A MAJOR PROJECT

On

MACHINE LEARNING ADOPTED MULTI-CLASS CLASSIFICATION OF PLANT DISEASES WITH SPARSE AND CATEGORICAL DATA

Submitted

In partial fulfillment for the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

in

Computer Science and Engineering (Data Science)

By

Name of team members	Rollno
Bandi Deepak	21641A67D8
K. Shivani Soumya	21641A67D2
Gaje Harshitha	21641A67F6
Yara Anil Kumar	21641A67J0

Under the Guidance of

Dr. AYESHA BANU

Head of the Department of CSE (Data Science).

Department of Computer Science & Engineering (Data science)

Vaagdevi College of Engineering

(UGC Autonomous, Accredited by NAAC with “A”)

Bollikunta, Khila Warangal (Mandal), Warangal Urban – 506005(T.S)

(2021-2025)

VAAGDEVI COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (DATA SCIENCE)

(UGC Autonomous, Accredited by NBA, Accredited by NAAC with "A")

Bollikunta, Khila Warangal (Mandal), Warangal Urban –506005(T.S)

CERTIFICATE

This is to certify that the major project entitled "**MACHINE LEARNING ADOPTED MULTI-CLASS CLASSIFICATION OF PLANT DISEASES WITH SPARSE AND CATEGORICAL DATA"** is submitted by **Bandi Deepak 21641A67D8, Shivani Soumya. K 21645A67D2, Gaje Harshitha 21641A67F6, Yara Anil Kumar 21641A67J0,** in partial fulfillment of the requirements for the award of the Degree in Bachelor of Technology in Computer Science and Engineering (Data Science) during the academic year 2024-2025.

Project Guide:	**Head of the Department:**
Dr. Ayesha Banu	Dr. Ayesha Banu

External Examiner

DECLARATION

We declare that the work reported in the project entitled "**MACHINE LEARNING ADOPTED MULTI-CLASS CLASSIFICATION OF PLANT DISEASES WITH SPARSE AND CATEGORICAL DATA**" is a record of work done by us in the partial fulfillment for the award of the degree of Bachelor of Technology in Computer Science and Engineering (Data Science), VAAGDEVI COLLEGE OF ENGINEERING (Autonomous), Affiliated to JNTUH, Accredited By NBA, under the guidance of Dr. Ayesha Banu, HOD of CSE (DS). We hereby declare that this project work bears no resemblance to any other project submitted at Vaagdevi College of Engineering or any other university/college for the award of the degree.

Bandi Deepak (21641A67D8)
K. Shivani Soumya (21641A67D2)
Gaje Harshitha (21641A67F6)
Yara Anil Kumar (21641A67J0)

ACKNOWLEDGEMENT

The development of the project though it was an arduous task, it has been made by the help of many people. We are pleased to express our thanks to the people whose suggestions, comments, criticisms greatly encouraged us in betterment of the project.

We would like to express our sincere gratitude and indebtedness to my project Guide **Dr. Ayesha Banu**, HOD of CSE (DS) for her valuable suggestions and interest throughout the completion of this project.

We are also thankful to Head of the Department **Dr. Ayesha Banu**, Associate Professor, CSE (DS) for providing excellent support in completing the project successfully.

We would like to express our sincere thanks and profound gratitude to **Dr. K. Prakash**, Principal of Vaagdevi College of Engineering, for his support, guidance and encouragement in the course of our project.

We are also thankful to Project Coordinators, for their valuable suggestions, encouragement and motivations for completing this project successfully.

We are thankful to all other faculty members for their encouragement.

Finally, we would like to take this opportunity to thank our family for their support through the work. We sincerely acknowledge and thank all those who gave directly or indirectly their support in completion of this work.

Bandi Deepak (21641A67D8)

K. Shivani Soumya (21641A67D2)

Gaje Harshitha (21641A67F6)

Yara Anil Kumar (21641A67J0)

ABSTRACT

Plant disease classification is a critical aspect of modern agriculture that has evolved from traditional, manual diagnostic methods to sophisticated, automated systems utilizing machine learning. Historically, farmers and agricultural experts relied on visual inspection, expert consultations, and laboratory tests to identify plant diseases—a process that, while effective in small-scale applications, was often subjective, time-consuming, and inconsistent, leading to delayed interventions and significant crop losses. The core problem lies in the inefficiency and high cost of these traditional approaches, which struggle to cope with large-scale agricultural demands and the complex nature of disease symptoms. This scenario underscores the urgent need for an innovative system that can swiftly and accurately classify plant diseases using sparse and categorical IoT data. The proposed system leverages modern data analytics and machine learning techniques to overcome these limitations. It incorporates advanced data preprocessing methods, such as filling missing values, label encoding, and applying Synthetic Minority Oversampling Technique (SMOTE) to address class imbalance, thereby ensuring high-quality input data. The system then employs a multi-class classification pipeline using several models—Gaussian Naive Bayes, Support Vector Machines, K-Nearest Neighbors, and a novel Decision Tree Classifier—which has shown superior performance across key metrics. Performance evaluation demonstrates that the Decision Tree model achieves an accuracy of 99.07%, with precision, recall, and F1-scores consistently above 98%, validating its reliability and robustness. The significance of this research lies in its potential to revolutionize plant disease management by providing real-time, data-driven insights that enable early diagnosis and targeted pesticide recommendations. This not only enhances crop yield and minimizes economic losses but also supports sustainable farming practices by reducing excessive chemical use. In summary, by addressing the shortcomings of traditional systems—such as subjectivity, delayed response, and scalability issues—and integrating cutting-edge machine learning algorithms, the proposed system offers a transformative solution for efficient and accurate plant disease classification, promising a new era in precision agriculture.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

Overview

Agricultural biodiversity is essential for providing humans with food and raw materials and is an essential component of human civilization [1, 2]. The disease can occur when pathogenic organisms such as fungi, bacteria, and nematodes; soil PH; temperature extremes; changes in the quantity of moisture and humidity in the air; and other elements continuously harm a plant. Plant diseases can have an impact on the growth, function, and structure of plants and crops, affecting the people that rely on them. Plant diseases can have a severe impact on crop health, leading to substantial damage, reduced yields, and financial losses for farmers. The majority of farmers still use manual methods to detect and classify plant ailments because it is difficult to do so early on, and this reduces productivity. Agriculture's productivity is a significant economic factor. As a result, disease identification and classification in plants are critical in agricultural industries [3]. If proper precautions are not taken, it can have serious consequences for plants by reducing the quality, quantity, or productivity of the corresponding products or services. Automatic disease detection and classification recognize symptoms at an early stage, i.e., when they first appear on plant leaves, lowering the amount of labor necessary to monitor large farms of crops. According to [4] Plant leaf disease is a major issue in rice production, and the disease has the potential to harm the crop, resulting in a drop in products. Farmers have a difficult time detecting and classifying plant leaf diseases. The timely and precise detection of plant diseases plays a vital role in implementing effective disease management and prevention strategies.

Research Motivation

The motivation behind this research stems from the need to improve crop management and yield by rapidly and accurately diagnosing plant diseases. With the advent of IoT sensors, a vast amount of environmental and plant data is available in real time. Leveraging this data through advanced machine learning techniques offers an opportunity to detect diseases at an early stage, thereby mitigating extensive crop damage. This project is driven by the goal of enhancing diagnostic precision, reducing the time lag associated with traditional methods, and enabling proactive disease management strategies that can ultimately lead to more sustainable agricultural practices.

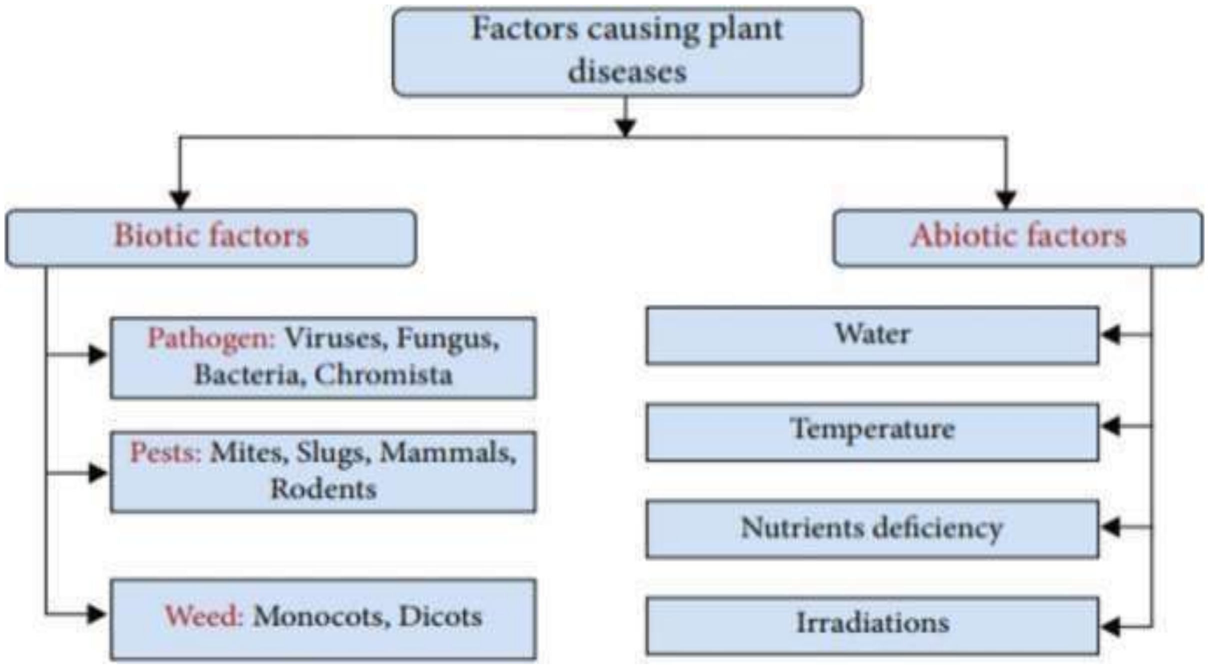

Factors causing plant diseases
Biotic factors
Abiotic factors
Pathogen: Viruses, Fungus, Bacteria, Chromista
Pests: Mites, Slugs, Mammals, Rodents
Weed: Monocots, Dicots
Water
Temperature
Nutrients deficiency
Irradiations

Fig. 1.1: Factors responsible for plant diseases [26].

Problem Definition

Plant diseases pose a significant threat to agricultural productivity and food security worldwide. Traditional diagnosis methods often rely on manual observation, expert consultations, and laboratory tests, which are time-consuming, subjective, and resource-intensive. Moreover, with the increasing scale of modern agriculture and the complexity of disease patterns, traditional approaches may not provide timely or accurate results. The core problem addressed by this project is to develop an automated, machine learning-based system capable of classifying multiple plant diseases using sparse and categorical IoT data, thereby reducing dependency on manual diagnostics and improving early detection and intervention.

Significance

The significance of this research lies in its potential to transform plant disease management. By automating the diagnosis process using machine learning, the system can provide:

- **Faster Diagnosis:** Immediate insights based on real-time data, allowing for quick remedial actions.
- **Increased Accuracy:** Reduced human error and variability through objective, data-driven analysis.

- **Cost-Effectiveness:** Lower reliance on expensive laboratory tests and expert consultations.
- **Scalability:** The ability to monitor and manage plant diseases across large agricultural areas efficiently.
- **Actionable Recommendations:** Beyond diagnosis, the system offers pesticide suggestions tailored to each disease, facilitating targeted interventions.

These improvements can significantly enhance crop yield, reduce economic losses, and contribute to more sustainable farming practices, which is crucial in the context of global food security.

Research Objective

The primary objective of this research is to design and develop a robust, machine learning-driven system for multi-class classification of plant diseases using sparse and categorical IoT data. The specific objectives include:

- **Data Integration and Preprocessing:** Developing techniques to handle sparse and heterogeneous IoT data, including missing values and categorical variables, through methods like label encoding and SMOTE for balancing the dataset.
- **Model Development:** Implementing and comparing multiple machine learning algorithms (e.g., Gaussian Naive Bayes, SVM, KNN, and Decision Tree Classifier) to determine the most effective model for disease classification.
- **System Integration:** Building a comprehensive platform that integrates data ingestion, preprocessing, model training, and prediction into an interactive GUI with user authentication.
- **Evaluation and Validation:** Conducting thorough performance evaluations using metrics like accuracy, precision, recall, and F1-score, with the goal of validating the system's effectiveness in real-world scenarios.
- **Actionable Insights:** Providing precise disease predictions along with corresponding pesticide recommendations to support timely and effective disease management strategies.

By achieving these objectives, the project aims to bridge the gap between advanced data analytics and practical agricultural applications, paving the way for innovative solutions in plant disease diagnostics and management.

Applications

Below are some key applications of the plant disease classification system, along with detailed explanations of how it can benefit the agricultural domain:

Real-Time Disease Detection and Monitoring: By leveraging IoT sensors and machine learning algorithms, the system can continuously monitor environmental conditions and plant health. This real-time detection allows farmers and agronomists to identify disease outbreaks at their earliest stages, facilitating prompt intervention and minimizing crop damage.

Precision Agriculture: The system supports precision agriculture by providing accurate, data-driven insights on plant health. By classifying diseases and suggesting targeted pesticide applications, it helps farmers optimize resource usage—reducing chemical inputs and promoting sustainable farming practices. This targeted approach not only improves crop yields but also minimizes environmental impact.

Automated Decision Support: With an integrated GUI that includes data visualization, model training, and prediction functionalities, the system acts as an automated decision support tool. It empowers users to make informed decisions based on comprehensive analysis, such as choosing the right treatment for a particular disease. This reduces dependency on expert consultations and streamlines the disease management process.

Cost Reduction and Increased Efficiency: Traditional diagnostic methods can be labor-intensive and costly. An automated, machine learning-based system significantly reduces the need for extensive laboratory testing and manual inspections. This leads to lower operational costs and more efficient use of time and resources, ultimately contributing to higher profitability for farmers.

Scalability in Large Agricultural Operations: For large-scale farming operations, monitoring vast fields manually is impractical. The system's ability to integrate data from multiple IoT sensors and process it centrally makes it scalable to cover extensive areas. This scalability ensures that even large farms can benefit from rapid disease diagnosis and timely interventions.

Data-Driven Insights and Continuous Improvement: The collection and analysis of extensive IoT data provide a valuable dataset for further research and development. Over time, this data can be used to refine predictive models, identify emerging disease trends, and improve overall agricultural practices. The system not only addresses immediate disease management challenges but also contributes to long-term improvements in crop management strategies.

1.1 EXISTING SYSTEM

In many agricultural settings, traditional plant disease diagnosis has relied on manual and expert-driven processes. The process typically involves the following steps:

1. **Field Observation:** Farmers or field workers visually inspect plants for symptoms like discoloration, spots, wilting, or other abnormalities. This initial observation is subjective and based on experience.

2. **Expert Consultation:** When symptoms are detected, farmers often consult with agricultural experts or extension officers. The diagnosis is made based on visual cues and historical knowledge of disease patterns.

3. **Laboratory Testing:** In some cases, plant samples are sent to a laboratory for detailed analysis. This may involve microbial culture, microscopic examinations, or chemical tests to confirm the suspected disease.

4. **Diagnosis and Recommendation:** Once the disease is identified, the expert recommends treatment options such as specific pesticides or cultural practices. The decision is made manually and often depends on the expertise and availability of treatment options.

5. **Manual Record Keeping:** The data about diseases and treatment outcomes are recorded manually or in simple spreadsheets. This information is used for future reference but is not integrated into an automated decision-making system.

Limitations

While the traditional system has been effective for many years, it comes with several limitations:

- **Subjectivity and Inconsistency:** Visual inspection is subjective and highly dependent on the observer’s expertise. This can lead to inconsistent diagnoses, especially in cases where symptoms are subtle or similar across diseases.

- **Delayed Response:** Laboratory testing and expert consultations take time. In fast-moving disease outbreaks, the delay can result in significant crop losses before an accurate diagnosis is made.

- **Limited Scalability:** Manual inspections and lab tests are resource-intensive and difficult to scale across large agricultural areas. This limits the ability to monitor widespread outbreaks effectively.
- **Data Fragmentation:** Record keeping is often decentralized and manual, which prevents the creation of a comprehensive and integrated dataset that could be used for further analysis or early warning systems.
- **High Cost:** Reliance on expert consultations and laboratory tests can be expensive, particularly for small-scale farmers. This increases the overall cost of disease management.

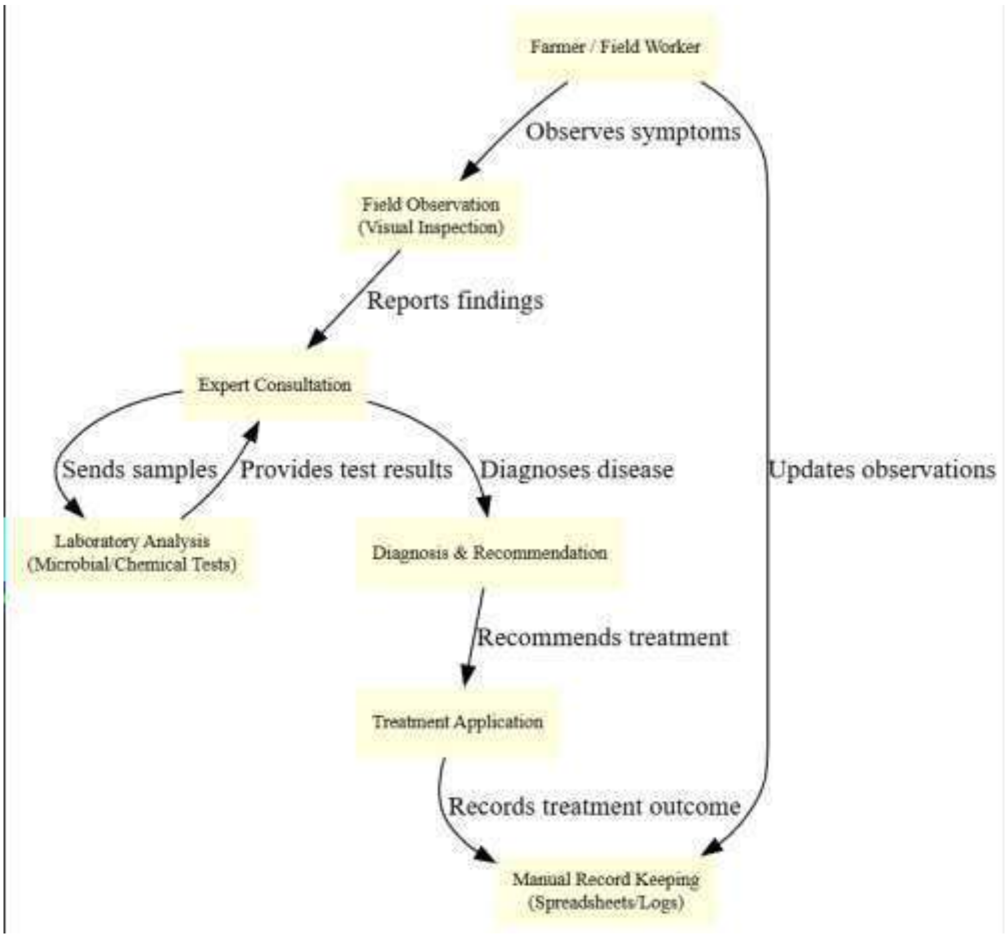


Fig. 1.1.1: Traditional system architecture.

The traditional system for plant disease diagnosis relies heavily on manual observation, expert consultation, and laboratory testing. Although this method has served agriculture for decades, its limitations—such as subjectivity, slow response times, scalability issues, fragmented data, and high costs—have led to the exploration of automated, machine learning-based systems.

1.2 PROPOSED SYSTEM

1.2.1 Overview

This project is a comprehensive Python application designed to diagnose plant diseases using machine learning. It integrates data preprocessing, visualization, machine learning, and a user-friendly interface to facilitate the diagnosis of plant diseases using IoT data. It offers a robust platform for both data scientists (Admins) and end-users, supporting the entire pipeline from data exploration to model training and real-time predictions with actionable recommendations. Here's an overview of its main components:

1. Graphical User Interface (GUI)

- **Tkinter-Based Interface:** The project uses Tkinter to create a full-screen GUI that includes background images, text widgets for logging and displaying results, and buttons for user interactions.
- **User Roles:** There are separate functionalities for Admin and User roles. Admins can upload datasets, perform exploratory data analysis (EDA), preprocess data, apply oversampling techniques (SMOTE), split data, and train multiple classifiers. Regular users have access to the prediction functionality.

2. Data Handling and Visualization

- **Dataset Upload and Display:** Users can load CSV datasets, which are then displayed in the GUI.
- **Exploratory Data Analysis (EDA):** The project provides several plotting functions (bar plots, violin plots, histograms, scatter plots, strip plots, and correlation heatmaps) to visualize different aspects of the data. This helps in understanding the distribution of features and the relationships between them.

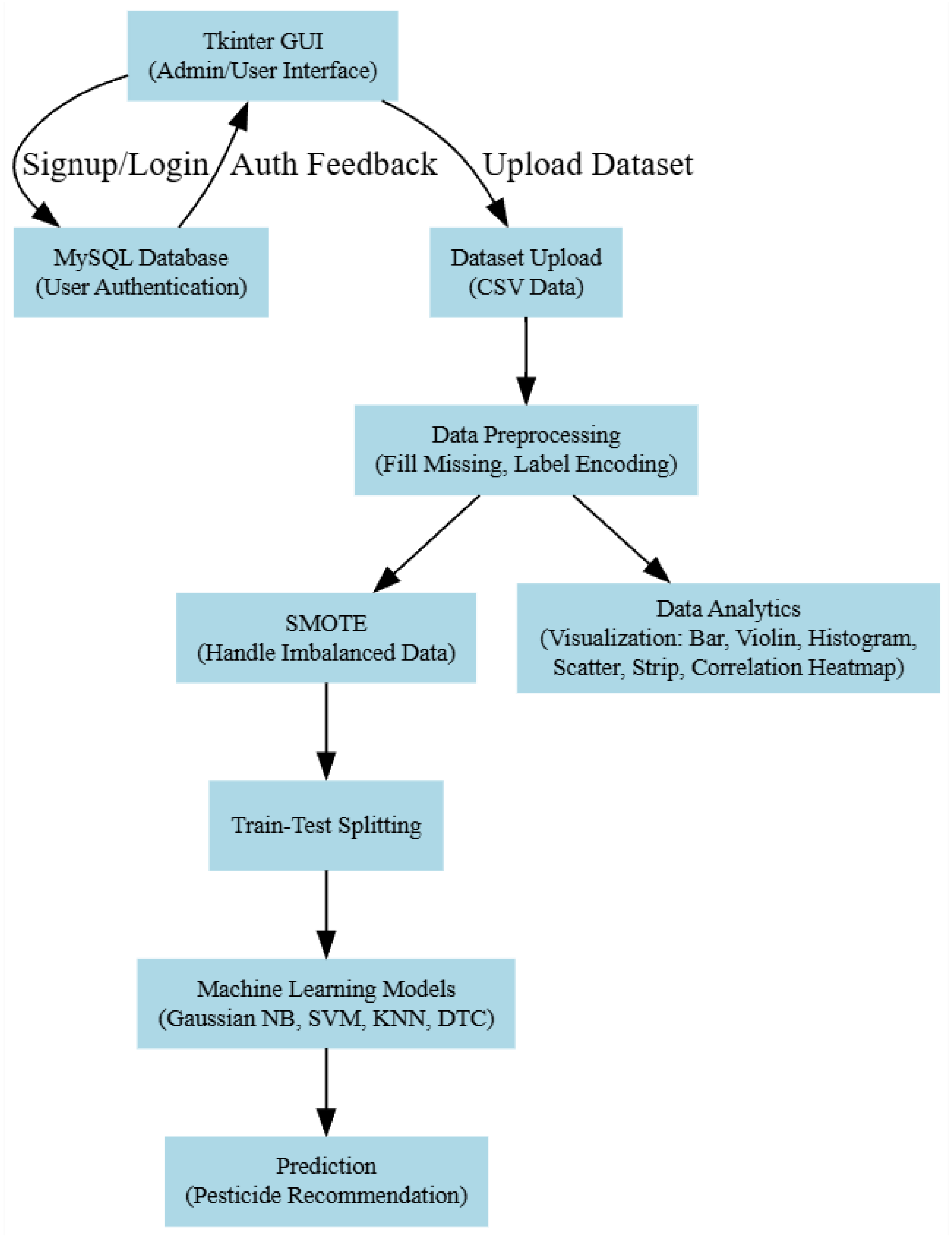


Fig. 1.2.1: System architecture of proposed plant disease classification with pesticide suggestion from categorical IoT data.

3. Data Preprocessing and Augmentation

- **Label Encoding:** Non-numeric (categorical) columns in the dataset are converted into numeric values using label encoding, making the data suitable for machine learning algorithms.

- **Handling Missing Values:** Missing data is replaced with zeros to maintain dataset consistency.
- **SMOTE for Imbalanced Data:** The project employs SMOTE (Synthetic Minority Oversampling Technique) to address class imbalance, which is visualized by comparing class distributions before and after the technique is applied.

4. Machine Learning and Model Training

- **Train-Test Splitting:** The data is split into training (80%) and testing (20%) sets.
- **Multiple Classifiers:** The project includes several machine learning models:
    - **Gaussian Naive Bayes (GNBC)**
    - **Support Vector Machine (SVC)**
    - **K-Nearest Neighbors (KNN)**
    - **Decision Tree Classifier (DTC)**

Models are either trained on-the-fly or loaded from saved files if they already exist.

- **Performance Metrics:** After training, the models are evaluated using precision, recall, F1 score, accuracy, and confusion matrices, providing insight into model performance.

5. Prediction and Recommendations

- **Test Data Prediction:** Users can upload new test data, and the trained model will predict the disease class.
- **Pesticide Recommendations:** Each predicted disease is mapped to a specific pesticide recommendation based on predefined lists.

6. Database Integration for User Authentication

- **MySQL Database Connection:** The project connects to a local MySQL database to manage user authentication.
- **Signup and Login Functionality:** Separate signup and login functions are provided for Admins and Users. After login, different sets of functionalities are made available according to the user's role.

1.2.2 Data Preprocessing

The data preprocessing module in this project is designed to prepare the raw dataset for machine learning tasks by ensuring that the data is clean, consistent, and in a numerical format that the algorithms can understand. Here are the key steps involved:

1. Handling Missing Values

- **Imputation:** In the DatasetPreprocessing() function, missing values are replaced with zeros using fillna(0). This step ensures there are no NaN values that could disrupt the machine learning process.

2. Label Encoding

- **Conversion of Categorical Data:** Since many machine learning algorithms require numeric input, any column with an object data type (typically containing categorical or text data) is transformed using the LabelEncoder from scikit-learn.
- **Process:**
    - The code iterates over each column in the dataset.
    - For columns of type 'object', LabelEncoder is applied to convert the string values into numeric labels.
    - This conversion allows the algorithms to process these features effectively.

4. Feature and Target Separation

- **Defining Inputs and Outputs:** After encoding, the dataset is split into:
    - **Features (X):** All columns except the target variable 'Label'.
    - **Target (Y):** The 'Label' column, which represents the disease categories.
- **Extracting Unique Labels:** The unique disease labels and their counts are also extracted, which can be useful for later analysis and evaluation.

1.2.3 EDA

The project’s data analytics module leverages several types of visualizations to help users understand the underlying structure and distribution of the data. Each visualization is designed to reveal different aspects of the dataset, particularly how various numerical features relate to the disease labels. Below is a detailed explanation of each visualization function:

Bar Plot: The bar plot is used to display the mean value of a specific numerical feature for each disease label. By grouping the data based on disease categories and calculating the average for features such as temperature or leaf spot size, the bar plot provides an immediate visual comparison between different classes. This visualization can reveal trends or discrepancies—for instance, a higher average value in one group may indicate a strong association between that feature and a particular disease.

Violin Plot: Unlike the bar plot, the violin plot offers a deeper insight into the data distribution for each disease label. It combines elements of a box plot with a kernel density plot, displaying not only summary statistics like the median and quartiles but also the full distribution of the data. The shape and width of each “violin” illustrate the variability and skewness of the feature across the labels, which helps in understanding whether some diseases are characterized by more consistent measurements or by a wider spread of values.

Histogram: Histograms are employed to visualize the frequency distribution of a numerical feature across disease labels. By overlaying histograms for each class, this visualization helps in understanding how often different values occur. It is particularly useful for identifying patterns such as normal distributions, skewness, or multiple peaks, which can signal that certain ranges of the feature are more prevalent in one disease compared to another.

Scatter Plot: The scatter plot examines the relationship between two numerical features by plotting them against each other. Each data point is color-coded according to its disease label, which makes it easier to identify clusters or trends within the data. This visualization is essential for uncovering correlations between variables and determining whether the features selected are effective in distinguishing between the different disease classes.

Strip Plot: The strip plot is another visualization that focuses on individual data points for a numerical feature, grouped by disease label. By adding a jitter effect, the strip plot avoids overlap and provides a clear view of the data’s spread within each category. This granular display is especially helpful for identifying outliers and understanding the overall distribution of data points in each group.

Correlation Heatmap: Finally, the correlation heatmap visualizes the pairwise correlations between all numerical features. Using a color gradient to represent correlation strength—from -1 (strong negative) to 1 (strong positive)—this heatmap makes it easy to identify relationships between variables. Recognizing these correlations is crucial, as it helps in detecting redundancy or multicollinearity among features, thereby guiding decisions in feature selection and engineering.

Together, these visualizations form a robust data analytics module that provides a comprehensive view of the dataset. They enable users to explore data distributions, identify key relationships, and uncover underlying patterns, all of which are essential steps in preparing the data for effective machine learning model training for plant disease detection.

1.2.4 SMOTE Algorithm

The SMOTE operation in the project is designed to tackle the issue of class imbalance in the dataset. In many real-world datasets, especially in classification problems like plant disease detection, some classes (diseases) may have far fewer examples than others. This imbalance can lead to biased machine learning models that perform poorly on the minority classes.

How SMOTE Works in the Project:

1. **Preprocessing:** The dataset is first preprocessed to fill in missing values and convert categorical features into numerical values. The features (X) and target labels (Y) are then separated.

2. **Applying SMOTE:** SMOTE (Synthetic Minority Oversampling Technique) is applied to the features and labels. This technique works by generating synthetic examples of the minority classes rather than simply duplicating existing ones. It does this by interpolating between existing minority class examples, effectively creating new, slightly varied samples. This leads to a more balanced class distribution, which in turn helps the classifiers to learn decision boundaries that better generalize across all classes.

3. **Visualization of Results:** After applying SMOTE, the code visualizes the class distribution before and after the oversampling process using bar plots. This side-by-side comparison helps in confirming that the minority classes have been adequately balanced.

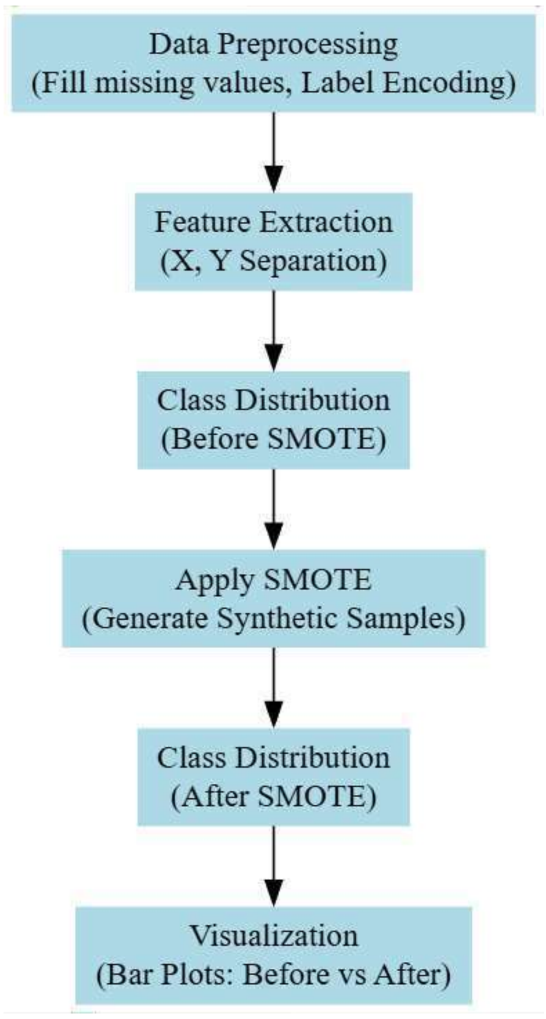


Fig. 1.2.2: Working of SMOTE algorithm.

1.2.5 Model Building and Training

1.2.5.1 Gaussian Naive Bayes (GNBC)

Gaussian Naive Bayes is a probabilistic classifier that applies Bayes' theorem with the "naive" assumption that features are conditionally independent given the class label. For continuous data, it assumes that the features follow a Gaussian (normal) distribution. The model calculates the likelihood of each feature given a class, multiplies these probabilities together (along with the prior probability of the class), and predicts the class with the highest posterior probability.

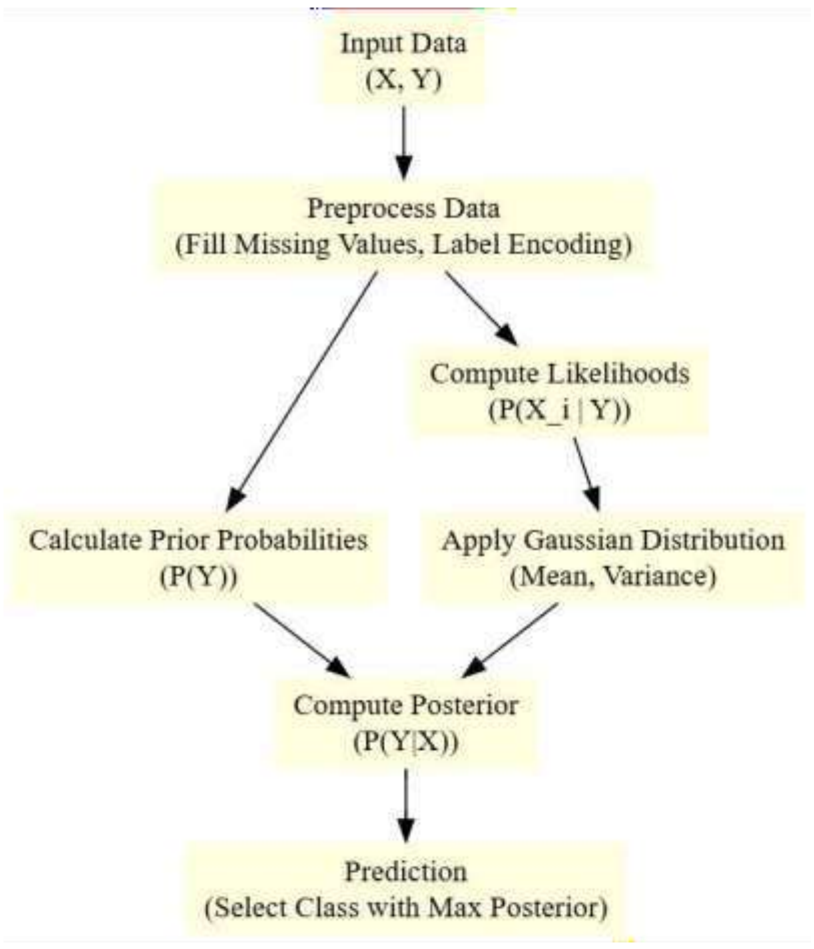

Input Data
(X, Y)
Preprocess Data
(Fill Missing Values, Label Encoding)
Compute Likelihoods
(P(X_i | Y))
Calculate Prior Probabilities
(P(Y))
Apply Gaussian Distribution
(Mean, Variance)
Compute Posterior
(P(Y|X))
Prediction
(Select Class with Max Posterior)

Fig. 1.2.3: Working of GNBC model.

