensor-to-Cloud vs. Central Gateway

The first idea that was rejected was for each sensor node to connect to the cloud via LoRa or WIFI while running its own MQTT client. This proposed idea would make data flow more decentralized and stable, as if one node failed, the entire system would not fail, and the other nodes could still communicate independently. Despite its benefits, there are some drawbacks to this. This idea requires the use of a separate microcontroller and radio hardware for each sensor, which makes the system more costly and contradicts our goal of creating a cost-effective agricultural system. As previously stated for our system's security features, there will be scheduled firmware updates, and updating each node individually may be complex and time-consuming.

As a result, GrowSync implements a centralized architecture to collect data from multiple sensor nodes, with ESP32 and Arduino Mega 256 serving as a fallback solution if WIFI is unavailable. The gateways then handle the secure connection to the cloud. This solution reduces the overall number of nodes that require internet connectivity, simplifies firmware management, and reduces energy consumption per node. Furthermore, because of the centralized architecture, this system will be easier to scale to larger farms as more nodes are added to the central main microcontroller.

2.2 Cellular (NB-IoT/LoRa vs. GSM) Connectivity

Initially, NB-IoT and Cat-M1 were considered for GrowSync's connectivity between microcontroller and sensors, however, while these technologies can provide a significant amount of coverage with no range issues, even in remote rural areas, NB-IoT modules and subscriptions can increase the cost and power, which does not align with GrowSync's goal of being low cost and inexpensive (He, 2024). As a result, LoRaWAN does not require subscriptions and can reduce costs while remaining energy efficient.

As a result, LoRaWAN and Wi-Fi were chosen, as previously stated. LoRaWAN is low-power and provides a longer range of communication with no recurring fees, making it ideal for the sensors used in GrowSync's architecture. LoRaWAN has low data latency and is vulnerable to environmental interference, but these limitations are acceptable given the device's low cost (He, 2024). However this can be solved during the machine learning and data preprocessing phase where any anomalies or missing data can be removed and interpolated.

2.3 Cloud Platform Choice

Last but not least, ThingSpeak and ONENET, which are alternative IoT clouds to AWS clouds, were also considered for GrowSync's architecture. These platforms are inexpensive and simpler, but they only provide basic data storage and visualization, making them unsuitable for our large-scale agriculture IoT system, which requires more detailed data visualization, analytical capabilities, security, and interconnectivity for integration. These other IoT clouds are incompatible with our goal of providing users with real-time and predictive data visualization/output for their agriculture zones.

The idea for GrowSync requires more than just data visualization, it also requires historical data storage for prediction, pest detection, mobile application alerts, and automated responses. As a result, Agroview, AWS IoT Core, Timestream, and S3 were chosen because they are a powerful platform for processing large volumes of data to monitor plant health and provide a scalable and secure environment. Due to these platforms being AWS-based, they are easily integrated with one another. Even though AWS is known to be more expensive than the other options, clouds are a pay-as-you-go platform that is easily scalable and charges only for the specific resources you actively use, based on consumption or time, rather than for pre-allocated capacity. So, it still aligns with our goals by being low cost, while providing the necessary real-time data visualization and prediction.

In conclusion, each alternative design was evaluated based on technical feasibility, cost, and operational impact. Overall, the final chosen design strikes a balance where it maximizes sensing accuracy, automation capabilities and security while keeping component costs low and power consumption under control.

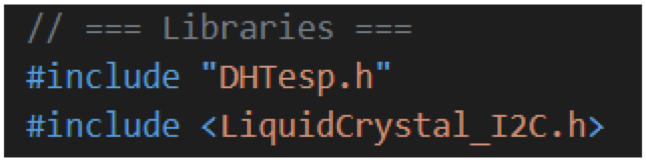
Question 2

3.0 Proof of Concept

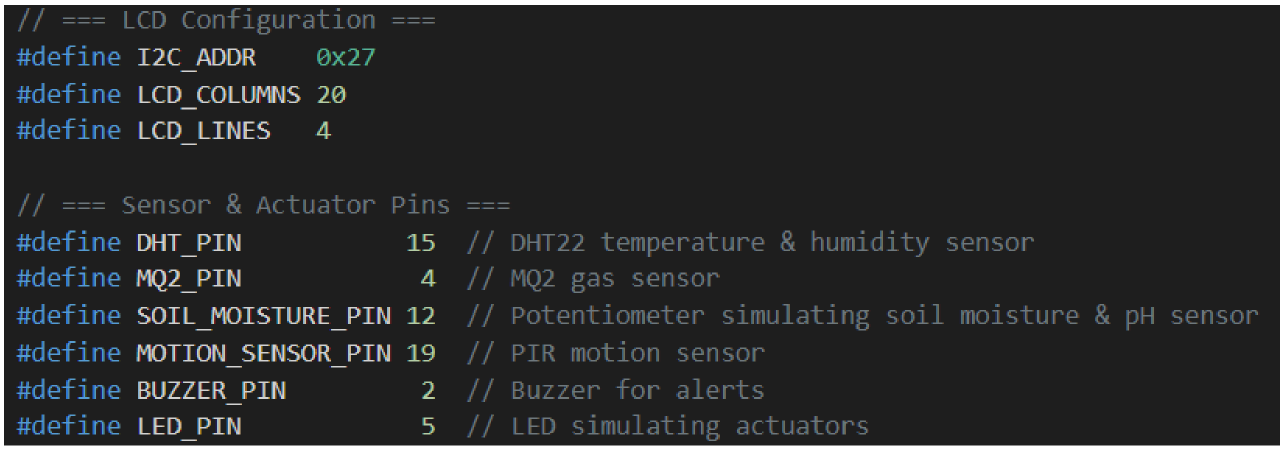
This section provides the proof of concepts developed for GrowSync which includes a Wokwi simulation, a Google Colab machine learning notebook and Figma Prototype of the mobile application. These three proofs of concept provide a better understanding and visualization of GrowSync.

3.1 Wokwi

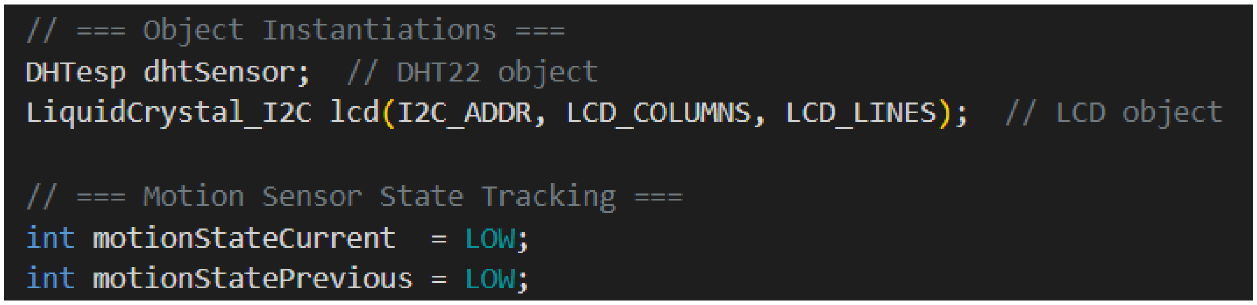
Link : https://wokwi.com/projects/436375068894299137


```
// === Libraries ===
#include "DHTesp.h"
#include <LiquidCrystal_I2C.h>
```

The libraries used are “DHTesp.h” for reading data from the DHT22 temperature and humidity sensor and “LiquidCrystal_I2C.h” for controlling the 20x4 I2C LCD display.

