

Automated Agriculture System: Accurate Water Management and Disease Analysis

24-25J-164





Our Team

Sharan U.

IT21381690

Nakulabasgaran Y.

IT21228230

Sankeethan Y.

IT21228162

Mayootharan P.

IT21290206

Content

01

Background

02

Research Problem

03

Objectives

04

Overall System Diagram



Background

Develop an automated agriculture system integrating sensors, data analytics, and machine learning for optimized irrigation and disease management.



What is Automated Agriculture System: Accurate Water Management and Disease Analysis



Why are we focusing on this topic?



What are the main problems we identified?

Research Problem

01

Irregular rainfall patterns, including droughts and floods, can disrupt planting schedules, affect germination, and reduce crop yield.

02

Extreme temperatures, both hot and cold, can stress crops, disrupt flowering and fruit set, and lead to yield losses.

03

Limited water availability can lead to reduced irrigation, affecting crop growth and yield.

04

Farmers often lack awareness of crop diseases and their symptoms, hindering effective outbreak management.

Objectives

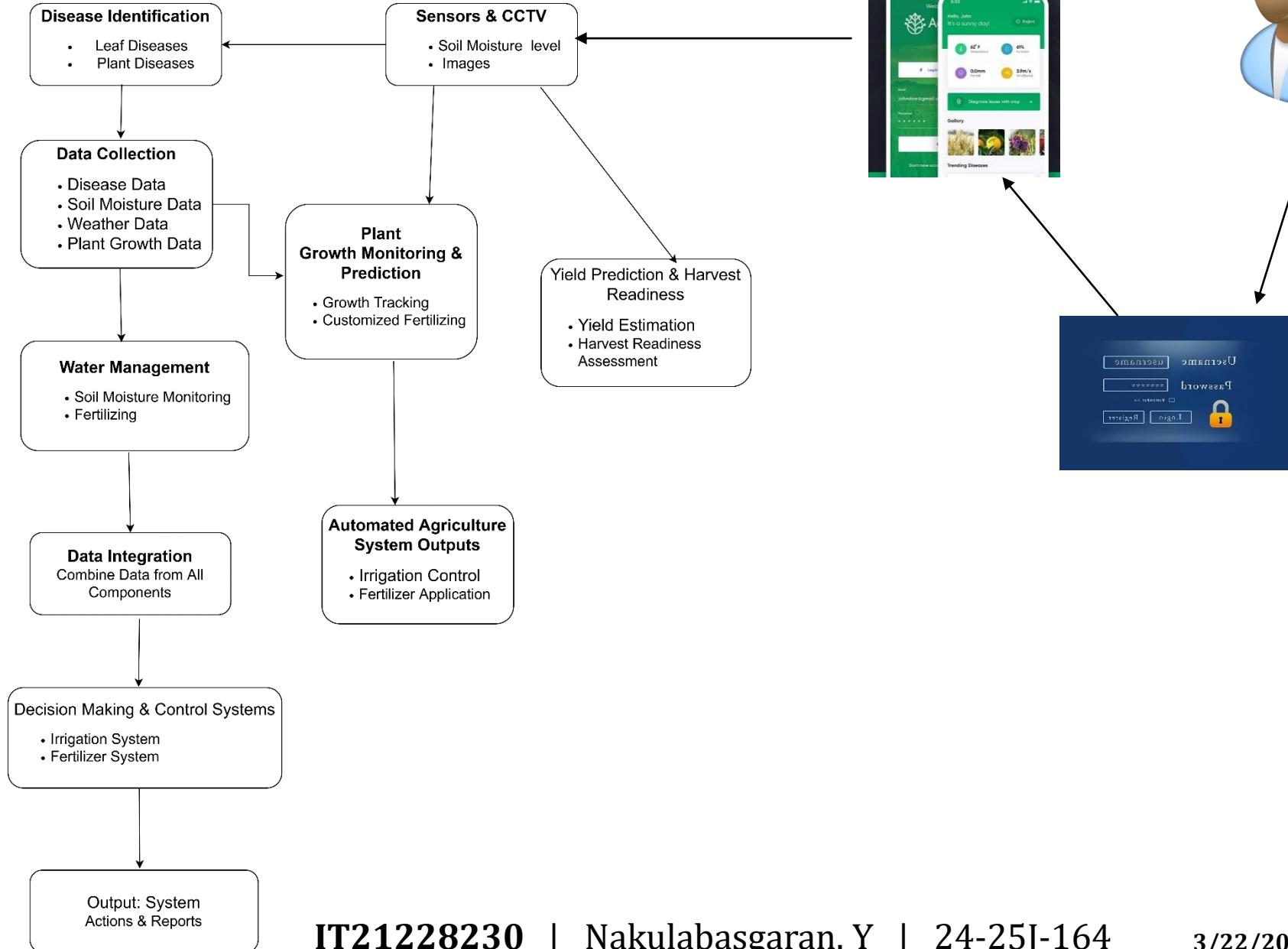
IMPLEMENT ADVANCED
WATER MANAGEMENT
TECHNOLOGY

DEVELOP A DISEASE
IDENTIFICATION SYSTEM

INTEGRATED WEATHER
FORECASTING AND PLANT
GROWTH MONITORING SYSTEM
BASED ON SOIL MOISTURE
ANALYSIS

DEVELOP A MODEL FOR
EFFICIENT YIELD
PREDICTION, HARVEST
READINESS, AND
FERTILIZER
OPTIMIZATION

Overall System Diagram



IT21228230 - Nakulabasgaran. Y

Information Technology

Water Management Technology



Introduction

O1

Background

O2

Research Problem

O3

Objectives



Background

This component focuses on integrating advanced technologies to optimize water usage in agricultural systems, ensuring sustainable practices and reducing water wastage while maintaining crop health.



Research Problem

Inconsistent watering practices can lead to over- or under-watering, affecting crop health and yield



OBJECTIVES

Water Conservation:
Minimize water waste by implementing precision irrigation techniques.

Automation:
Automate irrigation systems to deliver water precisely when and where needed.



Core Objectives



01

Soil Moisture-Based Watering

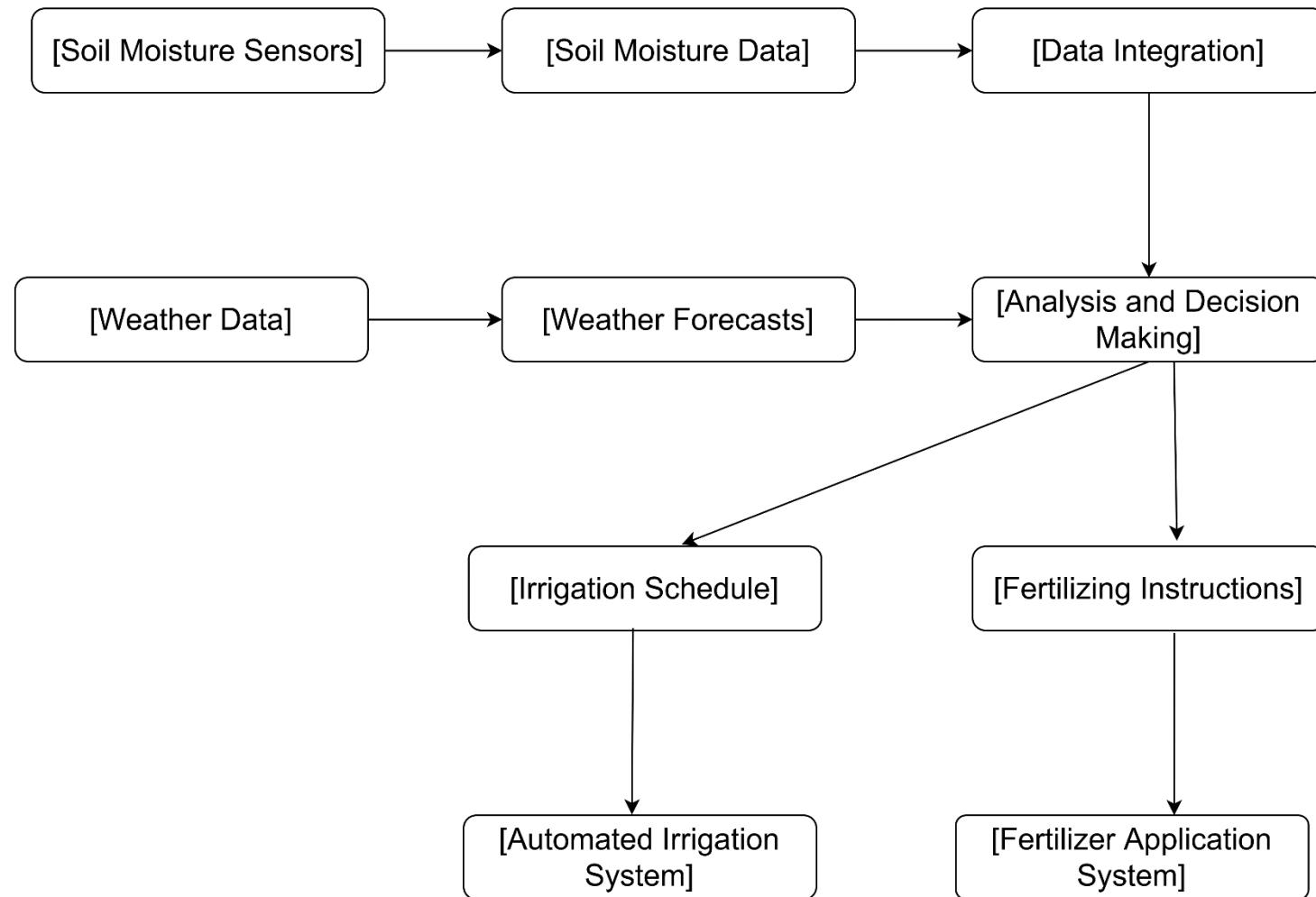
02

Weather Forecast Integration

03

Automated Watering System

System Diagram



Technologies



- Python



- Flask



- Firebase



- Arduino IDE
- PyCharm

Tools

IOT-Device Circuits Items



Submersible Mini pump :
It give water to soil when the sensor sends the signal that the moisture level in the soil is low.



Channel-5V-Relay-Module :
turn on and off the pump automatically as the required water level is reached



ESP32 Microcontroller:
For data processing and communication.

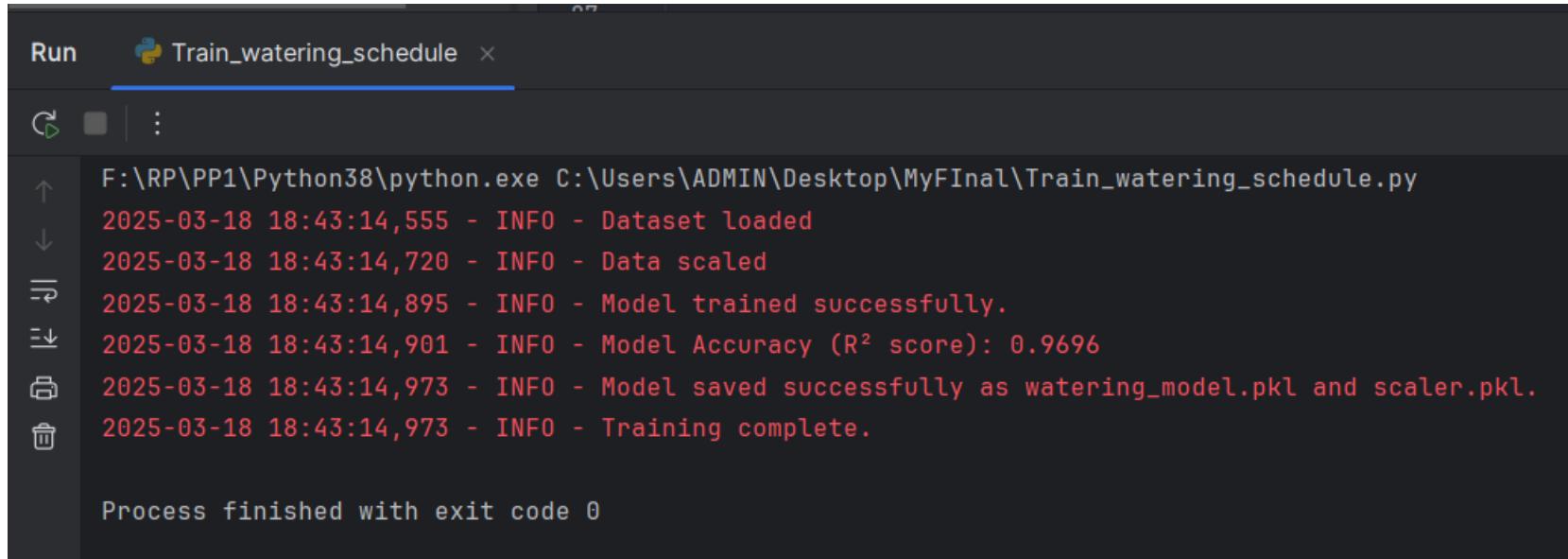
Model

```
1 import pandas as pd
2 from sklearn.preprocessing import StandardScaler
3 from sklearn.ensemble import RandomForestRegressor
4 import joblib
5 import logging
6 from sklearn.model_selection import train_test_split
7 from sklearn.metrics import r2_score
8
9 logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
10
11 def load_and_preprocess_data(file_path): 1usage
12     try:
13         dataset = pd.read_csv(file_path)
14         logging.info("Dataset loaded")
15     except Exception as e:
16         logging.error(f"Error loading dataset: {e}")
17         raise
18
19     # Check for missing values
20     if dataset.isnull().sum().any():
21         logging.warning("Missing values found in the dataset. Filling missing values with the mean.")
22         dataset.fillna(dataset.mean(), inplace=True)
23
24     # Split dataset
25     X = dataset[['Precipitation (mm)', 'Ambient Temp (°C)', 'Humidity (%)', 'Soil Temp (°C)', 'Soil Moisture (%)']]
26     y = dataset['Watering Duration (minutes)']
27
28     return X, y
29
30
```

```
31 def scale_data(X): 1usage
32     scaler = StandardScaler()
33     X_scaled = scaler.fit_transform(X)
34     logging.info("Data scaled")
35     return X_scaled, scaler
36
37
38 def train_model(X_train, y_train): 1usage
39     try:
40         model = RandomForestRegressor(n_estimators=100, random_state=42)
41         model.fit(X_train, y_train)
42         logging.info("Model trained successfully.")
43     except Exception as e:
44         logging.error(f"Error training model: {e}")
45         raise
46     return model
47
48
49 def evaluate_model(model, X_test, y_test): 1usage
50     y_pred = model.predict(X_test)
51     r2 = r2_score(y_test, y_pred)
52     logging.info(f"Model Accuracy (R² score): {r2:.4f}")
53
54
55 def save_model(model, scaler, model_file='watering_model.pkl', scaler_file='scaler.pkl'): 1usage
56     try:
57         joblib.dump(model, model_file)
58         joblib.dump(scaler, scaler_file)
59         logging.info(f"Model saved successfully as {model_file} and {scaler_file}.")
60     except Exception as e:
61         logging.error(f"Error saving model: {e}")
62
```

```
63     |     raise
64
65
66 def main():
67     dataset_file = 'optimal_watering_schedule.csv'
68     X, y = load_and_preprocess_data(dataset_file)
69
70     # Split data into train and test sets
71     X_train, X_test, y_train, y_test = train_test_split(*arrays: X, y, test_size=0.2, random_state=42)
72
73     X_train_scaled, scaler = scale_data(X_train)
74     X_test_scaled = scaler.transform(X_test)
75
76     model = train_model(X_train_scaled, y_train)
77
78     evaluate_model(model, X_test_scaled, y_test)
79
80     save_model(model, scaler)
81
82     logging.info("Training complete.")
83
84
85 ▶ if __name__ == "__main__":
86     main()
87
```

