OPTICROP AGRICULTURAL OPTIMISATION

MACHINE LEARNING

SmartInternz

www.smartinternz.com

OPTICROP AGRICULTURAL OPTIMISATION

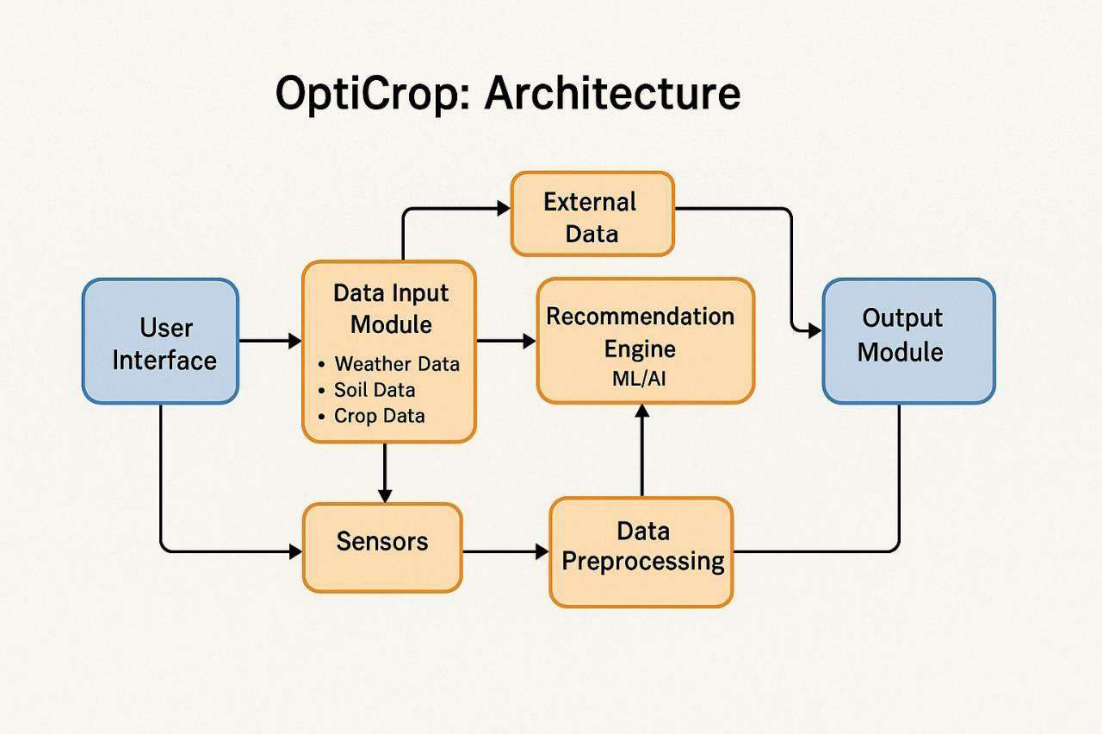
Agriculture is facing increased pressure to meet global food demands amidst climate change and resource limitations. Traditional farming methods often lack precision and real-time insights, resulting in lower yields and inefficient use of resources.

OptiCrop addresses these challenges through an intelligent system that empowers farmers to make data-driven decisions. By leveraging machine learning and environmental data, OptiCrop offers crop recommendations, irrigation schedules, fertilizer plans, pest alerts, and yield predictions.

This platform is designed to:

- Maximize crop yields and profitability
- Reduce operational costs through optimized resource use
- Promote environmentally sustainable farming practices

Technical Architecture:



Project Flow:

- The user interacts with the UI to enter agricultural data
- The integrated ML model analyzes data
- Model outputs optimized recommendations, which are displayed in the UI.

To accomplish this, we follow these stages:

- Define Problem / Problem Understanding
- Data Collection & Preparation
- Exploratory Data Analysis
- Model Building
- Performance Testing & Hyperparameter Tuning
- Model Deployment
- Project Demonstration & Documentation

Prior Knowledge:

To effectively understand and implement this project, participants should possess prior knowledge in the following key areas:

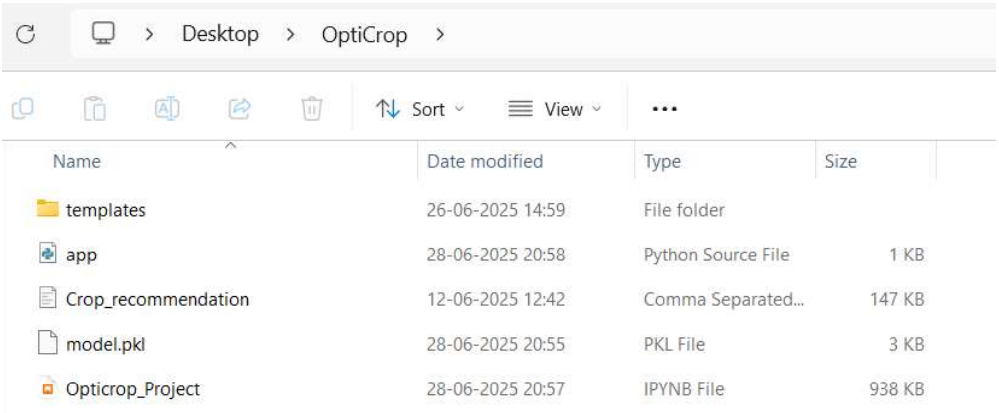
- **ML Concepts: Supervised and Unsupervised Learning -** A fundamental understanding of machine learning paradigms, particularly supervised learning (for tasks such as classification and regression) and unsupervised learning (for tasks likeclustering and dimensionality reduction), is essential. This includes knowing when and how to apply different learning approaches.
- **Decision Tree, Random Forest, KNN, Logistic Regression, SVM:** Familiarity with the theoretical underpinnings and practical application of these common classification algorithms is crucial. This involves understanding their strengths, weaknesses, and appropriate use cases in predictive modeling.
- **Evaluation Metrics: Accuracy, Precision, Recall, F1-Score:** A strong grasp of model evaluation metrics is necessary to assess the performance of machine

learning models accurately. Understanding how to interpret these metrics (especially in the context of imbalanced datasets) is vital for selecting the best-performing model.

- **Flask Web Development:** Basic knowledge of Flask, a Python micro-framework, is required for integrating the trained machine learning model into a web application. This includes understanding routing, templating (Jinja2), handling HTTP requests (GET/POST), and deploying simple Flask applications.
- **Data Processing & Visualization in Python:** Proficiency in Python libraries such as Pandas for data manipulation and NumPy for numerical operations is critical. Additionally, experience with data visualization libraries like Matplotlib and Seaborn for exploratory data analysis and presenting insights is highly beneficial.

Project Structure:

Create a project folder that includes:



- Flask application (app.py)
- HTML templates folder (templates)
- Trained model (model.pkl)
- Dataset folder (Crop_recommendation)
- Jupyter Notebook for model development (Opticrop_model.ipynb)

Milestone-1: Define Problem/Problem Understanding

Activity-1: Specify the Business Problem

Farmers today operate in an increasingly complex environment, contending with unpredictable weather patterns, soil degradation, pest outbreaks, and fluctuating market prices. Traditional farming practices, often reliant on intuition and historical methods, are insufficient to optimize resource allocation and maximize yields under these dynamic conditions. There is a significant business problem in the agricultural sector: a lack of adaptive, data-driven systems that can provide precise, real-time guidance for optimal decision-making regarding crop selection, irrigation management, nutrient application, and pest control. OptiCrop aims to bridge this critical gap by offering intelligent, actionable recommendations derived from the analysis of real-time and historical agricultural data, thereby transforming reactive farming into proactive, optimized cultivation.

