

Crop Disease Diagnosis for Maize and Coffee

1. Use Case Description

- **Problem:** Farmers in Rwanda and across Africa face significant challenges in identifying and managing diseases in maize and coffee crops. These diseases lead to reduced yields, economic losses, and food insecurity.
- **Solution:** A digital tool for diagnosing crop diseases and recommending treatments. □ **Relevance:**
 - Maize and coffee are critical crops for food security and economic stability in Rwanda.
 - Early diagnosis and treatment can prevent yield losses and improve farmers' livelihoods.
 - The tool will empower farmers with limited access to agricultural experts.

2. Stakeholders

Stakeholders are individuals or groups who have an interest in or are affected by the crop disease diagnosis system. Here's a detailed breakdown:

Primary Stakeholders:

- a) **Farmers:**
 - **Role:** End-users of the diagnostic tool.
 - **Needs:** Accurate and timely disease diagnosis, affordable treatment options, and easy-to-use tools.
- b) **Agricultural Experts:**
 - **Role:** Provide knowledge on disease symptoms, treatments, and prevention methods.
 - **Needs:** A reliable system to disseminate their expertise to farmers.
- c) **Government Agencies:**
 - **Role:** Support agricultural development and food security initiatives.
 - **Needs:** Data on disease prevalence to inform policy decisions and allocate resources.
- d) **NGOs/Development Organizations:**
 - **Role:** Promote sustainable farming practices and provide support to farmers.
 - **Needs:** Tools to enhance their outreach and impact.
- e) **Technology Providers:**
 - **Role:** Develop and maintain the diagnostic tool.
 - **Needs:** Clear requirements and feedback to improve the system.

Secondary Stakeholders:

a) Local Communities:

- o **Role:** Benefit from improved food security and economic stability.

b) Researchers: o **Role:** Study disease patterns and develop improved solutions. c) **Input Suppliers:**

- o **Role:** Provide seeds, fertilizers, and pesticides recommended by the system