Accuracy



```
Run Train_watering_schedule ×
F:\RP\PP1\Python38\python.exe C:\Users\ADMIN\Desktop\MyFinal\Train_watering_schedule.py
2025-03-18 18:43:14,555 - INFO - Dataset loaded
2025-03-18 18:43:14,720 - INFO - Data scaled
2025-03-18 18:43:14,895 - INFO - Model trained successfully.
2025-03-18 18:43:14,901 - INFO - Model Accuracy (R2 score): 0.9696
2025-03-18 18:43:14,973 - INFO - Model saved successfully as watering_model.pkl and scaler.pkl.
2025-03-18 18:43:14,973 - INFO - Training complete.

Process finished with exit code 0
```

BackEnd

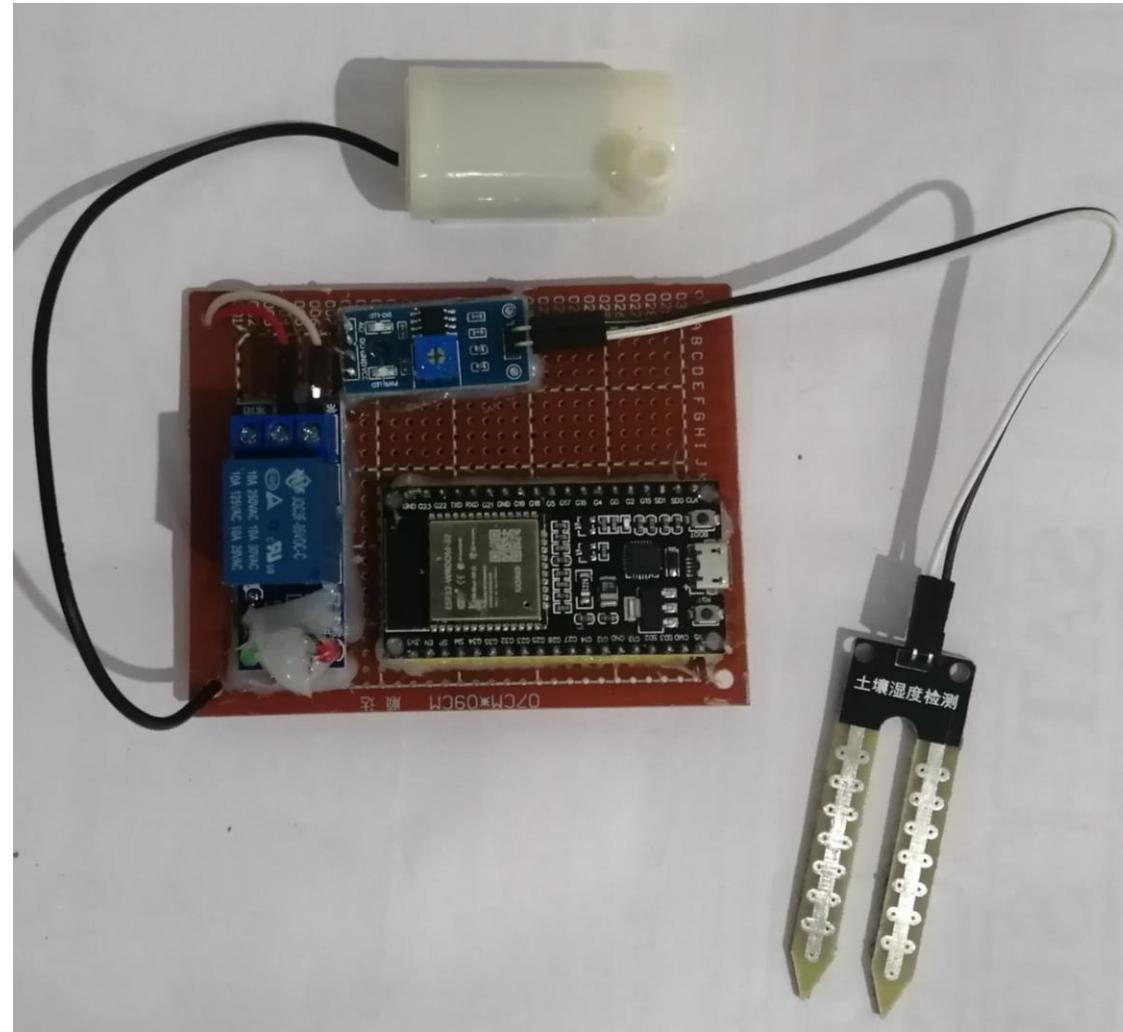
The screenshot shows a Python development environment with the following details:

- Project Structure:** The project is named "MyFinal" located at C:\Users\ADMIN\Desktop\MyFinal. It contains a "static" folder, a "templates" folder with "holder.html" and "index.html" files, and several Python files: "App.py", "FaskAPI.py", "optimal_watering_schedule.csv", "Predic_Watering.py", "scaler.pkl", "Train_watering_schedule.py", and "watering_model.pkl".
- Code Editor:** The file "Predic_Watering.py" is open, displaying the following code:

```
1 import pandas as pd
2 import joblib
3
4 # input data
5 input_data = {
6     'Precipitation (mm)': [4.5],
7     'Ambient Temp (°C)': [30],
8     'Humidity (%)': [54],
9     'Soil Temp (°C)': [25],
10    'Soil Moisture (%)': [24]
11 }
12
13 # Convert to DataFrame
14 input_df = pd.DataFrame(input_data)
15
16 # Load trained model
17 model = joblib.load('watering_model.pkl')
18 scaler = joblib.load('scaler.pkl')
19 scaled_input_data = scaler.transform(input_df)
20
21 # prediction
22 predicted_watering_duration = model.predict(scaled_input_data)
23 print(f'Predicted Watering Duration (minutes): {predicted_watering_duration[0]}')
```

- Run Tab:** The "Run" tab is active, showing the command F:\RP\PP1\Python38\python.exe C:\Users\ADMIN\Desktop\MyFinal\Predic_Watering.py and the output Predicted Watering Duration (minutes): 24.8.
- Status Bar:** The status bar at the bottom indicates Process finished with exit code 0.

Prototype



Requirements

Functional

- Analyze collected data from Sensor.
- Data Processing
- Irrigation systems

Non Functional

- Accuracy
- Reliability
- Cost-Effectiveness
- User-Friendliness
- Security



Completed

- Built prototype 80%
- Integrate Tools
- Model Training



To be Complete

- Connect the prototype with Database
- Web UI Design (Frontend)
- Testing



References

- [1] https://www.researchgate.net/publication/372523323_Automated_Pest_and_Disease_Identification_in_Agriculture_using_Image_Processing
- [2] <https://www.frontiersin.org/journals/plant-science/articles/10.3389/fpls.2024.1356260/full>
- [3] https://www.researchgate.net/publication/374505378_An_Automated_System_to_Detect_Plant_Disease_using_Deep_Learning
- [4]
<https://ieeexplore.ieee.org/document/9754145>
- [5]
<https://www.sciencedirect.com/science/article/abs/pii/S2214785321042115>
- [6] Hillel, D. (2004). *Introduction to Environmental Soil Physics*. Academic Press.
- [7] Allen, R. G., Pereira, L. S., Raes, D., & Smith, M. (1998). *Crop Evapotranspiration: Guidelines for Computing Crop Water Requirements*. FAO Irrigation and Drainage Paper 56.

IT21381690

Sharan.u

Information Technology

Disease Identification



Content

01

Background

02

Research problem

03

Research gab

04

Objectives



BACKGROUND



What are the diseases and how we are going to identify the diseases



What is object Tracking



What is Deep CNN

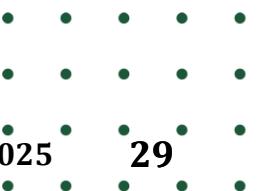


Research Problem

How to track or identify what are the disease in the tree



How can we notify about the disease for the farmers



Objectives

01 Early Detection

Detect diseases at an early stage to prevent their spread and minimize damage.



02 Improve Farmer Knowledge and Practices:

Educate and inform farmers about disease prevention, identification, and management techniques, promoting sustainable farming practices.

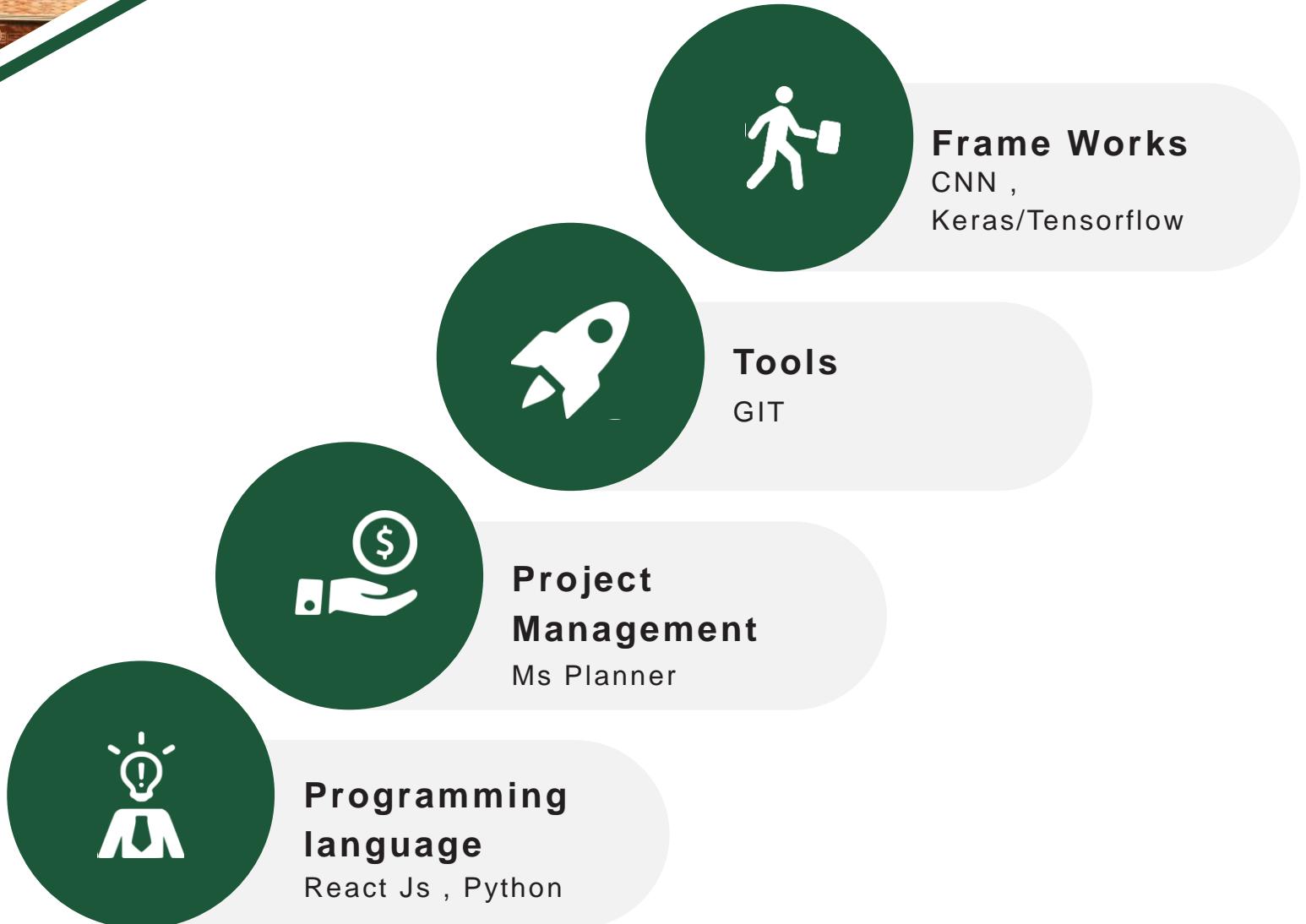


03 Reduce Economic Losses:

Minimize economic losses for farmers by preventing widespread crop failure and reducing the costs associated with disease management.

