VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belagavi – 590018

PROJECT REPORT

on

"AI-Driven Crop Disease Prediction and Management System"

Submitted in partial fulfilment for the award of the degree

Bachelor of Engineering

in

Computer Science and Engineering

Submitted by

Shashishekhara K	**1ST21CS192**
Shivashankar M G	**1ST21CS196**
Vinay G V	**1ST21CS237**
Praveen B M	**1ST22CS413**

Under the Guidance of

Prof. Deepa. S Bhat
Assistant Professor
Department of CSE
SaIT, Bengaluru

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SAMBHRAM INSTITUTE OF TECHNOLOGY

M. S. Palya, Bengaluru – 560097

2024-2025

SAMBHRAM INSTITUTE OF TECHNOLOGY

M. S. Palya, Bengaluru – 560097

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

Certified that the Project work entitled *"***AI-Driven Crop Disease Prediction and Management System***"* carried out by **Mr. Shashishekhara K ,**1ST21CS192**, Mr. Shivashankar M G,**1ST21CS196**, Mr. Vinay G V,**1ST21CS237**, Mr. Praveen B M,**1ST22CS413, bonafide students of **SAMBHRAM INSTITUTE OF TECHNOLOGY** in partial fulfilment for the award of **BACHELOR OF ENGINEERING IN COMPUTER SCIENCE AND ENGINEERING** of **VISVESVARAYA TECHNOLOGICAL UNIVERSITY**, Belagavi during the year **2024-2025**. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the Report deposited in the departmental library. The Project report has been approved as it satisfies the academic requirements in respect of Project work prescribed for the said Degree.

Prof. Deepa. S Bhat Assistant Professor Dept. of CSE SaIT, Bengaluru	**Dr. T. John Peter** HOD Dept. of CSE SaIT, Bengaluru	**Dr. H. G. Chandrakanth** Principal SaIT, Bengaluru

EXTERNAL VIVA:

Name of the Examiners Signature with date

1.

2.

SAMBHRAM INSTITUTE OF TECHNOLOGY

M. S. Palya, Bengaluru – 560097

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

We the students of 7th Semester, Dept. of CSE, SaIT doing the Final Year Project, declare that

[1] The Hardware/Software is not purchased/brought from any outside originations.

[2] The Hardware/Software is not from any other previous final year engineering projects of VTU.

[3] Our Project work is as per VTU norms, and we have followed the rules and regulations.

Violating any of the above conditions, we will accept the action taken by the Department/College/ VTU in this regard.

The Title of the Project is

" **AI-Driven Crop Disease Prediction and Management System** "

The project work is guided by:

Prof. Deepa.S Bhat
Asst. Professor
Dept. of CSE, SaIT.

No.	Student Name	USN	Signature
1	**Shashishekhara K**	**1ST21CS192**	_____
2	**Shivashankar M G**	**1ST21CS196**	_____
3	**Vinay G V**	**1ST21CS237**	_____
4	**Praveen B M**	**1ST22CS413**	_____

ABSTRACT

Farmers in many parts of India are having trouble growing crops because of the climate and soil. There could be no genuine assistant available to assist them with encouraging the right sorts of plants using current advancement. Due to illiteracy, farmers may not be able to benefit from advances in agricultural science and continue using human methods. This makes it difficult to achieve the desired yield. For instance, improper fertilization or unintentional rainfall patterns may be the cause of crop failure. In such circumstances, picking crops that are suitable for the soil's current conditions and the anticipated rainfall during planting would be the best course of action. Thus, we are presenting an information mining-based "Soil-Based Profile Profiling Framework." In light of the rancher's region's precipitation and soil input boundaries (NPK and pH), we provide a list of possible yields. Additionally, it suggests fertilizer that can be utilized to increase crop yields and enhance the soil's quality. The growing problem of crop failure is the focus of this desktop application.

ACKNOWLEDGEMENT

Any achievement, be it scholastic or otherwise does not depend solely on the individual efforts but on the guidance, encouragement and cooperation of intellectuals, elders, and friends. A number of personalities, in their own capacities have helped us in carrying out this project work. We would like to take this opportunity to thank them all.

We would like to express our heartfelt thanks to **Dr. H. G. Chandrakanth, Principal, Sambhram Institute of Technology**, whose valuable guidance has been the one that helped us to complete the project.

We would like to express our profound gratitude to **Dr. T. John Peter, HOD, Department of CSE, Sambhram Institute of Technology**, for his suggestions and his instructions have served as the major contribution towards the completion of the project.

We would like to extend our impassioned thanks and admiration to our guide, **Prof. Deepa. S Bhat, Assistant Professor, Department of CSE, Sambhram Institute of Technology**, for her able guidance, regular source of encouragement and assistance throughout this project.

We would like to thank all the teaching and non-teaching staff members of the Computer Science Department, who have helped us directly or indirectly for the successful completion of the project.

Finally, we would like to thank our Parents and Friends who have helped us with their valuable suggestions and guidance for the completion of our project.

Shashishekhara K	**1ST21CS192**
Shivashankar M G	**1ST21CS196**
Vinay G V	**1ST21CS237**
Praveen B M	**1ST22CS413**

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Project Description

In India, 118.6 million farmers rely on agriculture for their livelihood, according to the 2011 census. Understanding the soil conditions, when and where to apply compost, taking into account rainfall, maintaining crop quality, and understanding how various factors operate differently in different parts of the same field are some of the numerous issues that farmers have had to deal with in the past. like while furrowing While settling on significant horticultural choices that might be challenging to carry out all alone or on occasion, various variables and measurements should be considered. This program will provide a solution for agriculture that can assist farmers in increasing their overall productivity by monitoring the agricultural field. Rainfall reserves and soil boundaries, two examples of online weather data, can assist in determining which plants should be planted in a given location. This function introduces a desktop application that predicts the most profitable yields in the current climate and soil conditions using data analysis techniques.

The program will integrate environment and capacity office information. The most significant plants were anticipated in light of the ongoing normal circumstances utilizing an AI calculation. As a result, the farmer can cultivate a wide range of crops. Along these lines, the task encourages a system by solidifying data from various sources, data assessment, and deciding examination, getting to the powerful yield creation and growing the net incomes of farmers, helping them long term. For a fruitful gather, the rancher should deal with the dirt. In order to maximize a harvest's yield and choose the best compost, growers should be aware of the soil's macro and micronutrient content. One significant part of development is soil investigation. Most of individuals come up short on information important to appropriately and decisively plant crops. By analyzing parameters like Sodium (N), Potassium (K), and Phosphorus (P), as well as the pH value of the soil, the region, and the amount of rainfall, our project thus determines which soil- based plants are suitable. mentioned earlier.

Furthermore, AI-Driven Crop Disease Prediction and Management System goes beyond crop selection to offer precise fertilizer recommendations tailored to the needs of each farm. By considering soil nutrient levels, crop requirements, and environmental factors, the platform provides farmers with optimized fertilizer prescriptions, thereby enhancing nutrient management practices and reducing input costs. By utilizing fertilizers more efficiently, farmers can maximize crop yields while minimizing environmental impact, contributing to long-term sustainability in agriculture.

In addition to crop and fertilizer recommendations, AI-Driven Crop Disease Prediction and Management System incorporates state-of-the-art image recognition technology to facilitate early detection and diagnosis of plant diseases. Through a simple interface, farmers can upload images of diseased plants, allowing AI-powered system

disease management enables farmers to take timely remedial actions, preventing widespread crop damage and minimizing yield losses.

In conclusion, AI-Driven Crop Disease Prediction and Management System stands as a testament to the transformative potential of technology in agriculture. By harnessing the power of data analytics, artificial intelligence, and image recognition, AI-Driven Crop Disease Prediction and Management System empowers farmers to make informed decisions, optimize resource utilization, and enhance sustainability in farming practices.

CHAPTER 2

LITERATURE SURVEY

In recent years, the integration of technology in agriculture has gained considerable attention due to its potential to address challenges faced by farmers and enhance agricultural productivity. The development of platforms like AI-Driven Crop Disease Prediction and Management System, which leverage data analytics, image recognition, and geographical information systems (GIS), represents a significant step towards precision agriculture and sustainable farming practices.

Research on crop recommendation systems has highlighted the importance of considering environmental factors such as rainfall patterns, temperature, and soil characteristics in crop selection. Studies by Li et al. (2018) and Zhang et al. (2019) demonstrate the effectiveness of data-driven approaches in predicting crop suitability based on climatic conditions and soil properties. AI-Driven Crop Disease Prediction and Management System builds upon this research by providing farmers with real-time recommendations tailored to their specific location and environmental conditions, thereby enabling them to make informed decisions regarding crop selection.

Furthermore, fertilizer recommendation systems have emerged as essential tools for optimizing nutrient management and minimizing environmental impact in agriculture. Research by Gao et al. (2020) and Sahoo et al. (2021) emphasizes the importance of precision fertilizer application based on soil nutrient levels and crop requirements. AI-Driven Crop Disease Prediction and Management System extends this research by integrating fertilizer recommendation algorithms with geospatial data to provide customized fertilizer prescriptions, thereby enhancing resource efficiency and reducing fertilizer wastage.

The incorporation of image recognition technology for disease detection in plants represents another significant advancement in agricultural research. Studies by Mohanty et al. (2016) and Liu et al. (2020) demonstrate the potential of deep learning techniques for accurate and rapid identification of plant diseases from images. AI-Driven Crop Disease Prediction and Management System leverages these advancements by enabling farmers to diagnose plant diseases simply by uploading images, facilitating early detection and prompt management of diseases, thus minimizing crop losses.

Overall, the literature review underscores the importance of integrating technology into agriculture to address challenges related to crop selection, nutrient management, and disease detection. AI-Driven Crop Disease Prediction and Management System represents a promising solution that combines various technological advancements to empower farmers with actionable insights and recommendations, ultimately contributing to sustainable agriculture and food security.

CHAPTER 3

OBJECTIVES

The AI-driven platform aims to revolutionize agriculture by addressing the core challenges faced by farmers, improving efficiency, and promoting sustainability through innovative, data-driven solutions. Below are the detailed objectives of the platform:

AI-Driven Crop Disease Prediction and Management System follows a systematic approach. Let's investigate each stage:

1. Crop Recommendation:

The platform aims to analyze essential soil parameters such as nitrogen (N), phosphorus (P), potassium (K), soil pH, and moisture levels using advanced AI models capable of accurately interpreting scientific data. By integrating external factors like real-time weather conditions, temperature variations, and historical data, the system enhances prediction accuracy, ensuring well-informed agricultural decisions. These combined insights enable the platform to recommend crops best suited to specific soil and environmental conditions, minimizing trial-and-error approaches in farming. This not only saves farmers valuable time and resources but also helps optimize yields and promote sustainable agricultural practices.

2. Fertilizer Recommendation:

The platform is designed to develop advanced algorithms that accurately evaluate soil nutrient deficiencies and provide customized fertilizer recommendations tailored to specific crops and soil types. By incorporating weather patterns and farming history, it optimizes fertilizer application timing and dosage, ensuring maximum effectiveness and efficiency. Additionally, the system minimizes the environmental impact of over-fertilization, such as groundwater contamination, by promoting judicious fertilizer use aligned with sustainable farming principles. This approach helps farmers adopt cost-effective, environmentally sound practices that support long-term soil fertility and contribute to sustainable agricultural development.

3. Plant Disease Detection:

The platform leverages advanced image processing techniques and machine learning models, including convolutional neural networks (CNNs), to accurately identify plant diseases from images of affected leaves. It incorporates a comprehensive database of plant diseases, offering detailed descriptions of symptoms, possible causes, and actionable treatment recommendations to guide farmers effectively. By enabling early detection and providing precise intervention strategies, the platform significantly reduces crop loss, enhances overall productivity, and increases farmers' resilience to plant health challenges, ensuring sustainable and efficient agricultural practices.

4. Ease of Access and Usability:

The platform is designed with an intuitive and easy-to-navigate interface, ensuring accessibility for farmers with varying levels of technical expertise. It allows users to seamlessly input soil data or upload high-resolution images

of diseased plants for quick and accurate analysis. Actionable insights are provided in simple, farmer-friendly language, making the platform usable and effective across diverse agricultural communities. This user-centric design ensures that farmers can readily adopt and benefit from the platform's advanced AI-driven features.

5. Empowering Farmers and Bridging the Technology Gap:

The platform aims to empower farmers by providing advanced tools that enhance decision-making, boost productivity, and improve profitability through personalized, real-time recommendations. By bridging the gap between traditional agricultural practices and modern innovations, it makes AI technology accessible and beneficial, even for small-scale and marginal farmers. Additionally, the platform facilitates the adoption of precision agriculture, equipping farmers to adapt effectively to challenges such as climate change, shifting market demands, and the evolving agricultural landscape, thereby fostering resilience and sustainability in farming practices.

CHAPTER 4

SYSTEM DESIGN

4.1 System Architecture

The process begins with the user uploading an image of a plant leaf, which is then pre-processed through resizing, noise removal, and contrast adjustments. The refined image is segmented to isolate the leaf or specific areas, focusing on diseased spots or patterns. Features such as texture, color, and shape are extracted from the segmented image to identify disease-related patterns. Machine learning or deep learning models classify the leaf as either Healthy or Diseased. If diseased, the system provides treatment suggestions to help address the issue effectively.