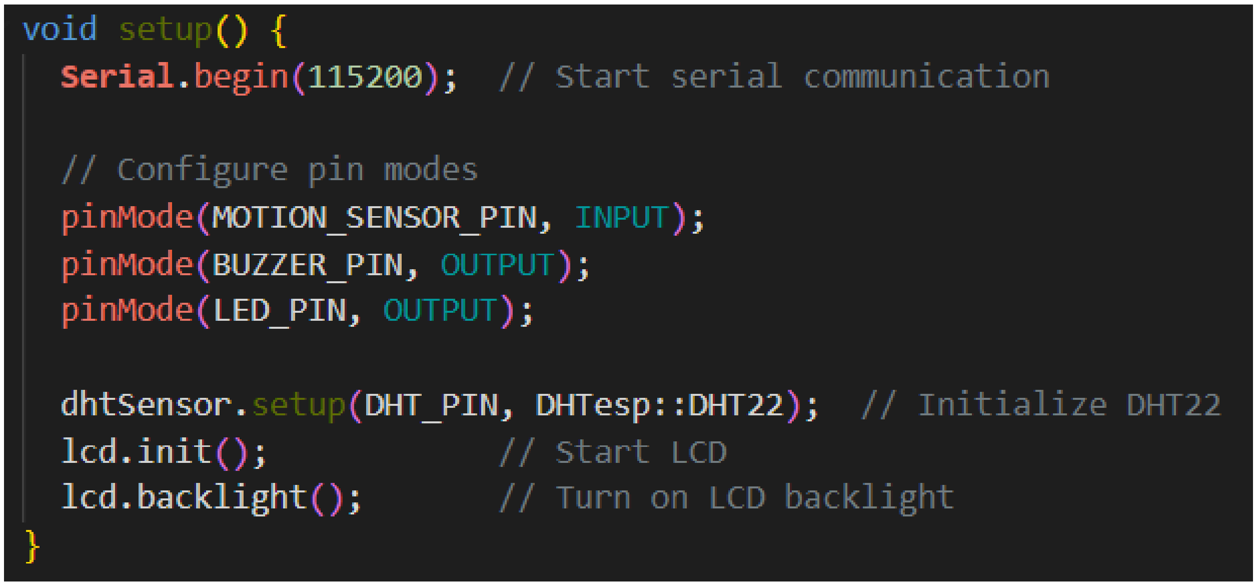

```
// === LCD Configuration ===
#define I2C_ADDR    0x27
#define LCD_COLUMNS 20
#define LCD_LINES   4

// === Sensor & Actuator Pins ===
#define DHT_PIN           15  // DHT22 temperature & humidity sensor
#define MQ2_PIN            4  // MQ2 gas sensor
#define SOIL_MOISTURE_PIN 12  // Potentiometer simulating soil moisture & pH sensor
#define MOTION_SENSOR_PIN 19  // PIR motion sensor
#define BUZZER_PIN         2  // Buzzer for alerts
#define LED_PIN            5  // LED simulating actuators
```

These two code blocks define the hardware pin connections for all sensors such as the DHT22, MQ2, PIR sensor and set the LCD I2C address to 0 x 27 with dimensions of 20 columns x 4 rows.



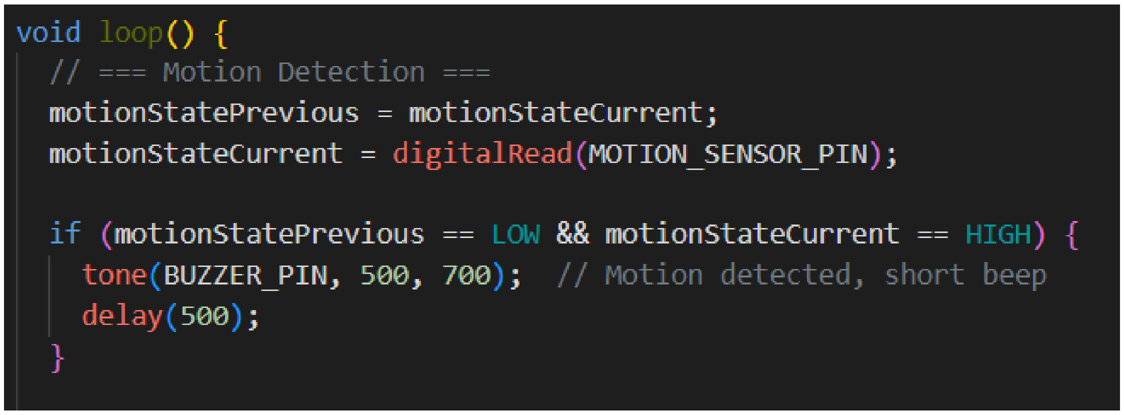
```
// === Object Instantiations ===
DHTesp dhtSensor;  // DHT22 object
LiquidCrystal_I2C lcd(I2C_ADDR, LCD_COLUMNS, LCD_LINES);  // LCD object

// === Motion Sensor State Tracking ===
int motionStateCurrent  = LOW;
int motionStatePrevious = LOW;
```

These two code blocks create objects for the DHT sensor (dhtSensor) and LCD (lcd) and initialize two variables to track the current and previous state of the motion sensor for change detection.