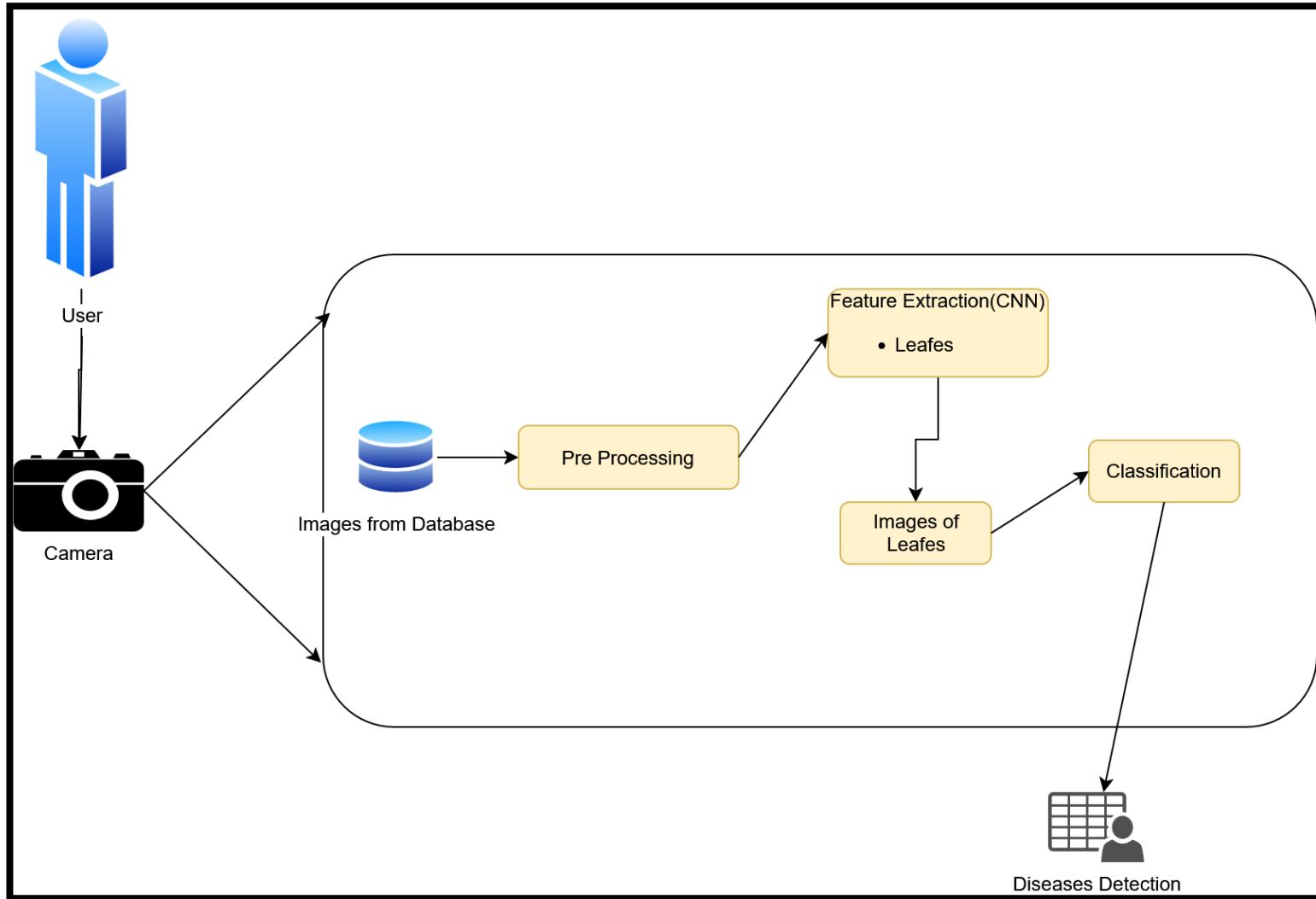


Tools and technology





System Diagram



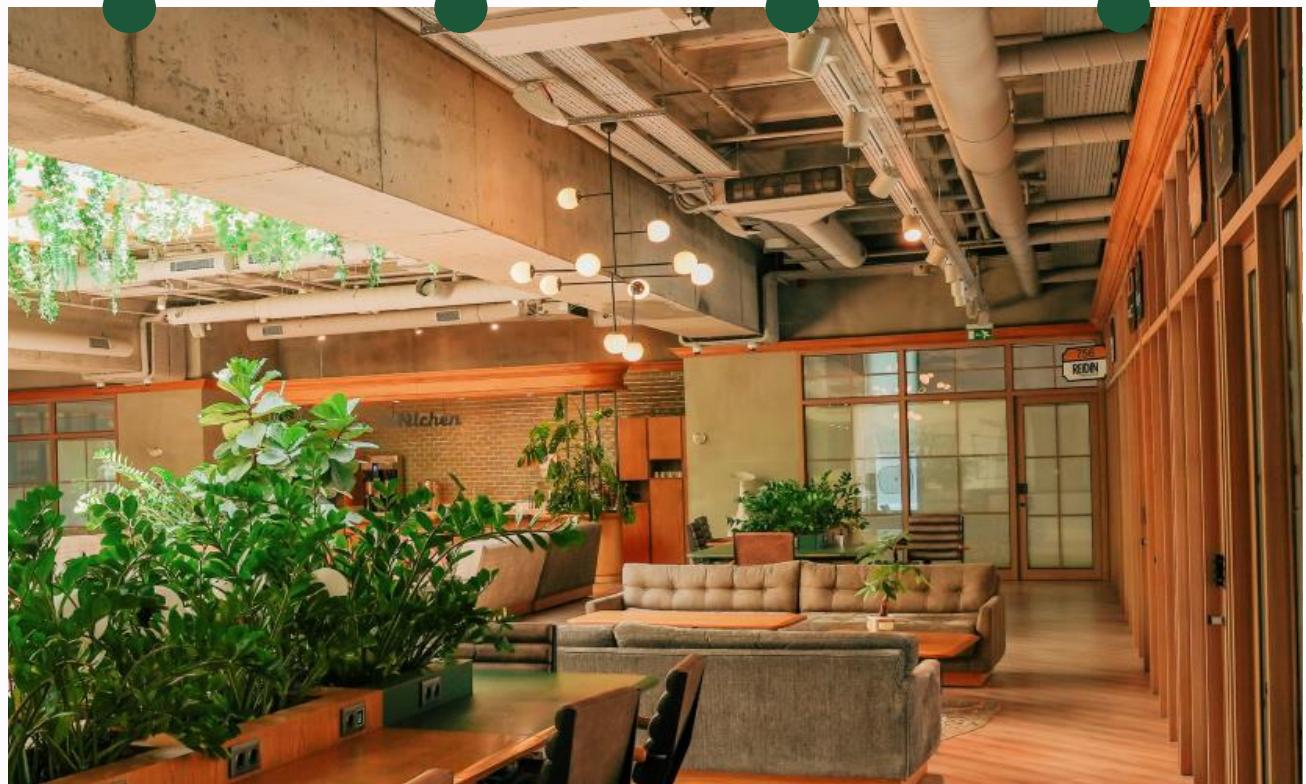
Requirements

Functional

- Image Capture and Input
- Image Preprocessing
- Disease Detection and Classification
- Notification and Reporting

Non Functional

- Accuracy
- performance
- Availability
- usability

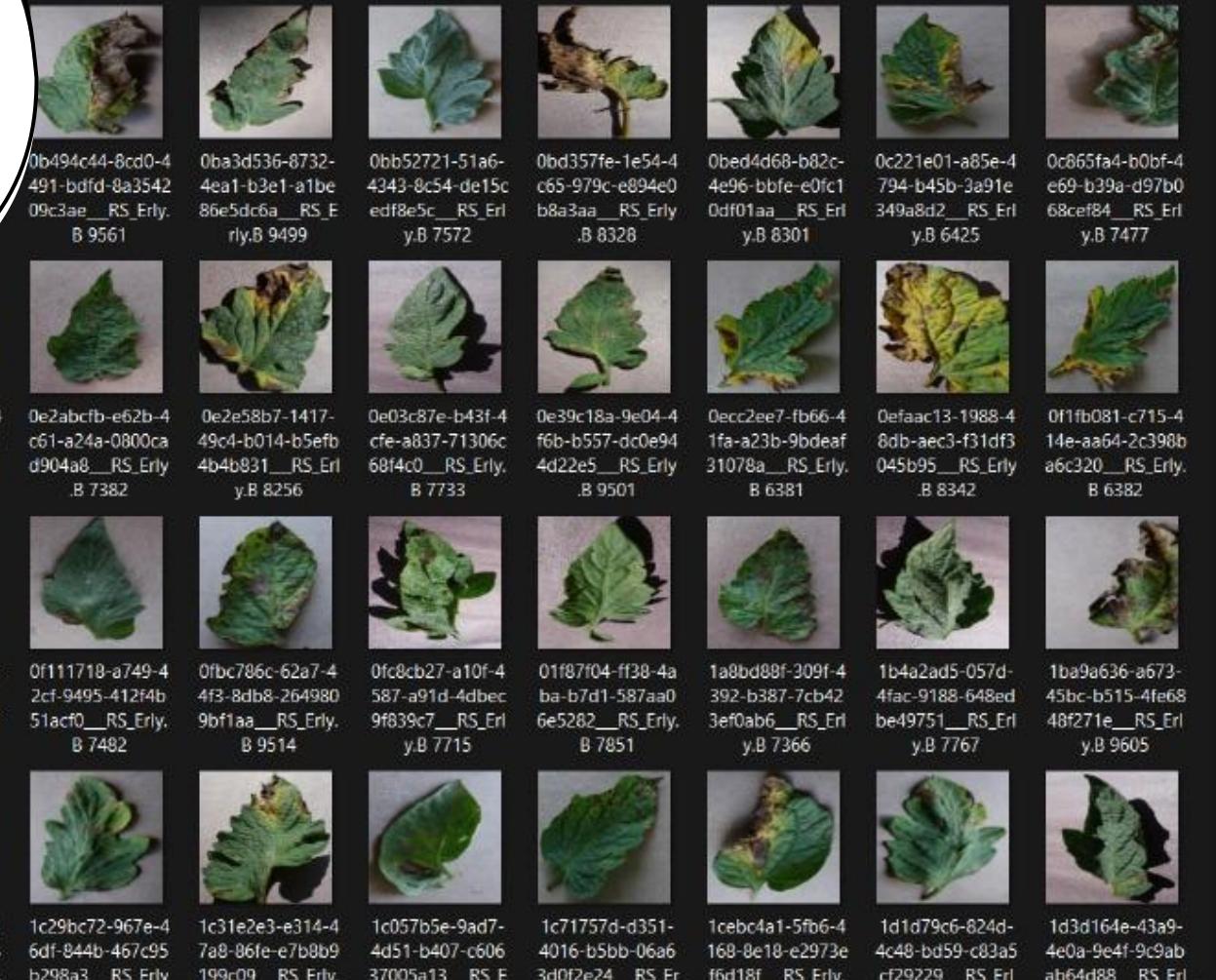


Completion of the Project



Data Set

- Pictures
 - Music
 - Videos
 - NEW
 - archive (9)
 - Tomato_Early_blight
 - Tomato_Late_blight
 - iCloud Drive
 - This PC
 - OS (C:)
 - Local Disk (D:)
 - Network
- 1,000 items |



Model Training

```
112/112 [=====] - 232s 2s/step - loss: 0.7848 - accuracy: 0.6086 - val_loss: 0.6873 - val_accuracy: 0.7232
Epoch 2/50
112/112 [=====] - 206s 2s/step - loss: 0.4766 - accuracy: 0.7984 - val_loss: 0.3200 - val_accuracy: 0.8549
Epoch 3/50
112/112 [=====] - 194s 2s/step - loss: 0.3934 - accuracy: 0.8357 - val_loss: 0.5766 - val_accuracy: 0.7879
Epoch 4/50
112/112 [=====] - 195s 2s/step - loss: 0.3598 - accuracy: 0.8578 - val_loss: 0.6928 - val_accuracy: 0.7165
Epoch 5/50
112/112 [=====] - 191s 2s/step - loss: 0.3421 - accuracy: 0.8609 - val_loss: 0.2980 - val_accuracy: 0.8683
Epoch 6/50
112/112 [=====] - 192s 2s/step - loss: 0.2760 - accuracy: 0.8835 - val_loss: 0.2895 - val_accuracy: 0.8973
Epoch 7/50
112/112 [=====] - 193s 2s/step - loss: 0.3122 - accuracy: 0.8718 - val_loss: 0.2510 - val_accuracy: 0.9018
Epoch 8/50
112/112 [=====] - 196s 2s/step - loss: 0.2323 - accuracy: 0.9099 - val_loss: 0.2089 - val_accuracy: 0.9241
Epoch 9/50
112/112 [=====] - 195s 2s/step - loss: 0.2402 - accuracy: 0.9023 - val_loss: 0.2551 - val_accuracy: 0.8884
Epoch 10/50
112/112 [=====] - 198s 2s/step - loss: 0.2254 - accuracy: 0.9115 - val_loss: 0.2434 - val_accuracy: 0.8951
Epoch 11/50
112/112 [=====] - 195s 2s/step - loss: 0.1837 - accuracy: 0.9303 - val_loss: 0.3928 - val_accuracy: 0.8616
Epoch 12/50
112/112 [=====] - 292s 3s/step - loss: 0.1876 - accuracy: 0.9219 - val_loss: 0.4521 - val_accuracy: 0.8103
Epoch 13/50
112/112 [=====] - 351s 3s/step - loss: 0.2197 - accuracy: 0.9113 - val_loss: 0.3225 - val_accuracy: 0.8728
Epoch 14/50
112/112 [=====] - 359s 3s/step - loss: 0.1689 - accuracy: 0.9351 - val_loss: 0.2297 - val_accuracy: 0.8973
Epoch 15/50
112/112 [=====] - 338s 3s/step - loss: 0.1404 - accuracy: 0.9490 - val_loss: 0.2506 - val_accuracy: 0.9085
Epoch 16/50
112/112 [=====] - 346s 3s/step - loss: 0.1326 - accuracy: 0.9507 - val_loss: 0.2217 - val_accuracy: 0.9062
Epoch 17/50
112/112 [=====] - 352s 3s/step - loss: 0.1352 - accuracy: 0.9527 - val_loss: 0.1301 - val_accuracy: 0.9509
Epoch 18/50
112/112 [=====] - 342s 3s/step - loss: 0.1323 - accuracy: 0.9541 - val_loss: 0.1370 - val_accuracy: 0.9509
Epoch 19/50
112/112 [=====] - 350s 3s/step - loss: 0.1291 - accuracy: 0.9524 - val_loss: 0.1514 - val_accuracy: 0.9397
Epoch 20/50
112/112 [=====] - 348s 3s/step - loss: 0.1681 - accuracy: 0.9378 - val_loss: 0.2712 - val_accuracy: 0.8929
Epoch 21/50
112/112 [=====] - 345s 3s/step - loss: 0.1082 - accuracy: 0.9569 - val_loss: 0.2783 - val_accuracy: 0.9062
Epoch 22/50
112/112 [=====] - 345s 3s/step - loss: 0.1052 - accuracy: 0.9600 - val_loss: 0.1062 - val_accuracy: 0.9710
Epoch 23/50
112/112 [=====] - 349s 3s/step - loss: 0.1240 - accuracy: 0.9538 - val_loss: 0.2360 - val_accuracy: 0.9420
Epoch 24/50
112/112 [=====] - 358s 3s/step - loss: 0.1032 - accuracy: 0.9616 - val_loss: 0.1834 - val_accuracy: 0.9330
Epoch 25/50
```

Model accuracy

Old

```
test_loss, test_accuracy = model.evaluate(test_generator)
print(f"Test accuracy: {test_accuracy:.4f}")

56/56 ━━━━━━━━ 120s 2s/step - accuracy: 0.8432 - loss: 0.4751
Test accuracy: 0.8481
```

