National College of Ireland

Configuration Manual

MSc Research Project
MSc in Data Analytics

Abhay Udaykumar Patil
Student ID: 23340304

School of Computing
National College of Ireland

Supervisor: Dr. David Hamill

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Abhay Udaykumar Patil

Student ID: 23340304

Programme: MSc. Data Analytics **Year:** 2025-2026

Module: Research Practicum. (MSCDAD_A_JAN25I)

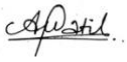
Lecturer: Dr. David Hamill

Submission Due Date: December 11, 2025.

Project Title: Machine Learning Model System for Multi-Crop Recommendation Using Integrated Soil and Weather Insights for Western Maharashtra Region.

Word Count: 1905 **Page Count:** 28

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:  (Abhay Udaykumar Patil)

Date: 10 / 12 / 2025.

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Machine Learning Model System for Multi-Crop Recommendation Using Integrated Soil and Weather Insights for Western Maharashtra Region.

Name: - Abhay Udaykumar Patil
Student ID: - 23340304

1 Introduction

This Configuration Manual is an elaborate description of hardware, software, and system settings that are needed in the implementation of the Crop Recommendation System (CRS). It describes the context under which the research was done, the preparation and pre-processing of the dataset, and the description of the practical workflow used to construct, test, and interpret machine learning models. This document serves as a guideline that will be used in the reproduction of the system into any other compatible computing environment.
The main aim of the CRS is to forecast the most appropriate crop under the circumstance of those environmental and soil nutrient conditions in five major districts of Western Maharashtra. This is done through monitored classification models with feature engineering and cross-validation, ROC analysis and SHAP-based interpretability strategies. The configuration is recorded in the following sections that permit this research workflow.

2 Hardware and Software Requirements

2.1 Hardware Configuration

This study was carried out on a personal laptop with the following settings:
Processor: 2.2 GHz 6-Core Intel Core i7.
Graphics: Radeon Pro 555X 4GB, Intel UHD Graphics 630 1536MB.
RAM Specification: 16 GB 2400 MHz DDR4

This setup is sufficient when training ensemble models, creating visualisations, and carrying out SHAP-based explainability. An increased RAM will decrease the time taken to compute large SHAP plots.

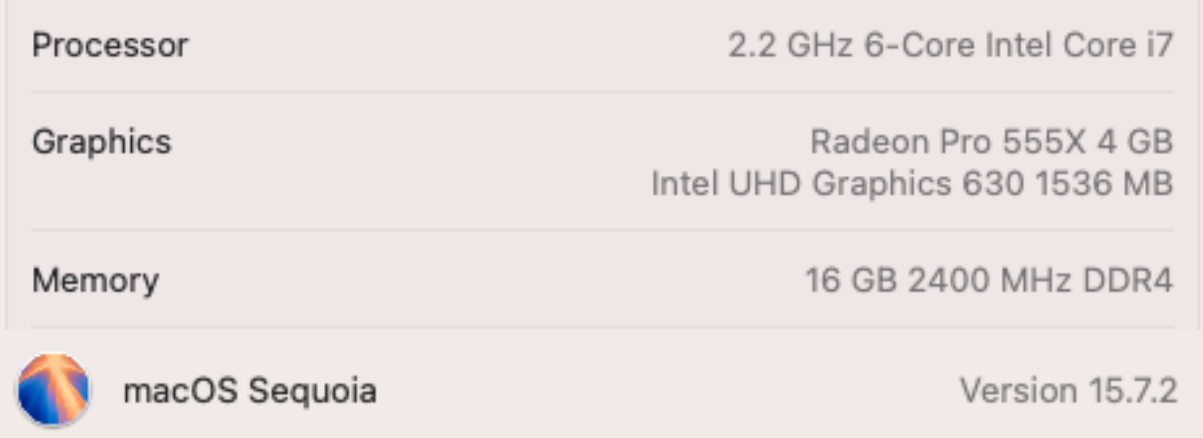


Figure 1: System Configuration

2.2 Software Configuration

The implementation was anticipated to be done on a prepared software environment that would provide smooth execution of data processing, modelling, and visualisation operations. The system is built on the python scientific computing environment and interactive notebooks.

- **Operating System:** macOS Sequoia Version 15.7.2
- **Development Environment:** Visual Studio Code with Jupyter extension
- **Programming Language:** Python 3.12.2

Such configuration is adaptable and enables the reproduction of the research in different operating systems with slight modifications.

3 Packages & Libraries

The following Python libraries were used:

- **Data Processing:** pandas, numpy
- **Visualisation:** matplotlib, seaborn
- **Machine Learning:** scikit-learn, RandomForestClassifier, DecisionTreeClassifier, LogisticRegression, accuracy_score, classification_report, confusion_matrix, train_test_split, cross_val_score.
- **Explainability:** shap
- **Statistical Tools:** scipy.stats (entropy)
- **Model Persistence:** joblib
- **Warnings Management:** warnings

The following libraries can be installed by running the following in your terminal or command prompt:
pip install pandas numpy matplotlib seaborn scikit-learn shap scipy joblib

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc
import shap
import joblib
from scipy.stats import entropy
import warnings
warnings.filterwarnings("ignore")
pd.set_option('display.max_columns', None)
pd.set_option('display.width', 200)
print(" Required Libraries are imported successfully.")
```

Figure 2: Imported Libraries and Modules for the Project

4 Dataset

The dataset used in this analysis can be downloaded from the following Kaggle: https://www.kaggle.com/datasets/sanchitagholap/crop-and-fertilizer-dataset-for-westernmaharashtra. After downloading the dataset, it has to be extracted onto the preferred folder in which you plan to code. The extraction must be in such a way that the extracted file is readily accessible from the environment where the coding is being performed because it will be the main dataset under analysis and model training. Changes the name of downloaded file to “Raw_data.csv” for easier understanding.
The code in Figure 3 shows how the dataset can be loaded into a Pandas DataFrame. It also gives an overview of the dataset after loading, showing the important features. These are the major features which, in this project, data pre-processing and machine learning modelling will be done.

```
df = pd.read_csv("Raw_data.csv")

print("\n Dataset has been loaded successfully!")
print(f"Shape: {df.shape[0]} rows × {df.shape[1]} columns")
print("\nColumns:", df.columns.tolist())
print("\nFirst 5 rows:\n", df.head())
print("\nDataset Info:")
print(df.info())
```

```
Columns: ['District_Name', 'Soil_color', 'Nitrogen', 'Phosphorus', 'Potassium', 'pH', 'Rainfall', 'Temperature', 'Crop', 'Fertilizer', 'Link']

First 5 rows:
   District_Name Soil_color  Nitrogen  Phosphorus  Potassium   pH  Rainfall  Temperature      Crop Fertilizer                        Link
0      Kolhapur      Black        75          50        100  6.5      1000           20  Sugarcane       Urea  https://youtu.be/2t5Am0xLTOo
1      Kolhapur      Black        80          50        100  6.5      1000           20  Sugarcane       Urea  https://youtu.be/2t5Am0xLTOo
2      Kolhapur      Black        85          50        100  6.5      1000           20  Sugarcane       Urea  https://youtu.be/2t5Am0xLTOo
3      Kolhapur      Black        90          50        100  6.5      1000           20  Sugarcane       Urea  https://youtu.be/2t5Am0xLTOo
4      Kolhapur      Black        95          50        100  6.5      1000           20  Sugarcane       Urea  https://youtu.be/2t5Am0xLTOo

Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4513 entries, 0 to 4512
Data columns (total 11 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   District_Name  4513 non-null   object
 1   Soil_color     4513 non-null   object
 2   Nitrogen       4513 non-null   int64
 3   Phosphorus     4513 non-null   int64
 4   Potassium      4513 non-null   int64
 5   pH             4513 non-null   float64
 6   Rainfall       4513 non-null   int64
 7   Temperature    4513 non-null   int64
 8   Crop           4513 non-null   object
 9   Fertilizer     4513 non-null   object
 10  Link           4513 non-null   object
dtypes: float64(1), int64(5), object(5)
memory usage: 388.0+ KB
None
```

Figure 3: Dataset Preview After Loading

This dataset comprises 4,513 rows and 11 columns, as shown in Figure 4 below, which summarizes the data types for each column. Also, checks for missing values and removal of unwanted feature columns .

```python
missing = df.isnull().sum()
missing_cols = missing[missing > 0]

print("\n Checking for Missing Values...\n")

if missing_cols.empty:
    print(" No missing values found.")
else:
    print(" Missing values detected:\n", missing_cols)
    df = df.dropna()
    print(f" Dropped missing rows. New shape: {df.shape}")

if "Link" in df.columns:
    df.drop(columns=["Link"], inplace=True)
    print(" Dropped 'Link' column.")

df = df.drop(columns=['Fertilizer'])
print(" Dropped 'Fertilizer' column.")
print("\n Data is cleaned.")
```

✓ 0.0s

```
Checking for Missing Values...

No missing values found.
Dropped 'Link' column.
Dropped 'Fertilizer' column.

Data is cleaned.
```

Figure 4: Missing Values, and Dropping unwanted columns.

5 Exploratory Data Analysis

The aim of EDA was to gain the insight on crop distribution, nutrient patterns, and the patterns across districts. This included:

- Statistical Summary of Dataset.
- Crop distribution bar chart
- District plot of crop count.
- Overview measurements of quantitative variables.
- Boxplots as a way of outliers spotting.
- Histograms to examine distributions of features.
- Correlation heatmap to identify the connections between nutrients, rainfall, and temperature.
- The visualisations can give information about the agronomic and regional framework of the data.

```
print("\n Summary Statistics:")
print(df.describe(include='all').transpose())
```

```
 Summary Statistics:
                count unique        top  freq       mean         std    min    25%    50%     75%     max
District_Name    4513      5   Kolhapur  1430        NaN         NaN    NaN    NaN    NaN     NaN     NaN
Soil_color       4513      7      Black  2260        NaN         NaN    NaN    NaN    NaN     NaN     NaN
Nitrogen       4513.0    NaN        NaN   NaN  95.409927   38.060648   20.0   60.0  105.0   125.0   150.0
Phosphorus     4513.0    NaN        NaN   NaN  54.341901   16.551991   10.0   40.0   55.0    65.0    90.0
Potassium      4513.0    NaN        NaN   NaN   63.59517   35.691911    5.0   40.0   55.0    75.0   150.0
pH             4513.0    NaN        NaN   NaN   6.715267    0.625198    5.5    6.0    6.5     7.0     8.5
Rainfall       4513.0    NaN        NaN   NaN  819.18901  251.730813  300.0  600.0  800.0  1000.0  1700.0
Temperature    4513.0    NaN        NaN   NaN  25.915134    5.897328   10.0   20.0   25.0    30.0    40.0
Crop             4513     16  Sugarcane  1010        NaN         NaN    NaN    NaN    NaN     NaN     NaN
District_Name: 5 unique values
Soil_color: 7 unique values
Crop: 16 unique values
```

Figure 5: Statistical Summary of Dataset.