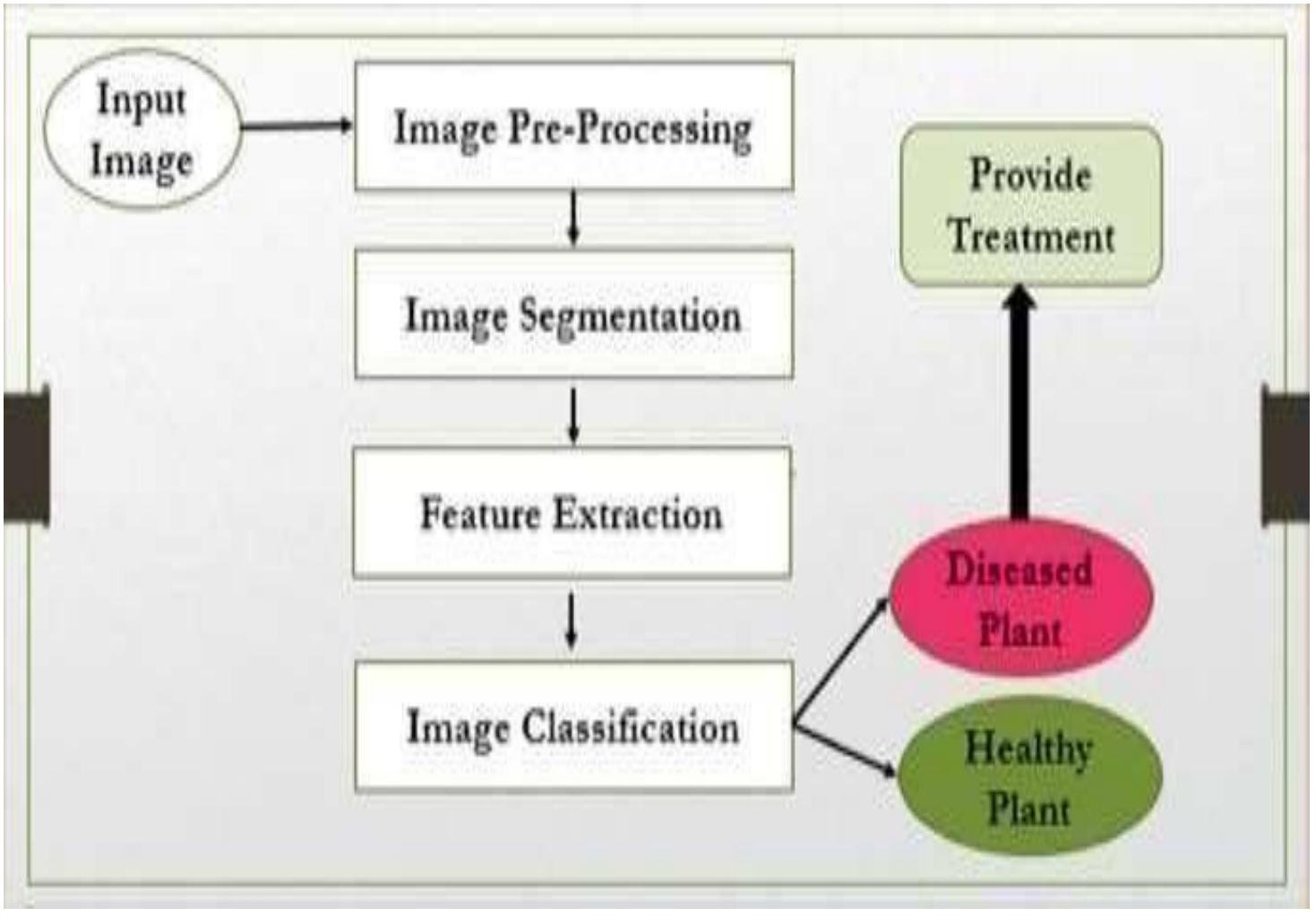


Fig. 4.1 System Architecture

4.1.1 Flow diagram for Crop Recommendation system

On entering the values of Nitrogen, phosphorus, and Potassium a post request is made to the flask API. After the model runs an HTTP response is sent to the front-end which tells the best crop a farmer can grow in the soil in order to get the best out of the land.

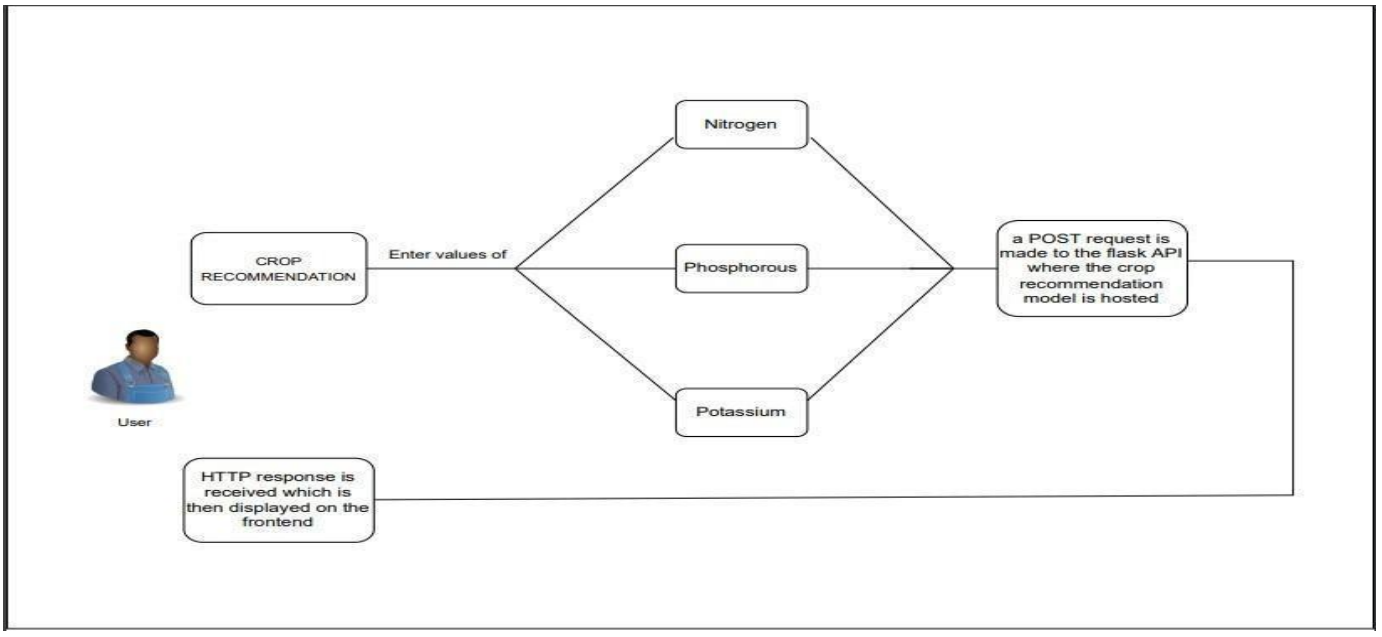


Figure 4.1.1 Flow diagram for Crop Recommendation system

4.1.2 Flow diagram for Fertilizer Recommendation system

The user has to enter the Nitrogen, Phosphorus, Potassium values along with the crop Name. A POST request is made to the flask API. Over here the fertilizer recommendation classifier is hosted. An HTTP response is sent to the front-end and in turn on the front-end the user gets recommendation to fertilizer

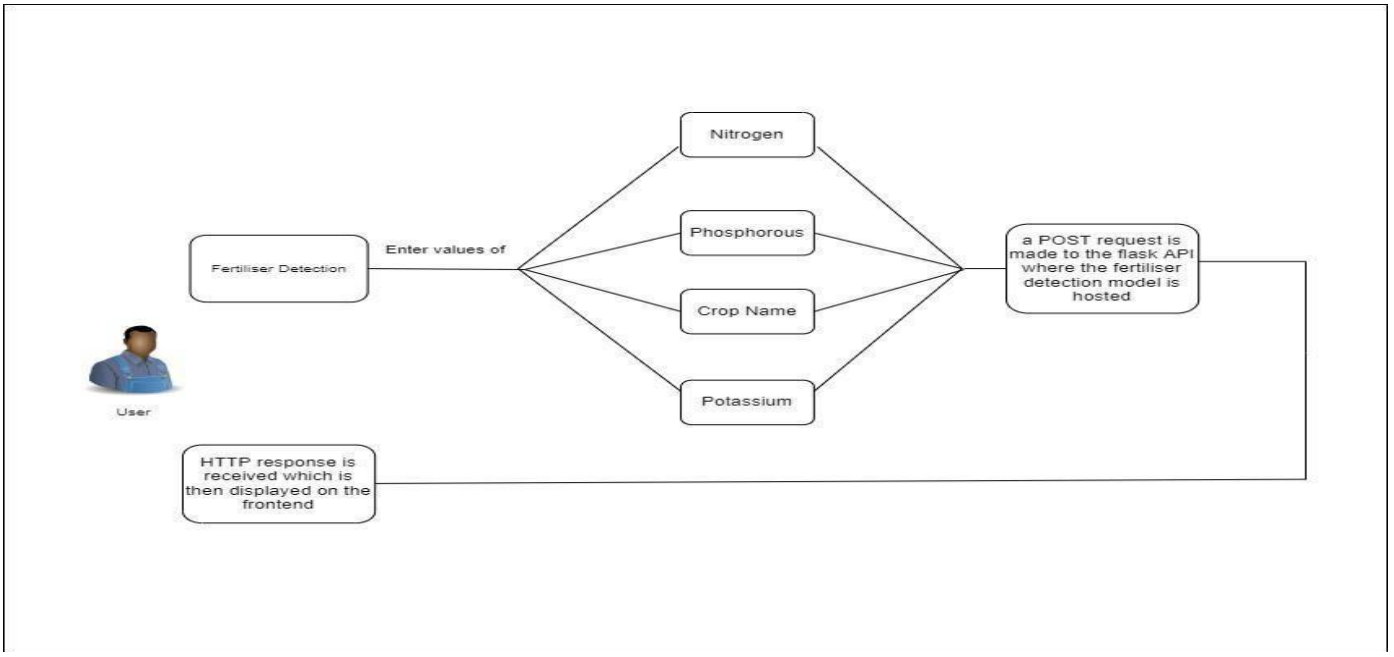


Figure 4.1.2 Flow diagram for Fertilizer Recommendation system

4.1.3 Flow diagram for Plant Disease Detection system

In disease detection the user has to click an image or directly upload it. The image is sent to the back end and is processed by the model. After the image has been processed an HTTP response is sent to the front-end. The user receives the disease the plant has and its remedies. The disease portal provides a detailed view of various plant diseases and the kinds of products that may be bought to cure the plants of the disease

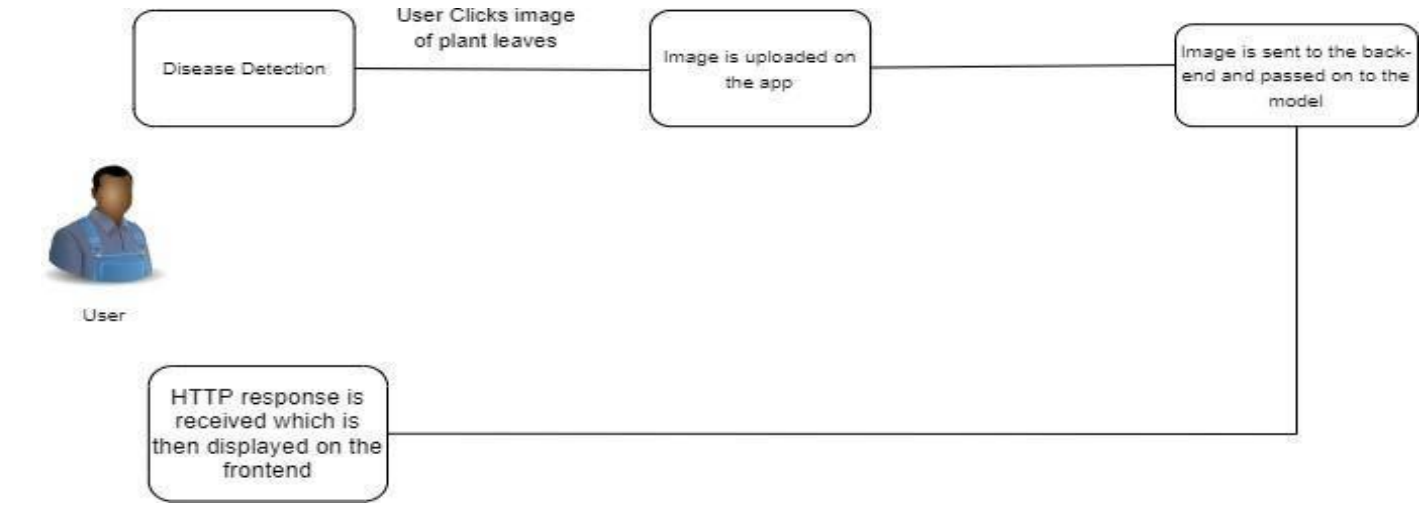


Figure4.1.3 Flow diagram for Plant Disease Detection system

4.1.4 Data Flow Diagram

The flowchart represents a workflow for crop recommendation based on farmer inputs and soil predictions. It begins with farmers providing data related to soil conditions and other relevant agricultural factors, which is collected and organized by the system. Soil-related predictions, such as nutrient composition and quality, are generated, along with crop suitability predictions based on the analyzed data. All inputs, predictions, and mappings are stored in a central datastore for further analysis and reference. The collected data is then mapped to identify patterns and correlations that assist in recommending suitable crops. Finally, the system outputs a list of crops tailored to the soil conditions and input data, helping farmers make informed decisions to optimize efficiency and yield.

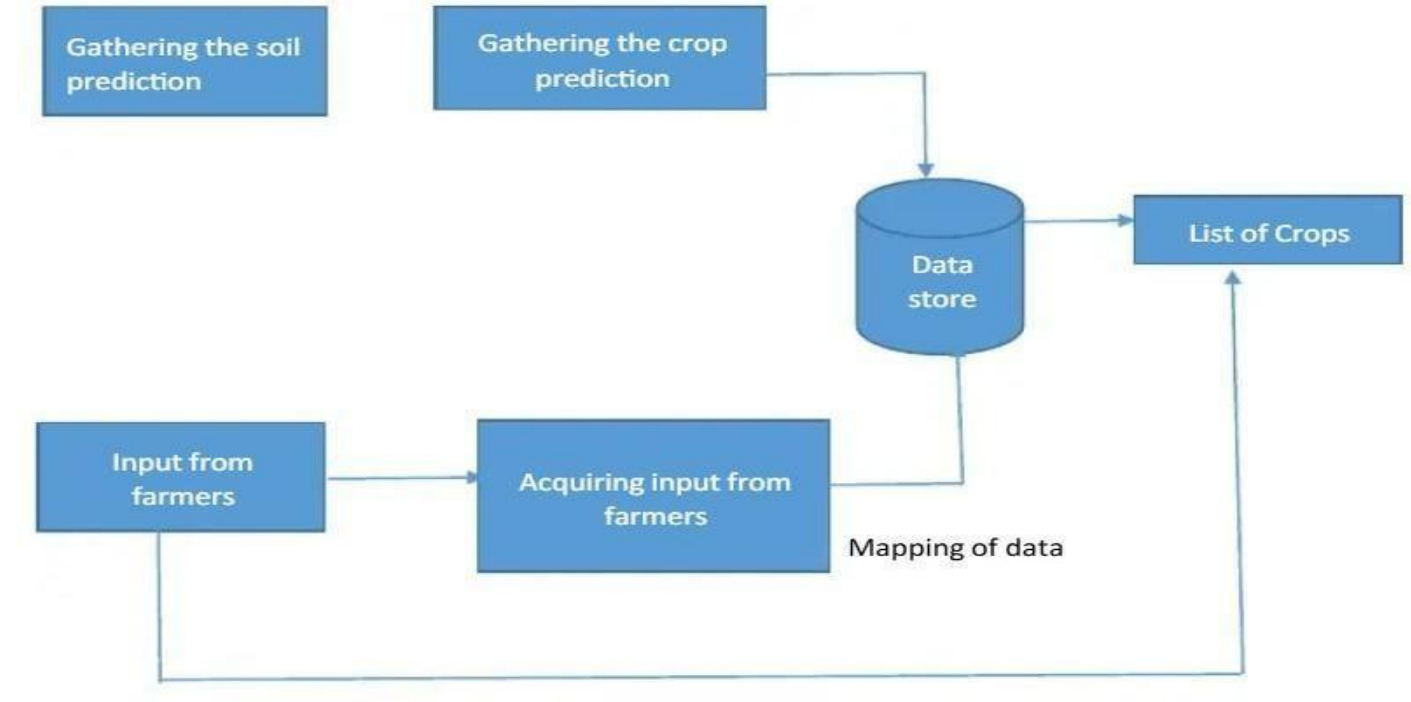


Fig. 4.1.4 Data Flow Diagram

4.1.5 Use Case Diagram

A use case diagram provides a visual representation of the functional requirements of a system and the interactions between its actors (users) and the system itself.