Key Points:

- **Assumption of Independence:** It simplifies the computation by assuming that each feature contributes independently.
- **Gaussian Distribution:** Suitable when the features are continuous and approximately normally distributed.
- **Fast and Simple:** Often used as a baseline due to its simplicity and computational efficiency.

1.2.5.2 Support Vector Machine

The SVM classifier aims to find the optimal hyperplane that separates classes in the feature space. In its simplest (linear) form, SVM finds the hyperplane with the maximum margin between classes. For non-linearly separable data, kernel functions are used to transform the data into a higher-dimensional space where a linear separation is possible.

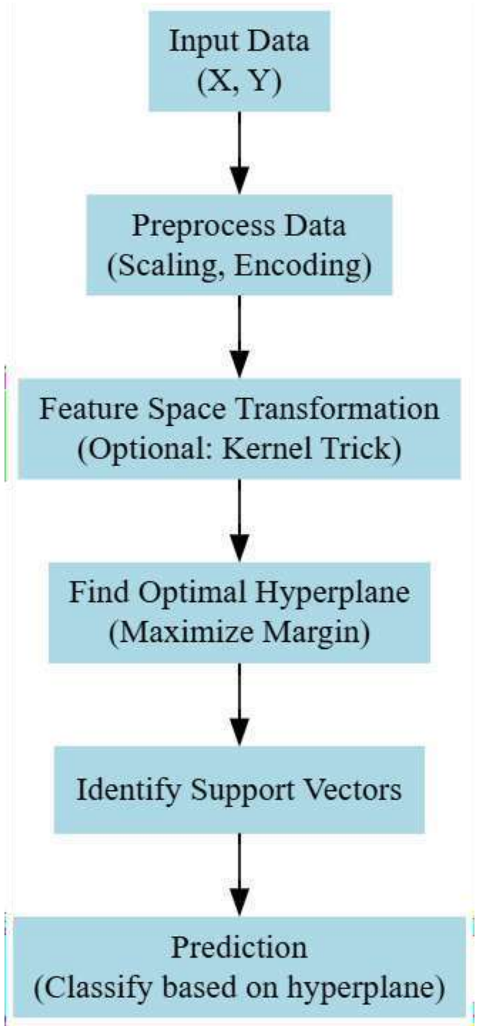


Fig. 1.2.4: Working of SVM classifier.

Key Points:

- **Margin Maximization:** The model maximizes the distance between the hyperplane and the closest data points from each class (support vectors).
- **Kernel Trick:** Enables non-linear classification by mapping data to higher dimensions.
- **Robust to Overfitting:** Particularly effective in high-dimensional spaces.

1.2.5.3 KNN Classifier

K-Nearest Neighbors is an instance-based, non-parametric algorithm. It works by storing all available cases and classifying new cases based on a similarity measure (e.g., Euclidean distance). In this project, the KNN model is configured with k=5k = 5k=5, meaning that for each test instance, the five closest neighbors are considered, and the class is predicted based on the majority vote among these neighbors.

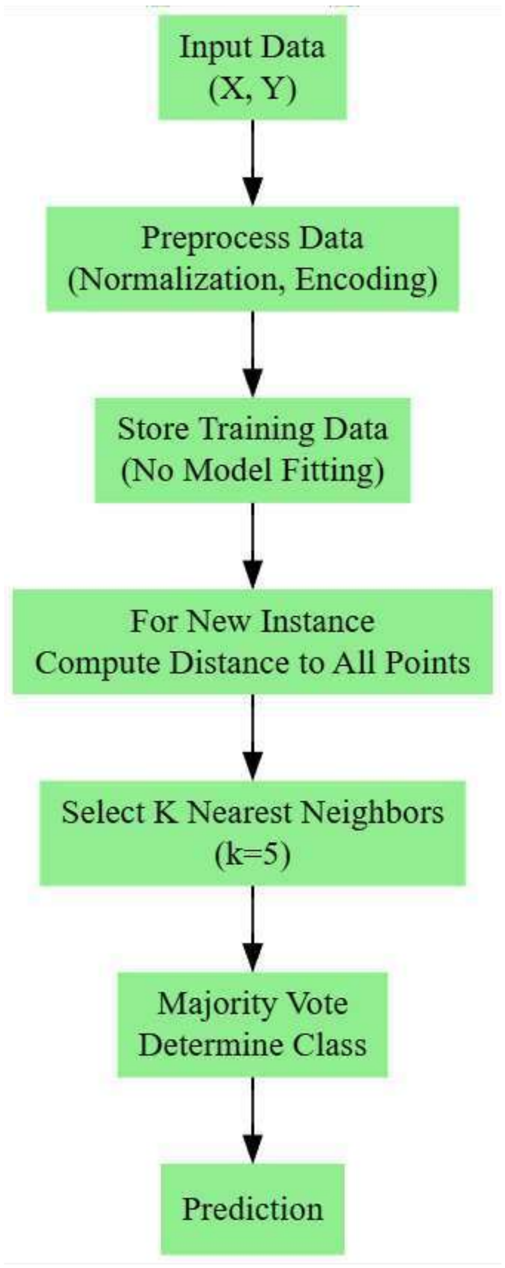


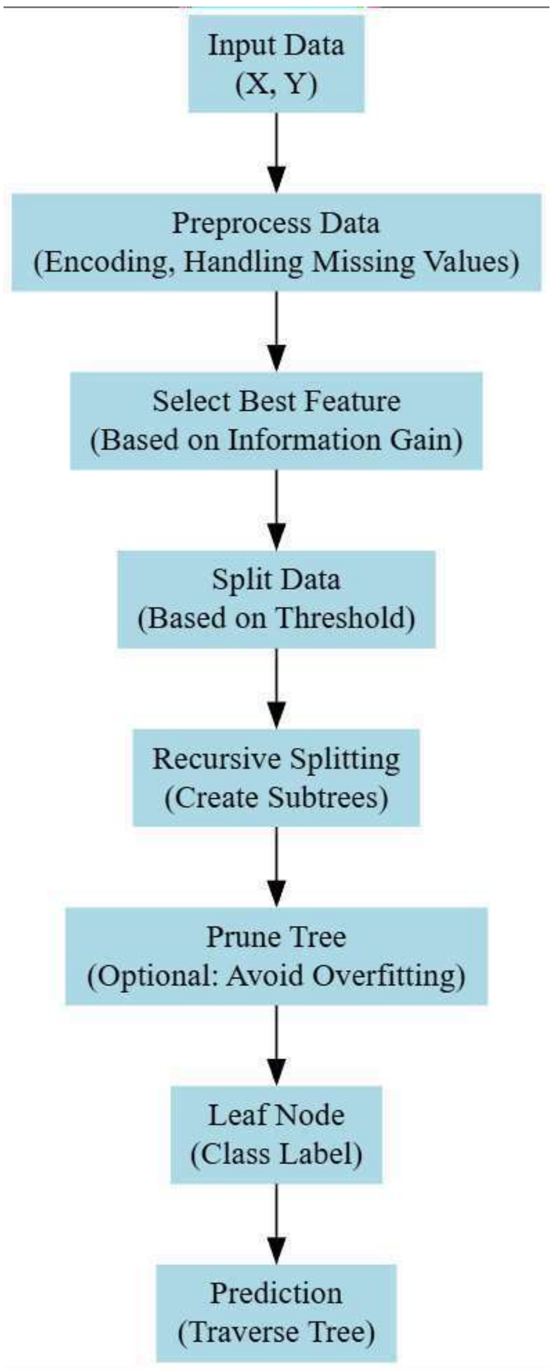
Fig. 1.2.5: Working flow of KNN classification.

Key Points:

- **Lazy Learning:** No explicit training phase; the model simply stores the data.
- **Distance-Based:** Classification is based on measuring the distance between data points.
- **Simple and Intuitive:** Easy to understand and implement but can be computationally expensive at prediction time.

1.2.5.4 DTC model

A Decision Tree is a flowchart-like structure where internal nodes represent tests on features, branches represent the outcome of these tests, and leaf nodes represent class labels. The tree is constructed by recursively splitting the dataset based on feature values that provide the maximum information gain or the best reduction in impurity (e.g., Gini impurity or entropy).



Input Data (X, Y)
Preprocess Data (Encoding, Handling Missing Values)
Select Best Feature (Based on Information Gain)
Split Data (Based on Threshold)
Recursive Splitting (Create Subtrees)
Prune Tree (Optional: Avoid Overfitting)
Leaf Node (Class Label)
Prediction (Traverse Tree)

Fig. 1.2.6: DTC model workflow.

Key Points:

- **Interpretable Model:** The decision-making process is easy to visualize and interpret.
- **Recursive Splitting:** The tree divides the dataset into smaller subsets based on the most informative features.
- **Prone to Overfitting:** Without proper pruning, decision trees can create overly complex trees that do not generalize well.

1.2.5.5 Comparative Discussion

After training multiple models such as GNB classifier, SVM classifier, KNN classifier, and DTC model, a comprehensive evaluation of performance metrics such as accuracy, precision, recall, and F1-score revealed that the DTC model achieved the best overall performance. The superior performance of the DTC model can be attributed to its ability to capture complex, non-linear relationships inherent in the plant disease dataset. Unlike probabilistic models like GNB classifier, which assume independence among features, the DTC approach effectively identifies and utilizes the most informative features through recursive splitting. This inherent feature selection mechanism allows the model to handle the diversity of sensor data and the subtle variations in plant disease symptoms more adeptly.

Additionally, the interpretability of the DTC model provides valuable insights into the decision-making process. The tree structure clearly outlines the sequence of feature-based decisions that lead to a final classification, making it easier to understand which factors most significantly contribute to the diagnosis. This transparency not only aids in model validation but also facilitates further refinement of feature engineering strategies. Moreover, while models like SVM and KNN offer robust performance in many contexts, they may struggle with high-dimensional, sparse, and categorical data unless extensively tuned or transformed. The DTC model, on the other hand, naturally accommodates such complexities without the need for extensive preprocessing beyond the standard data normalization and encoding steps.

In summary, the Decision Tree Classifier emerged as the best-performing model in this project due to its effective handling of non-linear relationships, automatic feature selection, and interpretability. Its success underscores the importance of choosing a model that aligns well with the data's characteristics, ultimately contributing to more reliable and accurate plant disease classification outcomes.

1.2.6 Database Integration

The database integration in this project is centered around managing user authentication for different roles, specifically Admin and User. This integration is implemented using a MySQL database, and the Python module pymysql is employed to establish a connection between the application and the database. Overall, the database integration in this project effectively supports user management by:

- Establishing a consistent connection to a local MySQL database.
- Enabling secure user signup by inserting user credentials into the database.
- Facilitating secure login by verifying credentials against the stored data.
- Providing tailored access based on user roles, thereby ensuring that administrative and user functionalities remain segregated.

This integration is essential for maintaining the integrity and security of the application while allowing it to scale and manage multiple users with different levels of access.

Database Connection

At the core of the integration is a simple connection function that sets up communication with a local MySQL database named *sparse_db*. The function creates and returns a connection object using the provided credentials (host, username, password, and database name). This connection is used by various parts of the application to execute SQL queries. By encapsulating the connection logic in a dedicated function, the project ensures that any interaction with the database remains consistent and manageable.

User Authentication

The database integration plays a crucial role in the signup and login functionalities of the project. During the signup process, when a new user (either Admin or User) provides their credentials, the application inserts a new record into the users table in the database. This table stores the username, password, and role, thereby allowing the system to differentiate between various types of users.

For the login process, the application queries the users table to verify the provided credentials against stored records. If a matching record is found, the user is granted access and is presented with functionalities tailored to their role. This ensures that sensitive administrative tasks are only accessible to authorized users, while regular users are provided with a restricted set of features such as prediction functionalities.

Error Handling and User Feedback

Both the signup and login operations incorporate error handling to manage potential issues such as database errors or incorrect input. For example, if an error occurs during the insertion of a new user or if the credentials do not match any records during login, the application displays appropriate error messages. This feedback mechanism not only enhances the user.

1.3 SOFTWARE REQUIREMENTS

Python is a high-level, interpreted programming language known for its simplicity and readability, which makes it a popular choice for beginners as well as experienced developers. Key features of Python include its dynamic typing, automatic memory management, and a rich standard library that supports a wide range of applications from web development to data science and machine learning. Its object-oriented approach and support for multiple programming paradigms allow developers to write clear, maintainable code. Python's extensive ecosystem of third-party packages further enhances its capabilities, enabling rapid development and prototyping across diverse fields.

Installation

First, download the appropriate installer from the official Python website (https://www.python.org/downloads/release/python-376/). For Windows users, run the executable installer and ensure to check the "Add Python to PATH" option during installation; for macOS and Linux, follow the respective package installation commands or use a package manager like Homebrew or apt-get. After installation, verify the setup by running python --version or python3 --version in your terminal or command prompt, which should display "Python 3.7.6." This version-specific installation supports all major functionalities and libraries compatible with Python 3.7.6, making it an excellent foundation for developing robust applications in areas such as data analysis, machine learning, and GUI development.

1.3.1 Python Packages

The project requires a robust set of software libraries and tools that work together to build an integrated system for plant disease classification. Below is an explanation of the key software requirements and the packages used:

- **Python:** The project is implemented in Python, which is chosen for its extensive ecosystem of libraries and its strong support for data analysis, machine learning, and GUI development.
- **Tkinter:** Used to build the graphical user interface (GUI) of the application. It handles tasks such as user authentication, data upload, and displaying results, making the system accessible to both admins and end-users.

- **PIL (Pillow):** Utilized for image processing, particularly for handling background images and other graphical elements within the GUI, thereby enhancing the visual appeal of the application.
- **Matplotlib & Seaborn:** These libraries are employed for data visualization. Matplotlib is used for creating standard plots, while Seaborn adds an extra layer of sophistication for statistical visualizations such as bar plots, violin plots, histograms, scatter plots, strip plots, and correlation heat maps.
- **Pandas & NumPy:** Essential for data manipulation and analysis. Pandas is used to load, preprocess, and analyze the CSV dataset, while NumPy supports numerical operations and data handling, which are crucial for processing large volumes of IoT data.
- **Scikit-learn (sklearn):** Provides the machine learning framework used in the project. It includes tools for model training, evaluation, train-test splitting, and data preprocessing (like label encoding). Models such as Gaussian Naive Bayes, SVM, KNN, and Decision Tree Classifier are implemented using scikit-learn.
- **Imbalanced-learn (imblearn):** Specifically used for implementing the SMOTE (Synthetic Minority Oversampling Technique) algorithm, which helps in addressing class imbalance in the dataset by generating synthetic samples for under-represented classes.
- **Joblib:** Utilized for saving and loading trained machine learning models. This ensures that once a model is trained, it can be stored and reused without retraining, thereby improving efficiency.
- **PyMySQL:** This package provides a means to connect to a MySQL database for handling user authentication. It facilitates operations such as user signup, login, and data storage, ensuring secure and persistent management of user credentials.

Each of these packages plays a crucial role in ensuring that the system is robust, scalable, and efficient—from data ingestion and preprocessing to model training, visualization, and deployment. The combination of these tools enables the creation of an integrated, user-friendly application for real-time plant disease classification and management.

1.3.2 FUNCTIONAL REQUIREMENTS

The functional requirements for the plant disease classification project define the specific actions that the system must be able to perform. These include:

- **User Management:**
  - **Signup and Login:** The system must allow both Admins and regular Users to create accounts and log in securely.
  - **Role-Based Access Control:** Once authenticated, the system should provide different functionalities based on the user role (Admin or User).
- **Dataset Handling:**
  - **Dataset Upload:** The system must allow Admins to upload a CSV file containing IoT data and plant measurements.
  - **Data Display:** After uploading, the system should display the dataset content (or a summary) in the GUI for verification.
- **Data Preprocessing:**
  - **Missing Value Handling:** The system must automatically replace or fill missing values (e.g., using fillna(0)).
  - **Label Encoding:** It should convert categorical data to numeric form using label encoding.
  - **Normalization:** If required, perform normalization on the dataset to standardize the features.
  - **SMOTE Application:** The system must apply the SMOTE algorithm to balance class distribution in the dataset.
- **Data Visualization and Analytics:**
  - **Exploratory Data Analysis (EDA):** The system should provide visualizations such as bar plots, violin plots, histograms, scatter plots, strip plots, and correlation

    heat maps to help understand the data distribution and relationships between features.

    - **Graphical Output:** Visual outputs must be rendered on the GUI for easy interpretation.

- **Model Training and Evaluation:**
    - **Train-Test Splitting:** The system must split the preprocessed data into training and testing sets (e.g., using an 80:20 ratio).
    - **Model Training:** The system should support training multiple machine learning models (Gaussian Naive Bayes, SVM, KNN, and Decision Tree Classifier).
    - **Model Saving/Loading:** It must store trained models using joblib, enabling reuse without retraining.
    - **Performance Metrics:** After training, the system should compute and display metrics like accuracy, precision, recall, and F1-score, along with confusion matrices for model evaluation.
- **Prediction and Recommendation:**
    - **Test Data Upload:** Regular Users must be able to upload new test datasets for prediction.
    - **Prediction Execution:** The system should process the test data and use the trained model to predict plant diseases.
    - **Pesticide Recommendation:** Based on the predicted disease label, the system should provide corresponding pesticide suggestions.
- **User Interface:**
    - **Interactive GUI:** The entire system must be accessible through a user-friendly Tkinter-based graphical interface.
    - **Feedback and Logs:** The GUI should display logs, status messages, and results at every stage, ensuring transparency and ease of use.

1.4 HARDWARE REQUIREMENTS

Python 3.7.6 can run efficiently on most modern systems with minimal hardware requirements. However, meeting the recommended specifications ensures better performance, especially for developers handling large-scale applications or computationally intensive tasks. By ensuring compatibility with hardware and operating system, can leverage the full potential of Python 3.7.6.

Processor (CPU) Requirements: Python 3.7.6 is a lightweight programming language that can run on various processors, making it highly versatile. However, for optimal performance, the following processor specifications are recommended:

- **Minimum Requirement**: 1 GHz single-core processor.
- **Recommended**: Dual-core or quad-core processors with a clock speed of 2 GHz or higher. Using a multi-core processor allows Python applications, particularly those involving multithreading or multiprocessing, to execute more efficiently.

Memory (RAM) Requirements: Python 3.7.6 does not demand excessive memory but requires adequate RAM for smooth performance, particularly for running resource-intensive applications such as data processing, machine learning, or web development.

- **Minimum Requirement**: 512 MB of RAM.
- **Recommended**: 4 GB or higher for general usage. For data-intensive operations, 8 GB or more is advisable.

Insufficient RAM can cause delays or crashes when handling large datasets or executing computationally heavy programs.

Storage Requirements: Python 3.7.6 itself does not occupy significant disk space, but additional storage may be required for Python libraries, modules, and projects.

- **Minimum Requirement**: 200 MB of free disk space for installation.
- **Recommended**: At least 1 GB of free disk space to accommodate libraries and dependencies.

Developers using Python for large-scale projects or data science should allocate more storage to manage virtual environments, datasets, and frameworks like TensorFlow or PyTorch.

Compatibility with Operating Systems: Python 3.7.6 is compatible with most operating systems but requires hardware that supports the respective OS. Below are general requirements for supported operating systems:

- **Windows**: 32-bit and 64-bit systems, Windows 7 or later.
- **macOS**: macOS 10.9 or later.
- **Linux**: Supports a wide range of distributions, including Ubuntu, CentOS, and Fedora.

The hardware specifications for the OS directly impact Python's performance, particularly for modern software development.

CHAPTER 2: LITERATURE SURVEY

This section offers an extensive overview of prior works in the context of this research, encompassing machine learning models with the goal of identifying their strengths, limitations, and research gaps, thus forming the basis for the proposed method. An extensive review by Li et al. [5] presents the recent research progress in using deep learning technology for crop leaf disease identification. The authors discuss the current trends, challenges, and unresolved issues pertaining to plant leaf disease detection using deep learning and advanced imaging techniques are discussed, with the aim of providing a valuable resource for researchers in the field.

Deep learning techniques were initially applied to plant image recognition with a focus on analyzing leaf vein patterns. A notable study employed a CNN architecture with 3–6 layers to successfully classify three leguminous plant species: white bean, red bean, and soybean [6]. The CNN model extracted and analyzed intricate features of leaf vein structures unique to each species, capturing low-level features such as edges and textures and progressing to more complex patterns. This hierarchical feature extraction enabled effective species differentiation based on leaf vein characteristics. The study demonstrated the potential of CNNs in plant species classification, highlighting the effectiveness of deep learning in capturing subtle biological variations for accurate identification, and laying the groundwork for further applications in plant image recognition.

Subsequently, Mohanty et al. [7] employed a classifier model to accurately identify 14 different crop species and 26 crop diseases, achieving an impressive accuracy of 99.35%. Building upon this work, Kawasaki et al. [8] proposed a CNN-based system specifically designed for the precise recognition of diseases affecting cucumber leaves, achieving a commendable accuracy rate of 94.9%. In an investigation by Ma et al. [9], a deep CNN was employed for the identification and recognition of four diseases in cucumber plants, including downy mildew, anthracnose, powdery mildew, and target leaf spots, based on their distinct symptoms. This study reported a recognition accuracy of 93.4%.

Impressive strides have been made in plant leaf disease recognition; however, the availability of diverse datasets remains limited. CNN training necessitates extensive and diverse datasets, which are currently scarce in the field of plant leaf disease recognition. Consequently, transfer learning has emerged as a viable solution to enhance the robustness of CNN classifiers. By adapting pre-trained CNNs and retraining them with smaller datasets showcasing different distributions, transfer learning has proven to be an effective approach in implementing deep learning models [10].

Promising results have been demonstrated by leveraging pre-trained CNN models from the ImageNet dataset and retraining them for leaf disease recognition [11].

Recently, MTL models have been explored in the field of plant disease detection, enabling the simultaneous execution of multiple tasks. These models can effectively handle tasks, such as severity estimation of leaf disease, identification of plant species, and classification of plant disease concurrently [12]. To address the limitations in information utilization and scalability of existing models, Lee et al. [13] introduced a novel Conditional Multi-Task Learning (CMTL) model in this context. This method employs an MTL mechanism with a conditional scheme that links host species and disease representations, mimicking how plant pathologists infer diseases from species information. This improves plant disease identification performance compared to traditional joint species–disease pair modeling. It efficiently utilizes available information and is scalable, not requiring distinctions between similar species or diseases, thus enhancing accuracy and suitability for real-world applications.

However, ViTs offer advantages over MTL frameworks in plant disease detection. They capture both local and global context information, enabling accurate analysis of plant images. ViTs also generalize well with limited labeled data through pre-training on large-scale datasets like ImageNet. Additionally, they provide flexibility and scalability by adapting to different disease detection tasks with a single model, reducing complexity and computational requirements.

In their work, Thakur et al. [14] introduced a hybrid model, named PlantViT, for automating plant disease detection, which integrates the strengths of a CNN and a ViT, incorporating a multi-head attention module. The model was evaluated on two extensive plant disease detection datasets, namely, PlantVillage and Embrapa. Experimental results showed that the model achieves detection accuracies of 98.61% and 87.87% on the PlantVillage and the Embrapa datasets, respectively. These results outperform the current state-of-the-art methods, highlighting the effectiveness of PlantViT in plant disease detection.

In [15], a lightweight deep learning approach leveraging the ViT architecture is proposed for real-time automated plant disease classification. The study rigorously compares various models, including ViT, traditional CNNs, and a hybrid of CNN and ViT. Extensive training and evaluation across multiple datasets reveal that while attention blocks significantly enhance accuracy, they also introduce latency in predictions. However, integrating attention blocks with CNN blocks strikes an optimal balance between accuracy and speed, offering a robust solution for real-time plant disease classification.

In recent studies, deep learning models have been employed to identify cassava leaf diseases, replacing the traditional and unreliable intuitive diagnosis conducted by farmers. To improve the disease detection accuracy in these species, Thai et al. [16] explored the use of the ViT model instead of a CNN. Experimental results demonstrate that the ViT model achieves competitive accuracy, surpassing popular CNN models such as EfficientNet and Resnet50 on the Cassava Leaf Disease Dataset. These findings highlight the potential superiority of the ViT model in analyzing leaf diseases.

An extended study by Thakur et al. [17] addressed the underexplored application of ViTs in plant pathology by introducing PlantXViT, a light-weight model based on CNNs with ViTs for efficient identification of various plant diseases across different crops, well suited for IoT-based smart agriculture. Evaluation on five datasets shows that PlantXViT outperforms state-of-the-art models, achieving average accuracies of over 93.55%, 92.59%, and 98.33% on Apple, Maize, and Rice datasets, respectively, even in challenging backgrounds. The model's explainability is also assessed using gradient-weighted class activation maps, demonstrating its ability to provide insights into the decision-making process.

Zhu et al. [18] introduced the Multiscale Convolution and Vision Transformer (MSCVT), a hybrid model that integrates multiscale convolutions with ViT techniques for precise crop disease identification. This approach combines the strengths of multiscale convolutions, which capture fine-grained local features, with ViT's ability to model long-range dependencies and global context. The study demonstrates that MSCVT outperforms single-scale CNNs and standalone ViT models by effectively capturing local and global details while optimizing computational efficiency. This balance ensures high accuracy in disease identification and practical deployment in resource-constrained environments, making MSCVT a valuable tool for early detection and management of crop diseases.

The ConvViT [19] is a lightweight ViT-based approach developed for apple leaf disease identification. It integrates convolutional and Transformer structures to effectively capture both global and local features of crop disease spots, enhancing overall robustness and accuracy. The patch embedding method is improved to preserve edge information and facilitate information exchange within the Transformer. By leveraging depthwise separable convolution and linear-complexity multi-head attention operations, ConvViT achieves substantial reduction in resource requirements. This model achieves comparable identification performance (96.85%) on a complex background apple leaf disease dataset, with significantly reduced parameters (32.7%) and FLOPs

(21.7%). This highlights the effectiveness and practicality of ConvViT as a disease identification model.

The Shuffle Convolution-based Lightweight Vision Transformer (SLViT) [20] is a hybrid network designed for the accurate diagnosis of sugarcane leaf diseases. The study demonstrates that SLViT, which combines a lightweight CNN architecture with a flexible plug-in Transformer encoder, surpasses the state-of-the-art models on the publicly available Plant Village dataset with respect to the metrics speed, weight, memory usage, and precision.

The effectiveness of MTL in enhancing correlated tasks has been well established. Existing approaches typically employ a backbone for initial feature extraction and utilize exclusive branches for each task. However, the sharing of information between branches is often accomplished through basic concatenation or summation of feature maps, which results in overlooking the local image features and neglecting the significance or correlation between tasks.

MTL-based ViTs offer a significant advancement in addressing the limitations of existing approaches. By integrating ViTs into the MTL framework, these models can capture both the global and local characteristics of images, enabling more comprehensive information exchange between tasks. This integration allows for a better understanding of the relationships and dependencies among tasks, leading to improved performance across multiple related tasks. MTL-based ViTs provide a more holistic approach to leveraging shared knowledge and extracting meaningful features, resulting in enhanced accuracy and robustness in correlated task scenarios.

In the recent years, ViTs have been used in performing multiple tasks simultaneously sharing the data across different tasks. The Unified Transformer (UniT) [21] model is proposed for the simultaneous learning of multiple tasks across diverse domains, encompassing object detection, natural language understanding, and multimodal reasoning. The UniT encodes input modalities using an encoder and generates task predictions using a shared decoder over the encoded input representations. Task-specific output heads are employed, and the entire model is trained jointly end-to-end with task losses. UniT distinguishes itself by employing shared model parameters across all tasks, accommodating a wide range of tasks across domains, and delivering robust performance across 7 tasks on 8 datasets with reduced parameter count.

Similarly, the MulT [22] framework proposes an end-to-end MTL-based Transformer for simultaneous learning of multiple high-level vision tasks. It outperforms state-of-the-art multitask CNNs and single-task Transformers, demonstrating the significance of shared attention across tasks. The MulT model shows robustness and generalization to new domains in multitask benchmarks. The study highlights the advantages of Transformer-based architectures for MTL, such as better scalability, improved feature representation, and the ability to capture complex dependencies between tasks, thereby providing a substantial contribution to the field and paving the way for future research and applications in domains requiring robust multitask learning solutions.

To address the challenges of constructing a comprehensive dataset for diagnosing and quantifying the COVID-19 severity using Chest X-ray (CXR) data, a novel Multi-task ViT architecture is proposed by Park et al. [23]. This approach leverages abundant unlabeled CXR data through self-attention modeling, overcoming the limitations of existing ViT models that are not optimized for CXR. By utilizing a backbone network trained on public datasets, the multi-task ViT extracts informative low-level features from the images, which serve as valuable resources for a robust ViT. Best performances are exhibited by the model in both diagnosis and severity quantification tasks, and remarkable generalization capability is demonstrated on external test datasets from diverse institutions. These promising results underscore the potential for widespread deployment of the model in the medical field.

Tian et al. [24] introduced the MultiTask ViT (MTViT), a representation learning method that leverages ViTs. MTViT introduces a multiple branch transformer that sequentially processes image patches associated with different tasks, allowing for information exchange through the Cross-Task attention (CA) module. Unlike previous models, MTViT utilizes ViT's self-attention mechanism for feature extraction, resulting in improved memory and computation efficiency. Experimental results on benchmark datasets demonstrate that MTViT outperforms or achieves comparable performance to existing CNN-based MTL methods.

Despite the widespread adoption of MTL-based ViTs in various computer vision applications, there remains a noticeable research gap in their application to plant disease detection. The work proposed by Goncalves et al. [25], known as MTLSegFormer, stands out as the only existing effort in exploring the potential of MTL-based ViTs to address plant disease detection challenges. MTLSegFormer is a semantic segmentation model designed for precision agriculture that leverages both MTL and attention mechanisms. It combines MTL and attention mechanisms, learning two feature maps for each task after extracting backbone features. The model assigns weights to important local regions for specific tasks, resulting in improved accuracy for correlated tasks in precision agriculture.

CHAPTER 3: PROJECT DESIGN AND MODULES

3.1 UML Diagrams

Unified Modeling Language (UML) is a standardized visual language used to model, design, and document the architecture of software systems. It provides a set of graphical notations to represent the structure and behavior of a system, making complex systems easier to understand and communicate among developers, stakeholders, and business analysts.

Key Points About UML

- **Standardized Notation:** UML offers a universal set of symbols and diagrams that standardize how software systems are described, which helps teams speak the same language regardless of their background or the programming language they use.
- **Types of Diagrams:** UML includes various diagrams that can be categorized into two main types:
    - **Structural Diagrams:** These describe the static aspects of the system. Examples include Class Diagrams, Component Diagrams, and Deployment Diagrams.
    - **Behavioral Diagrams:** These focus on the dynamic aspects and interactions within the system. Examples include Use Case Diagrams, Sequence Diagrams, Activity Diagrams, and Collaboration Diagrams.
- **System Documentation and Communication:** UML serves as an effective tool for documenting system requirements, design decisions, and the overall architecture. It helps bridge the gap between technical and non-technical stakeholders by providing clear, visual representations of the system.
- **Design and Analysis:** By modeling different aspects of a system, UML enables developers to analyze and validate the design early in the development process. This can lead to better decision-making, reduced complexity, and improved system maintainability.
- **Flexibility:** UML is versatile and can be used across a wide range of applications, from small-scale projects to large, complex systems. It supports object-oriented design principles and can be adapted to various methodologies such as Agile or Waterfall.

3.2 Use Case Diagram

A Use Case Diagram shows the interactions between Actors (people or external systems) and the Use Cases (high-level functionalities) in the system. In this project, we have two main actors: Admin and User. The Admin is responsible for uploading the dataset, performing data analysis, preprocessing, training models, and saving/loading model weights. The User primarily logs in to make predictions on new data.

Key use cases include:

1. Login (for both Admin and User)
2. Upload Dataset (Admin)
3. Data Preprocessing (Admin)
4. Apply SMOTE (Admin)
5. EDA (Admin)
6. Train/Test Split (Admin)
7. Train Models (Admin)
8. Predict Disease (User)
9. Signup (Admin or User)

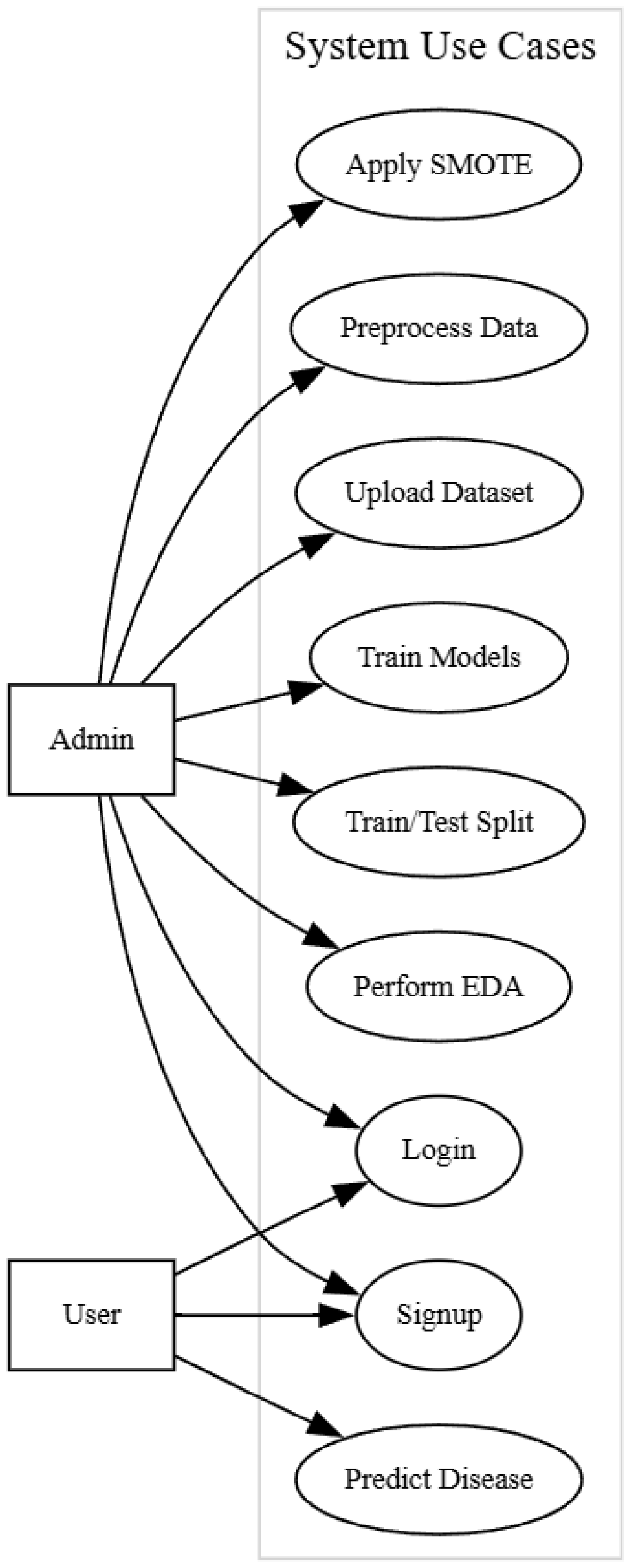


Fig. 3.2: Use case diagram.

3.3 Class Diagram

A Class Diagram outlines the system's classes (or major modules), their attributes, and methods, as well as the relationships among them.