New

```
[60]: scores
[60]: [0.11173706501722336, 0.96875]
```

Working component

The screenshot shows the Postman API client interface. On the left, the History sidebar lists multiple POST requests to `http://localhost:5000/predict`. The main workspace displays a POST request to `http://localhost:5000/predict`. The 'Body' tab is selected, showing a form-data key 'file' with the value '00c5c908-fc25-4710-a109-db143da23112_RS_Early_B 7778.JPG'. The 'Pretty' tab in the results section shows the JSON response:

```
1 "confidence": 99.95168447494507,  
2 "predicted_class": "Tomato_Early_blight"
```

BackEnd

```
1 import os
2 import tensorflow as tf
3
4 try:
5     from tensorflow.keras import layers, models
6 except ImportError:
7     print("Failed to import tensorflow.keras. Using tf.keras instead.")
8     from tensorflow import keras
9     layers = keras.layers
10    models = keras.models
11
12 from flask import Flask, request, jsonify
13 from werkzeug.utils import secure_filename
14 import numpy as np
15
16
17 app = Flask(__name__)
18
19 UPLOAD_FOLDER = 'uploads'
20 ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg'}
21 app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
22
23 os.makedirs(UPLOAD_FOLDER, exist_ok=True)
24
25
26 MODEL_PATH = "D:/archive (9)/tomato.h5"
27 INPUT_SHAPE = (256, 256, 3)
```



```
28
29
30     os.remove(file_path)
31     return jsonify({
32         'predicted_class': predicted_label,
33         'confidence': confidence
34     }), 200
35
36     except Exception as e:
37         if os.path.exists(file_path):
38             os.remove(file_path)
39         return jsonify({'error': f'Prediction failed: {str(e)}'}), 500
40
41     return jsonify({'error': 'Invalid file format'}), 400
42
43
44 @app.route('/health', methods=['GET'])
45 def health():
46     return jsonify({'status': 'healthy', 'model_loaded': model is not None}), 200
47
48 if __name__ == '__main__':
49     os.environ['TF_ENABLE_ONEDNN_OPTS'] = '0'
50     print(f"Starting Flask app with TensorFlow {tf.__version__}")
51     app.run(debug=True, host='0.0.0.0', port=5000)
```



```
@app.route('/predict', methods=['POST'])
def predict():
    if 'file' not in request.files:
        return jsonify({'error': 'No file part in the request'}), 400

    file = request.files['file']
    if file.filename == '':
        return jsonify({'error': 'No file selected'}), 400

    if file and allowed_file(file.filename):
        filename = secure_filename(file.filename)
        file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
        file.save(file_path)

        try:
            img_array = preprocess_image(file_path)
            predictions = model.predict(img_array)
            predicted_class_idx = np.argmax(predictions[0])
            confidence = float(predictions[0][predicted_class_idx]) * 100

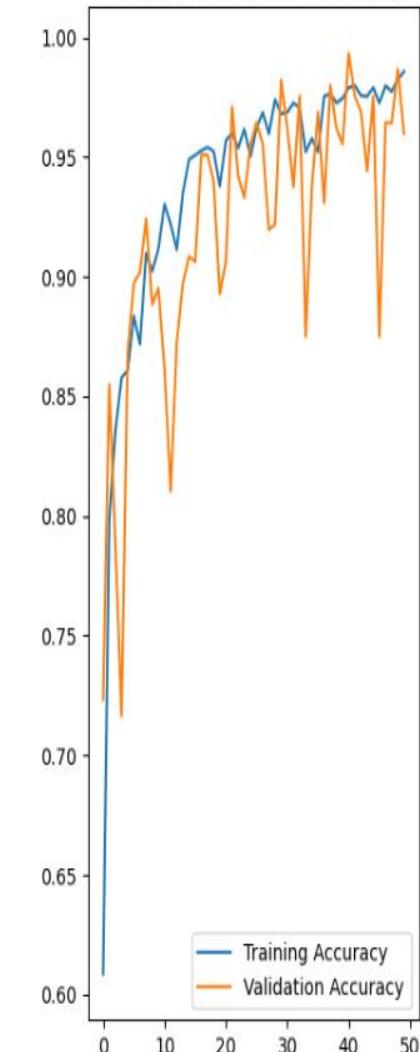
            class_names = ['Tomato_Early_blight', 'Tomato_Late_blight', 'Tomato_healthy']
            predicted_label = class_names[predicted_class_idx]

            os.remove(file_path)
            return jsonify({
                'predicted_class': predicted_label,
```

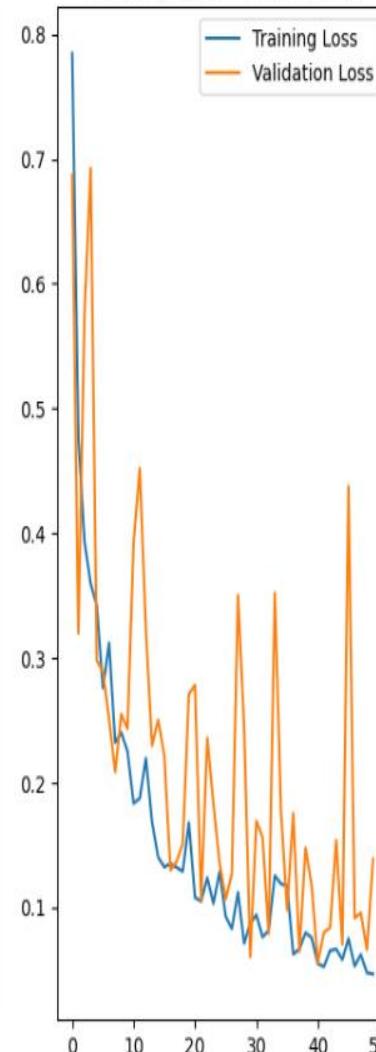


Accuracy Matix

Training and Validation Accuracy



Training and Validation Loss



IT21228162 - Sankeethan. Y

Information Technology

Integrated Weather Forecasting and Plant Growth Monitoring System Based on Soil Moisture Analysis



Introduction

01

Background

02

Research Problem

03

Objectives



Background

This component integrates weather forecasting and soil moisture analysis to monitor plant growth and predict future growth patterns. The goal is to optimize farming practices by aligning them with environmental conditions and crop needs.



Research Problem

Farmers need accurate predictions of plant growth to make informed decisions about fertilization and other interventions.



OBJECTIVES

**Weather
Forecast
Integration**

**Develop a
system for
monitoring
plant growth.**

**Predict
growth
outcomes
based on
customized
fertilization**

Methodology



01

02

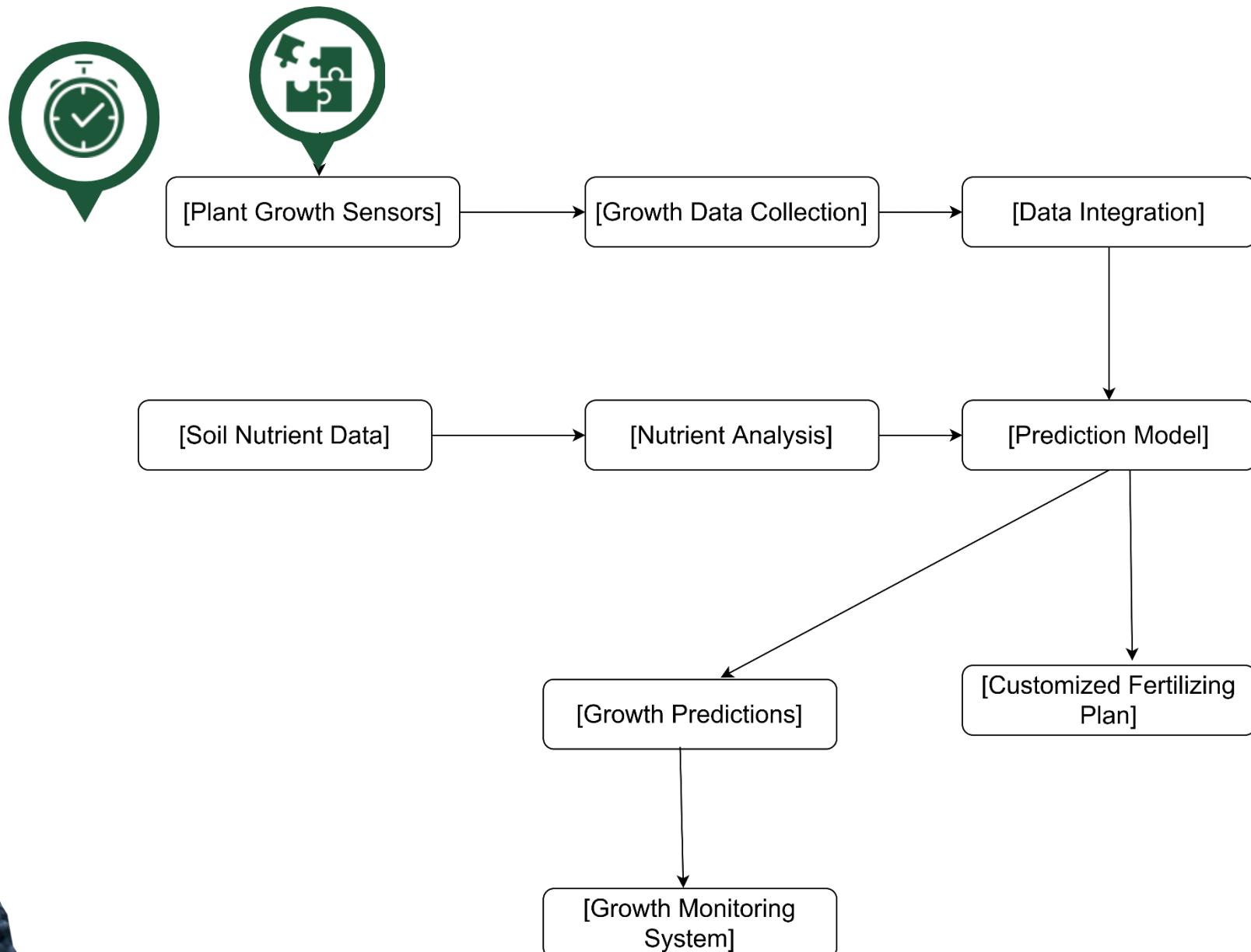
03

System Diagram

Technologies

Requirements

System Diagram



Technologies



- Python



- Flask



- Firebase



- Arduino IDE
- PyCharm

Requirements

Functional

- Growth monitoring software
- Machine learning models for prediction
- Analytical software for growth predictions



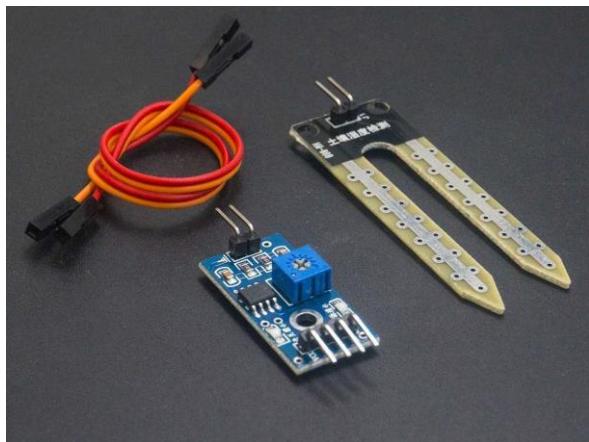
Non Functional

- Accuracy
- Performance
- Availability
- Usability

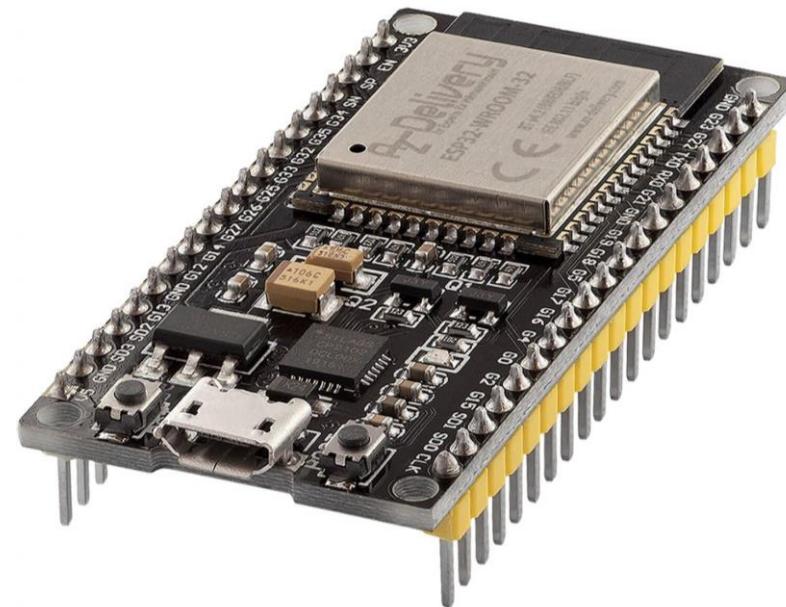


Tools

IOT-Device Circuits Items



Soil Moisture Sensor Module:
measure or estimate the amount
of water in the soil.



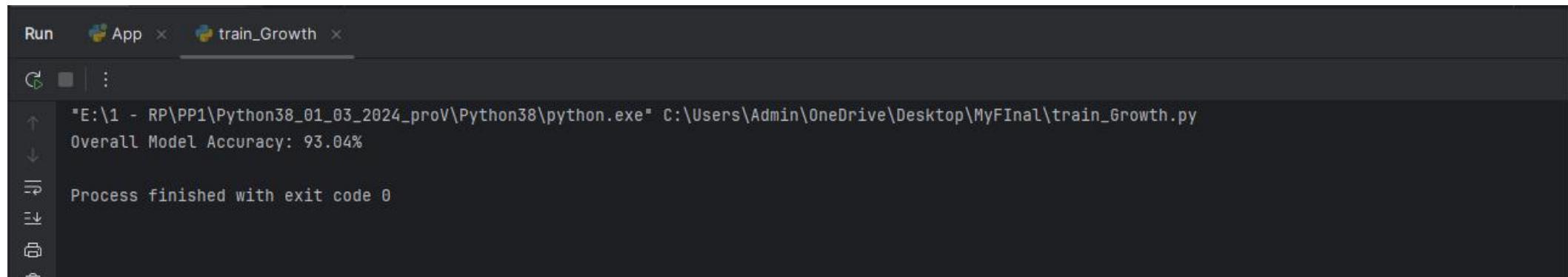
ESP32 Microcontroller:
For data processing and
communication.