Benefits:

- Improved Accuracy: Preprocessing ensures consistency and boosts detection precision.
- Automation: Enables quick, efficient plant disease identification.
- Scalability: Augmented datasets enhance model robustness across conditions.
- Farmer Assistance: Provides timely, actionable insights for intervention.
- Consistency: Validates performance with separate training and testing datasets.
- Adaptability: Model updates accommodate new diseases and crop types.
- Reduced Human Error: Minimizes misdiagnosis through automation.

Limitations:

- Dependency on Quality Data: Performance relies on high-quality, diverse images; poor quality affects accuracy.
- Generalization Issues: Struggles to adapt to conditions not represented in the training data.
- Computational Requirements: Requires substantial resources, limiting use in low-resource environments.
- Real-World Complexity: Overlapping leaves and inconsistent lighting may impact detection accuracy.
- Limited Context Awareness: Considers only visual features, ignoring environmental factors.
- Initial Cost and Time: Involves high initial costs and significant time investment for development.

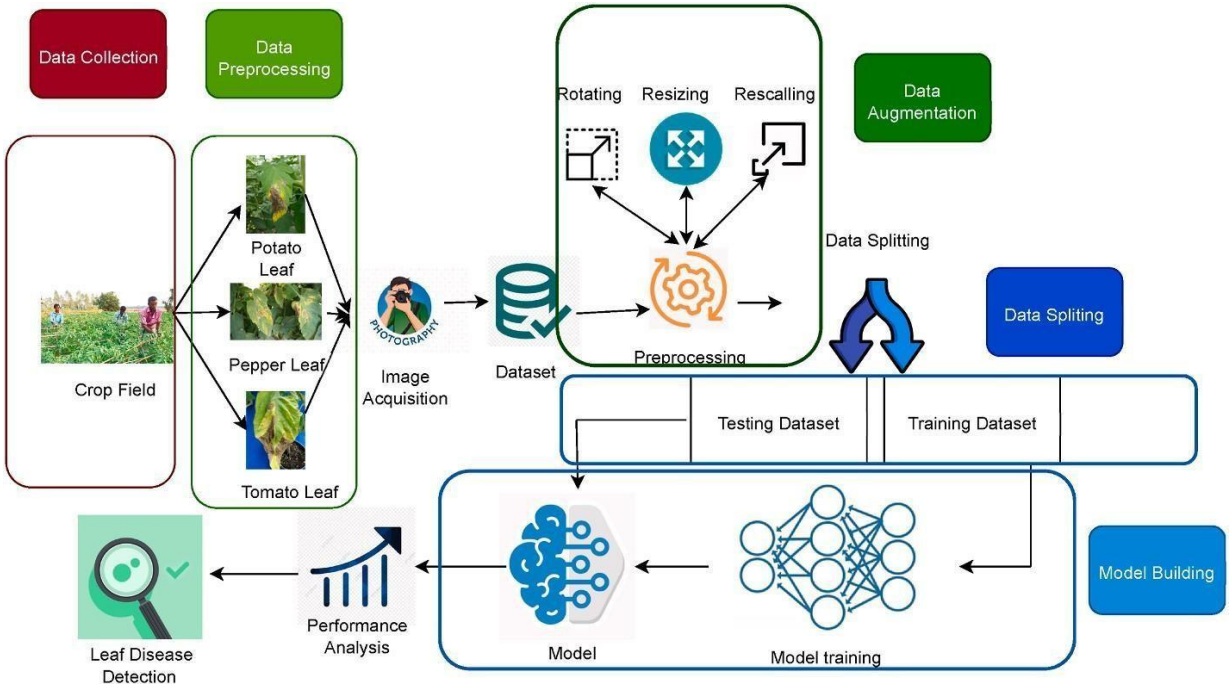


Fig. 4.1.5 Use Case Diagram

Actors:

- User Crop Field: Represents the environment where data (leaf images) is collected.
- Farmer or Photographer: Responsible for taking photos of crop leaves (Image Acquisition).
- Dataset: Stores images of potato, pepper, and tomato leaves.
- AI Model: Processes the dataset for training and testing.
- User or Analyst: Evaluates the model's performance and interprets the results.

Use Cases:

- Post/ Data Collection: Collecting images of crop leaves (potato, pepper, and tomato) from the crop field.
- Data Preprocessing: Rotating, resizing, and rescaling images. Augmenting data to ensure variety. Splitting data into training and testing datasets.
- Model Building: Training the machine learning model using the training dataset. Testing the model with the testing dataset.
- Leaf Disease Detection: Using the trained model to identify diseases in crop leaves.
- Performance Analysis: Measuring the model's accuracy and effectiveness.

4.1.6 Sequence Diagram

- Image Preprocessing: In this step, the uploaded image undergoes preprocessing to improve its quality and suitability for analysis.
- Preprocessing may involve operations like resizing, normalization, noise removal, and color adjustments.
- Segmentation: After preprocessing, the image is analyzed to segment or isolate the region of interest (e.g., the affected portion of the leaf). Techniques like thresholding, edge detection, or advanced methods such as U-Net models might be used here.
- Feature Extraction: Post-segmentation, relevant features are extracted from the segmented region.
- These features could include color, texture, shape, or other patterns essential for classification.
- CNN Classification: The extracted features are fed into a Convolutional Neural Network (CNN).
- The CNN classifies the image as either "Diseased" or "Healthy" based on the learned patterns.
- Final Output: The result is sent back to the user, providing information about the health status of the crop.

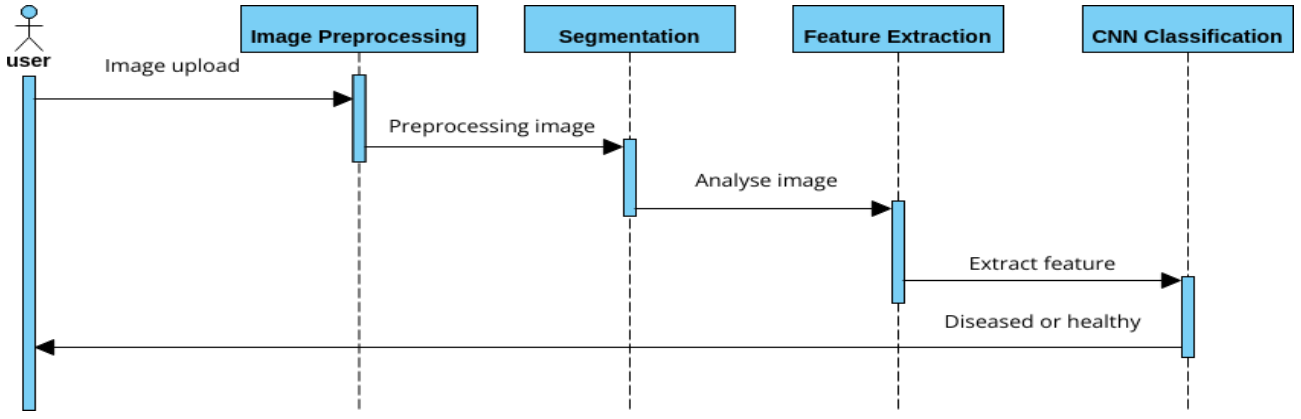


Fig.4.1.3 Sequence Diagram

4.1.7 Flow Chart

Training Process:

Training Dataset: Images of leaves are used as the initial dataset.

Data Annotation: This step involves labeling the dataset, marking specific areas or attributes (e.g., healthy or diseased regions on leaves).

Data Augmentation: To enhance the dataset, techniques like horizontal flip, vertical flip, rotation, zoom, and brightness adjustments are applied.

Data Processing: The images are resized to 256x256 pixels and normalized to prepare them for training.

Model Training: A machine learning model is trained using the processed data.

Saving Trained Model: The trained model is saved for future use.

Testing Process:

Testing Dataset: New images of leaves are used to test the trained model.

Data Processing: Like in training, the images are resized to 256x256 pixels and normalized.

Inference: The trained model predicts whether the leaf is healthy or identifies the type of disease.

Outcome: The system outputs the leaf's health status (healthy or specific disease type).

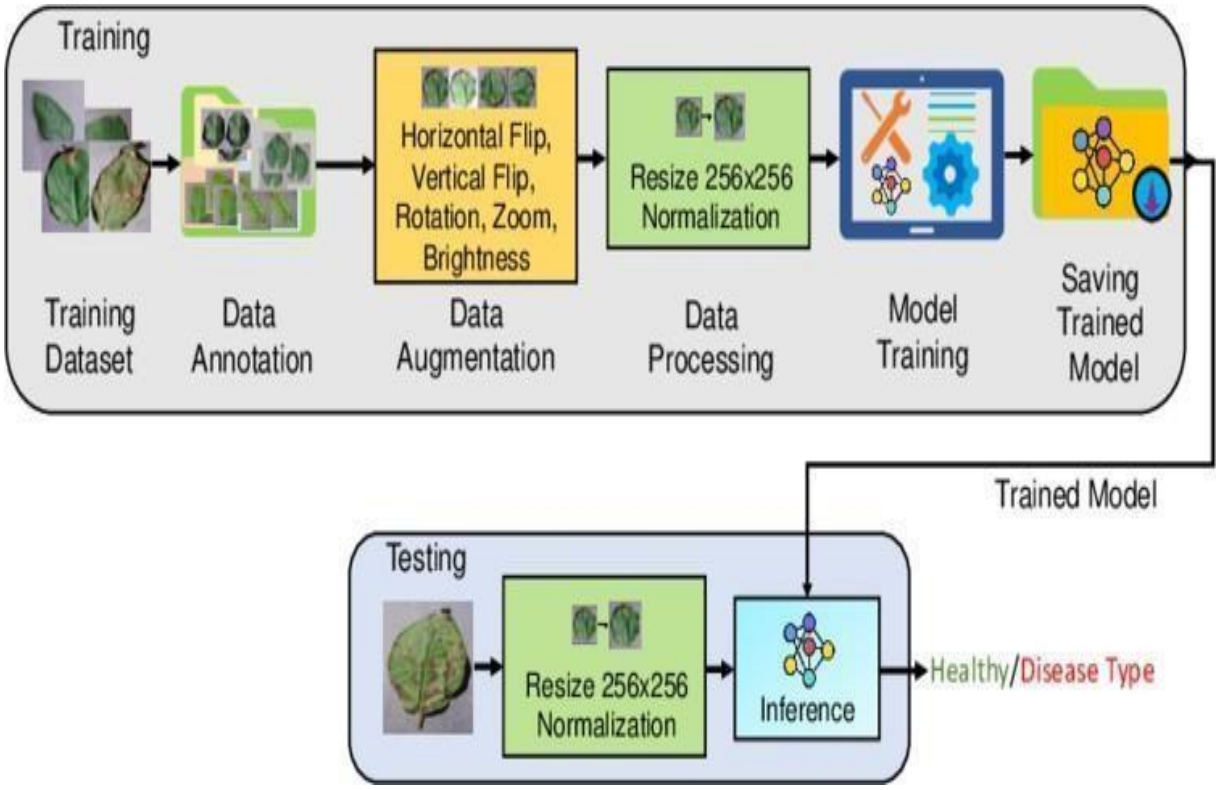


Fig. 4.1.4 Flow Chart

CHAPTER 5

SOFTWARE AND HARDWARE REQUIREMENTS

The software and hardware requirements for Cyberbullying detection using machine learning can vary depending on the complexity of the models and the size of the dataset. Here are some general requirements to consider:

5.1 Software Requirements

1. Python

Python stands out as a high-level, interpreted programming language renowned for its simplicity, readability, and versatility. One of Python's distinguishing features is its clean and concise syntax, which resembles pseudo- code and emphasizes readability. This readability makes Python an ideal language for both beginners and experienced developers alike.

Python's versatility manifests across various domains, contributing to its widespread adoption. In web development, frameworks like Django and Flask empower developers to build robust web applications efficiently.

Data science and machine learning represent another significant domain where Python shines. Libraries like NumPy, pandas, and SciPy provide powerful tools for data manipulation, analysis, and statistical computing.

2. Streamlit

Streamlit represents a groundbreaking addition to the Python ecosystem, revolutionizing the development of web applications, particularly those focused on machine learning and data science. At its core, Streamlit operates as a Python library that facilitates rapid development through a declarative, script-based approach. This abstraction layer empowers data scientists and machine learning engineers to focus on their core expertise without being encumbered by web development complexities.

Streamlit's intuitive API enables developers to create interactive elements, such as sliders, buttons, text inputs, and charts, with minimal code. By simply annotating Python variables or functions with decorators, developers can automatically generate corresponding user interface components, streamlining the development process significantly.

Another key feature of Streamlit is its seamless integration with Python's data science ecosystem. Developers can leverage popular libraries like Pandas, Matplotlib, Plotly, and scikit-learn within their Streamlit applications, facilitating data visualization, exploration, and analysis.

3. Pickle

Pickle, a core module in Python's standard library, serves as a fundamental tool for serializing and deserializing Python objects. The module, introduced by Fredrik Lundh in Python 1.4, enables developers to convert complex data structures, including custom objects, into a binary representation that can be stored persistently or transmitted over networks.

One of Pickle's distinguishing features is its support for arbitrary Python objects, including instances of user- defined classes. Unlike other serialization formats like JSON or XML, which are limited to primitive data types, Pickle can handle virtually any object that can be pickled, making it a versatile choice for preserving complex data structures.

Pickle's usage spans various domains, including data storage, inter-process communication, distributed computing, and machine learning. In the context of machine learning, Pickle plays a crucial role in persisting trained models, enabling developers to save model parameters, configurations, and preprocessing transformations for future use

Pickle serves as a foundational component in Python's ecosystem, providing a convenient and efficient mechanism for serializing and deserializing Python objects. While it offers significant benefits in terms of flexibility and ease of use, developers should be mindful of its limitations and security considerations when incorporating Pickle into their applications.

4. NLTK (Natural Language Toolkit)

NLTK (Natural Language Toolkit) stands as a comprehensive platform for building Python programs tailored to working with human language data. At its core, NLTK offers easy-to-use interfaces to over 50 corpora and lexical resources, including well-known datasets such as WordNet, Brown Corpus, and Treebank. NLTK's suite of text processing libraries encompasses a broad range of functionalities, including tokenization, stemming, lemmatization, part-of-speech tagging, parsing, and semantic reasoning. These capabilities form the building blocks for more complex NLP tasks.