```
for col in ["District_Name", "Soil_color", "Crop"]:
    if col in df.columns:
        print(f"{col}: {df[col].nunique()} unique values")

plt.figure(figsize=(10, 5))
sns.countplot(data=df, x="Crop",
              order=df["Crop"].value_counts().index,
              palette="viridis")
plt.title(" Crop Distribution")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

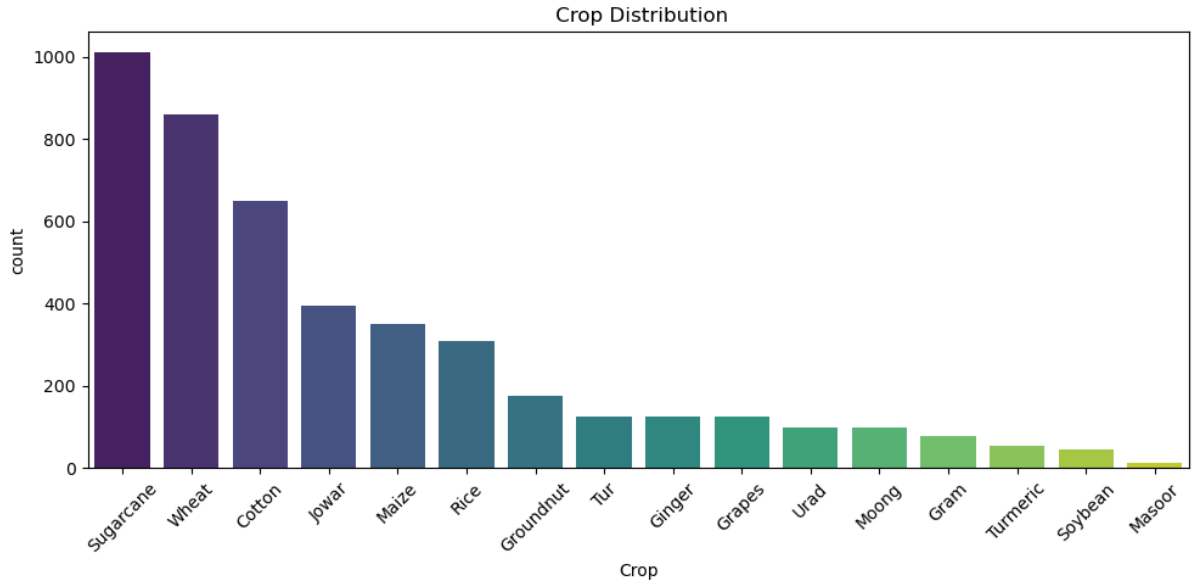


Figure 6: Crop distribution bar chart

```
plt.figure(figsize=(14, 6))
sns.countplot(data=df, x="District_Name", hue="Crop",
              order=sorted(df["District_Name"].unique()),
              palette="tab10")
plt.title(" District-wise Crop Distribution")
plt.xticks(rotation=45)
plt.legend(bbox_to_anchor=(1, 1))
plt.tight_layout()
plt.show()
```

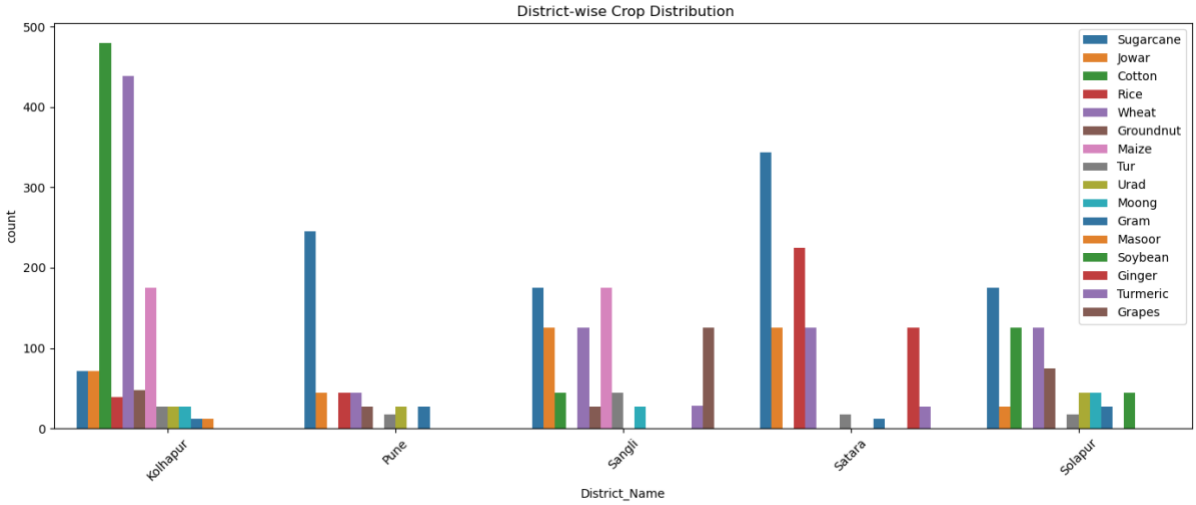


Figure 7: District-wise plot of crop count.

```
numeric_cols = ["Nitrogen", "Phosphorus", "Potassium", "pH", "Rainfall", "Temperature"]

for col in numeric_cols:
    plt.figure(figsize=(12, 5))
    sns.barplot(data=df, x="District_Name", y=col,
                order=sorted(df["District_Name"].unique()),
                palette="crest", ci=None)
    plt.title(f" Average {col} by District")
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```

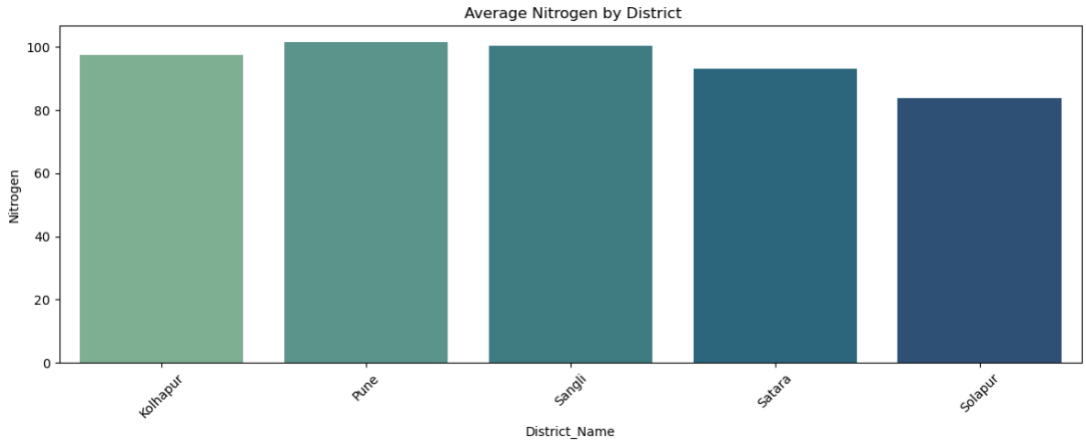


Figure 8: Average Nitrogen by District.

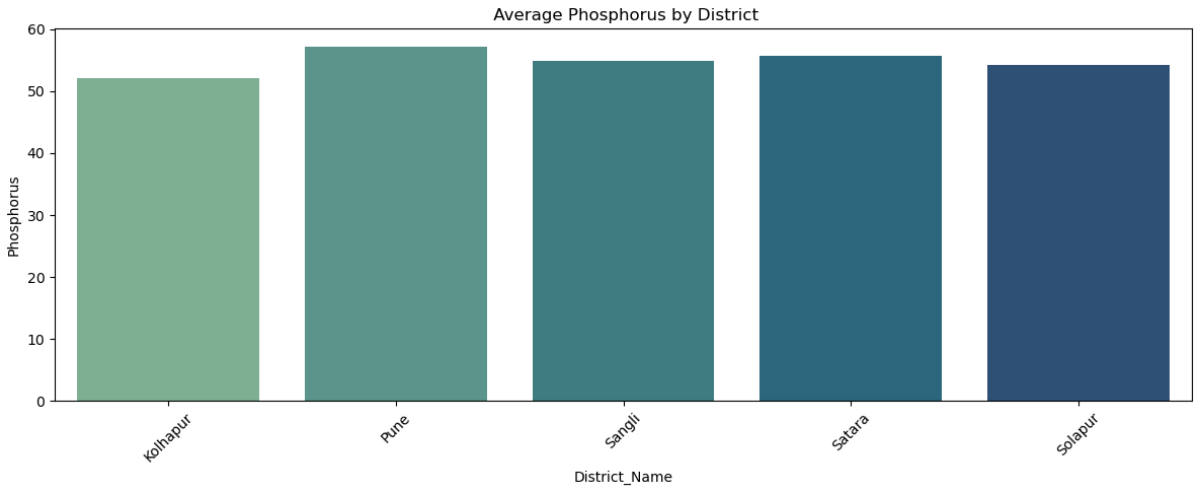


Figure 9: Average Phosphorus by District.

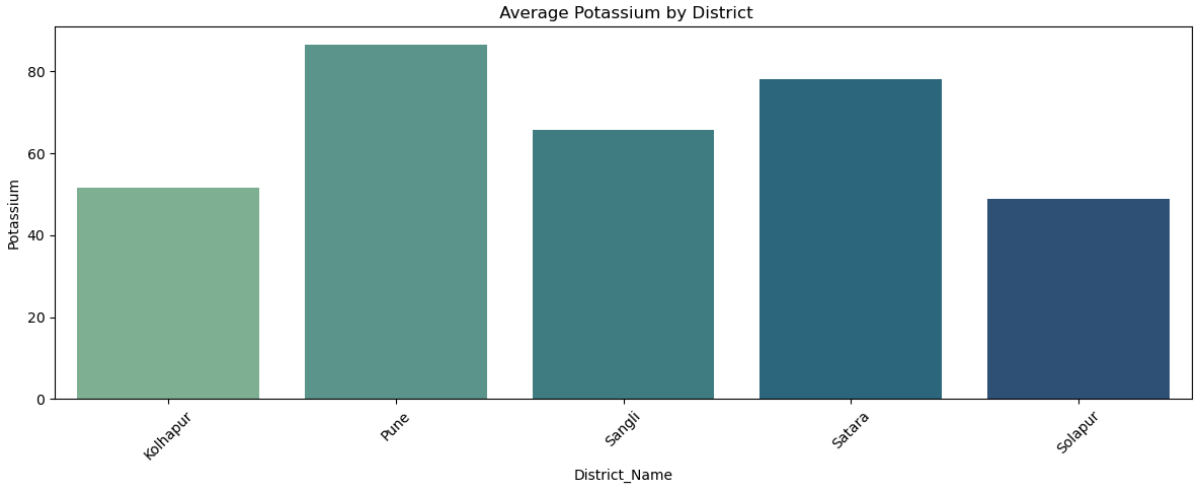


Figure 10: Average Potassium by District.

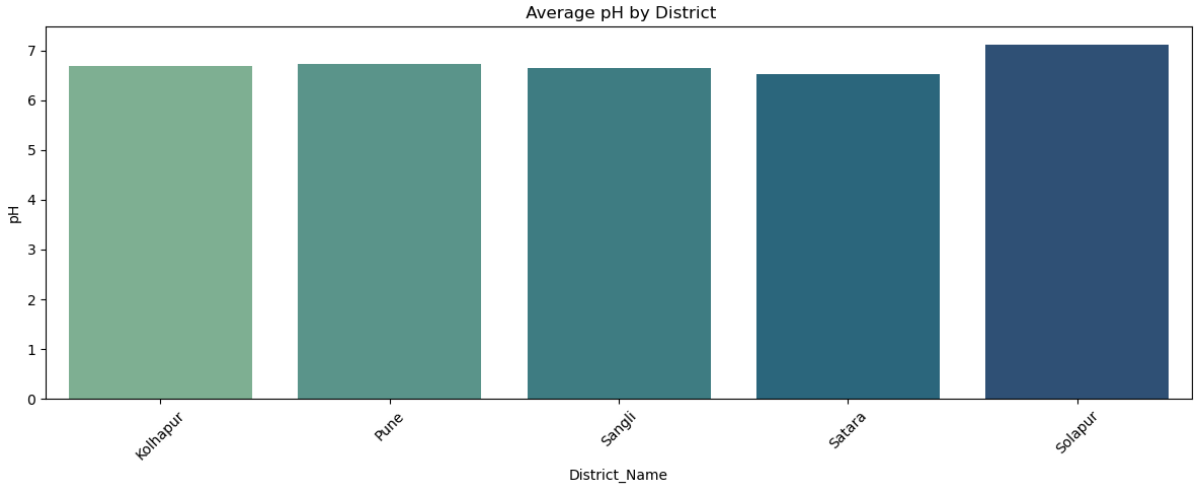


Figure 11: Average pH by District.

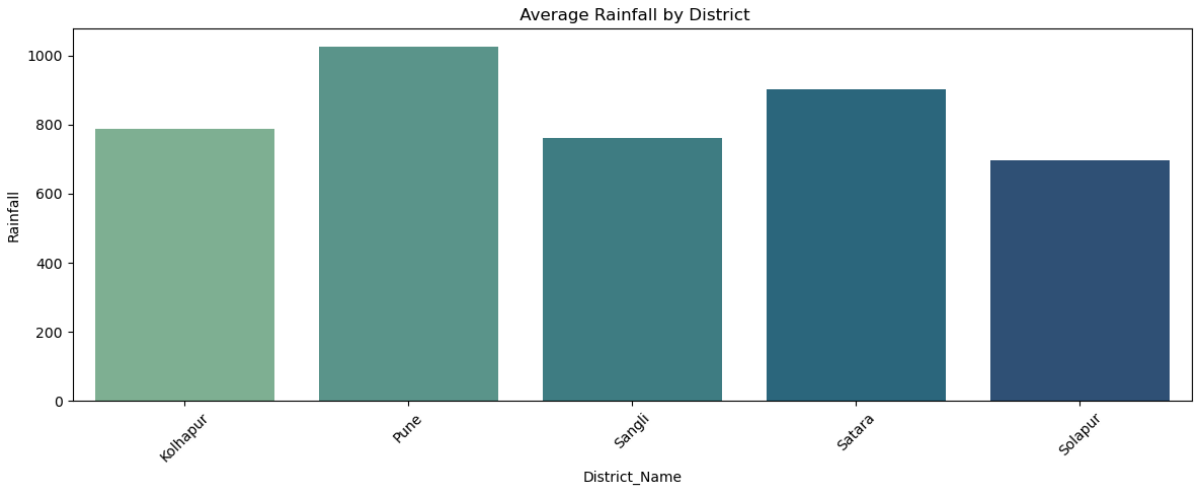


Figure 12: Average Rainfall by District.

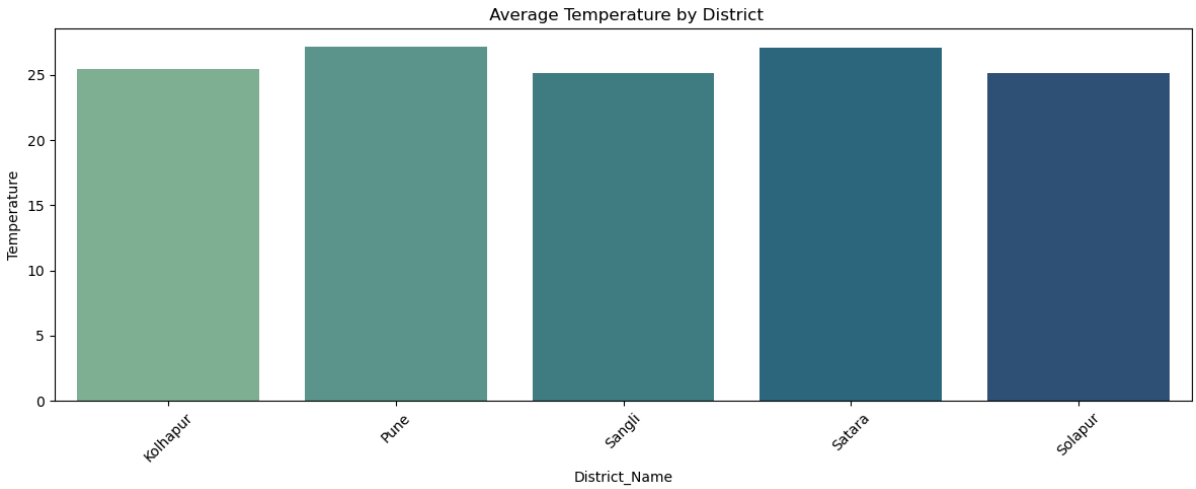


Figure 13: Average Temperature by District.

