Smart Farming in India: IoT-Enhanced Machine Learning for Predicting Rice Yields

Project report submitted to the Amrita Vishwa Vidyapeetham in partial fulfilment of the requirement for the Degree of

BACHELOR OF COMPUTER APPLICATIONS

Submitted by

Nandana Sumesh, Navaneeth R, Vimal Raj and Vismaya Rajesh,

(AM.SC.U3CSC21038, AM.SC.U3CSC21039, AM.SC.U3CSC21065 and AM.SC.U3CSC21067)

DEPARTMENT OF COMPUTER SCIENCE AND APPLICATIONS

AMRITA SCHOOL OF COMPUTING

AMRITA VISHWA VIDYAPEETHAM AMRITAPURI CAMPUS

JUNE 2024

DEPARTMENT OF COMPUTER SCIENCE AND APPLICATION
AMRITA VISHWA VIDYAPEETHAM AMRITAPURI CAMPUS

BONAFIDE CERTIFICATE

This is to certify that **Nandana Sumesh, Navaneeth R, Vimal Raj and Vismaya Rajesh, (AM.SC.U3CSC21038, AM.SC.U3CSC21039, AM.SC.U3CSC21065 and AM.SC.U3CSC21067)**, a bonafide Bachelor of Computer Applications student at the Amrita School of Computing, Amrita Vishwa Vidyapeetham, has completed their undergraduate project titled *Smart Farming in India: IoT-Enhanced Machine Learning for Predicting Rice Yields* under my guidance and supervision.

The project was completed successfully within the time frame allotted and the student has demonstrated a good understanding of the subject matter. The work carried out by the student is original and has not been submitted for any other academic purpose.

I certify that the student has fulfilled all the requirements necessary for the completion of the project and is eligible for the Bachelor of Computer Applications award. I wish all the success in their future endeavours.

Ms. Ani R
Project Guide

Dr. MG Thushara
Project Coordinator

Ms. Ani R
Chairperson
Dept. of CSA

Reviewer(s)

Place: Amritapuri Campus
Date :

DEPARTMENT OF COMPUTER SCIENCE AND APPLICATIONS

AMRITA VISHWA VIDYAPEETHAM
AMRITAPURI CAMPUS

DECLARATION

We, **Nandana Sumesh, Navaneeth R, Vimal Raj and Vismaya Rajesh, (AM.SC.U3CSC21038, AM.SC.U3CSC21039, AM.SC.U3CSC21065 and AM.SC.U3CSC21067)** hereby declare that this project entitled **Smart Farming in India: IoT-Enhanced Machine Learning for Predicting Rice Yields** is a record of the original work done by us under the guidance of **Ms. Ani R**, Amrita School of Computing, Amrita Vishwa Vidyapeetham that this work has not formed the basis for any degree/diploma/associationship/fellowship or similar awards to any candidate in any university to the best of our knowledge.

Signature of Student(s)

Ms. Ani R
Project Guide

Place : Amritapuri Campus
Date :

Acknowledgments

I would like to express my sincere gratitude to all those who have helped me in completing this project successfully. This project has given me the opportunity to enhance my skills and knowledge in the field of computer applications.

First and foremost, we would like to acknowledge Sri. Mata Amritanandamayi Devi our Chancellor and guiding light for giving us the knowledge and opportunity to undertake this project satisfactorily.

I want to thank my project guide, Ani Ma'am, for her constant support and guidance throughout the project. Her valuable suggestions and feedback have been instrumental in shaping this project.

I am also grateful to Thushara Ma'am, Project Coordinator, for providing me with the necessary resources and infrastructure to carry out this project. I want to thank the management of Amrita Vishwa Vidyapeetham, Amritapuri for allowing me to work on this project.

Last but not least, I would like to acknowledge the contributions of my fellow team members who have worked alongside me on this project. Their dedication and teamwork have been the key to the success of this project.

Nandana Sumesh
Navaneeth R
Vimal Raj
Vismaya Rajesh

Abstract

In recent years, the agricultural sector has been facing challenges in achieving optimal crop yields due to complex environmental factors and limited access to data-driven decision-making tools. This work presents an innovative IoT-based approach to predict crop yields, with the goal of improving crop productivity for farmers. Our system utilizes a network of IoT sensors, such as DHT22 and rainfall sensors, to gather real-time data on temperature, humidity, and rainfall. We have also explored the use of soil-integrated sensors to monitor specific soil properties such as Nitrogen, phosphorus, Potassium, pH, Electrical Connectivity, humidity, and temperature. This data is sent to a web interface for live display and stored for predictive modeling. We implemented several regression models—decision tree, random forest, ridge regression, and linear regression—to predict crop yields, with the random forest regression model achieving the highest accuracy and an error rate of 6.46 percent. We have also analyzed the stacking model but the random forest model's MAE and R2 metrics proved superior, leading to its selection for deployment. A user-friendly web-based interface is developed to allow farmers to interact with the system and predict crop yields effectively. This approach seamlessly integrates IoT technology with advanced data analytics, providing actionable insights that can optimize agricultural practices, thereby enhancing food security and improving livelihoods.

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1 Project Background

Agriculture plays a vital role in feeding the world's expanding population and is the primary livelihood for a large portion of the population in India. Improving food production to meet the growing demand is essential. Farmers need to adopt modern practices to maximize output and minimize losses. Machine learning has emerged as a powerful tool in forecasting and analysing crop growth, significantly enhancing precision in the agricultural sector. Smart farming integrates various technologies, such as IoT sensors, data analytics, and precision agriculture techniques, to boost productivity and optimize resource usage. The primary aim of smart agricultural systems is to improve field productivity, minimize resource consumption, and enhance overall crop yield. Our project was initiated to explore the potential of machine learning and IoT in predicting crop yields accurately, helping farmers make informed decisions and improve their agricultural practices.

1.2 Project Scope

1.2.1 Inclusions:

1. **Development of an IoT-based Crop Yield Prediction System:**

    - Utilizing IoT sensors like DHT22, pH, and rain sensors to monitor environmental conditions such as temperature, rainfall, and soil pH.
    - Collecting data from these sensors for analysis and prediction.
    - Implementing machine learning algorithms such as regression models like Linear Regression, Decision Tree Regressor, Random Forest Regressor, and Ridge Regression, to predict crop yields.

2. **Data Processing and Analysis:**

    - Cleaning and preprocessing the collected data to ensure accuracy and reliability.
    - Analysing the data to understand the relationships between environmental factors and crop yield.
    - Evaluating the performance of different regression models to select the most accurate one for predictions.

3. **Development of a User-Friendly Interface:**

    - Creating a website where farmers can see real-time data collected from the IoT sensors.
    - Allowing farmers to input additional parameters such as area in hectares and pH values. Upon clicking the model button, farmers can receive predictions for the upcoming years based on these inputs and other parameters that automatically get populated.

1.2.2 Exclusions:

1. **Deployment and Maintenance of IoT Infrastructure:**

    - Our Project does not cover the physical deployment and maintenance of IoT sensors in the field.
    - It focuses on the data analysis and prediction aspects, assuming the availability of sensor data.

2. **Economic and Feasibility Analysis:**

    - Our project does not include a detailed economic analysis of implementing IoT and machine learning systems in farming.
    - It aims to demonstrate the technical feasibility and potential benefits of such systems.

3. **Comprehensive Solution for All Crop Types:**

    - It specifically focuses on predicting the yield of rice using Random Forest Regressor.
    - It does not provide a one-size-fits-all solution for all types of crops and farming conditions.

2. Methodology

2.1 Development Approach

The development of this interactive crop yield prediction platform followed the Agile methodology. This approach allowed for iterative progress through collaboration, flexibility, and continuous feedback, ensuring that the evolving needs of users and the project were met efficiently.

2.2 Project Phases and Implementation

2.2.1 Planning and Requirement Analysis:

- **Objective**: Define the core functionalities and user requirements.
- **Activities**:
    - Established project goals and deliverables.
    - Created a detailed project plan outlining the scope, timeline, and milestones.

2.2.2 Implementation:

- **Objective**: Build the core components of the system.
- **Activities**:
    - **User Authentication**: Implemented user login functionality with Flask and SQLite3.
    - **Homepage**: Created a welcoming homepage with a "Predict" button and navigation menu.
    - **Live Data Display:** Developed the "Live Data" page to show real-time sensor readings from the ESP32.
    - **Prediction Model Integration**: Integrated a pre-trained machine learning model for crop yield prediction.
    - **ESP32 Communication**: Established communication between the Flask server and the ESP32 using HTTP and JSON.
    - **Real-Time Updates**: Implemented SocketIO for pushing real-time data updates to clients.
    - **Contact and Profile Management**: Added functionalities for user contact and profile management.

2.2.3 Testing:

- **Objective**: Ensure the system functions correctly and meets user requirements.
- **Activities**:
    - Conducted unit testing for individual components.
    - Performed integration testing to verify the interactions between components.

2.2.4 Deployment:

- **Objective**: Make the system available to users.
- **Activities**:
    - Deployed the Flask application on a web server.
    - Configured the database and ensured secure access.
    - Set up monitoring and logging to track system performance.

2.2.5 Maintenance and Iteration:

- **Objective**: Continuously improve the system based on user feedback.
- **Activities**:
  - Monitored system performance and user interactions.
  - Collected feedback and identified areas for improvement.
  - Planned and implemented iterative enhancements.