Activity-2: Business Requirements

To address the specified business problem, OptiCrop must fulfill several key requirements:

- **Crop Optimization for Sustainable Agriculture:** The system must accurately recommend the most suitable crops for specific land parcels based on soil characteristics, climate data, and water availability, promoting sustainable practices that conserve resources and reduce environmental impact.
- **Real-time Environmental Data Integration:** It needs to seamlessly integrate with various data sources, including IoT sensors for soil moisture and nutrient levels, weather APIs for current and forecasted conditions, and satellite imagery, to provide up-to-the-minute insights.
- **AI-based Farming Insights:** The core functionality must involve robust machine learning models capable of analyzing complex datasets to generate precise recommendations for irrigation schedules, fertilizer types and quantities, and early warnings for potential pest or disease outbreaks.
- **Easy-to-Use Farmer Dashboard:** The user interface (UI) must be intuitive and accessible, allowing farmers with varying levels of technical expertise to easily input data, understand recommendations, and visualize key metrics without extensive training. This includes clear visualizations and actionable alerts.

Activity-3: Literature Survey

A comprehensive literature review reveals that while various agricultural tools exist, many are siloed, focusing on single aspects like irrigation or pest management, or are based on rigid, rule-based systems that struggle with variability. There is a clear and documented need for integrated, machine-learning-driven systems that can synthesize diverse cross-functional data (e.g., weather, soil, market, historical yields) to provide holistic and adaptive recommendations. Existing research highlights the potential of AI and big data in agriculture but also points to the challenge of developing user-friendly, scalable solutions that can be adopted by a wide range of farmers. OptiCrop directly addresses this identified need by integrating multiple data streams and leveraging advanced ML algorithms to offer a comprehensive, dynamic, and user-centric agricultural optimization platform.

Activity-4: Social or Business Impact

The successful implementation of OptiCrop is expected to yield significant impacts on both societal and business levels:

- **Social Impact:** OptiCrop empowers individual farmers and agricultural communities by providing them with the tools to grow food more sustainably and efficiently. This leads to enhanced food security, reduced environmental footprint (e.g., less water waste, optimized fertilizer use), and potentially improved livelihoods for farmers through better yields and reduced input costs. It contributes to a more resilient and sustainable agricultural ecosystem.
- **Business Impact:** For agricultural businesses, OptiCrop directly translates into maximized profits and improved yield consistency. By optimizing planting strategies, irrigation, fertilization, and pest management, the system minimizes operational costs, reduces crop losses, and increases overall productivity. This leads to higher quality produce, more predictable harvests, and a stronger competitive advantage in the market, making farming a more economically viable and stable enterprise.

Milestone-2: Data Collection & Preparation

This milestone focuses on acquiring the necessary data and preparing it for machine learning model training. High-quality, well-prepared data is fundamental to the success of any machine learning project.

Activity-1: Collect the Dataset

The initial step involves gathering relevant agricultural datasets. These datasets are crucial for training the machine learning models to identify patterns and make accurate predictions.

- **Data Sources:** Data is sourced from popular open-source platforms such as Kaggle and CropWatch. These platforms provide a wide array of publicly available datasets related to agriculture.
- **Data Content:** The collected datasets typically include various parameters critical for agricultural optimization, such as:
    - **Soil Conditions:** pH levels, nutrient content (Nitrogen, Phosphorus, Potassium), organic matter.
    - **Crop Types:** Information on different crops, their growth cycles, and specific requirements.
    - **Weather Data:** Historical and real-time weather patterns including temperature, humidity, rainfall, and sunlight hours.
    - **Historical Yield Records:** Past production data for various crops under different conditions, which helps in predicting future yields

Activity 1.1: Importing the libraries:

```
import pandas as pd
import numpy as np
pd.set_option('max_colwidth', 20)
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', 50)
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
plt.rcParams['figure.figsize'] = (12,8)
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = 'all'
from ipywidgets import interact
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report
```

Activity 1.2: Read the Dataset:

```
data = pd.read_csv(r'C:\Users\psiri\Desktop\OptiCrop\Crop_recommendation.csv')
data.head()
```

	N	P	K	temperature	humidity	ph	rainfall	label
0	90	42	43	20.879744	82.002744	6.502985	202.935536	rice
1	85	58	41	21.770462	80.319644	7.038096	226.655537	rice
2	60	55	44	23.004459	82.320763	7.840207	263.964248	rice
3	74	35	40	26.491096	80.158363	6.980401	242.864034	rice
4	78	42	42	20.130175	81.604873	7.628473	262.717340	rice

Activity-2: Data Preparation

Once collected, raw data often contains inconsistencies, missing values, and noise that can negatively impact model performance. This activity involves a series of pre-processing steps to clean and transform the data into a suitable format for machine learning.

- Handling Missing Values:
  - The dataset is thoroughly checked for any missing or null values using functions like df.isna().any() and .sum().
  - If missing values are identified, appropriate imputation techniques (e.g., mean, median, mode imputation, or more advanced methods) are applied to ensure data completeness. For this project, if no null values are found, this step is skipped.

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2200 entries, 0 to 2199
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   N            2200 non-null   int64  
 1   P            2200 non-null   int64  
 2   K            2200 non-null   int64  
 3   temperature  2200 non-null   float64
 4   humidity     2200 non-null   float64
 5   ph           2200 non-null   float64
 6   rainfall     2200 non-null   float64
 7   label        2200 non-null   object 
dtypes: float64(4), int64(3), object(1)
memory usage: 137.6+ KB
```

```
data.isnull().sum()
```

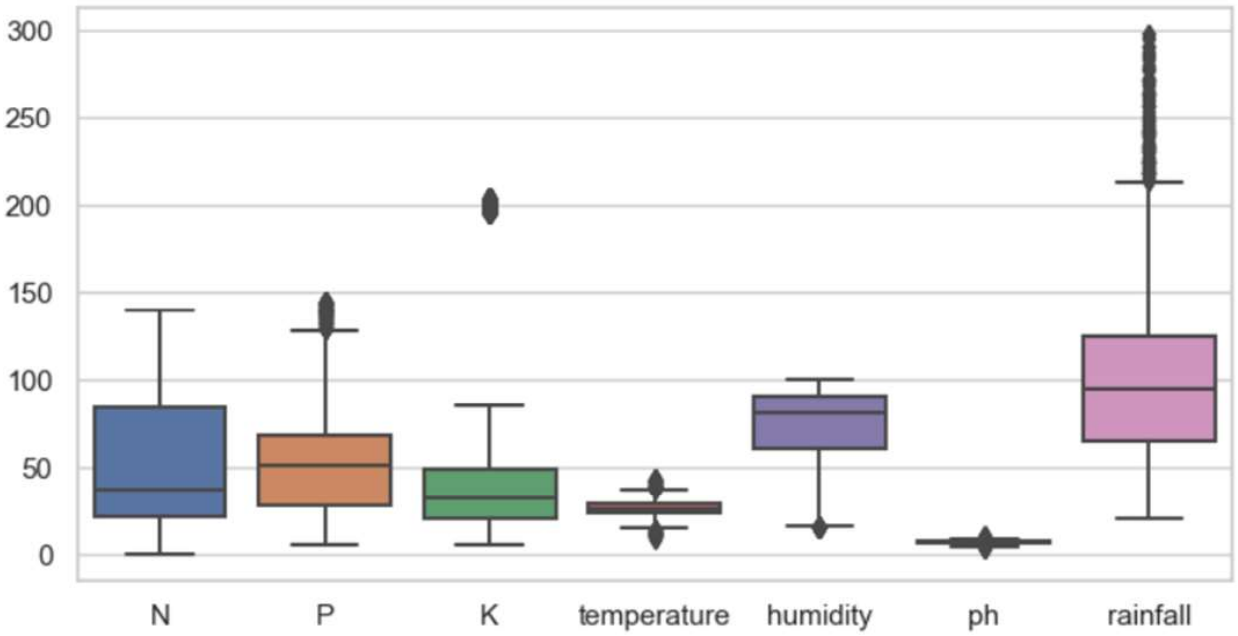
```
N              0
P              0
K              0
temperature    0
humidity       0
ph             0
rainfall       0
label          0
dtype: int64
```

- **Handling Outliers:**
  - Outliers, which are data points significantly different from other observations, are identified using visualization techniques such as box plots.Mathematical formulas, such as the Interquartile Range (IQR) method (where Upper Bound = Q3 + 1.5 * IQR and Lower Bound = Q1 - 1.5 * IQR), are used to define boundaries for outlier detection.

- Transformation techniques, like log transformation, are applied to mitigate the impact of outliers and normalize data distribution, improving model stability.