```
plt.figure(figsize=(10, 6))
sns.heatmap(df[numeric_cols].corr(), annot=True,
            cmap="coolwarm", fmt=".2f")
plt.title(" Correlation Heatmap")
plt.tight_layout()
plt.show()
```

Figure 14: Plotting the Correlation Heatmap

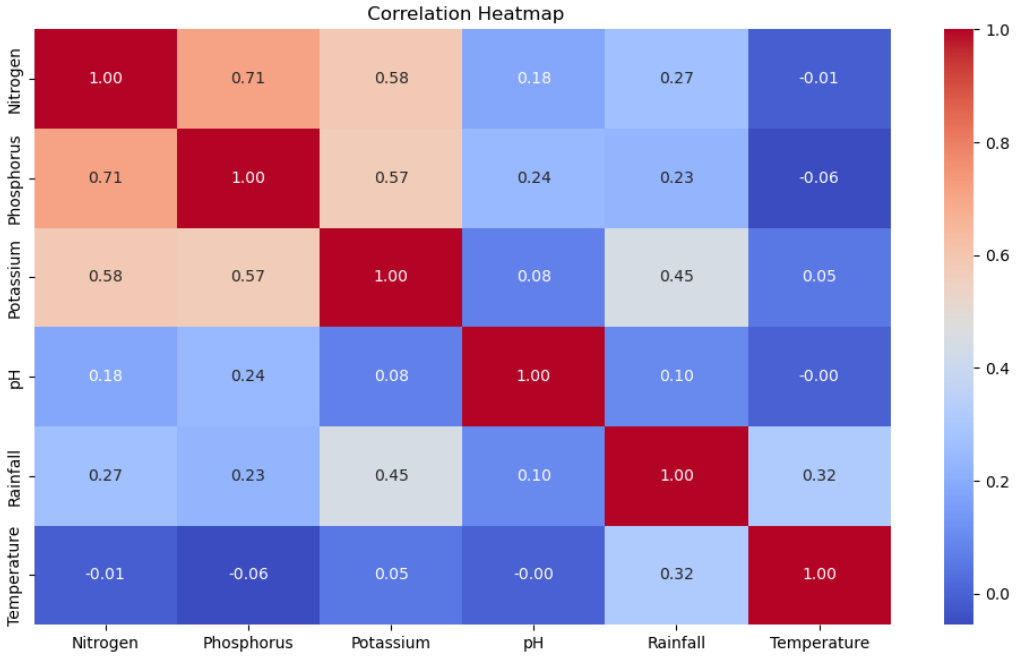


Figure 15: Correlation Heatmap Plot.

```
plt.figure(figsize=(12, 5))
sns.boxplot(data=df[numeric_cols], palette="coolwarm")
plt.title("Outlier Detection Across All Numeric Features")
plt.xticks(rotation=45)
plt.show()
```

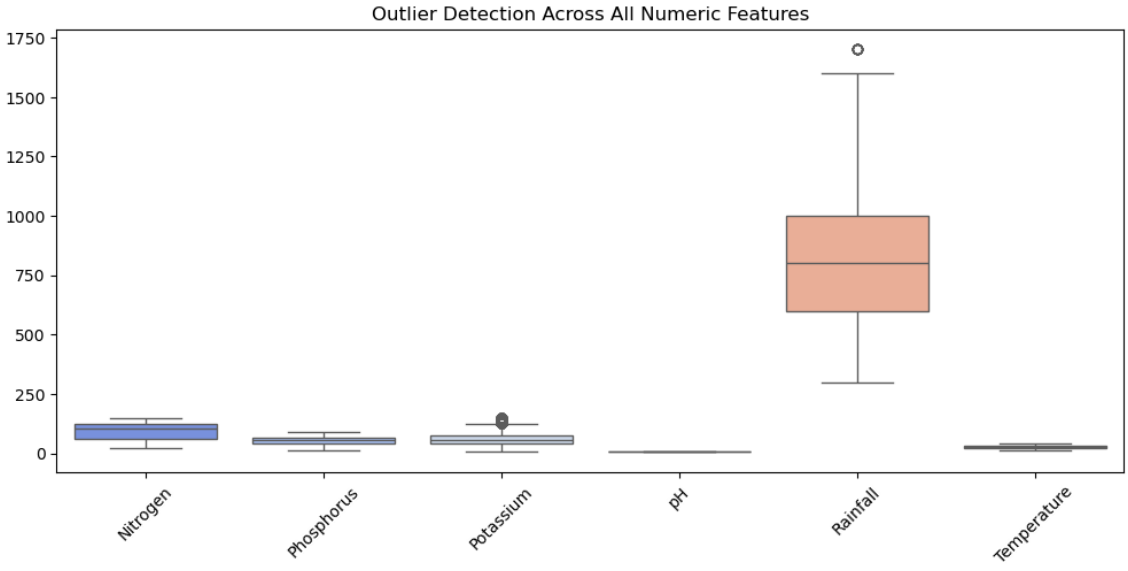


Figure 16: Outlier Detection Across All Numeric Features.

```
numeric_cols = df.select_dtypes(include=['int64', 'float64']).columns.tolist()
sns.pairplot(df[numeric_cols + ['Crop']], hue='Crop')
plt.suptitle('Pairplot of Numeric Features Colored by Crop', y=1.02)
plt.show()
```

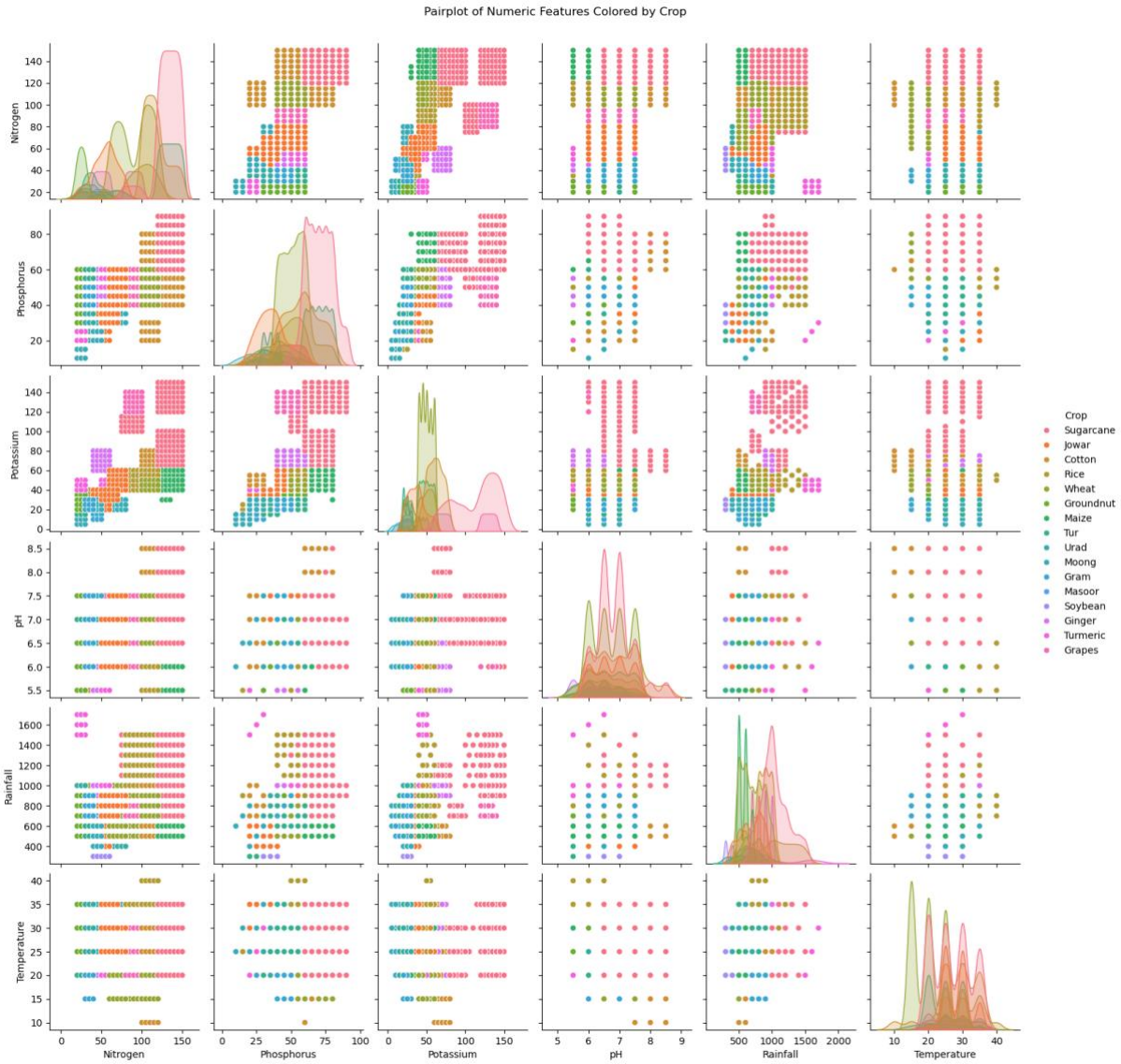


Figure 17: Pairplot of Numeric Features Colored by Crop.