NLTK's suite of text processing libraries encompasses a broad range of functionalities, including tokenization, stemming, lemmatization, part-of-speech tagging, parsing, and semantic reasoning. These capabilities form the building blocks for more complex NLP tasks.

5. HTML5 and CSS3

HTML5 (Hypertext Markup Language) is the standard markup language used for creating web pages and web applications. It provides the structure and content of a webpage, including text, images, links, and multimedia elements. CSS3 (Cascading Style Sheets) is a style sheet language used for describing the presentation and layout of HTML documents. CSS3 allows developers to customize the appearance of web pages by defining styles for fonts, colors, spacing, borders, and more.

5.2 Hardware Requirements

1. Processing Power

Machine learning tasks can be computationally intensive, especially when dealing with large datasets or complex models. A system with a multicore processor (e.g., Intel Core i5 or higher) or GPU acceleration can significantly speed up the training process.

2. Memory (RAM)

Sufficient RAM is essential for handling large datasets and training complex models. Aim for at least 8 GB of RAM, but more is recommended for larger datasets or deep learning models

3. Storage

Adequate storage is needed to store the historical Bitcoin price data, additional market data, and trained models. Depending on the size of the dataset, you may require several gigabytes or terabytes of storage.

4. Internet Connectivity

Access to the internet is crucial for collecting real-time or historical news articles, social media sentiment, or any other external data sources that may be relevant for the prediction models.

5. Cloud Services

For large-scale analysis or resource-intensive tasks, utilizing cloud platforms such as Amazon Web Services (AWS), Google Cloud Platform (GCP), or Microsoft Azure can provide access to scalable computing resources, storage, and specialized machine learning services.

CHAPTER 6

IMPLEMENTATION

Implementation refers to the process of putting a plan or idea into action. It involves taking the necessary steps to transform a concept or strategy into a tangible and functional reality. Implementation can be applied to various contexts, including software development, project management, business initiatives, and policy implementation.

6.1 Introduction to System Implementation

Implementation stage of an application creation is actualization of ideas, design and requirement specification into source code. The primary objective of implementation part of building a project is production of source codes with good style and comments, when necessary, by applying a proper and best coding technique which is suitable with the help of proper documents. Program codes are created in accordance with the structured coding techniques, which adheres to control flow, so that execution sequence follows the order in which codes are scripted. This makes the code unambiguous and more readable, which eases understanding, modifying, debugging, testing, and documentation of the programs.

6.2 Algorithms

6.2.1 Decision Tree

The decision tree algorithm is a machine learning model that predicts outcomes by splitting data into branches based on conditions at each node, making it highly interpretable and effective for multi-class classification tasks. In your AI-Driven Crop Disease Prediction and Management System, decision trees analyse features extracted from diseased leaf images, such as color and texture, to classify the type of disease affecting the crop. The system then suggests suitable treatments or management strategies based on these predictions. Its simplicity, speed, and explainability make it ideal for providing actionable insights to farmers in a user-friendly manner.

6.2.2 Naive Bayes

The Naive Bayes classifier is a probabilistic algorithm that uses Bayes' theorem, assuming feature independence. In your project, it classifies crop diseases by analyzing leaf features like color and texture. It computes probabilities for each disease class, enabling quick and reliable predictions. Its simplicity and efficiency make it ideal for small datasets and fast decision-making. This ensures effective disease identification and crop management.

6.2.3 Random Forest

Random Forest is an ensemble algorithm that combines multiple decision trees to improve accuracy and reduce overfitting. In your project, it analyzes features like texture, color, and shape from leaf images to classify crop diseases. By aggregating tree predictions, it ensures robust and reliable outcomes. Its ability to handle large datasets and diverse classes makes it ideal for disease diagnosis. This enhances the accuracy and effectiveness of

treatment recommendations.

6.2.4 XGBoost (Extreme Gradient Boosting)

XGBoost is a powerful machine learning algorithm that uses gradient boosting to build decision trees sequentially, improving prediction accuracy. In a Plant Disease Prediction Model, it classifies diseases by analyzing leaf features and correcting errors from previous trees. Known for its speed and performance, XGBoost handles complex datasets and outliers effectively. It reduces overfitting and provides high accuracy, making it suitable for plant disease identification. XGBoost's efficiency and robustness are ideal for real-world agricultural applications.

6.2.5 Support Vector Machine (SVM)

The Support Vector Machine (SVM) classifier is a machine learning algorithm that finds the optimal hyperplane to separate different classes of data. In a Plant Disease Prediction Model, SVM can classify diseases based on leaf features like texture and color. It's effective in high-dimensional spaces and provides accurate predictions. SVM can handle both linear and non-linear relationships using kernel functions. This makes it a strong choice for accurate and reliable crop disease identification.

6.3 Code

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>{{ title }}</title>
    <link rel="shortcut icon" href="{{ url_for('static', filename='images/favicon.ico') }}"/>
    <!-- for-mobile-apps -->
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta charset="utf-8">
    <meta name="keywords" content="Agro Harvest Responsive web template, Bootstrap Web Templates, Flat
Web Templates, Android Compatible web template,
Smartphone Compatible web template, free webdesigns for Nokia, Samsung, LG, SonyEricsson, Motorola web
design" />
    <style>
        html {
            font-size: 1rem;
        }
        @media (min-width: 576px) {
            html {
```

```
        font-size: 1.25rem;
    }
}
@media (min-width: 768px) {
    html {
        font-size: 1.5rem;
    }
}
@media (min-width: 992px) {
    html {
        font-size: 1.75rem;
    }
}
@media (min-width: 1200px) {
    html {
        font-size: 2rem;
    }
    html {
        font-size: 1rem;
    }
    h1 {
        font-size: 1.2rem;
    }
    h2 {
        font-size: 1.1rem;
    }
    @media (min-width: 768px) {
        html {
            font-size: 1.1rem;
        }
        h1 {
            font-size: 1.3rem;
        }
        h2 {
```

```
            font-size: 1.2rem;
        }
    }
    @media (min-width: 991px) {
        html {
            font-size: 1.2rem;
        }
        h1 {
            font-size: 1.5rem;
        }
        h2 {
            font-size: 1.4rem;
        }
    }
    @media (min-width: 1200px) {
        html {
            font-size: 1.2rem;
        }
        h1 {
            font-size: 1.7rem;
        }
        h2 {
            font-size: 1.6rem;
        }
    }
}
</style>
<script>
    addEventListener("load", function () {
        setTimeout(hideURLbar, 0);
    }, false);
    function hideURLbar() {
        window.scrollTo(0, 1);
    }
```

```
    </script>
    <script  src="https://code.jquery.com/jquery-3.3.1.slim.min.js"
      integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"
      crossorigin="anonymous"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"
      integrity="sha384-
UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"
      crossorigin="anonymous"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"
      integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"
      crossorigin="anonymous"></script>
    <script  src="https://code.jquery.com/jquery-3.5.1.slim.min.js"
      integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj"
      crossorigin="anonymous"></script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"
      integrity="sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo"
      crossorigin="anonymous"></script>
    </body>
    <!-- css files -->
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
      integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
crossorigin="anonymous">
    <link href="{{ url_for('static', filename='css/bootstrap.css') }}" rel='stylesheet' type='text/css' />
    <!-- bootstrap css -->
    <link href="{{ url_for('static', filename='css/style.css') }}" rel='stylesheet' type='text/css' />
    <!-- custom css -->
    <link href="{{ url_for('static', filename='css/font-awesome.min.css') }}" rel="stylesheet"><!-- fontawesome
css -->
    <!-- //css files -->
    <!-- <link rel="icon" type="image/png" href="{{ url_for('static', filename='images/favicon.png?') }}"> -->
    <script type="text/JavaScript" src="{{ url_for('static', filename='scripts/cities.js') }}"></script>
    <!-- google fonts -->
    <link href="//fonts.googleapis.com/css?family=Thasadith:400,400i,700,700i&amp;subset=latin-
ext,thai,vietnamese"
```

```
    rel="stylesheet">
  <!-- //google fonts -->
  <style>
    header {
      background-color: rgba(30, 30, 30, 1);
      margin-top: 0rem;
      display: block;
    }
  </style>
</head>
<body>
  <!-- Navigation -->
  <nav class="navbar navbar-expand-lg navbar-dark bg-dark static-top" style="background-color: #1C00ff00;">
    <div class="container">
      <a class="navbar-brand" href="{{ url_for('home') }}">
        <img src="{{ url_for('static', filename='images/go green.jpeg') }}" style="width: 90px; height: 80px"
alt="">
      </a>
      <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarResponsive"
        aria-controls="navbarResponsive" aria-expanded="false" aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
      </button>
      <div class="collapse navbar-collapse" id="navbarResponsive">
        <ul class="navbar-nav ml-auto">
          <li class="nav-item active">
            <a class="nav-link" href="{{ url_for('home') }}">Home
              <span class="sr-only">(current)</span>
            </a>
          </li>
          <li class="nav-item">
            <a class="nav-link" href="{{ url_for('crop_recommend') }}">Crop</a>
          </li>
          <li class="nav-item">+
            <a class="nav-link" href="{{ url_for('fertilizer_recommendation') }}">Fertilizer</a>
```

```
            </li>
            <li class="nav-item">
                <a class="nav-link" href="{{ url_for('disease_prediction') }}">Disease</a>
            </li>
        </ul>
    </div>
  </div>
</nav>
{% block body %} {% endblock %}
<!-- footer -->
<footer class="text-center py-5">
  <div class="container py-md-3">
    <!-- logo -->
    <h2 class="logo2 text-center">
      <a href="{{ url_for('home') }}">
        GO GREEN
      </a>
    </h2>
    <!-- //logo -->
    <!-- address -->
    <div class="contact-left-footer mt-4">
      <!-- <a href="community.html">Community</a> -->
      </p>
    </div>
    <div class="w3l-copy text-center">
      <p class="text-da">An Environmental Intelligence Startup<br> </p>
    </div>
    <p class="homelogo">
    <p>Made with  by Shashishekhara K</p>
      <p>Shivashankar M G</p>
      <p>Vinay G V</p>
      <p>Praveen B M
    </p>
    <h1 style="color: lightgreen; font-weight: bold;">Thank you</h1>
```

```
</div>
    </footer>
    <!-- //footer -->
    <!-- move top icon -->
    <a href="#home" class="move-top text-center"></a>
    <!-- //move top icon -->
</body>
</html>
```

Backend Code

```
# Importing essential libraries and modules
from flask import Flask, render_template, request, redirect
from markupsafe import Markup
import numpy as np
import pandas as pd
import requests
import config
import pickle
import io
import torch
from torchvision import transforms
from PIL import Image
from utils.model import ResNet9
from utils.disease import disease_dic
from utils.fertilizer import fertilizer_dic
# ==============================================================================
# ----------------------------LOADING THE TRAINED MODELS -----------------------------------------------
# Loading plant disease classification model
disease_classes = ['Apple_Apple_scab', 'Apple_____Black_rot', 'Apple___Cedar_apple_rust', 'Apple___healthy',
                   'Blueberry___healthy', 'Cherry_(including_sour)___Powdery_mildew',
'Cherry_(including_sour)___healthy',
                   'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot', 'Corn_(maize)___Common_rust_',
                   'Corn_(maize)___Northern_Leaf_Blight', 'Corn_(maize)___healthy', 'Grape___Black_rot',
                   'Grape___Esca_(Black_Measles)', 'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)', 'Grape___healthy',
                   'Orange___Haunglongbing_(Citrus_greening)', 'Peach___Bacterial_spot', 'Peach___healthy',
```

```
            'Pepper,_bell___Bacterial_spot', 'Pepper,_bell___healthy', 'Potato___Early_blight',
'Potato___Late_blight',
            'Potato___healthy', 'Raspberry___healthy', 'Soybean___healthy', 'Squash___Powdery_mildew',
            'Strawberry___Leaf_scorch', 'Strawberry___healthy', 'Tomato___Bacterial_spot',
'Tomato___Early_blight',
            'Tomato___Late_blight', 'Tomato___Leaf_Mold', 'Tomato___Septoria_leaf_spot',
            'Tomato___Spider_mites Two-spotted_spider_mite', 'Tomato___Target_Spot',
'Tomato___Tomato_Yellow_Leaf_Curl_Virus',
            'Tomato___Tomato_mosaic_virus', 'Tomato___healthy']

disease_model_path = 'models/plant_disease_model.pth'
disease_model = ResNet9(3, len(disease_classes))
disease_model.load_state_dict(torch.load(disease_model_path, map_location=torch.device('cpu')))
disease_model.eval()

# Loading crop recommendation model
crop_recommendation_model_path = 'models/RandomForest.pkl'
crop_recommendation_model = pickle.load(open(crop_recommendation_model_path, 'rb'))

#
==========================================================================
==============

# Custom functions for calculations

def weather_fetch(city_name):
    """
    Fetches and returns the temperature and humidity of a city.
    :params: city_name
    :return: temperature, humidity or None if the request fails
    """
    try:
        api_key = config.weather_api_key
        base_url = "http://api.openweathermap.org/data/2.5/weather?"
```

```
        complete_url = base_url + "appid=" + api_key + "&q=" + city_name
        response = requests.get(complete_url)
        x = response.json()

        if x["cod"] != "404":  # Check if the city is found
            y = x["main"]
            temperature = round((y["temp"] - 273.15), 2)
            humidity = y["humidity"]
            return temperature, humidity
        else:
            print(f"City {city_name} not found.")
            return None
    except Exception as e:
        print(f"Error fetching weather data for {city_name}: {e}")
        return None

def predict_image(img, model=disease_model):
    """
    Transforms image to tensor and predicts disease label
    :params: image
    :return: prediction (string)
    """
    transform = transforms.Compose([
        transforms.Resize(256),
        transforms.ToTensor(),
    ])
    image = Image.open(io.BytesIO(img))
    img_t = transform(image)
    img_u = torch.unsqueeze(img_t, 0)

    # Get predictions from model
    yb = model(img_u)
    _, preds = torch.max(yb, dim=1)
    prediction = disease_classes[preds[0].item()]
```

```
    return prediction

#
================================================================================
===============

# ---------------------------------------- FLASK APP ----------------------------------------------------

app = Flask(__name__)

# render home page
@app.route('/')
def home():
    title = 'Harvestify - Home'
    return render_template('index.html', title=title)

# render crop recommendation form page
@app.route('/crop-recommend')
def crop_recommend():
    title = 'Harvestify - Crop Recommendation'
    return render_template('crop.html', title=title)

# render fertilizer recommendation form page
@app.route('/fertilizer')
def fertilizer_recommendation():
    title = 'Harvestify - Fertilizer Suggestion'
    return render_template('fertilizer.html', title=title)