```
plt.figure(figsize=(8,4))
sns.boxplot(data)
```

```
<Figure size 800x400 with 0 Axes>

<Axes: >
```



- **Encoding Categorical Variables:**
  - Machine learning models typically require numerical input. Categorical features (e.g., soil type, crop variety) are converted into numerical representations using encoding techniques.
  - LabelEncoder from the sklearn.preprocessing library is utilized to transform categorical features into numerical ones based on their unique values.

- **Scaling Numerical Features:**
  - Features with different scales can disproportionately influence machine learning algorithms. Scaling ensures all numerical features contribute

equally to the model.

- StandardScaler is applied to transform the data to have a mean of 0 and a standard deviation of 1, following the formula: Xscaled=(X−Xmean)/Xstd.

- **Balancing the Dataset using SMOTE:**
  - In real-world agricultural datasets, certain outcomes (e.g., pest outbreaks) might be rare, leading to imbalanced classes. Imbalanced data can cause models to be biased towards the majority class.
  - The Synthetic Minority Over-sampling Technique (SMOTE) is employed to oversample the minority class, creating synthetic samples to balance the dataset and improve the model's ability to predict rare events accurately.
- **Splitting into Training and Testing Datasets:**
  - The prepared dataset is divided into two subsets: a training set and a testing set.
  - The target variable (e.g., yield prediction, crop recommendation) is separated into y, while the remaining features form x.
  - train_test_split() from sklearn.model_selection is used, with parameters for test_size (e.g., 20-30% for testing) and random_state for reproducibility. This split ensures the model is evaluated on unseen data.

Milestone-3: Exploratory Data Analysis

Activity-1: Descriptive Statistics

Exploratory Data Analysis (EDA) is a crucial step to understand the underlying structure of the data, identify patterns, detect anomalies, and test hypotheses. It involves both statistical and visual methods.

```
# 4. Descriptive Analysis
data.describe()
```

	N	P	K	temperature	humidity	ph	rainfall
count	2062.000000	2062.000000	2062.000000	2062.000000	2062.000000	2062.000000	2062.000000
mean	52.440349	47.706596	37.996605	25.770363	70.433297	6.502347	104.286486
std	37.246143	25.479349	33.049629	4.885812	22.569963	0.785073	56.385635
min	0.000000	5.000000	5.000000	9.724458	14.258040	3.504752	20.211267
25%	22.000000	27.000000	20.000000	23.080864	58.469697	6.016597	62.940621
50%	39.000000	48.000000	30.000000	25.768297	79.175605	6.469677	94.772563
75%	87.000000	64.000000	46.000000	28.614586	89.130631	6.956328	132.787974
max	140.000000	128.000000	205.000000	43.675493	99.981876	9.935091	298.560117

Activity-2: Visual Analysis

Visual analysis involves using graphical representations to explore and understand data. It allows for quick identification of patterns, trends, and outliers that might not be obvious from numerical data.

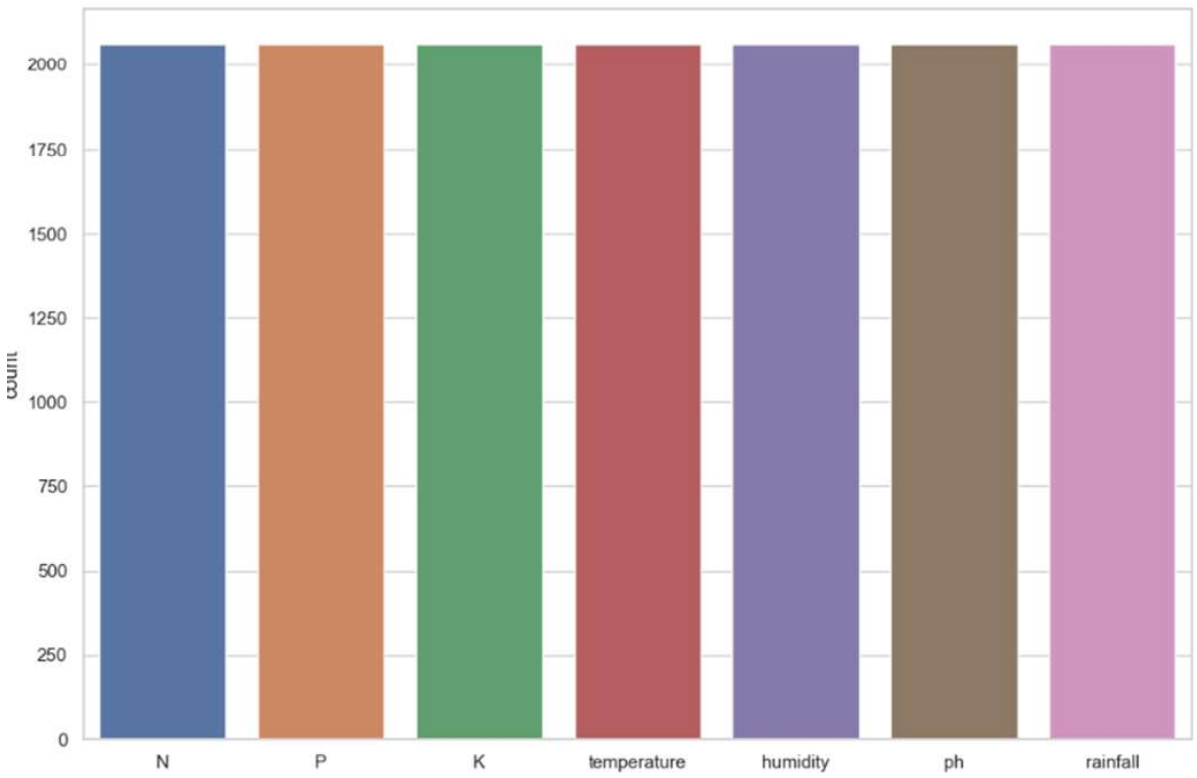
Activity 2.1: Univariate Analysis

This analysis focuses on understanding individual features within the dataset.

- **Count Plots (Seaborn):** Used for categorical features to visualize the frequency of each unique value. For example, a countplot could show the distribution of different soil types or crop varieties.

```
# Count Plot
sns.countplot(data)
```

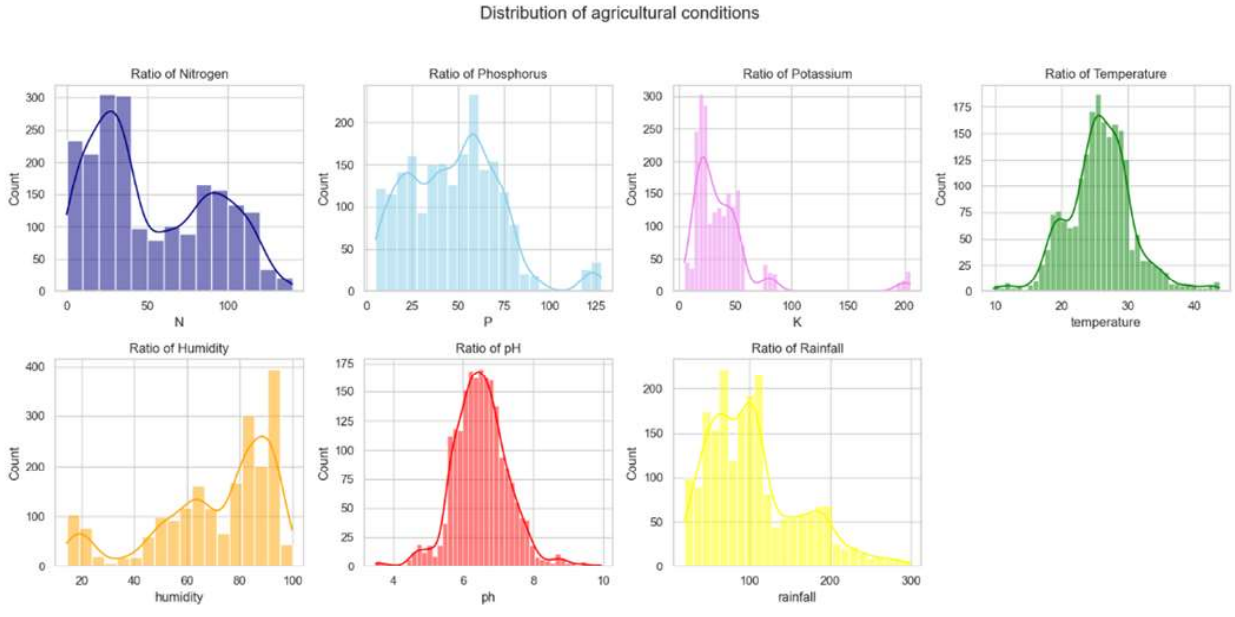
```
<Axes: ylabel='count'>
```



- **Histograms (Matplotlib/Seaborn):** Used for numerical features to display the distribution of data points. For instance, a histogram of 'temperature' could show if the data is normally distributed or skewed, and identify common temperature ranges.