```
plt.figure(figsize=(15, 12))
for i, col in enumerate(numeric_cols):
    plt.subplot((len(numeric_cols) + 3) // 4, 4, i+1)
    sns.histplot(df[col], kde=True)
    plt.title(f'Distribution of {col}')
plt.tight_layout()
plt.show()
```

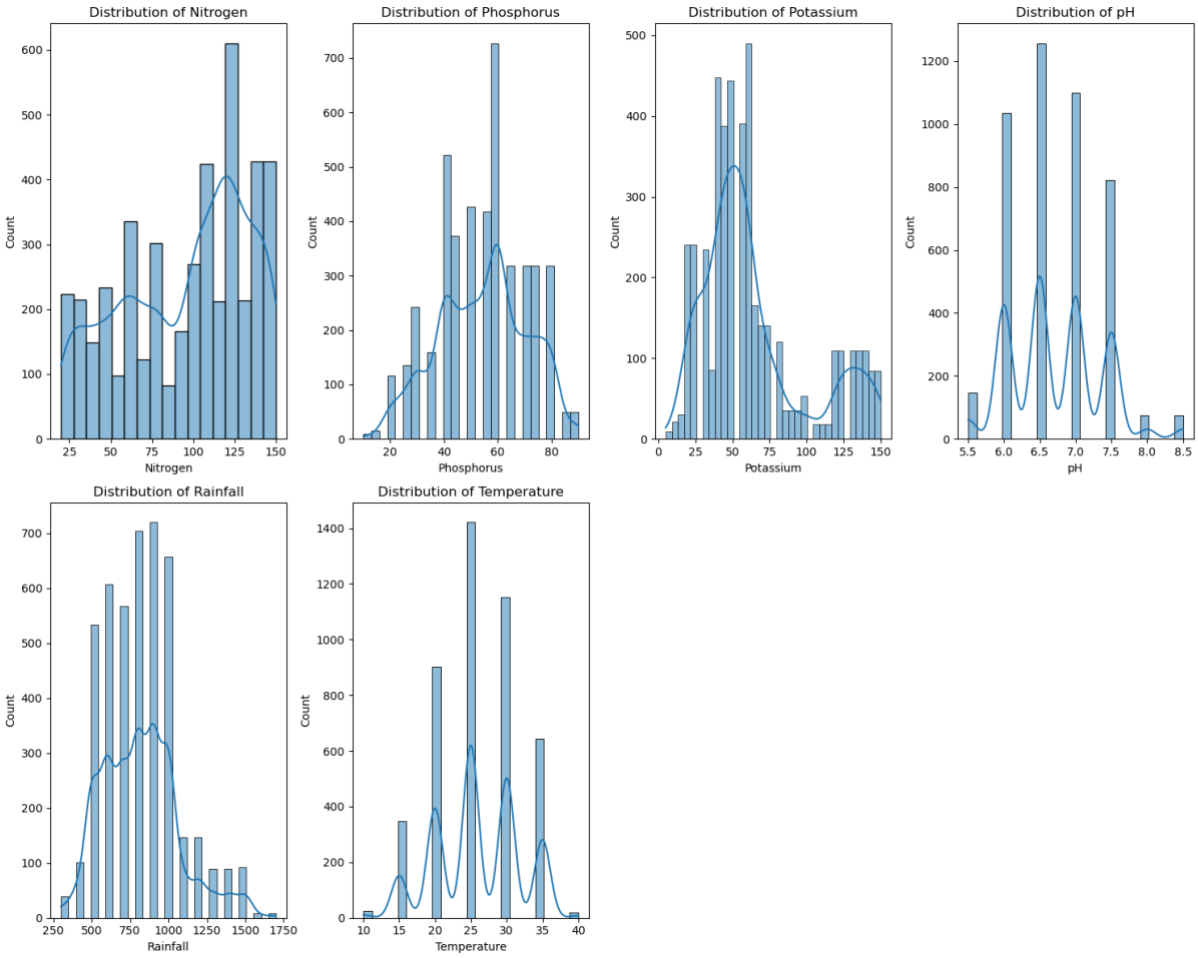


Figure 18: Distribution of the key features.

6 Data Preparation

6.1 Feature Engineering.

To enhance prediction capability two designed ratio features were made:

- N P ratio = Nitrogen/ (Phosphorus + 1)
- K_N_ratio = Potassium / (Nitrogen + 1)

These ratios represent nutrient balance relationship which affect the suitability of crops.

```
df["N_P_ratio"] = df["Nitrogen"] / (df["Phosphorus"] + 1)
df["K_N_ratio"] = df["Potassium"] / (df["Nitrogen"] + 1)

print("New feature columns added: N_P_ratio, K_N_ratio")
```

```
New feature columns added: N_P_ratio, K_N_ratio
```

Figure 19: Feature Engineering.

6.2 Defining Features & Target columns.

```
X = df.drop(columns=["Crop",])
y = df["Crop"]

print("\nFeatures:", X.columns.tolist())
```

```
Features: ['District_Name', 'Soil_color', 'Nitrogen', 'Phosphorus', 'Potassium', 'pH', 'Rainfall', 'Temperature', 'N_P_ratio', 'K_N_ratio']
```

Figure 20: Defining Features & Target columns

6.3 Train–Test Split & Label Encoding.

A traintest split of 80:20 in the stratified sampling implies that all the crop classes are represented in equal proportion in both subsets.

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)

print("\n Train/Test Split Complete")
print(" Train:", X_train.shape, "Test:", X_test.shape)

cat_cols = X_train.select_dtypes(include=["object"]).columns.tolist()
label_encoders = {}
cat_default_encoded = {}

for col in cat_cols:
    le = LabelEncoder()
    X_train[col] = le.fit_transform(X_train[col])
    X_test[col] = le.transform(X_test[col])
    label_encoders[col] = le
    cat_default_encoded[col] = X_train[col].mode()[0]
    print(f" Encoded: {col}")

crop_le = LabelEncoder()
y_train = crop_le.fit_transform(y_train)
y_test = crop_le.transform(y_test)
print("\n Target 'Crop' Encoded")
```

```
Train/Test Split Complete
Train: (3610, 10) Test: (903, 10)
Encoded: District_Name
Encoded: Soil_color

Target 'Crop' Encoded
```

Figure 21: Train–Test Split & Label Encoding

7 Machine Learning Models

The machine learning models developed to classify the most appropriate crop under a set of agricultural conditions were based on soil nutrient values, environmental attributes as well as engineered ratio features. Modelling phase involved the Logistic Regression, Decision Tree as well as the Random Forest classifier, all of which were trained using the refined features set. The standard classification measures like Accuracy, Precision, Recall and F1-score were used to rate the performance of these models and the cross-validation scores were used to rate the generalisation capability

7.1 Implementation of ML Model module.

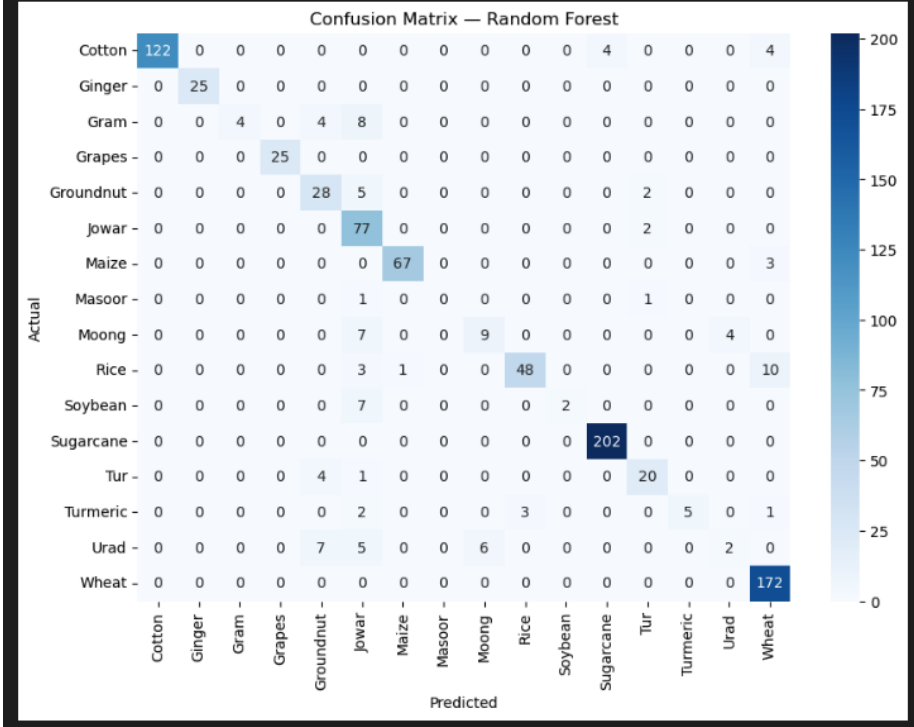
```
models = {
    "Random Forest": RandomForestClassifier(n_estimators=100, max_depth=8, min_samples_leaf=40,random_state=42),
    "Decision Tree": DecisionTreeClassifier(max_depth=10, min_samples_leaf=40,random_state=42),
    "Logistic Regression": LogisticRegression(max_iter=2000, random_state=42)
}

all_results = []

for name, model in models.items():
    print(f" Evaluating Model: {name}")


    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    acc = accuracy_score(y_test, y_pred)
    cv = cross_val_score(model, X_train, y_train, cv=5).mean()
    report = classification_report(y_test, y_pred,
                                   target_names=crop_le.classes_,
                                   output_dict=True)
    conf_mat = confusion_matrix(y_test, y_pred)

    all_results.append({
        "Model": name,
        "Accuracy": acc,
        "CrossVal": cv,
        "Precision (macro)": report["macro avg"]["precision"],
        "Recall (macro)": report["macro avg"]["recall"],
        "F1-score (macro)": report["macro avg"]["f1-score"]
    })

    print(f"\n Accuracy: {acc:.4f}")
    print(f" CV Mean: {cv:.4f}")
    print(f" Precision: {report['macro avg']['precision']:.4f}")
    print(f" Recall:    {report['macro avg']['recall']:.4f}")
    print(f" F1-score:  {report['macro avg']['f1-score']:.4f}")
```

```
    plt.figure(figsize=(10, 7))
    sns.heatmap(conf_mat, annot=True, fmt="d", cmap="Blues",
                xticklabels=crop_le.classes_, yticklabels=crop_le.classes_)
    plt.title(f"Confusion Matrix - {name}")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()

    print("\n Classification Report:")
    print(classification_report(y_test, y_pred, target_names=crop_le.classes_))

results_df = pd.DataFrame(all_results).sort_values("Accuracy", ascending=False)

print("\n Model Performance Comparison:\n")
print(results_df)

best_model_name = results_df.iloc[0]["Model"]
best_model = models[best_model_name]

print(f"\n BEST MODEL SELECTED: {best_model_name}")
```

Figure 22: Training and Evaluation of ML Models.

