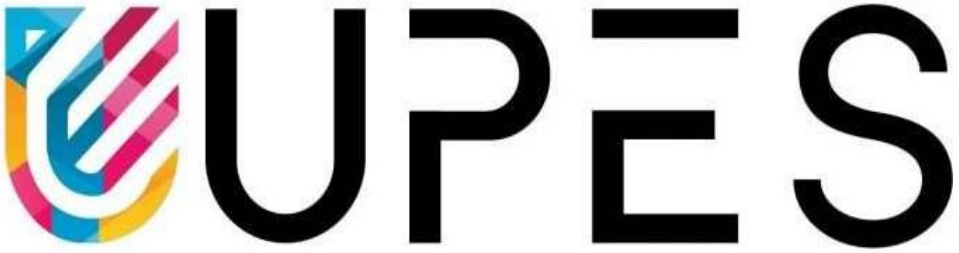


AGROLTYTICS
SMART AGRICULTIRAL INSIGHTS

Intelligent Machine Learning Based Crop Recommendation and Yield prediction system

Submitted in partial fulfillment of the requirements for the degree of
B.Tech in Computer Science and Engineering
for the course **"Fundamentals Of Data Science"**

By
Somya Vats (500119012)
Tanisha (500125283)

Under the Guidance of
Dr. Neeraj Chugh
Supervisor
Associate Professor, School of Computer Science

January-May 2025

AGROLYTICS

Intelligent Machine Learning Based Crop Recommendation and Yield prediction system

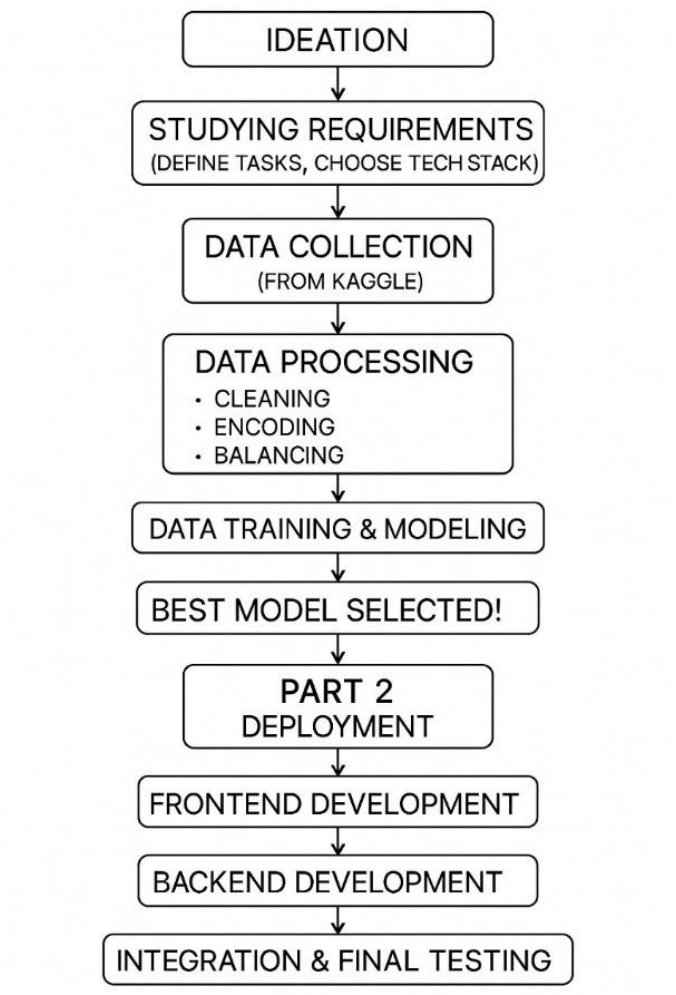
PROJECT HELP FILE

Agriculture remains vital to global food security but faces mounting challenges such as climate change, resource limitations, and population growth. To address these, this project presents an Agricultural Decision Support System comprising two components: a Crop Recommendation System and a Crop Yield Prediction System. The recommendation system analyzes environmental factors—like soil nutrients, temperature, humidity, and rainfall—using machine learning to suggest optimal crops for a given region. Meanwhile, the yield prediction system forecasts crop productivity based on historical and climatic data, aiding better planning and resource use.
Using algorithms such as Random Forest, Extra Trees Regressor, and XGBoost on a rich dataset, the system delivers actionable insights for farmers. Overall, this tool supports smarter, more sustainable farming and contributes to SDG 2: Zero Hunger.

Project Research Report: Agrolytics

Introduction to the Manual

This help manual provides a comprehensive overview of the design, development, and implementation of the **'Crop Recommendation and Yield Prediction System'**. It is intended to guide users and developers through the project's objectives, research background, technical choices, and development workflow. Aligned with Sustainable Development Goal 2 (Zero Hunger), the manual highlights the problem we aim to solve—supporting farmers with accurate crop recommendations and yield predictions based on data-driven insights. It documents how we gathered data, took guidance from mentors, and selected relevant tools and models using platforms like Kaggle and Google Colab.
Additionally, the manual covers the complete technical stack, including data preprocessing, model building, and deployment using Flask, along with a clear explanation of backend structure and saved model files. It serves as a reference for understanding the system's architecture and for anyone looking to maintain, replicate, or extend the project.
Below mentioned is the project workflow:



Ideation Phase

Problem Statement
India, being an agrarian economy, faces challenges related to soil fertility, improper crop selection, and unpredictable yield. To address these challenges, we have designed a dual-module system that:

- Recommends the most suitable crop based on environmental and soil conditions.
- Predicts the yield of the selected crop.

Research and Guidance
Under the guidance of our mentor, we explored publicly available datasets from Kaggle and government repositories. After careful evaluation, we chose datasets relevant to crop suitability and yield estimation. Our project combines practical field knowledge with analytical tools, making it a data-driven decision support system.