```
# 1. Univariate Analysis
# Histogram
sns.set(style="whitegrid")

features = ['N', 'P', 'K', 'temperature', 'humidity', 'ph', 'rainfall']
titles = ['Ratio of Nitrogen', 'Ratio of Phosphorus', 'Ratio of Potassium',
          'Ratio of Temperature', 'Ratio of Humidity', 'Ratio of pH', 'Ratio of Rainfall']
colors = ['navy', 'skyblue', 'violet', 'green', 'orange', 'red', 'yellow']

fig, axes = plt.subplots(2, 4, figsize=(16, 8))
axes = axes.flatten()

for i, feature in enumerate(features):
    sns.histplot(data[feature], kde=True, ax=axes[i], color=colors[i])
    axes[i].set_title(titles[i])

fig.delaxes(axes[-1])

plt.suptitle("Distribution of agricultural conditions", fontsize=16)
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()
```

```
<Axes: xlabel='N', ylabel='Count'>

Text(0.5, 1.0, 'Ratio of Nitrogen')

<Axes: xlabel='P', ylabel='Count'>

Text(0.5, 1.0, 'Ratio of Phosphorus')

<Axes: xlabel='K', ylabel='Count'>

Text(0.5, 1.0, 'Ratio of Potassium')

<Axes: xlabel='temperature', ylabel='Count'>

Text(0.5, 1.0, 'Ratio of Temperature')

<Axes: xlabel='humidity', ylabel='Count'>

Text(0.5, 1.0, 'Ratio of Humidity')

<Axes: xlabel='ph', ylabel='Count'>

Text(0.5, 1.0, 'Ratio of pH')

<Axes: xlabel='rainfall', ylabel='Count'>

Text(0.5, 1.0, 'Ratio of Rainfall')

Text(0.5, 0.98, 'Distribution of agricultural conditions')
```



- **Pie Charts:** Useful for showing the composition of a categorical feature, illustrating the proportion of each category within the whole (e.g., percentage breakdown of different irrigation methods, etc.)

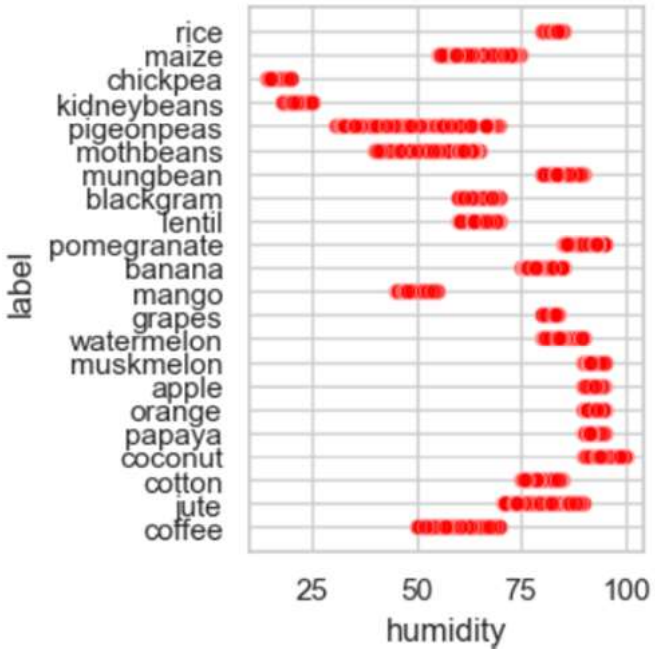
Activity 2.2: Bivariate Analysis

This analysis explores the relationship between two features.

- **Bar Plot (Seaborn):** Effective for visualizing the relationship between a categorical variable and a numerical variable. For example, a barplot could show the average yield for different fertilizer types or the total water consumption across various crop cycles.
- **Scatter Plot (Matplotlib/Seaborn):** Ideal for visualizing the relationship between two numerical variables, helping to identify correlations or clusters (e.g., relationship between soil moisture and crop growth rate).

```
# 2. Bivariate Analysis
# Scatter Plot
plt.subplot(2, 4, 7)
sns.scatterplot(x=data['humidity'], y=data['label'],alpha = 0.5, c = 'red', edgecolors = 'black')
```

```
<Axes: >

<Axes: xlabel='humidity', ylabel='label'>
```



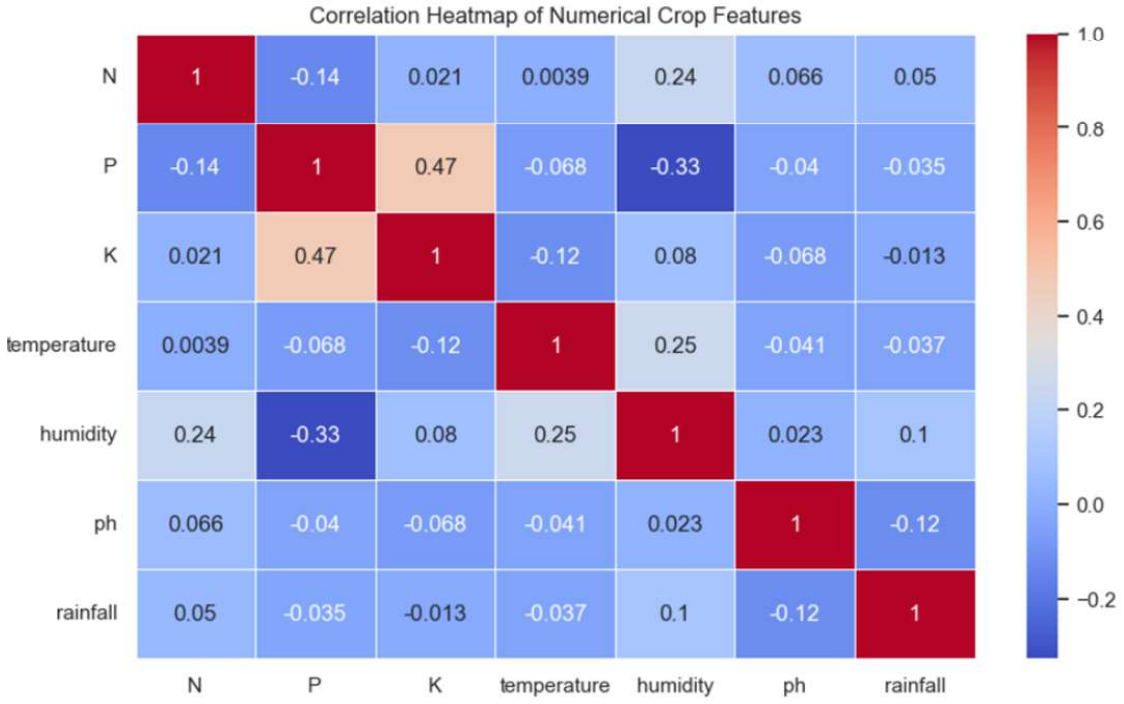
Activity 2.3: Multivariate analysis

This analysis investigates the relationships among three or more features simultaneously.

- **Heatmaps (Seaborn):** Particularly useful for visualizing correlation matrices between multiple numerical features. A heatmap can quickly highlight highly correlated features (e.g., correlation between soil pH, nutrient levels, and crop yield). Highly correlated features might indicate multicollinearity, which could necessitate dropping one of the correlated features to improve model performance and interpretability