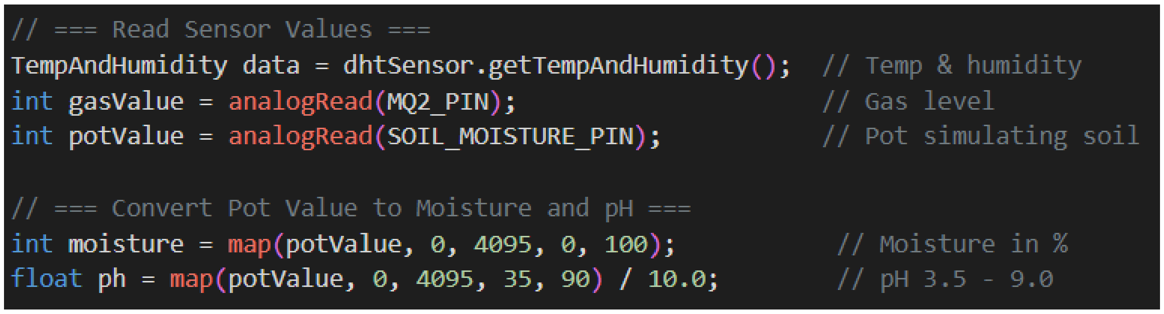
```
void setup() {
  Serial.begin(115200);  // Start serial communication

  // Configure pin modes
  pinMode(MOTION_SENSOR_PIN, INPUT);
  pinMode(BUZZER_PIN, OUTPUT);
  pinMode(LED_PIN, OUTPUT);

  dhtSensor.setup(DHT_PIN, DHTesp::DHT22);  // Initialize DHT22
  lcd.init();              // Start LCD
  lcd.backlight();         // Turn on LCD backlight
}
```

The “void setup()” initializes serial communication for debugging via “Serial.begin(115200)”. It also sets input/output modes for sensors and actuators. Lastly, it also starts the DHT22 and LCD operations.



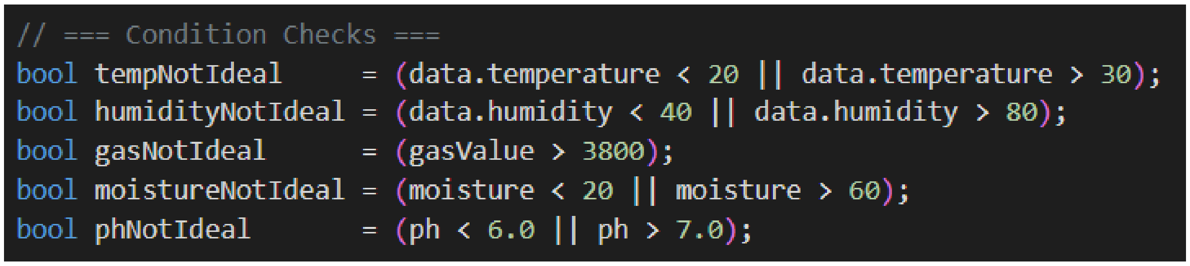
```
void loop() {
  // === Motion Detection ===
  motionStatePrevious = motionStateCurrent;
  motionStateCurrent = digitalRead(MOTION_SENSOR_PIN);

  if (motionStatePrevious == LOW && motionStateCurrent == HIGH) {
    tone(BUZZER_PIN, 500, 700);  // Motion detected, short beep
    delay(500);
  }
```

Then, we get into the "void loop()" code blocks and all the remaining code blocks will be located in this loop. The first part checks for motion using the PIR sensor and if a motion event is detected meaning LOW change to HIGH transition, the buzzer is triggered with a short beep of 500 Hz for 700 ms to indicate movement.



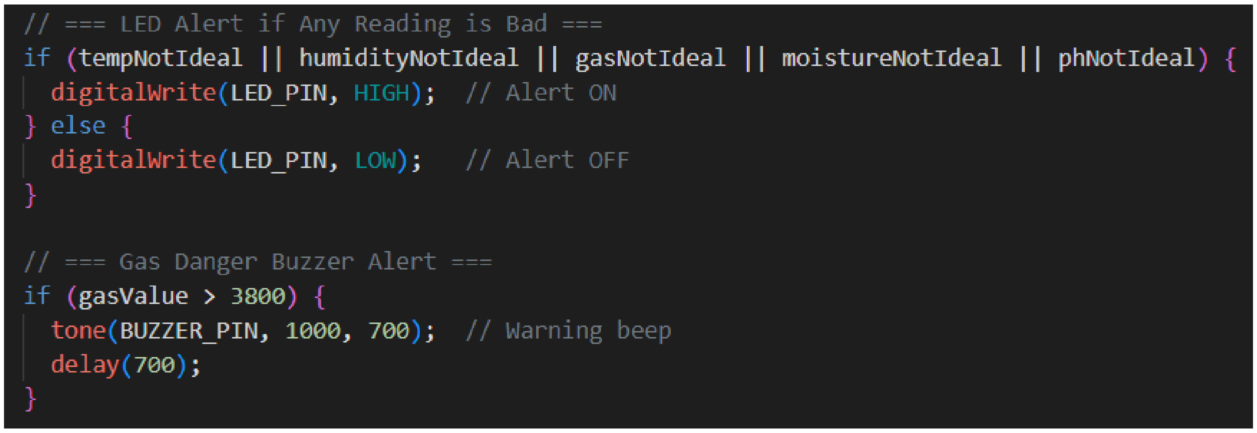
```
// === Read Sensor Values ===
TempAndHumidity data = dhtSensor.getTempAndHumidity();  // Temp & humidity
int gasValue = analogRead(MQ2_PIN);                     // Gas level
int potValue = analogRead(SOIL_MOISTURE_PIN);           // Pot simulating soil

// === Convert Pot Value to Moisture and pH ===
int moisture = map(potValue, 0, 4095, 0, 100);          // Moisture in %
float ph = map(potValue, 0, 4095, 35, 90) / 10.0;       // pH 3.5 - 9.0
```

In this section, the ESP32 reads the temperature & humidity from the DHT22 sensor, the gas level from the MQ2 sensor and soil moisture & pH from a potentiometer simulating the soil moisture and pH sensor. It then converts potentiometer value into moisture percentage from 0 to 100% and pH level from 3.5 to 9.0.



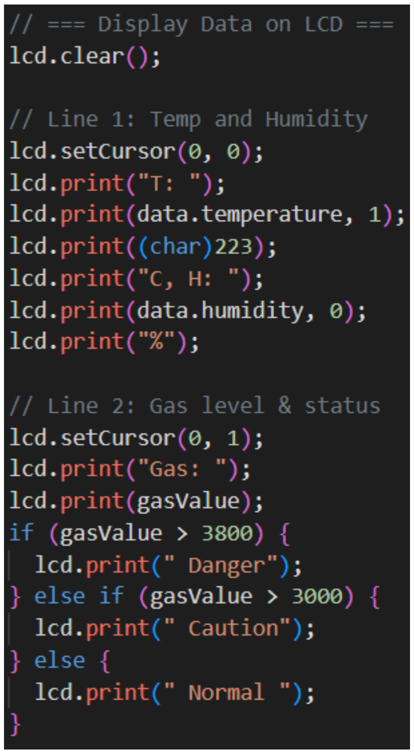
```
// === Condition Checks ===
bool tempNotIdeal     = (data.temperature < 20 || data.temperature > 30);
bool humidityNotIdeal = (data.humidity < 40 || data.humidity > 80);
bool gasNotIdeal      = (gasValue > 3800);
bool moistureNotIdeal = (moisture < 20 || moisture > 60);
bool phNotIdeal       = (ph < 6.0 || ph > 7.0);
```

Each sensor reading is compared to predefined ideal thresholds. If any value is outside its ideal range, a corresponding flag is set to indicate the issue.