3. Identified Rules (IF-THEN Rules)

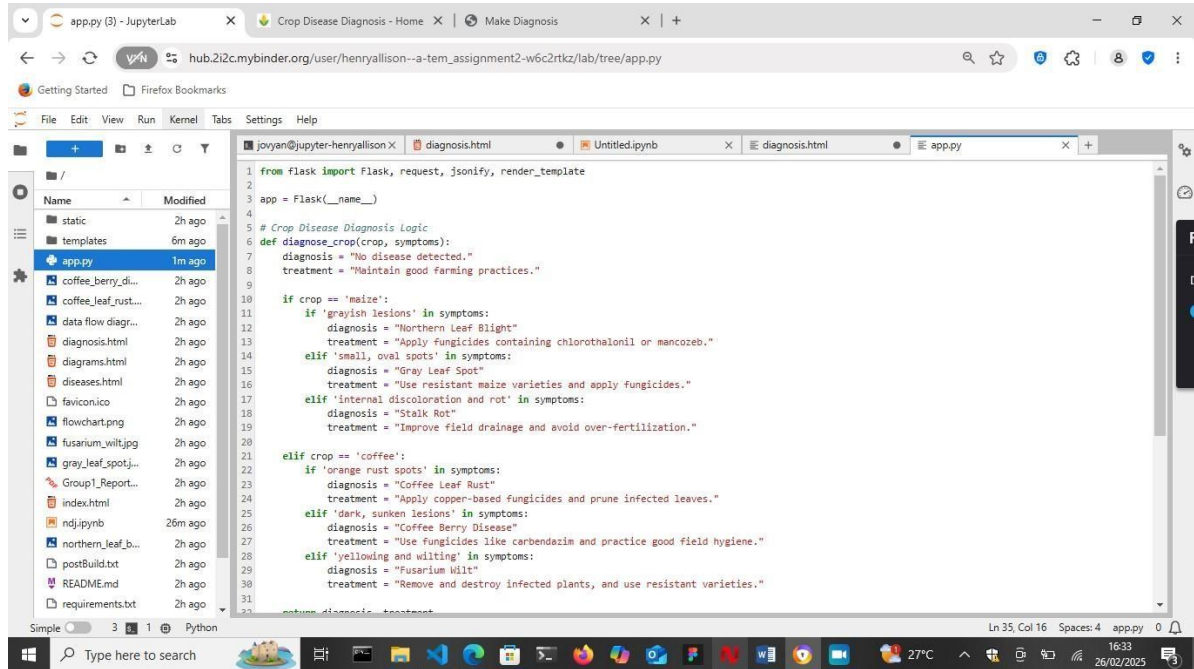
□ **Maize Diseases:**

- a) **IF** leaves have grayish lesions with yellow halos, **THEN** it is **Northern Leaf Blight**.
 - **Treatment:** Apply fungicides containing chlorothalonil or mancozeb.
- b) **IF** leaves have small, oval, or elongated spots with a gray center, **THEN** it is **Gray Leaf Spot**.
 - **Treatment:** Use resistant maize varieties and apply fungicides.
- c) **IF** the stalk has internal discoloration and rot, **THEN** it is **Stalk Rot**.
 - **Treatment:** Improve field drainage and avoid over-fertilization. □

Coffee Diseases:

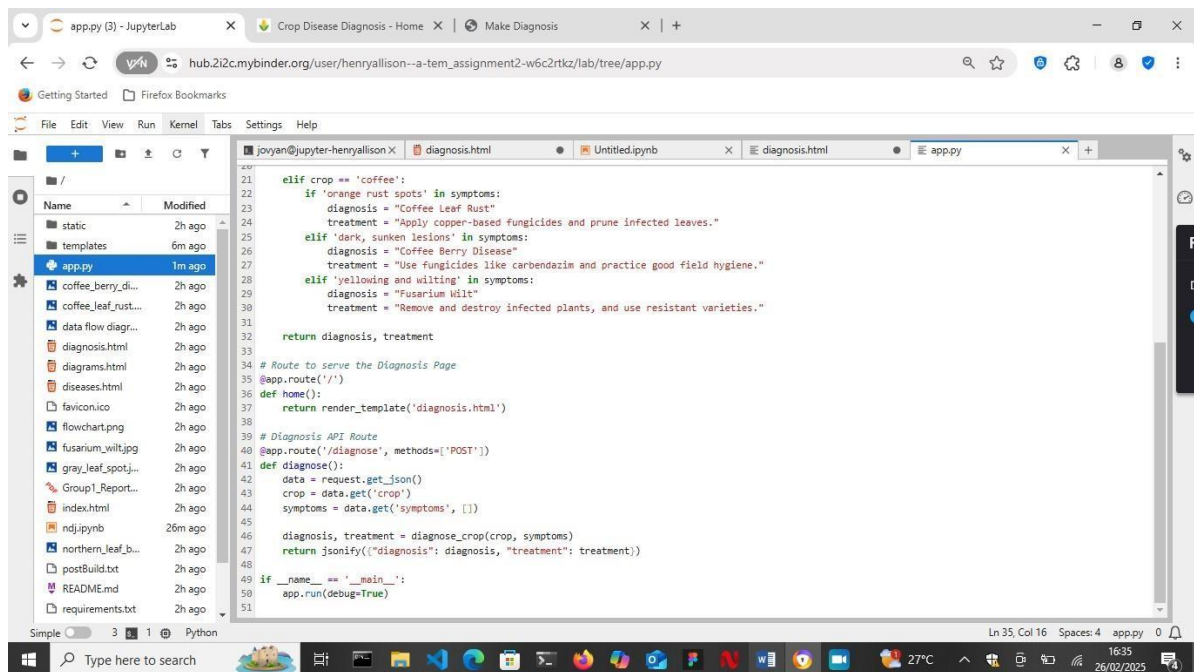
- a) **IF** leaves have orange rust spots, **THEN** it is **Coffee Leaf Rust**.
 - **Treatment:** Apply copper-based fungicides and prune infected leaves.
- b) **IF** berries have dark, sunken lesions, **THEN** it is **Coffee Berry Disease**.
 - **Treatment:** Use fungicides like carbendazim and practice good field hygiene.
- c) **IF** leaves show yellowing and wilting, **THEN** it is **Fusarium Wilt**.
 - **Treatment:** Remove and destroy infected plants, and use resistant varieties.