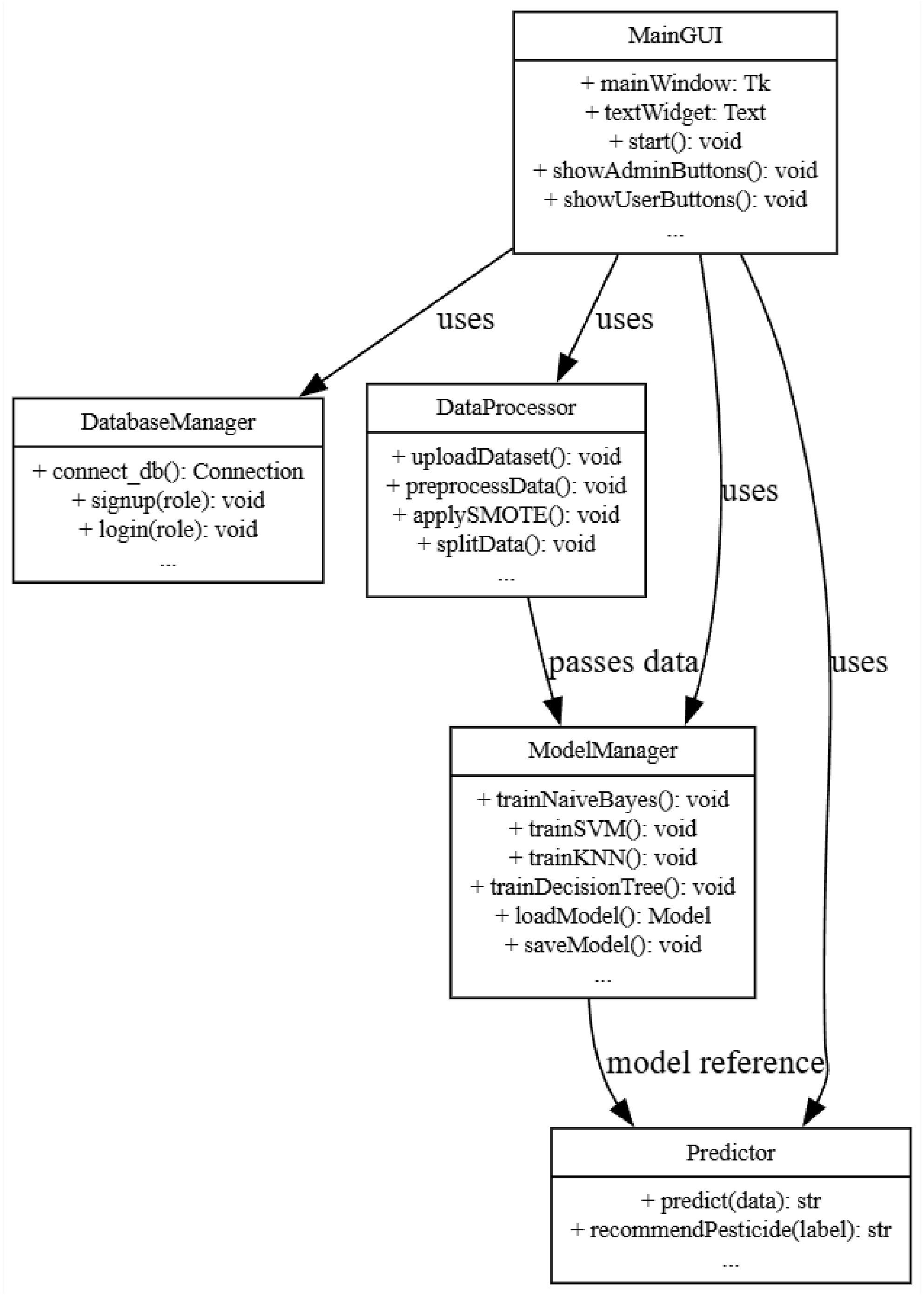


Fig. 3.3: Class diagram.

In this project, the main classes are:

1. MainGUI – Handles the Tkinter interface and event callbacks.

2. DatabaseManager – Manages the MySQL connection, user signup, and login.
3. DataProcessor – Responsible for dataset loading, cleaning, encoding, and SMOTE.
4. ModelManager – Trains and saves/loads the machine learning models.
5. Predictor – Uses the trained models for predictions and provides pesticide recommendations.

The relationships can include:

- **Association:** A class uses another class's functionality.
- **Aggregation/Composition:** A class contains or owns another class's instance.
- **Inheritance (optional):** If any classes inherit from a base class.

3.4 Sequence Diagram

A Sequence Diagram illustrates how objects or components interact in a specific use case over time. Below is a scenario describing how an Admin logs in, uploads a dataset, preprocesses data, applies SMOTE, trains a model, and then how a User logs in to make a prediction.

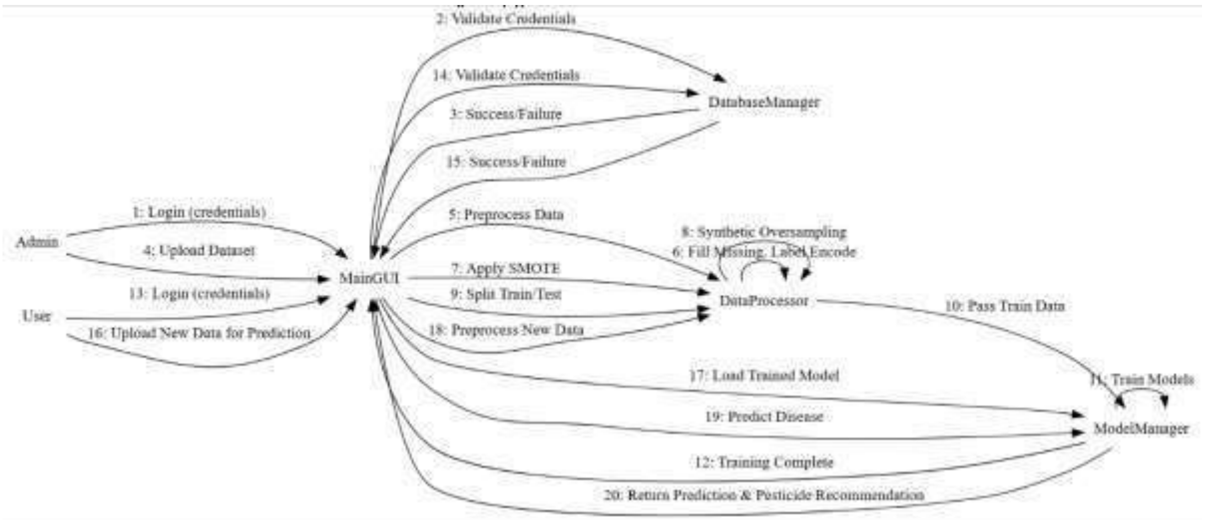


Fig. 3.4: Sequence diagram.

3.5 Data Flow Diagram

A Data Flow Diagram (DFD) focuses on how data moves through the system, illustrating processes, data stores, external entities, and data flows. In this context:

- **External Entities:** Admin and User.
- **Processes:** Data Preprocessing, Model Training, Prediction, etc.
- **Data Stores:** MySQL Database (for user credentials), local storage or memory (for dataset and model files).

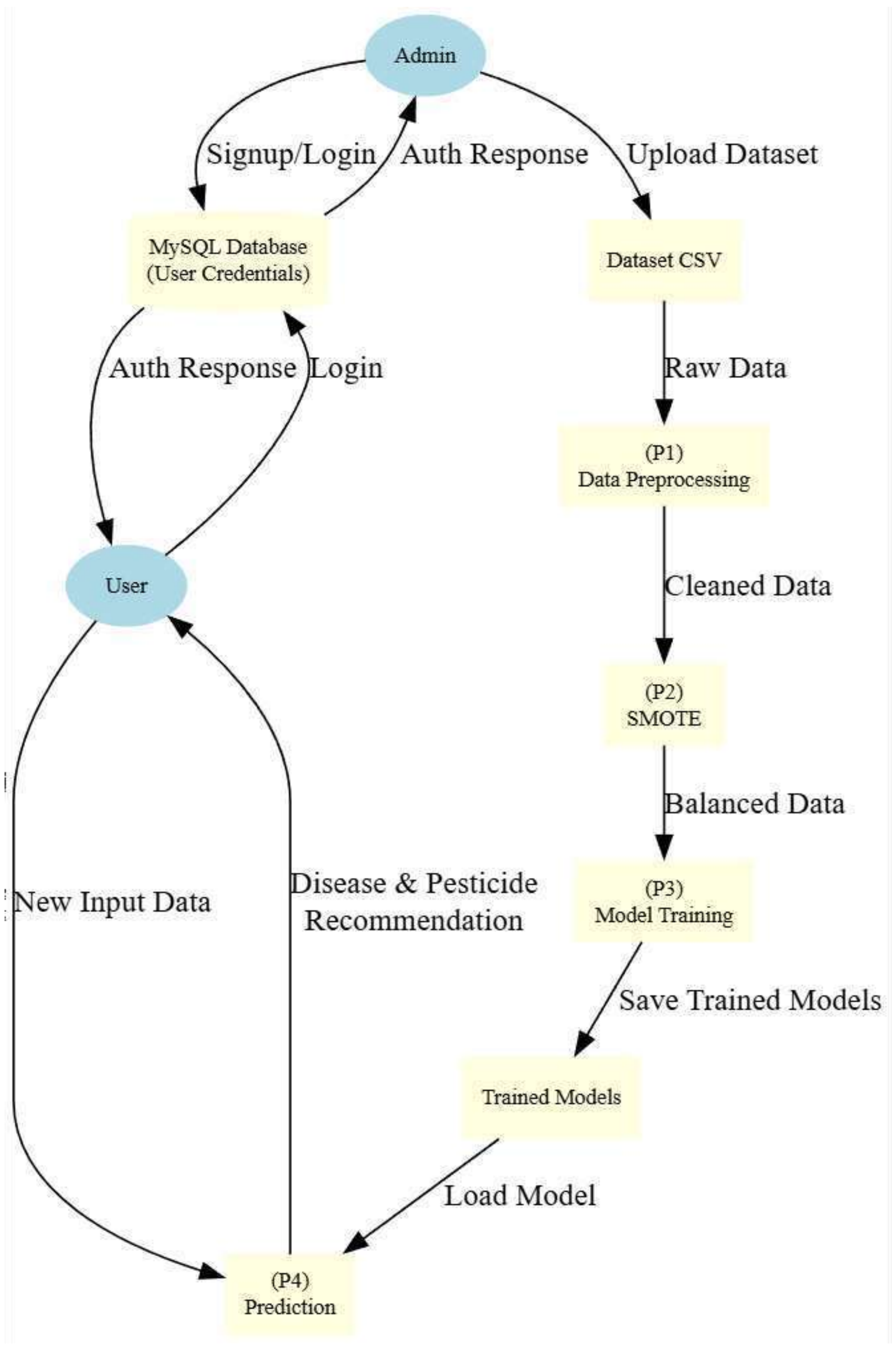


Fig. 3.5: Data flow diagram.

3.6 Deployment Diagram

A Deployment Diagram illustrates the physical layout of the system, showing how software components (or artifacts) are deployed on different nodes (servers, machines, devices). In this project, we typically have:

- **Local/Server Node:** Runs the Python application (Tkinter GUI, data preprocessing, SMOTE, model training, and prediction).
- **Database Node:** Hosts the MySQL database for user authentication and other persistent data.
- **Admin/User Device:** The same or separate machine(s) where the GUI is accessed.
  - In a local setup, Admin and User might run the GUI directly on the same machine that hosts the server.
  - In a client-server setup, Admin/User devices connect over a network to the server node.

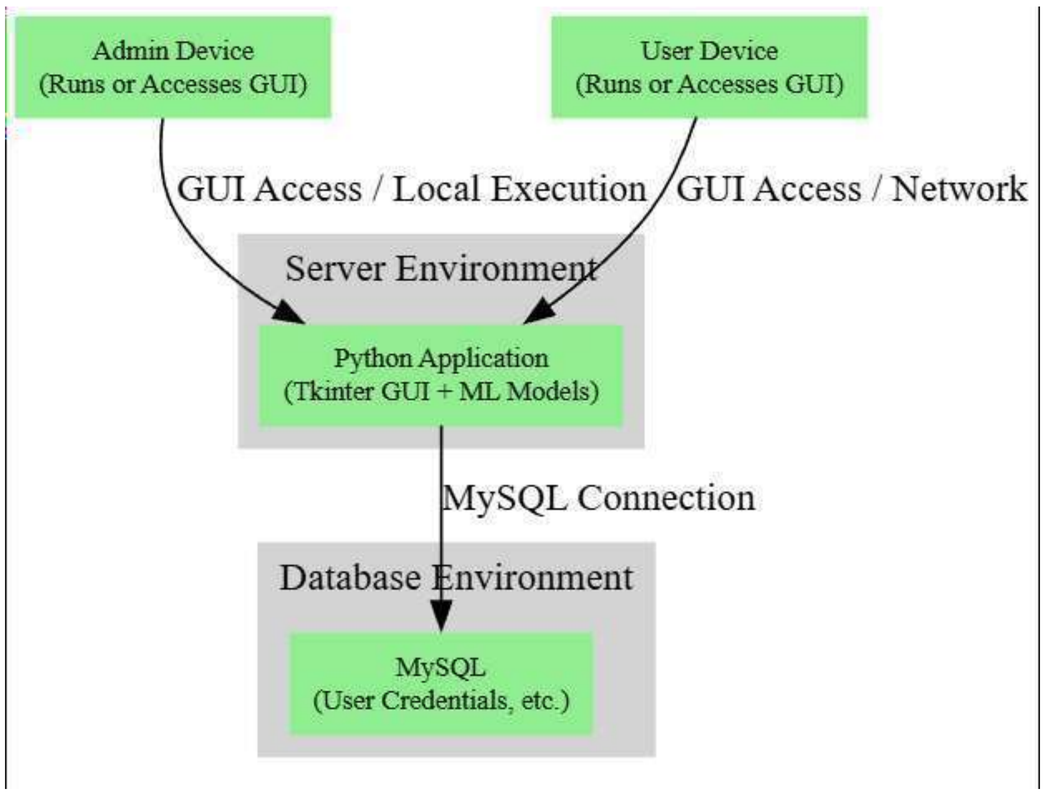


Fig. 3.6: Deployment diagram.

Explanation:

1. **Admin Device & User Device**: Where Admins and Users log in to interact with the system (upload data, view results, etc.).

2. **Server Environment**: Hosts the Python application, which includes the Tkinter GUI (if local) or a server-side component (if network-based).
3. **Database Environment**: Hosts the MySQL database used for storing user credentials (and potentially other persistent data).

3.7 Component Diagram

A Component Diagram shows the organization and dependencies among various high-level components or modules in the system. It differs from a Class Diagram by focusing on how modules interact via interfaces, rather than detailing the internal structure (attributes/methods).

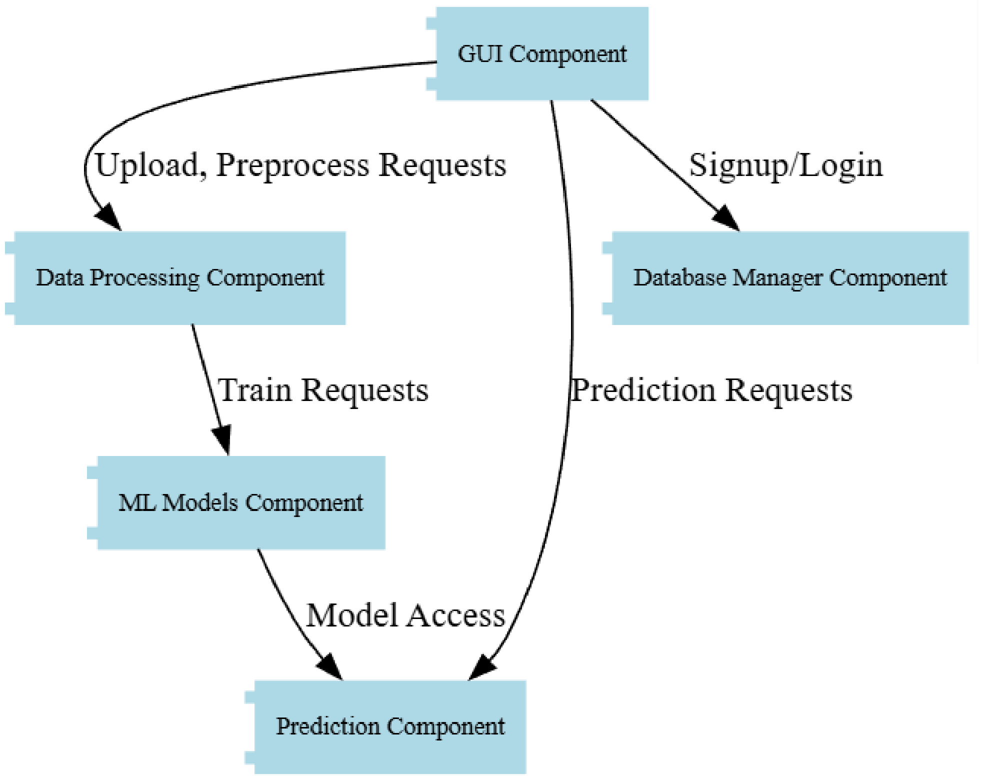


Fig. 3.7: Component diagram.

Key components in this project includes:

- **GUI Component**: Manages the Tkinter interface, user interactions, and event handling.
- **Data Processing Component**: Handles dataset operations (upload, cleaning, encoding, SMOTE).
- **ML Models Component**: Manages model training, saving/loading, and evaluation.

- **Prediction Component**: Uses trained models to predict diseases and provide pesticide recommendations.
- **Database Manager Component**: Interfaces with the MySQL database for user authentication.

Explanation:

1. **GUI Component**: Initiates requests to other components based on user actions (e.g., upload dataset, train model, predict disease).
2. **Data Processing Component**: Cleans the data, applies label encoding, handles SMOTE, and provides processed data to the ML Models component.
3. **ML Models Component**: Trains various algorithms (Gaussian NB, SVM, KNN, DTC) and can store/load model artifacts.
4. **Prediction Component**: Receives new input data and uses the trained models to predict the disease class, then suggests pesticides.
5. **Database Manager Component**: Responsible for user authentication and other interactions with the MySQL database.

3.8 Activity Diagram

This activity diagram visually captures the high-level workflow for both Admin and User operations in the plant disease classification project, highlighting the steps involved from user authentication to final output.

1. **Start:** The process begins at the "Start" node, indicating the initiation of the system.
2. **Login/Signup:** Users (both Admin and User) first perform login or signup operations. This step ensures that only authenticated users gain access to the system.
3. **Select Role:** After authentication, the user selects their role (Admin or User). This decision point directs the workflow into two distinct paths.

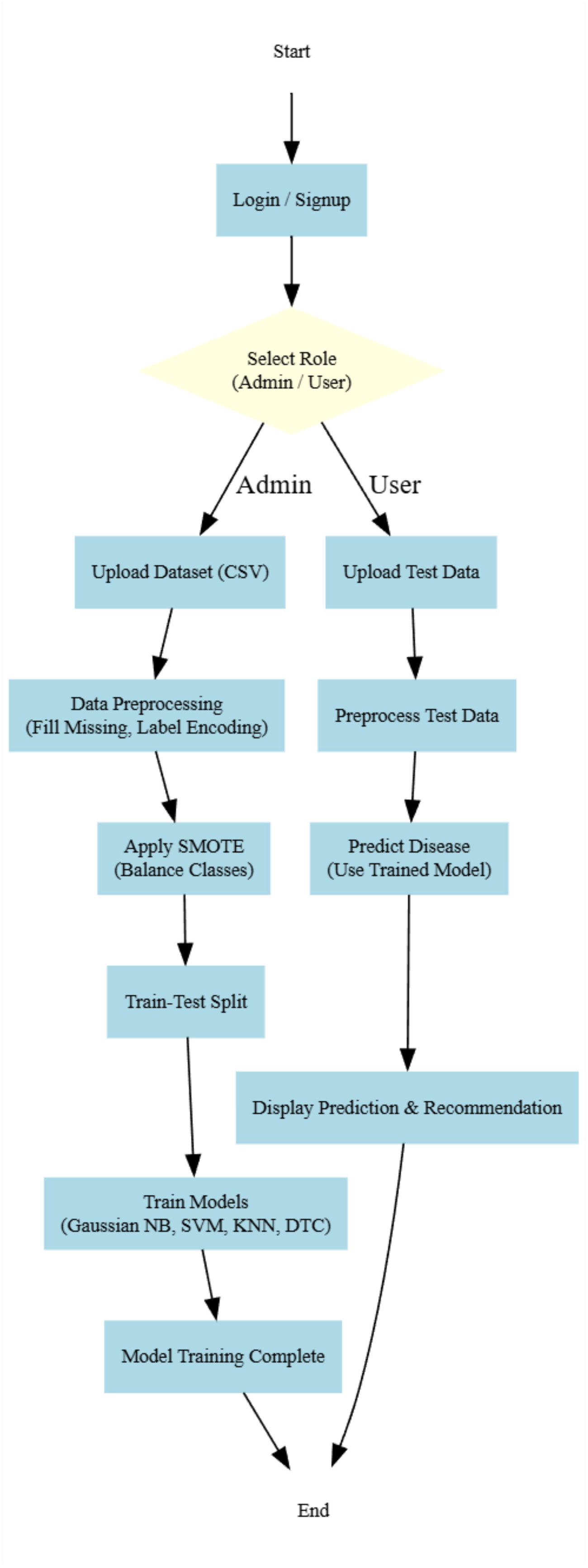


Fig. 3.8: Activity diagram.

4. **Admin Flow:**
    - **Upload Dataset:** The Admin uploads a dataset (in CSV format) to the system.
    - **Data Preprocessing:** The dataset undergoes preprocessing steps such as filling missing values and label encoding.
    - **Apply SMOTE:** SMOTE is applied to balance the dataset by generating synthetic samples for minority classes.
    - **Train-Test Split:** The processed dataset is split into training and testing sets.
    - **Train Models:** Various machine learning models (Gaussian NB, SVM, KNN, and DTC) are trained on the training data.
    - **Model Training Complete:** Once the models are trained, the process concludes for the Admin.
5. **User Flow:**
    - **Upload Test Data:** A User uploads new test data for prediction.
    - **Preprocess Test Data:** The test data is preprocessed to match the format expected by the trained model.
    - **Predict Disease:** The system uses the trained model to predict the disease.
    - **Display Prediction & Recommendation:** The prediction results and corresponding pesticide recommendations are displayed to the user.
6. **End:** Both the Admin and User flows converge at the "End" node, marking the completion of the respective processes.

3.9 Collaboration Diagram

A Collaboration Diagram (also known as a Communication Diagram in UML) focuses on object interactions organized around the structural relationships in a scenario. It is similar to a Sequence Diagram but emphasizes the links between objects rather than the strict time sequence.

Explanation:

1. **GUI** interacts with **Database Manager** to authenticate users.
2. **GUI** then calls **Data Processor** to upload and preprocess the dataset.
3. **Data Processor** applies SMOTE and passes the balanced data to **Model Manager** for training.
4. Once training is complete, **GUI** can request predictions from the **Predictor** object.
5. **Predictor** fetches the trained model from **ModelManager** and returns results (e.g., predicted disease, recommended pesticide) to the GUI.

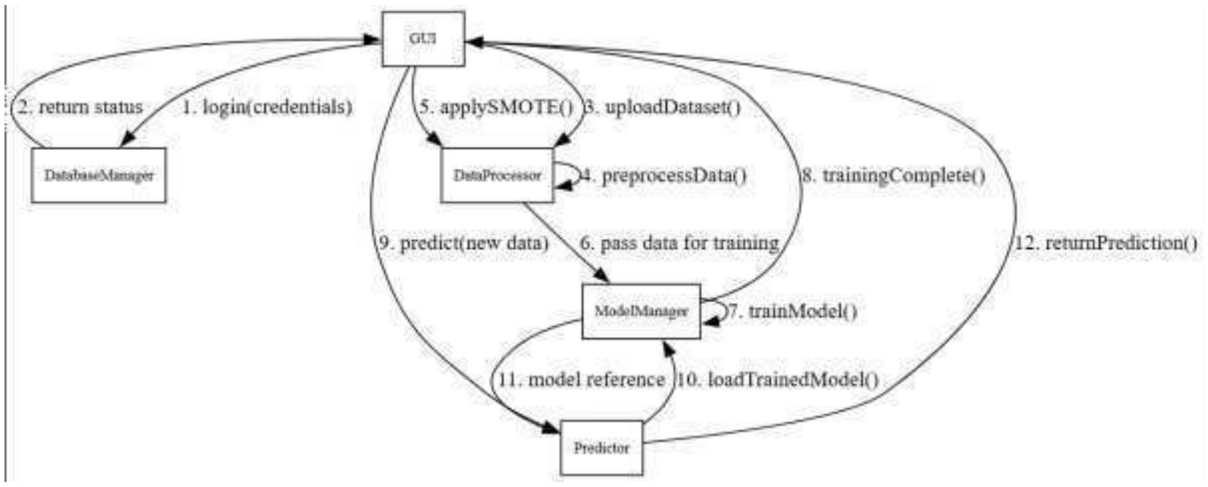


Fig. 3.9: Collaboration diagram.

CHAPTER 4: IMPLEMENTATION

```
from tkinter import messagebox

from tkinter import *

from tkinter import simpledialog

import tkinter

import tkinter as tk

from tkinter import messagebox

import pymysql

from PIL import Image, ImageTk


import matplotlib.pyplot as plt

from tkinter import simpledialog

from tkinter import filedialog

import os

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.preprocessing import LabelEncoder

from imblearn.over_sampling import SMOTE

from sklearn.model_selection import train_test_split

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

from sklearn.naive_bayes import GaussianNB

import joblib

import os

from sklearn.metrics import precision_score

from sklearn.metrics import recall_score

from sklearn.metrics import f1_score

from sklearn.metrics import accuracy_score,confusion_matrix,classification_report


labels = ['diaporthe-stem-canker', 'charcoal-rot', 'rhizoctonia-root-rot',

    'phytophthora-rot', 'brown-stem-rot', 'powdery-mildew',

    'downy-mildew', 'brown-spot', 'bacterial-blight',

    'bacterial-pustule', 'purple-seed-stain', 'anthracnose',

    'phyllosticta-leaf-spot', 'alternarialeaf-spot',

    'frog-eye-leaf-spot', 'diaporthe-pod-&-stem-blight',

    'cyst-nematode', 'herbicide-injury']


pesticides = ['Tebuconazole', 'Azoxystrobin', 'Carboxin',

    'Mefenoxam', 'Crop Rotation', 'Triazoles',

    'Metalaxyl', 'Tebuconazole', 'Copper Oxychloride',
        'Oxytetracycline', 'Azoxystrobin', 'Difenoconazole',
        'Thiophanate-methyl', 'Azoxystrobin',
        'Difenoconazole', 'Propiconazole',
        'Oxamyl', 'Amino Acid & Seaweed Extracts']

def uploadDataset():
    global filename, dataset, labels
    filename = filedialog.askopenfilename(initialdir="Dataset")
    text.delete('1.0', END)
    text.insert(END,filename+" loaded\n\n")
    dataset = pd.read_csv(filename)
    text.insert(END,str(dataset))

def bar_plot(df, column):
    """Bar plot showing mean values of a numerical column per Label."""
    plt.figure(figsize=(8, 5))
    sns.barplot(x="Label", y=column, data=df, estimator=lambda x: x.mean(), ci=None)
    plt.title(f'Bar Plot of {column} (Mean) by Label')
    plt.xlabel("Label")
    plt.ylabel(f'Mean {column}')
    plt.show()

def violin_plot(df, column):
    """Violin plot for a numerical column grouped by Label."""

    plt.figure(figsize=(8, 5))

    sns.violinplot(x="Label", y=column, data=df)

    plt.title(f'Violin Plot of {column} by Label')

    plt.xlabel("Label")

    plt.ylabel(column)

    plt.xticks(rotation=45)

    plt.show()


def strip_plot(df, column):

    """Strip plot for a numerical column grouped by Label."""

    plt.figure(figsize=(8, 5))

    sns.stripplot(x="Label", y=column, data=df, jitter=True, alpha=0.5)

    plt.title(f'Strip Plot of {column} by Label')

    plt.xlabel("Label")

    plt.ylabel(column)

    plt.xticks(rotation=45)

    plt.show()


def histogram(df, column):

    """Histogram for a numerical column, colored by Label."""

    plt.figure(figsize=(8, 5))

    for label in df["Label"].unique():
        subset = df[df["Label"] == label]
        plt.hist(subset[column], bins=30, alpha=0.5, label=str(label))
    plt.title(f'Histogram of {column} by Label')
    plt.xlabel(column)
    plt.ylabel("Frequency")
    plt.legend(title="Label")
    plt.show()

def scatter_plot(df, x_column, y_column):
    """Scatter plot for two numerical columns, colored by Label."""
    plt.figure(figsize=(8, 5))
    sns.scatterplot(x=x_column, y=y_column, hue="Label", data=df, alpha=0.6)
    plt.title(f'Scatter Plot of {x_column} vs {y_column}')
    plt.xlabel(x_column)
    plt.ylabel(y_column)
    plt.legend(title="Label")
    plt.show()

def strip_plot(df, column):
    """Strip plot for a numerical column grouped by Label."""
    plt.figure(figsize=(8, 5))
    sns.stripplot(x="Label", y=column, data=df, jitter=True, alpha=0.5)
    plt.title(f'Strip Plot of {column} by Label')
    plt.xlabel("Label")
    plt.ylabel(column)
    plt.xticks(rotation=45)
    plt.show()


def correlation_heatmap(df):
    """Heatmap of correlation matrix for numerical columns."""
    plt.figure(figsize=(10, 6))
    sns.heatmap(df.select_dtypes(include=['float64',                'int64']).corr(),                annot=True,
cmap='coolwarm', fmt=".2f")
    plt.title('Correlation Heatmap')
    plt.show()


def EDA():
    global filename, dataset, labels
    df=dataset
    bar_plot(df, "temp")                  # 1. Bar plot
    violin_plot(df, "leafspot-size")       # 2. Violin plot
    histogram(df, "plant-growth")          # 4. Histogram
    scatter_plot(df, "temp", "germination")# 5. Scatter plot
    strip_plot(df, "stem")                 # 6. Strip plot
    correlation_heatmap(df)                # 7. Correlation heatmap
    scatter_plot(df, "leafspot-size", "germination")# 5. Scatter plot
```

```
def DatasetPreprocessing():
    text.delete('1.0', END)
    global X, Y, dataset, label_encoder

    #dataset contains non-numeric values but ML algorithms accept only numeric values so by
applying Lable
    #encoding class converting all non-numeric data into numeric data
    dataset.fillna(0, inplace = True)
    label_encoder = []
    columns = dataset.columns
    types = dataset.dtypes.values
    for i in range(len(types)):
        name = types[i]
        if name == 'object': #finding column with object type
            le = LabelEncoder()
            dataset[columns[i]] = pd.Series(le.fit_transform(dataset[columns[i]].astype(str)))#encode
all str columns to numeric
            label_encoder.append(le)

    X = dataset.drop(['Label'], axis = 1)
    Y = dataset['Label']
    labels, label_count = np.unique(dataset['Label'], return_counts=True)
    text.insert(END,"Dataset Normalization & Preprocessing Task Completed\n\n")
    text.insert(END,str(dataset)+"\n\n")

def Dataset_SMOTE():
    text.delete('1.0', END)
    global X, Y, dataset, label_encoder
    labels, label_count = np.unique(dataset['Label'], return_counts=True)
    #dataset preprocessing such as replacing missing values, normalization and splitting dataset into
train and test
    smote = SMOTE(random_state=42)
    X, Y = smote.fit_resample(X, Y)
# Count after SMOTE
    labels_resampled, label_count_resampled = np.unique(Y, return_counts=True)
# Plotting
    plt.figure(figsize=(10, 5))
# Before SMOTE
    plt.subplot(1, 2, 1)
    plt.bar(labels, label_count, color='skyblue', alpha=0.8)
    plt.xlabel("Output Type")
    plt.ylabel("Count")
    plt.title("Before SMOTE")
    # After SMOTE

    plt.subplot(1, 2, 2)

    plt.bar(labels_resampled, label_count_resampled, color='lightgreen', alpha=0.8)

    plt.xlabel("Output Type")

    plt.ylabel("Count")

    plt.title("After SMOTE")

    plt.tight_layout()

    plt.show()




def Train_test_splitting():

    text.delete('1.0', END)

    global X, Y, dataset, label_encoder

    global X_train, X_test, y_train, y_test, scaler




    #splitting dataset into train and test where application using 80% dataset for training and 20%
for testing

    X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2) #split dataset into train
and test

    text.insert(END,"Dataset Train & Test Splits\n")

    text.insert(END,"Total Data found in dataset : "+str(X.shape[0])+"\n")

    text.insert(END,"80% dataset used for training  : "+str(X_train.shape[0])+"\n")
    text.insert(END,"20% dataset user for testing   : "+str(X_test.shape[0])+"\n")



def calculateMetrics(algorithm, testY, predict):
    global labels
    p = precision_score(testY, predict,average='macro') * 100
    r = recall_score(testY, predict,average='macro') * 100
    f = f1_score(testY, predict,average='macro') * 100
    a = accuracy_score(testY,predict)*100
    accuracy.append(a)
    precision.append(p)
    recall.append(r)
    fscore.append(f)
    text.insert(END,algorithm+" Accuracy  : "+str(a)+"\n")
    text.insert(END,algorithm+" Precision : "+str(p)+"\n")
    text.insert(END,algorithm+" Recall    : "+str(r)+"\n")
    text.insert(END,algorithm+" FSCORE    : "+str(f)+"\n\n")
    conf_matrix = confusion_matrix(testY, predict)
    ax = sns.heatmap(conf_matrix, xticklabels = labels, yticklabels = labels, annot = True,
cmap="viridis" ,fmt ="g");
    ax.set_ylim([0,len(labels)])
    plt.title(algorithm+" Confusion matrix")
    plt.ylabel('True class')
    plt.xlabel('Predicted class')

    plt.show()


#now train existing algorithm

def Existing_Classifier():

    text.delete('1.0', END)

    global accuracy, precision, recall, fscore

    global X_train, y_train, X_test, y_test

    accuracy = []

    precision = []

    recall = []

    fscore = []


    if os.path.exists('model/naive_bayes_model.pkl'):

        classifier = joblib.load('model/naive_bayes_model.pkl')

    else:

        classifier = GaussianNB()

        classifier.fit(X_train, y_train)

        joblib.dump(classifier,  'model/naive_bayes_model.pkl')


    y_pred_bnb = classifier.predict(X_test)

    calculateMetrics("GNBC Model", y_test, y_pred_bnb)
```

```
def Existing_Classifier1():
    text.delete('1.0', END)
    global accuracy, precision, recall, fscore
    global X_train, y_train, X_test, y_test
    accuracy = []
    precision = []
    recall = []
    fscore = []

    if os.path.exists('model/SVC.pkl'):
        classifier = joblib.load('model/SVC.pkl')
    else:
        classifier = SVC()
        classifier.fit(X_train, y_train)
        joblib.dump(classifier,  'model/SVC.pkl')

    y_pred_bnb = classifier.predict(X_test)
    calculateMetrics("SVM Classifier", y_test, y_pred_bnb)

def Proposed_Classifier():
    global classifier
    text.delete('1.0', END)
    global X_train, y_train, X_test, y_test
    if os.path.exists('model/KNeighborsClassifier.pkl'):
        # Load the model from the pkl file
        classifier = joblib.load('model/KNeighborsClassifier.pkl')
    else:
        # Train the classifier on the training data
        classifier = KNeighborsClassifier(n_neighbors=5)
        classifier.fit(X_train, y_train)
        # Save the model weights to a pkl file
        joblib.dump(classifier, 'model/KNeighborsClassifier.pkl')

    y_pred = classifier.predict(X_test)
    calculateMetrics("KNN Classifier", y_test, y_pred)

def Proposed_Classifier1():
    global classifier
    text.delete('1.0', END)
    global X_train, y_train, X_test, y_test
    if os.path.exists('model/DecisionTreeClassifier.pkl'):
        # Load the model from the pkl file
        classifier = joblib.load('model/DecisionTreeClassifier.pkl')
    else:
        # Train the classifier on the training data
        classifier = DecisionTreeClassifier()
        classifier.fit(X_train, y_train)

        # Save the model weights to a pkl file

        joblib.dump(classifier, 'model/DecisionTreeClassifier.pkl')


    y_pred = classifier.predict(X_test)

    calculateMetrics("DTC Model", y_test, y_pred)



def predict():


    global classifier,pesticides

    text.delete('1.0', END)

    filename = filedialog.askopenfilename(initialdir="Dataset")#upload test data

    dataset = pd.read_csv(filename)#read data from uploaded file

    dataset.fillna(0, inplace = True)#removing missing values

    predict = classifier.predict(dataset)


    for i in range(len(X)):

        text.insert(END,"Sample Test Data:" +str(dataset.iloc[i]))

        text.insert(END,"Output Classified As ===> "+labels[int(predict[i])]+"\n")

        text.insert(END,"Pesticides Suggested As ===> "+pesticides[int(predict[i])]+"\n")


        text.insert(END,"\n\n\n")
```

```
def connect_db():
    return pymysql.connect(host='localhost', user='root', password='root', database='sparse_db')

# Signup Functionality
def signup(role):
    def register_user():
        username = username_entry.get()
        password = password_entry.get()

        if username and password:
            try:
                conn = connect_db()
                cursor = conn.cursor()
                query = "INSERT INTO users (username, password, role) VALUES (%s, %s, %s)"
                cursor.execute(query, (username, password, role))
                conn.commit()
                conn.close()
                messagebox.showinfo("Success", f"{role} Signup Successful!")
                signup_window.destroy()
            except Exception as e:
```

```
            messagebox.showerror("Error", f"Database Error: {e}")
        else:
            messagebox.showerror("Error", "Please enter all fields!")

    signup_window = tk.Toplevel(main)
    signup_window.geometry("400x300")
    signup_window.title(f"{role} Signup")

    tk.Label(signup_window, text="Username").pack(pady=5)
    username_entry = tk.Entry(signup_window)
    username_entry.pack(pady=5)

    tk.Label(signup_window, text="Password").pack(pady=5)
    password_entry = tk.Entry(signup_window, show="*")
    password_entry.pack(pady=5)

    tk.Button(signup_window, text="Signup", command=register_user).pack(pady=10)

# Login Functionality
def login(role):
    def verify_user():
        username = username_entry.get()
        password = password_entry.get()
    if username and password:
        try:
            conn = connect_db()
            cursor = conn.cursor()
            query = "SELECT * FROM users WHERE username=%s AND password=%s AND
role=%s"
            cursor.execute(query, (username, password, role))
            result = cursor.fetchone()
            conn.close()
            if result:
                messagebox.showinfo("Success", f"{role} Login Successful!")
                login_window.destroy()
                if role == "Admin":
                    show_admin_buttons()
                elif role == "User":
                    show_user_buttons()
            else:
                messagebox.showerror("Error", "Invalid Credentials!")
        except Exception as e:
            messagebox.showerror("Error", f"Database Error: {e}")
    else:
        messagebox.showerror("Error", "Please enter all fields!")
    login_window = tk.Toplevel(main)
    login_window.geometry("400x300")
    login_window.title(f"{role} Login")

    tk.Label(login_window, text="Username").pack(pady=5)
    username_entry = tk.Entry(login_window)
    username_entry.pack(pady=5)

    tk.Label(login_window, text="Password").pack(pady=5)
    password_entry = tk.Entry(login_window, show="*")
    password_entry.pack(pady=5)

    tk.Button(login_window, text="Login", command=verify_user).pack(pady=10)

# Admin Button Functions
def show_admin_buttons():
    clear_buttons()
    tk.Button(main, text="Upload Dataset", command=uploadDataset, font=font1).place(x=70,
y=200)
    tk.Button(main, text="Data Analytics", command=EDA, font=font1).place(x=230, y=200)
    tk.Button(main, text="Preprocessing", command=DatasetPreprocessing,
font=font1).place(x=300, y=200)
```

```
    tk.Button(main, text="Apply SMOTE", command=Dataset_SMOTE, font=font1).place(x=500, y=200)
    tk.Button(main, text="Data Splitting", command=Train_test_splitting, font=font1).place(x=650, y=200)
    tk.Button(main, text="GNBC Model", command=Existing_Classifier, font=font1).place(x=900, y=200)
    tk.Button(main, text="SVM Classifier", command=Existing_Classifier1, font=font1).place(x=1000, y=200)
    tk.Button(main, text="KNN Classifier", command=Proposed_Classifier, font=font1).place(x=1100, y=200)
    tk.Button(main, text="DTC Model", command=Proposed_Classifier1, font=font1).place(x=1200, y=200)


# User Button Functions
def show_user_buttons():
    clear_buttons()
    tk.Button(main, text="Prediction", command=predict, font=font1).place(x=550, y=200)
    #tk.Button(main, text="Comparison Graph", command=comparison_graph, font=font1).place(x=400, y=400)


# Clear buttons before adding new ones
def clear_buttons():
    for widget in main.winfo_children():
        if isinstance(widget, tk.Button) and widget not in [admin_button, user_button]:
            widget.destroy()
```

```
# Main tkinter window


main = tk.Tk()

main.title("Machine Learning Adopted Multi-class Classification of Plant Diseases with Sparse
and Categorical IoT Data") #designing main screen

screen_width = main.winfo_screenwidth()

screen_height = main.winfo_screenheight()

main.geometry(f"{screen_width}x{screen_height}")

# Load and set background image

bg_image_path = "background.png"  # Replace with your image file path

bg_image = Image.open(bg_image_path)

bg_image = bg_image.resize((screen_width, screen_height), Image.LANCZOS)

bg_photo = ImageTk.PhotoImage(bg_image)

# Create a background label

bg_label = tk.Label(main, image=bg_photo)

bg_label.place(x=0, y=0, relwidth=1, relheight=1)





# Title

font = ('times', 18, 'bold')
```

```
title = tk.Label(main, text="Machine Learning Adopted Multi-class Classification of Plant Diseases with Sparse and Categorical IoT Data", bg='white', fg='black', font=font, height=2, width=100)

title.pack()


font1 = ('times', 12, 'bold')

text=Text(main,height=22,width=170)

scroll=Scrollbar(text)

text.configure(yscrollcommand=scroll.set)

text.place(x=100,y=250)

text.config(font=font1)



# Admin and User Buttons

font1 = ('times', 14, 'bold')



tk.Button(main, text="Admin Signup", command=lambda: signup("Admin"), font=font1, width=20, height=2, bg='LightBlue').place(x=100, y=100)


tk.Button(main, text="User Signup", command=lambda: signup("User"), font=font1, width=20, height=2, bg='LightGreen').place(x=400, y=100)
```

```
admin_button = tk.Button(main, text="Admin Login", command=lambda: login("Admin"), font=font1, width=20, height=2, bg='LightBlue')

admin_button.place(x=700, y=100)


user_button = tk.Button(main, text="User Login", command=lambda: login("User"), font=font1, width=20, height=2, bg='LightGreen')

user_button.place(x=1000, y=100)


main.config(bg='OliveDrab2')

main.mainloop()
```