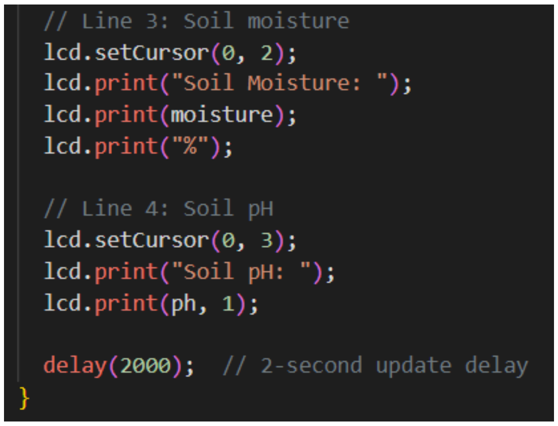


```
// === LED Alert if Any Reading is Bad ===
if (tempNotIdeal || humidityNotIdeal || gasNotIdeal || moistureNotIdeal || phNotIdeal) {
  digitalWrite(LED_PIN, HIGH);  // Alert ON
} else {
  digitalWrite(LED_PIN, LOW);   // Alert OFF
}

// === Gas Danger Buzzer Alert ===
if (gasValue > 3800) {
  tone(BUZZER_PIN, 1000, 700);  // Warning beep
  delay(700);
}
```

If any reading is not ideal, the LED turns on to represent automatic actuator activation such as fan, pump and valve. Besides, if the gas level is dangerously high as of more than 3800, a high-pitched buzzer is activated with 1000 Hz for 700 ms to warn local users.



```
// === Display Data on LCD ===
lcd.clear();

// Line 1: Temp and Humidity
lcd.setCursor(0, 0);
lcd.print("T: ");
lcd.print(data.temperature, 1);
lcd.print((char)223);
lcd.print("C, H: ");
lcd.print(data.humidity, 0);
lcd.print("%");

// Line 2: Gas level & status
lcd.setCursor(0, 1);
lcd.print("Gas: ");
lcd.print(gasValue);
if (gasValue > 3800) {
  lcd.print(" Danger");
} else if (gasValue > 3000) {
  lcd.print(" Caution");
} else {
  lcd.print(" Normal ");
}
```



```
  // Line 3: Soil moisture
  lcd.setCursor(0, 2);
  lcd.print("Soil Moisture: ");
  lcd.print(moisture);
  lcd.print("%");

  // Line 4: Soil pH
  lcd.setCursor(0, 3);
  lcd.print("Soil pH: ");
  lcd.print(ph, 1);

  delay(2000);  // 2-second update delay
}
```

These two figures show the code that displays temperature and humidity in the first line, gas level and status like “Danger”, “Caution” and “Normal” in the second line, soil moisture in the third line and pH in the fourth line. Lastly, a “delay(2000)” pauses execution for 2 seconds before the loop restarts to prevent excessive updates.

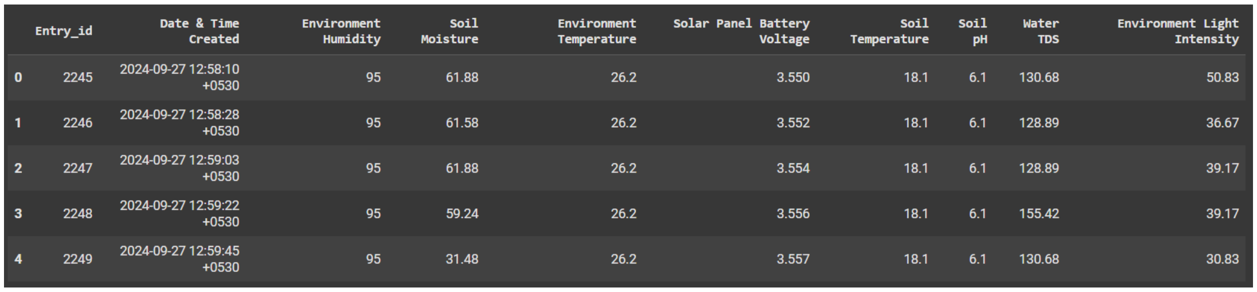
3.2 Google Colab

Link :
https://colab.research.google.com/drive/1Et2ByU6MqD0RVMDGOpqZU3JS1AxHKYqc?usp=sharing

Two machine learning functionalities of the mobile application are demonstrated using Google Colab, which are sensor data visualization and crop yield prediction.

3.2.1 Sensor Data Visualization

The sensor data dataset used for this section is obtained from Mendeley Data, which is funded by La Trobe University (DY, 2024). The dataset contains sensor data on environmental temperature, environmental humidity, environmental light intensity, solar panel battery voltage, soil moisture, soil temperature, soil humidity, soil pH, and water TDS, which are each separated into different excel files. Each row of sensor data is accompanied with a timestamp indicating the time of collection to allow time-series data analysis. Some of the sensor data shown in this section are not collected by the GrowSync system, and are only used for demonstration purposes.



	Entry_id	Date & Time Created	Environment Humidity	Soil Moisture	Environment Temperature	Solar Panel Battery Voltage	Soil Temperature	Soil pH	Water TDS	Environment Light Intensity
0	2245	2024-09-27 12:58:10 +0530	95	61.88	26.2	3.550	18.1	6.1	130.68	50.83
1	2246	2024-09-27 12:58:28 +0530	95	61.58	26.2	3.552	18.1	6.1	128.89	36.67
2	2247	2024-09-27 12:59:03 +0530	95	61.88	26.2	3.554	18.1	6.1	128.89	39.17
3	2248	2024-09-27 12:59:22 +0530	95	59.24	26.2	3.556	18.1	6.1	155.42	39.17
4	2249	2024-09-27 12:59:45 +0530	95	31.48	26.2	3.557	18.1	6.1	130.68	30.83

Figure ? Dataset Overview

The dataset is downloaded and merged together using common identifiers, which are *Entry_id* and *Date & Time Created* respectively. Figure ??? shows an initial overview of the merged dataset.