Technology Stack

Component	Tools/Technologies Used
Programming	Python, R
Data Analysis	Exploratory Data Analysis (EDA), Pandas, NumPy
Visualization	Matplotlib, Seaborn
Model Building	Scikit-learn, XGBoost
Data Platform	Google Colab
Backend	Python Flask
Frontend	HTML, CSS
Model Deployment	Joblib (for model saving/loading)

Data Preparation

The datasets used for this project were collected from Kaggle:

- **Crop Recommendation Dataset**: Link
- **Crop Yield Prediction Dataset**: Link

The datasets consist of multiple agricultural parameters like soil nutrients, weather conditions, and historical yield data, which were essential for building the crop recommendation and yield prediction models. These data points are instrumental in training machine learning algorithms to suggest suitable crops and predict their yield based on environmental factors.

To process and analyze the data, various data cleaning, transformation, and exploration

techniques were employed. We used **Pandas** for data manipulation and **NumPy** for numerical calculations, following best practices from the data science community (referencing materials like Scikit-learn's Supervised Learning Documentation for model implementation). Additionally, data visualization tools such as **Matplotlib** and **Seaborn** were utilized for better understanding and presentation of the relationships between variables.
For the preparation of the models, we also referred to websites like **GeeksforGeeks** for algorithmic insights and tips on improving model performance.

Part A: Crop Recommendation System

The full code is at google collab: crop recommendation

1. Data Collection

For this crop recommendation module, we utilized publicly available datasets from Kaggle, commonly used in agricultural analytics. These datasets include key agro-environmental features crucial for predicting crop viability:

- Soil Nutrient Levels: Nitrogen (N), Phosphorus (P), Potassium (K)
- Climatic Factors: Temperature (°C), Humidity (%)
- Soil Chemistry: pH value
- Rainfall: Measured in mm

These factors are selected because they directly influence crop productivity. By using this data, the model empowers farmers with data-driven insights, helping them optimize crop selection and enhance agricultural sustainability.

```
print("The shape of our data is: ", recommendation_data.shape)
print()
recommendation_data.head()

print("Info about our data= ", recommendation_data.info())
```

```
The shape of our data is:  (2200, 8)

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2200 entries, 0 to 2199
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Nitrogen     2200 non-null   int64
 1   Phosphorus   2200 non-null   int64
 2   Potassium    2200 non-null   int64
 3   Temperature  2200 non-null   float64
 4   Humidity     2200 non-null   float64
 5   pH_Value     2200 non-null   float64
 6   Rainfall     2200 non-null   float64
 7   Crop         2200 non-null   object
dtypes: float64(4), int64(3), object(1)
memory usage: 137.6+ KB
Info about our data=  None
```

```
recommendation_data.describe()
recommendation_data.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2200 entries, 0 to 2199
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Nitrogen     2200 non-null   int64
 1   Phosphorus   2200 non-null   int64
 2   Potassium    2200 non-null   int64
 3   Temperature  2200 non-null   float64
 4   Humidity     2200 non-null   float64
 5   pH_Value     2200 non-null   float64
 6   Rainfall     2200 non-null   float64
 7   Crop         2200 non-null   object
dtypes: float64(4), int64(3), object(1)
memory usage: 137.6+ KB
```

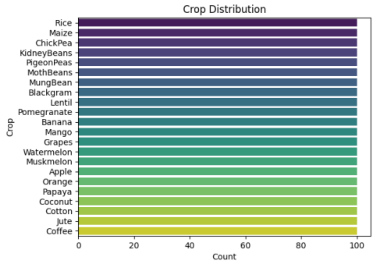
2. Data Description and Preprocessing

The dataset consists of both numerical features and a categorical target variable representing crop types. The preprocessing steps we followed include:

- Label Encoding: We converted the categorical crop names into numerical values using LabelEncoder. This transformation makes the data compatible with machine learning algorithms.
- Exploratory Data Analysis (EDA): We created visualizations (such as pair plots, box plots, and correlation heatmaps) to analyze the relationships between features and identify any anomalies or outliers.
- Data Balancing: The dataset was well-balanced across crop categories, so no additional balancing techniques like SMOTE (Synthetic Minority Oversampling Technique) were applied, preserving the natural distribution of crop types.

```
sns.countplot(y=recommendation_data["Crop"], order=recommendation_data["Crop"].value_counts().index, palette="viridis") # Changed "label" to "Crop"
plt.title("Crop Distribution")
plt.xlabel("Count")
plt.ylabel("Crop")
plt.show()
```

```
<ipython-input-28-c0220e9b0faa>:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

  sns.countplot(y=recommendation_data["Crop"], order=recommendation_data["Crop"].value_counts().index, palette="viridis") # Changed "label" to "Crop"
```




```
print("\n Unique Crop Values Before Encoding:\n", recommendation_data["Crop"].unique())
```

```
 Unique Crop Values Before Encoding:
['Rice' 'Maize' 'ChickPea' 'KidneyBeans' 'PigeonPeas' 'MothBeans'
'MungBean' 'Blackgram' 'Lentil' 'Pomegranate' 'Banana' 'Mango' 'Grapes'
'Watermelon' 'Muskmelon' 'Apple' 'Orange' 'Papaya' 'Coconut' 'Cotton'
'Jute' 'Coffee']
```

```
label_encoder = LabelEncoder()
recommendation_data ["Crop"] = label_encoder.fit_transform(recommendation_data ["Crop"])

print("\n Unique Crop Values After Encoding:\n", recommendation_data ["Crop"].unique())
```

```
 Unique Crop Values After Encoding:
[20 11  3  9 18 13 14  2 10 19  1 12  7 21 15  0 16 17  4  6  8  5]
```

3. Model Training and Evaluation

To identify the most accurate and robust model for crop prediction, multiple algorithms were tested and compared. Classification models were evaluated using metrics like accuracy, precision, recall, and F1-score, while regression models were assessed via the R^2 score.

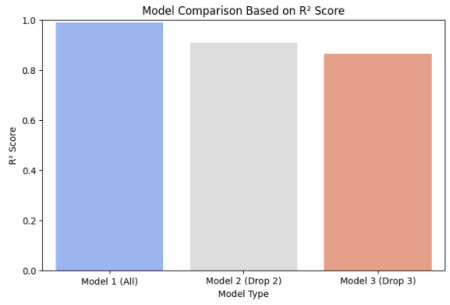
After thorough benchmarking, the **Extra Trees model** stood out across the board — hitting **over 97% F1-score** and the **highest R^2 value**. Its strength lies in its ability to handle complex patterns with great precision, making it the final choice for both classification and regression tasks.