2.3 Tools and Technologies

2.3.1 Hardware

To deploy an IoT-based system for monitoring efficient crop yield, a simple circuit model was adopted, including the following components:

2.3.1.1 IoT Sensors:

- **DHT22 Sensor**:
  - Used to monitor temperature and humidity.
  - Provides accurate and reliable data necessary for understanding environmental conditions affecting crop growth.
- **Rain Sensor**:
  - Measures rainfall.
  - Crucial for assessing soil moisture levels and predicting crop yield.
- **pH Sensor**:
  - Intended for monitoring soil pH levels.
  - Could not be obtained locally due to limited distribution channels.
  - Alternative methods using a soil pH probe and meter were considered impractical for continuous, real-time monitoring.

2.3.1.2 Hardware Components:

- **Breadboard**:
  - Provides a platform for building and testing the circuit without soldering.
- **Jumper Wires**:
  - Connects the sensors and other components on the breadboard to the ESP-WROOM-32.
- **ESP-WROOM-32**:
  - Acts as the main microcontroller for the system.
  - Facilitates the processing of sensor data and controls connected components.
  - Collects real-time data from sensors and stores it in the database.
  - Provides Wi-Fi connectivity for real-time data transmission.
  - Programmed using Arduino IDE software.

2.3.2 Software

2.3.2.1 Website Development:

- **Framework:**
  - Flask: Used for developing the backend of the website, providing a lightweight and flexible framework for integrating the prediction model.

- **Frontend:**
    - HTML, CSS, JavaScript: Used to ensure a user-friendly interface.
- **Features:**
    - User authentication via SQLite3 database.
    - Real-time data display from IoT sensors.
    - Crop yield prediction model allowing users to enter yearly average temperature, rainfall, pH, and area.

2.3.2.2 Arduino IDE:

- Used for programming the IoT sensors to accurately collect and transmit data.

2.3.2.3 Database:

- **SQLite3:**
    - Stores user data such as usernames and passwords.
    - Manages IoT sensor data (temperature, rainfall, pH) in hourly, daily, monthly, and yearly tables.
    - Ensures secure and organized data management for accurate predictions.

2.3.3 Model Development

2.3.3.1 Preprocessing:

- **Data Cleaning and Preparation:**
    - Dataset collected from Kaggle was cleaned to ensure accuracy.
    - Removed inconsistencies and outliers such as null and blank values.
- **Selection of Input Variables:**
    - Focused on relevant input variables: temperature, rainfall, pH, and area.
    - Excluded other numerical input variables due to poor correlation.
- **Data Splitting:**
    - Dataset divided into training (80%) and testing (20%) sets.

2.3.3.2 Regression Models:

- **Models Evaluated:**
    - Linear Regression
    - Ridge Regression
    - Decision Tree Regressor
    - Random Forest Regressor
- **Performance:**
    - Random Forest Regressor provided the highest accuracy in predicting crop yield.
    - Chosen for its ability to handle complex data patterns and provide robust predictions.

2.3.3.3 Model Stacking:

- **Technique:**
    - Employed a stacking regression approach to enhance predictive accuracy.
    - Combined multiple regression models to improve overall performance.
- **Base Models:**
    - Linear Regression

    - Ridge Regression
    - Random Forest Regression
    - Decision Tree Regressor
- **Meta-Model:**
    - Combined predictions from base models to produce final, improved predictions.
    - Included Decision Tree Regressor, Linear Regressor, Ridge Regressor, and Random Forest Regressor as final estimators.

3. Software Requirements Specification (SRS)

3.1 Purpose and Scope

The purpose of this Software Requirements Specification (SRS) is to define the functional and non-functional requirements for the IoT-based Crop Yield Prediction System. This system aims to provide an interactive platform for users to predict crop yield based on various environmental inputs, including temperature, rainfall, pH, and area. The SRS covers the key features and requirements of the system, including user authentication, data collection and processing, crop yield prediction, real-time data monitoring, and user interaction.

3.2 Functional Requirements

3.2.1 User Authentication

- Users can register and create accounts in the system.
- Users can log in to the system using their credentials.
- The system should validate user credentials and provide secure access to the platform.
- The system should maintain user session information and allow users to log out securely.

3.2.2 Crop Yield Prediction

- Users can input yearly average temperature, rainfall, pH, and area to predict the total crop yield.
- The system should have a pre-trained machine learning model integrated into the Flask backend to generate crop yield predictions based on the user inputs.
- The prediction model should provide accurate and reliable crop yield estimates.
- The system should display the predicted crop yield to the user.

3.2.3 Real-Time Data Monitoring

- The system should collect real-time temperature, humidity, and rainfall data from an ESP32 IoT device.
- The system should process the sensor data and store it in a SQLite3 database, calculating daily, monthly, and yearly averages.
- The system should use SocketIO to push real-time updates to connected clients, enabling dynamic data display without page refreshes.
- Users should be able to view the real-time sensor data on the "Live Data" page.

3.2.4 Data Management

- The system should store user credentials and sensor data in a SQLite3 database.
- The system should handle the insertion, updating, and retrieval of data from the database.
- The system should maintain the integrity and security of the stored data.

3.2.5 User Interface and Navigation

- The system should provide a user-friendly and intuitive web interface built using the Flask framework.
- The interface should include pages for user login, homepage, live data monitoring, crop yield prediction, contact, and user profile.
- The navigation menu should allow users to easily access the different functionalities of the system.

3.2.6 Communication and Protocols

- The system should use HTTP/HTTPS for communication between the client and server.
- The system should use WebSocket for real-time data updates between the client and server.
- The ESP32 IoT device should communicate with the Flask server using Wi-Fi and send sensor data in JSON format.

3.3 Non-Functional Requirements

3.3.1 Performance

- The system should be able to handle a reasonable number of concurrent user sessions without significant performance degradation.
- The system should provide fast response times for user interactions, such as form submissions and data retrievals.
- The system should be able to process and update real-time sensor data in a timely manner.

3.3.2 Security

- The system should implement secure password hashing and storage mechanisms for user credentials.
- The system should protect against common web application vulnerabilities, such as SQL injection and cross-site scripting (XSS).
- The system should enforce user session management and provide secure logout functionality.

3.3.3 Reliability

- The system should have high availability and minimal downtime.
- The system should be able to recover from expected and unexpected errors without data loss or corruption.
- The system should have a robust error-handling mechanism to provide meaningful feedback to users in case of issues.

3.3.4 Scalability

- The system should be designed to accommodate a growing number of users and sensor data without significant performance degradation.
- The database and data processing components should be able to scale as the system's usage and data volume increase.

3.3.5 Maintainability

- The system should have a modular and extensible design, allowing for easy maintenance and future enhancements.
- The codebase should be well-documented, adhering to coding best practices and standards.
- The system should provide clear error logging and debugging mechanisms to facilitate troubleshooting and issue resolution.

4. Software Design Document (SDD)

4.1 System Overview

Our website is designed to help farmers and agricultural enthusiasts predict crop yields using modern technology. Users begin by logging in with their credentials. Once logged in, they arrive at a welcoming homepage that includes a brief description of the site and a central "Predict" button to start the crop yield prediction process.

The site also has a "Live Data" section where users can view real-time information on temperature, humidity, and rainfall collected from sensors. To predict crop yields, users can enter yearly average values for temperature, rainfall, pH level, and the area of land they want to cultivate. The website uses an advanced machine learning model to predict potential crop yields based on these inputs.

Additionally, users can contact the admin for support, visit an "About Us" section for more information, and log out securely. Overall, our website uses technology to provide valuable insights into crop yields, making farming more efficient and informed.

4.1.1 Target Users

- **Farmers and Agricultural Experts**: Our primary users are farmers and agricultural experts who seek to predict crop yields based on various environmental data. By using our website, they can enter critical parameters such as yearly average temperature, rainfall, pH level, and land area to obtain accurate yield predictions. This data-driven approach allows them to optimize their farming practices, making informed decisions about planting, irrigation, and resource allocation. The real-time display of environmental data such as temperature, humidity, and rainfall further aids in immediate decision-making, ensuring they can respond promptly to changing conditions. By leveraging the sophisticated machine learning model integrated into our website, farmers and agricultural experts can enhance their productivity, reduce risks, and ultimately increase their yields and profitability.
- **System Administrators**: System administrators play a crucial role in ensuring the smooth operation and maintenance of our website. Their responsibilities include managing user authentication to ensure secure access, handling the reception and processing of sensor data from the ESP32, and maintaining the SQLite3 database where this data is stored. Administrators ensure that real-time updates are accurately pushed to users via WebSocket communication, maintaining the integrity and reliability of live data displays. They also manage the email contact form, facilitating effective communication between users and the admin for feedback, questions, or complaints. Additionally, system administrators oversee the profile management section, ensuring that users can securely view and update their details, access the "About Us" page, and log out securely. Their role is vital in maintaining the system's overall functionality, security, and user satisfaction, ensuring that the website operates efficiently and meets the needs of its users.