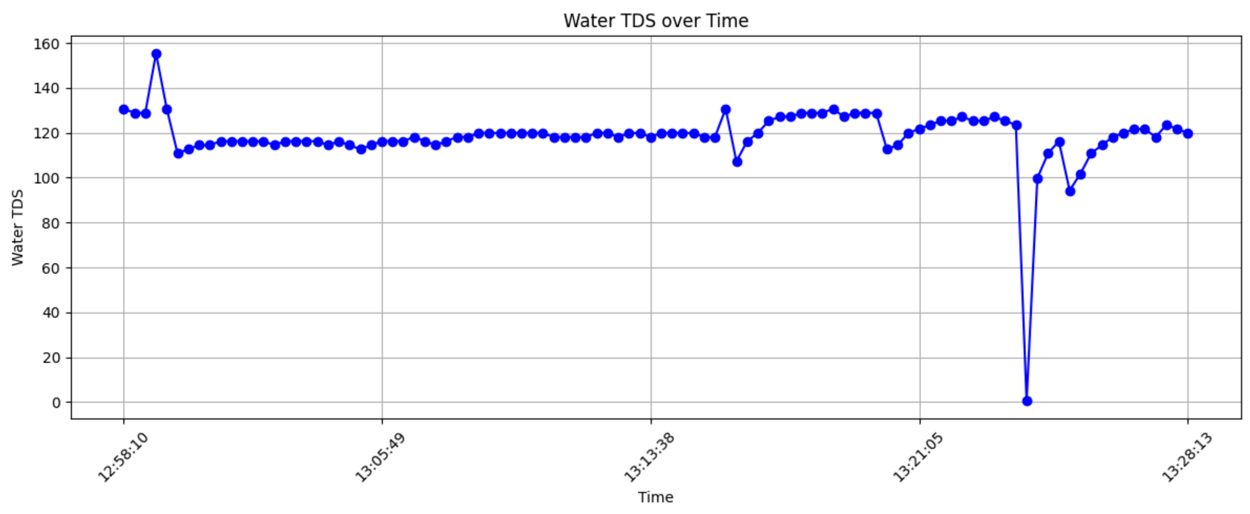


Figure ? Water TDS Visualization With Anomalies

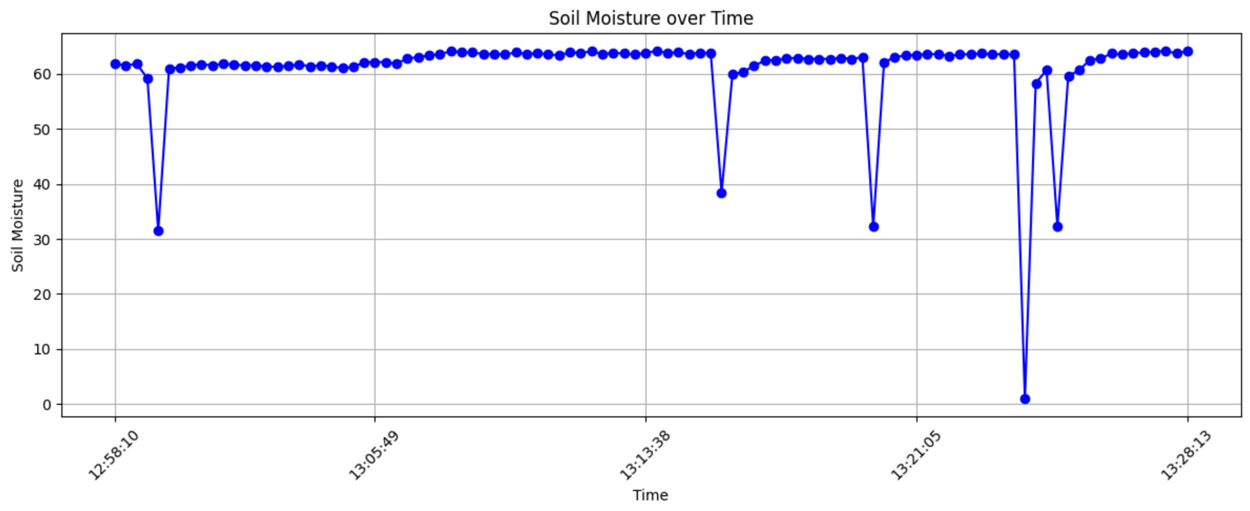
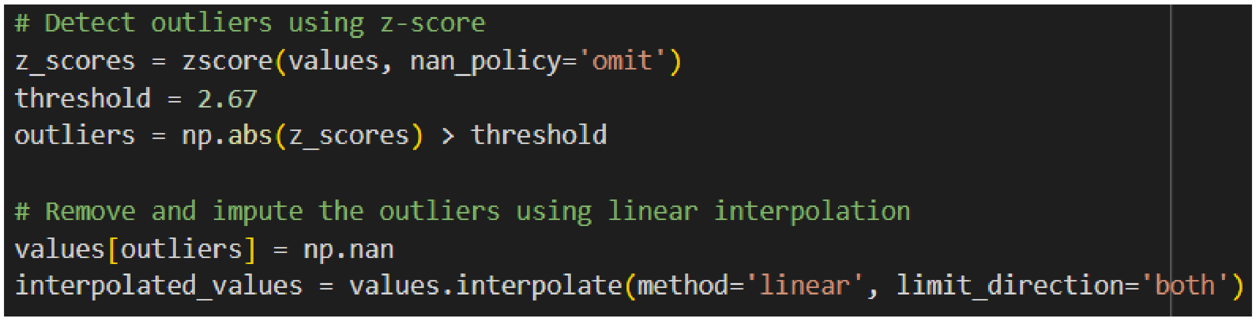


Figure ? Soil Moisture Visualization With Anomalies

Figure ??? and ??? shows some examples of time-series data, namely water TDS and soil moisture. It is observed that these two sensor data contains anomalies which stray far away from the normal trend, which will be removed before being presented to the user through the mobile application.



```
# Detect outliers using z-score
z_scores = zscore(values, nan_policy='omit')
threshold = 2.67
outliers = np.abs(z_scores) > threshold

# Remove and impute the outliers using linear interpolation
values[outliers] = np.nan
interpolated_values = values.interpolate(method='linear', limit_direction='both')
```

Figure ? Code Snippet of Outlier Removal and Imputation

The outliers are detected using a z-score threshold of 2.67, which are then removed and imputed using linear interpolation, ensuring that the imputed values are following the trend of the actual sensor data.