For deeper insight into the model's decisions, **Explainable AI (XAI)** techniques were used:

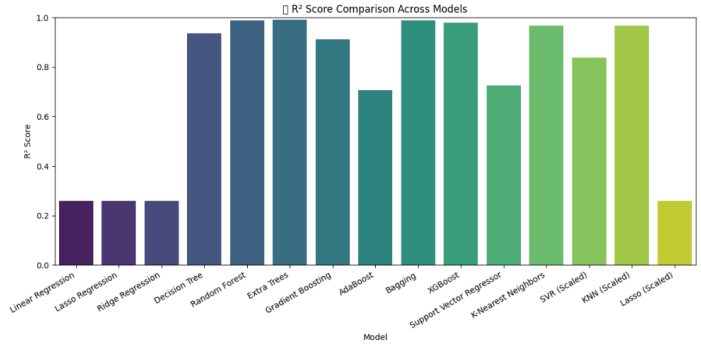
- **Feature Importance Plots** (via Random Forests) clearly highlighted the impact of critical variables such as Nitrogen, Phosphorus, Potassium, and Rainfall on crop suitability.

This blend of performance and interpretability made the Extra Trees model ideal for our application.

```
Model Performance Metrics:
+---------------------------+----------+--------+--------+
|          Model            | R2 Score |  MAE   |  MSE   |
+---------------------------+----------+--------+--------+
|  Model 1 (All Features)   |  0.9881  | 0.2003 | 0.4709 |
| Model 2 (Drop 2 Features) |  0.9078  | 0.6227 | 3.6472 |
| Model 3 (Drop 3 Features) |  0.864   | 0.8628 | 5.3793 |
+---------------------------+----------+--------+--------+
<ipython-input-11-2332376d90eb>:49: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue`

  sns.barplot(x=model_names, y=r2_scores, palette="coolwarm")
```




```
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128200 (\N{CHART WITH UPWARDS TREND}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```



Part B: Crop Yield Prediction System

The full code is at google collab: crop yield

1. Data Collection

For the crop yield prediction module, we utilized a Kaggle dataset containing historical agricultural data across various Indian states and crop types. The dataset included key attributes such as:

- **Crop Name**: Type of crop cultivated (e.g., Rice, Wheat, Maize)
- **State**: Geographic region of cultivation
- **Season**: Agricultural season (Kharif, Rabi, etc.)

- **Area**: Land area used for cultivation (in hectares)
- **Production**: Total crop output (in tonnes)

These features play a direct role in determining agricultural productivity and were foundational to building a model that estimates yield efficiently.

```
yield_data.describe()
```

	Unnamed: 0	Year	hg/ha_yield	average_rain_fall_mm_per_year	pesticides_tonnes	avg_temp
count	28242.000000	28242.000000	28242.000000	28242.00000	28242.000000	28242.000000
mean	14120.500000	2001.544296	77053.332094	1149.05598	37076.909344	20.542627
std	8152.907488	7.051905	84956.612897	709.81215	59958.784665	6.312051
min	0.000000	1990.000000	50.000000	51.00000	0.040000	1.300000
25%	7060.250000	1995.000000	19919.250000	593.00000	1702.000000	16.702500
50%	14120.500000	2001.000000	38295.000000	1083.00000	17529.440000	21.510000
75%	21180.750000	2008.000000	104676.750000	1668.00000	48687.880000	26.000000
max	28241.000000	2013.000000	501412.000000	3240.00000	367778.000000	30.650000

```
yield_data.shape
```

```
(28242, 8)
```

```
yield_data.dtypes
```

	0
Unnamed: 0	int64
Area	object
Item	object
Year	int64
hg/ha_yield	int64
average_rain_fall_mm_per_year	float64
pesticides_tonnes	float64
avg_temp	float64

dtype: object

2. Data Preprocessing

To prepare the dataset for machine learning, the following preprocessing steps were applied:

- **Label Encoding**: All categorical variables, including State, District, Crop Name, and Season, were converted into numerical format using label encoding to ensure compatibility with regression models.

```
print("\n Unique Area Values Before Encoding:\n", yield_data["Area"].unique())
```

```
Unique Area Values Before Encoding:
['Albania' 'Algeria' 'Angola' 'Argentina' 'Armenia' 'Australia' 'Austria'
'Azerbaijan' 'Bahamas' 'Bahrain' 'Bangladesh' 'Belarus' 'Belgium'
'Botswana' 'Brazil' 'Bulgaria' 'Burkina Faso' 'Burundi' 'Cameroon'
'Canada' 'Central African Republic' 'Chile' 'Colombia' 'Croatia'
'Denmark' 'Dominican Republic' 'Ecuador' 'Egypt' 'El Salvador' 'Eritrea'
'Estonia' 'Finland' 'France' 'Germany' 'Ghana' 'Greece' 'Guatemala'
'Guinea' 'Guyana' 'Haiti' 'Honduras' 'Hungary' 'India' 'Indonesia' 'Iraq'
'Ireland' 'Italy' 'Jamaica' 'Japan' 'Kazakhstan' 'Kenya' 'Latvia'
'Lebanon' 'Lesotho' 'Libya' 'Lithuania' 'Madagascar' 'Malawi' 'Malaysia'
'Mali' 'Mauritania' 'Mauritius' 'Mexico' 'Montenegro' 'Morocco'
'Mozambique' 'Namibia' 'Nepal' 'Netherlands' 'New Zealand' 'Nicaragua'
'Niger' 'Norway' 'Pakistan' 'Papua New Guinea' 'Peru' 'Poland' 'Portugal'
'Qatar' 'Romania' 'Rwanda' 'Saudi Arabia' 'Senegal' 'Slovenia'
'South Africa' 'Spain' 'Sri Lanka' 'Sudan' 'Suriname' 'Sweden'
'Switzerland' 'Tajikistan' 'Thailand' 'Tunisia' 'Turkey' 'Uganda'
'Ukraine' 'United Kingdom' 'Uruguay' 'Zambia' 'Zimbabwe']
```