4.1.2 Key Features

- **User Authentication**: Our website ensures secure access through a robust login system. Users authenticate themselves using their credentials, which initiates a session that keeps their information secure and private throughout their visit. This secure login mechanism protects user data and maintains the integrity of the system.
- **Home Page**: Upon successful login, users are greeted by a friendly and informative homepage. This page provides a brief introduction to the website. A prominent "Predict" button on the homepage serves as a gateway to the crop yield prediction feature. The homepage also includes easy navigation to other key sections of the site.
- **Prediction Model**: The core feature of our website is the crop yield prediction model. Users can access this feature through the "Predict" button on the homepage or via the navigation menu. The prediction page includes an input form where users can enter environmental parameters such as yearly average temperature, rainfall, pH level, and the area of land they wish to cultivate. The website leverages a sophisticated machine learning model to analyze these inputs and forecast the potential crop yield, providing users with valuable insights for their farming decisions.
- **Live Data Display**: In the "Live Data" section, users can view real-time readings of temperature, humidity, and rainfall. These readings are continuously collected from sensors connected to an ESP32 microcontroller. This feature keeps users updated with the latest environmental conditions, which can be crucial for timely decision-making in agriculture.
- **Sensor Data Handling**: Our website effectively handles sensor data received from the ESP32. The sensor data, which includes temperature, humidity, and rainfall readings, is processed and stored in an SQLite3 database. This data is then used to calculate daily, monthly, and yearly averages, providing users with comprehensive environmental insights over different time periods.
- **Email Contact Form**: To ensure effective communication and support, our website includes an email contact form. Users can use this form to send feedback, ask questions, or lodge complaints directly to the admin. This feature helps maintain a responsive and user-centric service.
- **Profile Management**: Users have access to a profile management section where they can view their profile details and make changes if necessary. This section also includes links to the "About Us" page, which provides detailed information about the website, its purpose, and its functionalities. Additionally, users can log out securely from this section, ensuring that their session is terminated properly, and their data remains secure.

4.2 Detailed Design

4.2.1 System Architecture

Our system consists of the following main components:

- **Flask Application**
  - **Routing:** Handles different endpoints for the application, such as login, home, prediction, live data, contact, and profile.
  - **Authentication:** Manages user login and session handling using secure password hashing and session cookies.
  - **Backend Processing:** Integrates with the prediction model to process input data and provide predictions.
  - **HTTP Server:** Receives sensor data from ESP32 and handles user interactions.
- **SQLite3 Database**

    - **User Data Storage:** Stores user credentials securely.
    - **Sensor Data Storage:** Maintains tables for hourly, daily, monthly, and yearly sensor data.
- **ESP32 Microcontroller**
    - **Sensor Integration:** Connects to DHT22 sensor for temperature and humidity readings, and an analog rain sensor for rainfall measurement
    - **Data Transmission:** Sends collected sensor data to the Flask server using HTTP POST requests over Wi-Fi.
    - **Wi-Fi Connectivity:** Establishes and maintains a connection to the local Wi-Fi network to communicate with the Flask server.
- **Socket IO**
    - **Real-Time Communication:** Facilitates real-time updates from the Flask server to connected clients.
    - **Dynamic Data Display:** Ensures the live data page is updated dynamically without requiring a page refresh.
- **Prediction Model**
    - **Machine Learning Model:** Utilizes a Random Forest Regression model created in Jupyter Notebook.
    - **Model Integration:** The trained model is saved and loaded into the Flask backend for making predictions based on user input.

4.2.2 User Interface (UI) Design

- **Login Page**: Provides a secure and straightforward way for users to access their accounts.
- **Homepage**: Welcomes users with a brief description and a central "Predict" button for crop yield predictions.
- **Prediction Model Page**: Allows users to input environmental parameters and receive crop yield predictions. Users can manually input values or use automatically populated yearly averages from sensors, with the data stored in an SQLite3 database and processed by a machine learning model.
- **Live Data Page**: Displays real-time readings of temperature, humidity, and rainfall from an ESP32, offering users up-to-date environmental data.
- **Contact Page**: Includes a simple form for users to send feedback or report issues to the admin.
- **Profile Management**: Lets users view and update their profile details, access the "About Us" page, and log out securely. Logging out clears the session and redirects to the login page.

4.2.3 Data Model

- **User Data**: Stores user profiles, login credentials, and activity logs to ensure personalized experiences and secure access.
- **Environmental Data**: This category encompasses data collected from sensors and other environmental monitoring devices. Captures real-time environmental factors like temperature, humidity, rainfall, pH levels, and any other relevant environmental factors, essential for accurate crop yield predictions and decision-making.
- **Prediction Data**: This component stores data related to crop yield predictions. Stores user-input parameters and the corresponding predicted crop yields generated by the machine learning model, enabling users to track predictions and assess model accuracy.

- **Contact Data**: Manages user communications, storing messages, feedback, and complaints submitted via the contact form for efficient admin response.
- **Session Data**: Tracks user sessions, ensuring secure interactions and proper session management within the system.