7.2 Random Forest Model Results.

```
Evaluating Model: Random Forest

Accuracy: 0.8948
CV Mean: 0.8762
Precision: 0.8038
Recall:    0.6701
F1-score:  0.6934
```

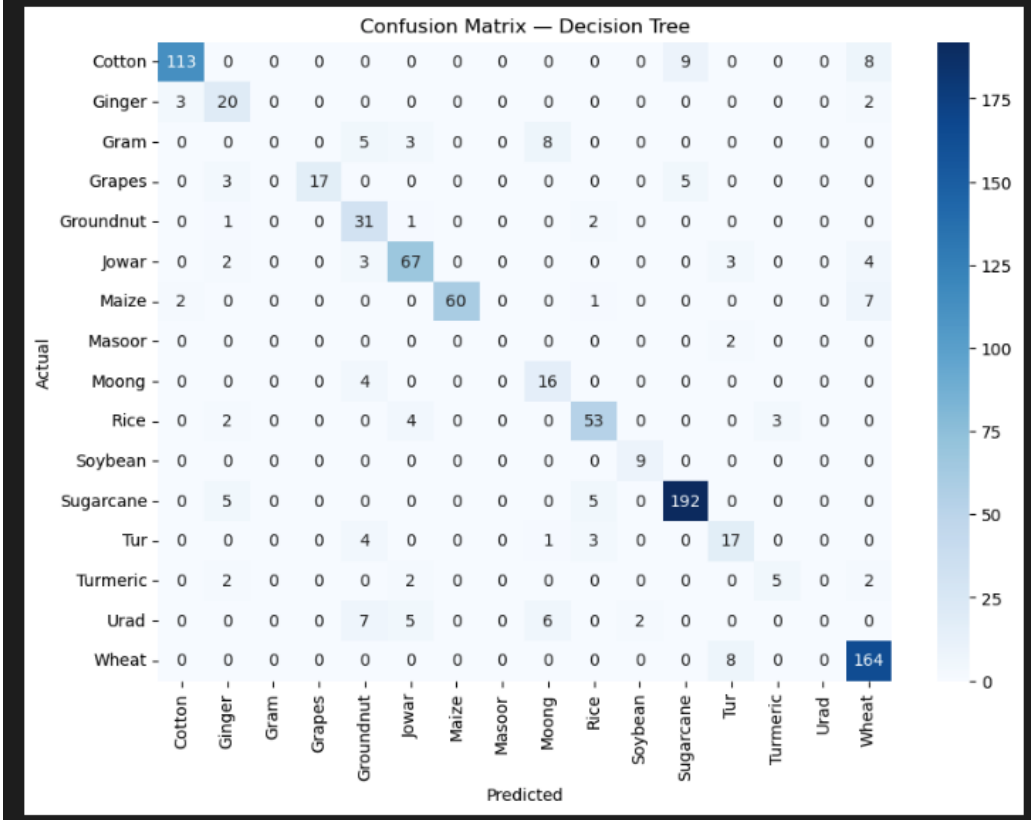


Classification Report:	precision	recall	f1-score	support
Cotton	1.00	0.94	0.97	130
Ginger	1.00	1.00	1.00	25
Gram	1.00	0.25	0.40	16
Grapes	1.00	1.00	1.00	25
Groundnut	0.65	0.80	0.72	35
Jowar	0.66	0.97	0.79	79
Maize	0.99	0.96	0.97	70
Masoor	0.00	0.00	0.00	2
Moong	0.60	0.45	0.51	20
Rice	0.94	0.77	0.85	62
Soybean	1.00	0.22	0.36	9
Sugarcane	0.98	1.00	0.99	202
Tur	0.80	0.80	0.80	25
Turmeric	1.00	0.45	0.62	11
Urad	0.33	0.10	0.15	20
Wheat	0.91	1.00	0.95	172
accuracy			0.89	903
macro avg	0.80	0.67	0.69	903
weighted avg	0.90	0.89	0.88	903

Figure 23: Random Forest Results

7.3 Decision Trees Model Results.

```
Evaluating Model: Decision Tree

Accuracy: 0.8461
CV Mean: 0.8175
Precision: 0.6302
Recall:    0.6646
F1-score:  0.6380
```

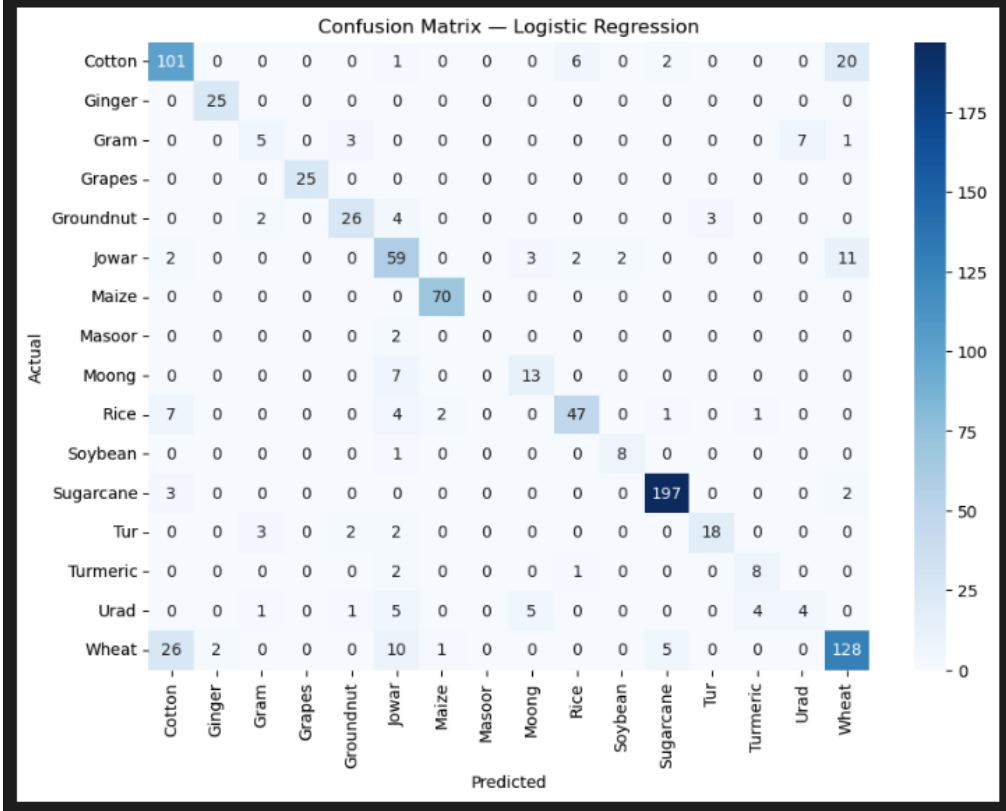



```
Classification Report:
              precision    recall  f1-score   support

      Cotton       0.96      0.87      0.91       130
      Ginger       0.57      0.80      0.67        25
        Gram       0.00      0.00      0.00        16
      Grapes       1.00      0.68      0.81        25
  Groundnut        0.57      0.89      0.70        35
       Jowar       0.82      0.85      0.83        79
       Maize       1.00      0.86      0.92        70
      Masoor       0.00      0.00      0.00         2
       Moong       0.52      0.80      0.63        20
        Rice       0.83      0.85      0.84        62
     Soybean       0.82      1.00      0.90         9
   Sugarcane       0.93      0.95      0.94       202
         Tur       0.57      0.68      0.62        25
    Turmeric       0.62      0.45      0.53        11
        Urad       0.00      0.00      0.00        20
       Wheat       0.88      0.95      0.91       172

    accuracy                           0.85       903
   macro avg       0.63      0.66      0.64       903
weighted avg       0.83      0.85      0.83       903
```

Figure 24: Decision Trees Results

7.4 Logistic Regression Model Results.

```
Evaluating Model: Logistic Regression

Accuracy: 0.8128
CV Mean: 0.7964
Precision: 0.7083
Recall:    0.7027
F1-score:  0.7014
```




```
Classification Report:
              precision    recall  f1-score   support

      Cotton       0.73      0.78      0.75       130
      Ginger       0.93      1.00      0.96        25
        Gram       0.45      0.31      0.37        16
      Grapes       1.00      1.00      1.00        25
   Groundnut       0.81      0.74      0.78        35
       Jowar       0.61      0.75      0.67        79
       Maize       0.96      1.00      0.98        70
      Masoor       0.00      0.00      0.00         2
       Moong       0.62      0.65      0.63        20
        Rice       0.84      0.76      0.80        62
     Soybean       0.80      0.89      0.84         9
   Sugarcane       0.96      0.98      0.97       202
         Tur       0.86      0.72      0.78        25
    Turmeric       0.62      0.73      0.67        11
        Urad       0.36      0.20      0.26        20
       Wheat       0.79      0.74      0.77       172

    accuracy                           0.81       903
   macro avg       0.71      0.70      0.70       903
weighted avg       0.81      0.81      0.81       903
```

Figure 25: Logistic Regression Model Results

7.5 Model Comparison

After training the machine learning models, a comparison was made between them to evaluate their performance using some key metrics: Accuracy, Precision, Recall, F1-score as well as cross-validation scores to determine generalisation ability. The best among the models to be tested was the Random Forest classifier which has better prediction accuracy and strength in relation to the types of crops. Random Forest was chosen as the most appropriate model to be deployed in the Crop Recommendation System because it can be used to model nonlinear relationships and interactions between soil and climate characteristics.

```
Model Performance Comparison:

                Model  Accuracy  CrossVal  Precision (macro)  Recall (macro)  F1-score (macro)
0       Random Forest  0.894795  0.876177           0.803788        0.670078          0.693360
1       Decision Tree  0.846069  0.817452           0.630209        0.664597          0.637971
2  Logistic Regression  0.812846  0.796399           0.708271        0.702673          0.701448

BEST MODEL SELECTED: Random Forest
```

Figure 26: Model Performance Comparison

7.5.1 Accuracy and Cross-validation Comparison

```python
plot_df = results_df.copy()

models = plot_df["Model"].tolist()
accuracy = plot_df["Accuracy"].values
cv_scores = plot_df["CrossVal"].values

x = np.arange(len(models))
width = 0.35

plt.figure(figsize=(10, 6))
bars1 = plt.bar(x - width/2, accuracy, width, label="Accuracy")
bars2 = plt.bar(x + width/2, cv_scores, width, label="Cross-Validation Mean")

plt.title("Model Comparison: Accuracy vs Cross-Validation Score")
plt.xlabel("Machine Learning Models")
plt.ylabel("Score")
plt.ylim(0.7, 1.0)
plt.xticks(x, models, rotation=20)
plt.legend()

# Add value labels on top of each bar
for bars in [bars1, bars2]:
    for bar in bars:
        h = bar.get_height()
        plt.annotate(f"{h:.3f}",
                     xy=(bar.get_x() + bar.get_width()/2, h),
                     xytext=(0, 3),
                     textcoords="offset points",
                     ha="center", va="bottom")

plt.tight_layout()
plt.show()
```

Figure 27: Accuracy and Cross-validation Comparison code.