Model

```
train_Growth.py x
1 import pandas as pd
2 import numpy as np
3 from sklearn.model_selection import train_test_split
4 from sklearn.ensemble import RandomForestRegressor
5 from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_percentage_error
6 import joblib
7
8 # Load dataset
9 df = pd.read_csv('plant_growth_data.csv')
10
11 # Prepare Features and Target Variables
12 X = df[['Soil Moisture (%)', 'Precipitation (mm)', 'Fertilizer Amount (kg)']]
13
14 # Targets
15 y_height = df['Plant Height (cm)']
16 y_leaf_area = df['Leaf Area (cm²)']
17
18 # Split the Data
19 X_train, X_test, y_height_train, y_height_test = train_test_split(*arrays: X, y_height, test_size=0.3, random_state=42)
20 _, _, y_leaf_area_train, y_leaf_area_test = train_test_split(*arrays: X, y_leaf_area, test_size=0.3, random_state=42)
21
22 # Train Models
23 height_model = RandomForestRegressor(n_estimators=100, random_state=42)
24 height_model.fit(X_train, y_height_train)
25
26 leaf_area_model = RandomForestRegressor(n_estimators=100, random_state=42)
27 leaf_area_model.fit(X_train, y_leaf_area_train)
```

```
train_Growth.py x
26 leaf_area_model = RandomForestRegressor(n_estimators=100, random_state=42)
27 leaf_area_model.fit(X_train, y_leaf_area_train)
28
29 # Evaluate Models
30 y_height_pred = height_model.predict(X_test)
31 height_mape = mean_absolute_percentage_error(y_height_test, y_height_pred) * 100
32 height_accuracy = 100 - height_mape
33
34 y_leaf_area_pred = leaf_area_model.predict(X_test)
35 leaf_area_mape = mean_absolute_percentage_error(y_leaf_area_test, y_leaf_area_pred) * 100
36 leaf_area_accuracy = 100 - leaf_area_mape
37
38 # Compute Overall Accuracy
39 overall_accuracy = (height_accuracy + leaf_area_accuracy) / 2
40
41
42 print(f"Overall Model Accuracy: {overall_accuracy:.2f}%")
43
44 # Save Models
45 joblib.dump(height_model, filename: 'height_model.pkl')
46 joblib.dump(leaf_area_model, filename: 'leaf_area_model.pkl')
```

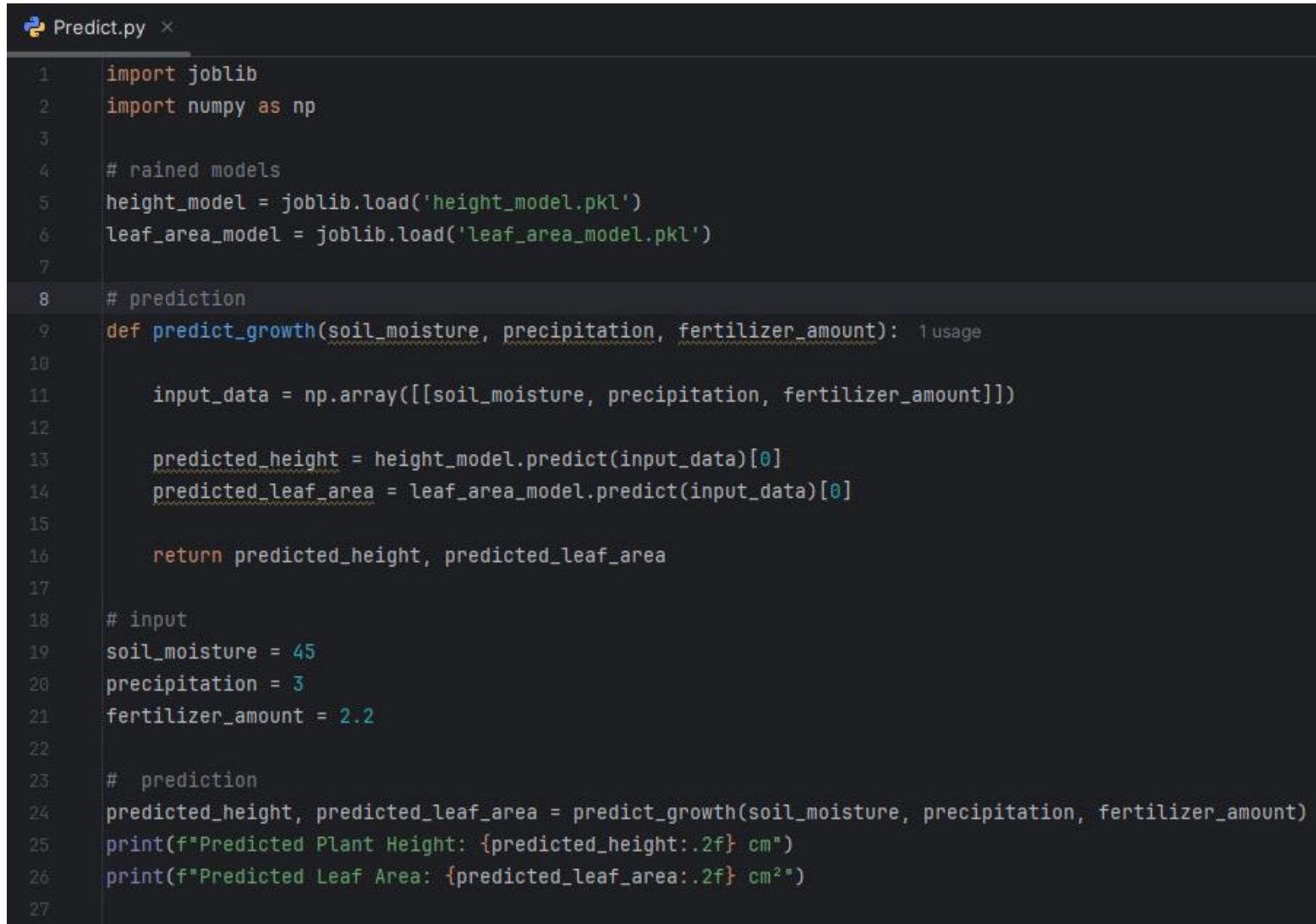
Accuracy



A screenshot of a terminal window titled "Run App train_Growth". The window shows the command "E:\1 - RP\PP1\Python38_01_03_2024_prov\Python38\python.exe" C:\Users\Admin\OneDrive\Desktop\MyFinal\train_Growth.py being run. The output displays "Overall Model Accuracy: 93.04%" followed by "Process finished with exit code 0". The terminal has a dark theme with light-colored text.

```
"E:\1 - RP\PP1\Python38_01_03_2024_prov\Python38\python.exe" C:\Users\Admin\OneDrive\Desktop\MyFinal\train_Growth.py
Overall Model Accuracy: 93.04%
Process finished with exit code 0
```

BackEnd

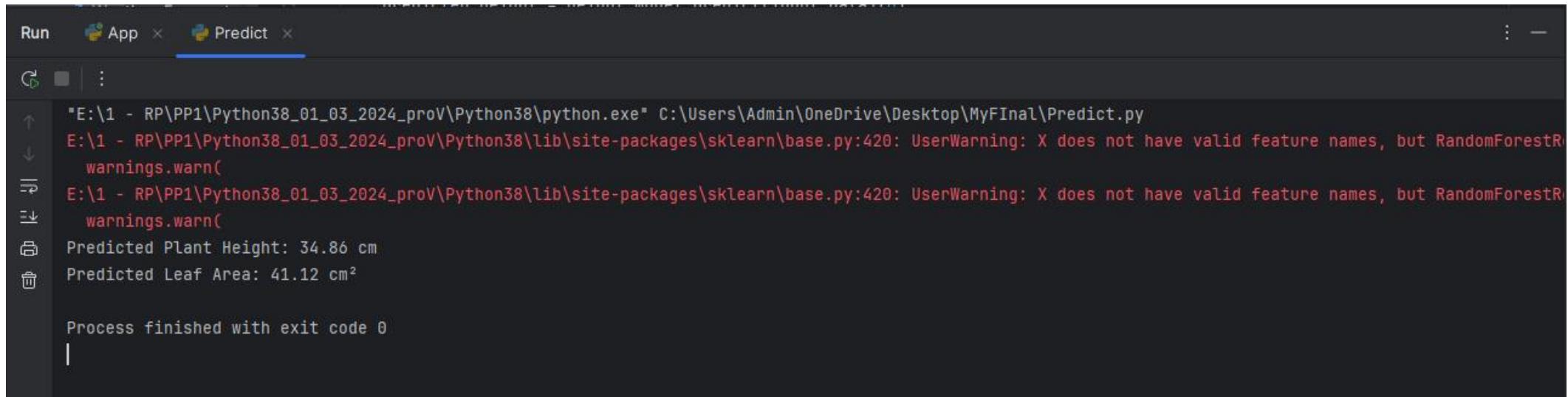


The image shows a screenshot of a code editor with a dark theme. The title bar says "Predict.py x". The code is written in Python and performs the following tasks:

- Imports `joblib` and `numpy`.
- Loads two trained models: `height_model` and `leaf_area_model`.
- Defines a function `predict_growth` that takes `soil_moisture`, `precipitation`, and `fertilizer_amount` as input. It uses these to predict plant height and leaf area.
- Creates input data as a numpy array with one row containing the three parameters.
- Predicts height and leaf area using the loaded models.
- Returns the predicted height and leaf area.
- Defines input variables: `soil_moisture = 45`, `precipitation = 3`, and `fertilizer_amount = 2.2`.
- Calls the `predict_growth` function with these inputs.
- Prints the predicted plant height and leaf area to the console.

```
1 import joblib
2 import numpy as np
3
4 # loaded models
5 height_model = joblib.load('height_model.pkl')
6 leaf_area_model = joblib.load('leaf_area_model.pkl')
7
8 # prediction
9 def predict_growth(soil_moisture, precipitation, fertilizer_amount): 1 usage
10
11     input_data = np.array([[soil_moisture, precipitation, fertilizer_amount]])
12
13     predicted_height = height_model.predict(input_data)[0]
14     predicted_leaf_area = leaf_area_model.predict(input_data)[0]
15
16     return predicted_height, predicted_leaf_area
17
18 # input
19 soil_moisture = 45
20 precipitation = 3
21 fertilizer_amount = 2.2
22
23 # prediction
24 predicted_height, predicted_leaf_area = predict_growth(soil_moisture, precipitation, fertilizer_amount)
25 print(f"Predicted Plant Height: {predicted_height:.2f} cm")
26 print(f"Predicted Leaf Area: {predicted_leaf_area:.2f} cm²")
```

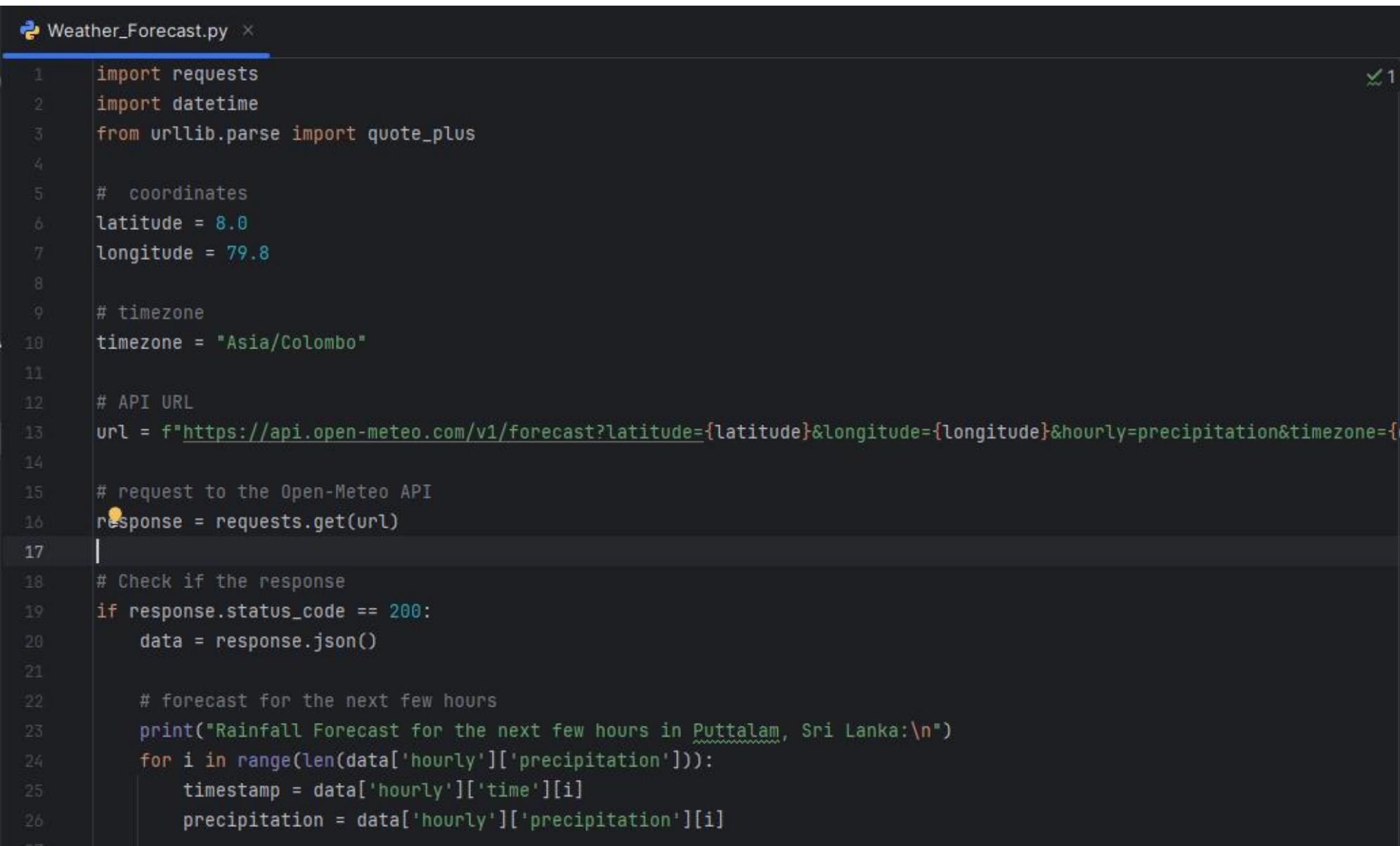
BackEnd



```
"E:\1 - RP\PP1\Python38_01_03_2024_proV\Python38\python.exe" C:\Users\Admin\OneDrive\Desktop\MyFinal\Predict.py
E:\1 - RP\PP1\Python38_01_03_2024_proV\Python38\lib\site-packages\sklearn\base.py:420: UserWarning: X does not have valid feature names, but RandomForestR
  warnings.warn(
E:\1 - RP\PP1\Python38_01_03_2024_proV\Python38\lib\site-packages\sklearn\base.py:420: UserWarning: X does not have valid feature names, but RandomForestR
  warnings.warn(
Predicted Plant Height: 34.86 cm
Predicted Leaf Area: 41.12 cm2

Process finished with exit code 0
```