```
print("\n Unique Item Values Before Encoding:\n", yield_data["Item"].unique())
```

```
Unique Item Values Before Encoding:
['Maize' 'Potatoes' 'Rice, paddy' 'Sorghum' 'Soybeans' 'Wheat' 'Cassava'
'Sweet potatoes' 'Plantains and others' 'Yams']
```

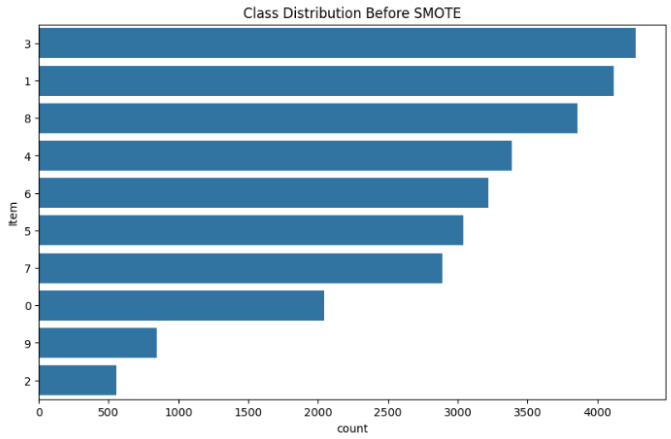
```
from sklearn.preprocessing import LabelEncoder

# Label encode 'Area'
le = LabelEncoder()
yield_data['Area'] = le.fit_transform(yield_data['Area'])
yield_data['Item'] = le.fit_transform(yield_data['Item'])
```

```
print("\n Unique Area Values After Encoding:\n", yield_data["Area"].unique())
print("\n Unique Item Values After Encoding:\n", yield_data["Item"].unique())
```

```
 Unique Area Values After Encoding:
 [  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53
  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71
  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89
  90  91  92  93  94  95  96  97  98  99 100]

 Unique Item Values After Encoding:
 [1 3 4 5 6 8 0 7 2 9]
```

- **SMOTE (Synthetic Minority Oversampling Technique)**: We applied SMOTE to address class imbalance in crop production records. This step was essential to ensure that the model was not biased toward more frequent yield patterns, thereby improving generalizability.

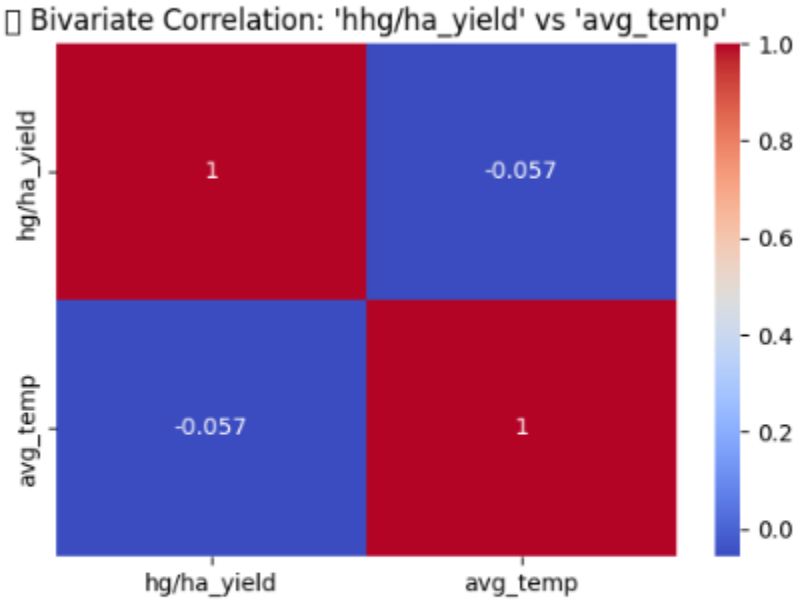
```
# Visual
plt.figure(figsize=(10, 6))
sns.countplot(y='Item', data=yield_data, order=yield_data['Item'].value_counts().index)
plt.title("Class Distribution Before SMOTE")
plt.show()
```




```
from imblearn.over_sampling import SMOTE

X = yield_data.drop('Item', axis=1)
y = yield_data['Item']

smote = SMOTE(random_state=42)
X_res, y_res = smote.fit_resample(X, y)

# Visualize balanced classes
plt.figure(figsize=(10, 6))
sns.countplot(y=y_res, order=pd.Series(y_res).value_counts().index)
plt.title("Class Distribution After SMOTE")
plt.show()
```

- **Feature Scaling**: Numerical features were standardized to bring all values to a similar range, improving model convergence and stability.
- **Data Cleaning**: Missing values, zero-area entries, and outliers were removed or treated to ensure data integrity and model robustness.