4.3 Diagrams

4.3.1 Architecture Diagram

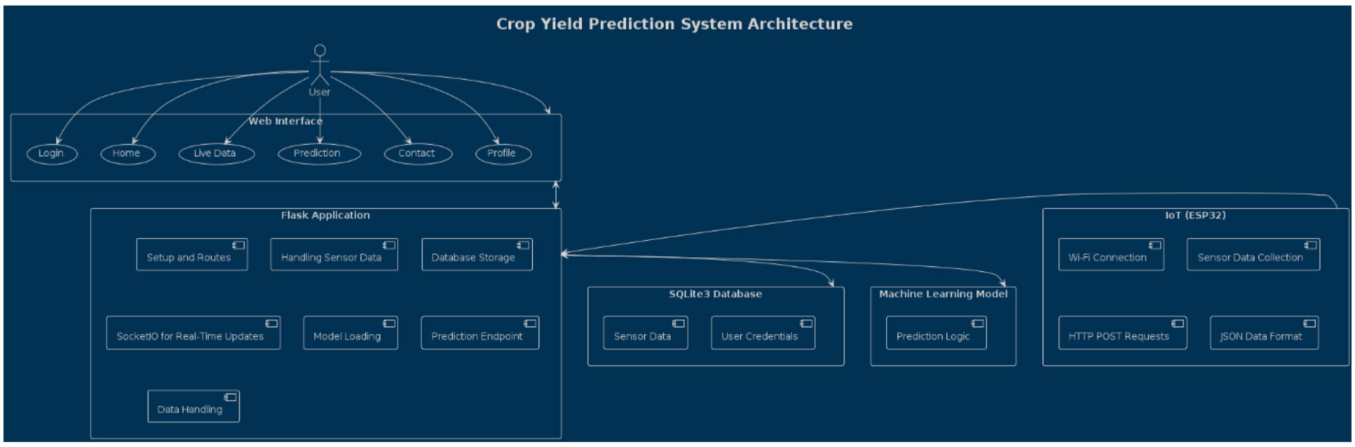


Figure 4.1 architecture diagram

4.3.2 Use Case Diagram

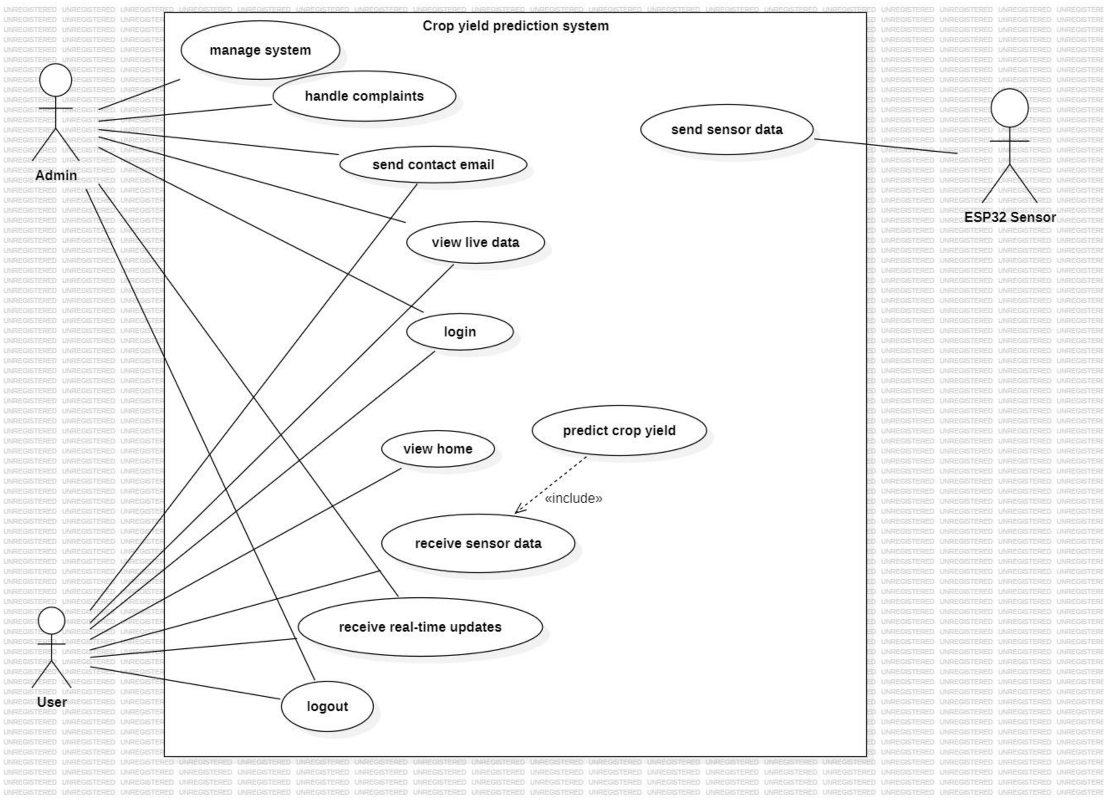


Figure 4.2 use case diagram

4.3.3 Activity Diagram

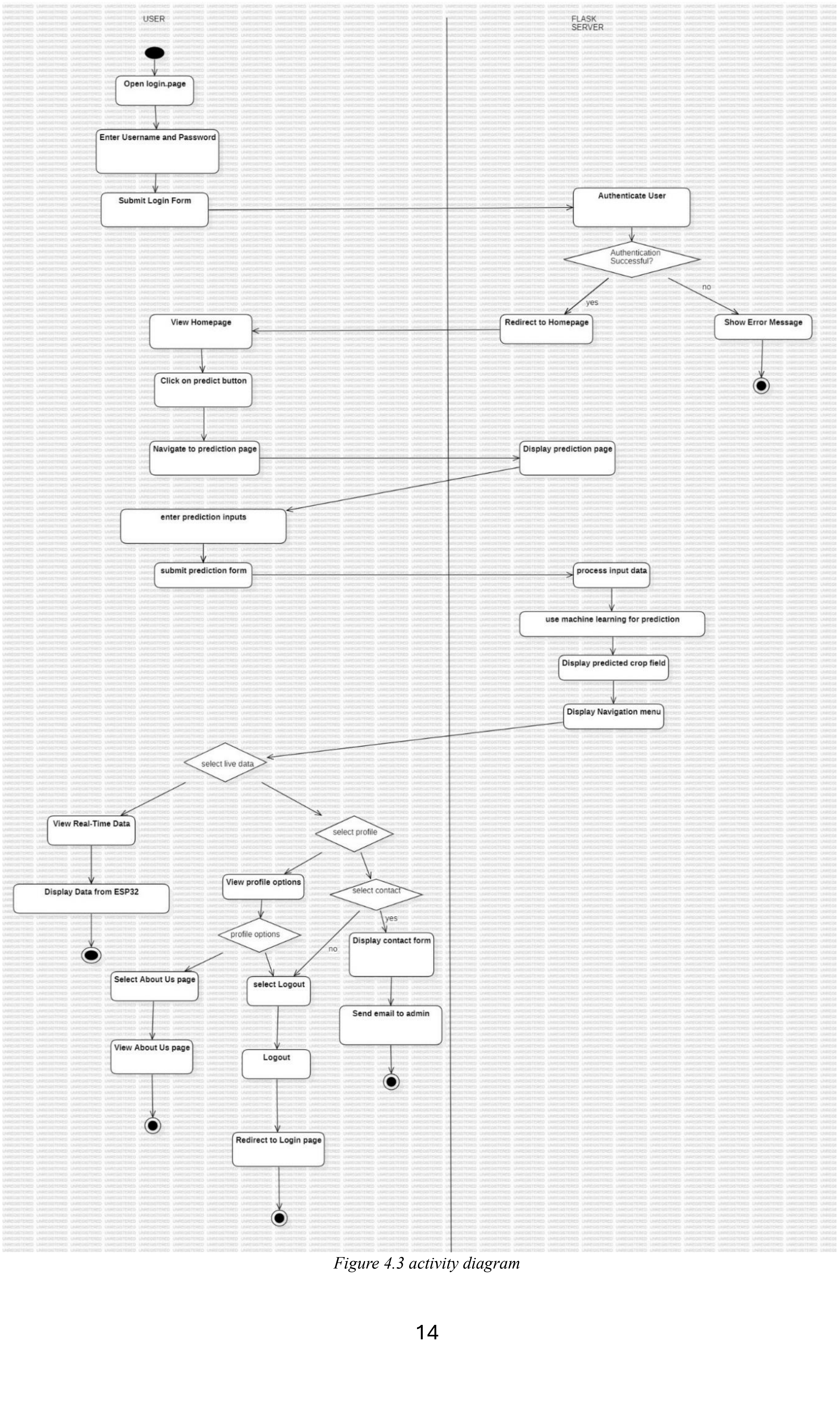


Figure 4.3 activity diagram

4.3.4 Data Flow Diagram

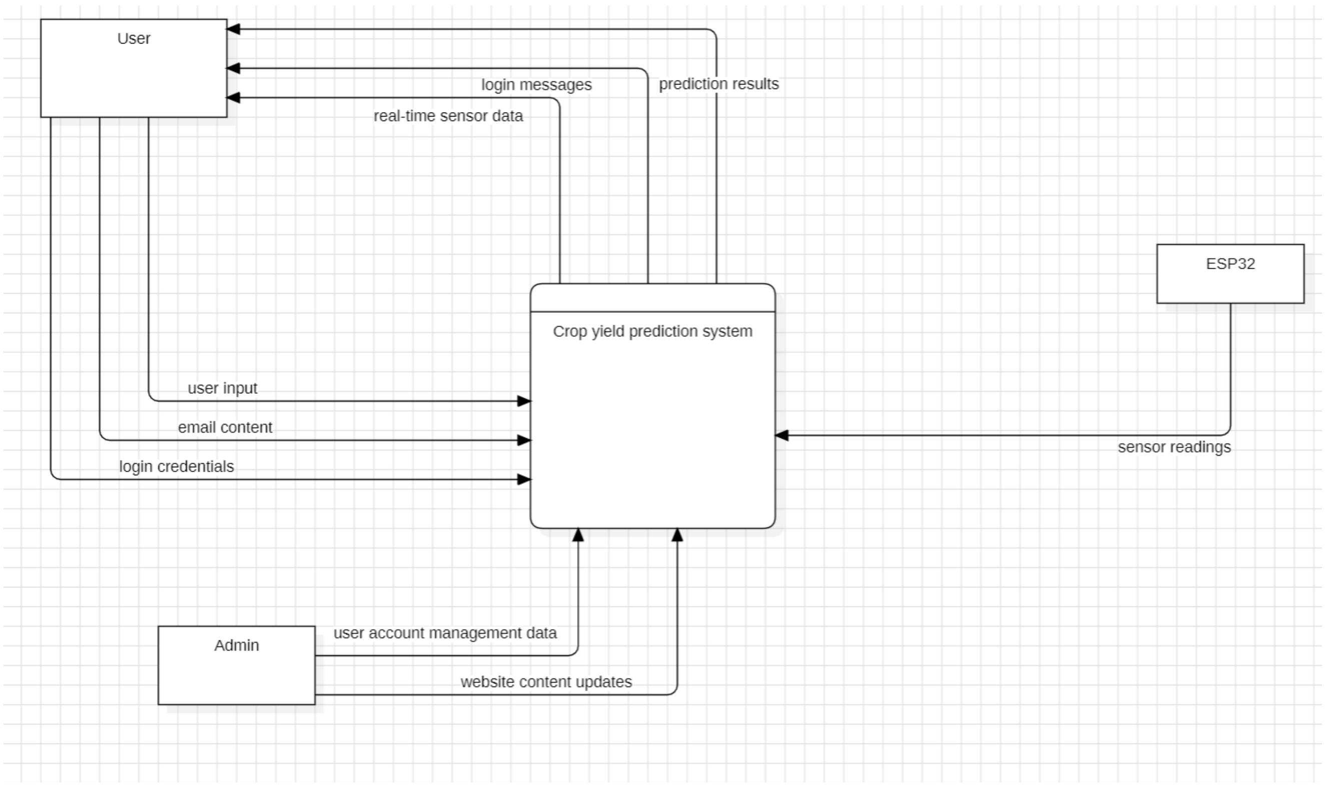


Figure 4.4 DFD context level

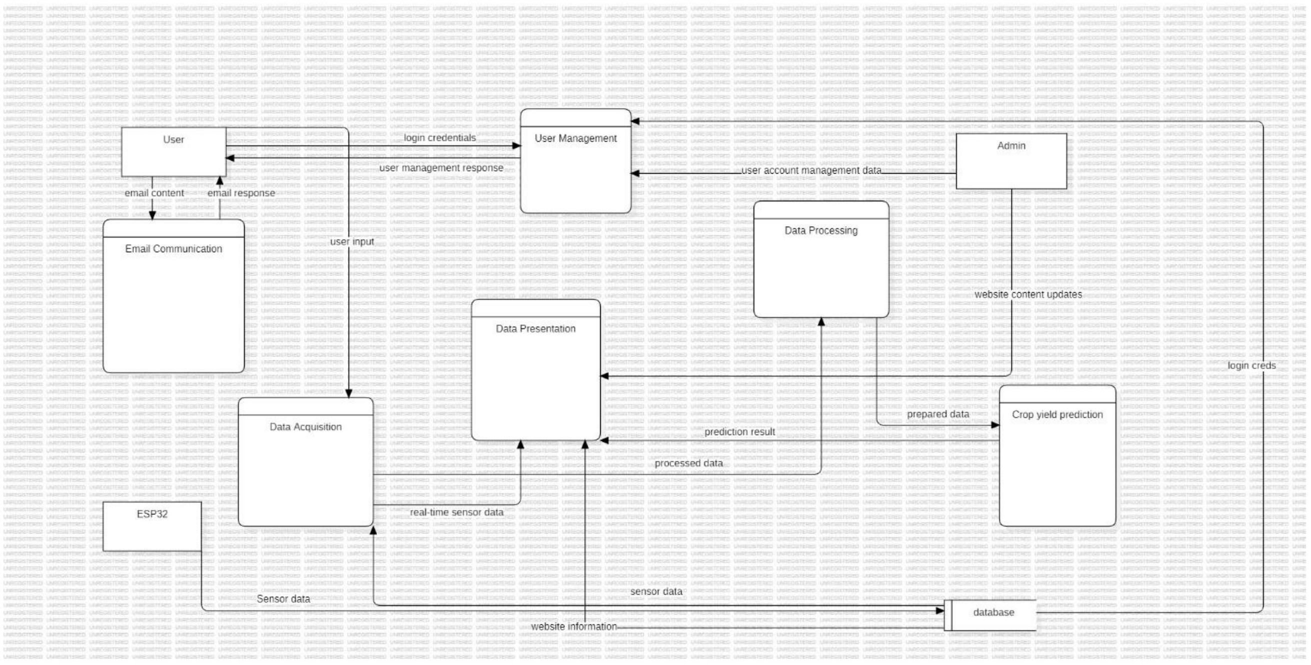


Figure 4.5 DFD level 0

4.4 Detailed Design

4.4.1 User Interface (UI) Design

The UI is designed to be intuitive and user-friendly, facilitating seamless interaction with the website's features.