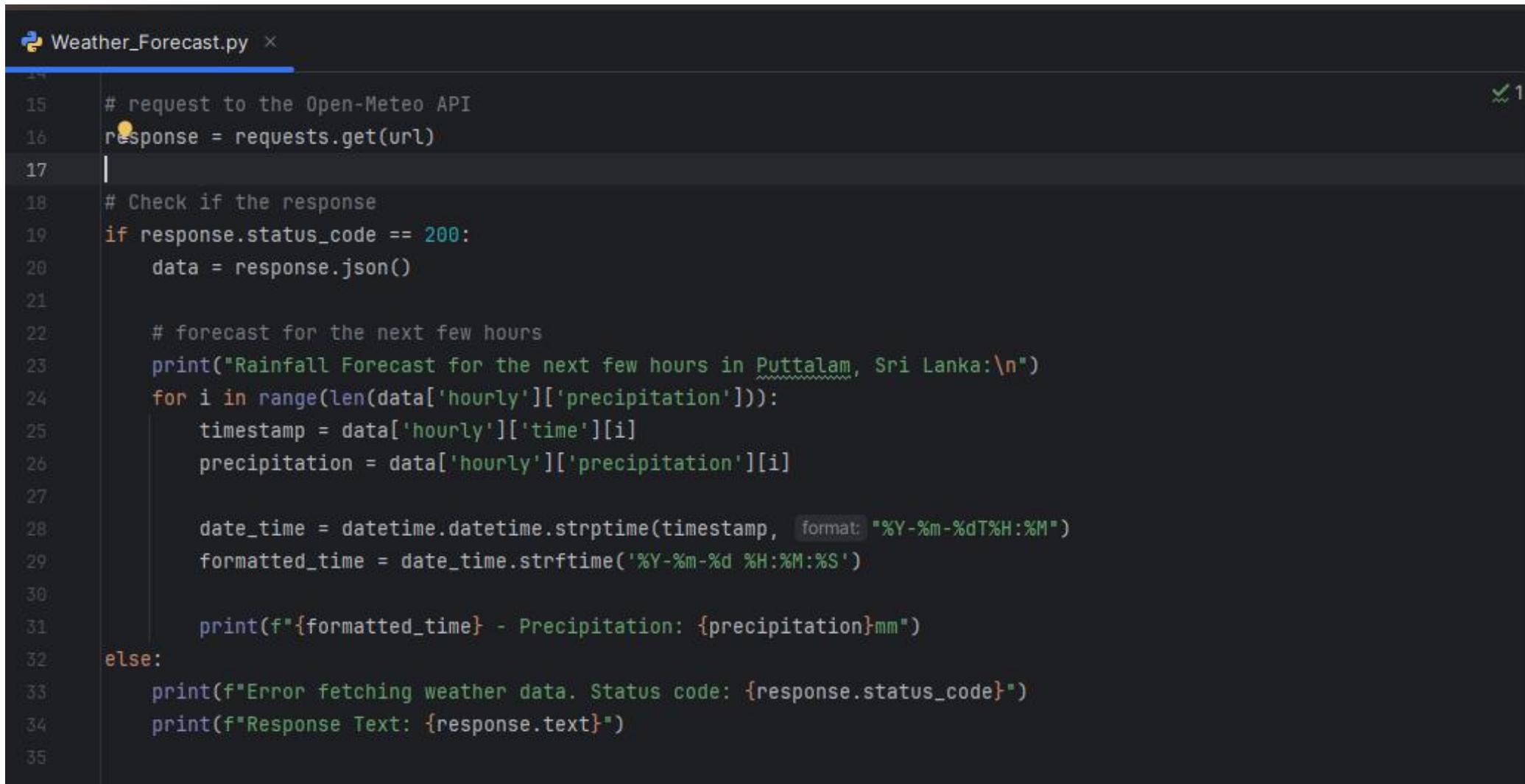
BackEnd



```
Weather_Forecast.py × 1

1 import requests
2 import datetime
3 from urllib.parse import quote_plus
4
5 # coordinates
6 latitude = 8.0
7 longitude = 79.8
8
9 # timezone
10 timezone = "Asia/Colombo"
11
12 # API URL
13 url = f"https://api.open-meteo.com/v1/forecast?latitude={latitude}&longitude={longitude}&hourly=precipitation&timezone={timezone}"
14
15 # request to the Open-Meteo API
16 response = requests.get(url)
17
18 # Check if the response
19 if response.status_code == 200:
20     data = response.json()
21
22     # forecast for the next few hours
23     print("Rainfall Forecast for the next few hours in Puttalam, Sri Lanka:\n")
24     for i in range(len(data['hourly']['precipitation'])):
25         timestamp = data['hourly']['time'][i]
26         precipitation = data['hourly']['precipitation'][i]
```

BackEnd



The screenshot shows a code editor window with a dark theme. The file is named "Weather_Forecast.py". The code itself is as follows:

```
15 # request to the Open-Meteo API
16 response = requests.get(url)
17 |
18 # Check if the response
19 if response.status_code == 200:
20     data = response.json()
21
22     # forecast for the next few hours
23     print("Rainfall Forecast for the next few hours in Puttalam, Sri Lanka:\n")
24     for i in range(len(data['hourly']['precipitation'])):
25         timestamp = data['hourly']['time'][i]
26         precipitation = data['hourly']['precipitation'][i]
27
28         date_time = datetime.datetime.strptime(timestamp, format: "%Y-%m-%dT%H:%M")
29         formatted_time = date_time.strftime('%Y-%m-%d %H:%M:%S')
30
31         print(f"{formatted_time} - Precipitation: {precipitation}mm")
32 else:
33     print(f"Error fetching weather data. Status code: {response.status_code}")
34     print(f"Response Text: {response.text}")
35
```

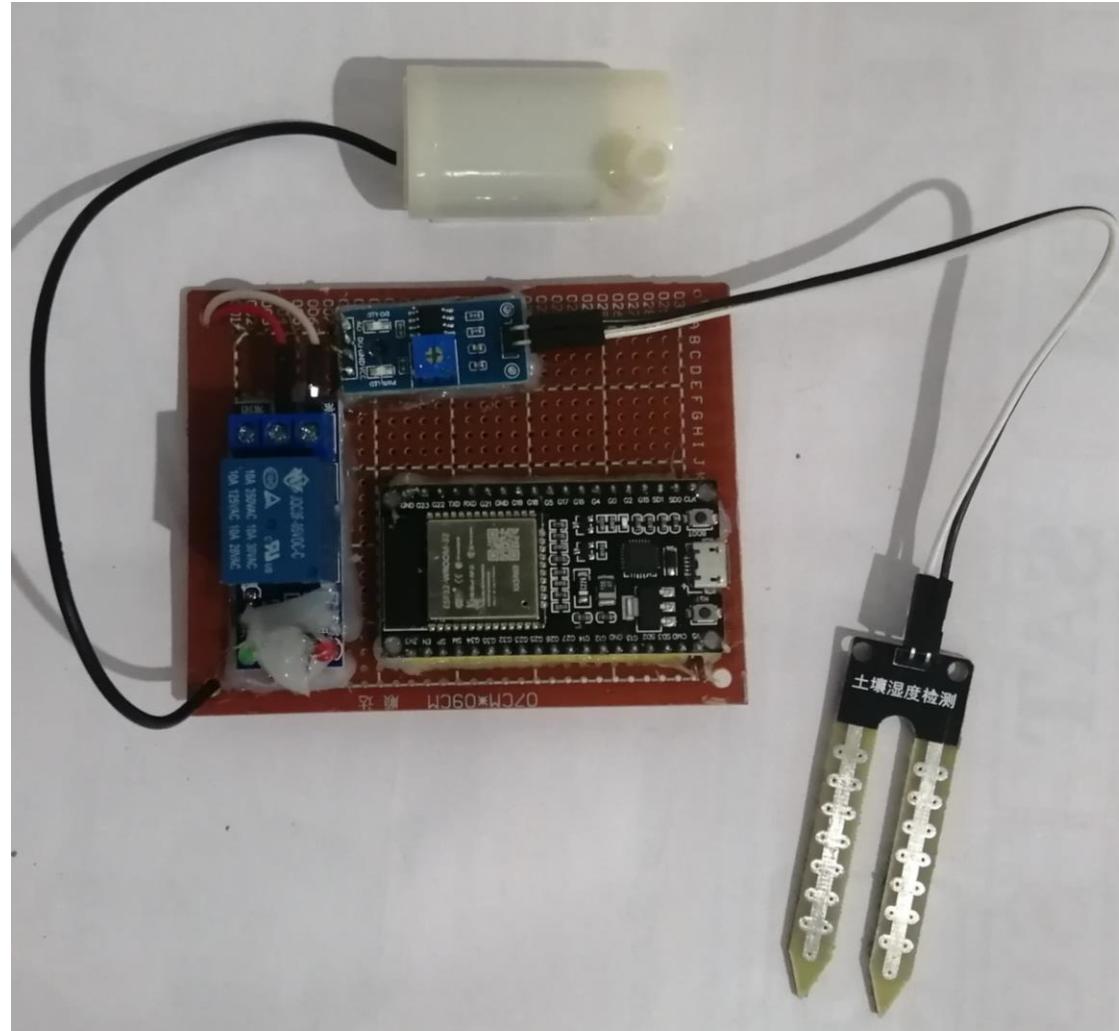
BackEnd

```
Run App Weather_Forecast ×

"E:\1 - RP\PP1\Python38_01_03_2024_proV\Python38\python.exe" C:\Users\Admin\OneDrive\Desktop\MyFinal\Weather_Forecast.py
Rainfall Forecast for the next few hours in Puttalam, Sri Lanka:

2025-03-18 00:00:00 - Precipitation: 0.0mm
2025-03-18 01:00:00 - Precipitation: 0.0mm
2025-03-18 02:00:00 - Precipitation: 0.0mm
2025-03-18 03:00:00 - Precipitation: 0.0mm
2025-03-18 04:00:00 - Precipitation: 0.0mm
2025-03-18 05:00:00 - Precipitation: 0.0mm
2025-03-18 06:00:00 - Precipitation: 0.0mm
2025-03-18 07:00:00 - Precipitation: 0.0mm
2025-03-18 08:00:00 - Precipitation: 0.0mm
2025-03-18 09:00:00 - Precipitation: 0.0mm
2025-03-18 10:00:00 - Precipitation: 0.0mm
2025-03-18 11:00:00 - Precipitation: 0.0mm
2025-03-18 12:00:00 - Precipitation: 0.0mm
2025-03-18 13:00:00 - Precipitation: 0.0mm
2025-03-18 14:00:00 - Precipitation: 0.0mm
```

Prototype



Completed

- Built prototype 80%
- Integrate Tools
- Model Training



To be Complete

- Connect the prototype with Database
- Web UI Design (Frontend)
- Testing



References

[1]<https://ieeexplore.ieee.org/document/9754145>

[2]<https://www.sciencedirect.com/science/article/abs/pii/S2214785321042115>

[3] Pettorelli, N. (2013). *The Normalization of Vegetation Indices*. Springer.

[4] Rouse, J. W., Haas, R. H., Schell, J. A., & Deering, D. W. (1974). *Monitoring Vegetation Systems in the Great Plains with ERTS*. NASA.

Predictive Analytics for Crop Yield, Harvest Readiness, and Fertilizer Optimization



Introduction

01

Background

02

Research Problem

03

Objectives



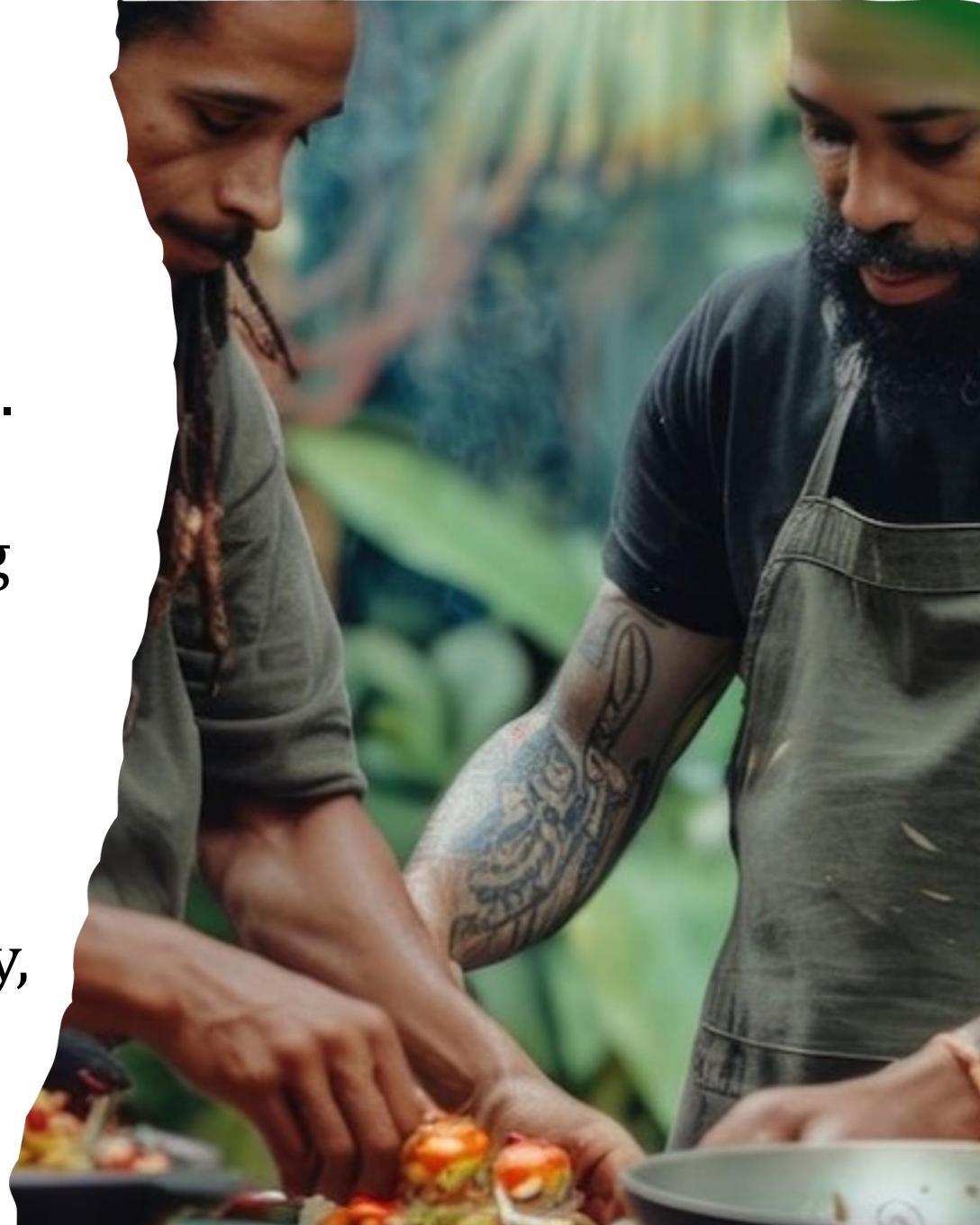
Background

Advances in precision agriculture are transforming traditional farming methods by leveraging IoT-based technologies and data analytics. This research focuses on developing a system that uses real-time data from sensors and agricultural department datasets to predict crop yield, harvest readiness, and fertilizer requirements for optimized resource use and productivity enhancement.