CHAPTER 5: TESTING

Testing ensures that the plant disease classification system works as intended, delivers accurate results, and can be used effectively in real-world agricultural settings. This phase examines the correctness of machine learning models, the smooth operation of the system's interface, and how well the system integrates data from IoT sensors. The goal is to verify that the system can detect plant diseases with high accuracy, even when the data is incomplete or contains categorical (non-numeric) variables.

5.1 Testing Environment

To test the system efficiently, a reliable environment was set up. This includes all necessary software tools, programming libraries, and hardware:

- **Python** was chosen due to its simplicity and strong support for machine learning.
- Libraries like **pandas**, **scikit-learn**, and **imbalanced-learn** were used for data processing, model training, and class balancing.
- Testing was done on both Windows and Ubuntu to ensure cross-platform compatibility.
- The hardware setup (i7 CPU, 16 GB RAM) ensured smooth processing of machine learning tasks and GUI functions.

5.2 Testing Methodology

5.2.1 Model Testing (Performance Testing)

This is about checking how well each machine learning model works. The dataset was split into two parts:

- **80% for training** the model
- **20% for testing** the model's performance on unseen data

The following metrics were used:

- **Accuracy**: How many predictions were correct out of all predictions
- **Precision**: How many of the predicted disease labels were actually correct
- **Recall**: How many actual diseases were correctly detected
- **F1-Score**: A balance between precision and recall

Additionally, **k-fold cross-validation (k=5)** was used. This technique splits the data into 5 equal parts and tests the model 5 times using a different test part each time. It ensures the results are not biased by one particular train-test split.

5.2.2 GUI Testing (Functional Testing)

The GUI was tested to ensure that users can:

- Log in securely
- Upload IoT data in CSV or text format
- View and understand disease prediction results
- See pesticide recommendations based on the predicted disease

Tests were performed to check how the system behaves with correct and incorrect inputs, as well as under slow internet conditions.

5.2.3 Integration Testing (System Testing)

This checks how well all parts of the system work together. It ensures:

- Data from IoT sensors is processed and cleaned properly
- Preprocessed data is sent to the correct machine learning model
- The model returns results that are displayed correctly in the GUI
- The recommendation system displays relevant pesticide info

All modules (data collection → preprocessing → prediction → GUI) were connected and tested as a full pipeline.

5.3 Test Results

Here's how each model performed after training and testing:

Model/Metric	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)
GNBC model	72.68	72.58	72.72	68.17
SVM classifier	92.36	92.92	92.56	92.09
KNN classifier	97.91	97.62	97.56	97.42
Proposed DTC model	99.07	98.88	98.76	98.86

The **Decision Tree Classifier** clearly outperforms the others in every category, which is why it was selected as the final model for deployment.

5.4 Error Analysis

Even though the accuracy was high, some misclassifications occurred:

- Some diseases had similar symptoms in the dataset (e.g., yellow spots or curled leaves)

- Certain disease classes were underrepresented, which made it harder for the model to learn those patterns

To fix this, the **SMOTE technique** was applied. It creates synthetic (artificial) examples of underrepresented classes to balance the dataset, which helped improve model performance significantly.

5.5 Usability and User Testing

Real users (farmers, agricultural students, or simulated users) tested the system. Their feedback included:

- The interface was clean and easy to navigate
- The classification results made sense and were well-labeled
- Pesticide recommendations helped guide actions without needing expert intervention

CHAPTER 6: RESULTS

6.1 Implementation Description

This research is a comprehensive Python application built using Tkinter for the graphical user interface (GUI) and integrates several modules to support a complete machine learning workflow for classifying plant diseases based on sparse and categorical IoT data. Below is an overview of its main components and functionality:

1. Imports and Libraries

- **Tkinter & PIL:** The application uses Tkinter for the GUI and the PIL (Pillow) library for handling images (such as background images).
- **Data Handling & Visualization:** It employs Pandas and NumPy for data manipulation and Matplotlib, Seaborn for plotting various types of charts (bar plots, violin plots, histograms, scatter plots, heatmaps, etc.).
- **Machine Learning and Data Preprocessing:** Scikit-learn modules (like train_test_split, classifiers such as GaussianNB, SVC, KNeighborsClassifier, and DecisionTreeClassifier) are used for model training and evaluation. Additionally, SMOTE from the imblearn library is applied to address class imbalance.
- **Database Interaction:** The application uses pymysql to connect to a MySQL database for handling user authentication (signup and login).
- **Joblib:** For model persistence, joblib is used to save and load trained machine learning models.

2. Data Exploration and Visualization

The code includes multiple functions for Exploratory Data Analysis (EDA):

- **Plotting Functions:**
  - **bar_plot:** Plots mean values of a numerical column grouped by the target label.
  - **violin_plot:** Creates violin plots to show the distribution of numerical data per label.
  - **strip_plot:** Uses a strip plot to show data points jittered across the labels.

    - **histogram:** Draws histograms to show the frequency distribution for each label.
    - **scatter_plot:** Generates scatter plots for two selected numerical columns, colored by label.
    - **correlation_heatmap:** Visualizes correlations between numerical features with a heatmap.
- **EDA Function:** The EDA() function sequentially calls the plotting functions (using columns such as "temp", "leafspot-size", "plant-growth", "germination", and "stem") to provide a comprehensive overview of the dataset.

3. Data Preprocessing

- **Dataset Upload:** The uploadDataset() function lets the user load a CSV file, displays its path and content in the text widget, and stores it as a Pandas DataFrame.
- **Label Encoding:** In DatasetPreprocessing(), non-numeric (categorical) columns are converted into numeric values using LabelEncoder so that they are compatible with machine learning algorithms. The target variable ('Label') is separated from the features.
- **Handling Missing Values:** The code replaces missing values using fillna(0).

4. Handling Imbalanced Data with SMOTE

- **Dataset_SMOTE():**
  This function applies SMOTE to oversample the minority classes in the dataset. It then compares the class distributions before and after applying SMOTE by plotting bar charts for both cases.

5. Training and Evaluating Machine Learning Models

- **Train-test Split:** The Train_test_splitting() function splits the preprocessed data into training (80%) and testing (20%) subsets.
- **Classifier Functions:** The code provides several classifier implementations:
    - **Existing_Classifier() for GaussianNB:** Loads a saved model (if available) or trains a new one.

  - **Existing_Classifier1() for SVM (SVC):** Similar approach for support vector machine.
  - **Proposed_Classifier() for K-Nearest Neighbors:** Trains or loads a KNN model.
  - **Proposed_Classifier1() for Decision Tree Classifier:** Trains or loads a decision tree model.
- **Performance Metrics:** The calculateMetrics() function computes precision, recall, f1 score, and accuracy. It also generates a confusion matrix for visual evaluation of model performance.

6. Prediction Functionality

- **predict():**
  After training, the application can load a new test CSV file and use the trained classifier to predict the disease label for each sample. It then prints out:
  - The test sample data.
  - The classified disease label (using the predefined labels list).
  - A corresponding suggested pesticide (using the predefined pesticides list).

7. User Authentication with Database Integration

- **Database Connection:** The connect_db() function connects to a local MySQL database named sparse_db.
- **Signup and Login:** Functions signup() and login() provide separate windows for user registration and login. Users are assigned roles (either "Admin" or "User"), and based on their role, different sets of functionalities (buttons) are shown:
  - **Admin:** Has access to dataset upload, data analytics, preprocessing, SMOTE, data splitting, and training various models.
  - **User:** Can access prediction functionality.
- **UI Control:** The functions show_admin_buttons() and show_user_buttons() manage the display of appropriate buttons depending on the user's role.

8. Graphical User Interface (GUI) Setup

- **Main Window:** A full-screen Tkinter window is created with a background image. The title label displays the application title:

"Machine Learning Adopted Multi-class Classification of Plant Diseases with Sparse and Categorical IoT Data"

- **Text Widget:** A text widget is used to display logs, dataset content, and results of different operations.
- **Buttons:** Buttons for signup and login (for both Admin and User) are placed at the top, while functionality-specific buttons are dynamically added after authentication.

6.2 Dataset Description

The dataset is designed for multi-class classification of plant diseases. It contains various environmental and plant-specific measurements, along with a target column indicating the disease label. The dataset is provided as a CSV (Comma-Separated Values) file, making it easy to load and manipulate using data analysis libraries such as Pandas.

In essence, this dataset combines sparse and categorical IoT data with plant-specific measurements to facilitate the multi-class classification of plant diseases. It has been structured to handle 18 disease categories, each reflecting a distinct pathology. The ultimate goal is to accurately predict plant diseases and suggest appropriate pesticide treatments, thereby enhancing both the efficiency and effectiveness of disease management in agricultural settings.

Features (Columns)

1. **temp**
   - *Type:* Numeric (float or integer).
   - *Description:* Represents temperature measurements from IoT sensors or manual observations.

2. **leafspot-size**
   - *Type:* Numeric.

    - *Description:* Indicates the size (or extent) of leaf spots, which is a common symptom for many plant diseases.

3. **plant-growth**
    - *Type:* Numeric.
    - *Description:* A measure (possibly on a certain scale) indicating overall plant growth or vigor.
4. **germination**
    - *Type:* Numeric.
    - *Description:* Reflects germination rate or related metric that captures seed viability or early growth.
5. **stem**
    - *Type:* Numeric.
    - *Description:* May represent stem diameter, length, or a damage index relevant to disease symptoms.
6. **Label**
    - *Type:* Categorical (converted to numeric via label encoding).
    - *Description:* The **target variable** representing the disease category. The code suggests **18 different disease classes**, including (but not limited to) “diaporthe-stem-canker,” “charcoal-rot,” “powdery-mildew,” “downy-mildew,” “bacterial-blight,” and others.

Depending on the data collection approach, there could be additional features capturing environmental conditions (e.g., humidity, rainfall) or plant attributes (e.g., leaf color, texture). However, the code specifically highlights the above columns for Exploratory Data Analysis (EDA) and model training.

3. Target Classes

The dataset appears to have **18 disease categories**, each labeled numerically after preprocessing. Some of the referenced diseases include:

- diaporthe-stem-canker
- charcoal-rot
- rhizoctonia-root-rot
- phytophthora-rot
- brown-stem-rot
- powdery-mildew
- downy-mildew
- brown-spot
- bacterial-blight
- bacterial-pustule
- purple-seed-stain
- anthracnose
- phyllosticta-leaf-spot
- alternarialeaf-spot
- frog-eye-leaf-spot
- diaporthe-pod-&-stem-blight
- cyst-nematode
- herbicide-injury

These labels are encoded into numeric form for training machine learning algorithms.

6.3 Results Analysis

Fig. 6.1 shows the overall design of the graphical user interface (GUI) for the project. It highlights the layout of the main dashboard where users can interact with multiple functionalities—including data uploading, visualization, and model prediction—with an innovative design tailored for multi-class classification of plant diseases. The interface integrates sections for input, processing feedback, and pesticide recommendations based on sparse and categorical IoT data.

Fig. 6.2**,** in this figure, the admin signup interface is displayed. It shows a dedicated window where new administrative users can enter their username and password. The design focuses on simplicity and clarity, ensuring that admins can quickly register their credentials before accessing the system's management features. Similar to the admin signup, Fig. 6.3 presents the signup interface for regular users. It allows users to create an account by providing necessary credentials. The interface is user-friendly, ensuring that even non-technical users can sign up with ease.

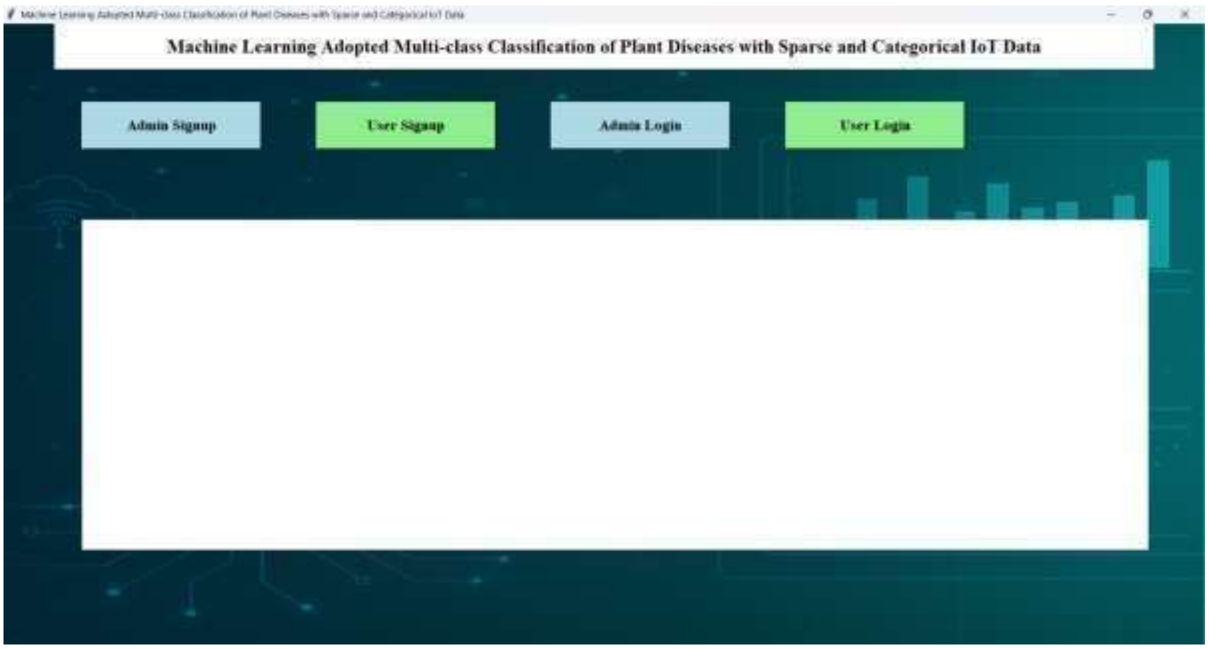
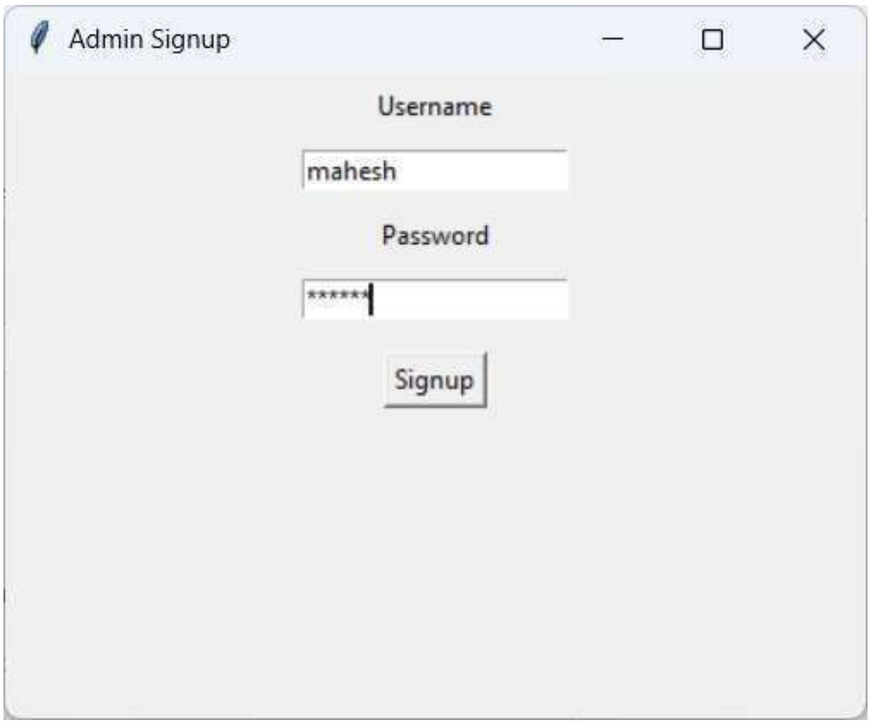

Machine Learning Adopted Multi-class Classification of Plant Diseases with Sparse and Categorical IoT Data
Admin Signup
User Signup
Admin Login
User Login

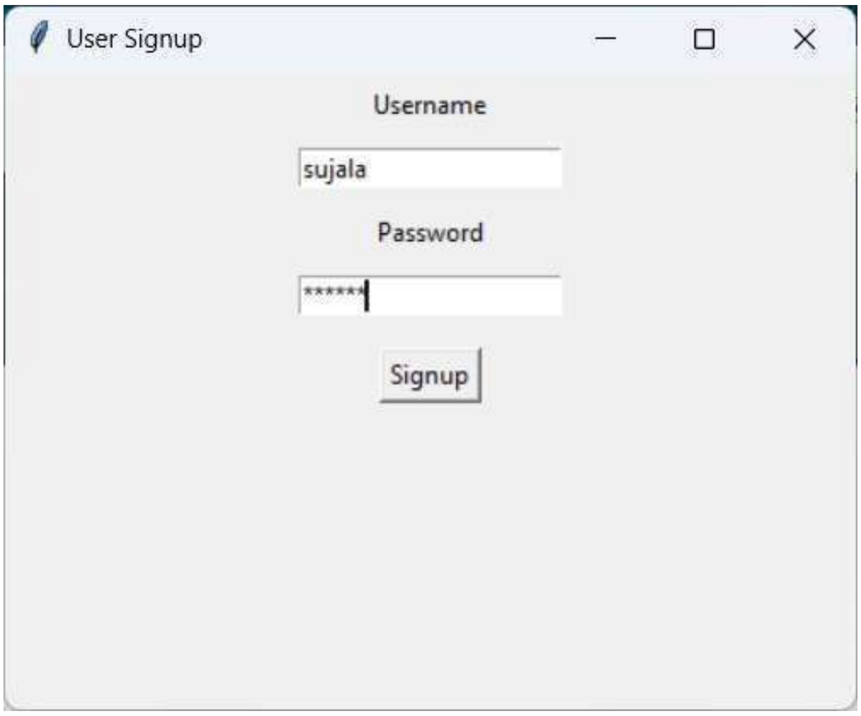
Fig. 6.1: Illustration of GUI application designed for multi-class classification of plant diseases with pesticide suggestion from sparse and categorical IoT data.


Admin Signup

Username: mahesh

Password: ******

Fig. 6.2: Admin signup interface.


User Signup

Username: sujala

Password: ******

Fig. 6.3: User signup interface.

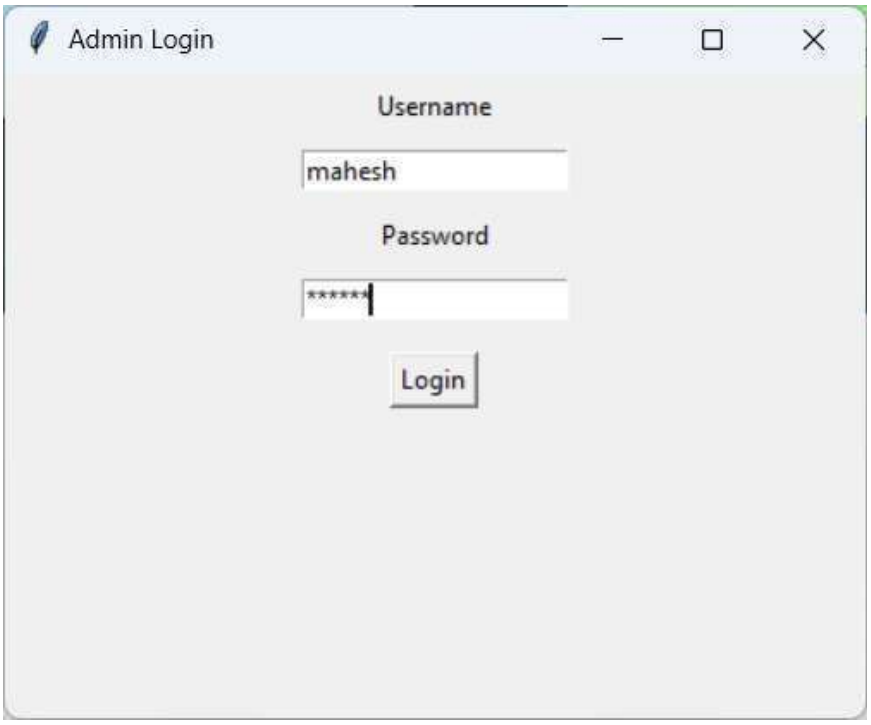
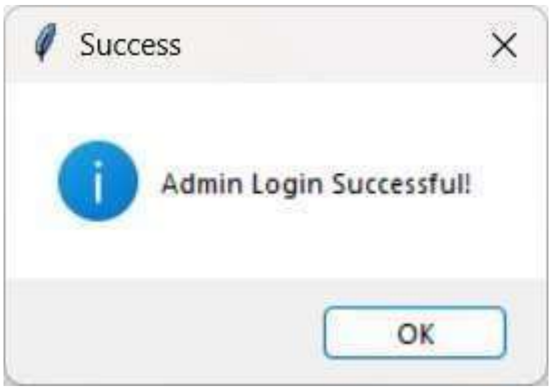


Fig. 6.4: Admin login interface.



Admin Login Successful!

Fig. 6.5: Successful login interface.

Fig. 6.4 illustrates the login screen designed for administrators. It features fields for entering the username and password, along with clear prompts and buttons that guide the admin through the authentication process, ensuring secure access to the advanced functionalities of the system.

Fig. 6.5 demonstrate after a successful login, this interface confirms that the credentials are valid, and access is granted. It often shows a welcome message or transition screen that confirms the user has been authenticated, providing immediate feedback to the admin.

In Fig. 6.6, the GUI displays additional buttons and features that are exclusive to the admin role. Options such as uploading a dataset, performing data preprocessing, applying SMOTE, splitting data, and training different machine learning models become available. This screen demonstrates the expanded functionality accessible to admins.

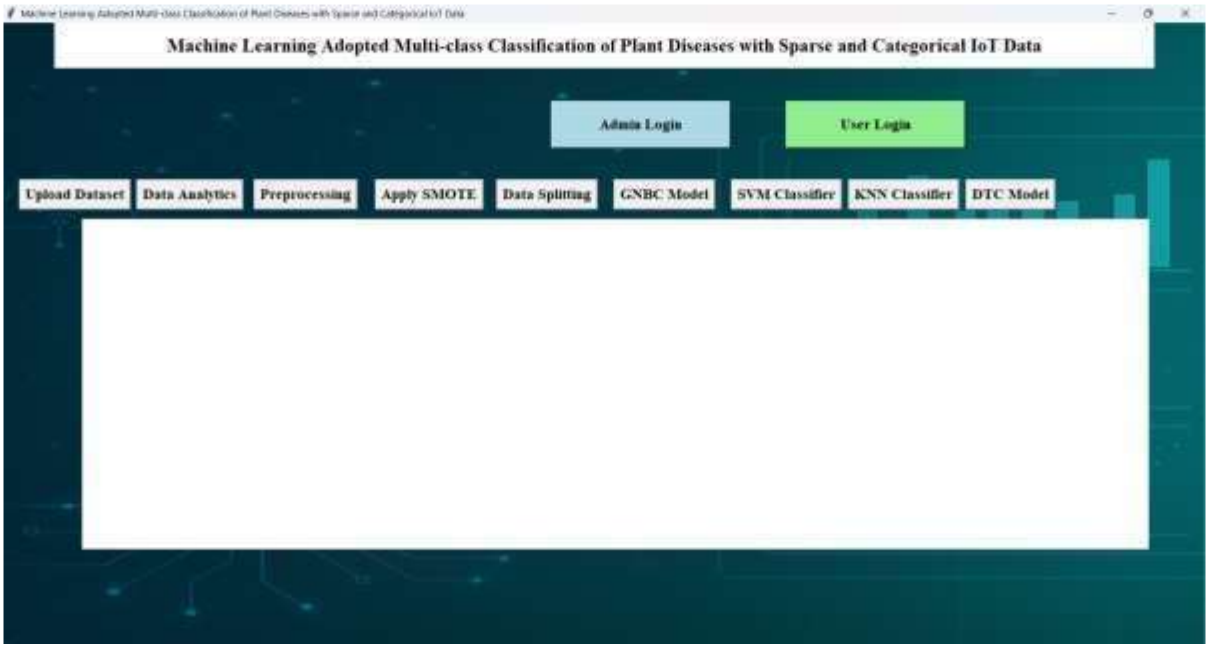
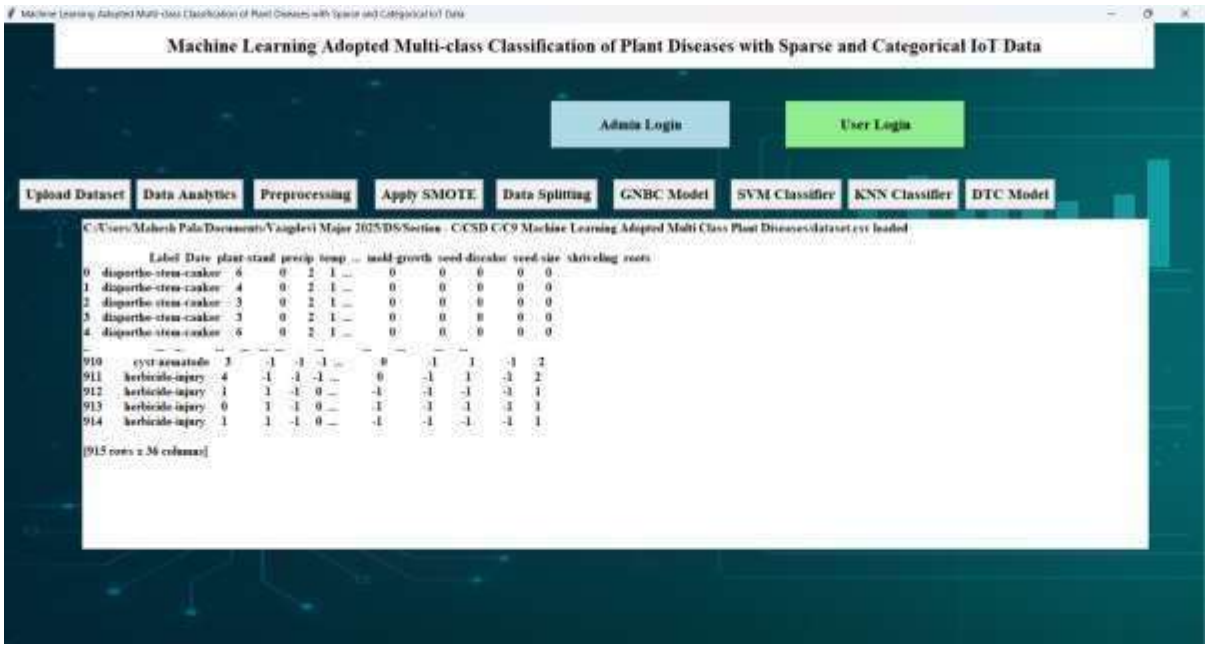


Fig. 6.6: Illustration of GUI application after successful login of admin.



```
C:/Users/Mahesh Pala/Documents/Vaagdevi Major 2025/DS/Section - C/CSD C/C9 Machine Learning Adopted Multi Class Plant Diseases/dataset.csv loaded

                  Label  Date  plant-stand  precip  temp  ...  mold-growth  seed-discolor  seed-size  shriveling  roots
0    diaporthe-stem-canker     6            0       2     1  ...            0              0          0           0      0
1    diaporthe-stem-canker     4            0       2     1  ...            0              0          0           0      0
2    diaporthe-stem-canker     3            0       2     1  ...            0              0          0           0      0
3    diaporthe-stem-canker     3            0       2     1  ...            0              0          0           0      0
4    diaporthe-stem-canker     6            0       2     1  ...            0              0          0           0      0
..                     ...   ...          ...     ...   ...  ...          ...            ...        ...         ...    ...
910          cyst-nematode     3           -1      -1    -1  ...            0             -1          1          -1      2
911        herbicide-injury    4           -1      -1    -1  ...            0             -1          1          -1      2
912        herbicide-injury    1            1      -1     0  ...           -1             -1         -1          -1      1
913        herbicide-injury    0            1      -1     0  ...           -1             -1         -1          -1      1
914        herbicide-injury    1            1      -1     0  ...           -1             -1         -1          -1      1

[915 rows x 36 columns]
```

Fig. 6.7: Illustration of GUI application after uploading the dataset.