#
================================================================================
===============

# RENDER PREDICTION PAGES
```

```
# render crop recommendation result page
@app.route('/crop-predict', methods=['POST'])
def crop_prediction():
    title = 'Harvestify - Crop Recommendation'
    try:
        # Log the received form data
        print("Received form data:", request.form)

        N = int(request.form['nitrogen'])
        P = int(request.form['phosphorous'])
        K = int(request.form['pottasium'])
        ph = float(request.form['ph'])
        rainfall = float(request.form['rainfall'])
        state = request.form.get("stt")
        city = request.form.get("city")

        print(f"N: {N}, P: {P}, K: {K}, pH: {ph}, Rainfall: {rainfall}, State: {state}, City: {city}")

        # Verify weather data
        weather = weather_fetch(city)
        if weather:
            temperature, humidity = weather
            print(f"Weather: Temp={temperature}, Humidity={humidity}")
            # Prepare input data
            data = np.array([[N, P, K, temperature, humidity, ph, rainfall]])
            print("Input Data:", data)
            # Attempt model prediction
            try:
                my_prediction = crop_recommendation_model.predict(data)
                final_prediction = my_prediction[0]
                print(f"Prediction: {final_prediction}")
                return render_template('crop-result.html', prediction=final_prediction, title=title)
            except Exception as e:
                print(f"Prediction error: {e}")
```

```
            return render_template('try_again.html', title=title, error_message="Prediction error")
        else:
            error_message = f"Weather data not available for city: {city}. Please check the city name or try again."
            return render_template('try_again.html', title=title, error_message=error_message)

    except ValueError as ve:
        print(f"ValueError (input issue): {ve}")
        return render_template('try_again.html', title=title, error_message="Invalid input values")

    except Exception as e:
        print(f"Error: {e}")
        return render_template('try_again.html', title=title, error_message="Unexpected error occurred")

# render fertilizer recommendation result page
@app.route('/fertilizer-predict', methods=['POST'])
def fert_recommend():
    title = 'Harvestify - Fertilizer Suggestion'
    crop_name = str(request.form['cropname'])
    N = int(request.form['nitrogen'])
    P = int(request.form['phosphorous'])
    K = int(request.form['pottasium'])

    df = pd.read_csv('C:/Users/91901/CropcareAI-main/CropcareAI-main/Data-raw/FertilizerData.csv')

    nr = df[df['Crop'] == crop_name]['N'].iloc[0]
    pr = df[df['Crop'] == crop_name]['P'].iloc[0]
    kr = df[df['Crop'] == crop_name]['K'].iloc[0]

    n = nr - N
    p = pr - P
    k = kr - K
    temp = {abs(n): "N", abs(p): "P", abs(k): "K"}
    max_value = temp[max(temp.keys())]
    if max_value == "N":
```

```
        if n < 0:
            key = 'NHigh'
        else:
            key = "Nlow"
    elif max_value == "P":
        if p < 0:
            key = 'PHigh'
        else:
            key = "Plow"
    else:
        if k < 0:
            key = 'KHigh'
        else:
            key = "Klow"

    response = Markup(str(fertilizer_dic[key]))

    return render_template('fertilizer-result.html', recommendation=response, title=title)

# render disease prediction result page
@app.route('/disease-predict', methods=['GET', 'POST'])
def disease_prediction():
    title = 'Harvestify - Disease Detection'

    if request.method == 'POST':
        if 'file' not in request.files:
            return redirect(request.url)
        file = request.files.get('file')
        if not file:
            return render_template('disease.html', title=title)
        try:
            img = file.read()
            prediction = predict_image(img)
            prediction = Markup(str(disease_dic[prediction]))
```

```
            return render_template('disease-result.html', prediction=prediction, title=title)
        except Exception as e:
            print(f"Error during disease prediction: {e}")
            return render_template('try_again.html', title=title, error_message="Error during prediction")
    return render_template('disease.html', title=title)

# ========================================================================================
import warnings
warnings.filterwarnings("ignore", category=UserWarning, module="flask")

if __name__ == '__main__':
    app.run(debug=True)

</style> """
disease_dic = {
    'Apple___Apple_scab': """
  <b>Crop</b>: Apple <br/>Disease:
  Apple Scab<br/>
        <br/> Cause of disease:

        <br/><br/> 1. Apple scab
  overwinters primarily in fallen leaves
  and in the soil. Disease development is
  favored by wet, cool weather that
  generally occurs in spring and early
  summer.

        <br/> 2. Fungal spores are carried by
  wind, rain or splashing water from the
  ground to flowers, leaves or fruit. During
  damp or rainy periods, newly opening
  apple leaves are extremely susceptible to
  infection. The longer the leaves remain
  wet, the more severe the infection will
```

be. Apple scab spreads rapidly between 55-75 degrees Fahrenheit.

 How to prevent/cure the disease

1. Choose resistant varieties when possible.

2. Rake under trees and destroy infected leaves to reduce the number of fungal spores available to start the disease cycle over again next spring

3. Water in the evening or early morning hours (avoid overhead irrigation) to give the leaves time to dry out before infection can occur.

4. Spread a 3- to 6-inch layer of compost under trees, keeping it away from the trunk, to cover soil and prevent splash dispersal of the fungal spores.""",

'Apple___Black_rot': """ Crop: Apple
Disease: Black Rot

 Cause of disease:

Black rot is caused by the fungus Diplodia seriata (syn Botryosphaeria obtusa).The fungus can infect dead tissue as well as living

 How to prevent/cure the disease

1. Prune out dead or diseased branches.

```
        tweet = re.sub(r':[^\s]+[\s]?',",tweet)
        # remove special characters
        tweet = re.sub('[^ a-zA-Z0-9]' , '', tweet)
        # remove RT
        tweet = re.sub('RT' , '', tweet)
        # remove Numbers
        tweet = re.sub('[0-9]', '', tweet)
    return tweet
    def transform_text(text):
        text = text.lower()
        text = nltk.word_tokenize(text)
            y=[]
        for i in text:


            if i.isalnum():
                y.append(i)
            text = y[:]


        y.clear()
     for i in text:
            if i not in stopwords.words('english') and i not in string.punctuation:
                y.append(i)
    text = y[:]


        y.clear()
    for i in text:
                y.append(ps.stem(i))
    return " ".join(y)
    tfidf = pickle.load(open('pickle/TFIDFvectorizer.pkl','rb'))
    model = pickle.load(open('pickle/bestmodel.pkl','rb'))
    st.markdown("---")
```

```
st.markdown("<br>", unsafe_allow_html=True)

input_text = st.text_area("**_Enter the text to analyze_**", key="**_Enter the text to analyze_**")
col1, col2 = st.columns([1,6])
with col1:

    button_predict = st.button('Predict')
with col2:
def clear_text():

        st.session_state["**_Enter the text to analyze_**"] = ""
 # clear button
    button_clear = st.button("Clear", on_click=clear_text)
st.markdown("---")
    # predict button animations
if button_predict:
    if input_text == "":

     st.snow()

     st.warning("Please provide some text!")
    else:
        with st.spinner("**_Prediction_** in progress. Please wait *    "):
            time.sleep(3)
    # 1. preprocess

 cleanText = clean_text(input_text)
 transformText = transform_text(cleanText)
 # 2. vectorize
vector_input = tfidf.transform([transformText])
```

```
# 3. predict
result = model.predict(vector_input)[0]


# result2 = model.predict_proba(vector_input)[0]
      #clf=svm.SVC(probability=True)
 # 4. display


      if result == 1 :
         st.subheader("Result")

         # st.markdown(result2)
      else:


         st.subheader("Result") # st.markdown(result2)
      st.markdown("---")
      st.subheader("Original  Text")
      expander_original = st.expander("Information", expanded=False)
      with expander_original:
         st.info("The text that the user provided!")
      st.text(input_text)
      st.markdown("---")
      st.subheader("Cleaned Text")
      expander_clean = st.expander("Information", expanded=False)
      with expander_clean:
         st.info("From original text has removed punctuation and special characters. Also, it has removed
hashtags, tags and emoji's!")

     st.text(cleanText)
      st.markdown("---")
      st.subheader("Transformed Text")

      expander_transform = st.expander("Information", expanded=False)
      with expander_transform:
```

```
        st.info("From Cleaned text has removed stopwords and transformed to lowercase. Also, it has be used
  Stemming!")

        st.text(transformText)
        st.markdown("---")
        st.subheader("Binary Prediction")
        expander_binary = st.expander("Information", expanded=False)
        with expander_binary:
            st.info("Binary Prediction from the Model!")
        if result == 1:
            st.markdown(":red["+ str(result) +"]")
        else:
            st.markdown(":green["+ str(result) +"]")
        st.markdown("---")
        st.subheader("Model Accuracy")


        expander_accuracy = st.expander("Information", expanded=False)
        with expander_accuracy:
            st.info("Model Accuracy using Random Forest (RF) Classifier!")
        st.warning("Accuracy:  **_91.70 %_**")


    import streamlit as st
from PIL import Image
hide_menu = """
  <style>
  #MainMenu{
     visibility:hidden;


  }


  footer{


     visibility:hidden;
```

```
}

</style>
showWarningOnDirectExecution = False
image = Image.open('icons/logo.png')
st.set_page_config(page_title = "Algorithms", page_icon = image)
st.markdown(hide_menu, unsafe_allow_html=True)
st.sidebar.image(image , use_column_width=True, output_format='auto')
st.sidebar.markdown("---")
st.sidebar.markdown(" <br> <br> <br> <br> <br> <br> <br> <h1 style='text-align: center; font-size: 18px;
color: #0080FF;'>© 2023 | Ioannis Bakomichalis</h1>", unsafe_allow_html=True)
st.markdown("---")
st.markdown("<br>", unsafe_allow_html=True)


data_choice = st.selectbox("Dataset", all_Datasets)


all_Vectorizers = ["Select a Vectorizer", "TF-IDF", "CountVectorizer"]
vect_choice = st.selectbox("Vectorizer", all_Vectorizers)
all_ML_models = ["Select a Machine Learning Algorithm", "Logistic Regression", "Decision Tree", "Random
Forest", "XGBoost", "Naive Bayes", "Support Vector Machine", "Bagging Decision Tree", "Boosting
Decision Tree"]
model_choice = st.selectbox("Machine Learning Algorithm", all_ML_models)
st.markdown("<br>", unsafe_allow_html=True)
st.markdown("---")
```

```
if data_choice == "Select a Dataset" and vect_choice != "Select a Vectorizer" and model_choice != "Select a Machine Learning Algorithm":
    st.warning(":white[You should select **_Dataset_**]")

elif data_choice != "Select a Dataset" and vect_choice == "Select a Vectorizer" and model_choice != "Select a Machine Learning Algorithm":
    st.warning(":white[You should select **_Vectorizer_**]")
elif data_choice != "Select a Dataset" and vect_choice != "Select a Vectorizer" and model_choice == "Select a Machine Learning Algorithm":
    st.warning(":white[You should select **_Machine Learning Algorithm_**]")

elif data_choice == "Select a Dataset" and vect_choice == "Select a Vectorizer" and model_choice != "Select a Machine Learning Algorithm":
    st.warning(":white[You should select **_Dataset_** and **_Vectorizer_**]")

elif data_choice == "Select a Dataset" and vect_choice != "Select a Vectorizer" and model_choice == "Select a Machine Learning Algorithm":
    st.warning(":white[You should select **_Dataset_** and **_Machine Learning Algorithm_**]")

elif data_choice != "Select a Dataset" and vect_choice == "Select a Vectorizer" and model_choice == "Select a Machine Learning Algorithm":
    st.warning(":white[You should select **_Vectorizer_** and **_Machine Learning Algorithm_**]")
else:
    # if token_choice == "Tokenizing":
        if vect_choice == "TF-IDF":

            if model_choice == "Logistic Regression":
                st.markdown("<br>", unsafe_allow_html=True)
```

```
        st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_90.88%_**]")
        elif model_choice == "Decision Tree":

            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_87.38%_**]")
        elif model_choice == "Random Forest":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_89.71%_**]")
        elif model_choice == "XGBoost":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_87.38%_**]")
        elif model_choice == "Naive Bayes":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_88.78%_**]")
        elif model_choice == "Support Vector Machine":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_89.71%_**]")
        elif model_choice == "Bagging Decision Tree":

            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_88.08%_**]")
        elif model_choice == "Boosting Decision Tree":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_87.38%_**]")
    elif vect_choice == "CountVectorizer":


        if model_choice == "Logistic Regression":
            st.markdown("<br>", unsafe_allow_html=True)
```

```
            st.subheader("Evaluation Metrics")
                st.success(":green[Accuracy: **_90.65%_**]")
            elif model_choice == "Decision Tree":
                st.markdown("<br>", unsafe_allow_html=True)
                st.subheader("Evaluation Metrics")
                st.success(":green[Accuracy: **_89.71%_**]")
            elif model_choice == "Random Forest":
                st.markdown("<br>", unsafe_allow_html=True)
                st.subheader("Evaluation Metrics")
                st.success(":green[Accuracy: **_90.18%_**]")
            elif model_choice == "XGBoost":
                st.markdown("<br>", unsafe_allow_html=True)
                st.subheader("Evaluation Metrics")
                st.success(":green[Accuracy: **_89.95%_**]")
            elif model_choice == "Naive Bayes":
                st.markdown("<br>", unsafe_allow_html=True)
                st.subheader("Evaluation Metrics")
                st.success(":green[Accuracy: **_90.88%_**]")
            elif model_choice == "Support Vector Machine":
                st.markdown("<br>", unsafe_allow_html=True)
                st.subheader("Evaluation Metrics")
                st.success(":green[Accuracy: **_90.88%_**]")
            elif model_choice == "Bagging Decision Tree":
                st.markdown("<br>", unsafe_allow_html=True)
                st.subheader("Evaluation Metrics")
                st.success(":green[Accuracy: **_89.25%_**]")
            elif model_choice == "Boosting Decision Tree":
                st.markdown("<br>", unsafe_allow_html=True)
                st.subheader("Evaluation Metrics")
                st.success(":green[Accuracy: **_89.01%_**]")
    elif  data_choice == "Cyber Troll Dataset":
        if vect_choice == "TF-IDF":
            if model_choice == "Logistic Regression":
                st.markdown("<br>", unsafe_allow_html=True)
```

```
        st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_76.08%_**]")
        elif model_choice == "Decision Tree":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_85.92%_**]")
        elif model_choice == "Random Forest":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_91.70%_**]")
        elif model_choice == "XGBoost":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_76.00%_**]")
        elif model_choice == "Naive Bayes":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy:  **_78.20%_**]")
        elif model_choice == "Support Vector Machine":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_80.50%_**]")
        elif model_choice == "Bagging Decision Tree":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_84.87%_**]")
        elif model_choice == "Boosting Decision Tree":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_91.22%_**]")
    elif vect_choice == "CountVectorizer":


        if model_choice == "Logistic Regression":
            st.markdown("<br>", unsafe_allow_html=True)
```