Python Flask backend codes



The screenshot shows a JupyterLab environment with a file browser on the left and a code editor in the center. The file browser lists files such as static, templates, app.py, coffee_berry_di..., coffee_leaf_rust..., data_flow_diagr..., diagnosis.html, diagrams.html, diseases.html, faviconico, flowchart.png, fusarium_wilt.jpg, gray_leaf_spotj..., Group1_Report..., index.html, ndj.ipynb, northern_leaf_b..., postBuild.txt, README.md, and requirements.txt. The code editor displays the following Python code:

```
1 from flask import Flask, request, jsonify, render_template
2
3 app = Flask(__name__)
4
5 # Crop Disease Diagnosis Logic
6 def diagnose_crop(crop, symptoms):
7     diagnosis = "No disease detected."
8     treatment = "Maintain good farming practices."
9
10    if crop == 'maize':
11        if 'grayish lesions' in symptoms:
12            diagnosis = "Northern Leaf Blight"
13            treatment = "Apply fungicides containing chlorothalonil or mancozeb."
14        elif 'small, oval spots' in symptoms:
15            diagnosis = "Gray Leaf Spot"
16            treatment = "Use resistant maize varieties and apply fungicides."
17        elif 'internal discoloration and rot' in symptoms:
18            diagnosis = "Stalk Rot"
19            treatment = "Improve field drainage and avoid over-fertilization."
20
21    elif crop == 'coffee':
22        if 'orange rust spots' in symptoms:
23            diagnosis = "Coffee Leaf Rust"
24            treatment = "Apply copper-based fungicides and prune infected leaves."
25        elif 'dark, sunken lesions' in symptoms:
26            diagnosis = "Coffee Berry Disease"
27            treatment = "Use fungicides like carbendazim and practice good field hygiene."
28        elif 'yellowing and wilting' in symptoms:
29            diagnosis = "Fusarium Wilt"
30            treatment = "Remove and destroy infected plants, and use resistant varieties."
31
32    return diagnosis, treatment
33
```



The screenshot shows the same JupyterLab environment, but the code editor now displays the continuation of the Python code:

```
21
22    elif crop == 'coffee':
23        if 'orange rust spots' in symptoms:
24            diagnosis = "Coffee Leaf Rust"
25            treatment = "Apply copper-based fungicides and prune infected leaves."
26        elif 'dark, sunken lesions' in symptoms:
27            diagnosis = "Coffee Berry Disease"
28            treatment = "Use fungicides like carbendazim and practice good field hygiene."
29        elif 'yellowing and wilting' in symptoms:
30            diagnosis = "Fusarium Wilt"
31            treatment = "Remove and destroy infected plants, and use resistant varieties."
32
33    return diagnosis, treatment
34
35 # Route to serve the Diagnosis Page
36 @app.route('/')
37 def home():
38     return render_template('diagnosis.html')
39
40 # Diagnosis API Route
41 @app.route('/diagnose', methods=['POST'])
42 def diagnose():
43     data = request.get_json()
44     crop = data.get('crop')
45     symptoms = data.get('symptoms', [])
46
47     diagnosis, treatment = diagnose_crop(crop, symptoms)
48     return jsonify({"diagnosis": diagnosis, "treatment": treatment})
49
50 if __name__ == '__main__':
51     app.run(debug=True)
```

Testing Report for Crop Disease Diagnosis System

1. Introduction

We conducted a comprehensive testing process for our Crop Disease Diagnosis System to ensure its reliability, usability, and performance. Our testing approach covered functional, usability, performance, and deployment aspects to identify potential issues and refine the system accordingly.

