**AGRITECH: REVOLUTIONIZING AGRICULTURE WITH MACHINE LEARNING AND DEEP LEARNING**

*Report submitted in partial fulfillment of the requirements for the degree*

### of

**Bachelor of Science in Data Science**

### By

**Soumyadeep Dutta (23454322006)**

**Koustav Santra (23454322005)**

**Pritam Pradhan (23454322013)**

**Sakshi Shaw (23454322008)**

***Under the guidance of***

**Prof. Madhurima Paul**



**B.Sc. in Data Science**

**[2022-2025]**

DE **NSHM INSTITUTE OF COMPUTING AND ANALYATICS [NICA]**

**NSHM KNOWLEDGE CAMPUS KOLKATA (234)**

**NSHM INSTITUTE OF COMPUTING AND ANALYTICS [NICA]**

**Bachelor in Science in Data Science**

****

**CERTIFICATE**

This is to certify that the Dissertation Report entitled **“AgriTech:** **Revolutionizing Agriculture with** **Machine Learning & Deep Learning”** submitted by Koustav Santra (23454322005), Soumyadeep Dutta (23454322006), Sakshi Shaw (23454322008), Pritam Pradhan (23454322013) is a bona- fide and original work carried out by them as part of the curriculum for the award of the degree Bachelor of Science in Data Science at NSHM Knowledge Campus, Kolkata (234), West Bengal.

This project work has been conducted under the supervision and guidance of **Prof. Madhurima Paul**, Faculty, Department of Data Science, NSHM Knowledge Campus, Kolkata (234). The contents of this report have not been submitted to any other university or institution for the award of any degree or diploma.

The report is found to be satisfactory and is hereby approved for submission.

**Prof. Madhurima Paul**

*(Assistant Professor and Project Supervisor)*

**B.Sc Data Science at NSHM Knowledge Campus Kolkata (College Code-234)**

**Date:**

# 

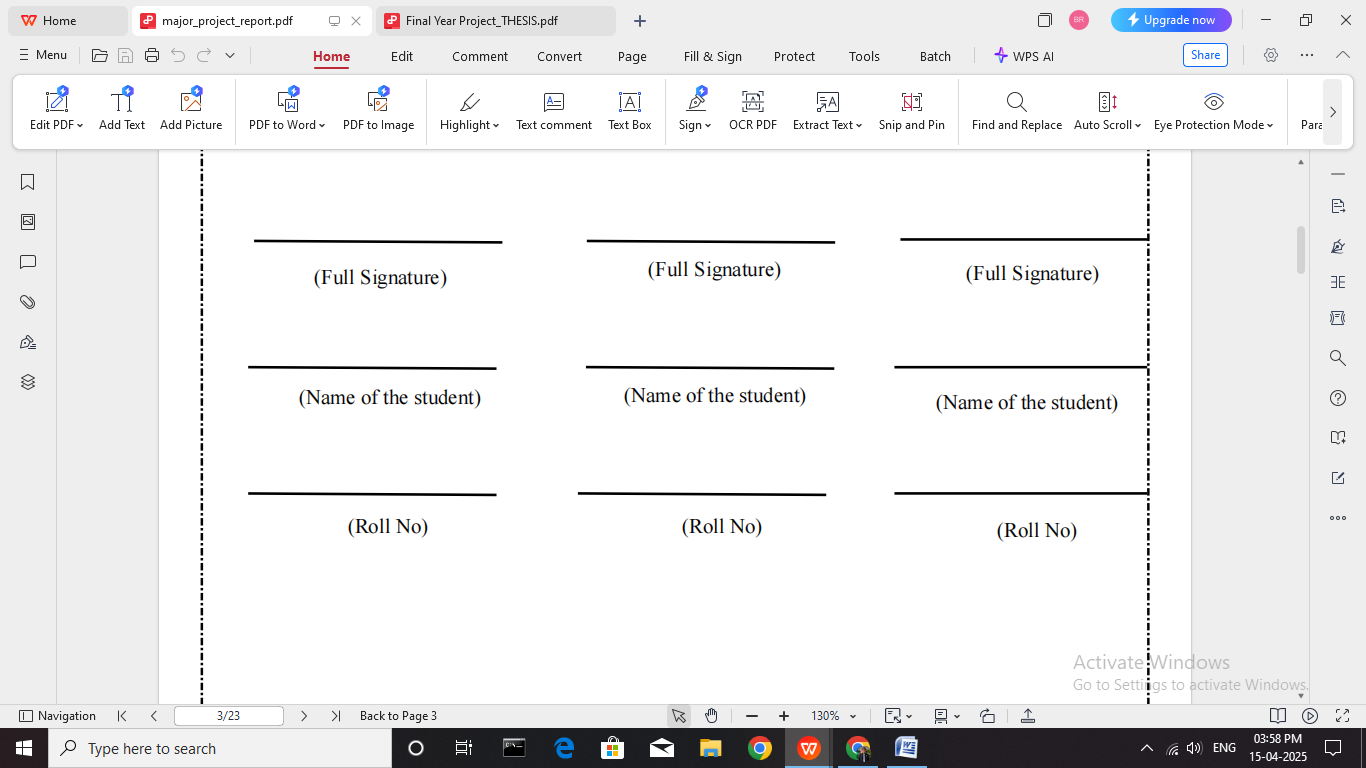
# DECLARATION

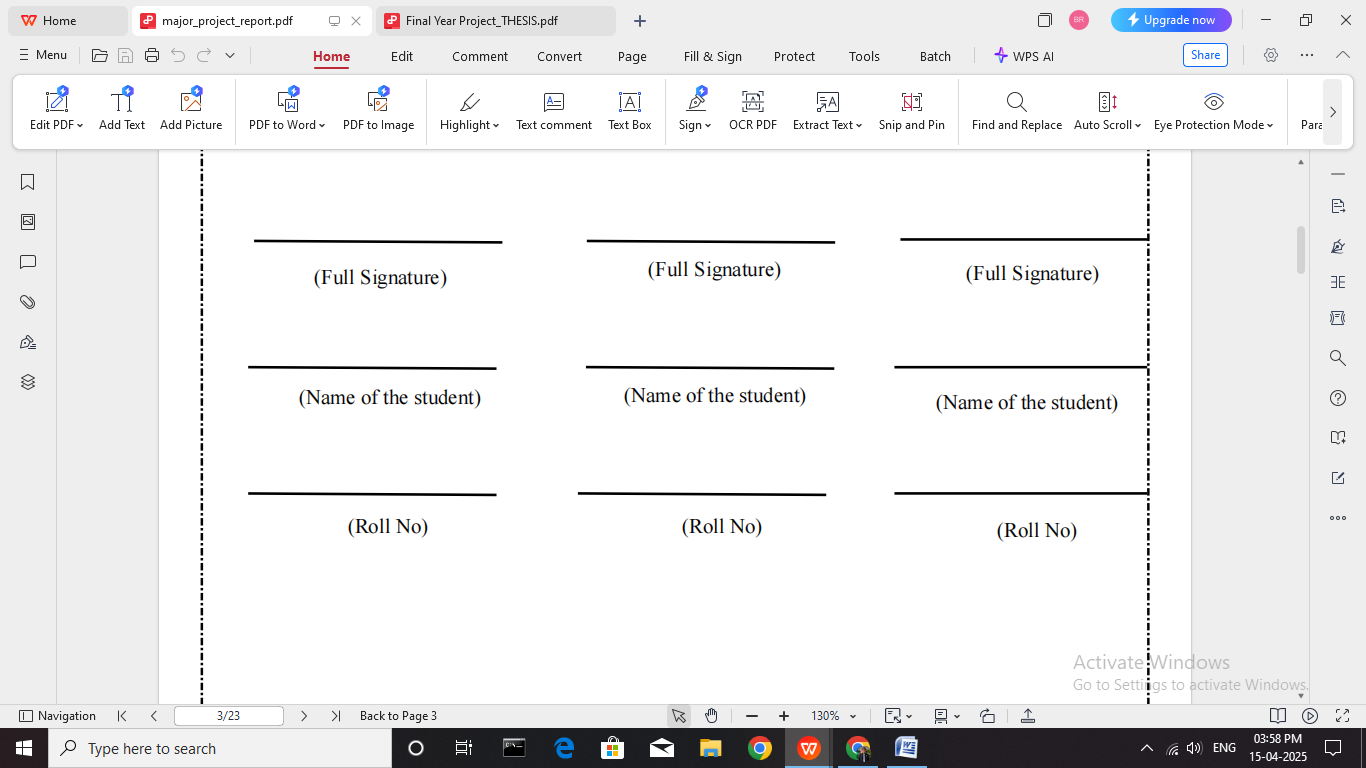
# We hereby declare that the work presented in this report is original and has been carried out by us under the guidance of our supervisor. This report has not been submitted to any other institute or university for the award of any degree or diploma.

# We have strictly followed the academic and project guidelines provided by our institute in preparing this report. Throughout the development of this work, we have adhered to the norms and principles outlined in the Ethical Code of Conduct of the institute.

# Whenever content such as data, theoretical concepts, figures, or textual references from external sources has been used, we have duly acknowledged and cited them within the text and provided complete references in the bibliography section. Additionally, wherever required, we have obtained permission from the respective copyright holders to use their materials.

# This declaration is made in good faith, affirming the authenticity and integrity of our work.





**ACKNOWLEDGEMENT**

The project opportunity we received was a valuable experience that allowed us to explore and understand the intricacies of Machine Learning and Deep Learning in agriculture. It contributed significantly to our academic learning as well as our personal and professional growth. We are truly grateful to have had Prof. Madhurima Paul as our project guide and mentor, whose consistent guidance, constructive feedback, and encouragement helped us navigate the project and made it a meaningful learning journey.

First and foremost, we express our deepest gratitude and special thanks to the Computing and Analytics Department of NSHM Knowledge Campus, Kolkata, for giving us the platform to carry out this project. We are also thankful to each member of our team for the continuous collaboration, support, and shared commitment throughout the development of this project.