```
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: U
  fig.canvas.print_figure(bytes_io, **kw)
```



Feature Importance:

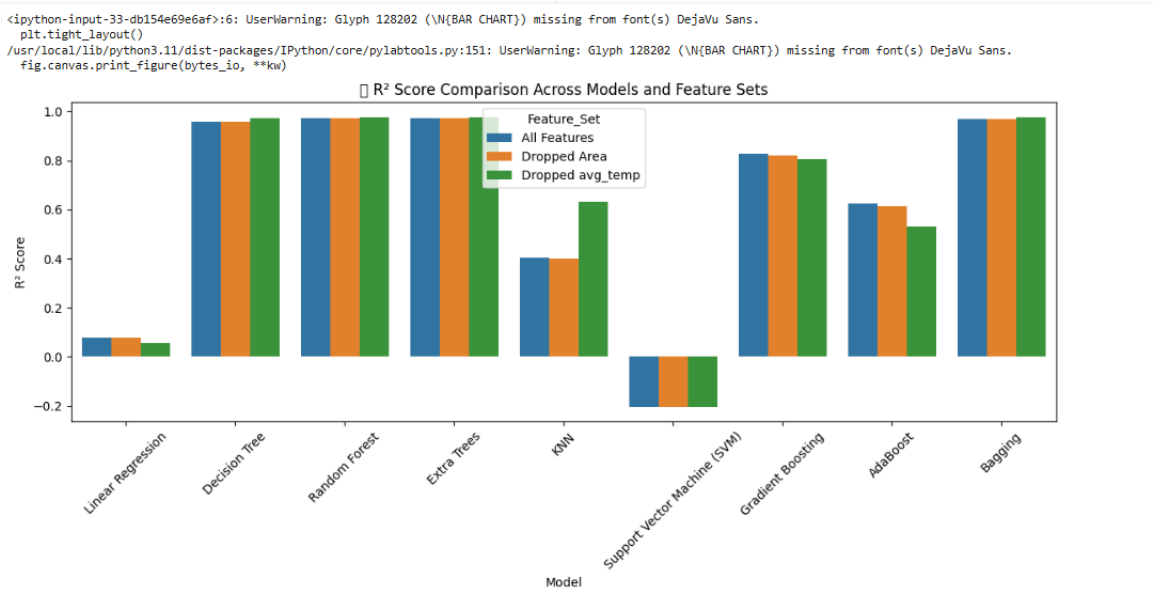
	Feature	Importance
1	hg/ha_yield	0.474432
3	pesticides_tonnes	0.152105
2	average_rain_fall_mm_per_year	0.150055
0	Area	0.116746
4	avg_temp	0.106662

3. Model Training

We developed and compared several regression models to predict crop yield based on the input features:

- Linear Regression
- Decision Tree Regressor
- Random Forest Regressor
- XGBoost Regressor

Among these, the **Extra Trees** achieved the highest performance with an R^2 score close to **0.99**, indicating excellent prediction capability.



Backend

Language & Framework:

- Python 3.9+:
  Used as the core language for scripting backend logic, model handling, and processing user inputs.
- Flask:
  A lightweight Python web framework chosen for its simplicity and flexibility. It manages HTTP routes, handles form submissions, renders HTML templates with Jinja2, and connects the frontend with backend logic smoothly.

Key Libraries Used:

- scikit-learn – For building and loading ML models (.sav files).
- pandas, numpy – For data preprocessing and numerical operations.
- joblib – For saving and loading trained models efficiently.
- matplotlib (if applicable) – For any graph/chart rendering in templates.

Backend Logic Flow:

This is how your Flask-based backend orchestrates the ML prediction process:

1. User Input Received

- The user fills out an HTML form with required inputs such as temperature, NPK values, pH, area, or crop type.
- The form is submitted using the **POST** method.
- Flask captures the request using request.form.get() to retrieve input values.

2. Data Preprocessing

- Inputs are **cleaned**, type-cast (e.g., string to float), and reshaped into a NumPy array suitable for model input.
- If any **categorical values** (like area or crop names) are involved, they are mapped to numerical codes using dictionaries like label_map.

3. Model Loading

- ML models (e.g., final_model_recommendation.sav, final_model_yield_prediction.sav) are **preloaded at the top** of the app using joblib.load().
- This avoids reloading models every time a request comes, improving performance.

```python
# 🌾 Yield Prediction
@app.route('/yield_prediction', methods=['GET', 'POST'])
def yield_prediction():
    if request.method == 'POST':
        area = int(request.form['area'])
        item = int(request.form['item'])
        average_rainfall = float(request.form['average_rainfall'])
        pesticides = float(request.form['pesticides'])

        features = [[area, item, average_rainfall, pesticides]]
        print("🧪 Yield Model Input Features:", features)

        yield_prediction = yield_model.predict(features)
        print("🧪 Yield Prediction Output:", yield_prediction)

        yield_predicted = yield_prediction[0]
        session['yield_prediction'] = yield_predicted
        return redirect(url_for('yield_result'))