2. Testing Procedures

2.1 Functional Testing

We performed the following functional tests:

- **Form Submission Test:** We tested whether users could enter crop symptoms and receive a diagnosis. The system successfully processed multiple symptom inputs and returned appropriate disease predictions.
- **Dropdown Selection Test:** We ensured that the crop selection dropdown correctly passed the selected value to the backend.
- **Dynamic Symptom Addition Test:** We validated that users could dynamically add multiple symptoms without encountering interface issues.
- **Backend API Response Test:** We verified that the Flask backend correctly processed input and returned JSON responses in the expected format.

2.2 Usability Testing

- **User Interface (UI) Evaluation:** We assessed the UI layout and accessibility on both desktop and mobile devices.
- **Navigation Responsiveness:** We tested how the navigation bar adapted to different screen sizes and whether the mobile menu functioned correctly.
- **Error Handling Test:** We deliberately submitted incomplete or incorrect inputs to observe how well the system guided users with validation messages.

2.3 Performance Testing

- **Response Time Evaluation:** We measured the time taken for form submission and diagnosis retrieval to ensure acceptable performance.
- **Load Testing:** We manually conducted multiple simultaneous user requests to check the system's capability to handle concurrent access.
- **Binder Hosting Stability:** We tested the backend's availability on Binder and observed its behavior over extended usage sessions.

2.4 Debugging Tests

- **Console Debugging:** We used browser developer tools to check for JavaScript errors that could affect form functionality.

- **Network Monitoring:** We analysed API request and response logs to ensure correct communication between the frontend and backend.
- **Server Log Inspection:** We examined Flask logs for error messages and traced execution paths to fix potential bugs.

3. Challenges Faced

3.1 Hosting Challenges on Binder

- **Session Timeout:** The Flask application hosted on Binder shuts down after a short period of inactivity, causing disruptions.
- **Cold Start Delays:** Binder takes time to start the server when users access the system after a period of inactivity.
- **Limited Resource Allocation:** Binder has restrictions on CPU and memory usage, which could affect performance under high load.

3.2 Frontend Limitations

- **Browser Compatibility Issues:** Certain UI elements did not render properly on older versions of Safari.
- **Delayed API Responses:** Under certain conditions, slow internet connections resulted in longer response times from the Flask backend.

3.3 System Logic Limitations

- **Limited Symptom Matching:** The current rule-based diagnosis system depends on predefined symptom-disease mappings, making it less flexible for new or uncommon symptoms.
- **Lack of Machine Learning Integration:** The system does not yet incorporate AI-based learning to improve accuracy dynamically.

4. Limitations of the System

- **Short-Term Hosting Reliability:** Since Binder is not a production-level hosting service, it is unsuitable for long-term deployment.
- **Scalability Concerns:** The Flask app is limited in handling a high number of concurrent users due to hosting constraints.
- **Static Rule-Based Diagnosis:** The system does not learn from past user inputs, limiting its adaptability to new disease patterns.
- **No Offline Mode:** The system requires an internet connection to function, making it inaccessible in areas with poor connectivity.

5. Conclusion and Recommendations

Our testing process helped identify key strengths and limitations in the Crop Disease Diagnosis System. While the system performs well in its current form, improvements are needed for long-term scalability and reliability. We recommend transitioning the backend to a more stable

hosting solution, such as PythonAnywhere or a dedicated cloud server, to overcome Binder's limitations. Additionally, integrating machine learning could enhance diagnosis accuracy and system intelligence over time. By addressing these challenges, we can improve the effectiveness of our system and ensure a more seamless experience for farmers seeking crop disease diagnoses.