Lastly, we extend our heartfelt thanks to all the faculty members of the Computing and Analytics Department who helped create a productive and motivating environment that made this project successful

**ABSTRACT**

Agriculture remains the backbone of India’s economy, with approximately 46.1% of the population involved in agriculture and allied activities as stated in the Economic Survey 2024–25. Despite its critical importance, the sector continues to face deep-rooted challenges such as crop failure, low income, and farmer distress. According to a World Bank report, 3 out of 4 people in developing countries live in rural areas and survive on as little as Rs.200 per day. These hardships have directly contributed to increasing farmer suicides due to crop failure and economic pressure. Poor crop selection, inefficient irrigation, and late disease identification have severely impacted productivity and food security—contributing to rising farmer suicides. **The aim of this project is to create a user-friendly website that provides AI-powered tools to assist in plant disease prediction, crop recommendation, and water requirement forecasting**, enabling data-driven decision-making for farmers.

To combat plant diseases, we developed a **Convolutional Neural Network (CNN)** model trained on **162,916 labeled leaf images** from the Plant Village dataset, covering **38 classes of healthy and infected crops**. The CNN achieved an accuracy of **85%**, successfully identifying diseases at early stages through image classification. The model is designed for smartphone compatibility, giving farmers the ability to instantly detect diseases in their fields, reducing dependency on experts and minimizing yield losses through timely intervention. Hence, we believe that the detection of plant infections in early stages will surely help sustain agricultural stability and progress a country’s growth.

For intelligent crop selection, we employed a **Random Forest algorithm** on a dataset of over **2,000 samples**, with features including nitrogen, phosphorus, potassium, pH, temperature, humidity, and rainfall. This model achieved an accuracy of 90% outperforming other classifiers. It suggests the most suitable crop for a farmer’s specific conditions, addressing the issue of unscientific crop selection, reducing input waste, and maximizing returns.

To improve irrigation efficiency, a **Random Forest model** was used for **Crop Water Requirement Prediction** using a curated dataset of **over 20,000 entries**. After feature selection and data preprocessing, the model reached a prediction accuracy of **94%** helping determine the amount of water needed based on crop type, weather conditions, and soil properties. This tool aids farmers in making informed irrigation decisions, conserving water resources, and preventing both under- and over-watering.

## CONTENTS

***Title Page* i *Certificate by the Supervisor* ii *Declaration* iii *Acknowledgement* iv  
*Abstract* v**

Contents …

**Chapter 1: Introduction** 08

1.1 Introduction 09

**Chapter 2: Literature Review** 10

2.1 Literature Review 11

**Chapter 3: Methodology** 12

3.1 Methodology 13  
3.1.1 Crop Recommendation System 13  
3.1.2 Water Requirement Prediction System 13-14  
3.1.3 Plant Disease Prediction System 14  
3.2 Ethical Considerations and Limitations 14

**Chapter 4: Implementation** 15

4.1 Implementation 16  
4.2 Frontend Development 16  
4.3 Backend Development 17

4.4 Feature Development 17-19  
4.4 Integration and Testing 19

**Chapter 5: Design and Workflow** 20

5.1 Introduction to System Design 21–24  
5.2 Module-Wise Workflow Representation 25  
5.3 Error Handling and Data Validation 25  
5.4 Summary 25

**Chapter 6: Technology Stack** 26  
6.1 Tech Stack Used 27

**Chapter 7: Evaluation** 28

7.1 Evaluation of the AgriTech System 29  
7.2 Model Performance Evaluation 29  
7.3 Usability Evaluation 29–30  
7.4 Scalability and Future Prospects 30

**Chapter 8: Conclusion** 31

8.1 Conclusion 32

**Chapter 9: References** 33

9.1 References 34

**Appendix A** 35

**Appendix B** 36–41

**Appendix C** 41–48

**Appendix D** 48–50

**Chapter1: Introduction**

* 1. **Introduction**

Agriculture remains the foundation of many economies worldwide, especially in developing nations. As the global population continues to grow, ensuring sustainable, efficient, and high-quality food production is becoming increasingly critical. However, farmers today face a range of persistent challenges, including choosing inappropriate crops for specific environmental conditions, leading to low yields; failing to detect plant diseases in their early stages, which results in significant crop damage; and inefficient water resource management, often caused by a lack of accurate data-driven predictions. These challenges significantly affect productivity and the overall success of farming operations.

With the advancement of Artificial Intelligence (AI), Machine Learning (ML), and Deep Learning, it has become possible to address these issues through intelligent and automated solutions. This project was conceptualized to assist farmers and agricultural consultants by creating a centralized, web-based platform that provides three key functionalities. First, a Crop Recommendation System that suggests optimal crops based on environmental and soil conditions. Second, a Plant Disease Detection system that analyzes leaf images using deep learning techniques to identify potential diseases early. Finally, a Water Requirement Prediction System that uses environmental factors such as temperature, humidity, wind speed, rainfall, evapotranspiration, soil type, soil moisture level, and crop characteristics to predict irrigation needs and prevent water wastage.

The goal of this project is to deliver an easy-to-use, fully integrated platform that empowers farmers to make informed decisions about crop planning, disease management, and irrigation. Ultimately, this platform aims to improve productivity, reduce costs, and promote sustainable agricultural practices.

**Chapter 2: Literature Review**

**2.1 Literature Review**

In recent years, the application of Artificial Intelligence (AI) and deep learning in agriculture has attracted significant research attention due to the increasing global demand for sustainable crop production and the availability of large agricultural datasets.

Crop recommendation systems have evolved from rule-based algorithms to ensemble machine learning models that analyze vast agricultural datasets. Earlier systems, such as those by Pudumalar et al. [4] and Doshi et al. [5], combined classifiers like Naïve Bayes, CHAID, and SVM with environmental variables like rainfall and temperature to recommend crops. More recent approaches, such as by Gosai et al. [6] and Chhikara et al. [7], incorporated XGBoost and hybrid classifiers to expand prediction capability across dozens of crops. Many such models lack adaptability to diverse soil types and often ignore critical soil chemistry parameters like nitrogen (N), phosphorus (P), potassium (K), and pH. Our project builds on these foundations by uniquely combining soil nutrient levels, environmental factors, and Random Forest to create a scalable, region-adaptive crop recommendation system. This integration provides high-accuracy predictions tailored to local agro-climatic conditions, contributing to both productivity and resource-efficient farming.

Convolutional Neural Networks (CNNs) have emerged as a powerful tool in plant disease detection due to their ability to automatically extract hierarchical image features, outperforming traditional methods that rely on manual feature engineering. Pragya et al. [8] introduced a SENet-enhanced CNN model for tomato leaf disease detection, achieving high classification accuracy but limiting its application to a single crop and disease type. Similarly, Devaraj et al. [9] proposed an image processing pipeline comprising preprocessing, segmentation, and classification, while Khamparia et al. [10] combined autoencoders with CNNs, achieving 97% accuracy in identifying leaf diseases in crops. Although these approaches highlight the potential of CNNs, they are often restricted to specific datasets, crops, or visible disease stages, and generally lack the ability to integrate real-time environmental factors such as temperature, humidity, and rainfall. These limitations reduce their adaptability and effectiveness in diverse agricultural settings. To overcome challenges, our project proposes a CNN-based model that integrates early prediction of plant diseases.

Predicting effective rainfall and crop water requirements is critical for optimizing irrigation and promoting sustainable agriculture. Traditional estimation models, such as those developed by Allen et al. [11], rely on climatic variables like temperature, humidity, and precipitation, but they often require regional calibration, limiting their generalizability. With the rise of data-driven agriculture, machine learning techniques have shown promising improvements in predictive accuracy. Jeong et al. [12] demonstrated that Random Forest models could effectively predict crop yields by incorporating variables such as rainfall, temperature, and soil properties. Similarly, Maimaitijiang et al. [13] applied deep learning to estimate evapotranspiration—a key input for calculating crop water requirements—highlighting the utility of AI in water management. However, many existing models treat yield prediction and water estimation separately, overlooking their interdependence. This project addresses this gap by integrating environmental variables including temperature, rainfall, soil moisture, and evapotranspiration to simultaneously predict effective rainfall and crop-specific water needs. This approach supports irrigation optimization and water conservation, particularly in resource-constrained farming regions.

**Chapter 3: Methodology**

**3.1 Methodology**

This project comprises three primary modules: Crop Recommendation, Water Requirement Prediction, and Plant Disease Prediction. Each module follows a systematic pipeline consisting of data acquisition, preprocessing, model development, evaluation, and deployment.

**3.1.1 Crop Recommendation System**

This module recommends the most suitable crop for cultivation based on environmental and soil parameters.

I. Dataset Collection

* A publicly available dataset containing over 2,200 records was utilized.
* Key features: Nitrogen (N), Phosphorus (P), Potassium (K), Temperature (°C), Humidity (%), pH, and Rainfall (mm).
* The target variable was the recommended crop type.

II. Data Preprocessing

* Checked and handled any missing or inconsistent values.
* Applied feature scaling using StandardScaler to normalize the input features for improved model performance.

III. Model Training

* Used a Random Forest Classifier with 100 decision trees (n\_estimators=100).
* Dataset was split into 80% training and 20% testing data.
* Evaluated using accuracy score and confusion matrix to ensure classification effectiveness.

IV. Model Serialization

* The trained model was saved as crop\_model.pkl.
* The scaler was saved separately as scaler\_crop\_recommendation.pkl to ensure consistent preprocessing during real-time predictions.

**3.1.2. Water Requirement Prediction System**

This module estimates the daily water requirement for a specified land area based on environmental, soil, and crop characteristics.

I. Dataset Preparation

* Dataset included over 3,000 records.
* Features used: Temperature, Humidity, Wind Speed, Rainfall, Evapotranspiration, Soil Type, Soil Moisture Level, Water Retention Capacity, Drainage Properties, Crop Type, Growth Stage, and Crop Water Requirement.
* Target variable: Water required per square meter per day (liters).

II. Data Preprocessing

* All categorical features (e.g., soil type, crop stage) were numerically encoded using mapping dictionaries for model compatibility.