Research Problem

Modern agriculture faces challenges such as unpredictable yields, inefficient resource utilization, and inconsistent harvest readiness. Manual methods of monitoring soil and environmental conditions are time-consuming and prone to errors. Additionally, over- or under-application of fertilizers contributes to soil degradation and economic losses. This research addresses these issues by using IoT-based solutions to improve efficiency, accuracy, and sustainability in farming.



OBJECTIVES

To develop a model for efficient yield prediction, harvest readiness, and fertilizer optimization using sensor data and agricultural datasets.

SUB - OBJECTIVES

- Monitor soil and environmental conditions using sensors.
- Implement predictive models for yield estimation and harvest timing.
- Develop a model to predict the fertilizer plan.
- Create a real-time notification system for farmers regarding harvest readiness.
- Test the system under various environmental conditions to ensure accuracy and reliability.

Methodology

01

Data Collection

02

Data Processing

Clean and preprocess sensor and historical data.

Store data in a centralized system using ESP32 for wireless communication.

03

Model Development

Develop machine learning models for yield prediction, harvest readiness, and fertilizer optimization.

Train models using historical and real-time data.

04

System Implementation

Integrate predictions with a user interface

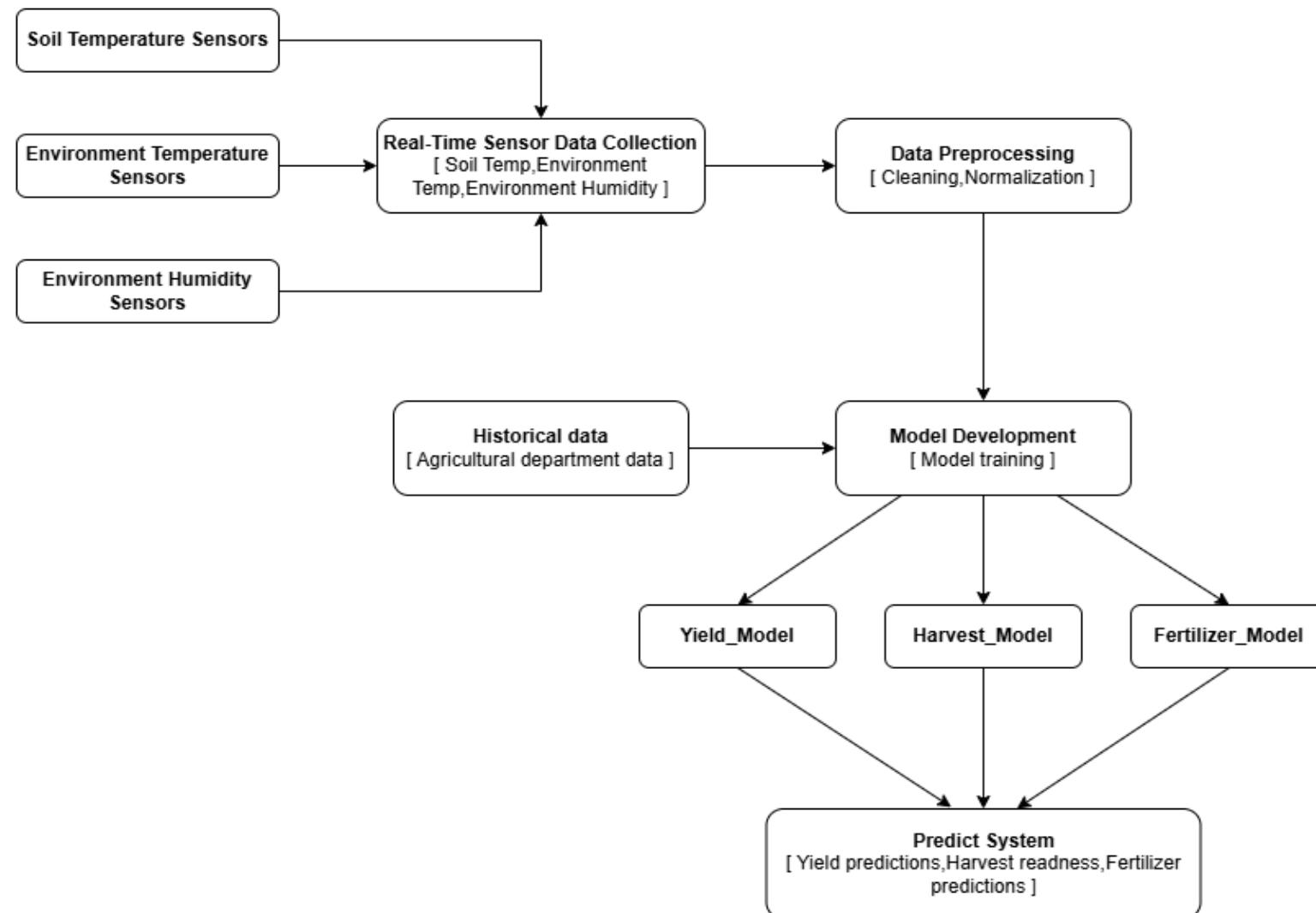
Set up notification systems for alerts and recommendations.

05

Validation and Optimization

Refine models to improve accuracy.

System Diagram



Technologies



- Python



- Flask



- Firebase



- Arduino IDE
- PyCharm

Requirements

Functional

- Monitor soil temperature, environmental temperature, and humidity in real time.
- Analyze collected data to predict crop yield and determine harvest readiness.
- Data processing

Non Functional

- **Accuracy:** The system must provide precise predictions and recommendations.
- **Reliability:** Ensure consistent data collection and processing.
- **Cost-Effectiveness:** Use affordable components and minimize operational costs.
- **User-Friendliness:** Interfaces should be simple for farmers with minimal technical expertise.
- **Security:** Safeguard sensitive data and communications.

Tools

IOT-Device Circuits Items



Soil Temperature Sensor :
DS18B20 or similar



**Environment Temperature sensor
,Environment humidity sensor :**
DHT22/DHT11



ESP32 Microcontroller:
For data processing and communication.

Model

The image shows two side-by-side screenshots of a Python development environment, likely PyCharm, displaying the same file, `TrainModel.py`, at different stages of its development.

Left Screenshot: This screenshot shows the initial state of the script. It imports necessary libraries like pandas, numpy, and various scikit-learn modules. It loads the agricultural data from a CSV file, performs preprocessing by mapping categorical growth stages (Vegetative, Flowering, Fruiting, Ripening, Harvesting) to numerical values (0, 1, 2, 3, 4), and maps harvest readiness (Yes/No) to binary values (1 or 0). It then splits the data into features (`X`) and labels (`y_fertilizer`, `y_yield`, `y_harvest`).

```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, accuracy_score
import joblib

# Load dataset
df = pd.read_csv('agricultural_data.csv')

# Preprocessing
df['Growth Stage'] = df['Growth Stage'].map({
    'Vegetative': 0,
    'Flowering': 1,
    'Fruiting': 2,
    'Ripening': 3,
    'Harvesting': 4
})

df['Harvest Readiness (Yes/No)'] = df['Harvest Readiness (Yes/No)'].map({
    'Yes': 1,
    'No': 0
})

# Split data
X = df[['Soil Temperature (°C)', 'Air Temperature (°C)', 'Humidity (%)', 'Growth Stage']]
y_fertilizer = df['Fertilizer Applied (kg/ha)']
y_yield = df['Predicted Yield (kg/ha)']
y_harvest = df['Harvest Readiness (Yes/No)']

# Normalize
```

Right Screenshot: This screenshot shows the script after adding model training and evaluation code. It trains three separate models: a Random Forest Regressor for fertilizer application, a Random Forest Regressor for yield, and a Random Forest Classifier for harvest readiness. It calculates Mean Squared Error (MSE) for the regression models and accuracy for the classification model. Finally, it saves all trained models and the scaler used for normalization.

```
# Train models
fertilizer_model = RandomForestRegressor(n_estimators=100, random_state=42)
yield_model = RandomForestRegressor(n_estimators=100, random_state=42)
harvest_model = RandomForestClassifier(n_estimators=100, random_state=42)

fertilizer_model.fit(X_train, y_fertilizer_train)
yield_model.fit(X_train, y_yield_train)
harvest_model.fit(X_train, y_harvest_train)

# Evaluate
fertilizer_pred = fertilizer_model.predict(X_test)
yield_pred = yield_model.predict(X_test)
harvest_pred = harvest_model.predict(X_test)

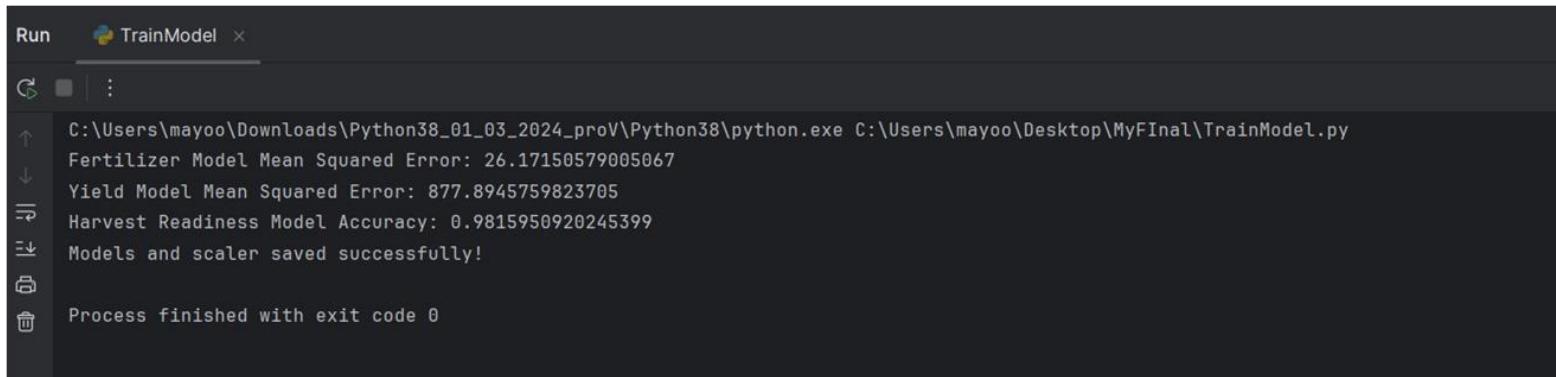
fertilizer_mse = mean_squared_error(y_fertilizer_test, fertilizer_pred)
yield_mse = mean_squared_error(y_yield_test, yield_pred)
harvest_accuracy = accuracy_score(y_harvest_test, harvest_pred)

print(f'Fertilizer Model Mean Squared Error: {fertilizer_mse}')
print(f'Yield Model Mean Squared Error: {yield_mse}')
print(f'Harvest Readiness Model Accuracy: {harvest_accuracy}')

# Save models
joblib.dump(fertilizer_model, filename='fertilizer_model.pkl')
joblib.dump(yield_model, filename='yield_model.pkl')
joblib.dump(harvest_model, filename='harvest_model.pkl')
joblib.dump(scaler, filename='scaler.pkl')

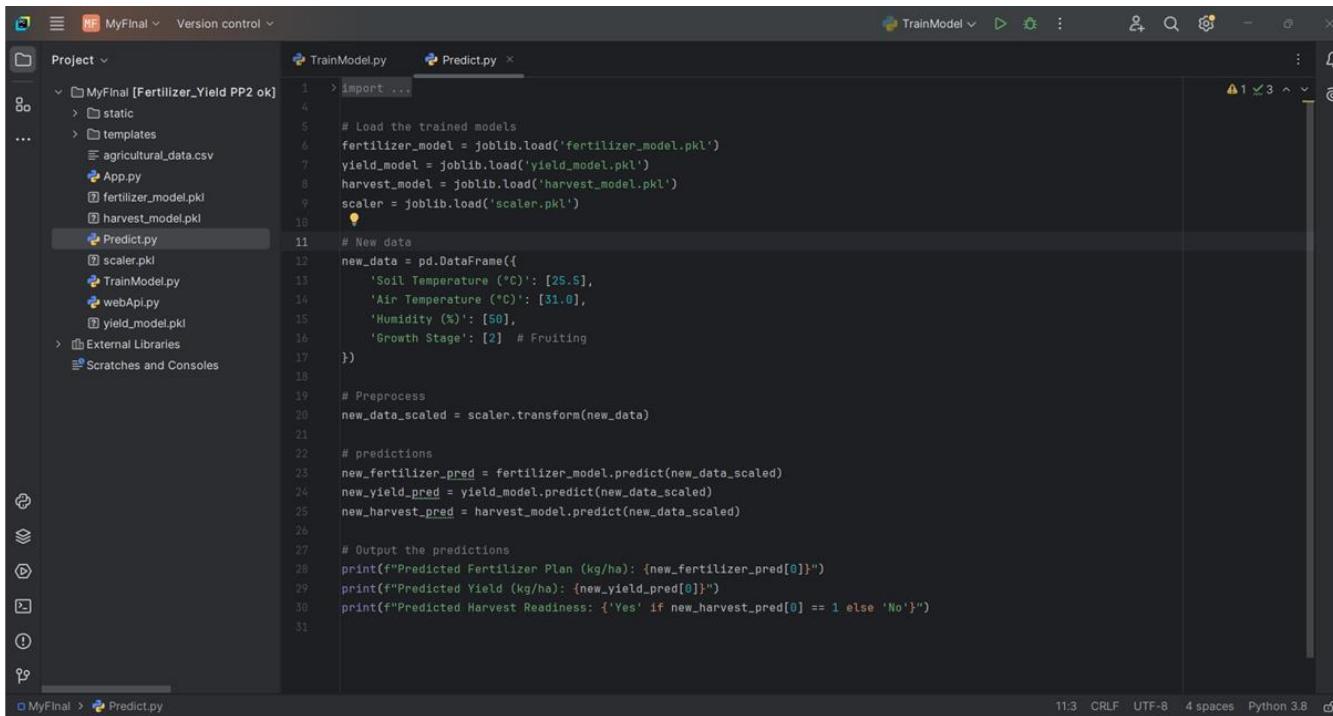
print("Models and scaler saved successfully!")
```