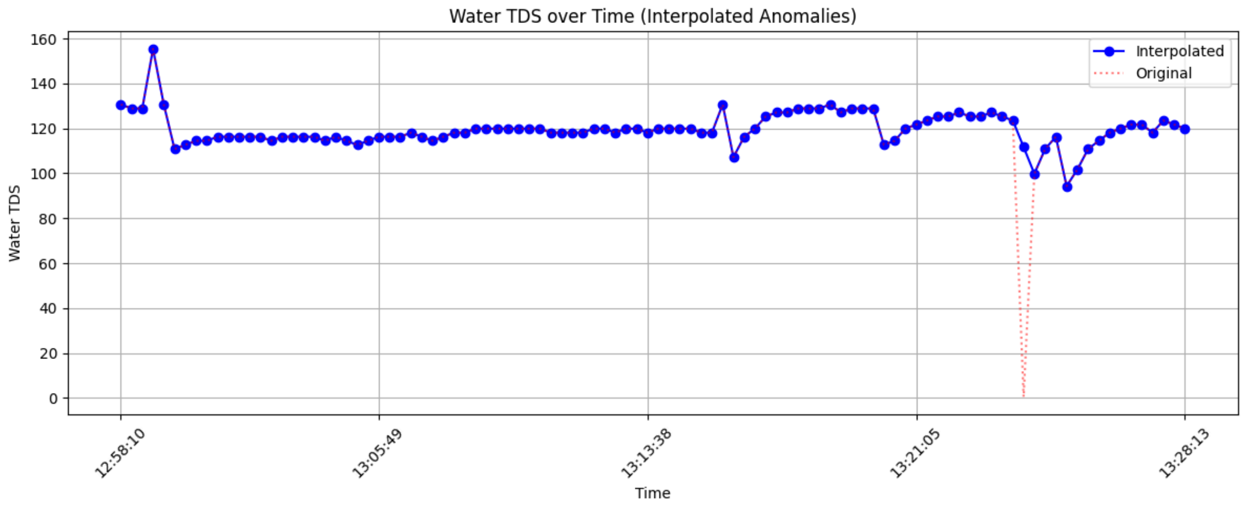


Figure ? Water TDS Visualization Without Anomalies

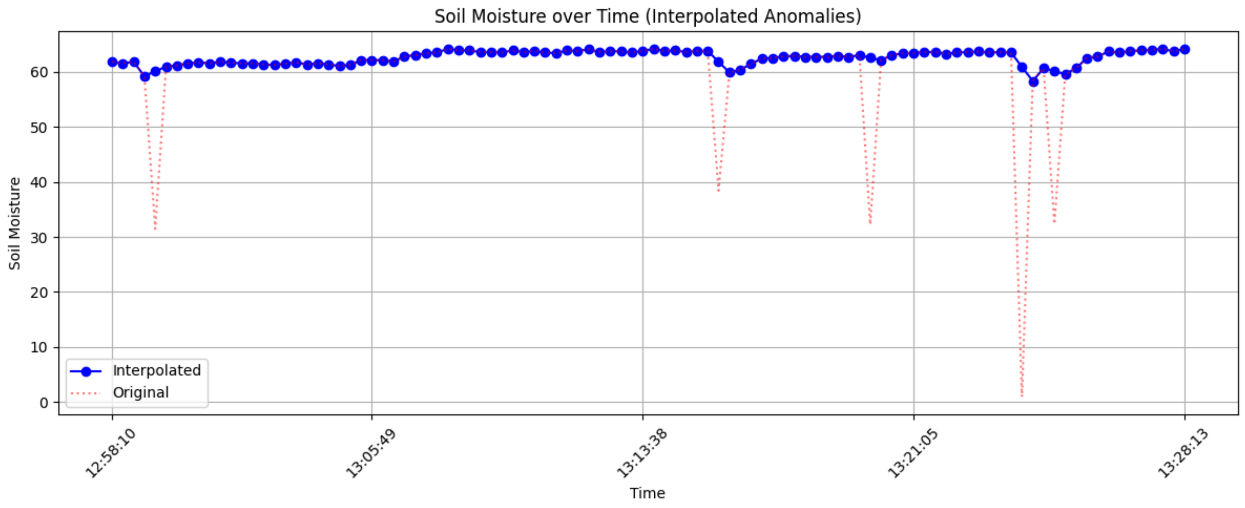
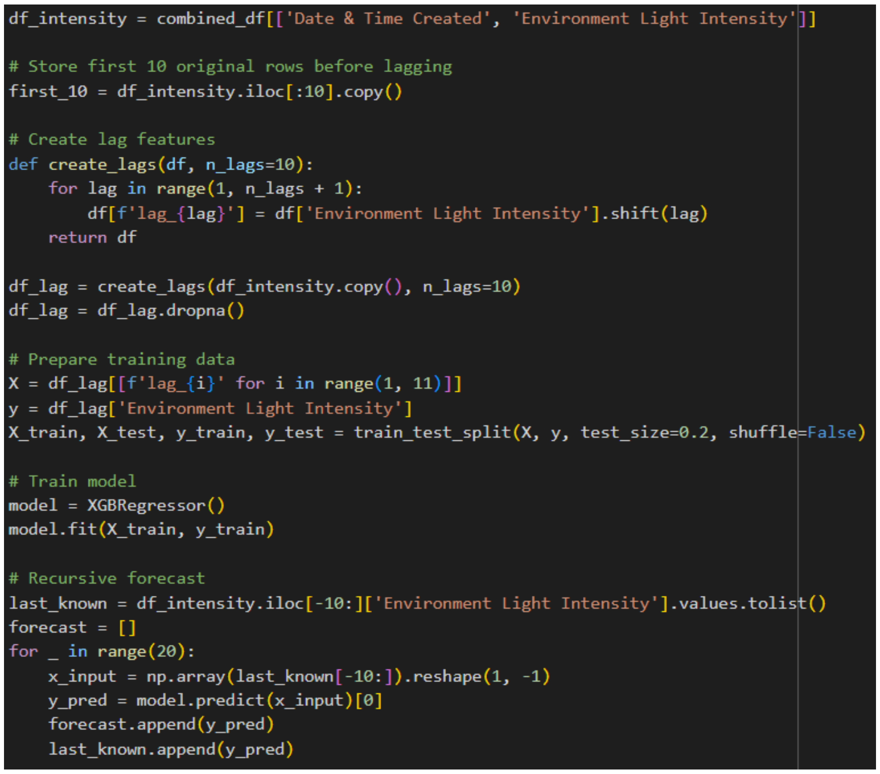


Figure ? Soil Moisture Visualization Without Anomalies

According to Figures ??? and ???, the original anomalies are removed and imputed with the interpolated values, thus removing the sensor data of any anomalies and inconsistencies, providing a more accurate representation of sensor data.



```python
df_intensity = combined_df[['Date & Time Created', 'Environment Light Intensity']]

# Store first 10 original rows before lagging
first_10 = df_intensity.iloc[:10].copy()

# Create lag features
def create_lags(df, n_lags=10):
    for lag in range(1, n_lags + 1):
        df[f'lag_{lag}'] = df['Environment Light Intensity'].shift(lag)
    return df

df_lag = create_lags(df_intensity.copy(), n_lags=10)
df_lag = df_lag.dropna()

# Prepare training data
X = df_lag[[f'lag_{i}' for i in range(1, 11)]]
y = df_lag['Environment Light Intensity']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)

# Train model
model = XGBRegressor()
model.fit(X_train, y_train)

# Recursive forecast
last_known = df_intensity.iloc[-10:]['Environment Light Intensity'].values.tolist()
forecast = []
for _ in range(20):
    x_input = np.array(last_known[-10:]).reshape(1, -1)
    y_pred = model.predict(x_input)[0]
    forecast.append(y_pred)
    last_known.append(y_pred)
```

Figure ? Code Snippet of Lag Feature Creation and Forecasting

Environment light intensity will be used to demonstrate the forecasting of sensor data. An XGBoost model is trained and used to forecast the next 20 intervals of environment light intensity. The model is trained using lag features, which are created from the previous 10 data points of each data point to allow the model to learn from past trends in sensor data (Reddy, 2025). For forecasting, the newly forecasted values are recursively used as the input value, until the forecasted intervals reach 20.