```
        st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_81.57%_**]")
        elif model_choice == "Decision Tree":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_84.60%_**]")
        elif model_choice == "Random Forest":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_86.47%_**]")
        elif model_choice == "XGBoost":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_73.48%_**]")
        elif model_choice == "Naive Bayes":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_79.73%_**]")
        elif model_choice == "Support Vector Machine":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_83.72%_**]")
        elif model_choice == "Bagging Decision Tree":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_81.75%_**]")
        elif model_choice == "Boosting Decision Tree":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_89.35%_**]")
    elif  data_choice == "Classified Tweets Dataset":
        if vect_choice == "TF-IDF":
            if model_choice == "Logistic Regression":
                st.markdown("<br>", unsafe_allow_html=True)
```

```
        st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_90.52%_**]")
        elif model_choice == "Decision Tree":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_89.10%_**]")
        elif model_choice == "Random Forest":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")

        elif model_choice == "XGBoost":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_90.95%_**]")
        elif model_choice == "Naive Bayes":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_87.89%_**]")
        elif model_choice == "Support Vector Machine":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_91.30%_**]")
        elif model_choice == "Bagging Decision Tree":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_91.17%_**]")
        elif model_choice == "Boosting Decision Tree":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")

            st.success(":green[Accuracy: **_90.24%_**]")
    elif vect_choice == "CountVectorizer":
        if model_choice == "Logistic Regression":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_91.07%_**]")
```

```
        elif model_choice == "Decision Tree":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_88.90%_**]")
        elif model_choice == "Random Forest":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_90.92%_**]")
        elif model_choice == "XGBoost":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_91.38%_**]")
        elif model_choice == "Naive Bayes":
            st.markdown("<br>", unsafe_allow_html=True)

            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy:  **_90.39%_**]")
        elif model_choice == "Support Vector Machine":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_90.24%_**]")
        elif model_choice == "Bagging Decision Tree":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_90.95%_**]")
        elif model_choice == "Boosting Decision Tree":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_89.40%_**]")
    elif  data_choice == " Classification Dataset": if
        vect_choice == "TF-IDF":
            if model_choice == "Logistic Regression":
                st.markdown("<br>", unsafe_allow_html=True)
                st.subheader("Evaluation Metrics")
                st.success(":green[Accuracy: **_86.10%_**]")
```

```
        elif model_choice == "Decision Tree":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_82.23%_**]")
        elif model_choice == "Random Forest":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_84.12%_**]")
        elif model_choice == "XGBoost":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_86.26%_**]")
        elif model_choice == "Naive Bayes":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_84.67%_**]")
        elif model_choice == "Support Vector Machine":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_86.27%_**]")

        elif model_choice == "Bagging Decision Tree":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_84.61%_**]")
        elif model_choice == "Boosting Decision Tree":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_83.54%_**]")
    elif vect_choice == "CountVectorizer":


        if model_choice == "Logistic Regression":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
```

```
            st.success(":green[Accuracy: **_86.10%_**]")
            elif model_choice == "Decision Tree":
                st.markdown("<br>", unsafe_allow_html=True)
                st.subheader("Evaluation Metrics")
                st.success(":green[Accuracy: **_82.07%_**]")
            elif model_choice == "Random Forest":
                st.markdown("<br>", unsafe_allow_html=True)
                st. subheader("Evaluation Metrics")
elif model_choice == "XGBoost": st.markdown("<br>",
unsafe_allow_html=True) st.subheader("Evaluation
Metrics") st.success(":green[Accuracy: **_86.26%_**]")
            elif model_choice == "Naive Bayes":
                st.markdown("<br>", unsafe_allow_html=True)
                st.subheader("Evaluation Metrics")
                st.success(":green[Accuracy: **_84.67%_**]")
            elif model_choice == "Support Vector Machine":
                st.markdown("<br>", unsafe_allow_html=True)
                st.subheader("Evaluation Metrics")
                st.success(":green[Accuracy: **_86.27%_**]")
            elif model_choice == "Bagging Decision Tree":
                st.markdown("<br>", unsafe_allow_html=True)
                st.subheader("Evaluation Metrics")
                st.success(":green[Accuracy: **_84.19%_**]")
            elif model_choice == "Boosting Decision Tree":
                st.markdown("<br>", unsafe_allow_html=True)
                st.subheader("Evaluation Metrics")

                st.success(":green[Accuracy:  **_83.55%_**]")


        elif  data_choice == " Types Dataset + Troll Dataset": if vect_choice ==
            "TF-IDF":
                if model_choice == "Logistic Regression":
                    st.markdown("<br>", unsafe_allow_html=True)
                    st.subheader("Evaluation Metrics")
                    st.success(":green[Accuracy: **_78.12%_**]")
```

```
        elif model_choice == "Decision Tree":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_85.45%_**]")
        elif model_choice == "Random Forest":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_90.29%_**]")
        elif model_choice == "XGBoost":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_75.52%_**]")
        elif model_choice == "Naive Bayes":

            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_79.95%_**]")
        elif model_choice == "Support Vector Machine":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_80.60%_**]")
        elif model_choice == "Bagging Decision Tree":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_84.84%_**]")
        elif model_choice == "Boosting Decision Tree":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_90.06%_**]")
    elif vect_choice == "CountVectorizer":


        if model_choice == "Logistic Regression":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
```

```
        st.success(":green[Accuracy: **_81.89%_**]")

    elif model_choice == "Decision Tree":
        st.markdown("<br>", unsafe_allow_html=True)
        st.subheader("Evaluation Metrics")
        st.success(":green[Accuracy: **_83.60%_**]")
    elif model_choice == "Random Forest":
        st.markdown("<br>", unsafe_allow_html=True)
        st.subheader("Evaluation Metrics")
        st.success(":green[Accuracy: **_85.52%_**]")
    elif model_choice == "XGBoost":
        st.markdown("<br>", unsafe_allow_html=True)
        st.subheader("Evaluation Metrics")
        st.success(":green[Accuracy: **_73.51%_**]")
    elif model_choice == "Naive Bayes":
        st.markdown("<br>", unsafe_allow_html=True)
        st.subheader("Evaluation Metrics")
        st.success(":green[Accuracy: **_79.58%_**]")
    elif model_choice == "Support Vector Machine":
        st.markdown("<br>", unsafe_allow_html=True)
        st.subheader("Evaluation Metrics")
        st.success(":green[Accuracy: **_83.24%_**]")

    elif model_choice == "Bagging Decision Tree":
        st.markdown("<br>", unsafe_allow_html=True)
        st.subheader("Evaluation Metrics")
        st.success(":green[Accuracy: **_82.45%_**]")
    elif model_choice == "Boosting Decision Tree":
        st.markdown("<br>", unsafe_allow_html=True)
        st.subheader("Evaluation Metrics")
        st.success(":green[Accuracy: **_89.16%_**]")

if vect_choice == "TF-IDF":


    if model_choice == "Logistic Regression":
```

```
        st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_84.57%_**]")
        elif model_choice == "Decision Tree":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_80.03%_**]")
        elif model_choice == "Random Forest":

            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_81.77%_**]")
        elif model_choice == "XGBoost":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_84.50%_**]")
        elif model_choice == "Naive Bayes":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_74.90%_**]")
        elif model_choice == "Support Vector Machine":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_84.72%_**]")
        elif model_choice == "Bagging Decision Tree":
            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_82.69%_**]")
        elif model_choice == "Boosting Decision Tree":

            st.markdown("<br>", unsafe_allow_html=True)
            st.subheader("Evaluation Metrics")
            st.success(":green[Accuracy: **_80.65%_**]")
    elif vect_choice == "CountVectorizer":