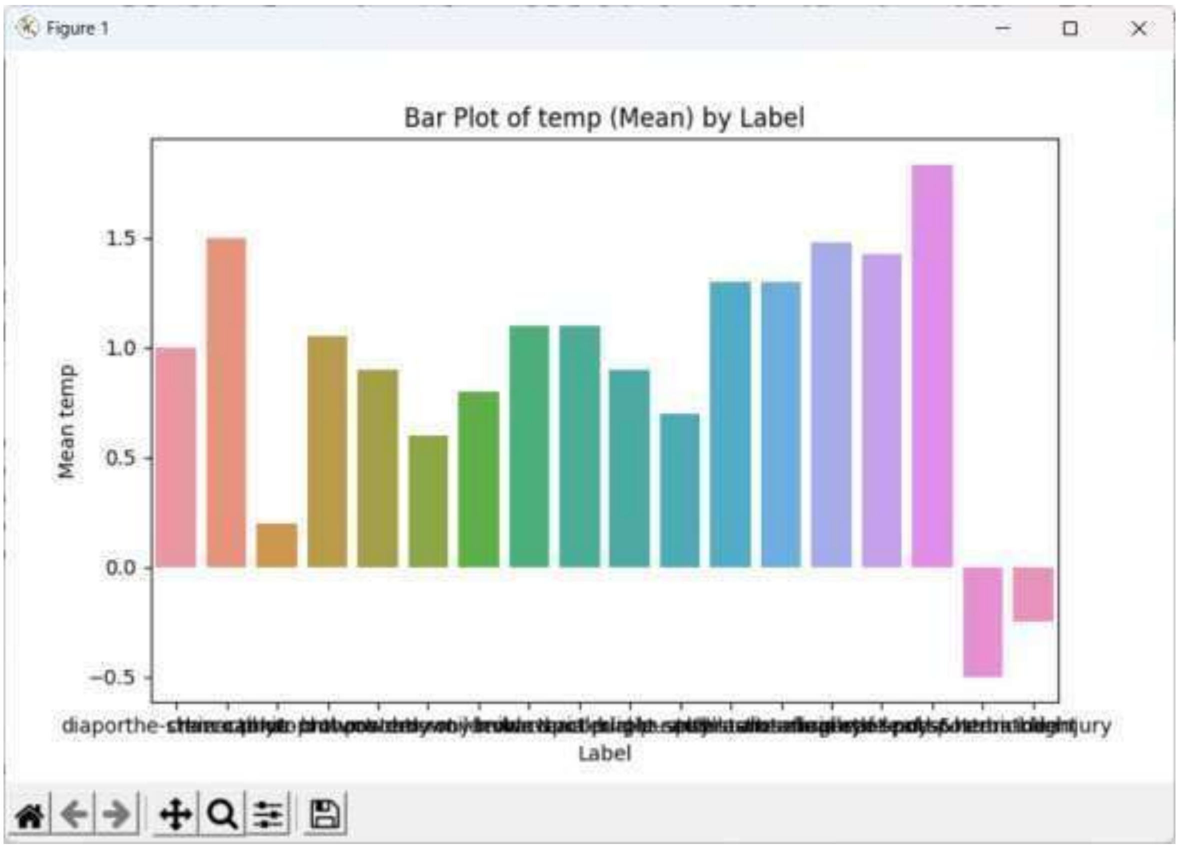


Figure 1
Bar Plot of temp (Mean) by Label
Mean temp
1.5
1.0
0.5
0.0
−0.5
Label

(a)

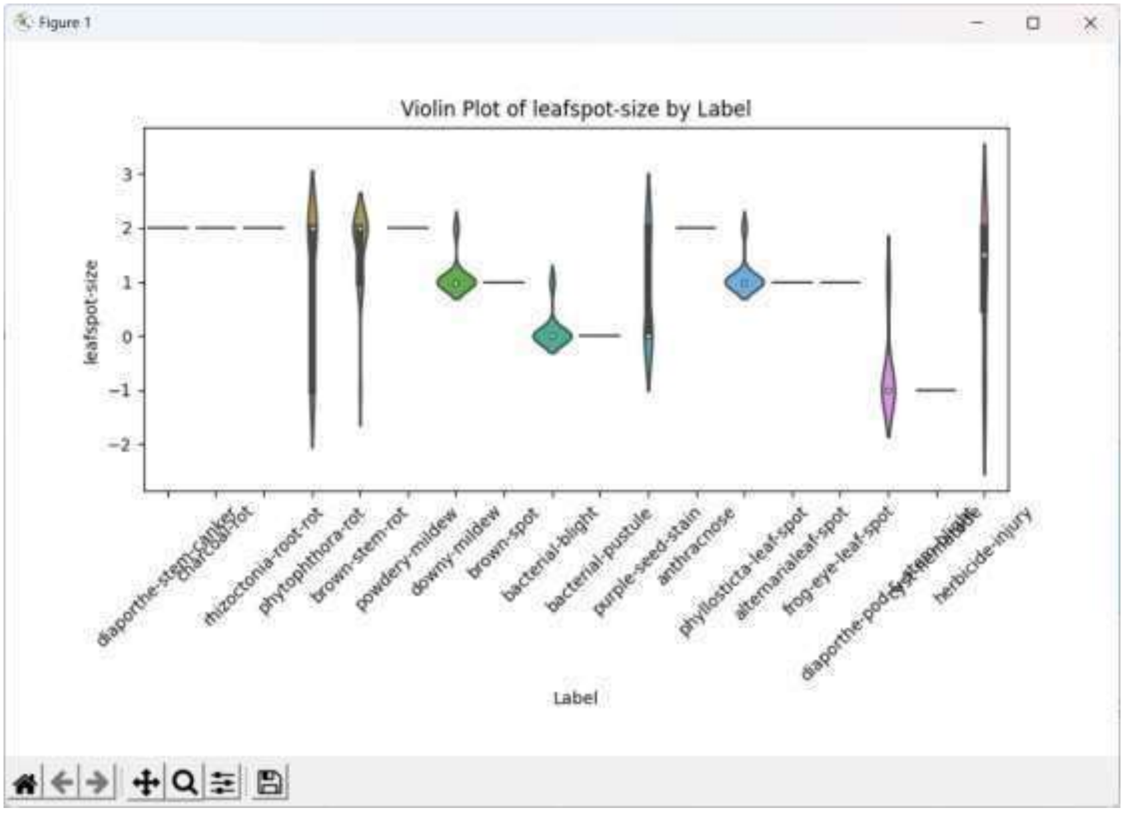


Figure 1
Violin Plot of leafspot-size by Label
leafspot-size
3
2
1
0
−1
−2
diaporthe-stem-canker
charcoal-rot
rhizoctonia-root-rot
phytophthora-rot
brown-stem-rot
powdery-mildew
downy-mildew
brown-spot
bacterial-blight
bacterial-pustule
purple-seed-stain
anthracnose
phyllosticta-leaf-spot
alternarialeaf-spot
frog-eye-leaf-spot
diaporthe-pod-&-stem-blight
herbicide-injury
Label

(b)

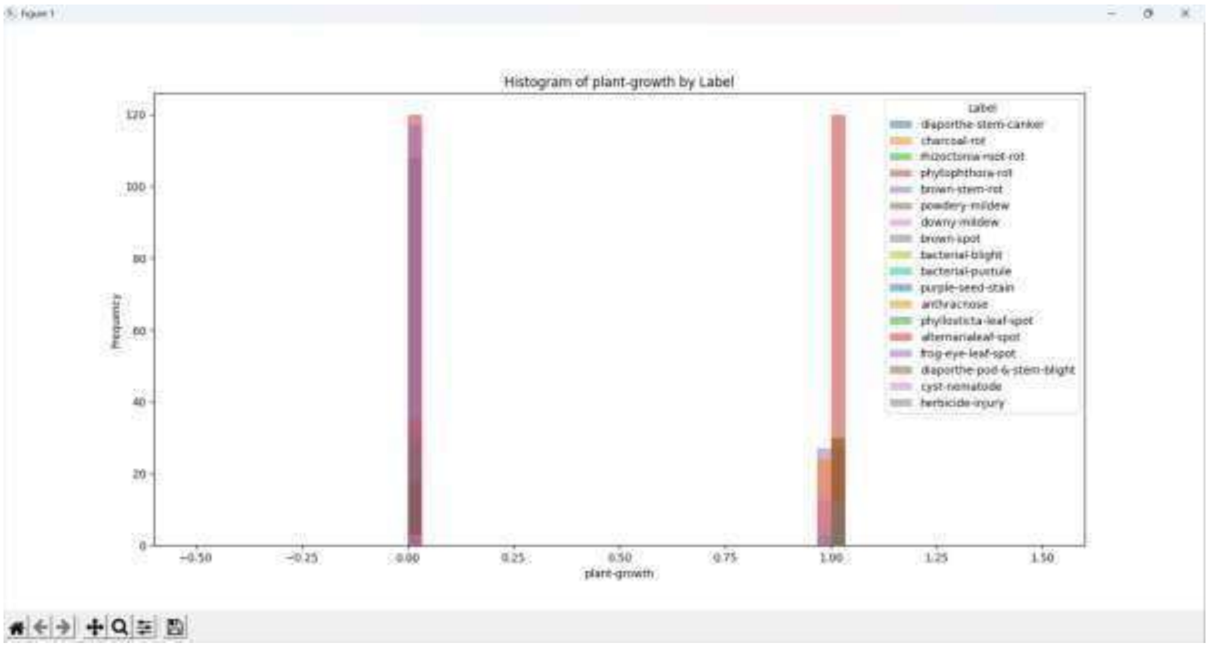


Figure 1
Histogram of plant-growth by Label
Frequency
plant-growth
Label
diaporthe-stem-canker
charcoal-rot
rhizoctonia-root-rot
phytophthora-rot
brown-stem-rot
powdery-mildew
downy-mildew
brown-spot
bacterial-blight
bacterial-pustule
purple-seed-stain
anthracnose
phyllosticta-leaf-spot
alternarialeaf-spot
frog-eye-leaf-spot
diaporthe-pod-&-stem-blight
cyst-nematode
herbicide-injury

(c)

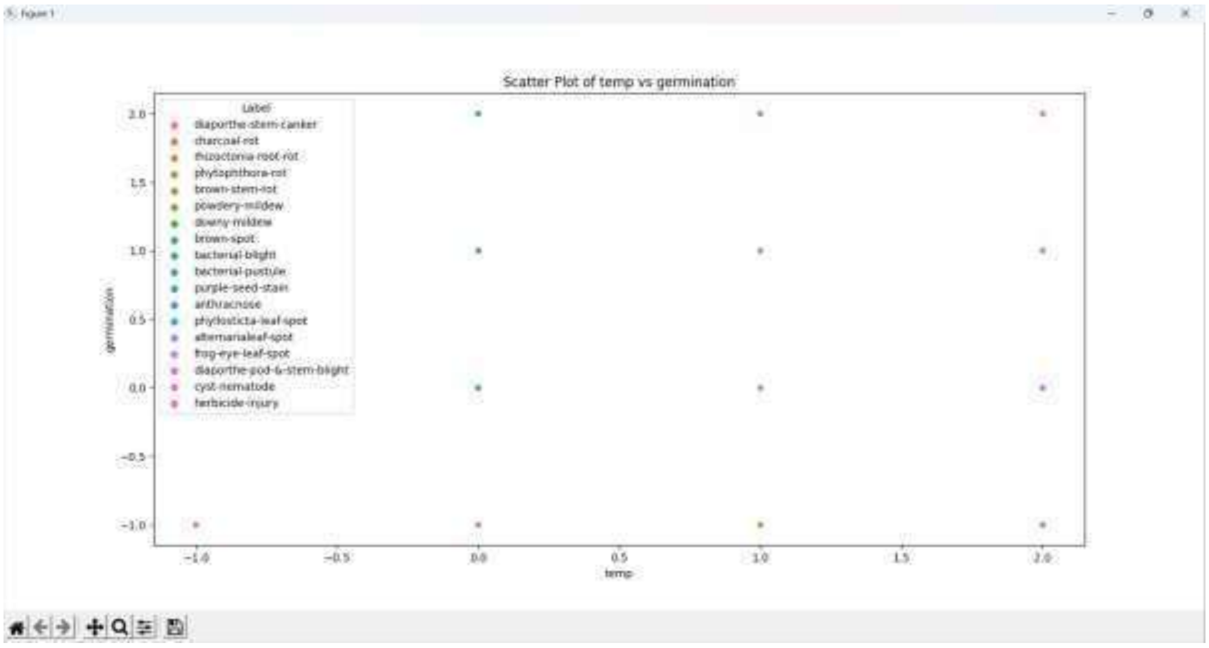


Figure 1
Scatter Plot of temp vs germination
germination
temp
Label
diaporthe-stem-canker
charcoal-rot
rhizoctonia-root-rot
phytophthora-rot
brown-stem-rot
powdery-mildew
downy-mildew
brown-spot
bacterial-blight
bacterial-pustule
purple-seed-stain
anthracnose
phyllosticta-leaf-spot
alternarialeaf-spot
frog-eye-leaf-spot
diaporthe-pod-&-stem-blight
cyst-nematode
herbicide-injury

(d)

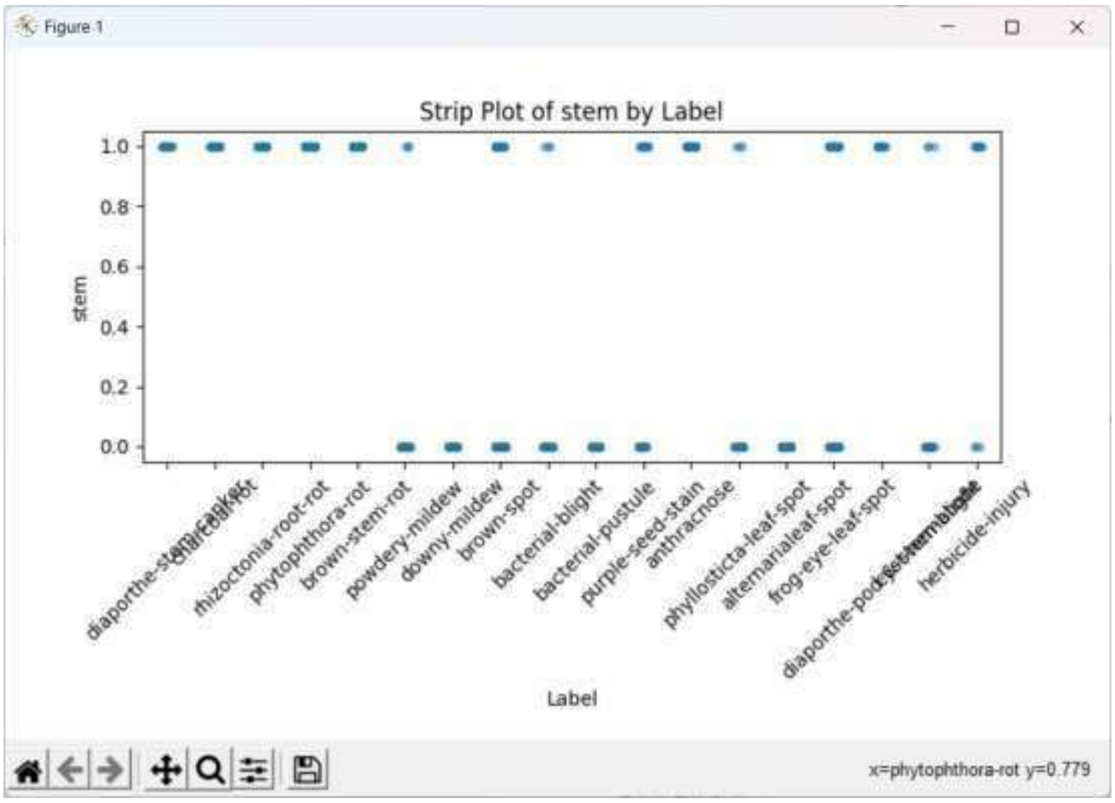


Figure 1
Strip Plot of stem by Label
stem
1.0
0.8
0.6
0.4
0.2
0.0
diaporthe-stem-canker
charcoal-rot
rhizoctonia-root-rot
phytophthora-rot
brown-stem-rot
powdery-mildew
downy-mildew
brown-spot
bacterial-blight
bacterial-pustule
purple-seed-stain
anthracnose
phyllosticta-leaf-spot
alternarialeaf-spot
frog-eye-leaf-spot
diaporthe-pod-&-stem-blight
herbicide-injury
Label
x=phytophthora-rot y=0.779

(e)

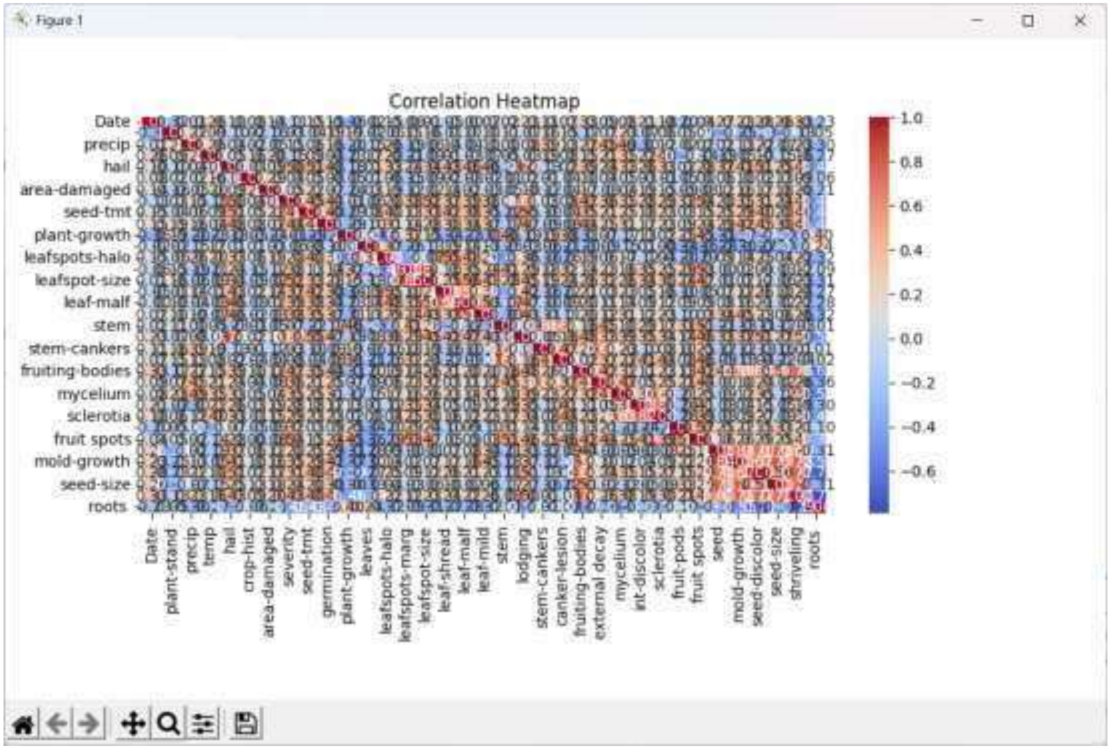
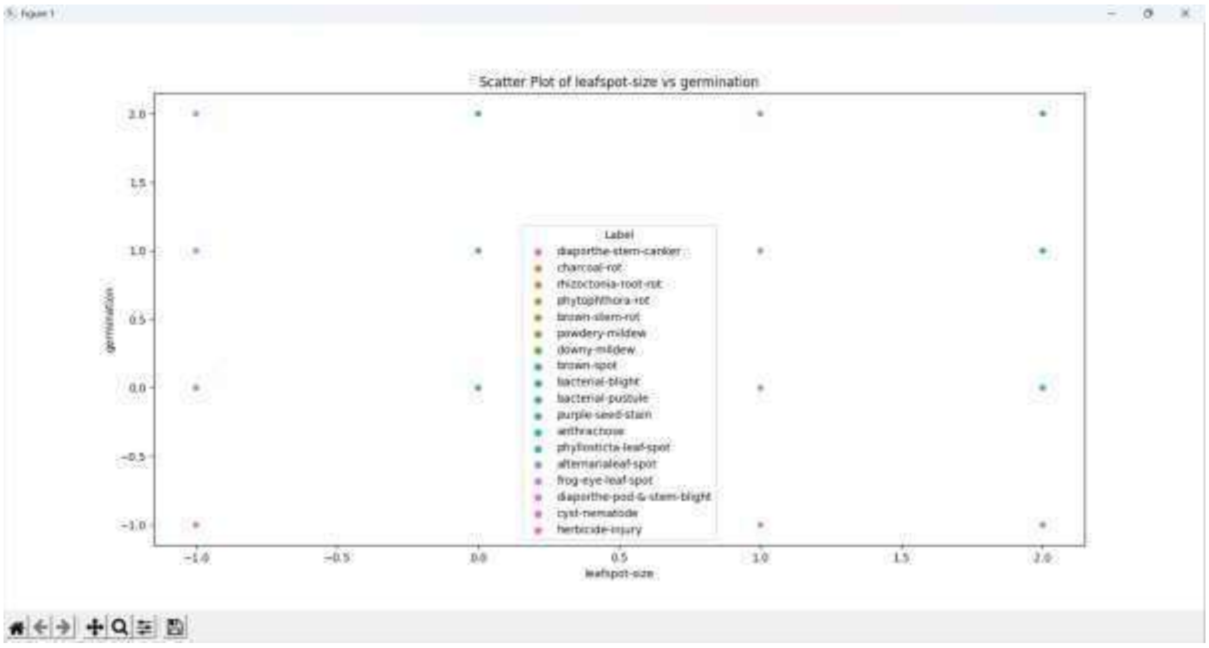


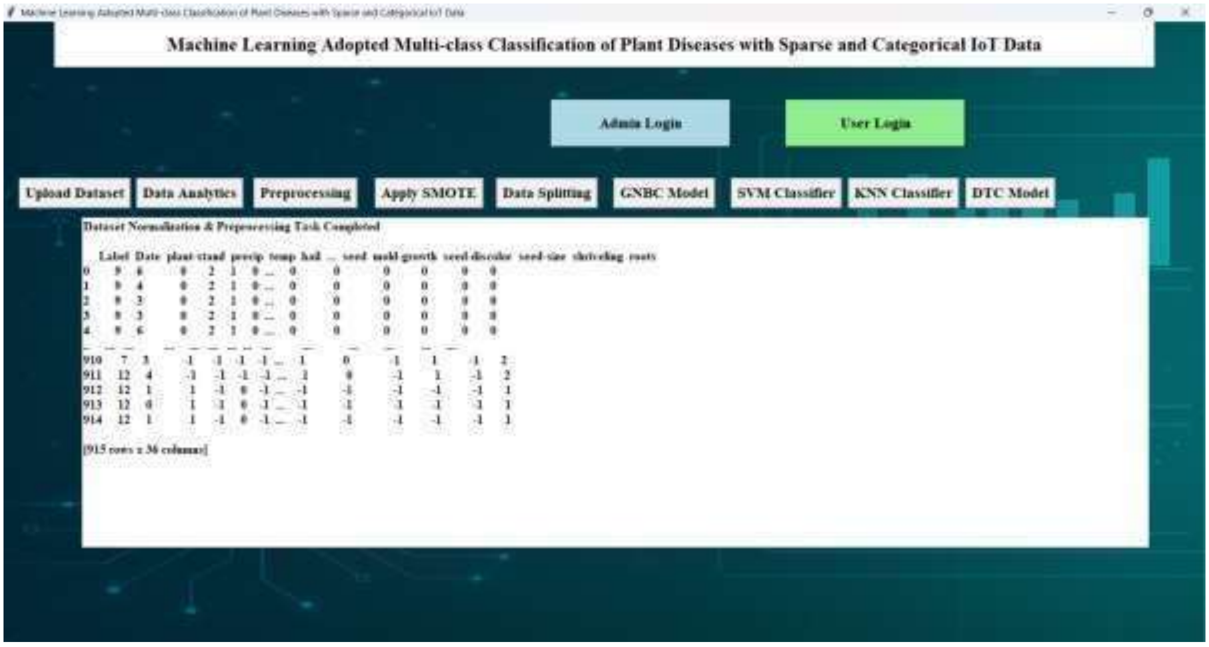
Figure 1
Correlation Heatmap
Date
precip
hail
area-damaged
seed-tmt
plant-growth
leafspots-halo
leafspot-size
leaf-malf
stem
stem-cankers
fruiting-bodies
mycelium
sclerotia
fruit spots
mold-growth
seed-size
roots
Date
plant-stand
precip
temp
hail
crop-hist
area-damaged
severity
seed-tmt
germination
plant-growth
leaves
leafspots-halo
leafspots-marg
leafspot-size
leaf-shread
leaf-malf
leaf-mild
stem
lodging
stem-cankers
canker-lesion
fruiting-bodies
external decay
mycelium
int-discolor
sclerotia
fruit-pods
fruit spots
seed
mold-growth
seed-discolor
seed-size
shriveling
roots
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

(f)



(g)

Fig. 6.8: Displaying the data visualization with various plots. (a) Bar plot. (b) Violin plot. (c) histogram plot. (d) scatter plot. (e) strip plot. (f) correlation heat map. (g) scatter plot.


Machine Learning Adopted Multi-class Classification of Plant Diseases with Sparse and Categorical IoT Data
Machine Learning Adopted Multi-class Classification of Plant Diseases with Sparse and Categorical IoT Data
Admin Login
User Login
Upload Dataset
Data Analytics
Preprocessing
Apply SMOTE
Data Splitting
GNBC Model
SVM Classifier
KNN Classifier
DTC Model

```
Dataset Normalization & Preprocessing Task Completed

    Label  Date  plant-stand  precip  temp  hail  ...  seed  mold-growth  seed-discolor  seed-size  shriveling  roots
0       9     6            0       2     1     0  ...     0            0              0          0           0      0
1       9     4            0       2     1     0  ...     0            0              0          0           0      0
2       9     3            0       2     1     0  ...     0            0              0          0           0      0
3       9     3            0       2     1     0  ...     0            0              0          0           0      0
4       9     6            0       2     1     0  ...     0            0              0          0           0      0
..    ...   ...          ...     ...   ...   ...  ...   ...          ...            ...        ...         ...    ...
910     7     3           -1      -1    -1    -1  ...     1            0             -1          1          -1      2
911    12     4           -1      -1    -1    -1  ...     1            0             -1          1          -1      2
912    12     1            1      -1     0    -1  ...    -1           -1             -1         -1          -1      1
913    12     0            1      -1     0    -1  ...    -1           -1             -1         -1          -1      1
914    12     1            1      -1     0    -1  ...    -1           -1             -1         -1          -1      1

[915 rows x 36 columns]
```

Fig. 6.9: Displaying the GUI application after applying data preprocessing and normalization.

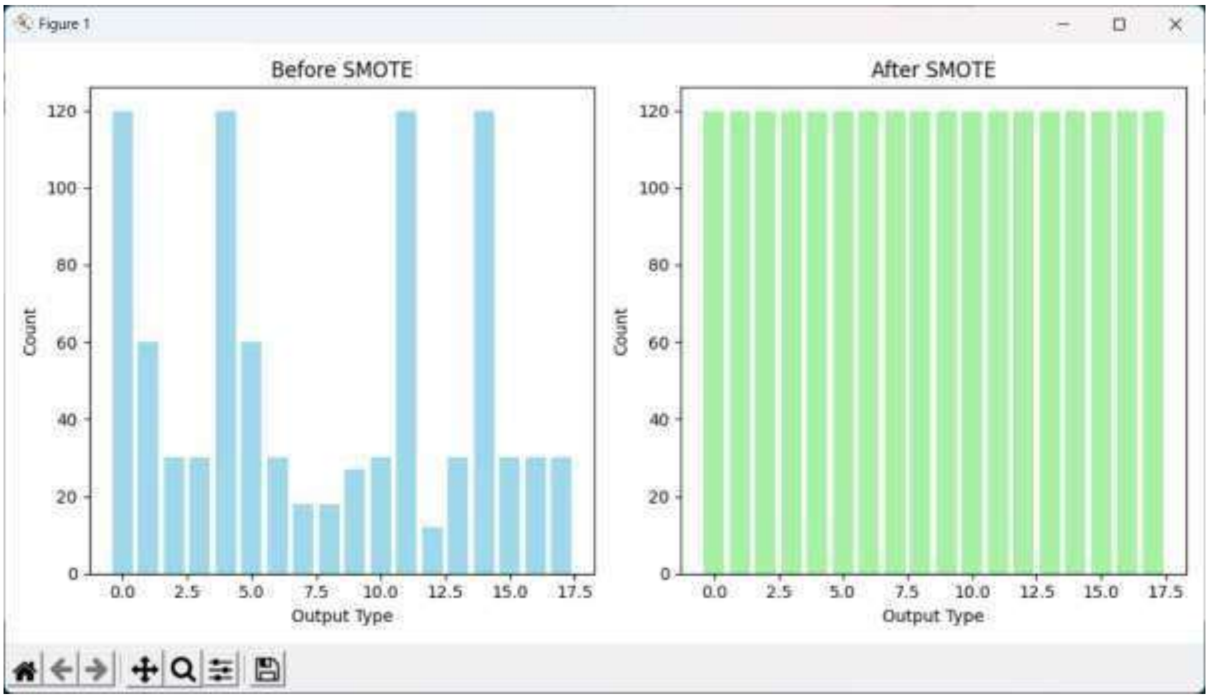
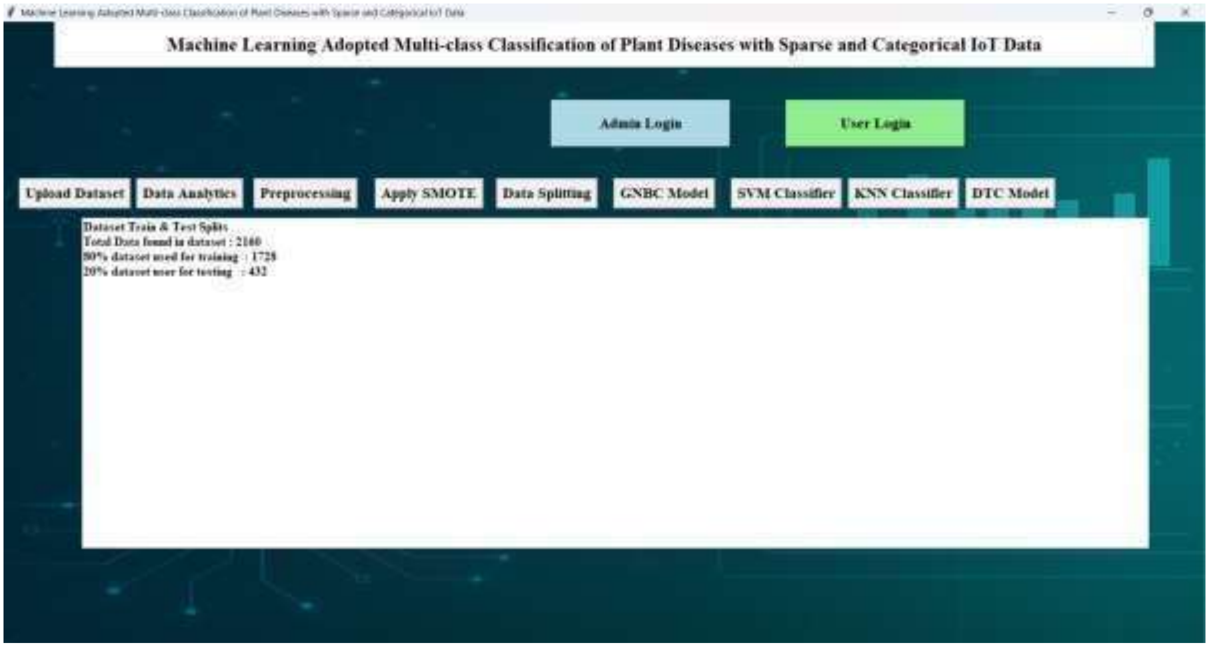


Fig. 6.10: Bar plot of count versus output before and after applying SMOTE algorithm.



```
Dataset Train & Test Splits
Total Data found in dataset : 2160
80% dataset used for training : 1728
20% dataset user for testing  : 432
```

Fig. 6.11: Illustration of GUI application after applying data splitting (80:20 ratio) operation.

Fig. 6.7 captures the state of the GUI immediately after an admin uploads a dataset. The file path is shown along with a preview or log of the dataset content in a text widget. It demonstrates how the system provides feedback about the dataset being loaded and ready for further processing.

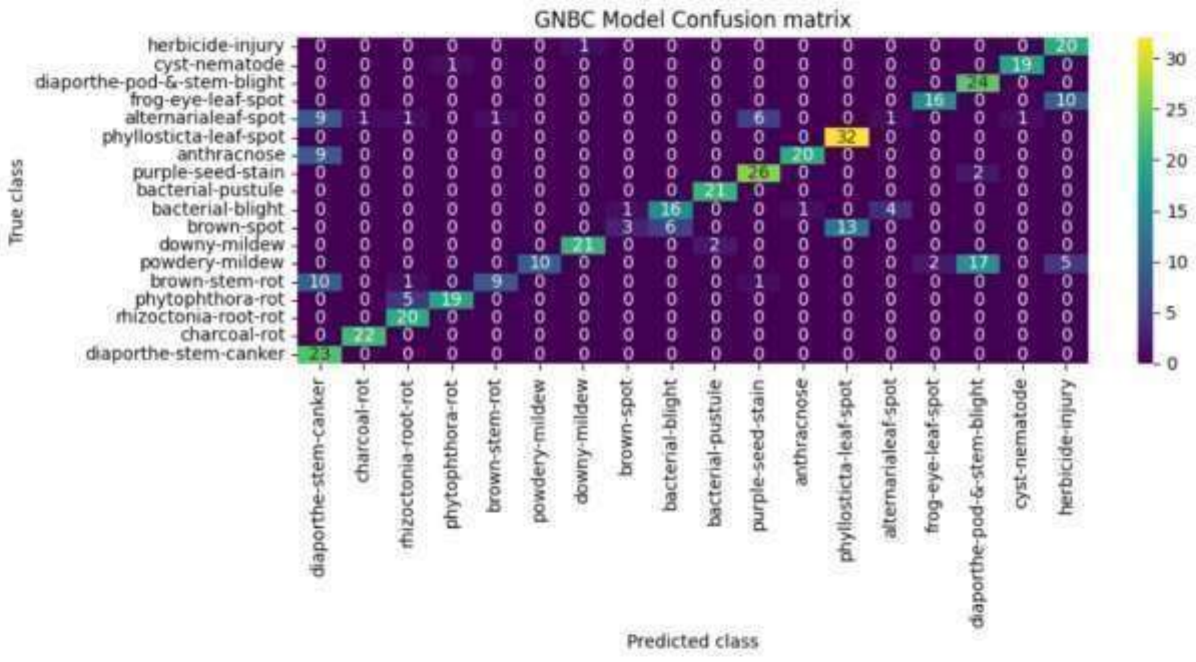
Fig. 6.8 is divided into sub-figures:

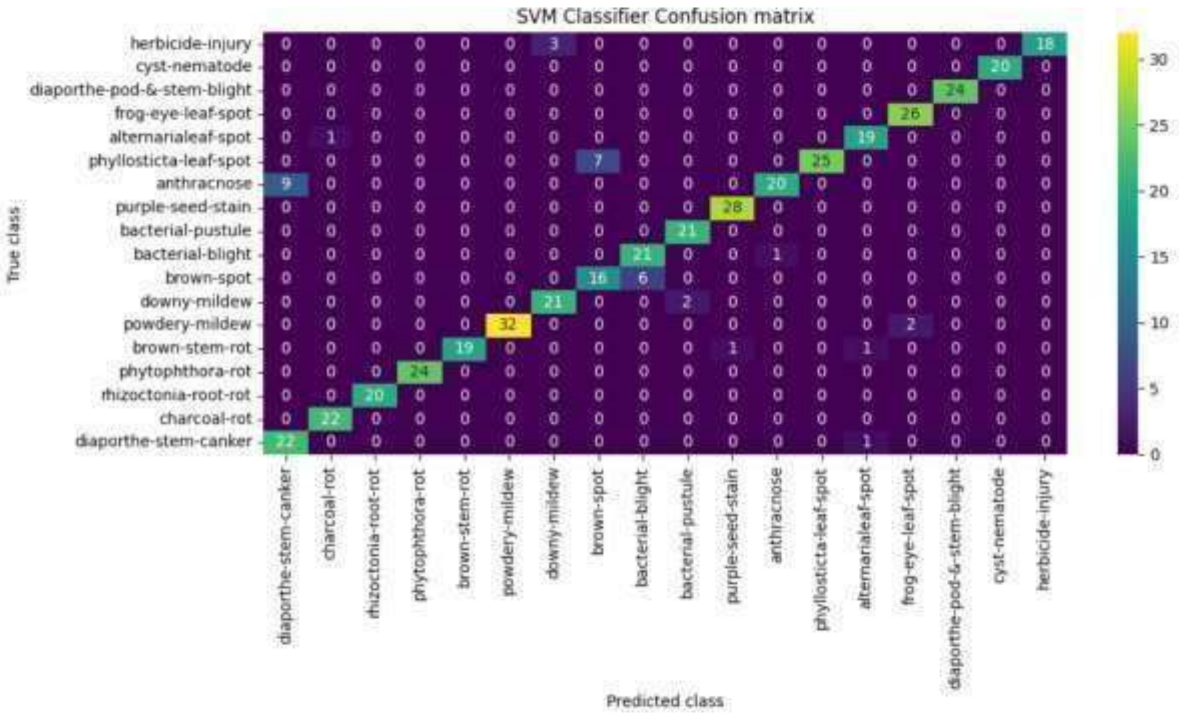
- **(a) Bar Plot:** Shows mean values of a selected numerical feature per disease label.
- **(b) Violin Plot:** Displays the distribution and density of a feature across disease labels.
- **(c) Histogram Plot:** Illustrates the frequency distribution of feature values.
- **(d) Scatter Plot:** Reveals relationships between two features with color-coded disease labels.
- **(e) Strip Plot:** Plots individual data points with jitter to show the spread within groups.
- **(f) Correlation Heat Map:** Visualizes pairwise correlations among numerical features using a color gradient.
- **(g) Another Scatter Plot:** Provides an additional perspective on the interaction between different variables.