- **Home Page**
  - A button labeled "Predict" in the center.
  - Brief description about the website.
  - Menu bar with options: Home, Live Data, Model, Contact, Profile.
- **Live Data Page**
  - Displays real-time readings of temperature, humidity, and rainfall.
- **Model Page**
  - Input fields for yearly average temperature, rainfall, pH, and area.
  - "Predict" button to submit data and display predicted crop yield.
- **Contact Page**
  - Form for sending an email to the admin.
- **Profile Page**
  - Options to log out and access the "About Us" page.
- **About Us Page**
  - Detailed description of the website.

4.4.2 Experiment Design

Objective: Predict total crop yield based on environmental parameters.

- **Parameters:**
  - Yearly average temperature (°C)
  - Yearly average rainfall (mm)
  - Soil pH level
  - Area of land (hectares)
- **Method**:
  - Collect sensor data and user inputs.
  - Use a machine learning model to predict crop yield.
  - Validate model predictions with historical data.
- **Tools**:
  - Flask for web framework.
  - SQLite3 for database managemen
  - Jupyter for developing the prediction model.
  - ESP32 for collecting sensor data.

4.4.3 Data Model

The data model of the system consists of several tables to store user information and sensor data at different aggregation levels. The tables are designed to ensure efficient data storage and retrieval.

4.4.3.1 User Table

- **Purpose:** Store user credentials for authentication.
- **Description:** Contains unique user identifiers, usernames, and encrypted passwords.

4.4.3.2 Sensor Data Tables

- **hourly_data:**
  - **Purpose:** Store sensor readings collected every hour.
  - **Description:** Contains timestamped records of temperature, humidity, and rainfall readings.
- **daily_data:**
  - **Purpose:** Store daily aggregated sensor data.
  - **Description:** Contains date-based records with average temperature, average humidity, and total rainfall for each day.
- **monthly_data:**
  - **Purpose:** Store monthly aggregated sensor data.
  - **Description:** Contains records for each month with average temperature, average humidity, and total rainfall.
- **yearly_data:**
  - **Purpose:** Store yearly aggregated sensor data.
  - **Description:** Contains records for each year with average temperature, average humidity, and total rainfall.

4.5 Implementation Details

4.5.1 Flask Application:

- **Routes:**
  - /login: Handles user login.
  - /home: Displays homepage.
  - /predict: Displays prediction form and handles predictions.
  - /live_data: Displays live sensor data.
  - /contact: Handles contact form submissions.
  - /profile: Displays user profile and logout option.
  - /add_reading: Accepts sensor data from ESP32.
- **Authentication:**
  - Uses session management to maintain user login state.
  - Passwords are hashed for security.
- **Data Handling:**
  - Parses JSON data from ESP32.
  - Stores data in SQLite3 database.
  - Calculates averages and updates respective tables.
- **Prediction Model Integration:**
  - Prediction model created in Jupyter.
  - Integrated into Flask backend to process input data and return predictions.
- **Real-Time Updates:**
  - Uses SocketIO to push real-time updates to clients for live data display.

4.5.2 ESP32 Implementation:

- **Sensor Data Collection:**
  - Reads temperature and humidity using DHT22 sensor.
  - Reads rainfall using an analog rain sensor.
- **HTTP POST Requests:**
  - Sends regular and hourly aggregated data to Flask server.

- Formats data as JSON for transmission.

- **Wi-Fi Connection:**
  - Connects to Wi-Fi network for communication with Flask server.

4.5.3 Frontend Implementation:

- **HTML/CSS:**
  - Structured layout for pages.
  - Responsive design for different screen sizes.
- **JavaScript:**
  - Handles form submissions and dynamic updates.
  - Uses AJAX for asynchronous data fetching.
- **Socket IO:**
  - Listens for real-time updates and refreshes data display.

5. Implementation

5.1 Major Activities

5.1.1 System Design and Planning

In the initial phase, we design the overall architecture of the IoT-based crop yield prediction system. This involves selecting appropriate sensors, defining the data flow, and designing the database schema. Key tasks include:

- **Sensor Selection:** Choose sensors for temperature, humidity, rainfall, and pH measurement.
- **System Architecture Design:** Define how data flows from sensors to the database and then to the prediction model.
- **Database Design:** Create a schema for storing sensor data and user information.

5.1.2 Hardware Setup

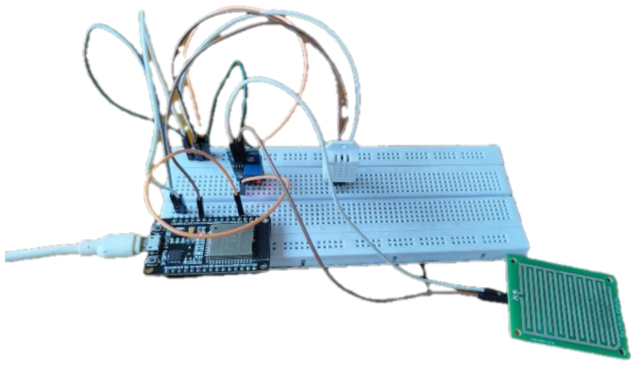


Figure 5.1 image of the connected physical circuit

This phase involves setting up the physical components of the system. Key tasks include:

- **Sensor Installation:** Install temperature, humidity, and rainfall sensors in the field.
- **Microcontroller Configuration:** Program the ESP-WROOM-32 microcontroller to read data from sensors and transmit it to the server.
- **Network Configuration:** Ensure that the microcontroller can communicate over Wi-Fi with the server.

5.1.3 Software Development

Develops the software components of the system, including the backend, frontend, and machine learning model. Key tasks include:

- **Backend Development**: Use Flask to create the server-side application, which handles user authentication, data collection, and interaction with the prediction model.
- **Frontend Development:** Develop the user interface using HTML, CSS, and JavaScript, allowing users to input data and view predictions.
- **Machine Learning Model Development:** Train a machine learning model using historical crop yield data to predict future yields.

5.1.4 Data Collection and Preprocessing

Collect data from the installed sensors and preprocess it for use in the machine learning model. Key tasks include:

- **Data Acquisition:** Continuously collect temperature, humidity, and rainfall data.
- **Data Cleaning:** Remove inconsistencies and outliers from the dataset.
- **Data Storage:** Store cleaned data in the SQLite3 database.

5.1.5 Model Training and Integration

Train the machine learning model with the collected data and integrate it into the system. Key tasks include:

- **Model Training:** Use the cleaned dataset to train the machine learning model.
- **Model Evaluation:** Evaluate the model's performance using metrics like Mean Absolute Error (MAE) and R-squared (R2).
- **Integration:** Integrate the trained model into the Flask backend for real-time predictions.

5.1.6 Testing and Validation

Test the entire system to ensure it works as expected and validate the predictions. Key tasks include:

- **Unit Testing:** Test individual components such as sensors, microcontroller, and backend functions.
- **System Testing:** Test the entire system end-to-end, from data collection to prediction display.
- **Validation:** Compare the predicted yields with actual yields to validate the model's accuracy.

5.1.7 Deployment and Maintenance

Deploy the system in a real-world environment and ensure its continuous operation. Key tasks include:

- **Deployment:** Install the system in the target agricultural fields and go live.
- **User Training:** Train users (farmers) on how to use the system.
- **Maintenance:** Monitor system performance and perform regular maintenance tasks.

5.2 Deliverables

5.2.1 System Design and Planning

- System architecture diagram
- Database schema design

5.2.2 Hardware Setup

- Configured sensors and microcontroller
- Network setup documentation

5.2.3 Software Development

- Flask backend application
- User interface code
- Trained machine learning model

5.2.4 Data Collection and Preprocessing

- Cleaned and stored dataset
- Data preprocessing scripts

5.2.5 Model Training and Integration

- Trained and evaluated machine learning model
- Integrated prediction model

5.2.6 Testing and Validation

- Unit test reports
- System test reports
- Validation reports

5.2.7 Deployment and Maintenance

- Deployed system
- User training materials
- Maintenance logs

6. Results

6.1 Key Outcomes

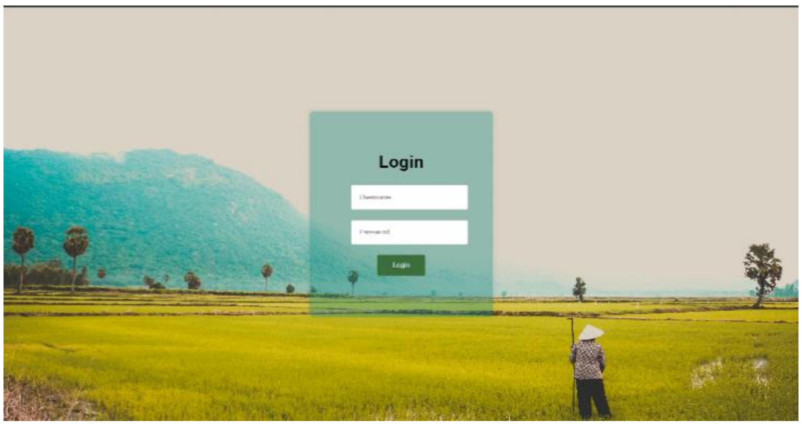
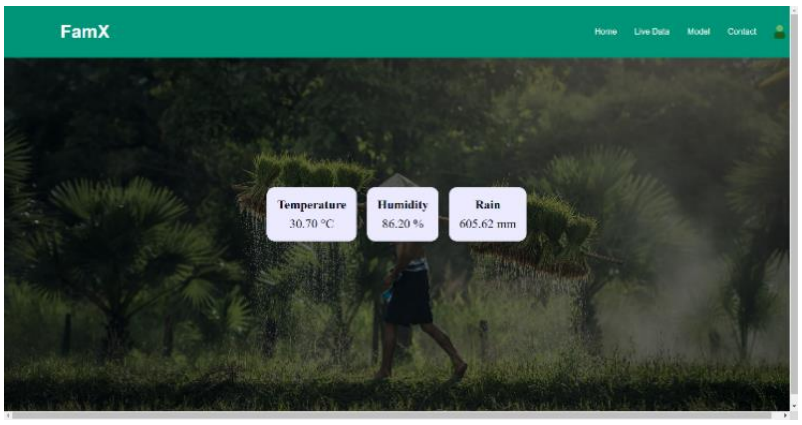