    return render_template('yield_prediction.html', area_label_map=area_label_map, item_label_map=item_label_map)
```

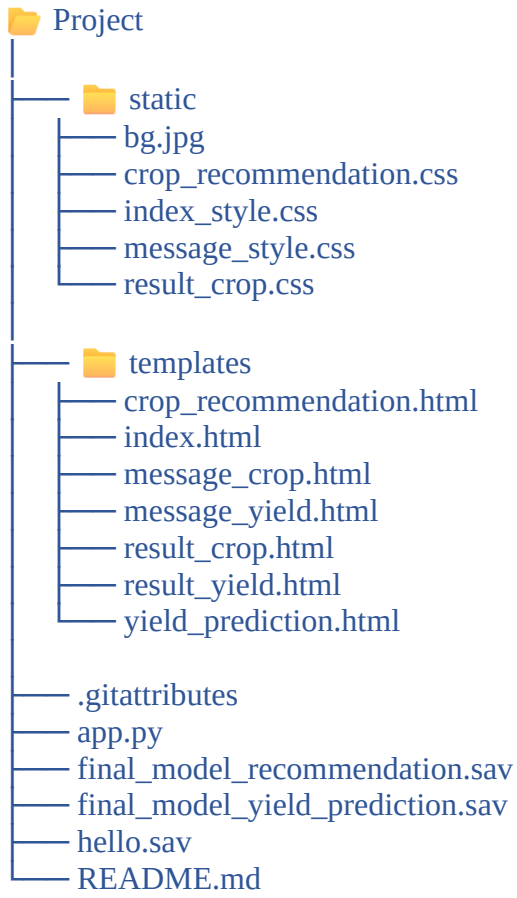
4. Prediction Made

- The cleaned and preprocessed input array is passed to the appropriate model's .predict() method.
- For **crop recommendation**, the numerical output is **decoded back** into a readable crop name using a reverse dictionary.
- For **yield prediction**, the output is a **numerical value**, representing the estimated yield in kg/hectare (or another relevant unit).

5. Result Sent to Frontend

- The predicted output is passed to the corresponding HTML template (e.g., result_crop.html or result_yield.html) using Flask's render_template().
- Dynamic values are injected into the template using **Jinja2**, and graphs or contextual information may be displayed for better user experience.

The full code can be accessed through our github:
https://github.com/vatssomya/Agrolytics



Integration

Model Integration with Flask Backend
After successfully training and saving the machine learning models, the focus shifted to deploying them in a web environment using Flask. This allows users to interact with the models via a user-friendly UI and receive predictions in real-time.

Integration Workflow:

- 1. **Model Deployment Setup**
  - Both models (final_model_recommendation.sav and final_model_yield_prediction.sav) are stored in the project directory.
  - These models are loaded once when the Flask server is initialized, ensuring faster response time for incoming predictions.

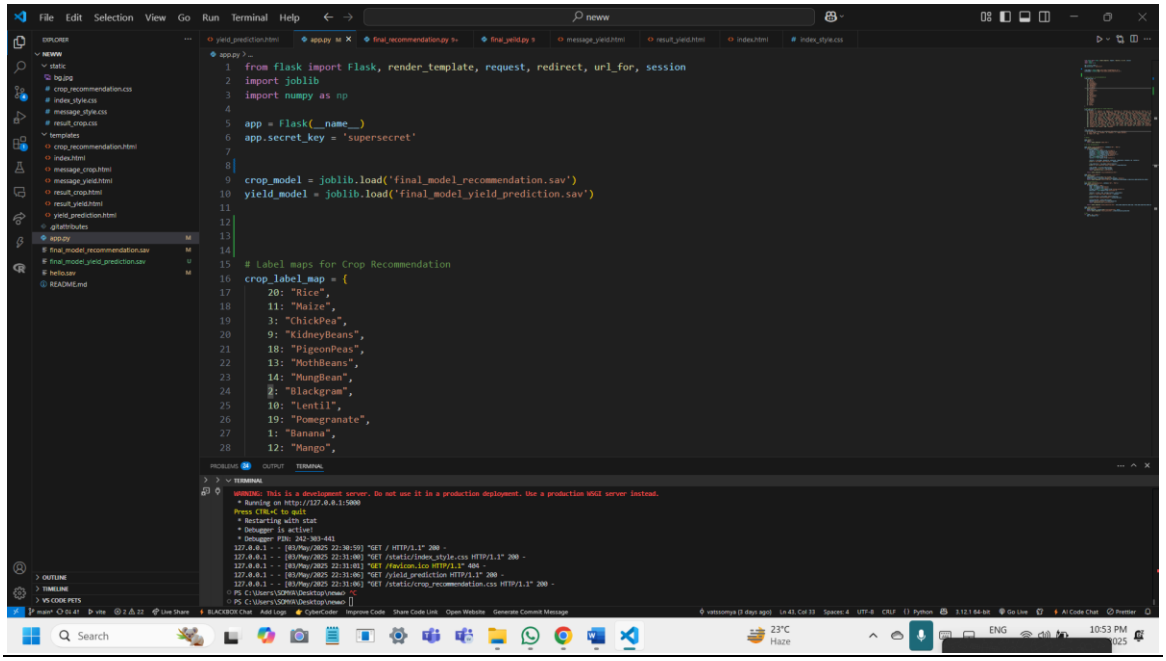
- 2. **Form Handling (Frontend to Backend)**
  - Users provide input data via HTML forms embedded in pages like crop_recommendation.html and yield_prediction.html.
  - These inputs include key features like:

- Soil nutrients: N, P, K
- Environmental conditions: Temperature, Humidity, pH
- Geographical data: Area name or crop type

- 3. **Data Processing for Prediction**
  - The backend retrieves form data using Flask's request.form.
  - Each input is validated and processed to match the structure expected by the model.
  - Encoding/mapping is done if the model was trained on encoded categorical features.

- 4. **Model Execution**
  - Based on the type of form (crop vs. yield), the corresponding model is used for prediction.
  - The model returns a prediction:
    - A class label (e.g., "Rice", "Wheat") for crop recommendation.
    - A numerical estimate (e.g., "2748.52 kg/ha") for yield prediction.

- 5. **Rendering Result Page**
  - The prediction is forwarded to a result page such as result_crop.html or result_yield.html.
  - The user sees a clear output, optionally with charts/graphs, along with any insights or next steps.



How One Can Use Our Project?

Pre-Requisites:

- **Python 3.9+**: Ensure Python 3.9 or later is installed on your system.
- **Required Packages**: The following Python libraries are required to run the project:
- Pandas
- NumPy
- Sklearn
- Streamlit (for frontend)
- Matplotlib/Seaborn (for data visualization)

Installation:

1. Clone the repository:

```
git clone https://github.com/somyavats/agrolytics
cd agrolytics
```

2. Install the necessary packages:

```
pip install -r requirements.txt
```

Run: Once the installation is complete, run the app using Streamlit:

```
streamlit run app.py
```

Usage:

- Open the application in a web browser.
- Fill out the form with input data such as crop type, rainfall, temperature, etc.
- Click on the **"Predict"** button.
- The system will display the crop health prediction along with recommendations and graphical representations of the results.

Directory Structure:

The frontend logic is mainly structured across two core folders: templates/ and static/. Below is a breakdown of the contents and their roles:

templates/ – HTML Templates
These files represent the dynamic web pages rendered by Flask.

- index.html: The homepage; introduces users to Agrolytics and provides access to both crop recommendation and yield prediction modules.
- crop_recommendation.html: Input form where users submit parameters (like soil, temperature, humidity) to get crop suggestions.
- yield_prediction.html: Input form for predicting the estimated yield based on field

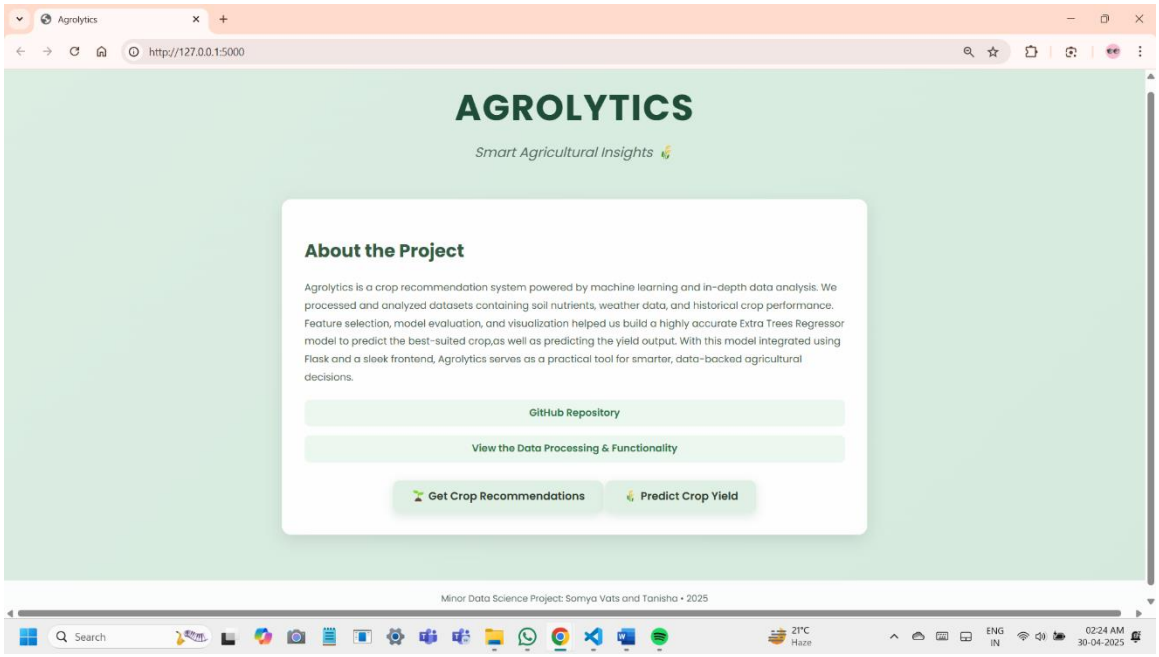
and environmental data.

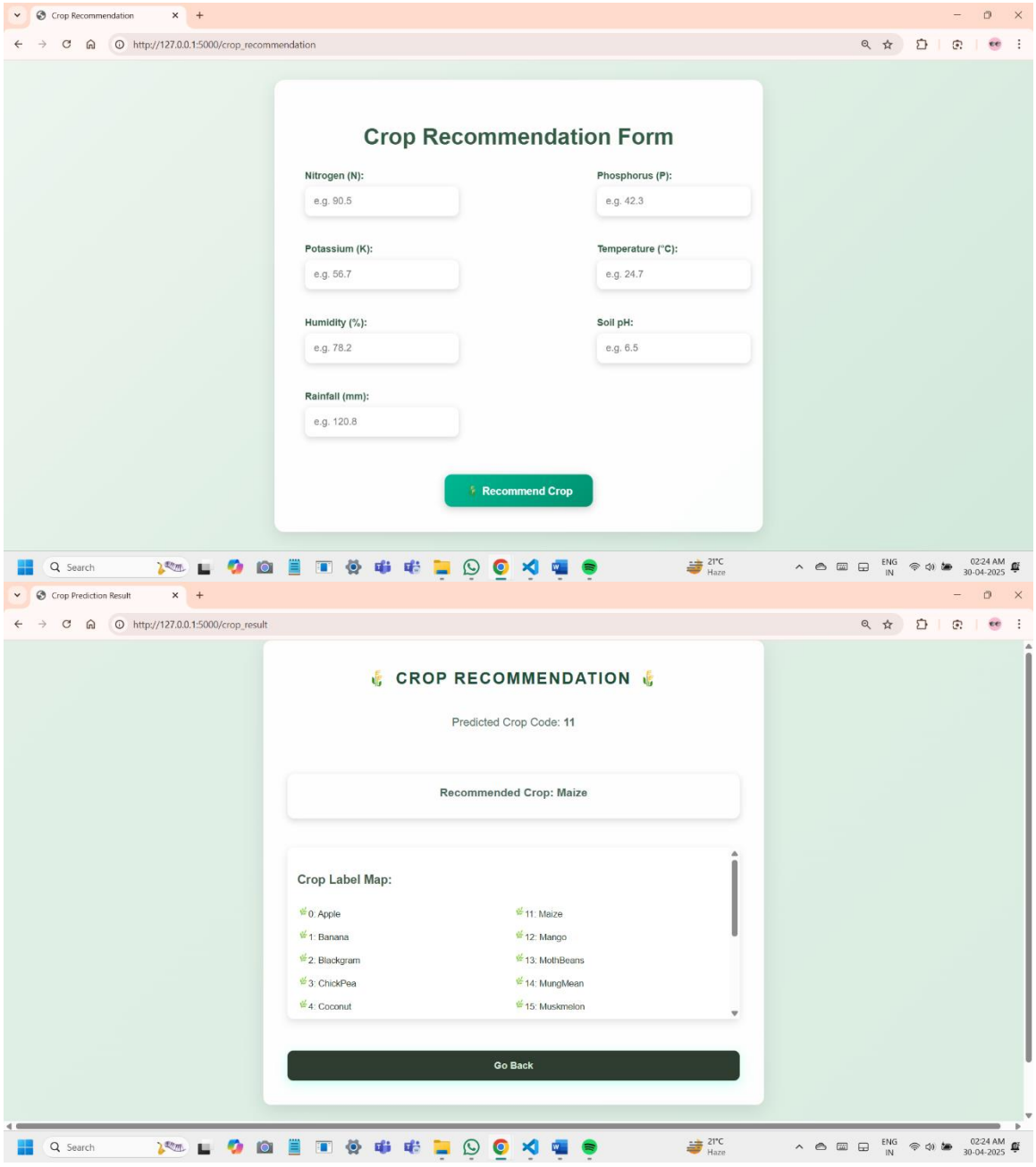
- result_crop.html: Displays the output from the crop recommendation model.
- result_yield.html: Shows predicted yield as per the ML model.
- message_crop.html and message_yield.html: Special pages to handle confirmations, alerts, or errors during input submission.

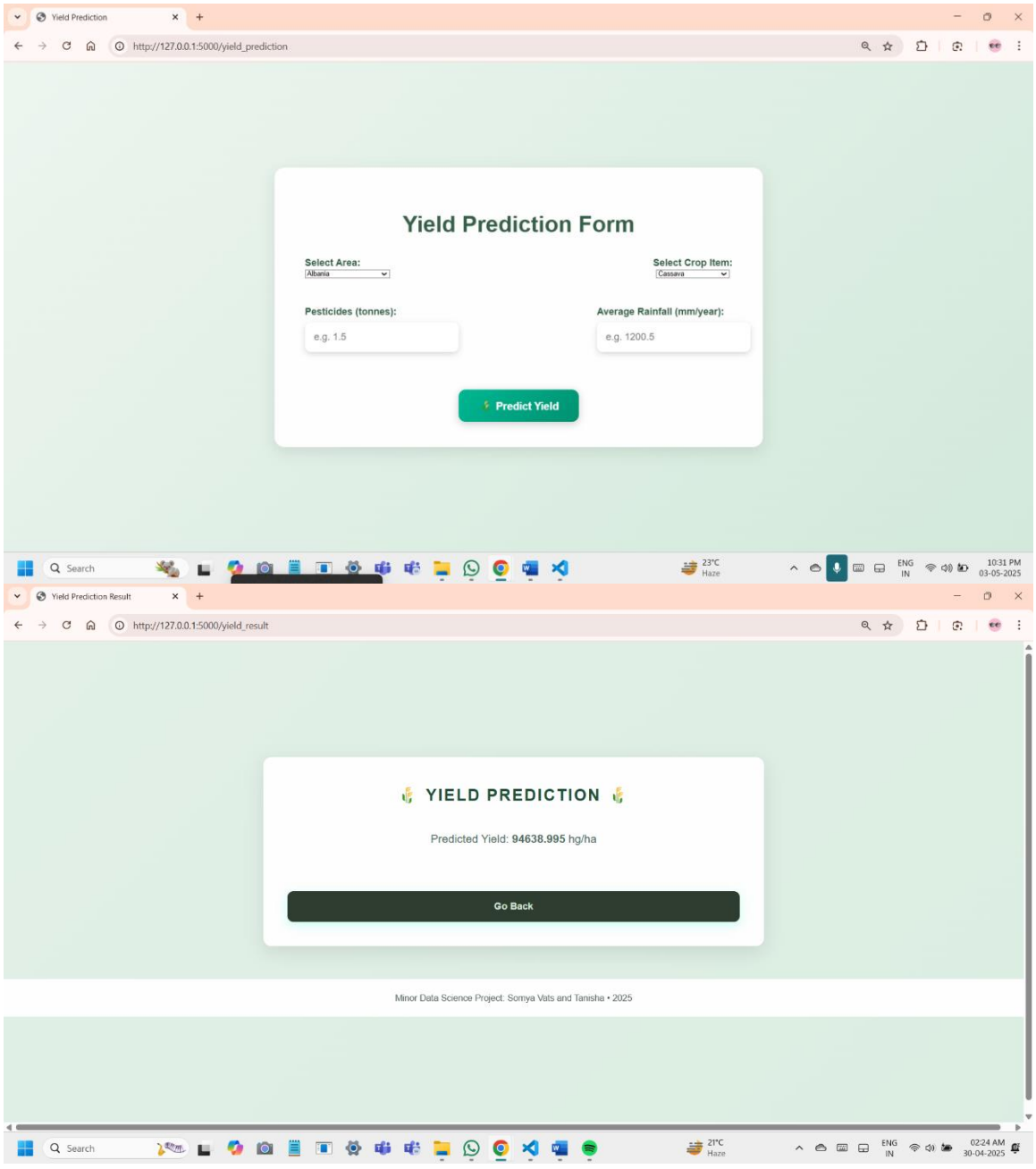
static/ – Styling and Assets

This folder contains all non-interactive, static files:

- **CSS Files**:
- index_style.css: Applies design to the homepage, including layout, fonts, and image styling.
- crop_recommendation.css and result_crop.css: Styles specific to crop-related pages.
- message_style.css: Defines alert box, confirmation styling, and UI feedback elements.




Crop Recommendation
http://127.0.0.1:5000/crop_recommendation
Crop Recommendation Form
Nitrogen (N):
e.g. 90.5
Phosphorus (P):
e.g. 42.3
Potassium (K):
e.g. 56.7
Temperature (°C):
e.g. 24.7
Humidity (%):
e.g. 78.2
Soil pH:
e.g. 6.5
Rainfall (mm):