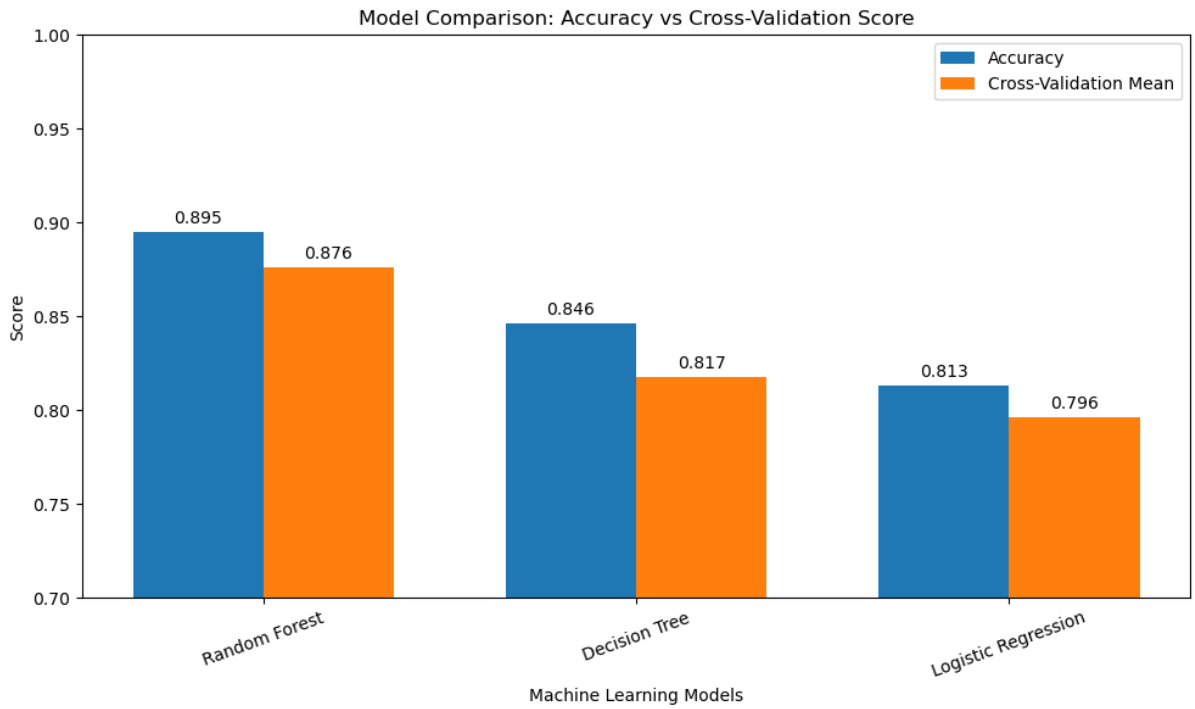


Figure 28: Accuracy and Cross-Validation Comparison Scores.

8 Model Stability Analysis

8.1 Data Leakage Test

A label shuffle test was employed to make sure that model accuracy was not a result of accidental leakage.

```
print("\n Checking for duplicate rows...")
print("Duplicate rows:", df.duplicated().sum())

print("\n Performing label shuffle test...")
y_shuffled = y_train.copy()
np.random.shuffle(y_shuffled)
best_model.fit(X_train, y_shuffled)
print("Leakage Test Accuracy:", best_model.score(X_test, y_test))
```

```
 Checking for duplicate rows...
Duplicate rows: 3

 Performing label shuffle test...
Leakage Test Accuracy: 0.22702104097452935
```

Figure 29: Data Leakage Test with results.

8.2 Noise Sensitivity Analysis

Numeric features were analysed by adding Gaussian noise between 5%30 to determine prediction robustness.

```python
def noise_test(model, X_test, y_test, noise_pct=0.10):
    X_noisy = X_test.copy()
    num_cols = X_noisy.select_dtypes(include=np.number).columns

    for col in num_cols:
        noise = np.random.normal(0, noise_pct * X_noisy[col].std(), size=len(X_noisy))
        X_noisy[col] += noise

    return model.score(X_noisy, y_test)

noise_levels = [0.05, 0.10, 0.20]
noise_results = []

for pct in noise_levels:
    acc = noise_test(best_model, X_test, y_test, noise_pct=pct)
    noise_results.append(acc)
    print(f"Accuracy with {int(pct*100)}% noise:", acc)

plt.figure(figsize=(7,5))
plt.plot([int(n*100) for n in noise_levels], noise_results, marker="o")
plt.xlabel("Noise Level (%)")
plt.ylabel("Accuracy")
plt.title("Noise Robustness of Crop Recommendation Model")
plt.grid(True)
plt.show()
```

```
Accuracy with 5% noise: 0.22812846068660023
Accuracy with 10% noise: 0.22148394241417496
Accuracy with 20% noise: 0.21926910299003322
```

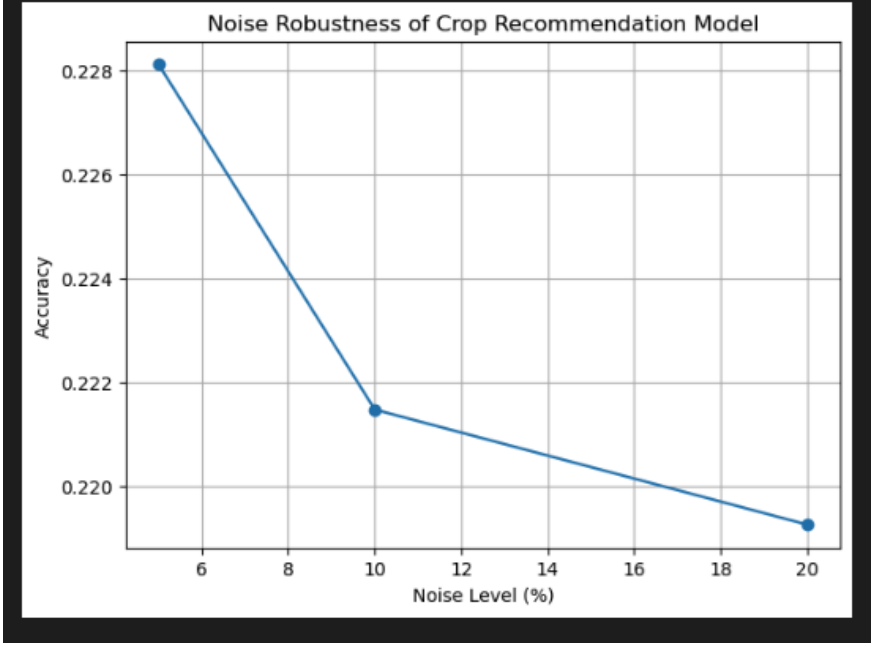


Figure 30: Noise Sensitivity Analysis Code & Results

8.3 ROC Curve for Best performing Model.

```
n_classes = len(crop_le.classes_)
y_test_bin = label_binarize(y_test, classes=np.arange(n_classes))

y_score = best_model.predict_proba(X_test)
fpr = {}
tpr = {}
roc_auc = {}


for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])


fpr["micro"], tpr["micro"], _ = roc_curve(y_test_bin.ravel(), y_score.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# Plotting the ROC curve.
plt.figure(figsize=(8, 6))
plt.plot(fpr["micro"], tpr["micro"], label=f"Random Forest (Micro AUC = {roc_auc['micro']:.3f})")
plt.plot([0, 1], [0, 1], '--', color='Orange')
plt.title("ROC Curve — Random Forest (Best Model)")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.grid(True)
plt.show()
```

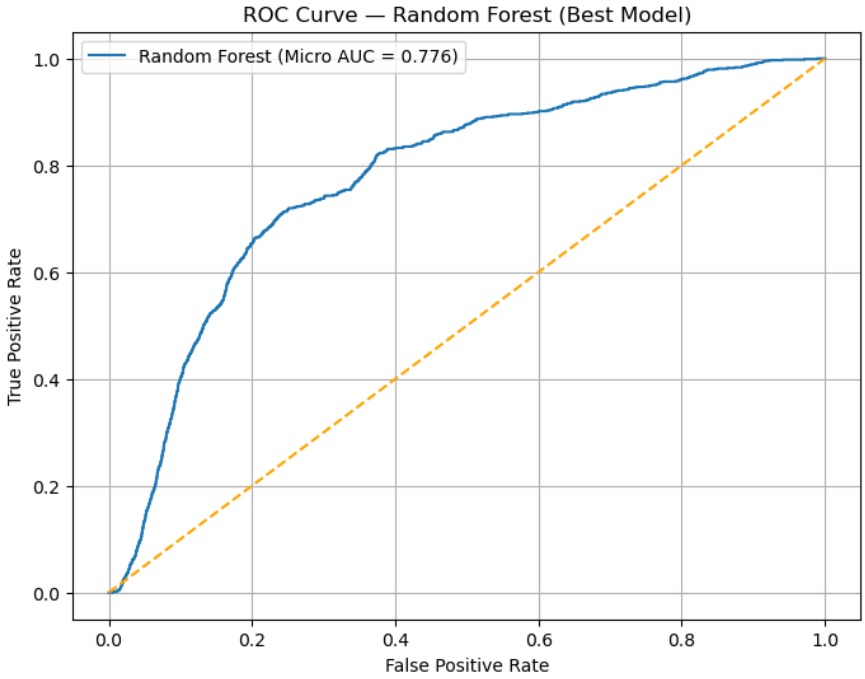


Figure 31: ROC Curve Code and plot for RF (Best Model)

9 SHAP Module

9.1 SHAP Summary Global:(All Crops included).

A TreeExplainer is generated on the Random Forest model. Then number of training samples of around 1500 samples taken is randomly selected
SHAP values of all of the classes are summed into a feature-level matrix and a violin summary plot is created to visualise how each feature drives the predictions above or below all crops.

```
print("\nGenerating SHAP  Summary Plot.")

explainer = shap.TreeExplainer(best_model)

sample_size = min(1500, X_train.shape[0])
sample_idx = np.random.choice(X_train.shape[0], sample_size, replace=False)
shap_sample = X_train.iloc[sample_idx]

shap_raw = explainer.shap_values(shap_sample)

if isinstance(shap_raw, list):
    shap_matrix = np.mean(np.abs(np.stack(shap_raw, axis=0)), axis=0)
else:
    shap_raw = np.asarray(shap_raw)
    shap_matrix = np.mean(np.abs(shap_raw), axis=2)

shap_matrix = shap_matrix[:, :X_train.shape[1]]
safe_max_display = X_train.shape[1]

plt.figure(figsize=(12,10))
shap.summary_plot(
    shap_matrix,
    shap_sample,
    feature_names=X_train.columns,
    plot_type="violin",
    max_display=safe_max_display,
    show=False
)
plt.title("SHAP  Summary for overall Feature Impact")
plt.show()
```

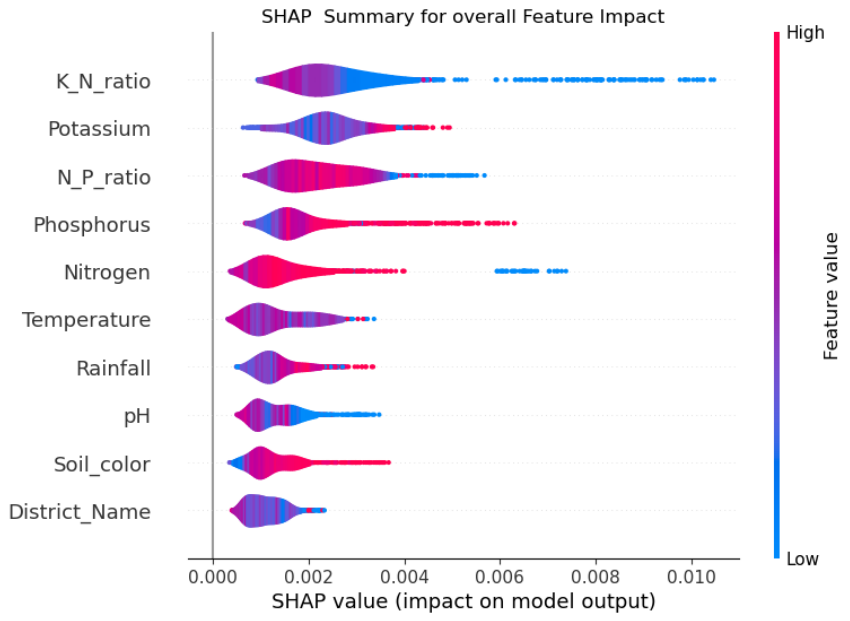


Figure 32: SHAP Summary Plot for Overall Feature Imapct

9.2 Global Average Importance of Features:

Mean absolute SHAP values per feature are calculated in each of the classes.
They are averaged over all the classes and represented in terms of a horizontal bar chart with the title of Averaged SHAP Feature Importance (All Crops).
Characteristics like K N ratio, NP ratio, district name and nitrogen are some of the most powerful drivers.