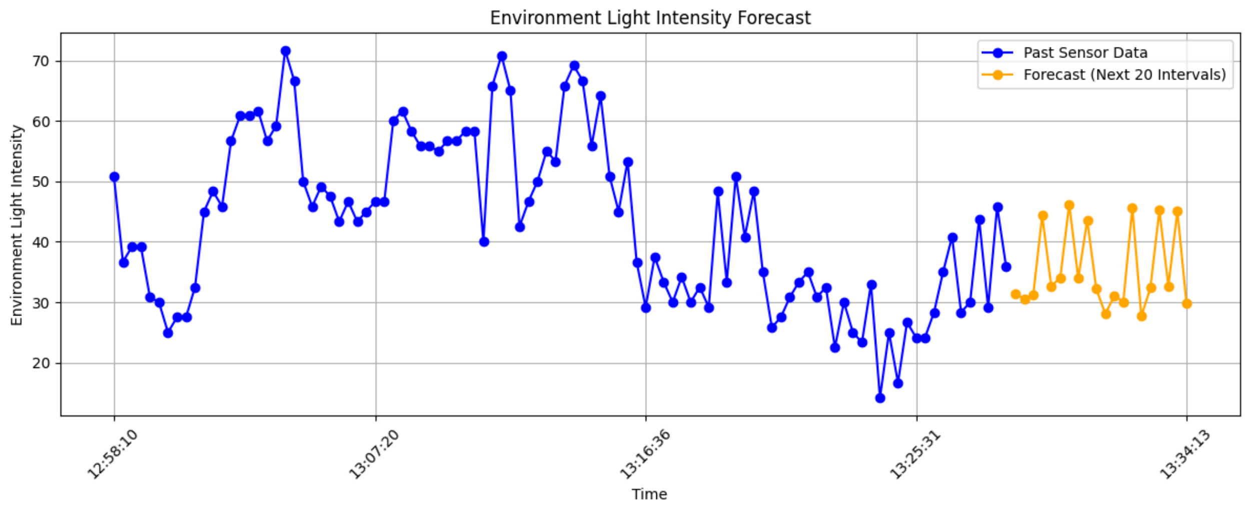
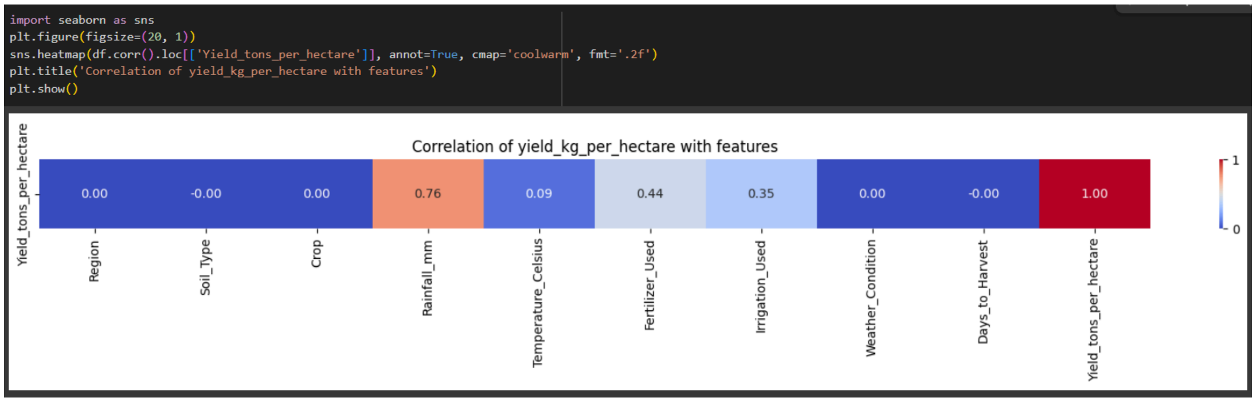


Figure ? Visualization of Environment Light Intensity Forecasting

According to Figure ???, the blue line graph represents the past sensor data that the model uses to predict the forecasted sensor data, which is represented by the yellow line graph. This feature allows the system or the user to take preemptive measures according to the forecasted values. For example, if the forecasted environment temperature is expected to increase for the next few hours, the irrigation system can be activated first to avoid watering the plants too late.

3.2.2 Crop Yield Prediction

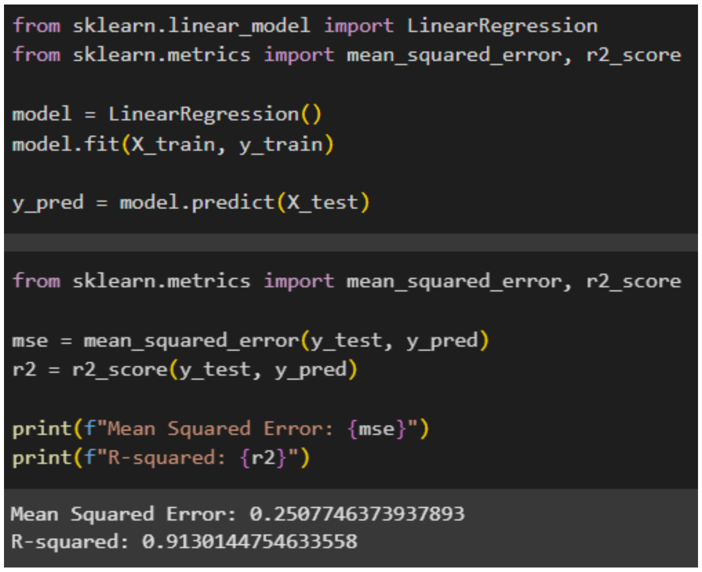
The dataset used for this section is obtained from Kaggle, which is published by Samuel Oti Attakorah (Samuel Oti Attakorah, 2024). The dataset contains 10 features, which are Region, Soil_Type, Crop, Rainfall_mm, Temperature_Celcius, Fertilizer_Used, Irrigation_Used, Weather_Condition, and Days_to_Harvest, with Yield_tons_per_hectare as the target feature.



```
import seaborn as sns
plt.figure(figsize=(20, 1))
sns.heatmap(df.corr().loc[['Yield_tons_per_hectare']], annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation of yield_kg_per_hectare with features')
plt.show()
```

Figure ? Correlation Matrix for Feature Selection

Basic preprocessing steps such as identifying and handling null and duplicate values, categorical encoding, and feature scaling are performed, which is followed by plotting the correlation matrix in Figure ???. According to the correlation matrix, only features which have correlation with the target feature are chosen, which are Rainfall_mm, Temperature_Celcius, Fertilizer_Used, and Irrigation_Used.