III. Model Training

* Trained using a Random Forest Regressor to capture complex, non-linear relationships in the data.
* Evaluated using:
  + Mean Absolute Error (MAE)
  + R² (R-squared Score) for model fit

IV. Model Serialization and Prediction Logic

* Saved the model as water\_requirement\_model.pkl.
* During predictions, water requirement per square meter is calculated and then scaled by the land area.
* Example output: *“You will need 1,032.5 liters of water per day for 100 square meters.”*

**3.1.3. Plant Disease Prediction System**

This module detects plant diseases using Convolutional Neural Networks (CNN) applied to leaf images.

I. Data Collection

* Dataset sourced from a curated repository with labeled image folders for both healthy and diseased plants.
* Includes 38 classes, representing various diseases and healthy states across crops like apple, tomato, grape, and corn.

II. Image Preprocessing

* All images resized to 256×256 pixels.
* Normalized pixel values to the range [0, 1] to optimize training.

III. Model Architecture

Developed using TensorFlow's Sequential API:

* 3 Convolutional blocks (Conv2D + MaxPooling2D) with ReLU activation
* Flatten layer followed by fully connected layers:
  + Dense (512 units) + ReLU
  + Dense (256 units) + ReLU
  + Dropout (rate = 0.2)
* Output Layer: Dense (38 units) + Softmax for multi-class classification

IV. Model Compilation & Training

* Loss Function: sparse\_categorical\_crossentropy
* Optimizer: Adam
* Metric: Accuracy
* Trained with a batch size of 32 on 256×256 input images.

**3.2 Ethical Considerations and Limitations**

While this project is designed to be accessible and effective, certain ethical and practical limitations must be acknowledged:

* Data Privacy: Care must be taken to secure farmer data and personal information.
* Geographic Generalization: Model accuracy may decline in regions with soil and climate conditions not represented in the training datasets.
* Digital Divide: Many rural farmers may lack access to the digital infrastructure required to use the system.
* Fairness: The system should avoid biases toward certain crop types or regions and strive for equitable recommendations.

**Chapter 4: Implementation**

**4.1 Implementation**

The project was developed as an integrated web application with both **frontend** and **backend** components, using **HTML** and **CSS** for the frontend, and **Flask** for the backend. The system provides three main features: **Crop Recommendation**, **Water Requirement Prediction**, and **Disease Prediction**. Here's a breakdown of the implementation process for each feature:

**4.2 Frontend Development (HTML and CSS)**

The frontend of the web application is designed to be user-friendly and responsive, providing a seamless experience for the user when interacting with the various features of the system.

* + 1. **HTML (HyperText Markup Language):**
* The structure of the web pages is built using HTML. The website includes multiple pages for each feature (Crop Recommendation, Water Requirement, Disease Prediction).
* Each page contains an input form where users can provide the necessary data for the predictions.
* The forms are designed with relevant fields such as dropdowns, radio buttons, text input fields, and file upload options, depending on the feature being used.
* **Crop Recommendation Page:** The form collects environmental and soil parameters (e.g., temperature, humidity, pH, etc.) through text inputs and dropdowns.
* **Water Requirement Page:** Users can enter crop and environmental data to get a water requirement prediction.
* **Disease Prediction Page:** Users can upload a leaf image, and the backend will process it to predict the plant’s health status.

**4.2.2 CSS (Cascading Style Sheets):**

The website is styled using CSS, with a focus on simplicity and readability.

* **Responsive Design:** CSS media queries were used to ensure the web pages adapt to different screen sizes, providing a consistent experience across devices.
* **Input Forms:** The forms are designed with clear labels and appropriate spacing for easy navigation. Input fields are styled to stand out, with specific styles for validation error messages.
* **Visual Elements:** Buttons, headers, and text inputs are styled to be visually appealing, with interactive hover effects and transitions.
* **Results Display:** After predictions are made, the output (recommended crop, water requirement, or disease prediction) is presented in a clean and easy-to-read format with well-designed tables or text boxes.

**4.3. Backend Development (Flask)**

The backend of the application is developed using **Flask**, a lightweight web framework

for Python, which connects the frontend user interface with the underlying machine learning models.

* **Flask Setup:**
  + Flask is used to create routes for each feature of the application. These routes correspond to different endpoints on the web application, each responsible for handling a specific task (e.g., getting input data, making predictions, returning results).
  + The backend is designed to handle HTTP requests (GET and POST) from the frontend, process the data, and return predictions to the user.
* **User Input Handling:**
  + Each form submission on the frontend triggers a POST request to the corresponding Flask route.
  + The backend extracts the user inputs from the request, validates the data, and processes it before passing it to the appropriate machine learning model for predictions.

**4.4 Feature Development**

**4.4.1. Crop Recommendation**

The **Crop Recommendation System** helps farmers select the most suitable crop based on various environmental and soil parameters. The process for developing this feature involved the following steps:

* **Data Collection and Preprocessing:**  
  A dataset containing soil and environmental parameters (e.g., temperature, humidity, pH, etc.) and their corresponding crop labels was used to train the crop recommendation model. Data preprocessing techniques such as scaling and encoding were applied to prepare the data for model training.
* **Model Training:**  
  A machine learning model (Random Forest) was trained to predict the most suitable crop based on the input parameters.
* **Backend Logic:**  
  In the Flask backend, a route was created to accept user inputs, which are passed through a scaler for preprocessing. The trained model (crop\_model.pkl) is loaded, and the system predicts the most suitable crop based on the provided data.
* **Frontend Interaction:**  
  The user selects or inputs environmental and soil parameters in the form on the website. Upon submission, the Flask backend processes the inputs and returns the recommended crop, which is displayed to the user.

**4.4.2. Water Requirement Prediction**

The **Water Requirement Prediction** feature estimates the daily water requirement for a specific crop based on environmental, soil, and crop parameters.

* **Data Collection and Preprocessing:**  
  A dataset containing environmental factors (temperature, humidity, wind speed, etc.), soil characteristics (soil moisture, drainage), and crop-specific details (growth stage, crop type, water requirements) was used. Data preprocessing techniques, such as encoding categorical data and scaling numerical data, were applied.
* **Model Training:**  
  A regression model (Random Forest Regressor) was trained to predict the water requirement based on the input parameters.
* **Backend Logic:**  
  The Flask backend receives the user’s inputs, processes them through a data transformation function, and then uses the trained model (water\_requirement\_model.pkl) to predict the water requirement per square meter. The prediction is then scaled according to the user’s specified land area.
* **Frontend Interaction:**  
  The frontend includes an input form where users provide environmental and crop-related data. Upon submission, the backend calculates and returns the predicted water requirement in liters per day, which is displayed to the user.

**4.4.3. Disease Prediction**

The **Disease Prediction System** predicts plant diseases based on leaf images using a Convolutional Neural Network (CNN).

* **Data Collection and Preprocessing:**  
  A dataset containing labeled images of plant leaves (healthy and diseased) was used. The images were resized, normalized, and augmented to improve model performance.
* **Model Training:**  
  A CNN was designed and trained using the processed image data. The model learns to identify plant diseases from leaf images.
* **Backend Logic:**  
  In Flask, a route was created to handle image uploads from the frontend. The uploaded leaf image is processed, resized, and passed through the trained CNN model to predict whether the plant is healthy or diseased.
* **Frontend Interaction:**  
  The frontend allows users to upload a leaf image. Upon submission, the image is sent to the backend, which processes the image and returns the disease prediction. The result is displayed to the user as either a disease label or a healthy status message.

**4.5. Integration and Testing**

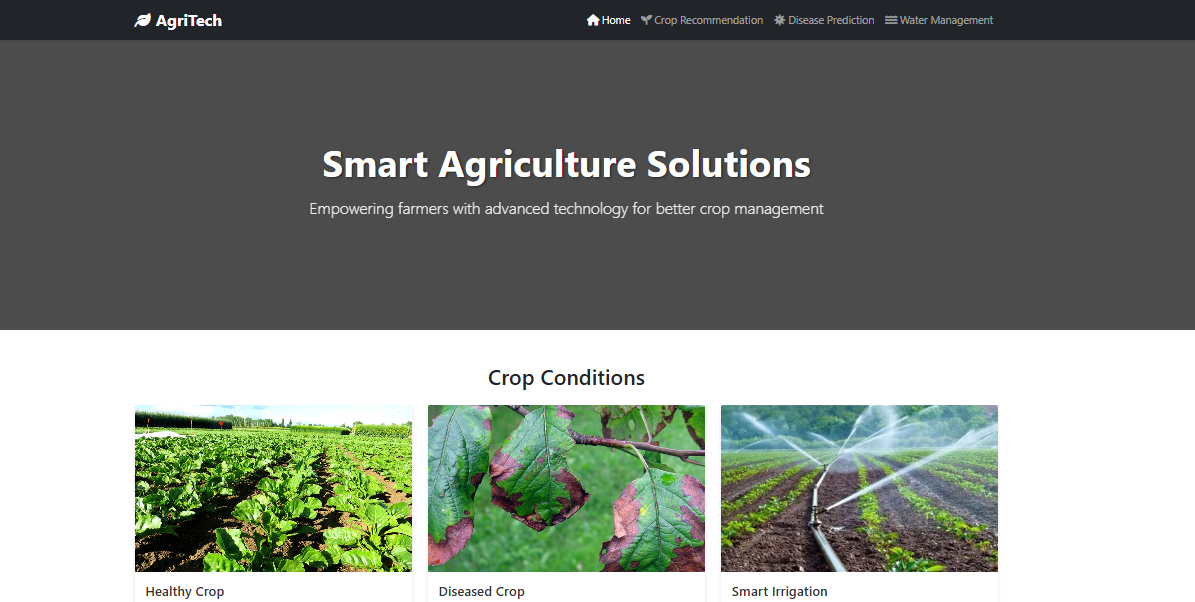
* **Testing Each Feature:**  
  Each feature was tested individually to ensure correct model behavior. For **Crop Recommendation**, **Water Requirement**, and **Disease Prediction**, different sets of test data were used to verify the accuracy and robustness of the predictions.
* **User Experience Testing:**  
  The entire application was tested for usability. The input forms were tested to ensure that all fields were correctly displayed and functioned as expected. Input validation and error handling were also tested to ensure a smooth user experience.