```
# Heat Map
numeric_df = data.select_dtypes(include='number')
correlation_matrix = numeric_df.corr()
plt.figure(figsize=(10,6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title("Correlation Heatmap of Numerical Crop Features")
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
<Axes: >
Text(0.5, 1.0, 'Correlation Heatmap of Numerical Crop Features')
```



Milestone-4: Model Building

Activity-1: Training the model in multiple algorithms

After the data has been thoroughly prepared, the next crucial step is to train various machine learning models. The goal is to identify which algorithm performs best for the specific agricultural prediction tasks (e.g., crop recommendation, yield prediction).

Activity 1.1: Algorithm Selection:

For this project, several widely used classification algorithms are applied due to their proven effectiveness in various domains:

- **K-Nearest Neighbors (KNN):** A non-parametric, lazy learning algorithm used for both classification and regression. It classifies a data point based on the majority class among its 'k' nearest neighbors in the feature space.
- **Logistic Regression:** A linear model used for binary classification. Despite its name, it's a classification algorithm that estimates the probability of an instance belonging to a particular class.

Activity 1.2: Implementation with Scikit-learn:

All these models are implemented using the Scikit-learn library in Python, which provides efficient and user-friendly tools for machine learning.

Training Process:

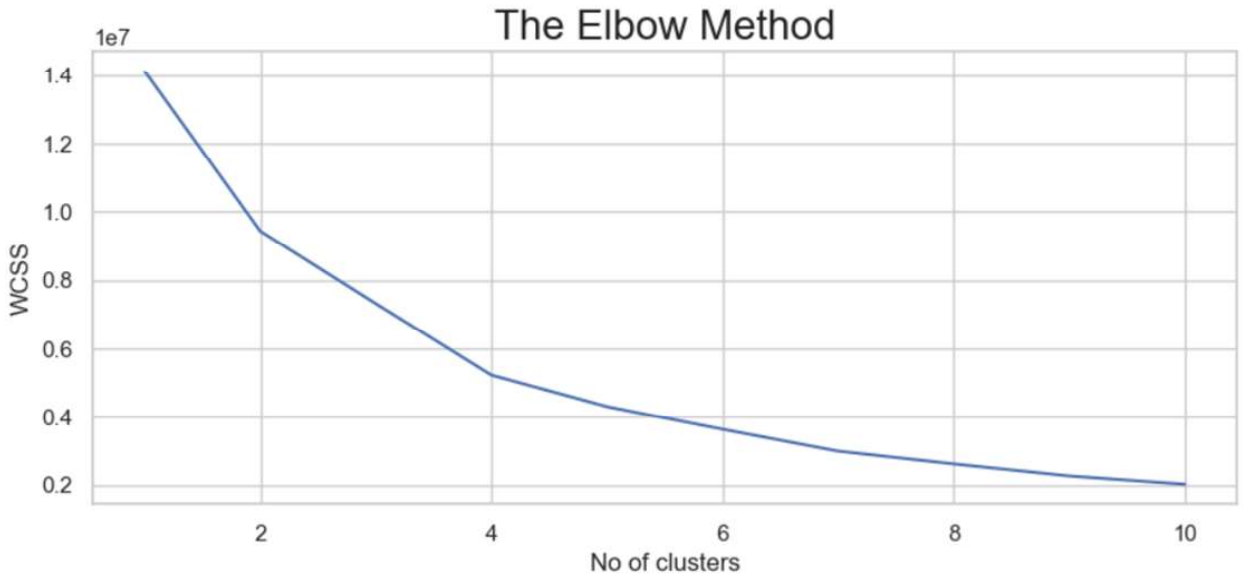
- Each chosen algorithm (e.g., Decision Tree Classifier, Random Forest Classifier) is imported and initialized.
- The fit() function is used to train the model on the X_train (features) and y_train (target variable) datasets. This step allows the model to learn the patterns and relationships from the training data.

Activity 1.3: K Means

KNN Model is imported from the sklearn library, then K Neighbors Classifier algorithm is initialised and training data is passed to the model with the .fit() function. Test data is predicted with the predict() function and saved in a variable. For evaluating the model, confusion matrix and classification report are used.

```python
# 1. K-Means
plt.rcParams['figure.figsize'] = (10, 4)
wcss = []

for i in range(1, 11):
    km = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_state=0)
    km.fit(x)  # x should be your selected feature set
    wcss.append(km.inertia_)

plt.plot(range(1, 11), wcss)
plt.title("The Elbow Method", fontsize=20)
plt.xlabel("No of clusters")
plt.ylabel("WCSS")
plt.show()
```

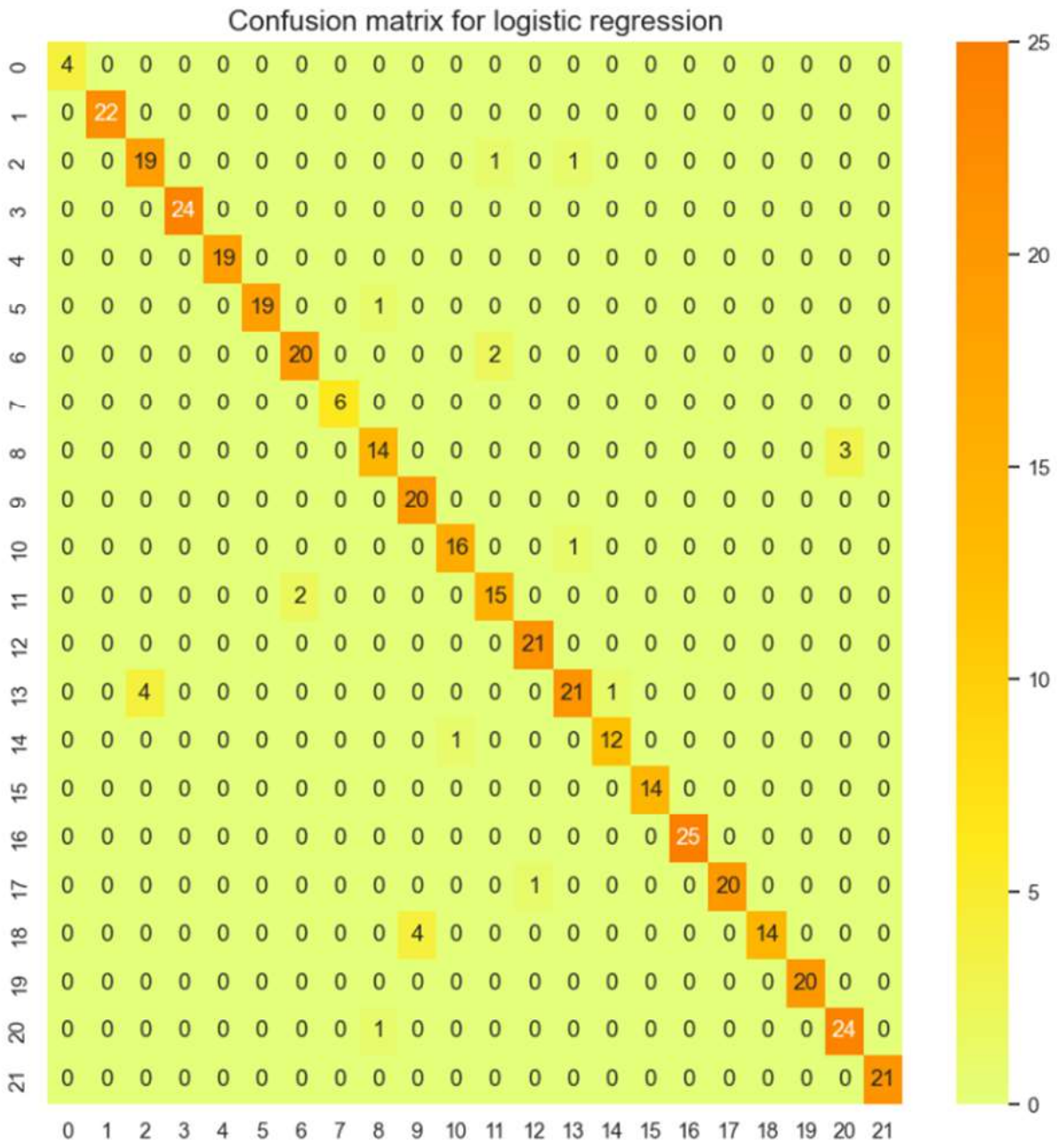
```
[<matplotlib.lines.Line2D at 0x2608a1e6510>]