Accuracy



```
Run TrainModel ×
C:\Users\mayoo\Downloads\Python38_01_03_2024_proV\Python38\python.exe C:\Users\mayoo\Desktop\MyFinal\TrainModel.py
Fertilizer Model Mean Squared Error: 26.17150579005067
Yield Model Mean Squared Error: 877.8945759823705
Harvest Readiness Model Accuracy: 0.9815950920245399
Models and scaler saved successfully!
Process finished with exit code 0
```

Backend



The screenshot shows a Python development environment with the following details:

- Project Structure:** The project is named "MyFinal [Fertilizer_Yield PP2 ok]". It contains a "static" folder, a "templates" folder with "agricultural_data.csv", an "App.py" file, and several model files: "fertilizer_model.pkl", "harvest_model.pkl", "scaler.pkl", "TrainModel.py", and "yield_model.pkl".
- Code Editor:** The "Predict.py" script is open. The code reads three trained models ("fertilizer_model.pkl", "yield_model.pkl", and "harvest_model.pkl") and a scaler ("scaler.pkl"). It creates a new data frame with specific input values (Soil Temperature, Air Temperature, Humidity, Growth Stage) and scales them using the loaded scaler. Then, it makes predictions using each model and prints the results.
- Terminal:** The terminal output shows the execution of the script and its results. The output is:

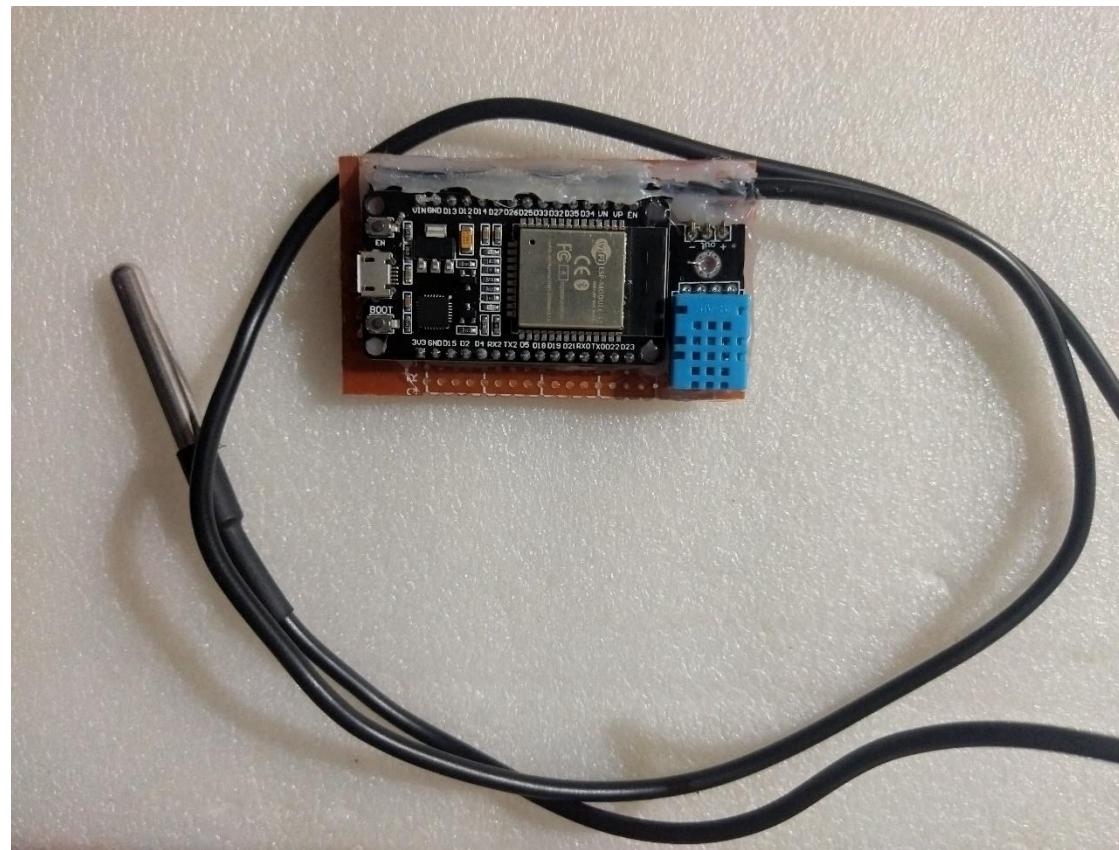
```
C:\Users\mayoo\Downloads\Python38_01_03_2024_proV\Python38\python.exe C:\Users\mayoo\Desktop\MyFinal\Predict.py
Predicted Fertilizer Plan (kg): 103.7718076828001
Predicted Yield (kg): 848.9043078939992
Predicted Harvest Readiness: Yes

Process finished with exit code 0
```

```
C:\Users\mayoo\Downloads\Python38_01_03_2024_proV\Python38\python.exe C:\Users\mayoo\Desktop\MyFinal\Predict.py
Predicted Fertilizer Plan (kg): 103.7718076828001
Predicted Yield (kg): 848.9043078939992
Predicted Harvest Readiness: Yes

Process finished with exit code 0
```

Prototype



Prototype Code

The image shows two side-by-side screenshots of the Arduino IDE 2.3.3 interface. Both windows have the title bar "sketch_dec3a | Arduino IDE 2.3.3" and the tab "ESP32 Dev Module".

Left Window (Board Manager):

- Boards Manager:** Shows the "ARDUINO" section with "Arduino AVR Boards by Arduino" (1.8.6 installed) and "Arduino ESP32 Boards by Arduino" (2.0.18-arduino.5 installed). It also lists "Arduino Mbed OS Edge Boards by Arduino" (4.2.4).
- Sketch:** Displays the following C++ code for "sketch_dec3a.ino":

```
1 #include <WiFi.h>
2 #include <WiFiClientSecure.h>
3 #include <firebaseESP32.h>
4
5 // Replace with your own credentials
6 #define WIFI_SSID "T"
7 #define WIFI_PASSWORD "123456789"
8 #define API_KEY "AizaSyASS_pFUqXhA8LmRdb1_Nh-Uqu0L4cac"
9 #define DATABASE_URL "https://mysensor-bbe50-default-rtdb.firebaseio.com/" // Replace with your Firebase database URL
10
11 // Set up Firebase
12 FirebaseDatabase fbData;
13 FirebaseConfig firebaseConfig;
14 FirebaseAuth firebaseAuth;
15
16 WiFiClientSecure client;
17
18 // Sensor readings (dummy values for illustration)
19 float soilTemperature = 25.6;
20 float envTemperature = 22.3;
21 float humidity = 45.2;
22
23 // Initialize WiFi and Firebase
24 void setup() {
25     Serial.begin(115200);
26
27     // Connect to WiFi
28     WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
29     while (WiFi.status() != WL_CONNECTED) {
30         delay(1000);
31         Serial.println("Connecting to WiFi...");
32     }
33     Serial.println("Connected to WiFi");
```

Right Window (Board Manager):

- Boards Manager:** Shows the "ARDUINO" section with "Arduino AVR Boards by Arduino" (1.8.6 installed) and "Arduino ESP32 Boards by Arduino" (2.0.18-arduino.5 installed). It also lists "Arduino Mbed OS Edge Boards by Arduino" (4.2.4).
- Sketch:** Displays the following C++ code for "sketch_dec3a.ino":

```
35 // Set up Firebase config
36 firebaseConfig.database_url = DATABASE_URL;
37 firebaseConfig.api_key = API_KEY;
38 firebase.begin(&firebaseConfig, &firebaseAuth);
39 }
40
41 void loop() {
42     // Set data in Firebase using Firebase methods
43     if (Firebase.RTDB.setFloat(&fbData, "/sensor_data/soil_temperature", soilTemperature)) {
44         Serial.println("Soil Temperature updated successfully!");
45     } else {
46         Serial.println("Failed to update Soil Temperature");
47         Serial.println(fbData.errorReason());
48     }
49
50     if (Firebase.RTDB.setFloat(&fbData, "/sensor_data/env_temperature", envTemperature)) {
51         Serial.println("Environmental Temperature updated successfully!");
52     } else {
53         Serial.println("Failed to update Environmental Temperature");
54         Serial.println(fbData.errorReason());
55     }
56
57     if (Firebase.RTDB.setFloat(&fbData, "/sensor_data/humidity", humidity)) {
58         Serial.println("Humidity updated successfully!");
59     } else {
60         Serial.println("Failed to update Humidity");
61         Serial.println(fbData.errorReason());
62     }
63
64     // Delay before the next loop
65     delay(2000); // Adjust the delay time as needed
66 }
```

Prototype Result

```
Output Serial Monitor ×  
Message (Enter to send message to 'ESP32 Dev Module' on 'COM7')  
  
Soil Temperature: 31.44 °C  
Environment Temperature: 31.60 °C  
Environment Humidity: 72.80 %  
Soil Temperature: 31.44 °C  
Environment Temperature: 31.60 °C  
Environment Humidity: 72.80 %  
Soil Temperature: 31.44 °C  
Environment Temperature: 31.60 °C  
Environment Humidity: 72.80 %  
Soil Temperature: 31.44 °C  
Environment Temperature: 31.60 °C  
Environment Humidity: 72.80 %  
Soil Temperature: 31.44 °C  
Environment Temperature: 31.60 °C  
Environment Humidity: 72.80 %  
Soil Temperature: 31.50 °C  
Environment Temperature: 31.60 °C  
Environment Humidity: 72.80 %
```

Completed

- Built prototype 80%
- Integrate Sensors & Tools
- Model Development
- Backend



To be Complete

- Connect the prototype with Database
- Web UI Design (Frontend)
- Testing



References

[1]

<https://ieeexplore.ieee.org/document/9754145>

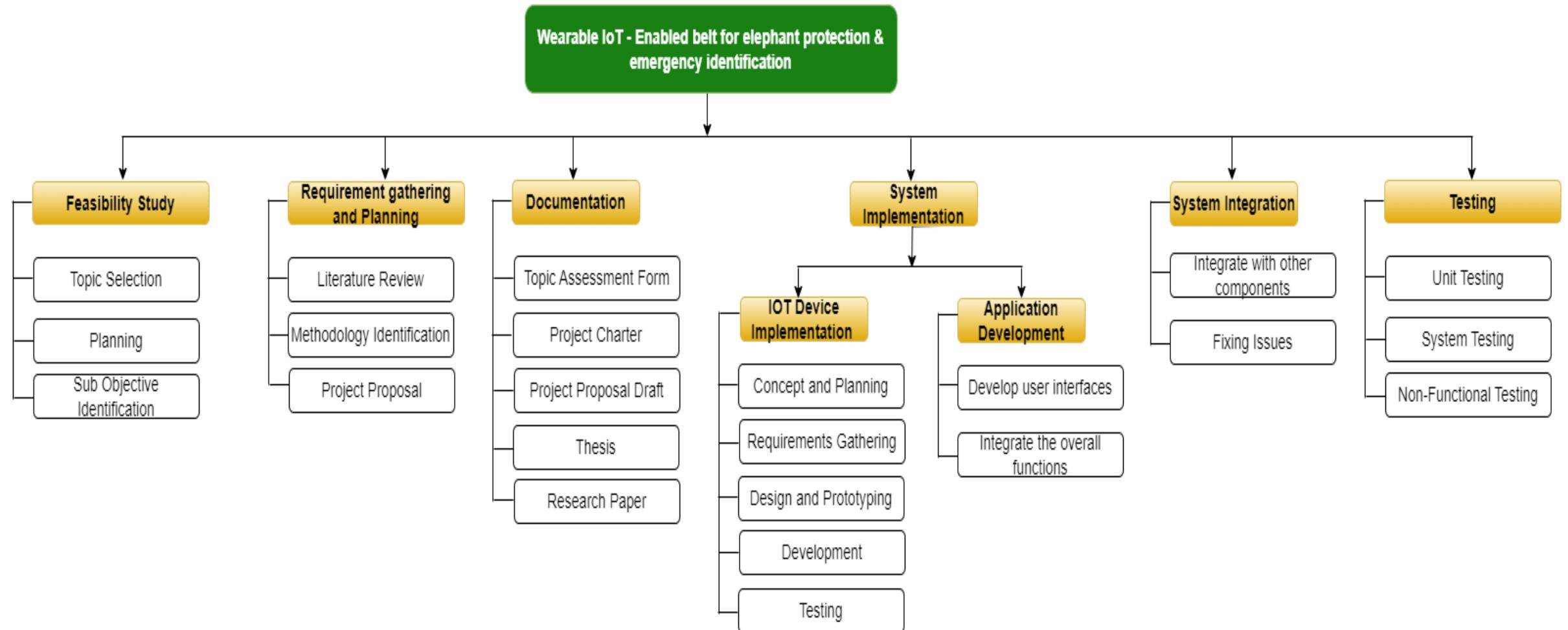
[2]

<https://www.sciencedirect.com/science/article/abs/pii/S2214785321042115>

[3] Tsafaris, S. A., & Blomberg, S. (2018). *Machine Learning for Crop Yield Prediction*. Springer.

[4] Yang, X., & Li, H. (2018). *Remote Sensing for Precision Agriculture: Yield Prediction and Harvest Management*. Wiley.

Work Breakdown Structure



Cost Management Plan

Budget

Description	Cost Rs
IOT – Device Budget	40,000
Other Expenses Budget	25,000
Total	65,000

Project Management Plan

The screenshot shows a project management interface with three main columns: Todo, In Progress, and Done.

- Todo:** Contains 4 items.
 - 2024-25J-164 #1: Development Environment Setup for Automated Agriculture System: Accurate Water Management and Disease Analysis
 - 2024-25J-164 #4: Development Environment setup for the disease Analysis
 - 2024-25J-164 #5: Software Designing For Disease Detection
 - 2024-25J-164 #6: Software Designing For water management technology
- In Progress:** Contains 1 item.
 - 2024-25J-164 #2: Create & Maintain the Meeting Log Book
- Done:** Contains 1 item.
 - 2024-25J-164 #3: Project Proposal Presentation

Each card includes a summary status, estimate (0), and a detailed description. There are also buttons for adding items and discarding changes.





Thanks