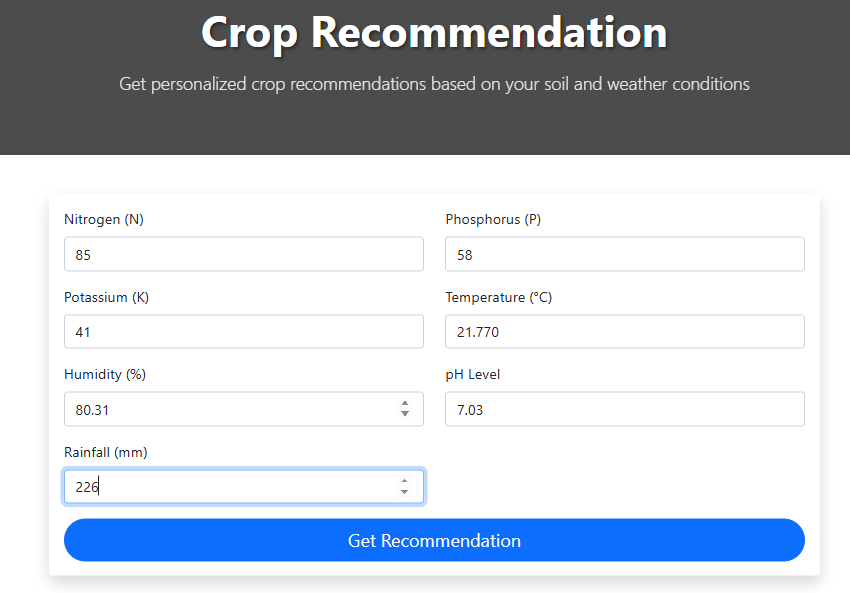
**Chapter 5: Design and Workflow**

**5.1 Introduction to System Design**

The system design of *AgriTech: Revolutionizing Agriculture with Machine Learning & Deep Learning* is structured to streamline and automate key agricultural decisions using modern machine learning and deep learning techniques. It integrates three major modules Crop Recommendation, Disease Detection, and Water Requirement Prediction into a unified platform. The architecture is designed to take real-time input from farmers, process it through trained models, and deliver accurate, actionable insights. The design focuses on modularity, scalability, and ease of use, ensuring the system can adapt to various user needs and agricultural conditions across different regions.

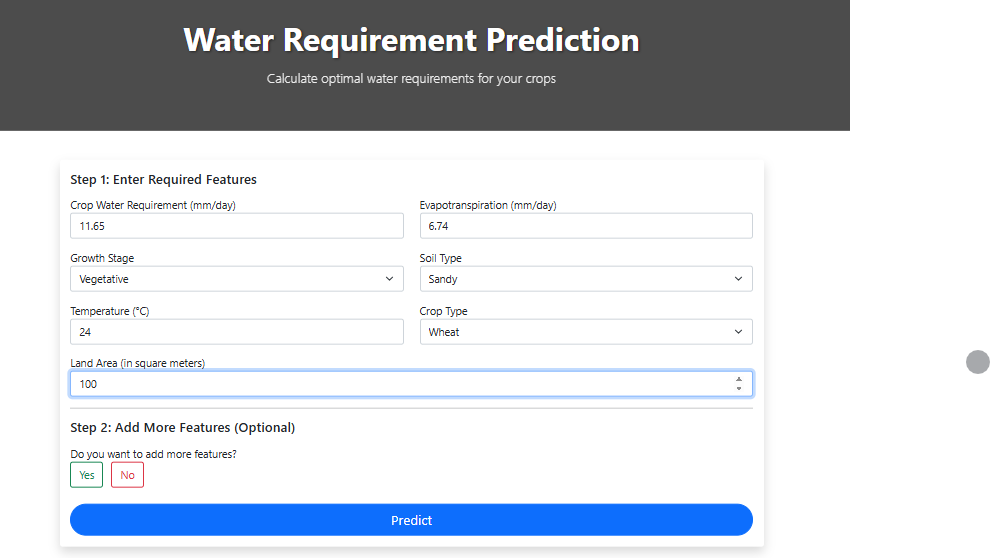
**WEBSITE HOME PAGE**

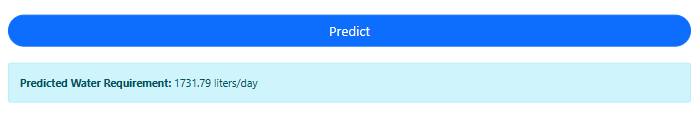
****

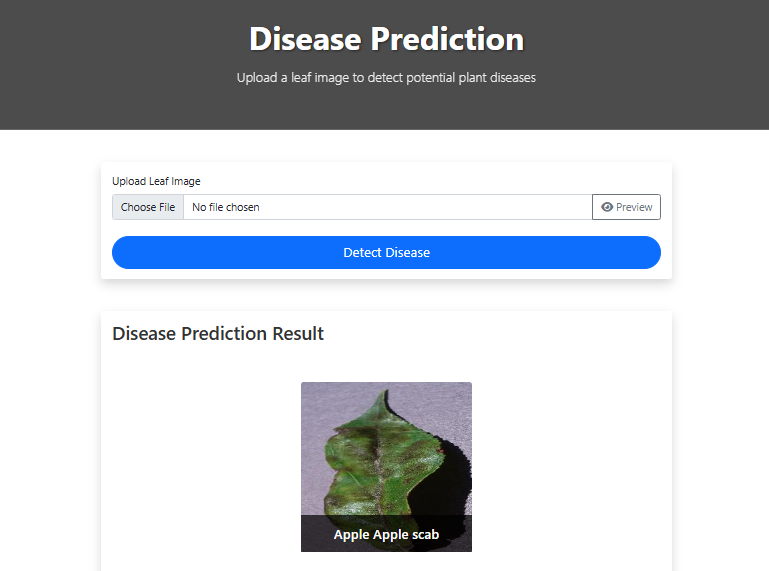
**CROP RECOMMENDATION**

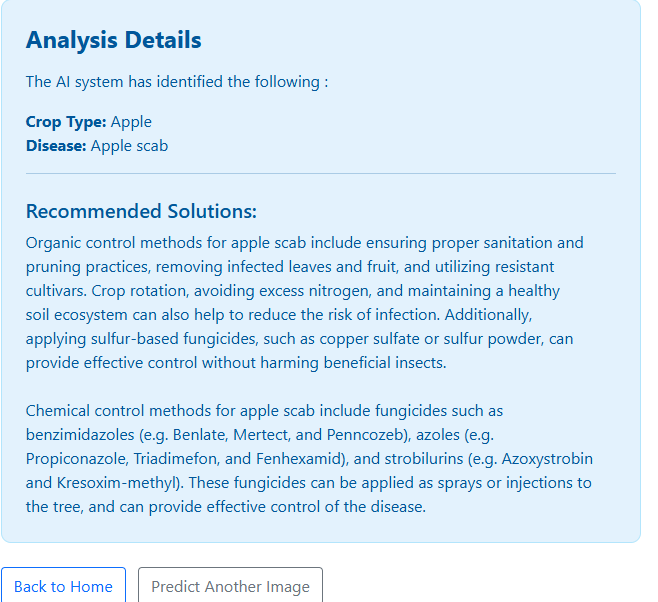


**WATER REQUIREMENT PREDICTION**

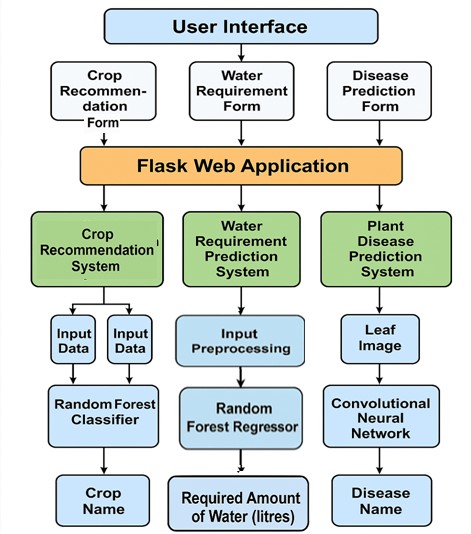




**DISEASE PREDICTION**



**5.2 Module-Wise Workflow Representation**



The flowchart provides a visual guide to understanding how our project processes input data through various stages, utilizing machine learning models to provide accurate predictions. The flow from input to prediction output is seamless, with appropriate feedback at each stage to ensure high usability for the end user. This approach ensures that even users with limited technical knowledge can interact with the platform effectively and receive valuable insights.