Text(0.5, 1.0, 'The Elbow Method')

Text(0.5, 0, 'No of clusters')

Text(0, 0.5, 'WCSS')
```



Activity 1.4: Logistic Regression Model

A linear model is used for binary classification. Despite its name, it's a classification algorithm that estimates the probability of an instance belonging to a particular class.

```python
# 2. Logistic Regression Model
model = LogisticRegression()
model.fit(x_train,y_train)
y_pred = model.predict(x_test)
```



Activity-2: Testing the model

Once the models are trained, it is essential to evaluate their performance on unseen data to ensure they generalize well and are not overfitting to the training data.

- **Prediction:** The trained models use the predict() function to make predictions on the X_test dataset. These predictions are then compared against the actual y_test values.
- **Evaluation Metrics:** Model performance is rigorously evaluated using a suite of

standard classification metrics:

- **Confusion Matrix:** A table used to describe the performance of a classification model on a set of test data for which the true values are known. It helps visualize the performance of an algorithm.
- **Accuracy:** The proportion of correctly classified instances out of the total instances. While a good general metric, it can be misleading in imbalanced datasets.
- **Precision:** The ratio of correctly predicted positive observations to the total predicted positive observations. It measures the model's ability to avoid false positives.
- **Recall (Sensitivity):** The ratio of correctly predicted positive observations to all observations in the actual class. It measures the model's ability to find all positive instances.
- **F1-Score:** The weighted average of Precision and Recall. It is a more robust metric than accuracy, especially for imbalanced datasets, as it balances both precision and recall.

- **Cross-validation:** While detailed in Milestone 5, preliminary testing might also involve understanding consistency across different data folds.

Milestone 5: Performance Testing & Hyperparameter Tuning

Activity-1: Compare Models

After training multiple models, it's critical to systematically compare their performance across various evaluation metrics to identify the most suitable model for the OptiCrop application.

- **Comprehensive Evaluation:** Instead of relying on a single metric, models are tested with multiple evaluation metrics (accuracy, precision, recall, F1-score, and support) to gain a holistic understanding of their strengths and weaknesses.
- **compareModel Function:** A custom function, compareModel, is defined to streamline the comparison process. This function takes the trained models and test data as input, calculates all relevant metrics for each model, and presents them in a structured format (e.g., a DataFrame or a table).
- **Result Analysis:** The output of the compareModel function displays the performance metrics for each algorithm (Decision Tree, Random Forest, KNN,

Logistic Regression, SVM).

- **Model Selection:** Based on this comprehensive comparison, the **Random Forest** model is identified as the best performer. It typically offers a strong balance between overall accuracy and robust performance across precision and recall, making it well-suited for agricultural prediction tasks where both false positives and false negatives can have significant implications.

```
km = KMeans(n_clusters=4,init="k-means++",max_iter=300,n_init=10,random_state=0)
y_means=km.fit_predict(x)

a=data['label']
y_means=pd.DataFrame(y_means)
z=pd.concat([y_means,a],axis=1)
z=z.rename(columns={0:'cluster'})

print("lets check the results after applying the K-Means clustering analysis \n")
print("Crops in First cluster:",z[z['cluster']==0]['label'].unique())
print()
print("Crops in Second cluster:",z[z['cluster']==1]['label'].unique())
print()
print("Crops in Third cluster:",z[z['cluster']==2]['label'].unique())
print()
print("Crops in Fourth cluster:",z[z['cluster']==3]['label'].unique())
```

```
lets check the results after applying the K-Means clustering analysis

Crops in First cluster: ['maize' 'chickpea' 'kidneybeans' 'pigeonpeas' 'mothbeans' 'mungbean'
 'blackgram' 'lentil' 'pomegranate' 'mango' 'muskmelon' 'apple' nan
 'orange' 'papaya']

Crops in Second cluster: ['rice' 'pigeonpeas' 'apple' nan 'orange' 'papaya' 'coconut' 'cotton'
 'jute']

Crops in Third cluster: ['maize' 'banana' nan 'grapes' 'watermelon' 'muskmelon' 'orange' 'papaya'
 'coconut' 'cotton' 'jute']

Crops in Fourth cluster: [nan 'grapes' 'muskmelon']
```

```
# 3. Evaluating the performance of the model and saving the model
plt.rcParams["figure.figsize"] = (10,10)
cm = confusion_matrix(y_test,y_pred)
sns.heatmap(cm,annot = True, cmap = 'Wistia')
plt.title("Confusion matrix for logistic regression", fontsize = 15)
plt.show()
```

```
<Axes: >

Text(0.5, 1.0, 'Confusion matrix for logistic regression')
```

```
# 4. Predict the best crop according to the given parameters
prediction = model.predict((np.array([[105,35,40,25,64,7,160]])))
print("The suggested crop for given climatic condition is: ",prediction)
```

```
The suggested crop for given climatic condition is:  ['coffee']
```

Milestone-6: Model Deployment

Activity-1: Save the Model

Saving the best-performing model is a crucial step that allows for its reuse without the need for retraining every time the application is run.

- **Purpose:** To persist the trained machine learning model to disk. This is essential for deployment, as it allows the web application to load the model and make predictions efficiently. It also ensures that the exact model used for evaluation is the one deployed.
- **Method:** Python's pickle module is commonly used for serializing and deserializing Python objects. The best model (identified in Milestone 5, likely Random Forest) is saved to a .pkl file (e.g., opticrops_model.pkl).

```
import pickle
pickle.dump(model, open(r'C:\Users\psiri\Desktop\OptiCrop\model.pkl', 'wb'))
```

Activity-2: Integrate with Web Framework

In this section, we will be building a web application that is integrated with the model we built. A UI is provided for users where they have to enter the values for predictions. The entered values are given to the saved model, and the prediction is showcased on the UI.

This Section has the following tasks:

- Building HTML Pages
- Building a server-side script
- Run a web application

Activity 2.1: Building HTML Pages:

- home.html
- about.html
- findyourcrop.html

Activity-2.2: Build Python code:
Import the libraries

```
import numpy as np
from flask import Flask, request, render_template
import pickle

app = Flask(__name__)

path = r"C:\Users\psiri\Desktop\OptiCrop\\"
model = pickle.load(open(path + 'model.pkl', 'rb'))
```

Load the saved model. Importing the Flask module in the project is mandatory. An object of the Flask class is our WSGI application. Flask constructor takes the name of the current module (name) as an argument.