        if model_choice == "Logistic Regression":
```

```
st.markdown("<br>", unsafe_allow_html=True)
    st.subheader("Evaluation Metrics")
    st.success(":green[Accuracy: **_84.57%_**]")
elif model_choice == "Decision Tree":
    st.markdown("<br>", unsafe_allow_html=True)
    st.subheader("Evaluation Metrics")
    st.success(":green[Accuracy: **_80.11%_**]")
elif model_choice == "Random Forest":
    st.markdown("<br>", unsafe_allow_html=True)
    st.subheader("Evaluation Metrics")
    st.success(":green[Accuracy: **_82.03%_**]")
elif model_choice == "XGBoost":
    st.markdown("<br>", unsafe_allow_html=True)
    st.subheader("Evaluation Metrics")
    st.success(":green[Accuracy: **_84.50%_**]")
elif model_choice == "Naive Bayes":
    st.markdown("<br>", unsafe_allow_html=True)
    st.subheader("Evaluation Metrics")
    st.success(":green[Accuracy: **_74.90%_**]")
elif model_choice == "Support Vector Machine":
    st.markdown("<br>", unsafe_allow_html=True)
    st.subheader("Evaluation Metrics")
    st.success(":green[Accuracy: **_84.72%_**]")
elif model_choice == "Bagging Decision Tree":
    st.markdown("<br>", unsafe_allow_html=True)
    st.subheader("Evaluation Metrics")
    st.success(":green[Accuracy: **_82.65%_**]")
 elif model_choice == "Boosting Decision Tree":
    st.markdown("<br>", unsafe_allow_html=True)
    st.subheader("Evaluation Metrics")
    st.success(":green[Accuracy: **_80.48%_**]")
```

CHAPTER 7

TESTING

Testing in machine learning is a critical process that evaluates the performance and generalization ability of a trained model on unseen data. It involves several techniques and methodologies aimed at assessing the model's accuracy, robustness, and reliability. In this comprehensive explanation, we'll delve into the fundamental concepts, various testing approaches, evaluation metrics, and best practices in machine learning testing.

7.1 Levels of Testing

7.1.1 Unit Testing

Unit testing involves testing individual components or functions of the ML pipeline in isolation. Each function responsible for tasks such as data preprocessing, feature extraction, model training, and evaluation is tested independently. The goal is to ensure that these components behave as expected and produce the correct outputs given specific inputs.

Techniques:

- Test-driven development (TDD) can be employed, where tests are written before implementing the functionality.
- Mocking frameworks can simulate the behavior of external dependencies, allowing isolated testing of components.

7.1.2 Integration Testing

Integration testing verifies the interactions between different modules or components of the ML system. It focuses on testing the data flow and communication between components, ensuring that they work together seamlessly.

Techniques:

- Testing various integration points, such as data input/output, API endpoints, and model interfaces.
- Using integration testing frameworks to automate the testing process and validate the interactions between components.

7.1.3 Validation Testing

Validation testing assesses the performance of the ML model on a separate dataset that was not used during training. The goal is to evaluate the model's generalization capability and its ability to make accurate predictions on unseen data.

Techniques:

- Splitting the dataset into training, validation, and test sets, with the validation set used for hyperparameter tuning.

- Cross-validation techniques, such as k-fold cross-validation, can be employed to evaluate model performance across multiple splits of the data.

7.1.4 Functional Testing

Functional testing evaluates whether the ML system meets the specified functional requirements. It focuses on testing features such as prediction accuracy, response time, and resource consumption.

Techniques:

- Defining test cases based on functional requirements and expected behavior.
- Using validation frameworks to automate the execution of test cases and verify the system's functionality.
- Validating the correctness of predictions and assessing the system's performance against predefined criteria.

7.1.5 Non-functional Testing

Non-functional testing focuses on aspects of the ML system other than its functionality, such as performance, scalability, security, and reliability. It ensures that the system meets requirements related to speed, robustness, security, and other non-functional attributes.

Techniques:

- Performance testing measures the system's response time, throughput, and resource utilization under various load conditions.
- Security testing assesses the system's resilience against potential vulnerabilities, such as data breaches or adversarial attacks.

7.1.6 Regression Testing

Regression testing involves re-running tests to ensure that recent code changes or updates haven't introduced new bugs or regression issues. It verifies that the model's performance hasn't degraded after modifications and that it continues to meet the specified requirements.

Techniques:

- Re-running existing test cases after code changes to detect regressions or unintended side effects.
- Automating regression test suites to ensure comprehensive coverage and timely detection of issues.
- Facilitating continuous integration and deployment by ensuring that new changes do not adversely affect existing functionality.

7.1.7 User Acceptance Testing (UAT)

User Acceptance Testing (UAT) involves involving end-users or stakeholders to validate whether the ML system meets their requirements and expectations. It provides feedback on usability, usefulness, and user experience, ensuring that the system meets the needs of its intended users.

CHAPTER 8

RESULTS

8.1 Home Page

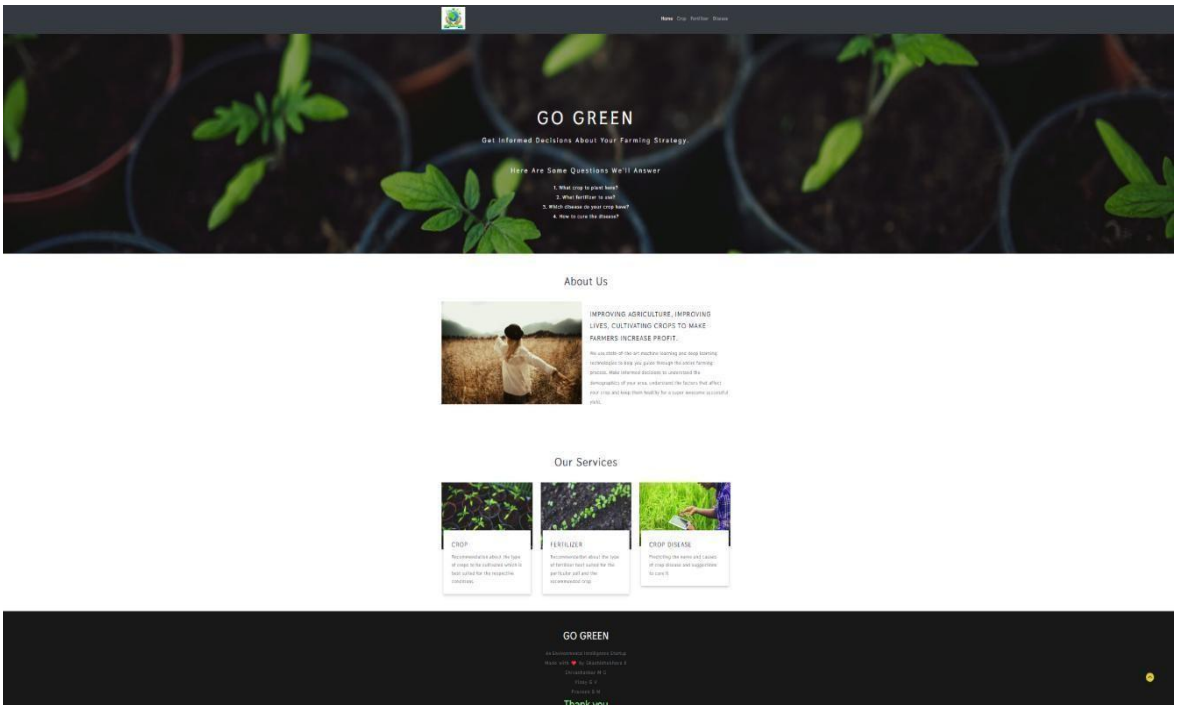


Fig.8.1 Home Page

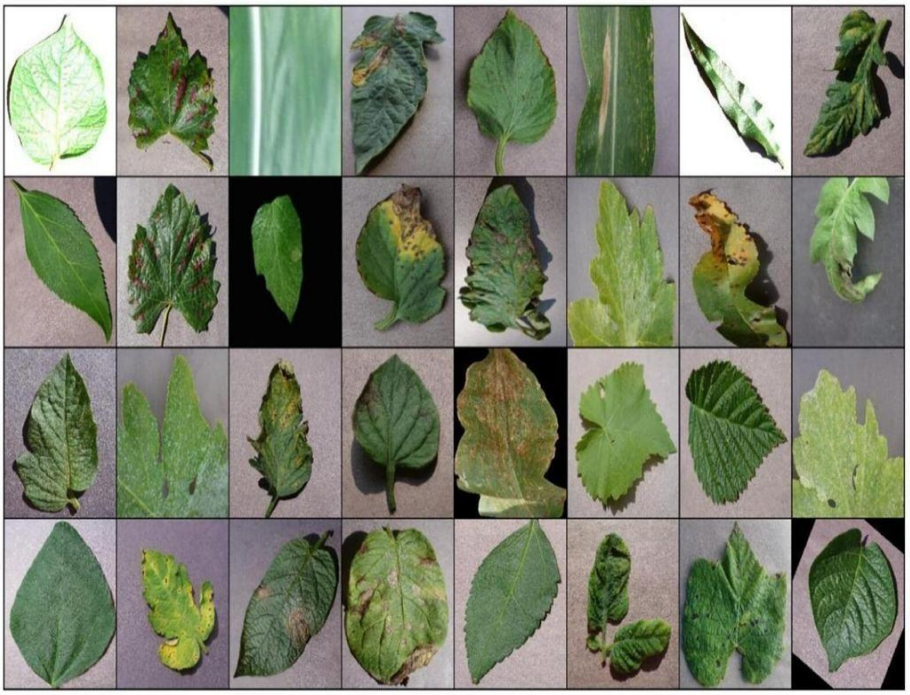


Figure 8.2 An example of Dataset page

8.2 Crop Recommendation

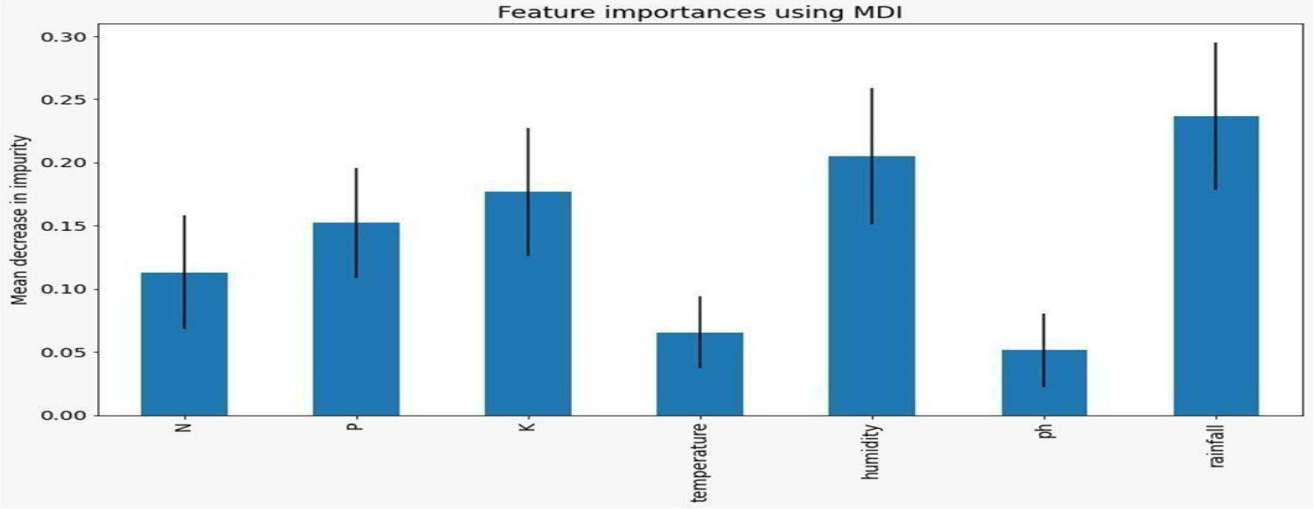


Figure 8.2.1 Feature importance for Crop Recommendation Random Forest model

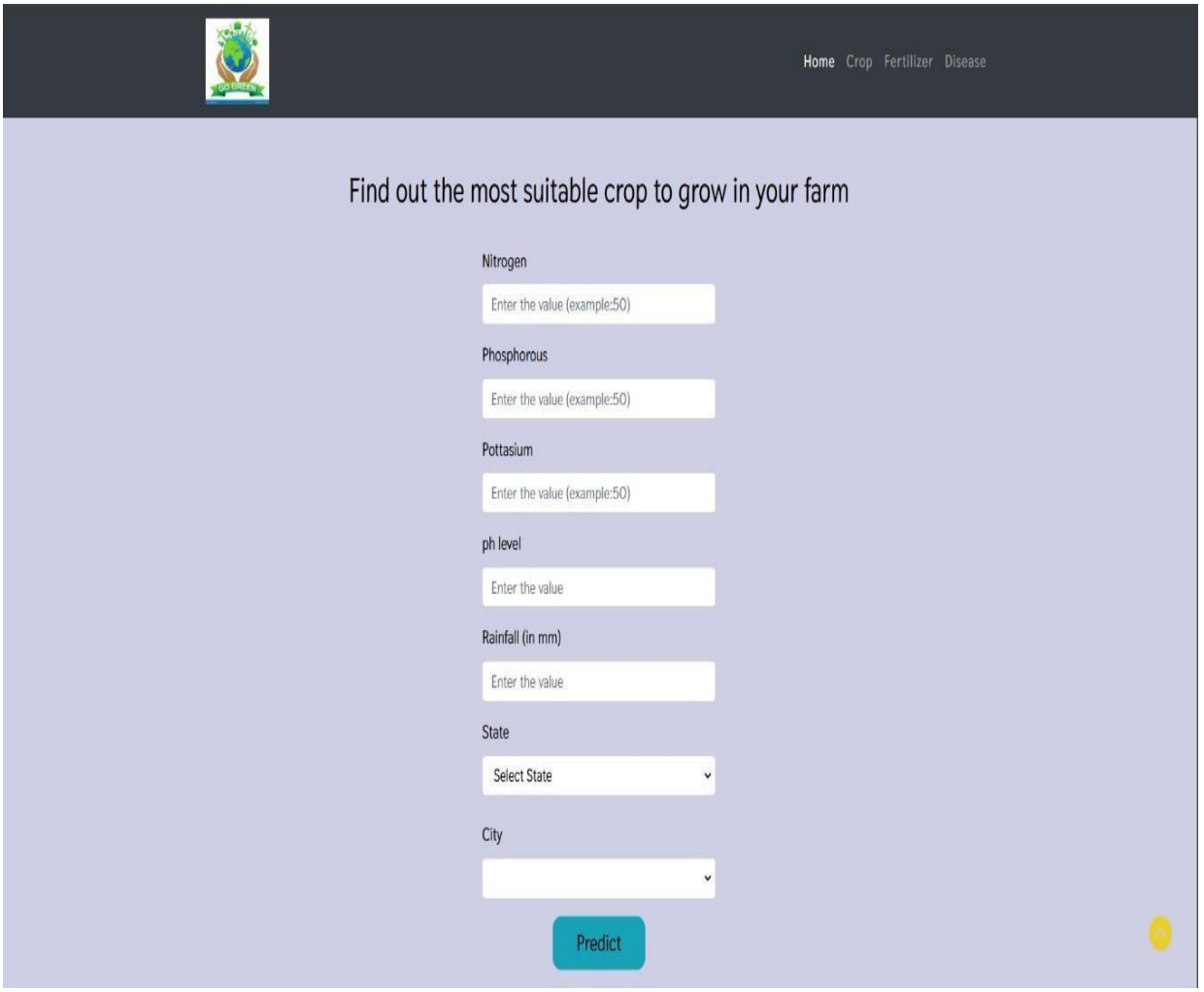


Fig.8.4 Crop Recommendation Model

Fig 8.2.2 Crop Recommendation Result

8.3 FERTILIZER RECOMMENDATION

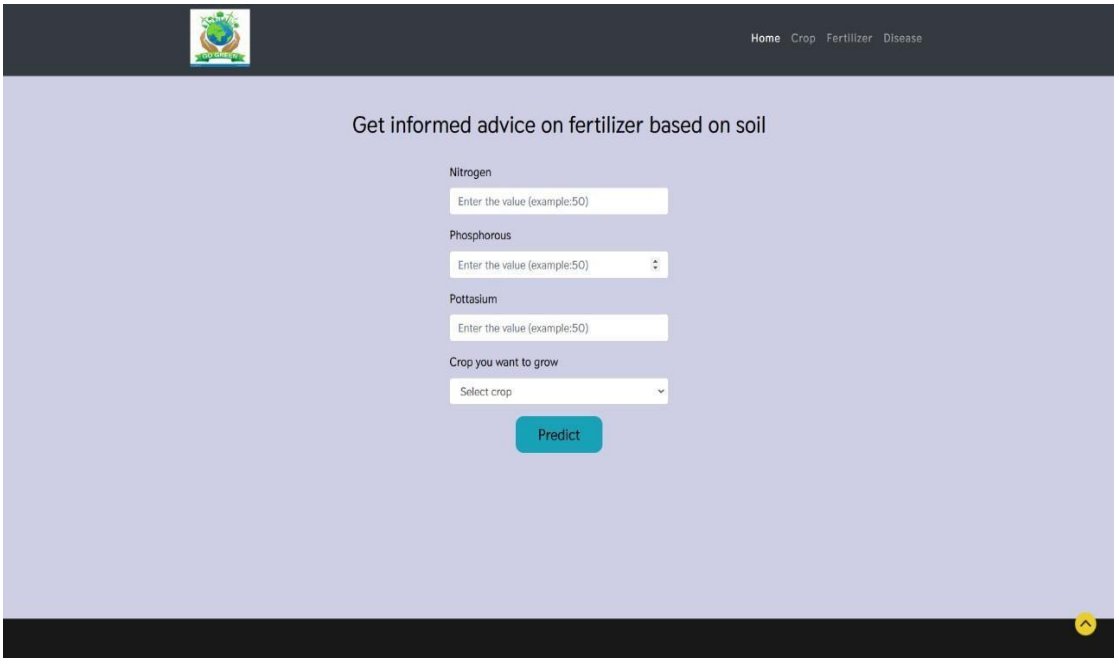


Fig.8.3.1 Fertilizer Recommendation Module

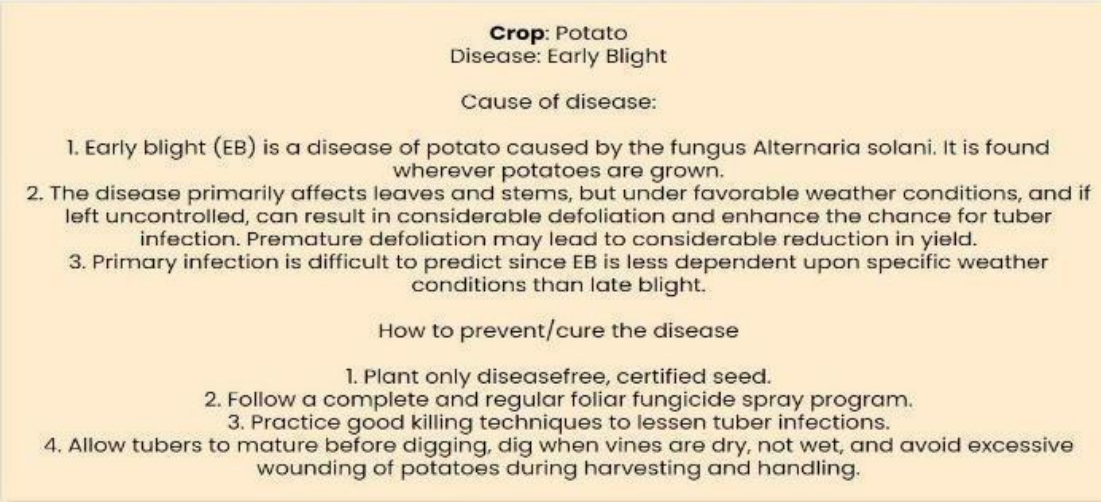
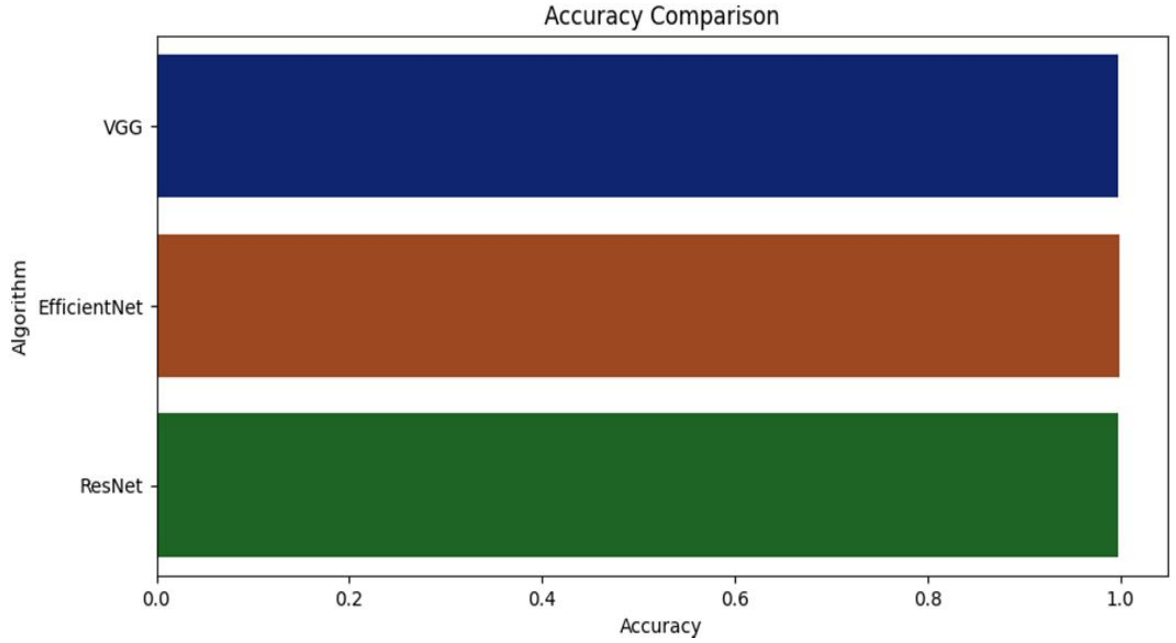


Fig 8.3.2 Disease Detected Result

8.4 DISEASE PREDICTION



8.4.1 Accuracy Comparison of Plant Disease Detection models

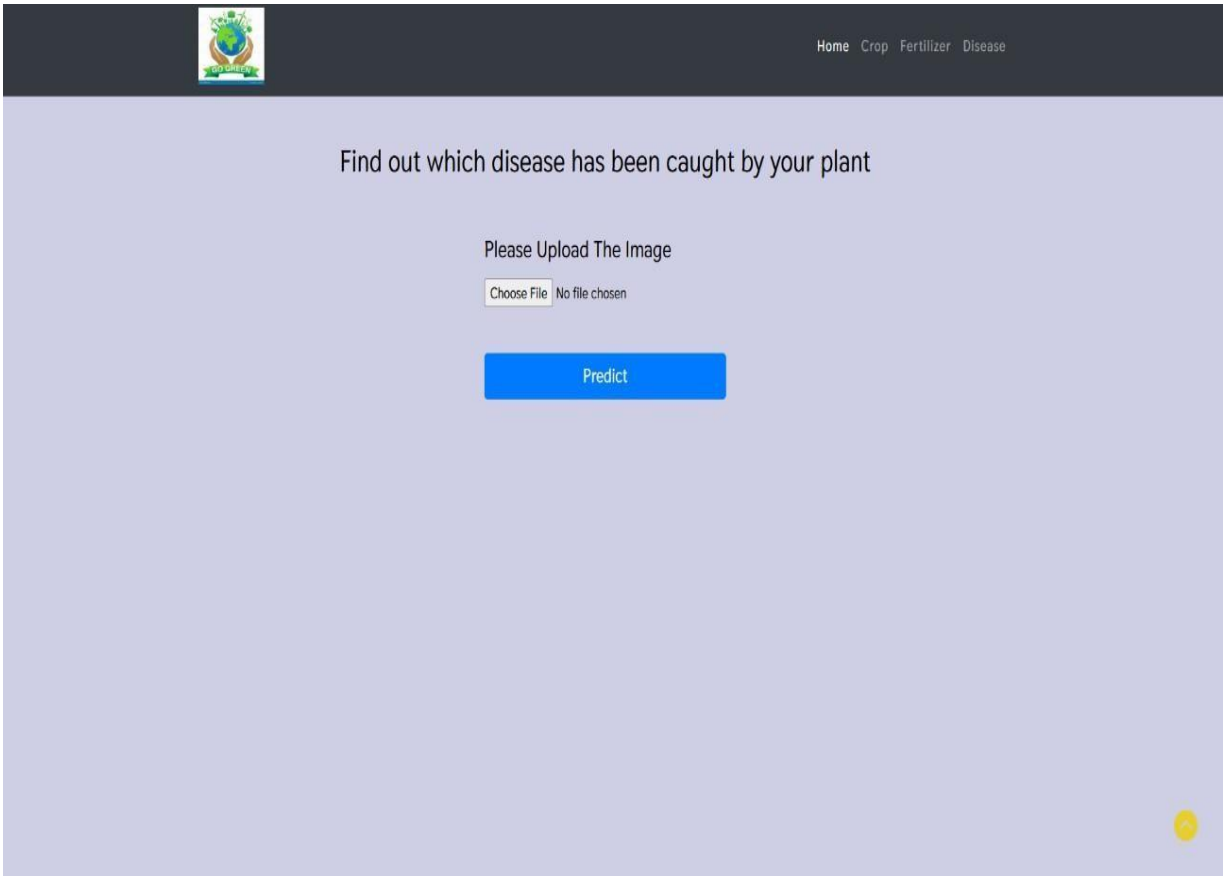


Fig.8.4.2 Disease Detection Page