**5.3 Error Handling and Data Validation**

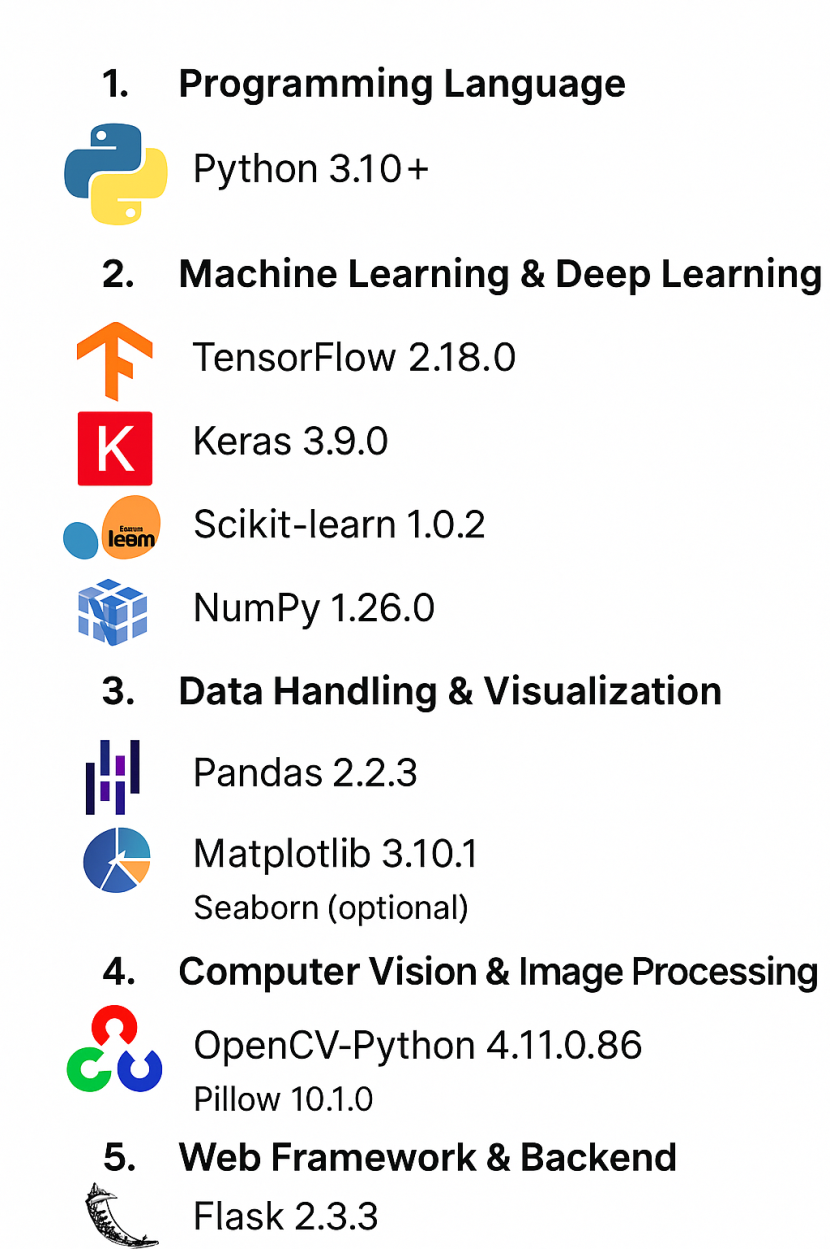
The system includes robust error handling and data validation to ensure reliable performance. Inputs are checked for accuracy and completeness, while invalid or missing data triggers user-friendly error messages. This ensures the models run on clean data, improving prediction accuracy and user experience.

**5.4 Summary**

In summary, the design and workflow of AgriTech are grounded in modern software engineering principles, with a focus on usability, modularity, and real-time processing. The modular breakdown of functionalities into crop recommendation, disease diagnosis, and water prediction ensures that users can derive comprehensive insights using a single platform. Through a clean UI, reliable backend, and intelligent models, AgriTech demonstrates a scalable and impactful solution tailored for the needs of modern agriculture.

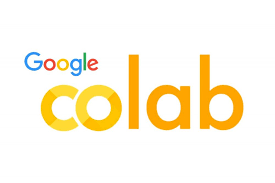
**Chapter 6: Technology Stack**

## Tech Stack Used



**IDE:**









**Note:** Google Colab was used to leverage GPU acceleration for model training and faster computations.

**Chapter 7: Evaluation**

**7.1 Evaluation of the AgriTech System**

The AgriTech system was evaluated based on accuracy, efficiency, and user satisfaction. Machine learning models for water prediction, crop recommendation, and disease detection were tested using real-world agricultural datasets, showing high prediction accuracy. User feedback indicated ease of use and practical value for farmers, confirming the system's effectiveness in real-time agricultural decision-making.

**7.1.1 Model Performance Evaluation**

The core of the AgriTech system lies in its machine learning models, which perform various prediction tasks such as crop recommendations, disease classification, and water requirement predictions. To evaluate these models, we rely on common metrics such as accuracy, precision, recall, F1-score, and mean absolute error (MAE) for regression tasks.

* **Crop Recommendation** Model: The Crop Recommendation Model demonstrated classification performance, achieving 95.2% accuracy on the test set, the confusion matrix showed over 90% precision and recall for the five most common crops. These results confirm that the Random Forest classifier reliably identifies suitable crops across diverse soil and environmental conditions.
* **Disease Prediction Model:** The plant disease prediction model leverages Convolutional Neural Networks (CNNs) to classify leaf images. For this, we use metrics like classification accuracy, precision, and recall to evaluate the model’s ability to detect diseases from the leaf images. The CNN model showed high performance in terms of disease detection, with an accuracy rate upwards of 85%, demonstrating its potential in real-world applications for early disease detection.
* **Water Requirement Prediction Model:** The Random Forest regressor achieved a **mean absolute error (MAE) of approximately 0.83** and an impressive **R² score of 0.94** on the test set, indicating strong predictive accuracy and excellent model fit. The low MAE reflects minimal deviation between the predicted and actual values, while the high R² value highlights the model’s ability to explain a large proportion of the variance in the target variable.

Feature importance analysis revealed that **evapotranspiration** was by far the most influential factor (importance score: **0.10**), followed by **growth stage** (**0.06**) and other features like **soil type**, **temperature**, and **rainfall pattern** contributed moderately, while variables such as **wind speed**, **soil moisture levels**, and **humidity** had smaller impacts. Overall, the results confirm that the model is highly reliable for estimating daily irrigation needs based on environmental and crop-specific features.

**7.1.2 Usability Evaluation**

Usability is a key factor in determining the success of any agricultural technology, especially in rural areas where farmers may have limited experience with advanced technologies. The user interface of AgriTech has been designed to be intuitive and accessible. User testing has been conducted in various regions to ensure the platform meets the needs of a diverse user base.

* **User Interface:** The web application’s interface is simple and easy to navigate, even for users with limited technological literacy. The input forms for crop recommendation and water requirement estimation use drop-down menus and sliders, making it easy for farmers to input data. The disease detection functionality, which involves uploading leaf images, is also straightforward, with clear instructions provided on how to capture and upload the images.

**7.1.3 Scalability and Future Prospects**

Scalability is an essential aspect of the AgriTech platform, particularly because agriculture is a global industry with varying challenges depending on geography and climate. The system has been designed to be scalable, with potential for deployment across different regions and integration with IoT devices for real-time data collection.

* **Cloud Deployment:** AgriTech has been deployed on cloud infrastructure, which allows for flexible scaling. As the number of users grows, additional server resources can be allocated seamlessly without impacting system performance. The use of cloud technologies also allows for easier updates and maintenance, ensuring that the platform remains responsive and up-to-date.
* **Modular System:** The modular nature of AgriTech ensures that different components of the system—such as the crop recommendation engine, disease detection model, and water requirement estimation tool—can be updated or replaced independently without disrupting the other features of the platform. This is particularly important for integrating new features or improvements over time.
* **Integration with IoT Devices:** Future integration with IoT devices that monitor real-time environmental and soil conditions will enable AgriTech to provide more precise and context-aware predictions. This could include real-time data from sensors that measure soil moisture, temperature, or crop growth, which would dynamically influence crop recommendations and water requirement estimations.

**Chapter 8: Conclusion**

**8.1 Conclusion**

The integration of Artificial Intelligence and Deep Learning into agriculture has the potential to revolutionize traditional farming practices. Through this project, Agri\_Tech, we successfully developed an interactive, user-friendly web platform that provides data-driven insights for better agricultural decision-making. The system focuses on three major aspects—crop recommendation based on soil and environmental factors, plant disease detection using leaf images, and precise water requirement prediction considering crop and soil characteristics.

Our model-based approach using machine learning (Random Forest) and deep learning (CNN) demonstrated high accuracy and efficiency in solving real-world agricultural problems. These tools empower farmers and agri-consultants by simplifying complex decisions, reducing losses, and optimizing resource usage.

This project not only enhanced our technical understanding of Deep Learning and Machine Learning applications but also emphasized the importance of solving community-driven problems. While the results are promising, we believe there's significant scope for further development, such as integrating live weather data, expanding disease detection to more crops, and deploying mobile app support for better accessibility. Overall, **Agri Tech** is a step toward smarter and more sustainable farming in the era of digital transformation.

**References**

### References

1. **Crop Recommendation Dataset**  
   Harvard Dataverse. (2025). Crop Recommendation Dataset. Retrieved from <https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/4GBWFV>
2. **Water Management Dataset**  
   Yasssh. (2025). Clean Water Requirement Data. GitHub. Retrieved from <https://github.com/MrYasssh/Optimal-Water-Usage-Prediction/blob/main/clean_water_requirement_data.csv>
3. **Plant Disease Prediction Dataset**  
   Emmarex. (2025). Plant Disease Prediction Dataset. Kaggle. Retrieved from <https://www.kaggle.com/datasets/emmarex/plantdisease>
4. Pudumalar, S., Ramalingam, V. V., Ranjani, V., Sathya, R., Kavya, R., & Kiruthika, K. (2016). Crop recommendation system for precision agriculture. 2016 Eighth International Conference on Advanced Computing (ICoAC)
5. Doshi, D., Gopalan, K., Kalavadwala, R., & Shukla, S. (2018). A machine learning based crop recommendation system. International Journal of Computer Applications, 179(38), 15–18.
6. Gosai, D., & Macwan, S. (2022). Crop recommendation system using machine learning. International Journal of Engineering Research & Technology (IJERT), 11(06), 151–155.
7. Chhikara, P., &Chhillar, R. S. (2023). Machine learning-based smart crop recommender and yield predictor. Journal of Agronomy and Agricultural Science, 7(11), 45–50.
8. Pragya, P., Sharma, V., & Sharma, V. (2019). SENet CNN based tomato leaf disease detection. *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, 8(11), 773–777.
9. Devaraj, K. R., Jaahnavi, S., & Indira, K. (2019). Identification of plant disease using image processing technique. *Proceedings of the International Conference on Communication and Signal Processing (ICCSP)*, 749–753.
10. Khamparia, A., Pandey, B., & Gupta, D. (2019). A hybrid CNN-autoencoder model for leaf disease detection in plants. *Journal of Computational and Theoretical Nanoscience*, 16(7), 2881–2886.
11. Allen, R.G., Pereira, L.S., Raes, D., & Smith, M. (1998). Crop evapotranspiration—Guidelines for computing crop water requirements. *FAO Irrigation and Drainage Paper No. 56*. Food and Agriculture Organization of the United Nations.
12. Jeong, J.H., Resop, J.P., Mueller, N.D., et al. (2016). Random forests for global and regional crop yield predictions. *PLOS ONE*, 11(6), e0156571.
13. Maimaitijiang, M., et al. (2020). Soybean yield prediction from UAV using multimodal data fusion and deep learning. *Remote Sensing of Environment*, 237, 111599.
14. **Convolutional Neural Networks (CNNs) for Beginners**  
    DeepLearning.ai. (2025). Convolutional Neural Networks (CNNs) for Beginners.