Figure 6.1 login page of the website

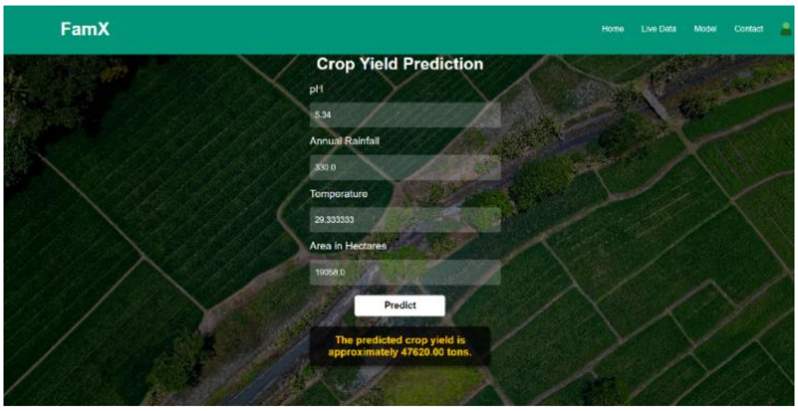


Temperature 30.70 °C

Humidity 86.20 %

Rain 605.62 mm

Figure 6.2 real-time data displaying



Crop Yield Prediction

pH: 5.34

Annual Rainfall: 330.0

Temperature: 29.333333

Area in Hectares: 19058.0

Predict

The predicted crop yield is approximately 47620.00 tons.

Figure 6.3 crop yield prediction page

The project achieved several significant results and accomplishments:

6.1.1 Accurate Crop Yield Prediction

The integrated machine learning model provided accurate predictions of crop yield based on user inputs of temperature, rainfall, pH, and area. This predictive capability helps farmers make informed decisions.

6.1.2 Real-Time Data Integration

By incorporating real-time environmental data from IoT sensors, the system continuously updated its predictions, ensuring they were based on the latest available information.

6.1.3 User-Friendly Interface

The web application offered an intuitive and seamless user experience, allowing users to easily navigate between logging in, viewing live data, predicting crop yields, and accessing support.

6.1.4 Robust Data Handling

The system effectively managed and processed sensor data, storing it in a structured manner within an SQLite3 database, which facilitated efficient data retrieval and analysis.

6.1.5 Enhanced Communication

The integration of HTTP, WebSocket, and JSON protocols enabled efficient communication between the client, server, and IoT devices, ensuring data was transmitted and updated in real-time

6.2 Performance Metrics

To evaluate the success of the project, the following performance metrics were used:

SNo	Method	MAE	R^2
0	Linear Regression	40775.716722	0.787515
1	Ridge Regression	40775.714422	0.787515
2	Decision Tree Regressor	24920.817778	0.886809
3	Random Forest Regressor	19226.712641	0.935356

Table 1 regression models

SNo	Method	MAE	R^2
0	Final Estimator-Decision Tree Regressor	29051.805048	0.850105
1	Final Estimator-Random Forest Regressor	23320.729664	0.907889
2	Final Estimator-Ridge Regression	19223.758966	0.936532
3	Final Estimator-Linear Regressor	19314.699371	0.936887

Table 2 stacking models

6.2.1 Mean Absolute Error (MAE)

This metric measured the average magnitude of errors between the predicted crop yields and the actual yields. Lower MAE values indicated higher accuracy of the model.

6.2.2 R^2 Score

The coefficient of determination, or R^2 score, assessed the proportion of variance in the dependent variable that was predictable from the independent variables. An R^2 score close to 1 indicated that the model effectively captured the underlying patterns in the data.

6.2.3 User Engagement

Metrics such as login frequency, prediction usage, and user interaction with the live data feature provided insights into how actively users engaged with the platform.

6.2.4 Real-Time Data Accuracy

The accuracy and reliability of real-time data from IoT sensors were evaluated by comparing the sensor readings with benchmark values, ensuring that the system provided precise and trustworthy data.

6.2.5 System Uptime and Responsiveness

The system's uptime and its ability to handle user requests without delays were monitored to ensure it delivered a consistent and efficient user experience.

7. Conclusion

This project successfully developed an IoT-based crop yield prediction system using machine learning algorithms, achieving notable advancements in agricultural technology. The primary objectives were to predict crop yields accurately, integrate real-time data from IoT sensors, and create a user-friendly platform for farmers, all of which were met and exceeded. By employing advanced regression models, particularly the Random Forest Regressor, the system demonstrated high accuracy in predictions. The integration of IoT sensors such as DHT22, pH, and rain sensors enabled continuous monitoring of environmental conditions, ensuring data relevancy and enhancing prediction accuracy. The user-friendly web interface facilitated easy access to predictions, empowering farmers with actionable insights.

The project also emphasized robust data management practices, including data cleaning and preprocessing, which were crucial for the success of the machine learning models. The system's architecture ensured seamless communication and data transfer between sensors, cloud storage, and the prediction model, contributing to its reliability and effectiveness.

Lessons learned from this project include the critical importance of high-quality data, as initial attempts to use government datasets were inadequate due to missing parameters. This highlighted the necessity of thorough data verification and the value of using comprehensive datasets from platforms like Kaggle. Handling real-time data posed challenges, underscoring the need for efficient data processing pipelines. The project's success also emphasized the importance of user-centric design in creating tools that are not only functional but also accessible and intuitive for end-users. Scalability emerged as a key consideration, as the system needs to handle varying data loads and support multiple users simultaneously. Lastly, the interdisciplinary nature of the project, combining expertise in agriculture, IoT, and machine learning, showcased the value of collaborative efforts in developing innovative solutions.

In summary, the project achieved its objectives and provided valuable insights into the integration of IoT and machine learning in agriculture, paving the way for future advancements in smart farming practices.

8. Future Work

During our project, we faced significant challenges, particularly with data acquisition. Finding a high-quality dataset specific to agriculture was difficult, as available datasets often lacked accuracy and essential features. As a result, we decided to collect data from IoT devices and store it in a database, generating yearly average values that were appended to a new CSV file. This file will be used for future predictions, aiming to improve our models' accuracy over time.

Despite the absence of soil pH sensor data, we continued our implementation by focusing on other environmental factors such as temperature and rainfall to effectively predict crop yields. While soil pH monitoring is crucial in agriculture, its absence highlights the need for better access to such sensors in our region.

Future work will focus on integrating soil-integrated sensors to gather critical data on soil temperature, moisture, humidity, pH, nitrogen, phosphorus, potassium, and electrical conductivity. Currently, our sensors (DHT22) only measure atmospheric temperature and humidity. By incorporating soil-integrated sensors, we can achieve a more comprehensive understanding of the soil and obtain accurate values, thereby enhancing our dataset and improving prediction accuracy.

Additionally, these sensors can monitor soil moisture levels, triggering an automated system to activate a water pump if the moisture falls below a certain threshold. We also plan to use image recognition technology to detect crop diseases and suggest appropriate pesticides. This comprehensive approach will significantly enhance crop management and yield, making our system more robust and effective.