```
@app.route('/')
def home():
    return render_template('home.html')

@app.route('/about')
def about():
    return render_template('about.html')

@app.route('/findyourcrop')
def findyourcrop():
    return render_template('findyourcrop.html')
```

Here we will be using a declared constructor to route to the HTML page that we have created earlier. In the above example, the '/' URL is bound with the index.html function. Hence, when the home page of the web server is opened in the browser, the HTML page will be rendered. Whenever you enter the values from the HTML page, the values can be retrieved using the POST Method. Retrieves the value from the UI.

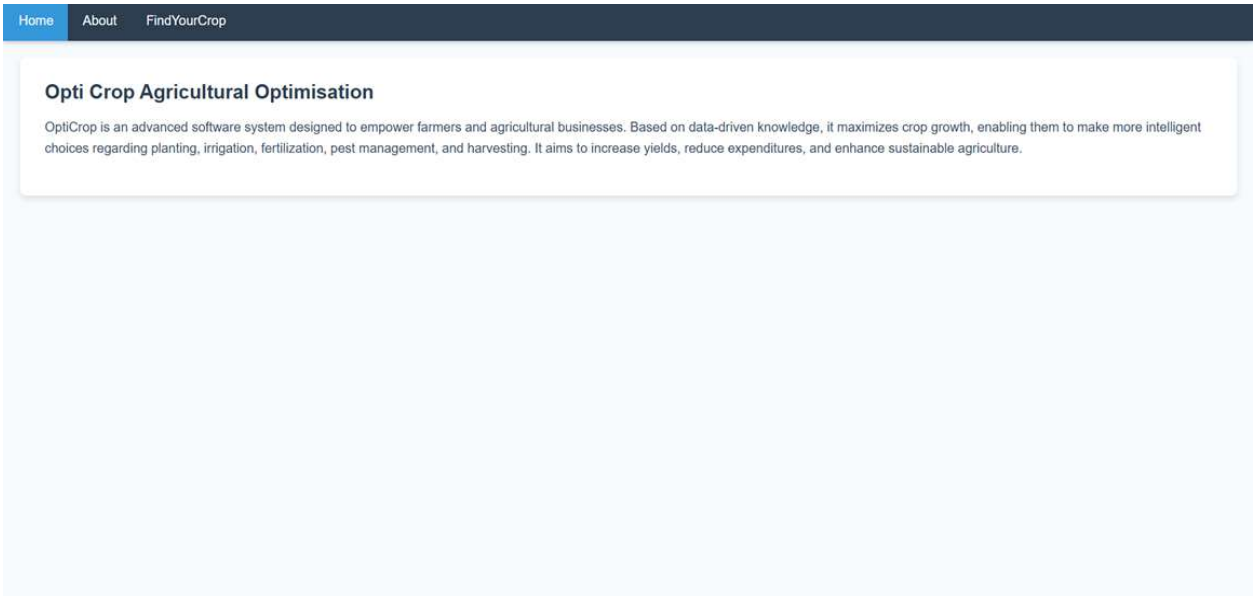
```python
import numpy as np
from flask import Flask, request, render_template
import pickle

app = Flask(__name__)

path = r"C:\Users\psiri\Desktop\OptiCrop\\"
model = pickle.load(open(path + 'model.pkl', 'rb'))

@app.route('/')
def home():
    return render_template('home.html')

@app.route('/about')
def about():
    return render_template('about.html')

@app.route('/findyourcrop')
def findyourcrop():
    return render_template('findyourcrop.html')

@app.route('/predict', methods=['POST'])
def predict():
    int_features = [float(x) for x in request.form.values()]
    features = [np.array(int_features)]
    prediction = model.predict(features)

    output = prediction[0]
    return render_template('findyourcrop.html', prediction_text=f"Best crop for given conditions is {output}")