**Appendix A (code for crop recommendation)**

import pandas as pd  
import pickle, import joblib  
from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.preprocessing import StandardScaler

# Load Dataset  
def load\_data():  
 csv\_path = "C:\\Users\\User\\Downloads\\major project\\Crop\_recommendation.csv"  
 data = pd.read\_csv(csv\_path)  
 return data  
  
crops = load\_data()

crops.info()

# Splitting Features and Target  
X = crops[['N', 'P', 'K', 'temperature', 'humidity', 'ph', 'rainfall']]  
y = crops['label'] # Target variable (Crop Name)  
  
# Scale features  
scaler = StandardScaler()  
X\_scaled = scaler.fit\_transform(X)

# Train Model  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)  
model = RandomForestClassifier(n\_estimators=100, random\_state=42)  
model.fit(X\_train, y\_train)

RandomForestClassifier(random\_state=42)  
  
*# Save the trained model*  
joblib.dump(model, 'crop\_model.pkl')  
  
*# Save the fitted scaler*  
joblib.dump(scaler, 'scaler.pkl')  
  
print("Model and Scaler saved successfully!")

**Appendix B (code for water requirement prediction)**

#importing libraries  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.model\_selection import train\_test\_split, GridSearchCV  
from sklearn.ensemble import RandomForestRegressor  
from sklearn.model\_selection import RandomizedSearchCV  
from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score  
from sklearn.preprocessing import LabelEncoder

df= pd.read\_csv('C:\\Users\\User\\Downloads\\major project\\clean\_water\_requirement\_data.csv')

X = df.drop(columns=['water\_requirement'])

y=df.iloc[:,-1]

numerical\_columns = df.select\_dtypes(include=['float64', 'int64']).columns  
numerical\_columns

Index(['temperature', 'humidity', 'wind\_speed', 'evapotranspiration',  
 'soil\_moisture\_levels', 'water\_retention\_capacity',  
 'crop\_water\_requirement', 'water\_requirement'],  
 dtype='object')

# Identify categorical columns  
categorical\_columns = df.select\_dtypes(include=['object', 'category']).columns  
  
# Count unique categories in each categorical column  
category\_counts = {col: df[col].nunique() for col in categorical\_columns}  
  
# Print the results  
for col, count in category\_counts.items():  
 print(f"{col}: {count} unique categories")

rainfall\_pattern: 3 unique categories  
soil\_type: 5 unique categories  
drainage\_properties: 3 unique categories  
crop\_type: 5 unique categories  
growth\_stage: 6 unique categories

print(df['soil\_type'].unique())   
print(df['rainfall\_pattern'].unique())  
print(df['drainage\_properties'].unique())  
print(df['crop\_type'].unique())  
print(df['growth\_stage'].unique())

['sandy' 'silty' 'clay' 'peaty' 'loamy']  
['moderate' 'low' 'high']  
['good' 'poor' 'moderate']  
['wheat' 'rice' 'soybean' 'cotton' 'maize']  
['vegetative' 'fruiting' 'flowering' 'seedling' 'maturity' 'reproductive']

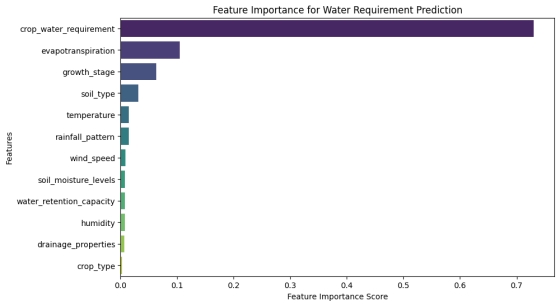
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

X\_train.shape

(16000, 12)

label\_encoders = {}  
for col in categorical\_columns:  
 le = LabelEncoder()  
 df[col] = le.fit\_transform(df[col]) # Convert categorical to numerical  
 label\_encoders[col] = le # Store encoders for inverse transformation

# Encode categorical features  
categorical\_columns = ['rainfall\_pattern', 'soil\_type', 'drainage\_properties', 'crop\_type', 'growth\_stage']  
df\_encoded = df.copy()  
label\_encoders = {}  
  
for col in categorical\_columns:  
 le = LabelEncoder()  
 df\_encoded[col] = le.fit\_transform(df[col])  
 label\_encoders[col] = le  
  
# Define features and target variable  
X = df\_encoded.drop(columns=['water\_requirement']) # Features  
y = df\_encoded['water\_requirement'] # Target  
  
# Split data into training and test sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
# Train RandomForestRegressor  
model = RandomForestRegressor(n\_estimators=100, random\_state=42)  
model.fit(X\_train, y\_train)  
  
# Predict and evaluate  
y\_pred = model.predict(X\_test)  
mae = mean\_absolute\_error(y\_test, y\_pred)  
r2 = r2\_score(y\_test, y\_pred)  
  
# Feature Importance Analysis  
feature\_importances = pd.Series(model.feature\_importances\_, index=X.columns).sort\_values(ascending=False)  
  
# Plot feature importances  
plt.figure(figsize=(10, 6))  
sns.barplot(x=feature\_importances.values, y=feature\_importances.index, palette="viridis")  
plt.xlabel("Feature Importance Score")  
plt.ylabel("Features")  
plt.title("Feature Importance for Water Requirement Prediction")  
plt.show()  
  
# Return evaluation metrics  
print("Mean Absolute Error",mae)  
print("Mean R Squared",r2)  
print("Feature Importance:")  
print(feature\_importances)  
  
 sns.barplot(x=feature\_importances.values, y=feature\_importances.index, palette="viridis")



Mean Absolute Error 0.8272287948093372  
Mean R Squared 0.9444194482428152  
Feature Importance:  
crop\_water\_requirement 0.730679  
evapotranspiration 0.104857  
growth\_stage 0.063204  
soil\_type 0.031183  
temperature 0.014904  
rainfall\_pattern 0.014499  
wind\_speed 0.008801  
soil\_moisture\_levels 0.007768  
water\_retention\_capacity 0.007322  
humidity 0.007169  
drainage\_properties 0.006933  
crop\_type 0.002680

correlation\_matrix = df.corr()  
print(correlation\_matrix['water\_requirement'])

temperature 0.082502  
humidity -0.016958  
wind\_speed 0.062858  
evapotranspiration 0.306207  
rainfall\_pattern 0.089451  
soil\_type 0.038873  
soil\_moisture\_levels 0.063450  
water\_retention\_capacity 0.063427  
drainage\_properties 0.038285  
crop\_type 0.046013  
crop\_water\_requirement 0.840243  
growth\_stage 0.095651  
water\_requirement 1.000000

# Define categorical mappings  
soil\_type\_mapping = {'sandy': 3, 'silty': 4, 'clay': 0, 'peaty': 2, 'loamy': 1}  
rainfall\_pattern\_mapping = {'moderate': 2, 'low': 1, 'high': 0}  
drainage\_properties\_mapping = {'good': 0, 'poor': 2, 'moderate': 1}  
crop\_type\_mapping = {'wheat': 4, 'rice': 2, 'soybean': 3, 'cotton': 0, 'maize': 1}  
growth\_stage\_mapping = {'vegetative': 5, 'fruiting': 1, 'flowering': 0, 'seedling': 4, 'maturity': 2, 'reproductive': 3}  
  
# Reverse mappings for display  
reverse\_mappings = {  
 'growth\_stage': {v: k for k, v in growth\_stage\_mapping.items()},  
 'soil\_type': {v: k for k, v in soil\_type\_mapping.items()},  
 'rainfall\_pattern': {v: k for k, v in rainfall\_pattern\_mapping.items()},  
 'drainage\_properties': {v: k for k, v in drainage\_properties\_mapping.items()},  
 'crop\_type': {v: k for k, v in crop\_type\_mapping.items()},  
}  
  
# Select top 5 features based on importance  
top\_5\_features = feature\_importances.index[:5].tolist()  
  
# Store user input  
user\_input = {}  
  
print("Please enter values for the following features:")  
  
# Get input for the top 5 features  
for feature in top\_5\_features:  
 if feature in reverse\_mappings: # If it's categorical, show category names  
 print(f"\nAvailable categories for {feature}:")  
 for num, cat in reverse\_mappings[feature].items():  
 print(f"{num}: {cat}")  
  
 user\_value = input(f"Enter the number corresponding to {feature}: ").strip()  
 if user\_value.isdigit() and int(user\_value) in reverse\_mappings[feature]:  
 user\_input[feature] = int(user\_value)  
 else:  
 print(f"Invalid choice! Defaulting to first category: {reverse\_mappings[feature][0]}")  
 user\_input[feature] = 0 # Default to first category  
 else: # Numerical input  
 user\_input[feature] = float(input(f"Enter value for {feature}: "))  
  