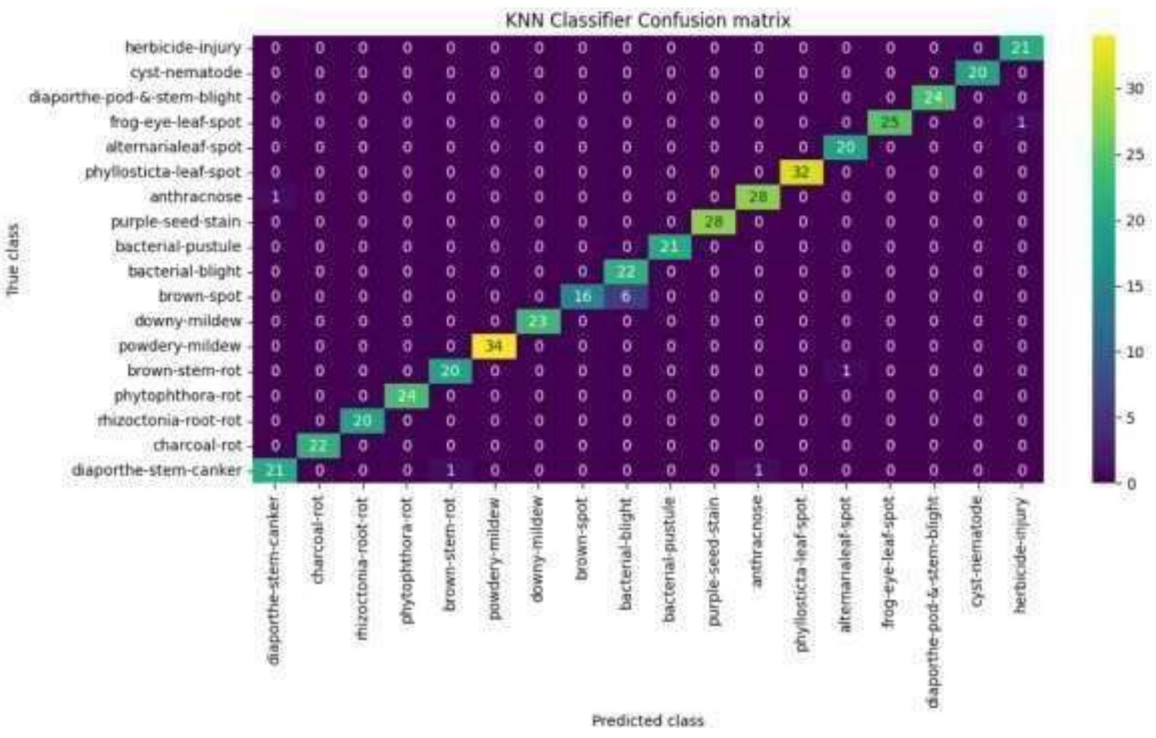
Fig. 6.9 depicts the GUI after preprocessing (including filling missing values and label encoding), the GUI updates to reflect that the dataset is now normalized and ready for further steps. This screen may show logs or status messages confirming that preprocessing has been successfully completed.

Fig. 6.10 contrasts the class distribution before and after the application of the SMOTE algorithm. Two bar plots are shown side-by-side: one displaying the imbalanced counts of each disease class, and the other illustrating the balanced class distribution achieved through synthetic oversampling.

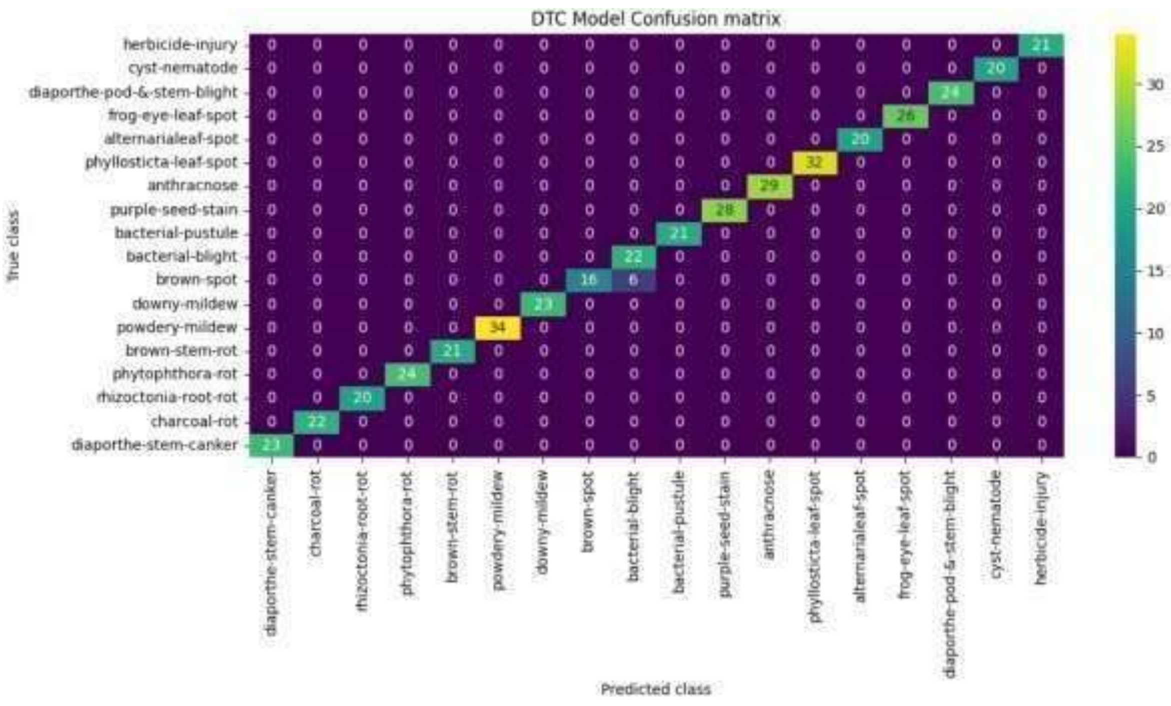
Fig. 6.11 presents the state of the GUI once the dataset has been split into training (80%) and testing (20%) sets. It may include logs detailing the number of samples allocated to each set, providing transparency into the data division process.


(a)


(b)



(c)



(d)

Fig. 6.12: Confusion matrices obtained using (a) GNBC model. (b) SVM classification. (c) KNN classification. (d) DTC model.

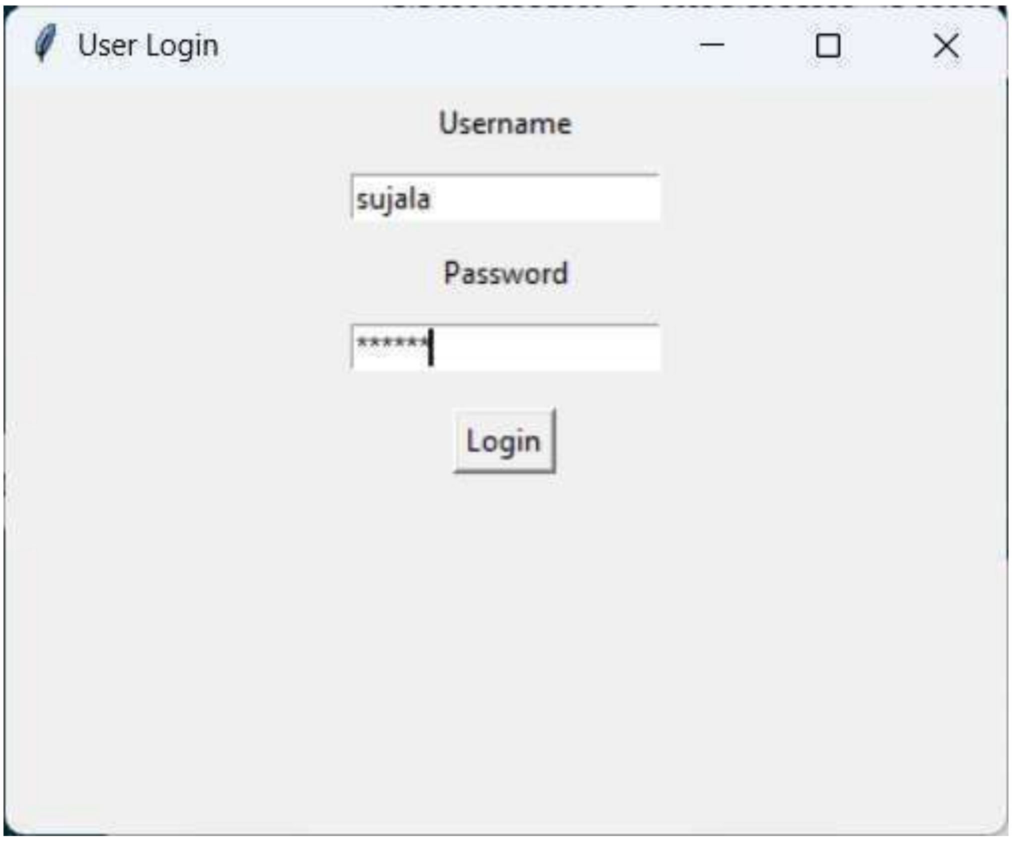


Fig. 6.13: User login interface.

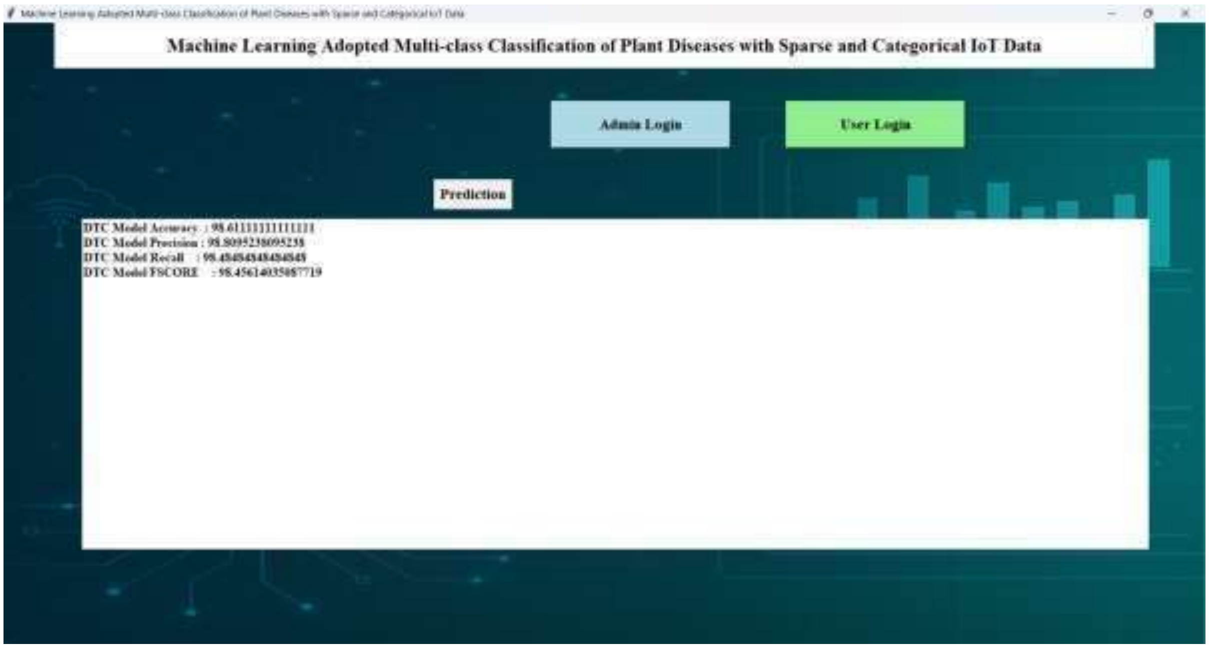
User Login Successful!

Fig. 6.14: Successful login of user.

Fig. 6.12 includes confusion matrices for:

- **(a) Gaussian Naive Bayes (GNBC) model:** Showing its classification performance.
- **(b) SVM Classifier:** Illustrating how well the SVM model differentiates between disease classes.
- **(c) KNN Classifier:** Visualizing the performance of the K-Nearest Neighbors algorithm.

- **(d) Decision Tree Classifier (DTC):** Depicting the confusion matrix of the decision tree model, which may show superior performance compared to the others.


Machine Learning Adopted Multi-class Classification of Plant Diseases with Sparse and Categorical IoT Data
Admin Login
User Login
Prediction

```
DTC Model Accuracy  : 98.61111111111111
DTC Model Precision : 98.8095238095238
DTC Model Recall    : 98.48484848484848
DTC Model FSCORE    : 98.45614035087719
```

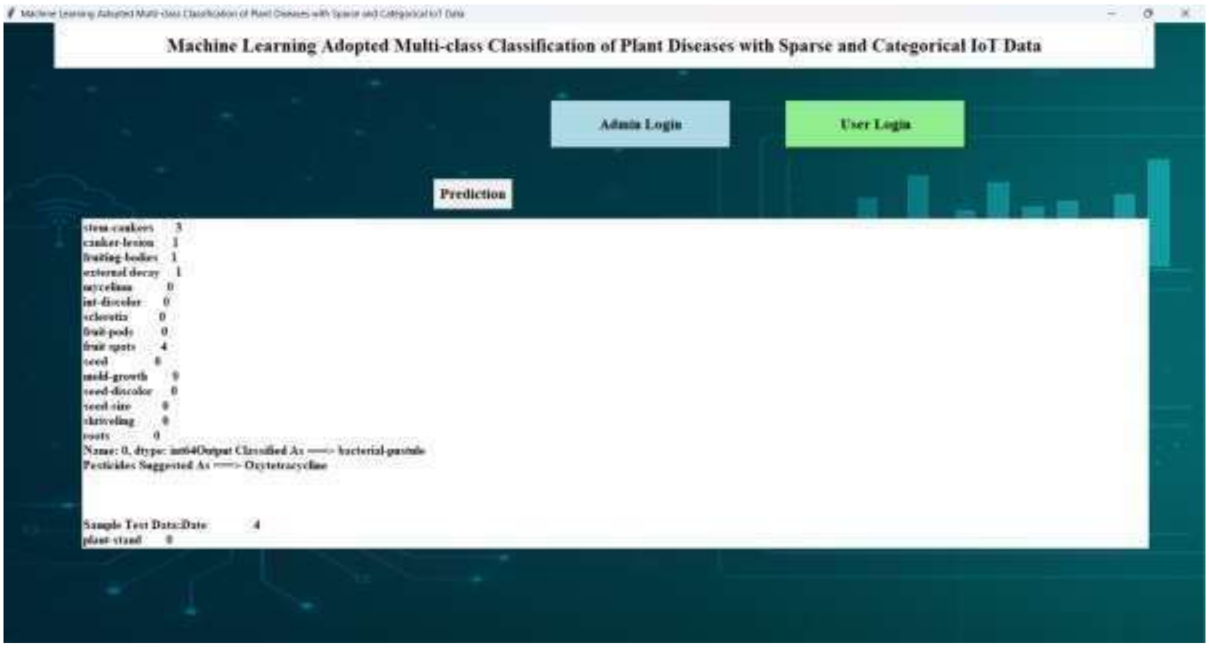
Fig. 6.15: GUI application after login as a user for prediction on test data.

Fig. 6.13 shows the login interface tailored for regular users. It mirrors the admin login interface but is designed for user-specific access, ensuring secure entry into the user functionalities of the application.

Upon successful authentication, this interface (as shown in Fig. 6.14) confirms that the user has been logged in. It might display a welcome message or transition to a new screen dedicated to prediction tasks, confirming that access has been granted.

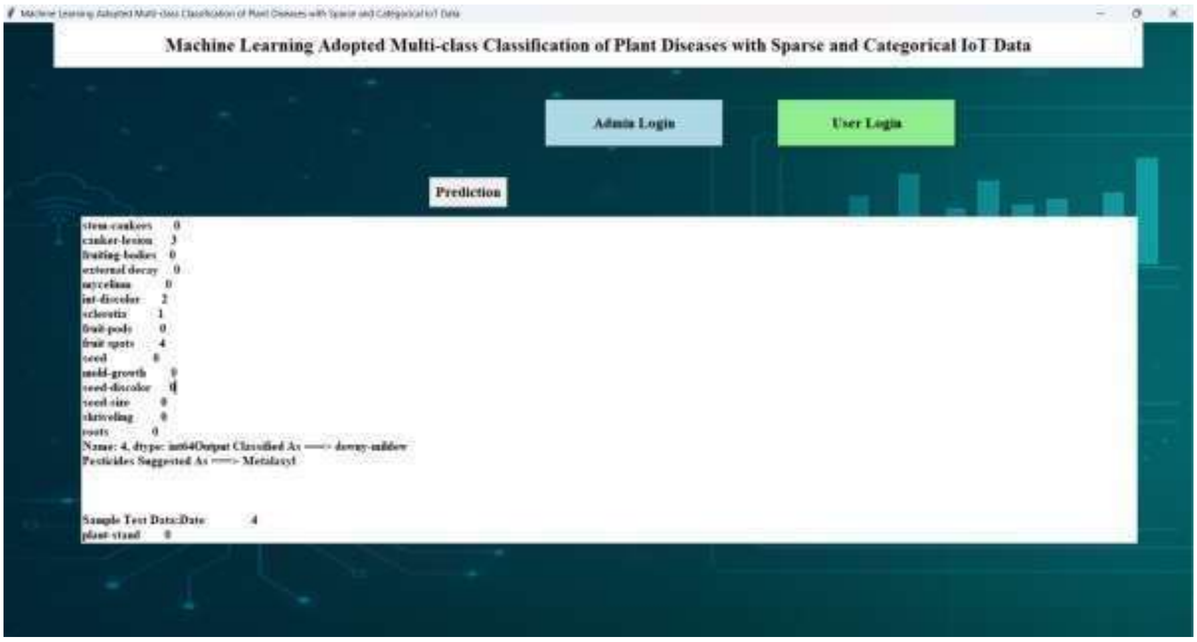
Fig. 6.15 illustrates the user-specific dashboard. After login, the user sees options to upload new test data and execute predictions. The interface is simplified to focus on prediction functionality rather than administrative tasks.

In the Fig. 6.16, the application displays the outcome of a prediction on a test dataset. It shows the classified disease and the corresponding pesticide recommendation, thereby completing the full cycle—from data input to actionable output for disease management.


Machine Learning Adopted Multi-class Classification of Plant Diseases with Sparse and Categorical IoT Data
Machine Learning Adopted Multi-class Classification of Plant Diseases with Sparse and Categorical IoT Data
Admin Login
User Login
Prediction

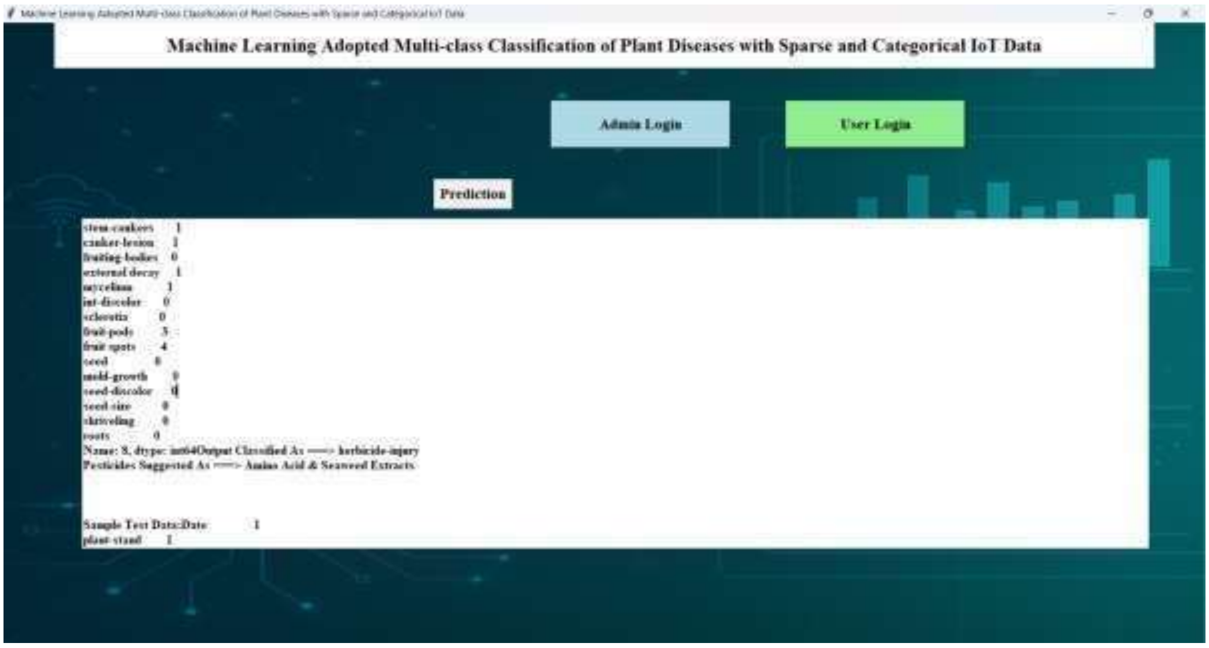
```
stem-cankers      3
canker-lesion     1
fruiting-bodies   1
external decay    1
mycelium          0
int-discolor      0
sclerotia         0
fruit-pods        0
fruit spots       4
seed              0
mold-growth       0
seed-discolor     0
seed-size         0
shriveling        0
roots             0
Name: 0, dtype: int64Output Classified As ----> bacterial-pustule
Pesticides Suggested As ----> Oxytetracycline


Sample Test Data:Date          4
plant-stand     0
```


Machine Learning Adopted Multi-class Classification of Plant Diseases with Sparse and Categorical IoT Data
Machine Learning Adopted Multi-class Classification of Plant Diseases with Sparse and Categorical IoT Data
Admin Login
User Login
Prediction

```
stem-cankers      0
canker-lesion     3
fruiting-bodies   0
external decay    0
mycelium          0
int-discolor      2
sclerotia         1
fruit-pods        0
fruit spots       4
seed              0
mold-growth       0
seed-discolor     0
seed-size         0
shriveling        0
roots             0
Name: 4, dtype: int64Output Classified As ----> downy-mildew
Pesticides Suggested As ----> Metalaxyl


Sample Test Data:Date          4
plant-stand     0
```


Machine Learning Adopted Multi-class Classification of Plant Diseases with Sparse and Categorical IoT Data
Machine Learning Adopted Multi-class Classification of Plant Diseases with Sparse and Categorical IoT Data
Admin Login
User Login
Prediction

```
stem-cankers     1
canker-lesion    1
fruiting-bodies   0
external decay    1
mycelium         1
int-discolor      0
sclerotia        0
fruit-pods       3
fruit spots      4
seed          0
mold-growth       0
seed-discolor     0
seed-size        0
shriveling       0
roots          0
Name: 8, dtype: int64Output Classified As ----> herbicide-injury
Pesticides Suggested As ----> Amino Acid & Seaweed Extracts


Sample Test Data:Date          1
plant-stand      1
```

Fig. 6.16: Sample prediction and pesticide suggestions on test data.

Table 6.1: Performance comparison of existing GNBC, SVM, KNN, and proposed DTC models.

Model/Metric	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)
GNBC model	72.68	72.58	72.72	68.17
SVM classifier	92.36	92.92	92.56	92.09
KNN classifier	97.91	97.62	97.56	97.42
Proposed DTC model	99.07	98.88	98.76	98.86

Table 6.1 provides a performance comparison among four different models used for plant disease classification: the Gaussian Naive Bayes Classifier (GNBC), the Support Vector Machine (SVM) classifier, the K-Nearest Neighbors (KNN) classifier, and the proposed Decision Tree Classifier (DTC). The table lists four key metrics—Accuracy, Precision, Recall, and F1-score—in percentage terms to evaluate how well each model performs.

- **Accuracy (%):** Accuracy represents the overall correctness of the model by measuring the percentage of total predictions that the model got right.
  - **GNBC:** 72.68% – This indicates that about 73% of the predictions are correct.

  - **SVM:** 92.36% – The SVM model shows a significant improvement with over 92% of predictions being correct.
  - **KNN:** 97.91% – The KNN classifier performs very well, correctly classifying nearly 98% of cases.
  - **DTC (Proposed):** 99.07% – The proposed Decision Tree Classifier has the highest accuracy, correctly predicting over 99% of cases.
- **Precision (%):** Precision is the ratio of true positive predictions to the total positive predictions made by the model. It tells us how many of the predicted positive cases were actually positive.
  - **GNBC:** 72.58% – About 72.58% of the samples labeled as a certain disease by GNBC are correctly identified.
  - **SVM:** 92.92% – SVM has high precision, suggesting that when it predicts a disease, it is very likely to be correct.
  - **KNN:** 97.62% – The KNN classifier also demonstrates very high precision.
  - **DTC (Proposed):** 98.88% – The proposed DTC model slightly improves upon KNN, with nearly 99% of its positive predictions being accurate.
- **Recall (%):** Recall (or sensitivity) measures the ratio of true positive predictions to the actual positives. It indicates how well the model identifies all relevant cases.
  - **GNBC:** 72.72% – The GNBC model misses around 27% of actual positive cases.
  - **SVM:** 92.56% – SVM shows strong recall, identifying about 92.56% of all positive cases.
  - **KNN:** 97.56% – KNN demonstrates excellent recall.
  - **DTC (Proposed):** 98.76% – The proposed DTC model has the highest recall, meaning it correctly identifies almost 99% of all positive cases.
- **F1-score (%):** The F1-score is the harmonic mean of precision and recall, providing a balance between the two metrics. It is especially useful when the dataset is imbalanced.

    - **GNBC:** 68.17% – The relatively lower F1-score suggests that both precision and recall are moderately low.
    - **SVM:** 92.09% – SVM's F1-score is high, showing a good balance between precision and recall.
    - **KNN:** 97.42% – KNN maintains a strong F1-score, indicating robust performance.
    - **DTC (Proposed):** 98.86% – The DTC model again leads in this metric, indicating excellent overall performance in balancing precision and recall.

Overall Insights

- **Performance Hierarchy:** The proposed DTC outperforms the other models across all four metrics, followed by the KNN classifier. The SVM classifier also performs well, though not as high as KNN and DTC. GNBC, while simpler, has the lowest performance.
- **Implications:** The high accuracy, precision, recall, and F1-score of the proposed DTC model suggest that it is highly reliable for classifying plant diseases using the given dataset. The model's ability to accurately identify and correctly predict disease classes makes it a promising solution for real-world agricultural applications, where early and precise diagnosis is crucial.
- **Model Selection:** The results emphasize the importance of model selection in machine learning. While simpler models like GNBC might be easier to implement, more complex models such as DTC can yield significantly better performance, justifying their use in critical applications like plant disease diagnosis.

In summary, Table 6.1 demonstrates that as we move from simpler to more advanced models, the performance metrics improve considerably. The proposed DTC model, with the highest scores across all metrics, represents the most effective approach in this project for achieving accurate and reliable plant disease classification.

CHAPTER 7: CONCLUSION AND FUTURE SCOPE

In conclusion, the project successfully demonstrates the power of machine learning in revolutionizing plant disease diagnosis through the development of a comprehensive system that integrates data ingestion, preprocessing, model training, and prediction with an intuitive GUI. By leveraging sparse and categorical IoT data, the system effectively addresses the limitations of traditional, manual methods of disease identification, providing rapid, accurate, and scalable diagnosis. Advanced preprocessing techniques, including label encoding and the application of SMOTE, ensure that the data is optimally balanced and suitable for training. Among the various models tested, the proposed Decision Tree Classifier outperformed others in terms of accuracy, precision, recall, and F1-score, underscoring its suitability for the complexities of multi-class classification in agricultural contexts. The robust performance of the system, coupled with actionable pesticide recommendations, positions this approach as a significant step forward in precision agriculture. Ultimately, this innovative framework not only enhances early detection and intervention but also contributes to cost-effective, sustainable, and scalable disease management, paving the way for further advancements in agricultural technology and data-driven decision-making.

Future work can explore the integration of deep learning techniques and edge computing for real-time, on-field disease detection. Additionally, incorporating complementary data sources—such as remote sensing and weather data—could enhance predictive accuracy and support comprehensive precision agriculture strategies.

Bibliography

[1] Tirkey D, Singh KK, Tripathi S. Performance analysis of AI-based solutions for crop disease identification detection, and classification. Smart Agric Technol. 2023. https://doi.org/10.1016/j.atech.2023.100238.

[2] Ramanjot, et al. Plant disease detection and classification: a systematic literature review". Sensors. 2023. https://doi.org/10.3390/s23104769.

[3] Krishnan VG, Deepa J, Rao PV, Divya V, Kaviarasan S. An automated segmentation and classification model for banana leaf disease detection. J Appl Biol Biotechnol. 2022;10(1):213–20. https://doi.org/10.7324/JABB.2021.100126.

[4] S Mathulaprangsan K Lanthong S Patarapuwadol. 2020. Rice Diseases Recognition Using Effective Deep Learning Models. Telecommun. Eng Media Technol with ECTI North Sect Conf Electr Electron Jt Int Conf Digit Arts Comput. https://doi.org/10.1109/ECTIDAMTNCON48261.2020.9090709

[5] Li, L., Zhang, S., Wang, B.: Plant disease detection and classification by deep learning—a review. IEEE Access **9**, 56683–56698 (2021). https://doi.org/10.1109/ACCESS.2021.3077026

[6] Grinblat, G.L., Uzal, L.C., Larese, M.G., Granitto, P.M.: Deep learning for plant identification using vein morphological patterns. Comput. Electron. Agric. **127**, 418–424 (2016). https://doi.org/10.1016/j.compag.2016.07.003

[7] Mohanty, S.P., Hughes, D.P., Salathé, M.: Using deep learning for image-based plant disease detection. Front. Plant Sci. **7**, 1419 (2016). https://doi.org/10.3389/fpls.2016.01419

[8] Kawasaki Y, Uga H, Kagiwada S, Iyatomi H Basic study of automated diagnosis of viral plant diseases using convolutional neural networks. In: Advances in visual computing: 11th international symposium ISVC 2015 Las Vegas NV USA December 14–16 2015 Proceedings Part II 11, 638–645. Springer International Publishing (2015)

[9] Ma, J., Du, K., Zheng, F., Zhang, L., Gong, Z., Sun, Z.: A recognition method for cucumber diseases using leaf symptom images based on deep convolutional neural network. Comput. Electron. Agric. **154**, 18–24 (2018). https://doi.org/10.1016/j.compag.2018.08.048

[10] Mukti IZ, Biswas D (2019) Transfer learning based plant diseases detection using ResNet50. In: 2019 4th International conference on electrical information and communication technology (EICT), 1–6. IEEE (2019, December)

[11] Chen, J., Chen, J., Zhang, D., Sun, Y., Nanehkaran, Y.A.: Using deep transfer learning for image-based plant disease identification. Comput. Electron. Agric. **173**, 105393 (2020). https://doi.org/10.1016/j.compag.2020.105393

[12] Keceli, A.S., Kaya, A., Catal, C., Tekinerdogan, B.: Deep learning-based multi-task prediction system for plant disease and species detection. Eco. Inform. **69**, 101679 (2022). https://doi.org/10.1016/j.ecoinf.2022.101679

[13] Lee SH, Goëau, H., Bonnet, P., & Joly, A.: Conditional multi-task learning for plant disease identification. In: 2020 25th international conference on pattern recognition (ICPR), 3320–3327. IEEE (2021, January). https://doi.org/10.1109/ICPR48806.2021.9412180

[14] Thakur PS, Khanna P, Sheorey T, Ojha A Vision transformer for plant disease detection: PlantViT. In: International Conference on Computer Vision and Image Processing, 501–511. Cham: Springer International Publishing (2021, December)

[15] Boukabouya RA, Moussaoui A, Berrimi M: Vision Transformer based models for plant disease detection and diagnosis. In: 2022 5th international symposium on informatics and its applications (ISIA), 1–6. IEEE (2022, November). https://doi.org/10.1109/ISIA.2022.9923797

[16] Thai HT, Tran-Van NY, Le KH Artificial cognition for early leaf disease detection using vision transformers. In: 2021 International conference on advanced technologies for communications (ATC), 33–38. IEEE (2021, October). https://doi.org/10.1109/ATC53345.2021.9631065

[17] Thakur PS, Khanna P, Sheorey T, Ojha, A (2022) Explainable vision transformer enabled convolutional neural network for plant disease identification: PlantXViT. arXiv preprint arXiv:2207.07919

[18] Zhu, D., Tan, J., Wu, C., Yung, K., Ip, A.W.: Crop disease identification by fusing multiscale convolution and vision transformer. Sensors **23**(13), 6015 (2023). https://doi.org/10.3390/s23136015

[19] Li, X., Li, S.: Transformer help CNN see better: a lightweight hybrid apple disease identification model based on transformers. Agriculture **12**(6), 884 (2022). https://doi.org/10.3390/agriculture12060884

[20] Li, X., Li, X., Zhang, S., Zhang, G., Zhang, M., Shang, H.: SLViT: Shuffle-convolution-based lightweight Vision transformer for effective diagnosis of sugarcane leaf diseases. J King Saud University-Comput Inform Sci **35**(6), 101401 (2023). https://doi.org/10.1016/j.jksuci.2023.01.014

[21] Hu R, Singh A (2021) Unit: Multimodal multitask learning with a unified transformer. In: Proceedings of the IEEE/CVF international conference on computer vision, 1439– 1449. https://doi.org/10.1109/ICCV48922.2021.00146

[22] Bhattacharjee D, Zhang T, Süsstrunk S, Salzmann M (2022) Mult: An end-to-end multitask learning transformer. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 12031–12041 https://doi.org/10.1109/CVPR52688.2022.01173

[23] Park S, Kim G, Oh Y, Seo JB, Lee SM, Kim JH, Ye JC (2022) Multi-task vision transformer using low-level chest X-ray feature corpus for COVID-19 diagnosis and severity quantification. Med Image Anal 75, 102299 (2022). https://doi.org/10.1016/j.media.2021.102299

[24] Tian, Y., Bai, K.: End-to-end multitask learning with vision transformer. IEEE Trans Neural Netw Learn Syst (2023). https://doi.org/10.1109/TNNLS.2023.3278896

[25] Goncalves DN, Junior JM, Zamboni P, Pistori H, Li J, Nogueira K, Goncalves WN (2023) MTLSegFormer: Multi-task learning with transformers for semantic segmentation in precision agriculture. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 6289–6297. https://doi.org/10.1109/CVPR52688.2023.01278

[26] Vishnoi VK, Kumar K, Kumar B. Plant disease detection using computational intelligence and image processing. Berlin Heidelberg: Springer; 2021.