if __name__ == "__main__":
    app.run(debug=True, use_reloader=False)
```

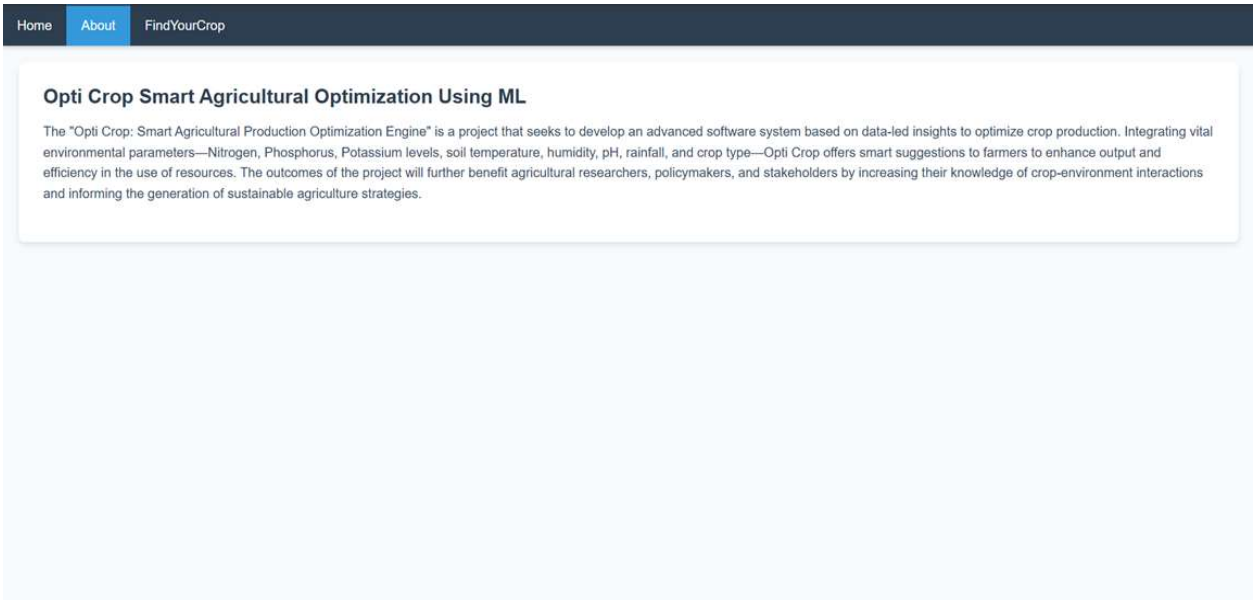
Activity-3: Run the Web App

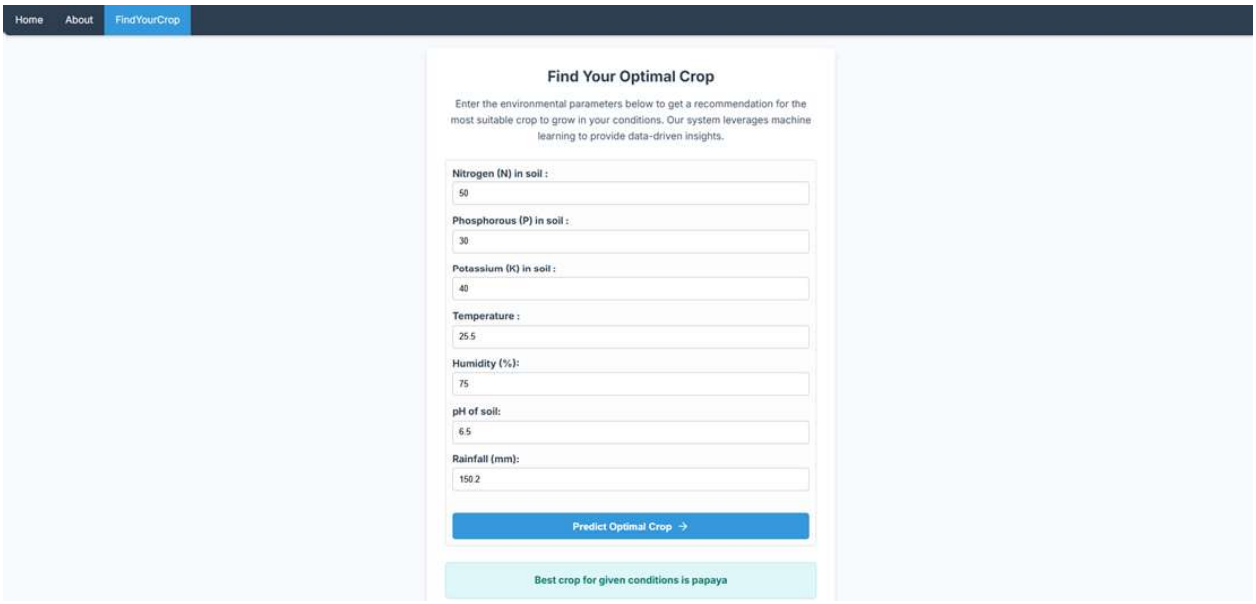
This activity describes the simple steps required to launch the developed web application locally for testing and demonstration.

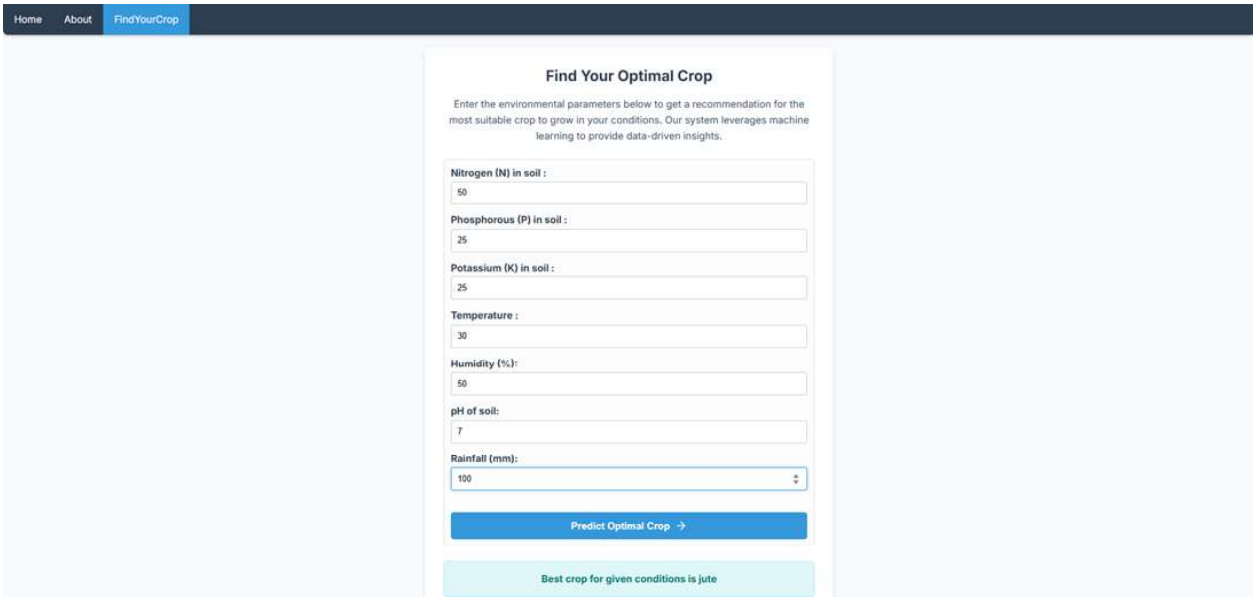
- **Open Anaconda Prompt:** Begin by opening the Anaconda Prompt (or your preferred command-line interface).
- **Navigate to Project Folder:** Use the cd command to navigate to the directory where your app.py script and other project files are located.
- **Run Flask Application:** Execute the command python app.py. This command starts the Flask development server.

- **Access in Browser:** Once the server is running, open a web browser and navigate to the specified local URL, typically http://127.0.0.1:5000. This will load the web application's home page (index.html).
- **Interact and Test:** From the web page, users can input the required agricultural parameters, click the "Predict" or "Submit" button, and observe the model's recommendations or predictions displayed on the UI.









Milestone-7: Project Demonstration & Documentation

This final milestone focuses on presenting the completed project and providing comprehensive documentation for future reference and reproducibility.

Activity-1: Record Explanation Video

A project demonstration video is an effective way to showcase the functionality, user interface, and overall flow of the OptiCrop application.

- **Purpose:** To provide a visual and auditory walkthrough of the entire project, from data input to prediction output. This is particularly useful for stakeholders, evaluators, or future users who need a quick understanding of the system.
- **Content:** The video typically covers:
    - An overview of the problem OptiCrop addresses.
    - A demonstration of the user interface, showing how to input data.
    - Live examples of predictions or recommendations generated by the system.
    - A brief explanation of the underlying machine learning process and its benefits.
    - Highlighting key features and the value proposition of OptiCrop.

Activity-2: Step-by-step Documentation

Comprehensive documentation is vital for the long-term viability, maintainability, and understanding of any software project.

- **Purpose:** This document (the one you are currently reading) serves as the official project report. It details every step of the project development, from problem definition to deployment.
- **Content:** The documentation includes:
    - **Problem Understanding:** Detailed explanation of the business problem, requirements, literature review, and social/business impact.
    - **Data Aspects:** Information on data collection, sources, and all data preparation steps (missing values, outliers, encoding, scaling, balancing, splitting).
    - **Exploratory Data Analysis:** Insights gained from descriptive statistics and visual analyses (univariate, bivariate, multivariate).
    - **Model Development:** Description of algorithms used, training procedures, and testing methodologies.
    - **Performance Evaluation:** Comparison of models and results from cross-validation.
    - **Deployment Details:** How the model is saved and integrated into the web framework, along with instructions for running the application.

- **References:** A list of books and websites consulted during the project.
- **Appendix:** Supplementary materials such as UI screenshots, code snippets, and the Jupyter Notebook for model training.

- **Format:** The document is structured logically with clear headings and subheadings, making it easy to navigate and understand. It adheres to a standard project report template to ensure consistency and professionalism.

Conclusion

The OptiCrop Agricultural Optimisation project successfully demonstrates how machine learning can revolutionize traditional agricultural practices. By providing farmers with intelligent, data-driven tools, the platform empowers them to make more informed decisions regarding crop planning, resource management, and pest control. This leads to significant benefits, including optimized crop yields, reduced operational waste (e.g., water, fertilizers), and ultimately, improved profitability. Furthermore, OptiCrop promotes environmentally friendly farming methods, contributing to sustainable agriculture and enhanced food security for the future. The successful development and deployment of this system underscore the transformative potential of AI in addressing contemporary challenges in the agricultural sector.

References

Books:

- **Aurélien Géron – Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow:** A foundational text for practical machine learning, covering various algorithms and their implementations.
- **Sebastian Raschka & Vahid Mirjalili – Python Machine Learning:** A comprehensive guide to machine learning concepts and applications using Python libraries.
- **Qin Zhang – Precision Agriculture Technology for Crop Farming:** A specialized book focusing on the technological advancements and applications in precision agriculture.

Websites

- **https://www.kaggle.com:** A popular platform for data science and machine learning competitions, providing access to numerous datasets and community-contributed notebooks.
- **https://scikit-learn.org:** The official documentation for Scikit-learn, a widely used machine learning library in Python, offering guides, examples, and API references.
- **https://towardsdatascience.com:** A publication featuring articles and tutorials on data science, machine learning, and AI, often providing practical insights and code examples.
- **https://cropwatch.unl.edu:** A resource from the University of Nebraska-Lincoln, providing current crop and weather information, useful for agricultural data collection.
- **https://www.agriculture.com:** A website offering news, insights, and resources related to farming and agricultural practices.

Appendix

The Appendix contains supplementary materials that provide additional detail and evidence for the project's development and results.

- **Screenshots of UI and predictions:** Visual captures of the OptiCrop web application's user interface, demonstrating its design, input forms, and how the model's predictions or recommendations are displayed to the user. This helps in understanding the user experience and the final output of the system.
- **Snippets of Flask API and HTML:** Key code excerpts from the app.py Flask application (e.g., routing, model loading, prediction logic) and the HTML templates (e.g., input forms, result display sections). These snippets illustrate the technical implementation of the web integration.
- **Model training notebook (Jupyter):** The complete Jupyter Notebook file (.ipynb) used for developing and training the machine learning models. This includes all steps from data loading and preprocessing to EDA, model selection, training, evaluation, and hyperparameter tuning. It serves as a fully reproducible record of the machine learning pipeline.