# Ask user if they want to add extra features  
extra\_features = []  
print("\nDo you want to add more features? (yes/no)")  
choice = input().strip().lower()  
  
if choice == "yes":  
 remaining\_features = [f for f in feature\_importances.index if f not in top\_5\_features]  
 print("\nAvailable additional features:")  
 for i, feature in enumerate(remaining\_features, 1):  
 print(f"{i}. {feature}")  
   
 selected\_indices = input("Enter the numbers of features to add (comma-separated): ")  
 selected\_indices = [int(i.strip()) - 1 for i in selected\_indices.split(",")]  
  
 for i in selected\_indices:  
 feature = remaining\_features[i]  
 if feature in reverse\_mappings: # If categorical, show category names  
 print(f"\nAvailable categories for {feature}:")  
 for num, cat in reverse\_mappings[feature].items():  
 print(f"{num}: {cat}")  
  
 user\_value = input(f"Enter the number corresponding to {feature}: ").strip()  
 if user\_value.isdigit() and int(user\_value) in reverse\_mappings[feature]:  
 user\_input[feature] = int(user\_value)  
 else:  
 print(f"Invalid choice! Defaulting to first category: {reverse\_mappings[feature][0]}")  
 user\_input[feature] = 0 # Default to first category  
 else: # If numerical  
 user\_input[feature] = float(input(f"Enter value for {feature}: "))  
  
# Ask user for the land area  
land\_area = float(input("\nEnter the area of the land in square meters (m²): "))  
  
# Prepare input for model  
final\_features = top\_5\_features + extra\_features  
input\_data = np.array([user\_input[feature] for feature in final\_features]).reshape(1, -1)  
  
# Ensure all required features are present (fill missing ones with mean)  
model\_features = X.columns  
final\_input = np.zeros((1, len(model\_features)))  
for i, feature in enumerate(model\_features):  
 if feature in user\_input:  
 final\_input[0, i] = user\_input[feature]  
 else:  
 final\_input[0, i] = X\_train[feature].mean() # Fill missing with mean  
  
# Make prediction  
prediction = model.predict(final\_input)  
print(f"\n✅ Predicted Water Requirement: {prediction[0]:.2f} mm/day")  
  
# Convert the water requirement to liters based on the land area  
total\_water\_needed = prediction[0] \* land\_area  
print(f"✅ Total Water Needed: {total\_water\_needed:.2f} liters/day")

**Appendix C (code for plant disease prediction)**

import numpy as np  
import tensorflow as tf  
import os  
from tensorflow.keras import layers  
from tensorflow.keras.preprocessing.image import load\_img,ImageDataGenerator  
from tensorflow.keras.models import Sequential,load\_model  
from tensorflow.keras.layers import Conv2D,MaxPool2D,Dense,Dropout,Flatten  
from tensorflow.keras.layers import MaxPooling2D  
import matplotlib.pyplot as plt  
from tensorflow import keras

import gdown  
  
file\_id = "1kXAc3BcHtWAARqW6gRYXEhKQrKfusneA" # Extracted from the link  
file\_name = "plant\_disease\_merged.zip" # Saving with the correct name  
  
gdown.download(f"https://drive.google.com/uc?id={file\_id}", file\_name, quiet=False)

Downloading...  
From (original): https://drive.google.com/uc?id=1kXAc3BcHtWAARqW6gRYXEhKQrKfusneA  
From (redirected): https://drive.google.com/uc?id=1kXAc3BcHtWAARqW6gRYXEhKQrKfusneA&confirm=t&uuid=fbbdaa7e-3370-4e11-b75c-355194a15aaa  
To: /content/plant\_disease\_merged.zip  
100%|██████████| 2.19G/2.19G

{"type":"string"}

#zip file extraction  
  
import zipfile  
import os  
  
with zipfile.ZipFile(file\_name, 'r') as zip\_ref:  
 zip\_ref.extractall("/content/extracted1") # Extract to a folder

dataset\_path ='/content/extracted1/plant\_disease\_merged'  
batch = 32  
img\_size = 256  
base\_dir = dataset\_path

# generator is a function that yields batches of data instead of loading everything into memory at once.  
#It is commonly used for handling large datasets efficiently.  
  
# Load Training Dataset (80%)  
train\_ds = tf.keras.utils.image\_dataset\_from\_directory(  
 base\_dir,  
 seed=123, # Ensures reproducibility in data splitting  
 validation\_split=0.2, # 20% of the dataset is reserved for validation  
 subset="training", # Load only the training dataset (80%)  
 batch\_size=batch, # Number of images per batch  
 image\_size=(img\_size, img\_size) # Resize images to (img\_size, img\_size)  
)  
  
# Load Validation Dataset (20%)  
validation\_ds = tf.keras.utils.image\_dataset\_from\_directory(  
 base\_dir,  
 seed=123, # Ensures reproducibility in data splitting  
 validation\_split=0.2, # 20% of the dataset is reserved for validation  
 subset="validation", # Load only the validation dataset  
 batch\_size=batch, # Number of images per batch  
 image\_size=(img\_size, img\_size) # Resize images to (img\_size, img\_size)  
)

Found 162916 files belonging to 38 classes.  
Using 130333 files for training.  
Found 162916 files belonging to 38 classes.  
Using 32583 files for validation.

def normalize(img, label):  
 img = tf.cast(img, tf.float32) / 255.0 # Convert to float32 and scale  
 return img, label  
  
# Apply normalization to both datasets  
train\_ds = train\_ds.map(normalize)  
validation\_ds = validation\_ds.map(normalize)

# Create CNN model  
model=Sequential()  
model.add(Conv2D(32,kernel\_size=(3,3),padding='valid',activation='relu',input\_shape=(256,256,3)))  
model.add(MaxPooling2D(pool\_size=(2,2),strides=2,padding='valid'))  
  
model.add(Conv2D(32,kernel\_size=(3,3),padding='valid',activation='relu')) # here 32 filters are used wit each size 3\*3  
model.add(MaxPooling2D(pool\_size=(2,2),strides=2,padding='valid'))  
  
model.add(Conv2D(32,kernel\_size=(3,3),padding='valid',activation='relu'))  
model.add(MaxPooling2D(pool\_size=(2,2),strides=2,padding='valid'))  
  
model.add(Flatten())  
  
model.add(Dropout(0.2))  
model.add(Dense(512, activation='relu'))  
model.add(Dense(256, activation='relu'))  
model.add(Dense(38, activation='softmax')) # 38 classes → Softmax activation

/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base\_conv.py:107: UserWarning: Do not pass an `input\_shape`/`input\_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.  
 super().\_\_init\_\_(activity\_regularizer=activity\_regularizer, \*\*kwargs)

model.summary()

Model: "sequential"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓  
┃ Layer (type) ┃ Output Shape ┃ Param # ┃  
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩  
│ conv2d (Conv2D) │ (None, 254, 254, 32) │ 896 │  
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤  
│ max\_pooling2d (MaxPooling2D) │ (None, 127, 127, 32) │ 0 │  
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤  
│ conv2d\_1 (Conv2D) │ (None, 125, 125, 32) │ 9,248 │  
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤  
│ max\_pooling2d\_1 (MaxPooling2D) │ (None, 62, 62, 32) │ 0 │  
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤  
│ conv2d\_2 (Conv2D) │ (None, 60, 60, 32) │ 9,248 │  
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤  
│ max\_pooling2d\_2 (MaxPooling2D) │ (None, 30, 30, 32) │ 0 │  
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤  
│ flatten (Flatten) │ (None, 28800) │ 0 │  
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤  
│ dropout (Dropout) │ (None, 28800) │ 0 │  
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤  
│ dense (Dense) │ (None, 512) │ 14,746,112 │  
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤  
│ dense\_1 (Dense) │ (None, 256) │ 131,328 │  
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤  
│ dense\_2 (Dense) │ (None, 38) │ 9,766 │  
└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘

Total params: 14,906,598 (56.86 MB)

Trainable params: 14,906,598 (56.86 MB)

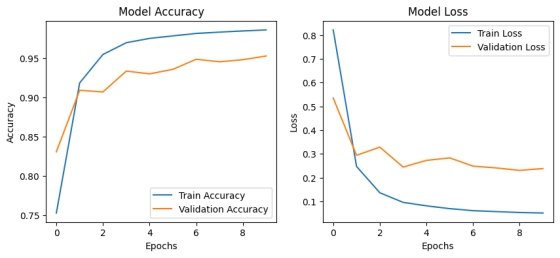
Non-trainable params: 0 (0.00 B)

model.compile(optimizer='adam',loss='sparse\_categorical\_crossentropy',metrics=['accuracy'])

history=model.fit(train\_ds,epochs=10,validation\_data=validation\_ds)

Epoch 1/10  
4073/4073 ━━━━━━━━━━━━━━━━━━━━ 203s 48ms/step - accuracy: 0.6096 - loss: 1.3718 - val\_accuracy: 0.8304 - val\_loss: 0.5353  
Epoch 2/10  
4073/4073 ━━━━━━━━━━━━━━━━━━━━ 189s 46ms/step - accuracy: 0.9016 - loss: 0.3007 - val\_accuracy: 0.9089 - val\_loss: 0.2940  
Epoch 3/10  
4073/4073 ━━━━━━━━━━━━━━━━━━━━ 201s 46ms/step - accuracy: 0.9488 - loss: 0.1523 - val\_accuracy: 0.9068 - val\_loss: 0.3285  
Epoch 4/10  
4073/4073 ━━━━━━━━━━━━━━━━━━━━ 187s 46ms/step - accuracy: 0.9676 - loss: 0.1012 - val\_accuracy: 0.9334 - val\_loss: 0.2446  
Epoch 5/10  
4073/4073 ━━━━━━━━━━━━━━━━━━━━ 187s 46ms/step - accuracy: 0.9746 - loss: 0.0834 - val\_accuracy: 0.9299 - val\_loss: 0.2728  
Epoch 6/10  
4073/4073 ━━━━━━━━━━━━━━━━━━━━ 201s 46ms/step - accuracy: 0.9771 - loss: 0.0734 - val\_accuracy: 0.9356 - val\_loss: 0.2832  
Epoch 7/10  
4073/4073 ━━━━━━━━━━━━━━━━━━━━ 186s 46ms/step - accuracy: 0.9803 - loss: 0.0648 - val\_accuracy: 0.9485 - val\_loss: 0.2484  
Epoch 8/10  
4073/4073 ━━━━━━━━━━━━━━━━━━━━ 201s 45ms/step - accuracy: 0.9822 - loss: 0.0608 - val\_accuracy: 0.9452 - val\_loss: 0.2410  
Epoch 9/10  
4073/4073 ━━━━━━━━━━━━━━━━━━━━ 202s 45ms/step - accuracy: 0.9850 - loss: 0.0516 - val\_accuracy: 0.9479 - val\_loss: 0.2306  
Epoch 10/10  
4073/4073 ━━━━━━━━━━━━━━━━━━━━ 185s 46ms/step - accuracy: 0.9859 - loss: 0.0505 - val\_accuracy: 0.9526 - val\_loss: 0.2384

import matplotlib.pyplot as plt  
  
# Extract loss and accuracy  
train\_loss = history.history['loss']  
val\_loss = history.history['val\_loss']  
train\_acc = history.history['accuracy']  
val\_acc = history.history['val\_accuracy']  
  