PUBLISHED PAPER

Machine Learning Adopted Multi-Class Classification of Plant Diseases with Sparse and Categorical IoT Data

Dr. Ayesha Banu[1*], Shivani Soumya Kandakatla[2], Gaje Harshitha[2], Bandi Deepak[2], Yara Anil Kumar[2]

[1]Professor & Head, [2]UG Student, [1,2]Department of Computer Science and Engineering (Data Science), Vaagdevi College of Engineering, Bollikunta, Warangal, Telangana.

*Corresponding Email: ayeshabanuvce@gmail.com

ABSTRACT

Plant disease classification is a critical aspect of modern agriculture that has evolved from traditional manual diagnostic methods to sophisticated automated systems utilizing machine learning. Historically, farmers and experts relied on visual inspection, consultations, and laboratory tests to identify plant diseases—a process that, while effective for small-scale applications, was often subjective, time-consuming, and inconsistent, leading to delayed interventions and crop losses. The core problem lies in the inefficiency and high cost of these traditional methods, which struggle with large-scale agricultural demands and the complex nature of disease symptoms. This highlights the need for an innovative system that can swiftly and accurately classify plant diseases using sparse and categorical IoT data. The proposed system uses modern data analytics and machine learning techniques to overcome these limitations. It includes data preprocessing methods such as filling missing values, label encoding, and applying Synthetic Minority Oversampling Technique (SMOTE) to address class imbalance, ensuring high-quality input data. The system employs a multi-class classification pipeline using models like Gaussian Naive Bayes, Support Vector Machines, K-Nearest Neighbors, and a novel Decision Tree Classifier, which demonstrates superior performance. Evaluation shows the Decision Tree model achieves 99.07% accuracy, with precision, recall, and F1-scores consistently above 98%, validating its robustness. This research has the potential to transform plant disease management by enabling real- time, data-driven insights for early diagnosis and targeted pesticide use, enhancing crop yield, minimizing economic losses, and promoting sustainable farming practices.

Keywords: Plant disease identification, IoT sensor data, Predictive analytics, Decision analytics, Machine Learning, Data balancing, SMOTE algorithm.

1. INTRODUCTION

Agricultural biodiversity is essential for providing humans with food and raw materials and is an essential component of human civilization [1, 2]. The disease can occur when pathogenic organisms such as fungi, bacteria, and nematodes; soil PH; temperature extremes; changes in the quantity of moisture and humidity in the air; and other elements continuously harm a plant. Plant diseases can have an impact on the growth, function, and structure of plants and crops, affecting the people that rely on them. Plant diseases can have a severe impact on crop health, leading to substantial damage, reduced yields, and financial losses for farmers. The majority of farmers still use manual methods to detect and classify plant ailments because it is difficult to do so early on, and this reduces productivity. Agriculture’s productivity is a significant economic factor. As a result, disease identification and classification in plants are critical in agricultural industries [3]. If proper precautions are not taken, it can have serious consequences for plants by reducing the quality, quantity, or productivity of the corresponding products or services. Automatic disease detection and classification recognize symptoms at an early stage, i.e., when they first appear on plant leaves, lowering the amount of labor necessary to monitor large farms of crops. According to [4] Plant leaf disease is a major issue in rice production, and the disease has the potential to harm the crop, resulting in a drop in products. Farmers have a difficult time detecting and classifying plant leaf diseases. The timely and precise detection of plant diseases plays a vital role in implementing effective disease management and prevention strategies.

2. LITERATURE SURVEY

This section offers an extensive overview of prior works in the context of this research, encompassing machine learning models with the goal of identifying their strengths, limitations, and research gaps, thus forming the basis for the proposed method. An extensive review by Li et al. [5] presents the recent research progress in using deep learning technology for crop leaf disease identification. The authors discuss the current trends, challenges, and unresolved issues pertaining to plant leaf disease detection using deep learning and advanced imaging techniques are discussed, with the aim of providing a valuable resource for researchers in the field. Deep learning techniques were initially applied to plant image recognition with a focus on analyzing leaf vein patterns. A notable study employed a CNN architecture with 3–6 layers to successfully classify three leguminous plant species: white bean, red bean, and soybean [6]. The CNN model extracted and analyzed intricate features of leaf vein structures unique to each species, capturing low-level features such as edges and textures and progressing to more complex patterns. This hierarchical feature extraction enabled effective species differentiation based on leaf vein characteristics. The study demonstrated the potential of CNNs in plant species classification, highlighting the effectiveness of deep learning in capturing subtle biological variations for accurate identification, and laying the groundwork for further applications in plant image recognition.

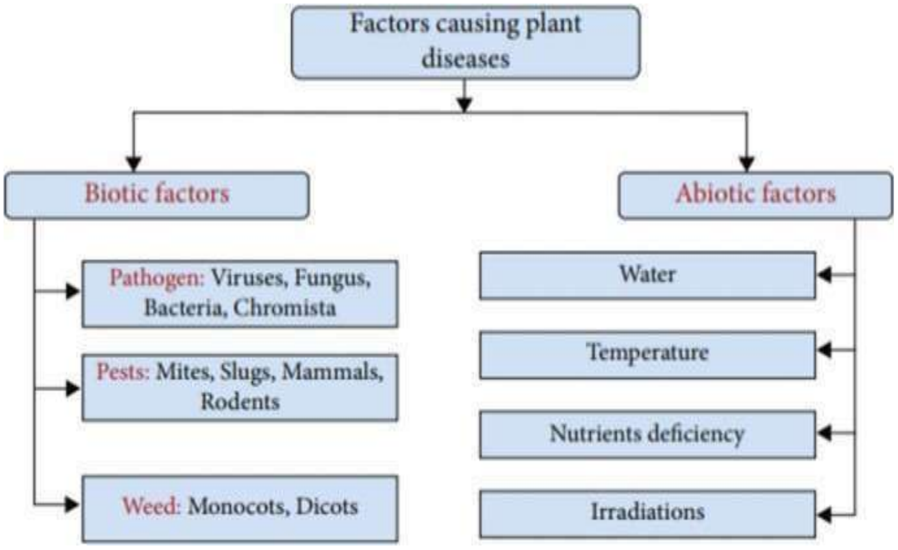

Factors causing plant diseases
Biotic factors
Abiotic factors
Pathogen: Viruses, Fungus, Bacteria, Chromista
Pests: Mites, Slugs, Mammals, Rodents
Weed: Monocots, Dicots
Water
Temperature
Nutrients deficiency
Irradiations

Fig. 1: Factors responsible for plant diseases [26].

Subsequently, Mohanty et al. [7] employed a classifier model to accurately identify 14 different crop species and 26 crop diseases, achieving an impressive accuracy of 99.35%. Building upon this work, Kawasaki et al. [8] proposed a CNN-based system specifically designed for the precise recognition of diseases affecting cucumber leaves, achieving a commendable accuracy rate of 94.9%. In an investigation by Ma et al. [9], a deep CNN was employed for the identification and recognition of four diseases in cucumber plants, including downy mildew, anthracnose, powdery mildew, and target leaf spots, based on their distinct symptoms. This study reported a recognition accuracy of 93.4%. Impressive strides have been made in plant leaf disease recognition; however, the availability of diverse datasets remains limited. CNN training necessitates extensive and diverse datasets, which are currently scarce in the field of plant leaf disease recognition. Consequently, transfer learning has emerged as a viable solution to enhance the robustness of CNN classifiers. By adapting pre-trained CNNs and retraining them with smaller datasets showcasing different distributions, transfer learning has proven to be an effective approach in implementing deep learning models [10]. Promising results have been demonstrated by leveraging pre-trained CNN models from the ImageNet dataset and retraining them for leaf disease recognition [11].

Recently, MTL models have been explored in the field of plant disease detection, enabling the simultaneous execution of multiple tasks. These models can effectively handle tasks, such as severity estimation of leaf disease, identification of plant species, and classification of plant disease concurrently [12]. To address the limitations in information utilization and scalability of existing models, Lee et al. [13] introduced a novel Conditional Multi-Task Learning (CMTL) model in this context. This method employs an MTL mechanism with a conditional scheme that links host species and disease representations, mimicking how plant pathologists infer diseases from species information. This improves plant disease identification performance compared to traditional joint species–disease pair modeling. It efficiently utilizes available information and is scalable, not requiring distinctions between similar species or diseases, thus enhancing accuracy and suitability for real-world applications. However, ViTs offer advantages over MTL frameworks in plant disease detection. They capture both local and global context information, enabling accurate analysis of plant images. ViTs also generalize well with limited labeled data through pre-training on large-scale datasets like ImageNet. Additionally, they provide flexibility and scalability by adapting to different disease detection tasks with a single model, reducing complexity and computational requirements.In their work, Thakur et al. [14] introduced a hybrid model, named PlantViT, for automating plant disease detection, which integrates the strengths of a CNN and a ViT, incorporating a multi-head attention module. The model was evaluated on two extensive plant disease detection datasets, namely, PlantVillage and Embrapa. Experimental results showed that the model achieves detection accuracies of 98.61% and 87.87% on the PlantVillage and the Embrapa datasets, respectively. These results outperform the current state-of- the-art methods, highlighting the effectiveness of PlantViT in plant disease detection.

In [15], a lightweight deep learning approach leveraging the ViT architecture is proposed for real-time automated plant disease classification. The study rigorously compares various models, including ViT, traditional CNNs, and a hybrid of CNN and ViT. Extensive training and evaluation across multiple datasets reveal that while attention blocks significantly enhance accuracy, they also introduce latency in predictions. However, integrating attention blocks with CNN blocks strikes an optimal balance between accuracy and speed, offering a robust solution for real-time plant disease classification.In recent studies, deep learning models have been employed to identify cassava leaf diseases, replacing the traditional and unreliable intuitive diagnosis conducted by farmers. To improve the disease detection accuracy in these species, Thai et al. [16] explored the use of the ViT model instead of a CNN. Experimental results demonstrate that the ViT model achieves competitive accuracy, surpassing popular CNN models such as EfficientNet and Resnet50 on the Cassava Leaf Disease Dataset. These findings highlight the potential superiority of the ViT model in analyzing leaf diseases. An extended study by Thakur et al. [17] addressed the underexplored application of ViTs in plant pathology by introducing PlantXViT, a light-weight model based on CNNs with ViTs for efficient identification of various plant diseases across different crops, well suited for IoT-based smart agriculture. Evaluation on five datasets shows that PlantXViT outperforms state-of-the-art models, achieving average accuracies of over 93.55%, 92.59%, and 98.33% on Apple, Maize, and Rice datasets, respectively, even in challenging backgrounds. The model's explainability is also assessed using gradient-weighted class activation maps, demonstrating its ability to provide insights into the decision-making process.Zhu et al. [18] introduced the Multiscale Convolution and Vision Transformer (MSCVT), a hybrid model that integrates multiscale convolutions with ViT techniques for precise crop disease identification. This approach combines the strengths of multiscale convolutions, which capture fine-grained local features, with ViT's ability to model long-range dependencies and global context. The study demonstrates that MSCVT outperforms single-scale CNNs and standalone ViT models by effectively capturing local and global details while optimizing computational efficiency. This balance ensures high accuracy in disease identification and practical deployment in resource-constrained environments, making MSCVT a valuable tool for early detection and management of crop diseases.

The ConvViT [19] is a lightweight ViT-based approach developed for apple leaf disease identification. It integrates convolutional and Transformer structures to effectively capture both global and local features of crop disease spots, enhancing overall robustness and accuracy. The patch embedding method is improved to preserve edge information and facilitate information exchange within the Transformer. By leveraging depthwise separable convolution and linear-complexity multi-head attention operations, ConvViT achieves substantial reduction in resource requirements. This model achieves comparable identification performance (96.85%) on a complex background apple leaf disease dataset, with significantly reduced parameters (32.7%) and FLOPs (21.7%). This highlights the effectiveness and practicality of ConvViT as a disease identification model. The Shuffle Convolution-based Lightweight Vision Transformer (SLViT) [20] is a hybrid network designed for the accurate diagnosis of sugarcane leaf diseases. The study demonstrates that SLViT, which combines a lightweight CNN architecture with a flexible plug-in Transformer encoder, surpasses the state-of-the-art models on the publicly available Plant Village dataset with respect to the metrics speed, weight, memory usage, and precision.

The effectiveness of MTL in enhancing correlated tasks has been well established. Existing approaches typically employ a backbone for initial feature extraction and utilize exclusive branches for each task. However, the sharing of information between branches is often accomplished through basic concatenation or summation of feature maps, which results in overlooking the local image features and neglecting the significance or correlation between tasks. MTL-based ViTs offer a significant advancement in addressing the limitations of existing approaches. By integrating ViTs into the MTL framework, these models can capture both the global and local characteristics of images, enabling more comprehensive information exchange between tasks. This integration allows for a better understanding of the relationships and dependencies among tasks, leading to improved performance across multiple related tasks. MTL-based ViTs provide a more holistic approach to leveraging shared knowledge and extracting meaningful features, resulting in enhanced accuracy and robustness in correlated task scenarios. In the recent years, ViTs have been used in performing multiple tasks simultaneously sharing the data across different tasks. The Unified Transformer (UniT) [21] model is proposed for the simultaneous learning of multiple tasks across diverse domains, encompassing object detection, natural language understanding, and multimodal reasoning. The UniT encodes input modalities using an encoder and generates task predictions using a shared decoder over the encoded input representations. Task-specific output heads are employed, and the entire model is trained jointly end-to-end with task losses. UniT distinguishes itself by employing shared model parameters across all tasks, accommodating a wide range of tasks across domains, and delivering robust performance across 7 tasks on 8 datasets with reduced parameter count.

Similarly, the MulT [22] framework proposes an end-to-end MTL-based Transformer for simultaneous learning of multiple high-level vision tasks. It outperforms state-of-the-art multitask CNNs and single-task Transformers, demonstrating the significance of shared attention across tasks. The MulT model shows robustness and generalization to new domains in multitask benchmarks. The study highlights the advantages of Transformer-based architectures for MTL, such as better scalability, improved feature representation, and the ability to capture complex dependencies between tasks, thereby providing a substantial contribution to the field and paving the way for future research and applications in domains requiring robust multitask learning solutions. To address the challenges of constructing a comprehensive dataset for diagnosing and quantifying the COVID-19 severity using Chest X-ray (CXR) data, a novel

Multi-task ViT architecture is proposed by Park et al. [23]. This approach leverages abundant unlabeled CXR data through self-attention modeling, overcoming the limitations of existing ViT models that are not optimized for CXR. By utilizing a backbone network trained on public datasets, the multi-task ViT extracts informative low-level features from the images, which serve as valuable resources for a robust ViT. Best performances are exhibited by the model in both diagnosis and severity quantification tasks, and remarkable generalization capability is demonstrated on external test datasets from diverse institutions. These promising results underscore the potential for widespread deployment of the model in the medical field.

Tian et al. [24] introduced the MultiTask ViT (MTViT), a representation learning method that leverages ViTs. MTViT introduces a multiple branch transformer that sequentially processes image patches associated with different tasks, allowing for information exchange through the Cross-Task attention (CA) module. Unlike previous models, MTViT utilizes ViT's self-attention mechanism for feature extraction, resulting in improved memory and computation efficiency. Experimental results on benchmark datasets demonstrate that MTViT outperforms or achieves comparable performance to existing CNN-based MTL methods. Despite the widespread adoption of MTL-based ViTs in various computer vision applications, there remains a noticeable research gap in their application to plant disease detection. The work proposed by Goncalves et al. [25], known as MTLSegFormer, stands out as the only existing effort in exploring the potential of MTL-based ViTs to address plant disease detection challenges. MTLSegFormer is a semantic segmentation model designed for precision agriculture that leverages both MTL and attention mechanisms. It combines MTL and attention mechanisms, learning two feature maps for each task after extracting backbone features. The model assigns weights to important local regions for specific tasks, resulting in improved accuracy for correlated tasks in precision agriculture.

3. PROPOSED METHODOLOGY

This research is a comprehensive Python application designed to diagnose plant diseases using machine learning. It integrates data preprocessing, visualization, machine learning, and a user-friendly interface to facilitate the diagnosis of plant diseases using IoT data. It offers a robust platform for both data scientists (Admins) and end-users, supporting the entire pipeline from data exploration to model training and real-time predictions with actionable recommendations. Here's an overview of its main components:

Step-1: Graphical User Interface (GUI)

The research uses Tkinter to create a full-screen GUI that includes background images, text widgets for logging and displaying results, and buttons for user interactions. There are separate functionalities for Admin and User roles. Admins can upload datasets, perform exploratory data analysis (EDA), preprocess data, apply oversampling techniques (SMOTE), split data, and train multiple classifiers. Regular users have access to the prediction functionality.

Step-2: Data Handling and Visualization

Users can load CSV datasets, which are then displayed in the GUI. The project provides several plotting functions such as bar plots, violin plots, histograms, scatter plots, strip plots, and correlation heatmaps to visualize different aspects of the data. This helps in understanding the distribution of features and the relationships between them.

Step-3: Data Preprocessing and Augmentation: Non-numeric (categorical) columns in the dataset are converted into numeric values using label encoding, making the data suitable for machine learning algorithms. Missing data is replaced with zeros to maintain dataset consistency. The project employs

SMOTE (Synthetic Minority Oversampling Technique) to address class imbalance, which is visualized by comparing class distributions before and after the technique is applied.

Step-4: Machine Learning and Model Training

The data is split into training (80%) and testing (20%) sets. The project includes several machine learning models: Gaussian Naive Bayes (GNBC), Support Vector Machine (SVC), K-Nearest Neighbors (KNN), and Decision Tree Classifier (DTC). Models are either trained on-the-fly or loaded from saved files if they already exist. After training, the models are evaluated using precision, recall, F1 score, accuracy, and confusion matrices, providing insight into model performance.

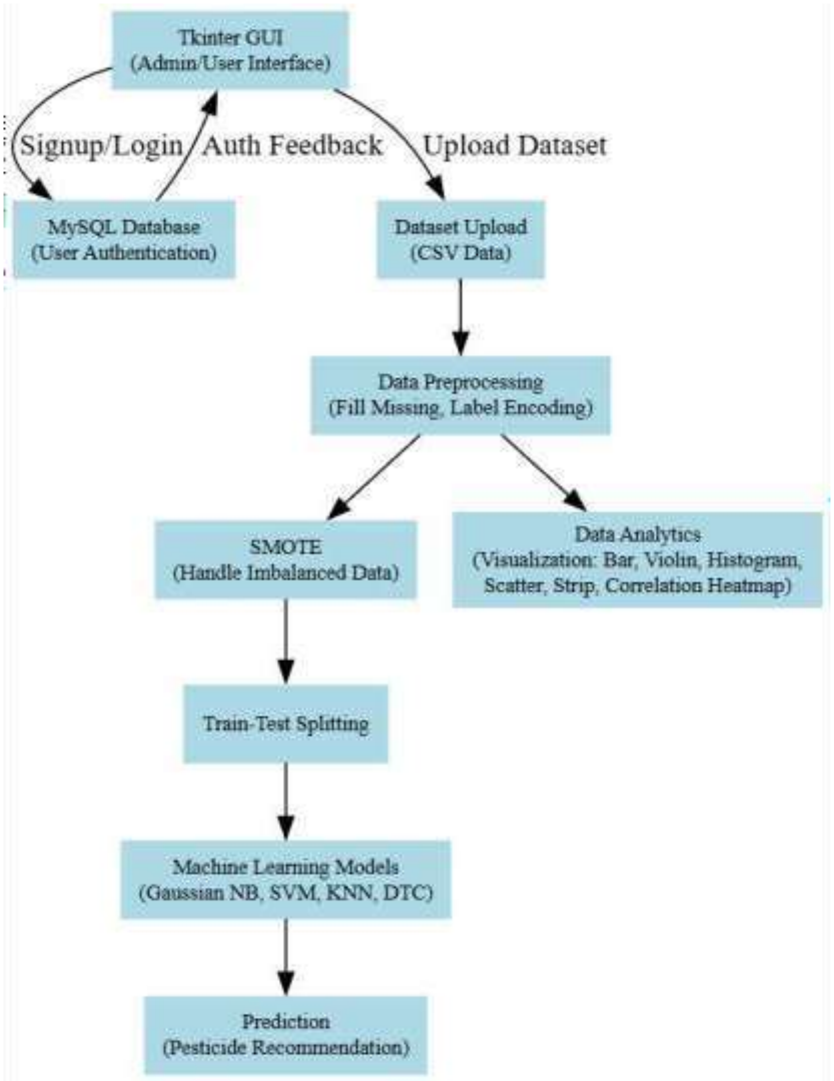

Tkinter GUI (Admin/User Interface)
Signup/Login
Auth Feedback
Upload Dataset
MySQL Database (User Authentication)
Dataset Upload (CSV Data)
Data Preprocessing (Fill Missing, Label Encoding)
SMOTE (Handle Imbalanced Data)
Data Analytics (Visualization: Bar, Violin, Histogram, Scatter, Strip, Correlation Heatmap)
Train-Test Splitting
Machine Learning Models (Gaussian NB, SVM, KNN, DTC)
Prediction (Pesticide Recommendation)

Fig. 2: System architecture of proposed plant disease classification with pesticide suggestion from categorical IoT data.

Step-5: Prediction and Recommendations

Users can upload new test data, and the trained model will predict the disease class. Each predicted disease is mapped to a specific pesticide recommendation based on predefined lists.

Step-6: Database Integration for User Authentication

The project connects to a local MySQL database to manage user authentication. Separate signup and login functions are provided for Admins and Users. After login, different sets of functionalities are made available according to the user's role.

3.2 Model Building and Training

3.2.1 Gaussian Naive Bayes (GNBC)

Gaussian Naive Bayes is a probabilistic classifier that applies Bayes' theorem with the "naive" assumption that features are conditionally independent given the class label. For continuous data, it assumes that the features follow a Gaussian (normal) distribution. The model calculates the likelihood of each feature given a class, multiplies these probabilities together (along with the prior probability of the class), and predicts the class with the highest posterior probability.

3.2.2 Support Vector Machine

The SVM classifier aims to find the optimal hyperplane that separates classes in the feature space. In its simplest (linear) form, SVM finds the hyperplane with the maximum margin between classes. For non-linearly separable data, kernel functions are used to transform the data into a higher-dimensional space where a linear separation is possible.

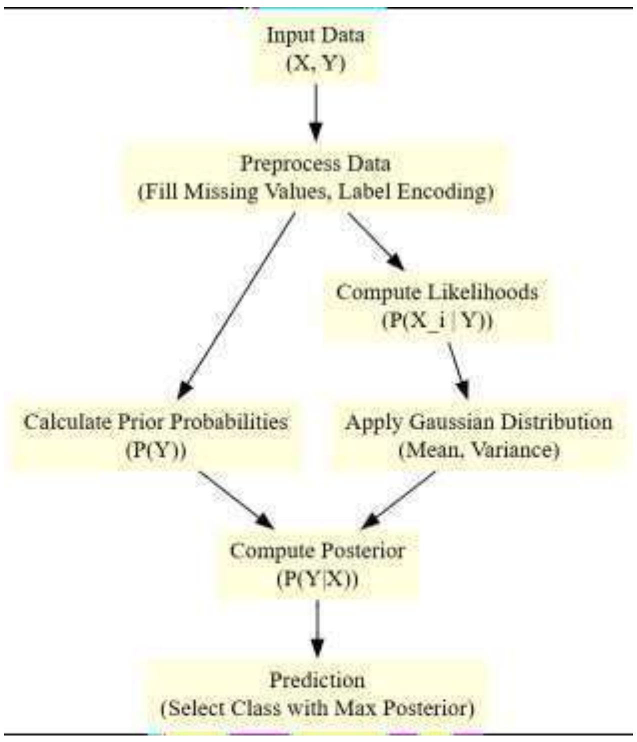
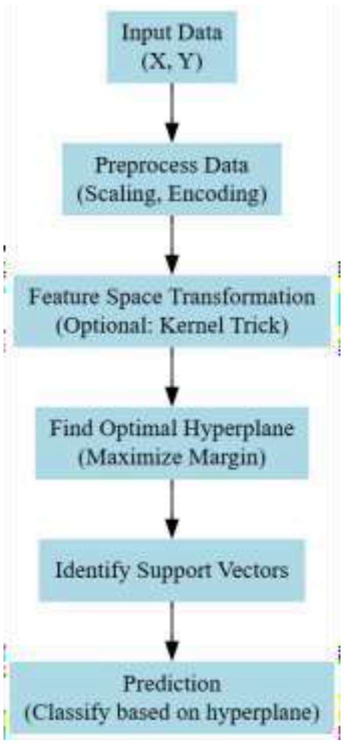

Input Data
(X, Y)
Preprocess Data
(Fill Missing Values, Label Encoding)
Compute Likelihoods
(P(X_i | Y))
Calculate Prior Probabilities
(P(Y))
Apply Gaussian Distribution
(Mean, Variance)
Compute Posterior
(P(Y|X))
Prediction
(Select Class with Max Posterior)


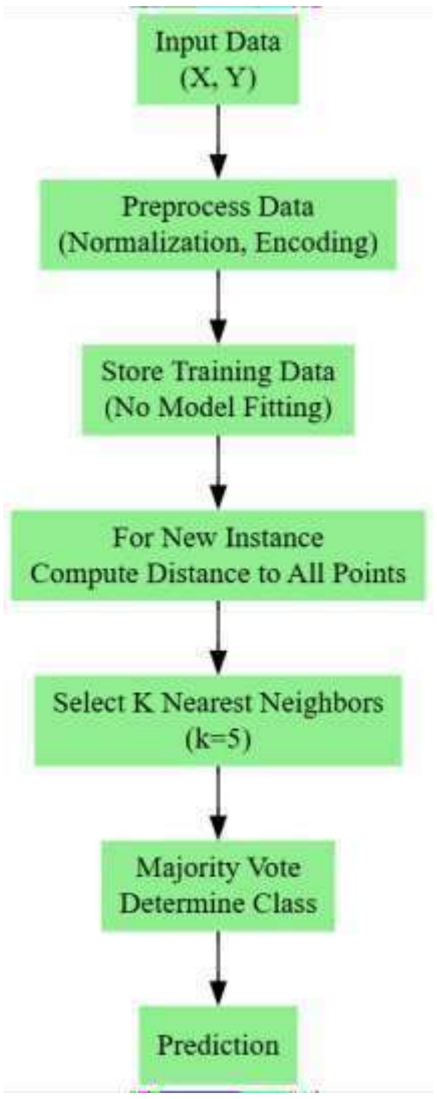
Fig. 3: Working of GNBC model (left). working of SVM classifier (right).

3.2.3 KNN Classifier

K-Nearest Neighbors is an instance-based, non-parametric algorithm. It works by storing all available cases and classifying new cases based on a similarity measure (e.g., Euclidean distance). In this project, the KNN model is configured with k=5k = 5k=5, meaning that for each test instance, the five closest neighbors are considered, and the class is predicted based on the majority vote among these neighbors.

3.2.4 DTC model

A Decision Tree is a flowchart-like structure where internal nodes represent tests on features, branches represent the outcome of these tests, and leaf nodes represent class labels. The tree is constructed by recursively splitting the dataset based on feature values that provide the maximum information gain or the best reduction in impurity (e.g., Gini impurity or entropy).



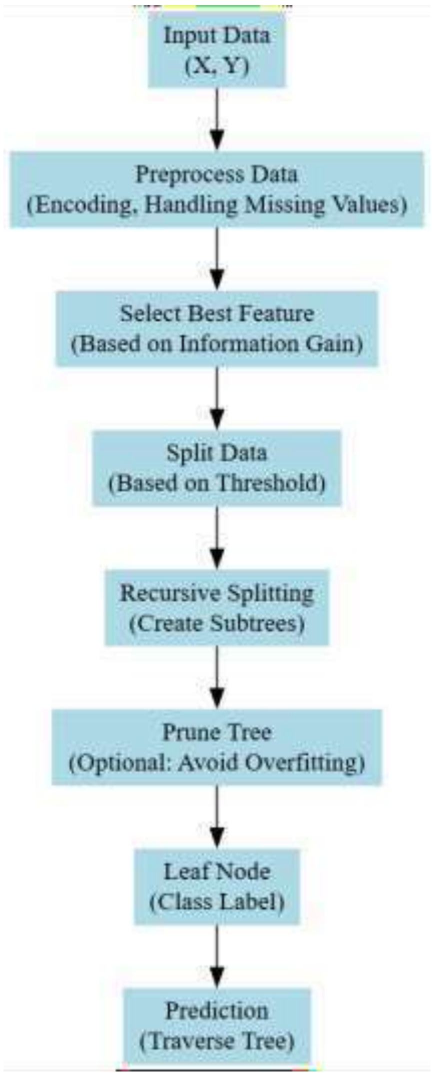


Fig. 4: Working flow of KNN classification (left). DTC model workflow (right).

3.2.5 Comparative Discussion

After training multiple models such as GNB classifier, SVM classifier, KNN classifier, and DTC model, a comprehensive evaluation of performance metrics such as accuracy, precision, recall, and F1-score revealed that the DTC model achieved the best overall performance. The superior performance of the DTC model can be attributed to its ability to capture complex, non-linear relationships inherent in the plant disease dataset. Unlike probabilistic models like GNB classifier, which assume independence among features, the DTC approach effectively identifies and utilizes the most informative features through recursive splitting. This inherent feature selection mechanism allows the model to handle the diversity of sensor data and the subtle variations in plant disease symptoms more adeptly.

Additionally, the interpretability of the DTC model provides valuable insights into the decision-making process. The tree structure clearly outlines the sequence of feature-based decisions that lead to a final classification, making it easier to understand which factors most significantly contribute to the diagnosis. This transparency not only aids in model validation but also facilitates further refinement of feature engineering strategies. Moreover, while models like SVM and KNN offer robust performance in many contexts, they may struggle with high-dimensional, sparse, and categorical data unless extensively tuned or transformed. The DTC model, on the other hand, naturally accommodates such complexities without the need for extensive preprocessing beyond the standard data normalization and encoding steps.

In summary, the Decision Tree Classifier emerged as the best-performing model in this project due to its effective handling of non-linear relationships, automatic feature selection, and interpretability. Its success underscores the importance of choosing a model that aligns well with the data's characteristics, ultimately contributing to more reliable and accurate plant disease classification outcomes.

4. RESULTS AND DISCUSSION

4.1 Dataset description

The dataset is designed for multi-class classification of plant diseases. It contains various environmental and plant-specific measurements, along with a target column indicating the disease label. The dataset is provided as a CSV (Comma-Separated Values) file, making it easy to load and manipulate using data analysis libraries such as Pandas. In essence, this dataset combines sparse and categorical IoT data with plant-specific measurements to facilitate the multi-class classification of plant diseases. It has been structured to handle 18 disease categories, each reflecting a distinct pathology. The ultimate goal is to accurately predict plant diseases and suggest appropriate pesticide treatments, thereby enhancing both the efficiency and effectiveness of disease management in agricultural settings.

The dataset includes several key features. The "temp" column is numeric and represents temperature measurements from IoT sensors or manual observations. "Leafspot-size" is also numeric and indicates the size or extent of leaf spots, a common symptom of many plant diseases. "Plant-growth" is a numeric feature that measures overall plant growth or vigor, possibly on a defined scale. The "germination" column is numeric as well and reflects the germination rate or a related metric capturing seed viability or early growth. The "stem" column may represent stem diameter, length, or a damage index relevant to disease symptoms and is also numeric. The "Label" column is categorical, later converted to numeric values using label encoding. It represents the target variable indicating the disease category. The dataset is focused on these columns for exploratory data analysis (EDA) and model training, although additional features such as humidity, rainfall, leaf color, or texture could be present depending on the data collection method.

There are 18 disease categories in the dataset, each encoded numerically after preprocessing. Some of the referenced diseases include diaporthe-stem-canker, charcoal-rot, rhizoctonia-root-rot, phytophthora-rot, brown-stem-rot, powdery-mildew, downy-mildew, brown-spot, bacterial-blight, bacterial-pustule, purple-seed-stain, anthracnose, phyllosticta-leaf-spot, alternarialeaf-spot, frog-eye- leaf-spot, diaporthe-pod-&-stem-blight, cyst-nematode, and herbicide-injury. These labels are essential for training machine learning algorithms to accurately classify plant diseases and recommend targeted interventions.

4.2 Results description

This research is a comprehensive Python application built using Tkinter for the graphical user interface (GUI) and integrates several modules to support a complete machine learning workflow for classifying plant diseases based on sparse and categorical IoT data. Fig. 5 contrasts the class distribution before and after the application of the SMOTE algorithm. Two bar plots are shown side-by-side: one displaying the imbalanced counts of each disease class, and the other illustrating the balanced class distribution achieved through synthetic oversampling.

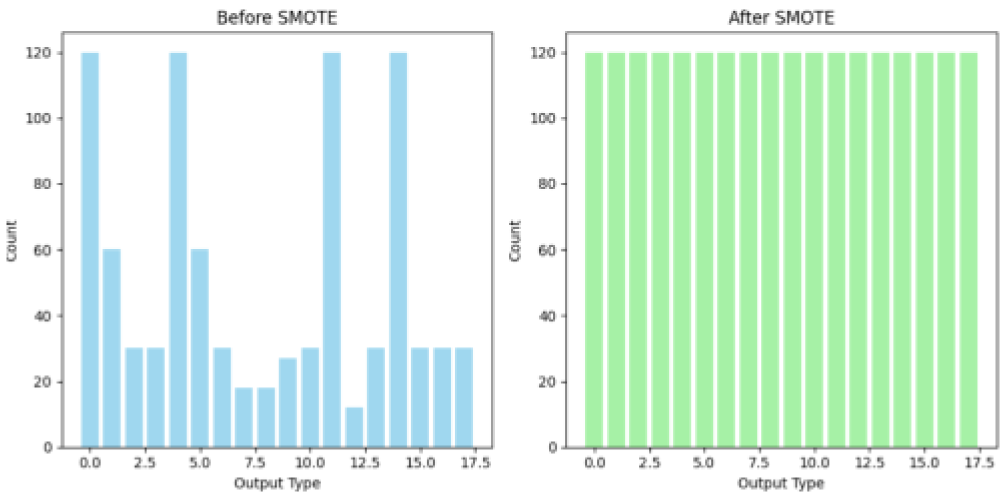
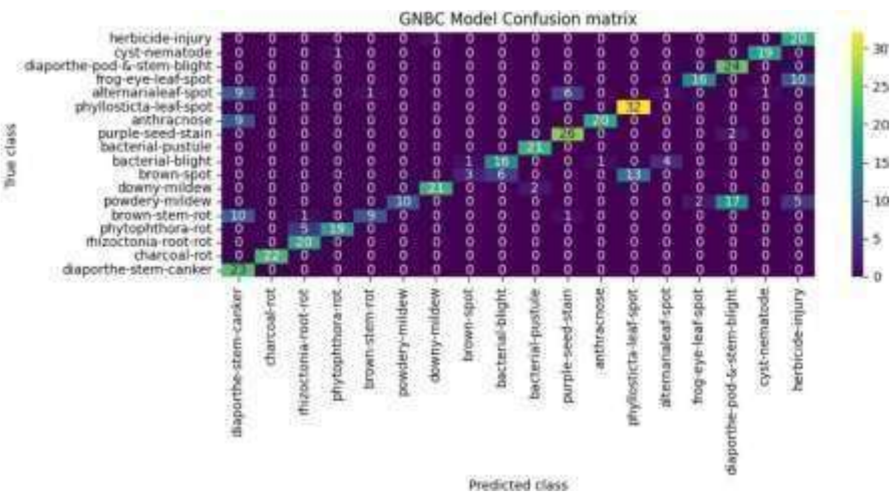
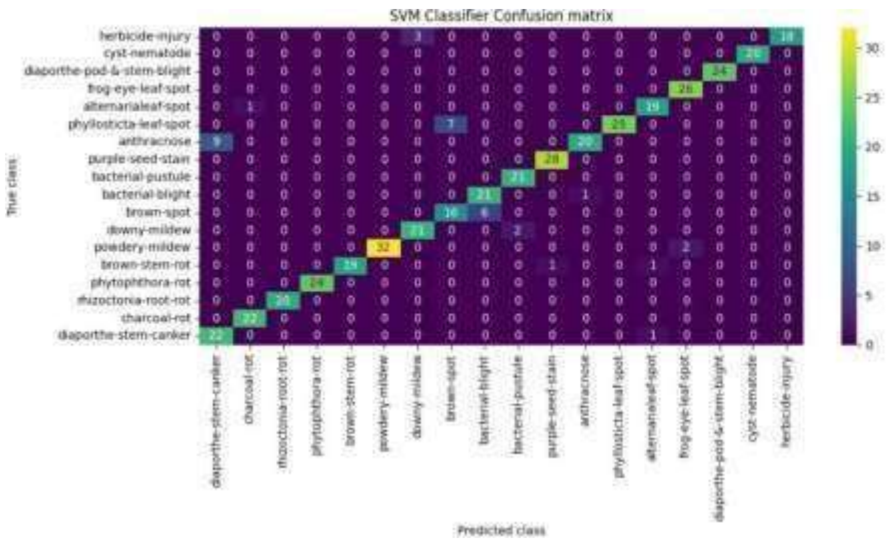


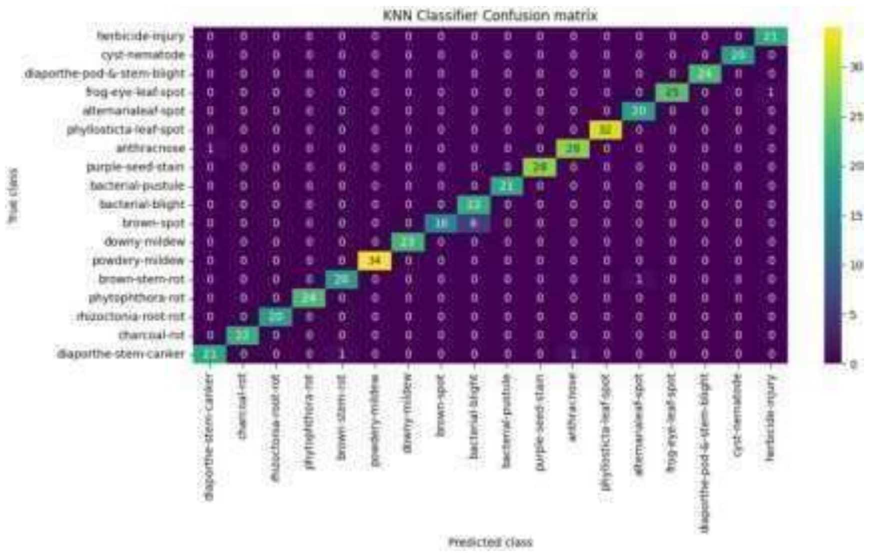
Fig. 5: Bar plot of count versus output before and after applying SMOTE algorithm.