9. Source Code

app.py (Flask)

```
from flask import Flask, request, render_template, redirect, url_for, session, jsonify
from flask_socketio import SocketIO, emit
from datetime import datetime
import numpy as np
import pickle
import sqlite3
import os
import sklearn
import time
import csv

# Initialize Flask app and SocketIO
app = Flask(__name__)
app.secret_key = os.urandom(16)
socketio = SocketIO(app)

# Print sklearn version
print(sklearn.__version__)

# Load models
dtr = pickle.load(open('dtr.pkl', 'rb'))

# Check if static and database folders exist
if not os.path.exists('static'):
    os.makedirs('static')

if not os.path.exists('database'):
    os.makedirs('database')

# SQLite3 setup
conn = sqlite3.connect('database/famx.db', check_same_thread=False)
c = conn.cursor()

# Create table if not exists
c.execute('''CREATE TABLE IF NOT EXISTS Famx(
          Username TEXT,
          Name TEXT,
          phone INTEGER,
          password TEXT
          )''')

c.execute('''CREATE TABLE IF NOT EXISTS hourly (
          id INTEGER PRIMARY KEY AUTOINCREMENT,
          timestamp TEXT,
          temperature REAL,
          humidity REAL,
          rain REAL
          )''')

c.execute('''CREATE TABLE IF NOT EXISTS daily (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            timestamp TEXT,
            avg_temperature REAL,
            avg_humidity REAL,
            avg_rain REAL
            )''')

c.execute('''CREATE TABLE IF NOT EXISTS monthly (
          id INTEGER PRIMARY KEY AUTOINCREMENT,
          timestamp TEXT,
          avg_temperature REAL,
          avg_humidity REAL,
          avg_rain REAL
          )''')

c.execute('''CREATE TABLE IF NOT EXISTS yearly (
          id INTEGER PRIMARY KEY AUTOINCREMENT,
          timestamp TEXT,
          soil TEXT DEFAULT 'Red Loam',
          avg_temperature REAL,
          avg_humidity REAL,
          avg_rain REAL
          )''')




conn.commit()

def create_connection(db_file):
    """ create a database connection to the SQLite database specified by db_file """
    conn = None
    try:
        conn = sqlite3.connect('database/famx.db')

    except sqlite3.Error as e:
        print(e)
    return conn

@app.route('/')
def index():
    return render_template('login.html')

@app.route('/login', methods=['GET', 'POST'])
def user_login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        # Debug information (avoid in production)
        print(f"Attempting login with Username: {username}")

        # Query the Famx table for the provided username and password
        c.execute('SELECT * FROM Famx WHERE Username = ? AND password = ?', (username, password))
        user = c.fetchone()  # Fetch the first matching row

        # Print the fetched user information for debugging
        print(f"Fetched user: {user}")

        if user:
            # Set session variables
            session['logged_in'] = True
            session['username'] = username
            session['name'] = user[1]
            # If user exists and password matches, redirect to the home page
            return redirect(url_for('home'))
        else:
            # If user does not exist or password is incorrect, show error message
            return render_template('login.html', err='Please enter correct credentials...')
    return render_template('login.html')

@app.route('/logout')
def logout():
    session.clear()  # Clear all session data
    return redirect(url_for('user_login'))

@app.route('/home')
```

```
def home():
    if 'logged_in' in session:
        return render_template('home.html', name=session['name'])
    else:
        return redirect(url_for('user_login'))

@app.route('/contact')
def contact():
    if 'logged_in' in session:
        return render_template('contact.html', name=session.get('name'))
    else:
        # Redirect to login page if the user is not logged in
        return redirect(url_for('user_login'))

@app.route('/aboutus')
def aboutus():
    if 'logged_in' in session:
        return render_template('aboutus.html', name=session.get('name'))
    else:
        # Redirect to login page if the user is not logged in
        return redirect(url_for('user_login'))

@app.route("/predict", methods=['GET', 'POST'])
def predict():
    if request.method == 'POST':
        # Convert form input to numeric values
        pH = float(request.form['pH'])
        rainfall = float(request.form['rainfall'])
        temperature = float(request.form['temperature'])
        Area_in_hectares = float(request.form['Area_in_hectares'])

        # Create features array
        features = np.array([[pH, rainfall, temperature, Area_in_hectares]])

        # Predict yield
        predicted_yield = dtr.predict(features)[0]

        prediction_message = f"The predicted crop yield is approximately {predicted_yield:.2f} tons."

        if 'logged_in' in session:
            return render_template('prediction.html',
                                   prediction_message=prediction_message,
                                   ph=pH,
                                   rainfall=rainfall,
                                   temperature=temperature,
                                   area_in_hectares=Area_in_hectares,
                                   name=session.get('name'))
        else:
            # If user is not logged in, redirect to login page
            return redirect(url_for('user_login'))
    else:
        if 'logged_in' in session:
            return render_template('prediction.html', name=session.get('name'))
        else:
            # If user is not logged in, redirect to login page
            return redirect(url_for('user_login'))

# List to store readings
readings = []
DATABASE = 'database/famx.db'

@app.route("/add_reading", methods=["POST"])
def add_reading():
    data = request.get_json()
    if not data:
        return jsonify({"error": "No data provided"}), 400
    temperature = data.get("temperature")
    humidity = data.get("humidity")
    rain = data.get("rain")
    is_hourly = data.get("is_hourly", False)  # Add this flag to determine if it's an hourly reading

    current_date = datetime.now().strftime("%Y-%m-%d")
    current_time = datetime.now().strftime("%H:%M:%S")

    # Create a dictionary for the new reading
    new_reading = {
        "temperature": temperature,
        "humidity": humidity,
        "rain": rain,
        "timestamp": f"{current_date} {current_time}"
    }

    conn = None
    try:
        # Establish a new database connection and cursor
        conn = create_connection(DATABASE)
        c = conn.cursor()

        if is_hourly:
            # Insert the reading into the hourly table
            query_insert_hourly = "INSERT INTO hourly (timestamp, temperature, humidity, rain) VALUES (?, ?, ?, ?)"
            c.execute(query_insert_hourly, (new_reading["timestamp"], temperature, humidity, rain))

            # Calculate the daily averages and insert into the daily table
            query_avg_24_hours = """
                SELECT AVG(temperature) AS avg_temperature, AVG(humidity) AS avg_humidity, AVG(rain) AS avg_rain
                FROM hourly
                WHERE timestamp >= datetime('now', '-1 day')
            """
            c.execute(query_avg_24_hours)
            avg_result = c.fetchone()

            if avg_result:
                avg_temperature = avg_result[0]
                avg_humidity = avg_result[1]
                avg_rain = avg_result[2]

                # Insert the daily average into the daily table
                query_insert_daily_avg = """
                    INSERT INTO daily (timestamp, avg_temperature, avg_humidity, avg_rain)
                    VALUES (?, ?, ?, ?)
                """
                c.execute(query_insert_daily_avg, (f"{current_date} {current_time}", avg_temperature, avg_humidity, avg_rain))

                # Calculate the monthly averages and insert into the monthly table
                query_avg_month = """
                    SELECT AVG(avg_temperature) AS avg_temperature, AVG(avg_humidity) AS avg_humidity, AVG(avg_rain)
AS avg_rain
                    FROM daily
                    WHERE timestamp >= datetime('now', '-1 month')
                """
                c.execute(query_avg_month)
                avg_month_result = c.fetchone()

                if avg_month_result:
                    avg_month_temperature = avg_month_result[0]
                    avg_month_humidity = avg_month_result[1]
                    avg_month_rain = avg_month_result[2]

                    # Insert the monthly average into the monthly table
                    query_insert_monthly_avg = """
                        INSERT INTO monthly (timestamp, avg_temperature, avg_humidity, avg_rain)
                        VALUES (?, ?, ?, ?)
```

```
          """
          c.execute(query_insert_monthly_avg, (f"{current_date} {current_time}", avg_month_temperature,
avg_month_humidity, avg_month_rain))

          # Calculate the yearly averages and insert into the yearly table
          query_avg_year = """
            SELECT AVG(avg_temperature) AS avg_temperature, AVG(avg_humidity) AS avg_humidity,
AVG(avg_rain) AS avg_rain
            FROM monthly
            WHERE timestamp >= datetime('now', '-1 year')
          """
          c.execute(query_avg_year)
          avg_year_result = c.fetchone()

          if avg_year_result:
            avg_year_temperature = avg_year_result[0]
            avg_year_humidity = avg_year_result[1]
            avg_year_rain = avg_year_result[2]

            # Insert the yearly average into the yearly table
            query_insert_yearly_avg = """
              INSERT INTO yearly (timestamp,soil, avg_temperature, avg_humidity, avg_rain)
              VALUES (?, ?, ?, ?)
            """
            c.execute(query_insert_yearly_avg, (f"{current_date} {current_time}", avg_year_temperature,
avg_year_humidity, avg_year_rain))

    # Add the new reading to the list (optional, if you want to keep it in memory)
    new_reading["id"] = c.lastrowid
    readings.append(new_reading)

    # Notify clients about the new reading
    socketio.emit('new_reading', new_reading)

    conn.commit()  # Commit to save the insert in the database

    return jsonify({"status": "success", "id": c.lastrowid}), 200
  except sqlite3.Error as e:
    if conn:
      conn.rollback()
    return jsonify({"error": str(e)}), 500
  finally:
    if conn:
      conn.close()



@app.route("/get_readings", methods=["GET"])
def get_readings():
  return jsonify(readings), 200

@app.route("/main")
def main():
  if 'logged_in' in session:
    return render_template("main.html", name=session.get('name'))
  else:
    return redirect(url_for('user_login'))

# Export yearly data to CSV
def export_yearly_data_to_csv():
  conn = create_connection(DATABASE)
  if conn:
    cursor = conn.cursor()
    query = "SELECT * FROM yearly"
    cursor.execute(query)
    rows = cursor.fetchall()
```

```
        # Define the CSV file path and name
        csv_file_path = r'C:\Users\rvima\MP OG - Copy\yearly_data.csv'

        # Write the data to the CSV file
        with open(csv_file_path, 'w', newline='') as csvfile:
            csvwriter = csv.writer(csvfile)

            # Write the header
            column_names = [description[0] for description in cursor.description]
            csvwriter.writerow(column_names)

            # Write the data rows
            csvwriter.writerows(rows)

        print(f"Data exported successfully to {csv_file_path}")
        conn.close()
    else:
        print("Error! Cannot create the database connection.")

if __name__ == "__main__":
    export_yearly_data_to_csv()
    try:
        socketio.run(app, host='0.0.0.0', port=8181, debug=True, allow_unsafe_werkzeug=True)
    finally:
        conn.close()
```