User Manual for Crop Disease Diagnosis System

Index

1. **Introduction**
2. **User Goals**
3. **User Stories**
4. **System Features**
5. **How to Use the System**
6. **Common Diseases**
7. **Download Report**
8. **System Diagram**
9. **Conclusion**

1. Introduction

The Crop Disease Diagnosis System is designed to assist farmers in Rwanda with diagnosing diseases affecting maize and coffee crops. By utilizing this web-based application, users can identify potential crop issues based on observed symptoms and receive tailored treatment recommendations.

2. User Goals

- Quickly diagnose crop diseases to minimize loss.
- Access reliable treatment recommendations.
- Enhance overall knowledge of common agricultural diseases.
- Contribute to improved crop yields and food security in Rwanda.

3. User Stories

User Story 1: Aimee the Coffee Farmer

Aimee is a coffee farmer in the Northern Province of Rwanda. Recently, she noticed her coffee plants had yellowing leaves and stunted growth. Concerned about her crop, Aimee visits the Crop Disease Diagnosis System.

1. **Using the System:** She selects "Make Diagnosis" and chooses "Coffee" from the dropdown menu. Aimee describes the symptoms as "yellow leaves and small fruits."
2. **Receiving Diagnosis:** The system processes her input and identifies possible diseases such as coffee leaf rust. It provides Aimee with treatment options, including fungicides and cultural practices to improve crop health.

3. **Taking Action:** Armed with this information, Aimee applies the recommended treatments and monitors her plants closely, leading to healthier crops and a better harvest.

User Story 2: Joseph the Maize Farmer

Joseph is a maize farmer in the Eastern Province. This season, he notices some of his maize plants are wilting and developing brown spots. Unsure of the cause, he turns to the diagnosis system.

1. **Using the System:** Joseph accesses the "Make Diagnosis" page and selects "Maize." He describes the symptoms as "wilting plants with brown spots."
2. **Receiving Diagnosis:** The system diagnoses his maize plants with potential diseases like maize blight. It suggests immediate actions, including fungicide application and proper irrigation techniques.
3. **Taking Action:** Joseph follows the recommendations and successfully reduces the impact of the disease, ensuring a more bountiful harvest.

4. System Features

- **User-Friendly Interface:** Easy navigation for all users, including those with limited technical skills.
- **Disease Diagnosis:** A comprehensive decision tree analyzes user input to provide accurate diagnoses and treatment recommendations.
- **Common Diseases Database:** Information on prevalent diseases affecting maize and coffee, helping users identify issues quickly.
- **Downloadable Resources:** A PDF manual for offline access to the system's functionalities and guides.
- **System Diagram:** Visual representation of how the system operates, illustrating the flow of information from user input to diagnosis.

5. How to Use the System

1. **Navigate to the Home Page:** Access the main overview of the system and its functionalities.
2. **Select Diagnosis:** Click on the "Make Diagnosis" link.
3. **Choose Crop Type:** From the dropdown menu, select either "Coffee" or "Maize."
4. **Input Symptoms:** Describe the symptoms observed in your crops in the text area provided.
5. **Submit the Form:** Click on the submit button to receive diagnosis and treatment recommendations.
6. **Explore Other Features:** Use the navigation bar to access common diseases, download the PDF, or view the system diagram.

6. Common Diseases

This section details the most common diseases affecting maize and coffee in Rwanda, including symptoms, causes, and treatment options. Users can refer to this information to enhance their understanding of crop health.

7. Download Report

A "Download PDF" button is available for users to download the user manual, ensuring they can reference the system's functionalities offline.

8. System Diagram

The system diagram visually represents the architecture of the application, showing how user input flows through the decision tree in the backend to provide diagnoses and recommendations.

9. Conclusion

The Crop Disease Diagnosis System is an invaluable tool for farmers in Rwanda, empowering them to manage crop health effectively. By utilizing this system, farmers can make informed decisions, ultimately improving yields and contributing to food security.