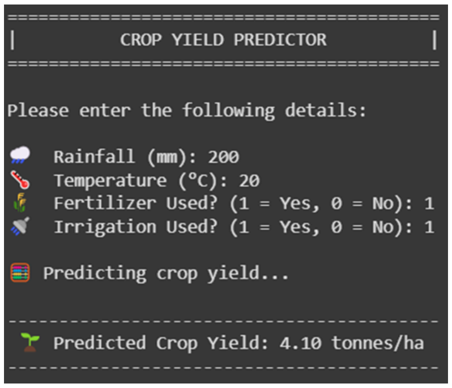
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
```

```
from sklearn.metrics import mean_squared_error, r2_score

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")
```

```
Mean Squared Error: 0.2507746373937893
R-squared: 0.9130144754633558
```

Figure ? Code Snippet of Model Training and Evaluation

A linear regression model is trained with the input features and its performance is evaluated using the testing set. Its R-squared value is 0.913, which indicates that it is able to explain 91.3% of the variance in the target feature using the input features. Its low MSE value of 0.25 also suggests that it has low prediction errors overall, making it a reliable model in predicting crop yield.



```
========================================
|          CROP YIELD PREDICTOR         |
========================================

Please enter the following details:

🌧️  Rainfall (mm): 200
🌡️  Temperature (°C): 20
🌾  Fertilizer Used? (1 = Yes, 0 = No): 1
🚿  Irrigation Used? (1 = Yes, 0 = No): 1

🧮 Predicting crop yield...

----------------------------------------
 🌱 Predicted Crop Yield: 4.10 tonnes/ha
----------------------------------------
```

Figure ? Crop Yield Prediction Program

A crop yield predictor program is created using the trained model, which asks the user for specific information such as amount of rainfall in millimeters(mm), temperature in degrees Celcius (°C), and whether fertilizer and irrigation are used. The program will predict the crop yield based on the input data, which in this case is 4.10 tonnes/ha as shown in Figure ???. This feature allows the user to estimate the amount of crop yield based on their inputs, allowing them to satisfy their curiosity and plan in advance to reap higher crop yields.

3.3 Figma

The Figma prototype for GrowSync's mobile application can be accessed at the following link: https://www.figma.com/proto/73r9f7dUAbqF3gHYwSBsat/IoT-Assignment-3?node-id=0-1&t=XCJMvt8gefwDakDR-1

This link displays GrowSync mobile application prototype, which focuses on the app's security, real-time alerts, and data visualization. The login interface uses Role-Based Access Control (RBAC), which ensures that users are assigned the appropriate permissions based on their roles as farm owner or farm worker. After entering valid credentials (email and password), users must complete Multi-Factor Authentication (MFA) with a one-time password (OTP), which adds an additional layer of security before accessing the system.

The alert system includes real-time alerts that appear directly on the user interface, encouraging users to take immediate action. Users can choose whether to control specific actuators based on the alert. To ensure secure operations, any actuator activation requires secondary verification via OTP, reinforcing the system's multi-layered security approach.

Lastly, users can navigate the app to access various functionalities such as interactive dashboards, zone maps, data analytics, user profiles, and archived notifications, all accessible through interface buttons.

The Figma File for GrowSync's mobile application can be accessed at the following link: https://www.figma.com/design/73r9f7dUAbqF3gHYwSBsat/IoT-Assignment-3?node-id=0-1

References

AWS. (2016). *AWS IoT Core - Amazon Timestream*. Amazon.com. https://docs.aws.amazon.com/timestream/latest/developerguide/IOT-Core.html

Dizdarević, J., Carpio, F., Jukan, A., & Masip-Bruin, X. (2019). A Survey of Communication Protocols for Internet of Things and Related Challenges of Fog and Cloud Computing Integration. *ACM Computing Surveys*, *51*(6), 1–29. https://doi.org/10.1145/3292674

Dragino. (2025, April 17). *Dragino LSPH01-US915 LoRaWAN pH and Temperature Sensor*. Embedded Works. https://embeddedworks.net/product/sens614/

Fidelis, I., & Idim, I. (2020). Design and Implementation of Solar Powered Automatic Irrigation System. *American Journal of Electrical and Computer Engineering*, *4*(1), 1. https://doi.org/10.11648/j.ajece.20200401.11

Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, *29*(7), 1645–1660. https://doi.org/10.1016/j.future.2013.01.010

He, N. (2024, March 19). *LoRaWAN VS NB-IoT: How Do They Compare and Differ*. MOKOSmart #1 Smart Device Solution in China. https://www.mokosmart.com/lorawan-vs-nb-iot-how-do-they-compare-and-differ/

Hsu, T.-C. (2024). Designing a Secure and Scalable Service Agent for IoT Transmission through Blockchain and MQTT Fusion. *Applied Sciences*, *14*(7), 2975–2975. https://doi.org/10.3390/app14072975

Jawad, H., Nordin, R., Gharghan, S., Jawad, A., & Ismail, M. (2017). Energy-Efficient Wireless Sensor Networks for Precision Agriculture: A Review. *Sensors*, *17*(8), 1781. https://doi.org/10.3390/s17081781

Quartz. (2022). *MQ135 Air Quality Gas Sensor Module*. QuartzComponents. https://quartzcomponents.com/products/mq-135-air-quality-gas-sensor-module?srsltid=AfmBOopqrysv8qTdAeran1PRny5QqE2DyXwW4oC4W6efKb5u4_MsKxXL

Rejeb, A., Keogh, J. G., & Treiblmaier, H. (2019). Leveraging the Internet of Things and Blockchain Technology in Supply Chain Management. *Future Internet*, *11*(7), 161. https://doi.org/10.3390/fi11070161

Singh, R. K., Aernouts, M., De Meyer, M., Weyn, M., & Berkvens, R. (2020). Leveraging LoRaWAN Technology for Precision Agriculture in Greenhouses. Sensors, 20(7), 1827. https://doi.org/10.3390/s20071827

DY, S. (2024). Smart Agriculture and Plant Health Monitoring using IoT. Mendeley Data, 1. https://doi.org/10.17632/65jxyrxv7b.1

Reddy, Y. (2025). Forecasting Using Xgboost (lag and decomposition) - yashwanth reddy - Medium. Medium. https://medium.com/@byashwanth77/forecasting-using-xgboost-lag-and-decomposition-864815bd98c5

Samuel Oti Attakorah. (2024). Agriculture Crop Yield. Kaggle.com. https://www.kaggle.com/datasets/samuelotiattakorah/agriculture-crop-yield