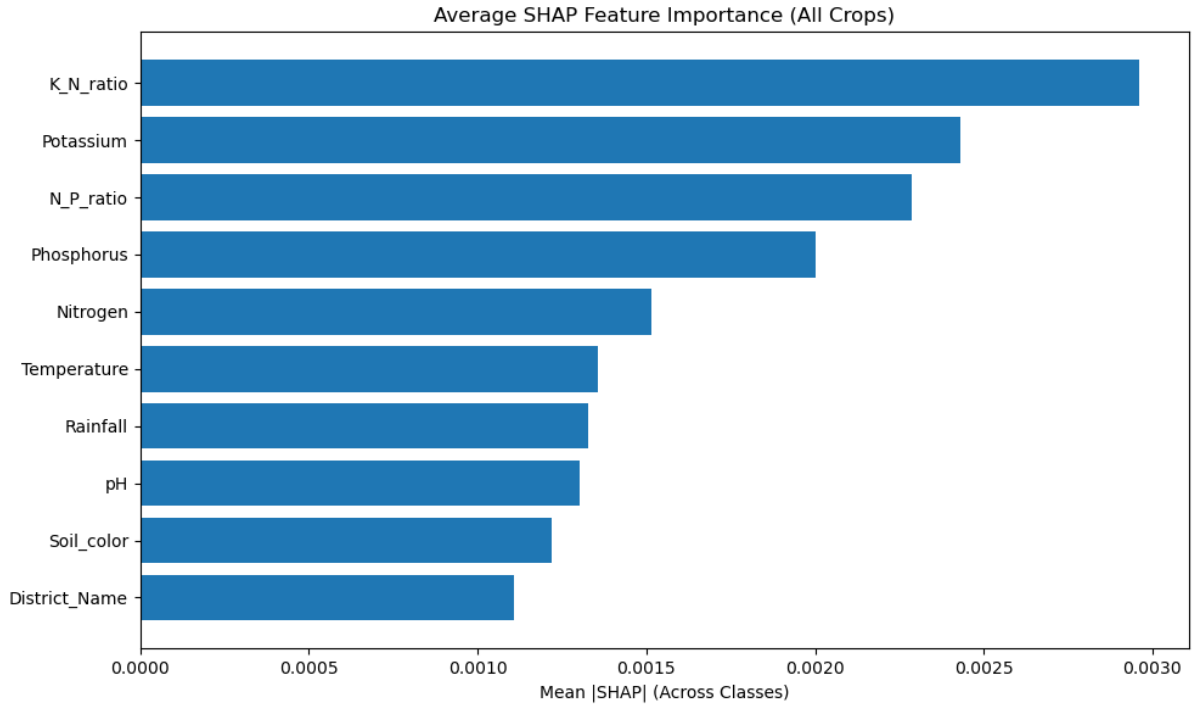
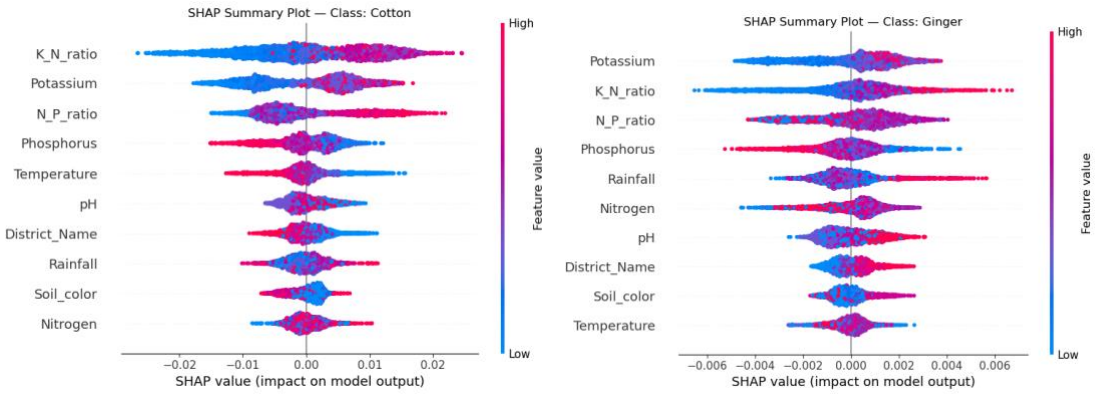
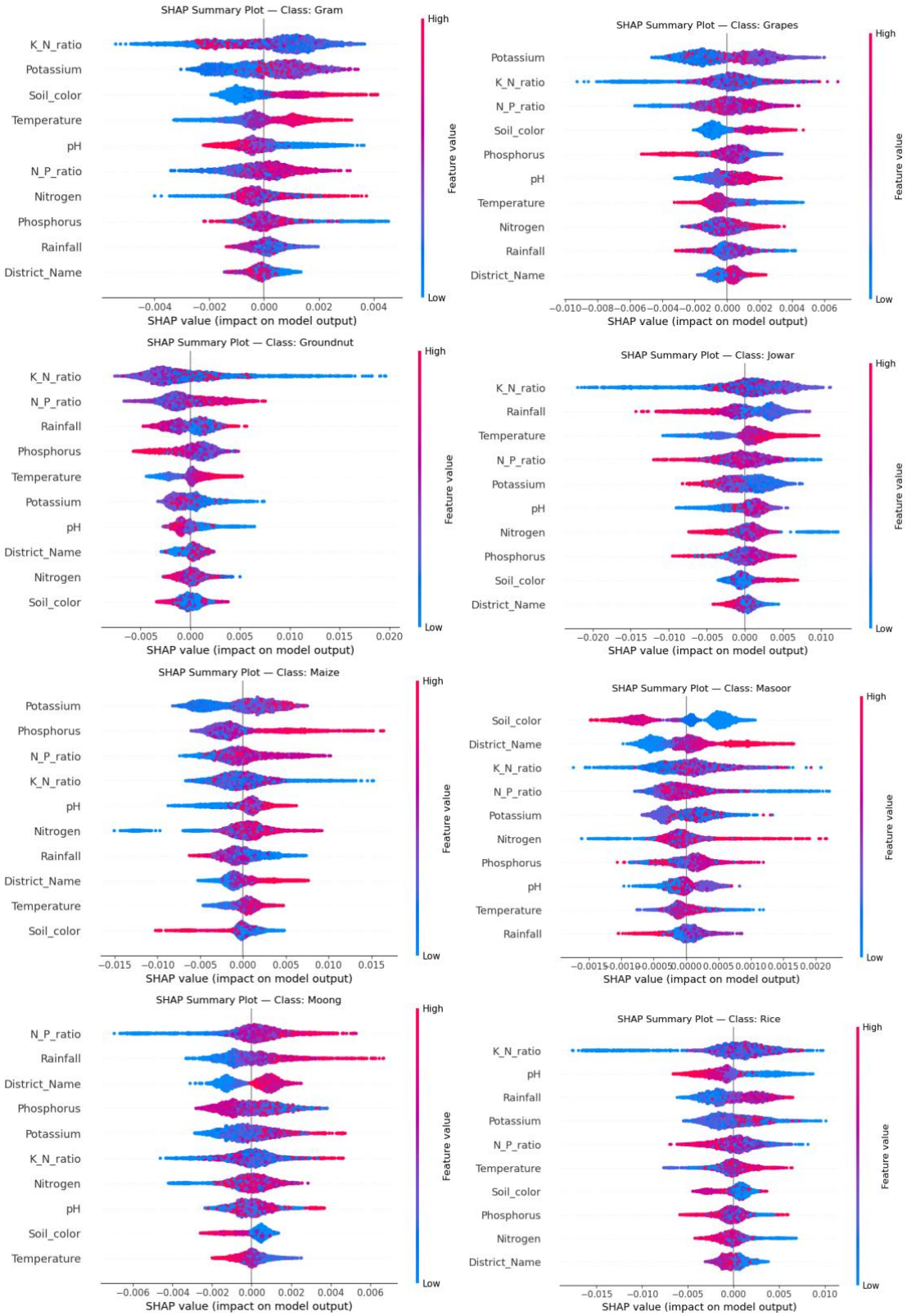


Figure 33: Average Importance of Features.

9.3 Per-Class SHAP Violin Plots:

A normalisation custom function is used to normalise different SHAP output shapes.
A SHAP summary plot of each of the 16 crop classes are created which displays the features that influence the predictions in that respective crop (e.g., Sugarcane, Wheat, Cotton).




SHAP Summary Plot — Class: Gram
K_N_ratio
Potassium
Soil_color
Temperature
pH
N_P_ratio
Nitrogen
Phosphorus
Rainfall
District_Name
−0.004 −0.002 0.000 0.002 0.004
SHAP value (impact on model output)
High
Feature value
Low
SHAP Summary Plot — Class: Grapes
Potassium
K_N_ratio
N_P_ratio
Soil_color
Phosphorus
pH
Temperature
Nitrogen
Rainfall
District_Name
−0.010 −0.008 −0.006 −0.004 −0.002 0.000 0.002 0.004 0.006
SHAP value (impact on model output)
High
Feature value
Low
SHAP Summary Plot — Class: Groundnut
K_N_ratio
N_P_ratio
Rainfall
Phosphorus
Temperature
Potassium
pH
District_Name
Nitrogen
Soil_color
−0.005 0.000 0.005 0.010 0.015 0.020
SHAP value (impact on model output)
High
Feature value
Low
SHAP Summary Plot — Class: Jowar
K_N_ratio
Rainfall
Temperature
N_P_ratio
Potassium
pH
Nitrogen
Phosphorus
Soil_color
District_Name
−0.020 −0.015 −0.010 −0.005 0.000 0.005 0.010
SHAP value (impact on model output)
High
Feature value
Low
SHAP Summary Plot — Class: Maize
Potassium
Phosphorus
N_P_ratio
K_N_ratio
pH
Nitrogen
Rainfall
District_Name
Temperature
Soil_color
−0.015 −0.010 −0.005 0.000 0.005 0.010 0.015
SHAP value (impact on model output)
High
Feature value
Low
SHAP Summary Plot — Class: Masoor
Soil_color
District_Name
K_N_ratio
N_P_ratio
Potassium
Nitrogen
Phosphorus
pH
Temperature
Rainfall
−0.0015 −0.0010 −0.0005 0.0000 0.0005 0.0010 0.0015 0.0020
SHAP value (impact on model output)
High
Feature value
Low
SHAP Summary Plot — Class: Moong
N_P_ratio
Rainfall
District_Name
Phosphorus
Potassium
K_N_ratio
Nitrogen
pH
Soil_color
Temperature
−0.006 −0.004 −0.002 0.000 0.002 0.004 0.006
SHAP value (impact on model output)
High
Feature value
Low
SHAP Summary Plot — Class: Rice
K_N_ratio
pH
Rainfall
Potassium
N_P_ratio
Temperature
Soil_color
Phosphorus
Nitrogen
District_Name
−0.015 −0.010 −0.005 0.000 0.005 0.010
SHAP value (impact on model output)
High
Feature value
Low