# Plot Accuracy  
plt.figure(figsize=(10, 4))  
plt.subplot(1, 2, 1)  
plt.plot(train\_acc, label='Train Accuracy')  
plt.plot(val\_acc, label='Validation Accuracy')  
plt.xlabel("Epochs")  
plt.ylabel("Accuracy")  
plt.legend()  
plt.title("Model Accuracy")  
  
# Plot Loss  
plt.subplot(1, 2, 2)  
plt.plot(train\_loss, label='Train Loss')  
plt.plot(val\_loss, label='Validation Loss')  
plt.xlabel("Epochs")  
plt.ylabel("Loss")  
plt.legend()  
plt.title("Model Loss")  
  
plt.show()



val\_loss, val\_acc = model.evaluate(validation\_ds)  
print(f"Validation Loss: {val\_loss:.4f}, Validation Accuracy: {val\_acc:.4f}")

1019/1019 ━━━━━━━━━━━━━━━━━━━━ 21s 20ms/step - accuracy: 0.9524 - loss: 0.2373  
Validation Loss: 0.2384, Validation Accuracy: 0.9526

# folder name set with folder number  
import os  
  
data\_dir ='/content/extracted1/plant\_disease\_merged'  
class\_names = sorted(os.listdir(data\_dir)) # Ensure a consistent order  
  
# Create a mapping  
class\_to\_index = {cls\_name: i for i, cls\_name in enumerate(class\_names)}  
index\_to\_class = {i: cls\_name for cls\_name, i in class\_to\_index.items()}  
index\_to\_class

{0: 'Apple\_\_\_Apple\_scab',  
 1: 'Apple\_\_\_Black\_rot',  
 2: 'Apple\_\_\_Cedar\_apple\_rust',  
 3: 'Apple\_\_\_healthy',  
 4: 'Blueberry\_\_\_healthy',  
 5: 'Cherry\_(including\_sour)\_\_\_Powdery\_mildew',  
 6: 'Cherry\_(including\_sour)\_\_\_healthy',  
 7: 'Corn\_(maize)\_\_\_Cercospora\_leaf\_spot Gray\_leaf\_spot',  
 8: 'Corn\_(maize)\_\_\_Common\_rust\_',  
 9: 'Corn\_(maize)\_\_\_Northern\_Leaf\_Blight',  
 10: 'Corn\_(maize)\_\_\_healthy',  
 11: 'Grape\_\_\_Black\_rot',  
 12: 'Grape\_\_\_Esca\_(Black\_Measles)',  
 13: 'Grape\_\_\_Leaf\_blight\_(Isariopsis\_Leaf\_Spot)',  
 14: 'Grape\_\_\_healthy',  
 15: 'Orange\_\_\_Haunglongbing\_(Citrus\_greening)',  
 16: 'Peach\_\_\_Bacterial\_spot',  
 17: 'Peach\_\_\_healthy',  
 18: 'Pepper,\_bell\_\_\_Bacterial\_spot',  
 19: 'Pepper,\_bell\_\_\_healthy',  
 20: 'Potato\_\_\_Early\_blight',  
 21: 'Potato\_\_\_Late\_blight',  
 22: 'Potato\_\_\_healthy',  
 23: 'Raspberry\_\_\_healthy',  
 24: 'Soybean\_\_\_healthy',  
 25: 'Squash\_\_\_Powdery\_mildew',  
 26: 'Strawberry\_\_\_Leaf\_scorch',  
 27: 'Strawberry\_\_\_healthy',  
 28: 'Tomato\_\_\_Bacterial\_spot',  
 29: 'Tomato\_\_\_Early\_blight',  
 30: 'Tomato\_\_\_Late\_blight',  
 31: 'Tomato\_\_\_Leaf\_Mold',  
 32: 'Tomato\_\_\_Septoria\_leaf\_spot',  
 33: 'Tomato\_\_\_Spider\_mites Two-spotted\_spider\_mite',  
 34: 'Tomato\_\_\_Target\_Spot',  
 35: 'Tomato\_\_\_Tomato\_Yellow\_Leaf\_Curl\_Virus',  
 36: 'Tomato\_\_\_Tomato\_mosaic\_virus',  
 37: 'Tomato\_\_\_healthy'}

from google.colab import files  
  
# Upload image  
uploaded = files.upload()  
  
# Get the uploaded file name (path)  
img\_path = list(uploaded.keys())[0]  
  
# Print the path  
print(f"Image saved at: {img\_path}")

<IPython.core.display.HTML object>

Saving RS\_Rust 1563.JPG to RS\_Rust 1563 (1).JPG  
Image saved at: RS\_Rust 1563 (1).JPG

#image pre-processing  
from tensorflow.keras.preprocessing import image  
  
img = image.load\_img(img\_path, target\_size=(256, 256)) # Resize to model input size  
img\_array = image.img\_to\_array(img) # Convert to NumPy array  
img\_array = np.expand\_dims(img\_array, axis=0) # Add batch dimension  
img\_array = img\_array / 255.0 # Normalize (if required)  
  
#prediction = model.predict(img\_array)  
predicted\_index = model.predict(img\_array).argmax() # Get the index of the highest probability  
predicted\_class = index\_to\_class[predicted\_index]  
  
print(f"Predicted Class: {predicted\_class}")

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step  
Predicted Class: Corn\_(maize)\_\_\_Common\_rust\_

model.save("dp001.1.h5") # Saves in HDF5 format  
  
# Saves in your Google Drive  
# model.save("/content/drive/MyDrive/dp001.1.h5")

#model.save("full\_merged\_data.keras") # Saves in Keras format  
model.save("/content/drive/MyDrive/full\_merged\_data.keras")

**Appendix D (Backend code)**

from flask import Flask, render\_template, request, redirect, url\_for

import os

from werkzeug.utils import secure\_filename

from disease\_prediction import pre\_process

#from disease\_prediction.config import UPLOAD\_FOLDER, allowed\_file

from crop\_prediction import crop\_pre\_process

from water\_requirement import water\_requirement\_pre\_process

from flask import Flask, render\_template, request, redirect, url\_for, send\_from\_directory

app = Flask(\_\_name\_\_)

# website home page

@app.route("/", methods=['GET'])

def home():

return render\_template('index.html')

# crop prediction page

@app.route("/crop", methods=['GET','POST'])

def crop\_model():

if request.method=="GET":

return render\_template('crop\_prediction.html')

else:

N = float(request.form['N'])

P = float(request.form['P'])

K = float(request.form['K'])

temperature = float(request.form['temperature'])

humidity = float(request.form['humidity'])

ph = float(request.form['ph'])

rainfall =float(request.form['rainfall'])

result=crop\_pre\_process.crop\_model(N,P,K,temperature,humidity,ph,rainfall)

print(result)

return render\_template('crop\_prediction.html',result1=result)

#disease prediction page

UPLOAD\_FOLDER = os.path.join(os.path.dirname(\_\_file\_\_), 'static', 'uploads')

os.makedirs(UPLOAD\_FOLDER, exist\_ok=True)

# Configure Flask to use the upload folder

app.config['UPLOAD\_FOLDER'] = UPLOAD\_FOLDER

@app.route("/disease", methods=["GET", "POST"])

def disease\_model():

if request.method == 'POST':

if 'file' not in request.files:

return render\_template('disease\_prediction.html', error='No file part')

file = request.files['file']

if file.filename == '':

return render\_template('disease\_prediction.html', error='No selected file')

if file:

filename = secure\_filename(file.filename)

filepath = os.path.join(app.config['UPLOAD\_FOLDER'], filename)

file.save(filepath)

# Build path for displaying the image in HTML

image\_path = url\_for('static', filename='uploads/' + filename)result = pre\_process.model\_train(filepath)

print(result)

return render\_template('disease\_prediction.html', result=result, image\_path=image\_path)

return render\_template('disease\_prediction.html')

# water management page

@app.route("/water", methods=['GET','POST'])

def water\_model():

if request.method=="GET":

return render\_template('water\_management.html')

else:

# Required fields

land\_area = float(request.form['land\_area'])

crop\_water\_requirement = request.form['crop\_water\_requirement']

evapotranspiration = request.form['evapotranspiration']

growth\_stage = request.form['growth\_stage']

soil\_type = request.form['soil\_type']

temperature = request.form['temperature']

# Optional fields (use .get)

rainfall\_pattern = request.form.get('rainfall\_pattern')

wind\_speed = request.form.get('wind\_speed')

soil\_moisture\_levels = request.form.get('soil\_moisture\_levels')

water\_retention\_capacity = request.form.get('water\_retention\_capacity')

humidity = request.form.get('humidity')

drainage\_properties = request.form.get('drainage\_properties')

crop\_type = request.form.get('crop\_type')

# calling the function and passing the input parameters

result = water\_requirement\_pre\_process.transform(land\_area,

temperature, humidity, wind\_speed, evapotranspiration,

rainfall\_pattern, soil\_type, soil\_moisture\_levels,

water\_retention\_capacity, drainage\_properties,

crop\_type, crop\_water\_requirement, growth\_stage)

print(result)

return render\_template('water\_management.html',prediction=result)

if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=True, port=5000)