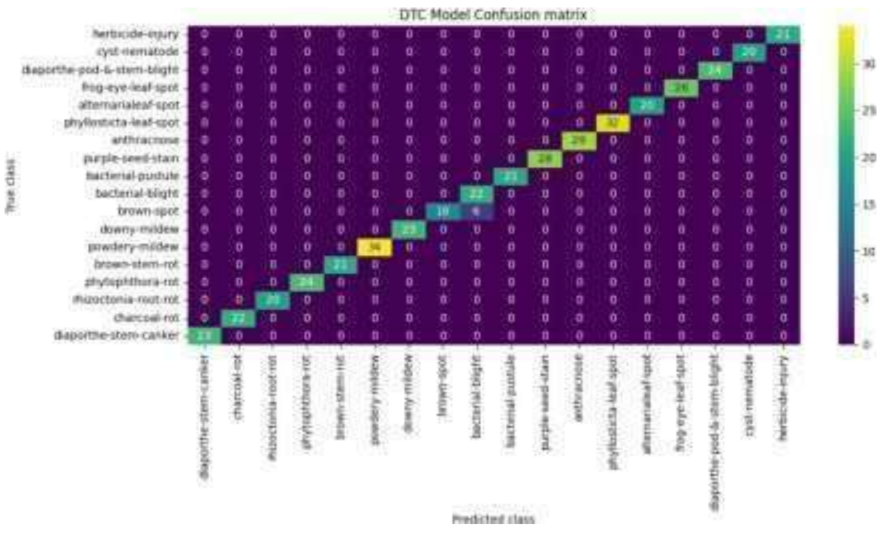
(a)



(b)



(c)



(d)

Fig. 8: Confusion matrices obtained using (a) GNBC model. (b) SVM classification. (c) KNN classification. (d) DTC model.

Fig. 8 includes confusion matrices for:

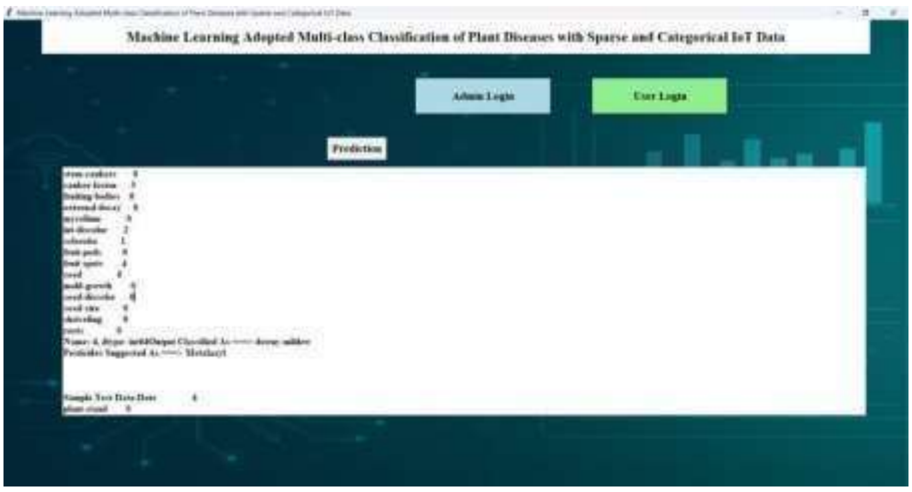
(a) Gaussian Naive Bayes (GNBC) model: Showing its classification performance.

(b) SVM Classifier: Illustrating how well the SVM model differentiates between disease classes.

(c) KNN Classifier: Visualizing the performance of the K-Nearest Neighbors algorithm.

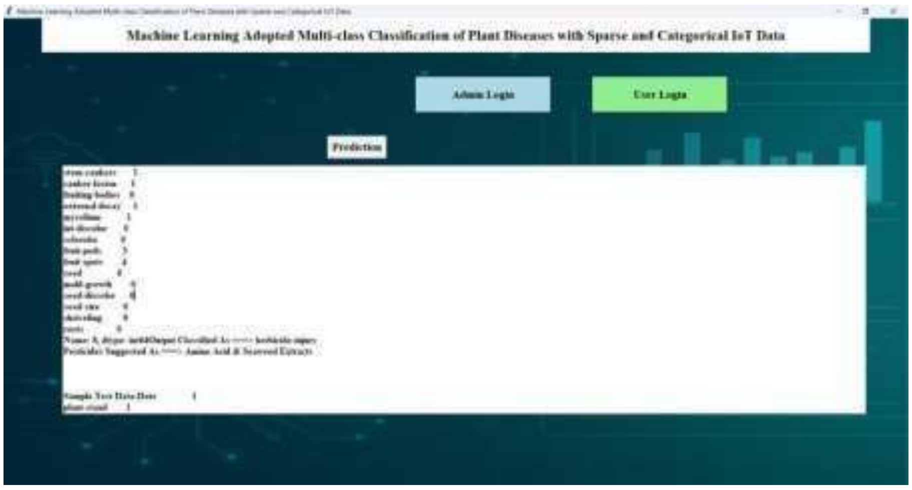
(d) Decision Tree Classifier (DTC): Depicting the confusion matrix of the decision tree model, which may show superior performance compared to the others.

In the Fig. 9, the application displays the outcome of a prediction on a test dataset. It shows the classified disease and the corresponding pesticide recommendation, thereby completing the full cycle—from data input to actionable output for disease management.


Machine Learning Adopted Multi-class Classification of Plant Diseases with Sparse and Categorical IoT Data
Admin Login
User Login
Prediction

```
stem-cankers      0
canker-lesion     3
fruiting-bodies   0
external decay    0
mycelium          0
int-discolor      2
sclerotia         1
fruit-pods        0
fruit spots       4
seed              0
mold-growth       0
seed-discolor     0
seed-size         0
shriveling        0
roots             0
Name: 4, dtype: int64Output Classified As ----> downy-mildew
Pesticides Suggested As ----> Metalaxyl


Sample Test Data Date        4
plant-stand      0
```


Machine Learning Adopted Multi-class Classification of Plant Diseases with Sparse and Categorical IoT Data
Admin Login
User Login
Prediction

```
stem-cankers      1
canker-lesion     1
fruiting-bodies   0
external decay    1
mycelium          1
int-discolor      0
sclerotia         0
fruit-pods        3
fruit spots       4
seed              0
mold-growth       0
seed-discolor     0
seed-size         0
shriveling        0
roots             0
Name: 8, dtype: int64Output Classified As ----> herbicide-injury
Pesticides Suggested As ----> Amino Acid & Seaweed Extracts


Sample Test Data Date        1
plant-stand      1
```

Fig. 9: Sample prediction and pesticide suggestions on test data.

Table. 1: Performance comparison of existing GNBC, SVM, KNN, and proposed DTC models.

Model/Metric	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)
GNBC model	72.68	72.58	72.72	68.17
SVM classifier	92.36	92.92	92.56	92.09
KNN classifier	97.91	97.62	97.56	97.42
Proposed DTC model	99.07	98.88	98.76	98.86

Table. 1 presents a comparative analysis of four machine learning models—Gaussian Naive Bayes Classifier (GNBC), Support Vector Machine (SVM), K-Nearest Neighbors (KNN), and the proposed Decision Tree Classifier (DTC)—based on key performance metrics: Accuracy, Precision, Recall, and F1-score. The GNBC model achieved an accuracy of 72.68%, with a precision of 72.58%, recall of 72.72%, and an F1-score of 68.17%, indicating moderate performance across all metrics. In contrast, the SVM classifier significantly improved the results, with 92.36% accuracy, 92.92% precision, 92.56% recall, and a balanced F1-score of 92.09%. The KNN classifier performed even better, attaining 97.91% accuracy, 97.62% precision, 97.56% recall, and an F1-score of 97.42%, showcasing its robustness and reliability. However, the proposed Decision Tree Classifier (DTC) outperformed all other models with an outstanding accuracy of 99.07%, precision of 98.88%, recall of 98.76%, and F1-score of 98.86%.

These results demonstrate the superior effectiveness of the proposed DTC model in accurately classifying plant diseases, offering a highly reliable solution for precision agriculture.

5. CONCLUSION

In conclusion, the project successfully demonstrates the power of machine learning in revolutionizing plant disease diagnosis through the development of a comprehensive system that integrates data ingestion, preprocessing, model training, and prediction with an intuitive GUI. By leveraging sparse and categorical IoT data, the system effectively addresses the limitations of traditional, manual methods of disease identification, providing rapid, accurate, and scalable diagnosis. Advanced preprocessing techniques, including label encoding and the application of SMOTE, ensure that the data is optimally balanced and suitable for training. Among the various models tested, the proposed Decision Tree Classifier outperformed others in terms of accuracy, precision, recall, and F1-score, underscoring its suitability for the complexities of multi-class classification in agricultural contexts. The robust performance of the system, coupled with actionable pesticide recommendations, positions this approach as a significant step forward in precision agriculture. Ultimately, this innovative framework not only enhances early detection and intervention but also contributes to cost-effective, sustainable, and scalable disease management, paving the way for further advancements in agricultural technology and data-driven decision-making.

REFERENCES

[1] Tirkey D, Singh KK, Tripathi S. Performance analysis of AI-based solutions for crop disease identification detection, and classification. Smart Agric Technol. 2023. https://doi.org/10.1016/j.atech.2023.100238.

[2] Ramanjot, et al. Plant disease detection and classification: a systematic literature review". Sensors. 2023. https://doi.org/10.3390/s23104769.

[3] Krishnan VG, Deepa J, Rao PV, Divya V, Kaviarasan S. An automated segmentation and classification model for banana leaf disease detection. J Appl Biol Biotechnol. 2022;10(1):213–20. https://doi.org/10.7324/JABB.2021.100126.

[4] S Mathulaprangsan K Lanthong S Patarapuwadol. 2020. Rice Diseases Recognition Using Effective Deep Learning Models. Telecommun. Eng Media Technol with ECTI North Sect Conf Electr Electron Jt Int Conf Digit Arts Comput. https://doi.org/10.1109/ECTIDAMTNCON48261.2020.9090709

[5] Li, L., Zhang, S., Wang, B.: Plant disease detection and classification by deep learning—a review. IEEE Access **9**, 56683–56698 (2021). https://doi.org/10.1109/ACCESS.2021.3077026

[6] Grinblat, G.L., Uzal, L.C., Larese, M.G., Granitto, P.M.: Deep learning for plant identification using vein morphological patterns. Comput. Electron. Agric. **127**, 418–424 (2016). https://doi.org/10.1016/j.compag.2016.07.003

[7] Mohanty, S.P., Hughes, D.P., Salathé, M.: Using deep learning for image-based plant disease detection. Front. Plant Sci. **7**, 1419 (2016). https://doi.org/10.3389/fpls.2016.01419

[8] Kawasaki Y, Uga H, Kagiwada S, Iyatomi H Basic study of automated diagnosis of viral plant diseases using convolutional neural networks. In: Advances in visual computing: 11th international symposium ISVC 2015 Las Vegas NV USA December 14–16 2015 Proceedings Part II 11, 638–645. Springer International Publishing (2015)

[9] Ma, J., Du, K., Zheng, F., Zhang, L., Gong, Z., Sun, Z.: A recognition method for cucumber diseases using leaf symptom images based on deep convolutional neural network. Comput. Electron. Agric. **154**, 18–24 (2018). https://doi.org/10.1016/j.compag.2018.08.048

[10] Mukti IZ, Biswas D (2019) Transfer learning based plant diseases detection using ResNet50. In: 2019 4th International conference on electrical information and communication technology (EICT), 1–6. IEEE (2019, December)

[11] Chen, J., Chen, J., Zhang, D., Sun, Y., Nanehkaran, Y.A.: Using deep transfer learning for image-based plant disease identification. Comput. Electron. Agric. **173**, 105393 (2020). https://doi.org/10.1016/j.compag.2020.105393

[12] Keceli, A.S., Kaya, A., Catal, C., Tekinerdogan, B.: Deep learning-based multi-task prediction system for plant disease and species detection. Eco. Inform. **69**, 101679 (2022). https://doi.org/10.1016/j.ecoinf.2022.101679

[13] Lee SH, Goëau, H., Bonnet, P., & Joly, A.: Conditional multi-task learning for plant disease identification. In: 2020 25th international conference on pattern recognition (ICPR), 3320–3327. IEEE (2021, January). https://doi.org/10.1109/ICPR48806.2021.9412180

[14] Thakur PS, Khanna P, Sheorey T, Ojha A Vision transformer for plant disease detection: PlantViT. In: International Conference on Computer Vision and Image Processing, 501–511. Cham: Springer International Publishing (2021, December)

[15] Boukabouya RA, Moussaoui A, Berrimi M: Vision Transformer based models for plant disease detection and diagnosis. In: 2022 5th international symposium on informatics and its applications (ISIA), 1–6. IEEE (2022, November). https://doi.org/10.1109/ISIA.2022.9923797

[16] Thai HT, Tran-Van NY, Le KH Artificial cognition for early leaf disease detection using vision transformers. In: 2021 International conference on advanced technologies for communications (ATC), 33–38. IEEE (2021, October). https://doi.org/10.1109/ATC53345.2021.9631065

[17] Thakur PS, Khanna P, Sheorey T, Ojha, A (2022) Explainable vision transformer enabled convolutional neural network for plant disease identification: PlantXViT. arXiv preprint arXiv:2207.07919

[18] Zhu, D., Tan, J., Wu, C., Yung, K., Ip, A.W.: Crop disease identification by fusing multiscale convolution and vision transformer. Sensors **23**(13), 6015 (2023). https://doi.org/10.3390/s23136015

[19] Li, X., Li, S.: Transformer help CNN see better: a lightweight hybrid apple disease identification model based on transformers. Agriculture **12**(6), 884 (2022). https://doi.org/10.3390/agriculture12060884

[20] Li, X., Li, X., Zhang, S., Zhang, G., Zhang, M., Shang, H.: SLViT: Shuffle-convolution-based lightweight Vision transformer for effective diagnosis of sugarcane leaf diseases. J King Saud University-Comput Inform Sci **35**(6), 101401 (2023). https://doi.org/10.1016/j.jksuci.2023.01.014

[21] Hu R, Singh A (2021) Unit: Multimodal multitask learning with a unified transformer. In: Proceedings of the IEEE/CVF international conference on computer vision, 1439–1449. https://doi.org/10.1109/ICCV48922.2021.00146

[22] Bhattacharjee D, Zhang T, Süsstrunk S, Salzmann M (2022) Mult: An end-to-end multitask learning transformer. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 12031–12041 https://doi.org/10.1109/CVPR52688.2022.01173

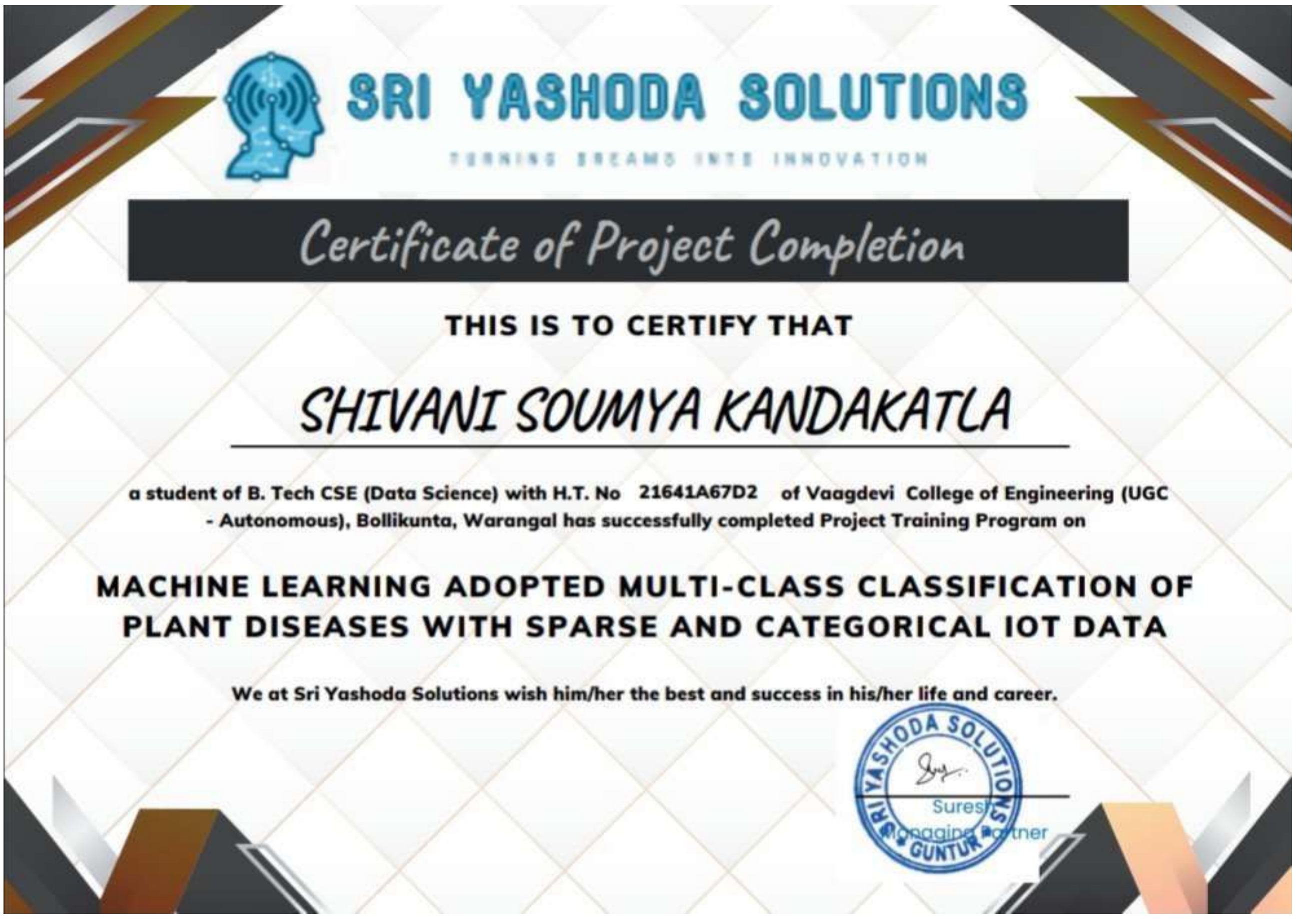
[23] Park S, Kim G, Oh Y, Seo JB, Lee SM, Kim JH, Ye JC (2022) Multi-task vision transformer using low-level chest X-ray feature corpus for COVID-19 diagnosis and severity quantification. Med Image Anal 75, 102299 (2022). https://doi.org/10.1016/j.media.2021.102299

[24] Tian, Y., Bai, K.: End-to-end multitask learning with vision transformer. IEEE Trans Neural Netw Learn Syst (2023). https://doi.org/10.1109/TNNLS.2023.3278896

[25] Goncalves DN, Junior JM, Zamboni P, Pistori H, Li J, Nogueira K, Goncalves WN (2023) MTLSegFormer: Multi-task learning with transformers for semantic segmentation in precision

agriculture. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 6289–6297. https://doi.org/10.1109/CVPR52688.2023.01278

[26] Vishnoi VK, Kumar K, Kumar B. Plant disease detection using computational intelligence and image processing. Berlin Heidelberg: Springer; 2021.



SRI YASHODA SOLUTIONS

TURNING DREAMS INTO INNOVATION

Certificate of Project Completion

THIS IS TO CERTIFY THAT

SHIVANI SOUMYA KANDAKATLA

a student of B. Tech CSE (Data Science) with H.T. No 21641A67D2 of Vaagdevi College of Engineering (UGC - Autonomous), Bollikunta, Warangal has successfully completed Project Training Program on

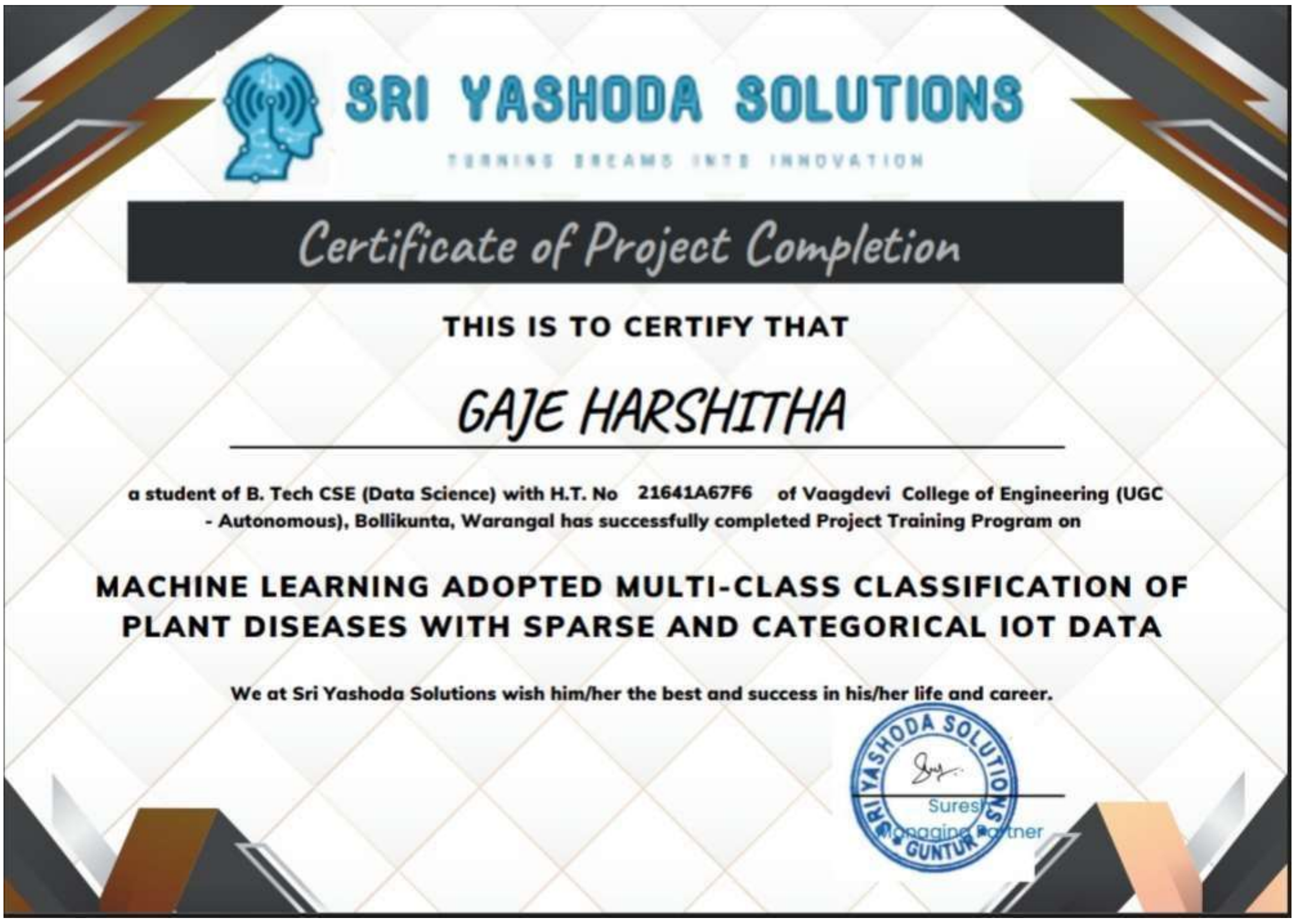
MACHINE LEARNING ADOPTED MULTI-CLASS CLASSIFICATION OF PLANT DISEASES WITH SPARSE AND CATEGORICAL IOT DATA

We at Sri Yashoda Solutions wish him/her the best and success in his/her life and career.

SRI YASHODA SOLUTIONS GUNTUR

[illegible]

Suresh
Managing Partner



SRI YASHODA SOLUTIONS

TURNING DREAMS INTO INNOVATION

Certificate of Project Completion

THIS IS TO CERTIFY THAT

GAJE HARSHITHA

a student of B. Tech CSE (Data Science) with H.T. No 21641A67F6 of Vaagdevi College of Engineering (UGC - Autonomous), Bollikunta, Warangal has successfully completed Project Training Program on

MACHINE LEARNING ADOPTED MULTI-CLASS CLASSIFICATION OF PLANT DISEASES WITH SPARSE AND CATEGORICAL IOT DATA

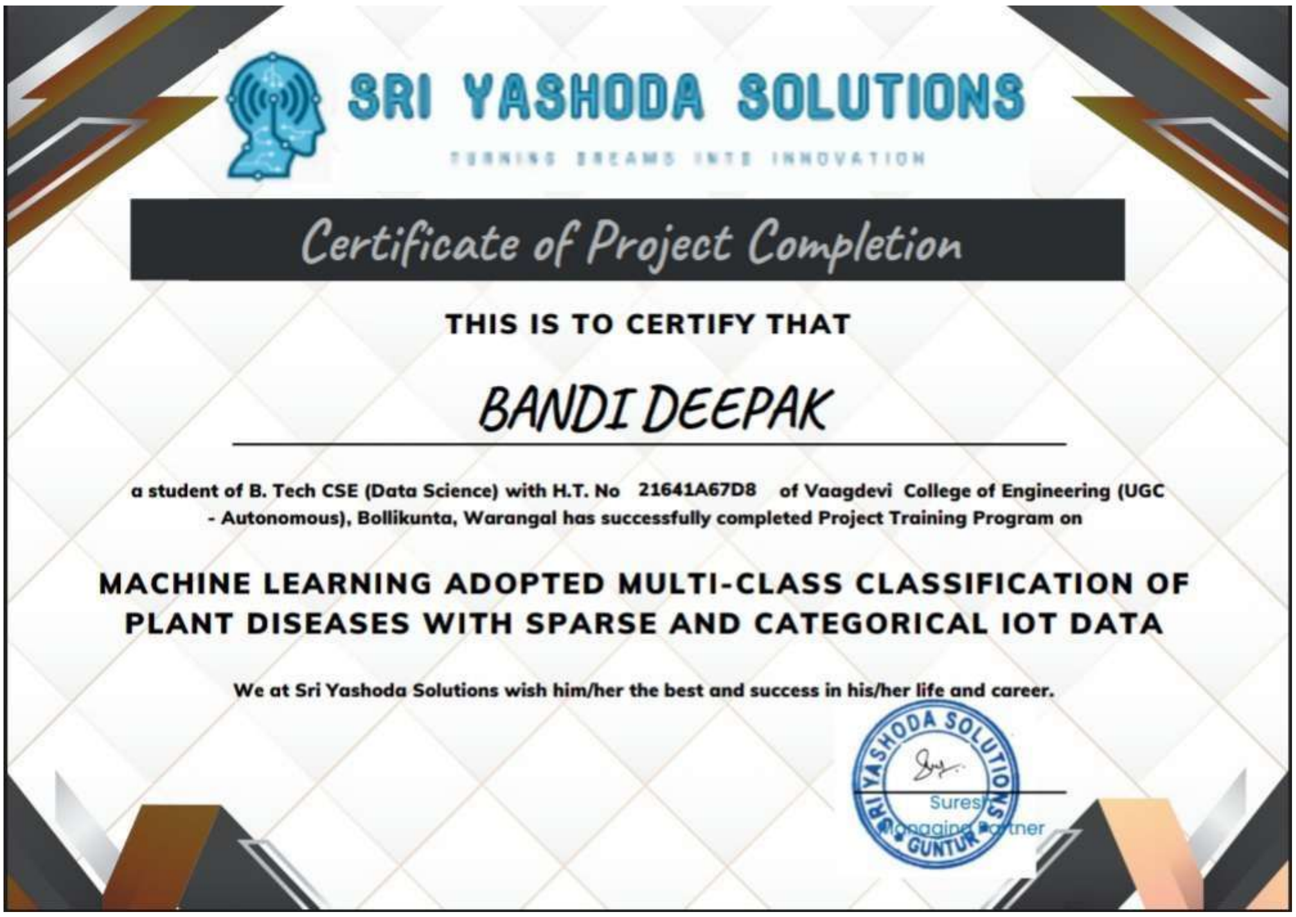
We at Sri Yashoda Solutions wish him/her the best and success in his/her life and career.

SRI YASHODA SOLUTIONS GUNTUR

[illegible]

Suresh

Managing Partner



SRI YASHODA SOLUTIONS

TURNING DREAMS INTO INNOVATION

Certificate of Project Completion

THIS IS TO CERTIFY THAT

BANDI DEEPAK

a student of B. Tech CSE (Data Science) with H.T. No 21641A67D8 of Vaagdevi College of Engineering (UGC - Autonomous), Bollikunta, Warangal has successfully completed Project Training Program on

MACHINE LEARNING ADOPTED MULTI-CLASS CLASSIFICATION OF PLANT DISEASES WITH SPARSE AND CATEGORICAL IOT DATA

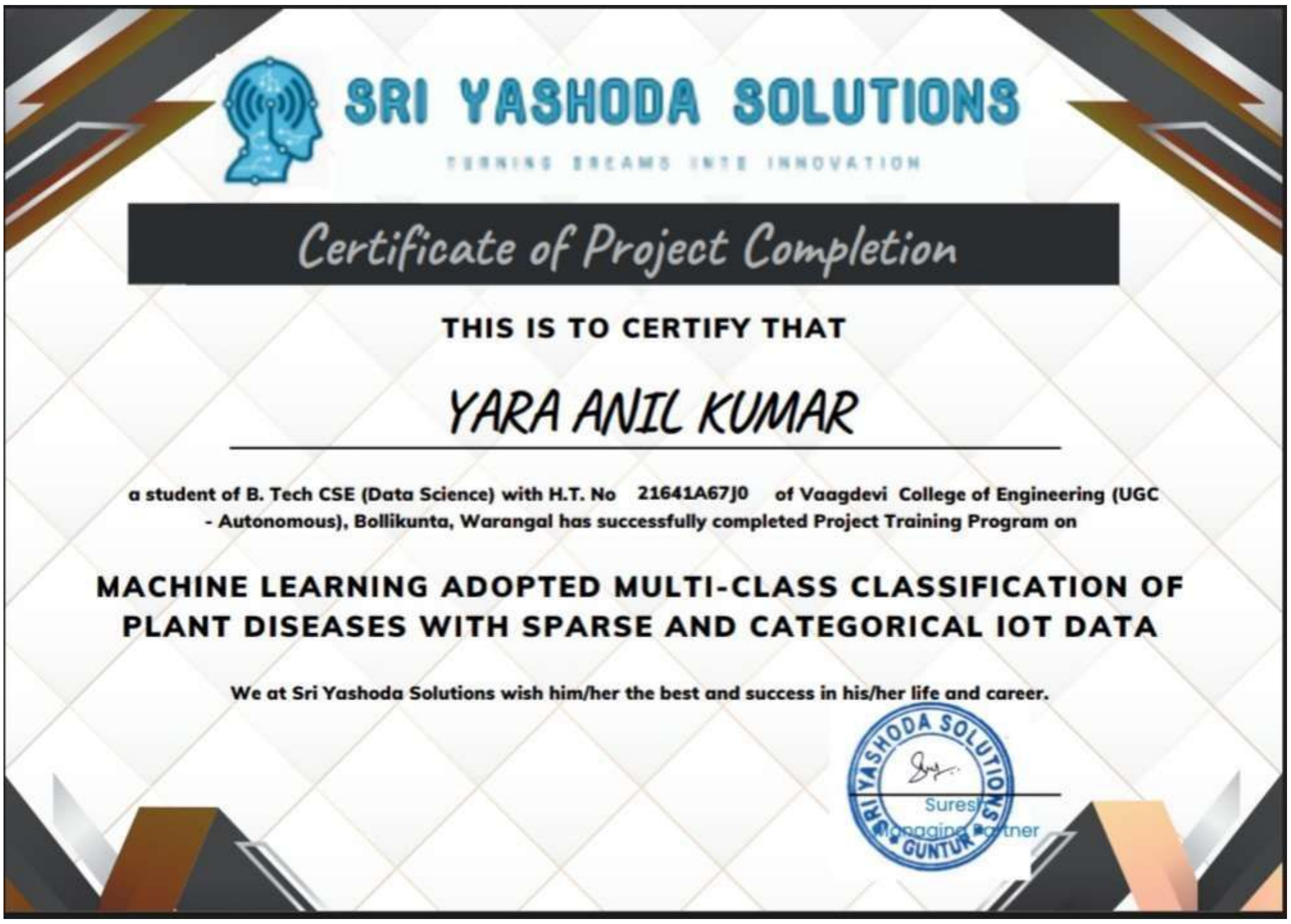
We at Sri Yashoda Solutions wish him/her the best and success in his/her life and career.

SRI YASHODA SOLUTIONS GUNTUR

[illegible]

Suresh

Managing Partner



SRI YASHODA SOLUTIONS

TURNING DREAMS INTO INNOVATION

Certificate of Project Completion

THIS IS TO CERTIFY THAT

YARA ANIL KUMAR

a student of B. Tech CSE (Data Science) with H.T. No 21641A67J0 of Vaagdevi College of Engineering (UGC - Autonomous), Bollikunta, Warangal has successfully completed Project Training Program on

MACHINE LEARNING ADOPTED MULTI-CLASS CLASSIFICATION OF PLANT DISEASES WITH SPARSE AND CATEGORICAL IOT DATA

We at Sri Yashoda Solutions wish him/her the best and success in his/her life and career.

SRI YASHODA SOLUTIONS GUNTUR

[illegible]

Suresh

Managing Partner