e.g. 120.8
Recommend Crop
Crop Prediction Result
http://127.0.0.1:5000/crop_result
CROP RECOMMENDATION
Predicted Crop Code: 11
Recommended Crop: Maize
Crop Label Map:
0: Apple
1: Banana
2: Blackgram
3: ChickPea
4: Coconut
11: Maize
12: Mango
13: MothBeans
14: MungMean
15: Muskmelon
Go Back



Conclusion

In a world where agriculture is increasingly challenged by climate change, resource constraints, and a growing population, data-driven technologies can play a transformative role. Through this project, *Agrolytics: Intelligent Machine Learning-Based Crop Recommendation and Yield Prediction System*, we demonstrated how machine learning can empower farmers to make smarter and more sustainable decisions.
By building two integrated modules—Crop Recommendation and Yield Prediction—we showcased the application of algorithms like Extra Trees, Random Forest, and XGBoost

in extracting meaningful insights from agro-environmental data. The recommendation module helps determine the most suitable crops based on soil and climate parameters, while the yield prediction module provides reliable forecasts to optimize planning and resource allocation.

Our model achieved high accuracy and interpretability, thanks to robust preprocessing, exploratory analysis, and explainable AI methods. The use of open-source tools, government and Kaggle datasets, and platforms like Google Colab ensured accessibility and reproducibility.

This system contributes directly to Sustainable Development Goal 2 (Zero Hunger) by supporting precision agriculture, minimizing trial-and-error for farmers, and maximizing productivity with minimal environmental impact. While this project lays the groundwork, future improvements—like real-time data integration, mobile app deployment, and satellite data inclusion—can further elevate its practical impact in the field.

Agrolytics is not just a technical solution—it’s a step toward building a more food-secure and sustainable future.