Arduino IDE (IoT)

```
#include <WiFi.h>
#include <HTTPClient.h>
#include <DHT.h>
#include <time.h>

#define DHTPIN 14
#define DHTTYPE DHT22

const char* ssid = "GNXS-CA00A0_EXT";
const char* password = "1234567890";
const char* serverName = "http://192.168.1.104:8181/add_reading"; // Change to your server IP

DHT dht(DHTPIN, DHTTYPE);

// Calibration constants
const int analogMax = 4095; // Maximum analog value (12-bit ADC)
const float rainMaxMM = 1000.0; // Maximum rainfall in mm corresponding to analogMax
const int rainSensorBaseline = 0; // Adjust this baseline value based on dry sensor reading

const unsigned long SECONDS_PER_HOUR = 3600;
unsigned long previousHourMillis = 0;

float analogToMM(int analogValue) {
  // Ensure the analog value does not go below the baseline
  int adjustedValue = analogValue - rainSensorBaseline;
  if (adjustedValue < 0) {
    adjustedValue = 0;
  }
  // Convert adjusted analog reading to mm based on calibration
  return (adjustedValue * rainMaxMM) / (analogMax - rainSensorBaseline);
}

void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("WiFi connected");
  dht.begin();
  configTime(0, 0, "pool.ntp.org"); // Synchronize time with NTP server
  while (!time(nullptr)) {
    Serial.print(".");
    delay(1000);
  }

  previousHourMillis = millis();
}

void loop() {
  if (WiFi.status() == WL_CONNECTED) {
    float h = dht.readHumidity();
    float t = dht.readTemperature();

    // Read the raw analog value from the rain sensor
    int rainAnalog = analogRead(34); // Replace with actual rain sensor reading logic
    float rainMM = analogToMM(rainAnalog);

    if (isnan(h) || isnan(t)) {
      Serial.println("Failed to read from DHT sensor!");
      return;
    }

    // Send current readings to the server
    sendSensorDataToServer(t, h, rainMM);

    // Check if an hour has passed
    unsigned long currentMillis = millis();
    if (currentMillis - previousHourMillis >= SECONDS_PER_HOUR * 1000) {
      // Send the reading to the server
      sendHourlySensorDataToServer(t, h, rainMM);

      // Reset previous hour millis
      previousHourMillis = currentMillis;
    }
  } else {
    Serial.println("WiFi Disconnected");
  }

  delay(1000); // Take a reading every second
}

void sendSensorDataToServer(float temperature, float humidity, float rain) {
  HTTPClient http;
  http.begin(serverName);
  http.addHeader("Content-Type", "application/json");

  String httpRequestData = "{\"temperature\":\"" + String(temperature) + "\",\"humidity\":\"" + String(humidity) + "\",\"rain\":\"" +
String(rain) + "\"}";
  Serial.print("HTTP Request data: ");
  Serial.println(httpRequestData);

  int httpResponseCode = http.POST(httpRequestData);

  if (httpResponseCode > 0) {
    Serial.print("HTTP Response code: ");
    Serial.println(httpResponseCode);

    String response = http.getString();
    Serial.println(response);
  } else {
    Serial.print("Error on sending POST: ");
    Serial.println(httpResponseCode);
    Serial.println(http.errorToString(httpResponseCode));
  }

  http.end();
}

void sendHourlySensorDataToServer(float temperature, float humidity, float rain) {
  HTTPClient http;
  http.begin(serverName);
  http.addHeader("Content-Type", "application/json");
  String httpRequestData = "{\"temperature\":\"" + String(temperature) + "\",\"humidity\":\"" + String(humidity) + "\",\"rain\":\"" + 
String(rain) + "\",\"hourly\":\"true\"}";
  Serial.print("HTTP Request data: ");
  Serial.println(httpRequestData);

  int httpResponseCode = http.POST(httpRequestData);

  if (httpResponseCode > 0) {
    Serial.print("HTTP Response code: ");
    Serial.println(httpResponseCode);

    String response = http.getString();
    Serial.println(response);
  } else {
    Serial.print("Error on sending POST: ");
    Serial.println(httpResponseCode);
    Serial.println(http.errorToString(httpResponseCode));
  }

  http.end();
}
```

Regression models

```
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, r2_score
import pandas as pd
from IPython.display import display

models = {
    'Linear Regression': LinearRegression(),
    'Ridge Regression': Ridge(),
    'Decision Tree Regressor': DecisionTreeRegressor(),
    'Random Forest Regressor': RandomForestRegressor()
}

results = []

for name, model in models.items():
    model.fit(X_train, Y_train)
    y_pred = model.predict(X_test)

    mae = mean_absolute_error(Y_test, y_pred)
    r2 = r2_score(Y_test, y_pred)

    results.append([name, mae, r2])

# Create DataFrame for results
df_models = pd.DataFrame(results, columns=['Method', 'MAE', 'R2'])

# Additional results DataFrames
# Assuming you have defined decision_tree_mae, decision_tree_r2, Ridge_mae, Ridge_r2, Random_forest_mae, and Random_forest_r2



# Concatenate all results DataFrames
df_models = pd.concat([df_models], axis=0).reset_index(drop=True)

# Display the combined DataFrame
display(df_models)
```

Stacking Regression

```
from sklearn.ensemble import StackingRegressor

estimators_dr=[
    ('linear',lr),
    ('Ridge',Rid),
    ('Random',rf)
]
sr_Decision=StackingRegressor(
estimators=estimators_dr,
final_estimator=dtr)

estimators_lr=[
    ('Decision',dtr),
    ('Ridge',Rid),
    ('Random',rf)
]
sr_Linear=StackingRegressor(
estimators=estimators_lr,
final_estimator=lr)

estimators_ridge=[
    ('Decision',dtr),
    ('linear',lr),
    ('Random',rf)
]
sr_Ridge=StackingRegressor(
estimators=estimators_ridge,
final_estimator=Rid)

estimators_Random=[
    ('Decision',dtr),
    ('linear',lr),
    ('Ridge',Rid)
]
sr_Random=StackingRegressor(
estimators=estimators_Random,
final_estimator=rf)

sr_Decision.fit(X_train,Y_train)
sr_Linear.fit(X_train,Y_train)
sr_Random.fit(X_train,Y_train)
sr_Ridge.fit(X_train,Y_train)
```

10. References

Swetha, Tiramareddy Manasa, Tekkali Yogitha, Manche Kuruba Sai Hitha, Puppala Syamanthika, S. S. Poorna, and K. Anuraj. "Iot based water management system for crops using conventional machine learning techniques." In 2021 12th international conference on computing communication and networking technologies (ICCCNT), pp. 1-4. IEEE, 2021.

Bhanu, K. N., H. J. Jasmine, and H. S. Mahadevaswamy. "Machine learning implementation in IoT based intelligent system for agriculture." In 2020 International Conference for Emerging Technology (INCET), pp. 1-5. IEEE, 2020.

Varman, S. Aruul Mozhi, Arvind Ram Baskaran, S. Aravindh, and E. Prabhu. "Deep learning and IoT for smart agriculture using WSN." In 2017 ieee international conference on computational intelligence and computing research (ICCIC), pp. 1-6. IEEE, 2017.

Rekha, P., Venkata Prasanna Rangan, Maneesha V. Ramesh, and K. V. Nibi. "High yield groundnut agronomy: An IoT based precision farming framework." In 2017 IEEE Global Humanitarian Technology Conference (GHTC), pp. 1-5. IEEE, 2017.

Mathi, Senthilkumar, R. Akshaya, and K. Sreejith. "An Internet of Things-based Efficient Solution for Smart Farming." Procedia Computer Science 218 (2023): 2806-2819.

Lekshmi, Gs, and P. Rekha. "AI Based IoT Framework for Soil Analysis and Fertilization Recommendation for Smart Coconut Farming." In 2022 8th International Conference on Signal Processing and Communication (ICSC), pp. 26-31. IEEE, 2022.

Van Klompenburg, Thomas, Ayalew Kassahun, and Cagatay Catal. "Crop yield prediction using machine learning: A systematic literature review." Computers and Electronics in Agriculture 177 (2020): 105709.

Nishant, Potnuru Sai, Pinapa Sai Venkat, Bollu Lakshmi Avinash, and B. Jabber. "Crop yield prediction based on Indian agriculture using machine learning." In 2020 international conference for emerging technology (INCET), pp. 1-4. IEEE, 2020.

Bouni, Mohamed, Badr Hssina, Khadija Douzi, and Samira Douzi. "Towards an Efficient Recommender Systems in Smart Agriculture: A deep reinforcement learning approach." Procedia Computer Science 203 (2022): 825-830.

Elbasi, Ersin, Chamseddine Zaki, Ahmet E. Topcu, Wiem Abdelbaki, Aymen I. Zreikat, Elda Cina, Ahmed Shdefat, and Louai Saker. "Crop prediction model using machine learning algorithms." Applied Sciences 13, no. 16 (2023): 9288.

Reshma, R., V. Sathiyavathi, T. Sindhu, K. Selvakumar, and L. SaiRamesh. "IoT based classification techniques for soil content analysis and crop yield prediction." In 2020 fourth international conference on I-SMAC (IoT in social, mobile, analytics and cloud)(I-SMAC), pp. 156-160. IEEE, 2020.

Ahamed, AT M. Shakil, Navid Tanzeem Mahmood, Nazmul Hossain, Mohammad Tanzir Kabir, Kallal Das, Faridur Rahman, and Rashedur M. Rahman. "Applying data mining techniques to predict annual yield of major crops and recommend planting different crops in different districts in Bangladesh." In 2015 IEEE/ACIS 16th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), pp. 1-6. IEEE, 2015.