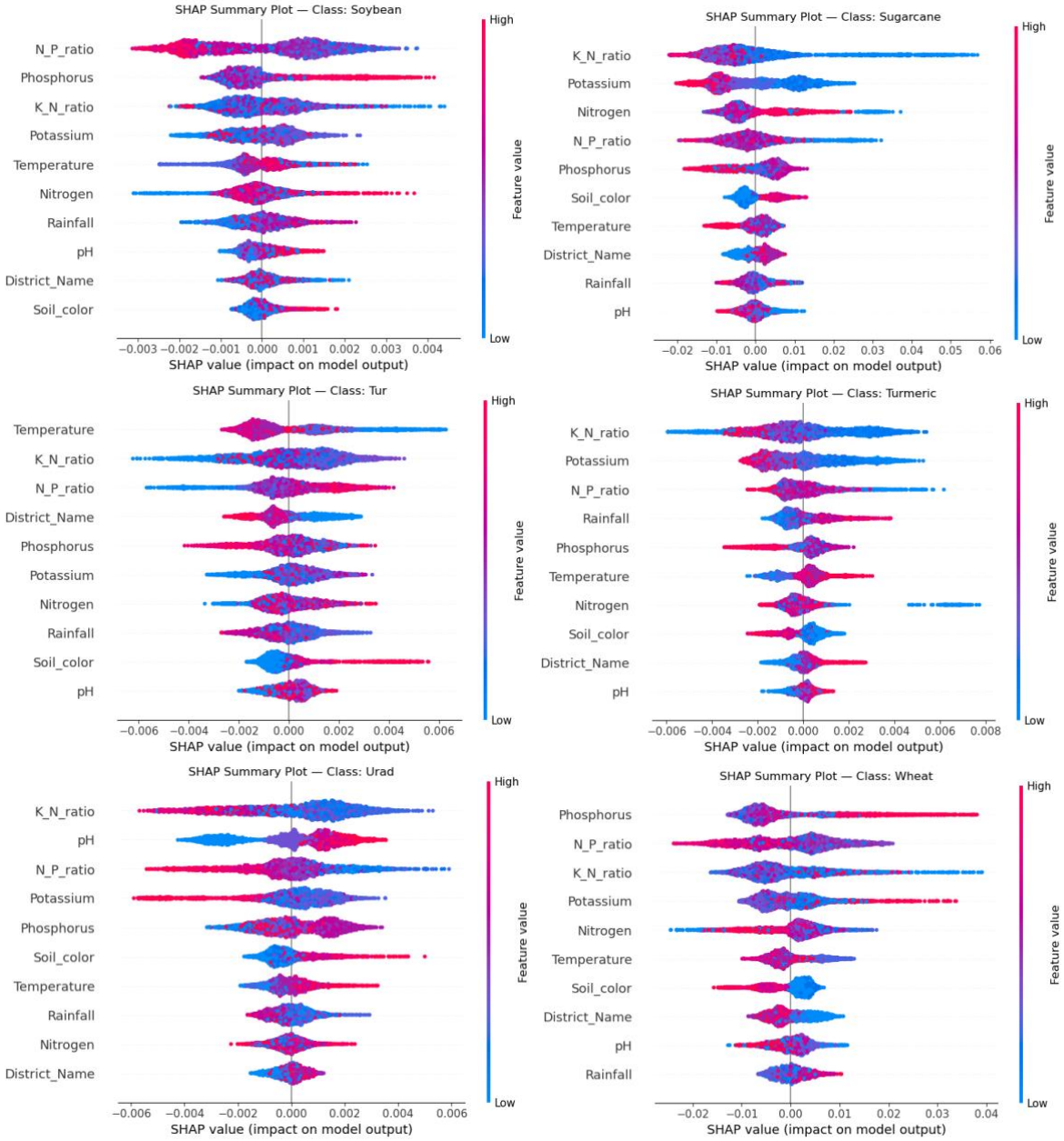


Figure 34: SHAP Summary Plots for All 16 Crops.

10 Top-3 Crop Recommendation Module.

The module recommends top-3 crops and also provides reasoning behind that choice of crop. It works in the following way:

- The input of the farmer is in the form of a dictionary, which includes the information on the district, soil colour, nitrogen, phosphorus, potassium, pH, rainfall, and temperatures.
- Then Label Encoders are used to convert categorical values to numbers in the same manner they were during model training.
- In case one of the input features is not provided, the input is automatically filled using the median of the training dataset to prevent errors.
- The trained model of the Random Forest models the probability of all crops and with the help of the function the 3 most probable crops are identified.

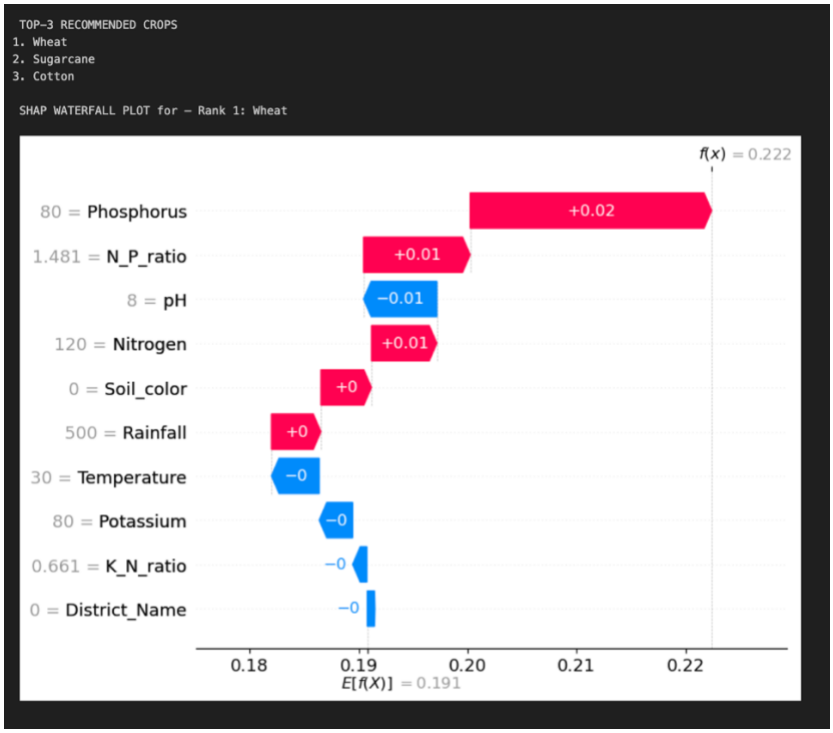
- SHAP values are done on each of these top 3 crops. A waterfall plot is then established to indicate how each input (e.g., nitrogen, pH, rainfall) contributed towards the choice that the model made in that particular crop.

This will ensure that besides the system giving the best crops, it will also give the explaination behind the recommendation and hence the predictions made by the system can be easily understood by the farmers.

```
def shap_waterfall_top3(
    input_data: dict,
    model=best_model,
    encoders=label_encoders,
    fallback=cat_default_encoded,
    crop_encoder=crop_le,
    explainer=explainer,
    X_train_df=X_train,
    top_n=3
):



    df_input = pd.DataFrame([input_data])

    for col, le in encoders.items():
        if col in df_input.columns:
            vals = df_input[col].astype(str).values
            df_input[col] = [
                le.transform([v])[0] if v in le.classes_ else fallback[col]
                for v in vals
            ]


    for col in X_train_df.columns:
        if col not in df_input.columns:
            df_input[col] = X_train_df[col].median()

    df_input = df_input[X_train_df.columns]


    probs = model.predict_proba(df_input)[0]
    top_indices = probs.argsort()[-top_n:][::-1]

    top_crops = [(crop_encoder.inverse_transform([i])[0], probs[i]) for i in top_indices]


    print("\n TOP-3 RECOMMENDED CROPS")


    for rank, (crop_name, p) in enumerate(top_crops, 1):
        print(f"{rank}. {crop_name} ")


    shap_values_input = explainer(df_input)

    for rank, cls in enumerate(top_indices, 1):

        crop_name = crop_encoder.inverse_transform([cls])[0]


        print(f"\n SHAP WATERFALL PLOT for — Rank {rank}: {crop_name}")

        # Extract SHAP vector for this class
        shap_vec = shap_values_input.values[0][:, cls]
        base_value = shap_values_input.base_values[0][cls]

        explanation = shap.Explanation(
            values=shap_vec,
            base_values=base_value,
            data=df_input.iloc[0],
            feature_names=df_input.columns
        )

        shap.plots.waterfall(explanation, max_display=15)

example_input = X_test.iloc[0].to_dict()
print("\nExample Input:", example_input)
shap_waterfall_top3(example_input)
```

Figure 35: Top-3 Crop Recommendation.

Figure 36: Example Input for Top-3 Crop Recommendation system.



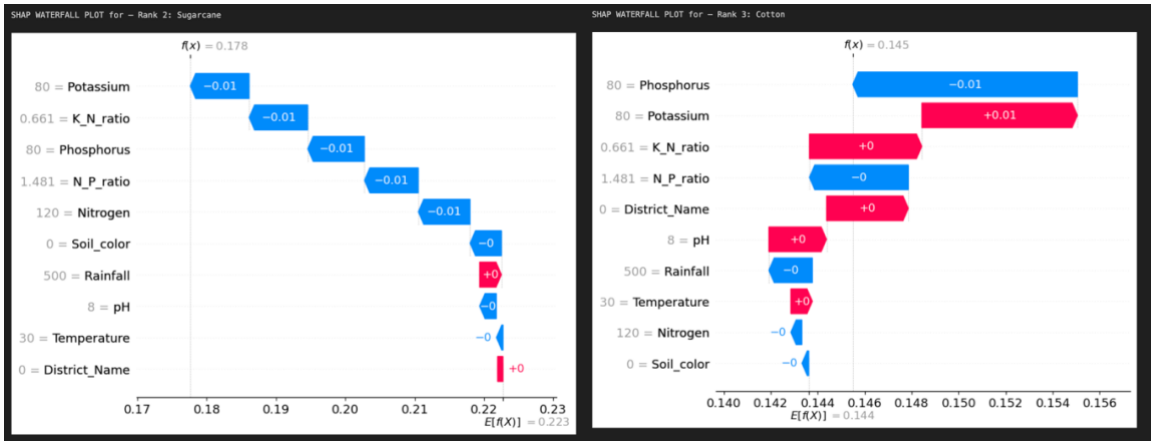


Figure 37: Output SHAP Waterfall Plots for Top-3 Crop Recommended.

11 Saving the Model & Encoder Files.

```
joblib.dump(best_model, "best_crop_model.pkl")
joblib.dump(label_encoders, "feature_label_encoders.pkl")
joblib.dump(crop_le, "crop_label_encoder.pkl")
joblib.dump(cat_default_encoded, "feature_fallback_encoded.pkl")

print(" Model and encoders saved successfully!")
```

```
Model and encoders saved successfully!
```

Figure 38: Code & output for Saving Model & encoder files.