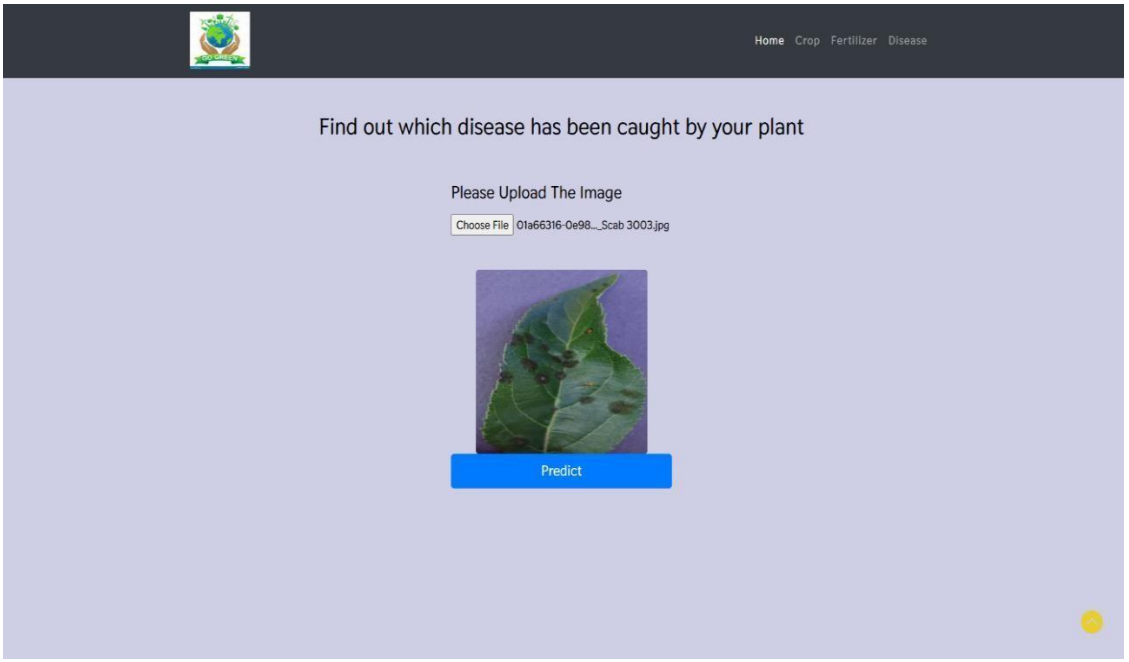


Fig.8.4.3 Disease Detection Model

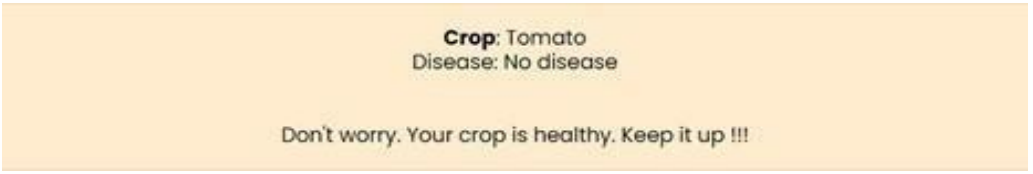


Fig 8.4.4 Disease Detected Result

8.5 Algorithm Selection

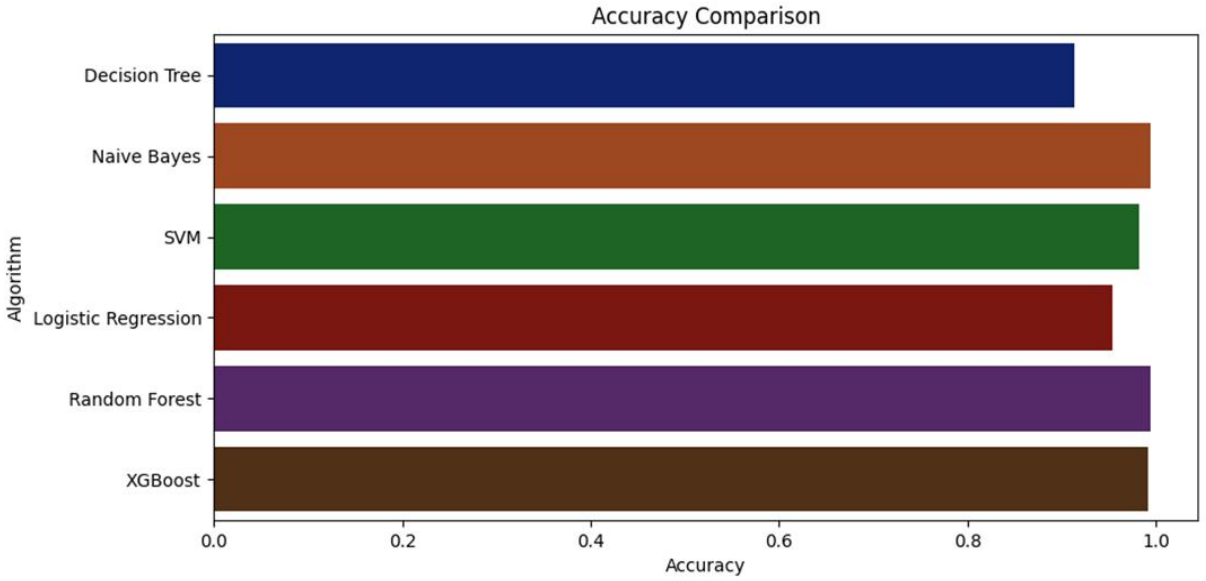


Figure 8.5 Accuracy Comparison of Algorithm Selection

1. Crop Recommendation Results
   - Results are shown in Table I and a bar graph for easy comparison.
   - XG Boost performs the best, followed by Random Forest and Naive Bayes.
2. Model Selection
   - Random Forest was chosen for the application due to its cross-validation accuracy of 0.995.
   - It provides easy interpretability of feature importance.
3. Feature Importance (Using Random Forest)
   - Precipitation is the most crucial factor in crop selection.
   - General water content and soil quality are also significant.
   - Humidity ranks second in importance.
   - Nutrient elements (Potassium - K, Phosphorus - P) also contribute to the recommendation process.
4. Understanding Feature Importance
   - The model helps identify which factors influence crop selection the most.
   - Figure 6 displays the feature importance computed using the Random Forest mode

CONCLUSION AND FUTURE ENHANCEMENTS

CONCLUSION

In this project, we propose a user-friendly web application system based on machine learning and web scraping, called the ‘Farmer’s Assistant.’ Our system offers a comprehensive suite of features designed to assist farmers in making informed decisions. These include crop recommendation using the Random Forest algorithm, fertilizer recommendation through a rule-based classification system, and crop disease detection leveraging the Efficient Net model to analyse leaf images. Users can easily input data through intuitive forms on our user interface and receive results promptly.

Additionally, our platform aims to bridge the gap between technology and agriculture by providing actionable insights based on real-time data analysis. The system is designed to handle diverse inputs, such as soil nutrient levels, crop types, and disease-prone leaf images, making it adaptable for various agricultural contexts. With features like responsive design for mobile compatibility and fast computation through an optimized backend, the Farmer’s Assistant is built for reliability and scalability.

In the future, we plan to expand the application to include features such as automatic soil testing integration, seasonal growth predictions, and personalized farming advice. Additionally, incorporating live weather data and geographical analysis will help fine-tune crop and fertilizer recommendations for specific regions. The system will also enable users to track historical recommendations and yields, providing valuable insights into long-term farming practices. Furthermore, the inclusion of a community forum within the platform will allow farmers to share their experiences, thus creating a collaborative learning environment. Future updates will also focus on enhancing the platform’s accessibility by supporting multiple languages, local farming practices, and adapting to new agricultural technologies.

FUTURE ENHANCEMENTS

While our application runs very smoothly, we have several directions in which we can improve it further. Firstly, for crop and fertilizer recommendations, we can integrate features to display the availability of recommended items on popular shopping websites, enabling users to purchase them directly through our application. Another improvement for fertilizer recommendation involves leveraging data on various brands and products tailored to specific N, P, and K values, ensuring more precise suggestions.

In the future, we aim to employ advanced machine learning algorithms and deep learning models to enhance the accuracy of recommendations, incorporating additional factors such as weather conditions, geographic data, and seasonal trends. We also intend to include real-time soil testing hardware integration to automate data collection, thereby improving the system's reliability and ease of use.

Moreover, an intuitive dashboard could be developed, providing users with insights such as nutrient trends, historical recommendations, and yield predictions. Integration with drone technology for real-time crop monitoring and soil analysis is another potential advancement. Furthermore, we aim to collaborate with agricultural experts to include pest control recommendations and organic farming solutions, ensuring holistic agricultural advice. Finally, multilingual support and personalized recommendations for specific farming practices could make the system more inclusive and adaptable for diverse farming communities worldwide.

REFERNCES

[1]. Predicting Yield of the Crop Using Machine Learning Algorithm by P. Priya, U. Muthaiah & M. Balamurugan, International Journal of Engineering Sciences & Research Technology (IJESRT), April 2023.

[2]. Machine learning approach for forecasting crop yield based on climatic parameters by S.Veenadhari, Dr.Bharat Misra &Dr.CD Singh, International Conference on Computer Communication and Informatics (ICCCI -2014), Jan, 2022.

[3]. A Survey on Crop Prediction using Machine Learning Approach by Sriram Rakshith.K, Dr.Deepak.G, Rajesh M, Sudharshan K S, Vasanth S & Harish Kumar, International Journal for Research in Applied Science & Engineering Technology (IJRASET) Volume 7, Issue IV, Apr 2020

[4]. Heuristic Prediction of Crop Yield using Machine Learning Technique by S. Pavani, Augusta Sophy Beulet P, International Journal of Engineering and Advanced Technology (IJEAT) Volume-9, December 2020 Sadal O.I.,Marwa S.M.,Wala T.A. (2012) "Linear Alkylbenzene Production from Kerosene" Seminar presented to the Department of Chemical engineering University of Khartoum.

[5]. Kale, Shivani S., and Preeti S. Patil. "A Machine Learning Approach to Predict Crop Yield and Success Rate." In 2019 IEEE Pune Section International Conference (PuneCon), pp. 1-5. IEEE, 2019. .

[6]. Kumar, Y. Jeevan Nagendra, V. Spandana, V. S. Vaishnavi, K. Neha, and V. G. R. R. Devi. "Supervised Machine learning Approach for Crop Yield Prediction in Agriculture Sector." In 2020 5th International Conference on Communication and Electronics Systems (ICCES), pp. 736-741. IEEE, 2020..

[7]. Nigam, Aruvansh, Saksham Garg, Archit Agrawal, and Parul Agrawal. "Crop yield prediction using machine learning algorithms." In 2019 Fifth International Conference on Image Information Processing (ICIIP), pp. 125-130. IEEE, 2019.