

# Configuration Manual

MSc Research Project  
Data Analytics

Vikraman Kannappan Sankar  
Student ID: x23104881

School of Computing  
National College of Ireland

Supervisor: Prof. Jorge Basilio

**National College of Ireland**  
**MSc Project Submission Sheet**  
**School of Computing**



**Student Name:** Vikraman Kannappan Sankar  
**Student ID:** x23104881  
**Programme:** MSc Data Analytics **Year:** 2023 - 24  
**Module:** MSc Research Project  
**Lecturer:** Prof. Jorge Basilio  
**Submission Due Date:** 12/08/2024  
**Project Title:** Impact of Historical Weather Data on Crop Selection and Fertilizer Recommendation using Machine Learning Stacking Approach: Indian Cities

**Word Count:** 1046 **Page Count:** 14

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:** Vikraman Kannappan Sankar

**Date:** 12<sup>th</sup> August 2024

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST**

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

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

<b>Office Use Only</b>	
Signature:	
Date:	
Penalty Applied (if applicable):	

# Configuration Manual

Vikraman Kannappan Sankar

Student ID: x23104881

## 1 Introduction

The configuration manual that offers a detailed guide for recreating the experiment setup and the findings of the research project on how the impact of historical weather data crop selection and fertilizer recommendations of the Indian cities using a machine learning stacking approach. Study evaluates the various machine learning models Random Forest, Decision Tree, K-Nearest Neighbours and MLP Classifiers. This manual contains all the technical information which will include the software, packages and modules along with version to ensure the consistent environment for the re-running the experiment. This set of instructions will help the user to configure the environment and conduct the case experiment and analyse the results. By using this handbook, the user can reproduce the experimental setup and can contribute to the understating and the improvement of the recommendation system for the real-word applications.

## 2 Development Environment

The development environment used for this research is the local windows operating systems with GPU. Both the hardware and the software specification details are mentioned below.

The dataset used for the projects crop production statistics – India data, city-specific historical weather data, and records of fertilizers data

### 2.1 Hardware Specification

- Processor: AMD Ryzen 7 5800HS 3.20 GHz
- RAM: 16.0 GB (15.4 GB usable)
- GPU – NIVIDA RTX 3050

This above-mentioned Hardware Specs based Local System was used to create the environment, to re-run the setup it is not necessary to have the same specification to re-create the environment.

### 2.2 Software Specification

- Operating System: Windows 11 or any other operating system can be used.
- Programming Language: Python version 3.11.5

3.11.5 | packaged by Anaconda, Inc. | (main, Sep 11 2023, 13:26:23) [MSC v.1916 64 bit (AMD64)]

Figure 1: Python Version

- Integrated Development Environment (IDE): Jupyter Notebook 6.5.4 or higher version.

#### Server Information:

You are using Jupyter Notebook.

The version of the notebook server is: **6.5.4**

The server is running on this version of Python:

Figure 2: Jupyter Notebook

## 2.3 Python Libraries required

Figure 3 display the list of the essential Python Libraries required for the execution of the code. This mentioned python libraries can be installed using the pip command.

- Numpy
- Pandas
- Scikit-learn
- Matplotlib
- Seaborn
- Pickle

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pickle
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from pandas.plotting import parallel_coordinates
from sklearn.metrics import accuracy_score, mean_absolute_error, mean_squared_error, recall_score, precision_score, f1_score,
from sklearn.model_selection import GridSearchCV
```

Figure 3 Libraries Used

## 3 Data Source

Total three dataset was used for this study. All the Dataset for this study is sourced from Kaggle Platform.

- Crop Production Statistics – India Data: -  
<https://www.kaggle.com/datasets/nikhilmahajan29/crop-production-statistics-india>

	State	District	Crop	Crop_Year	Season	Area	Production	Yield
0	Andaman and Nicobar Island	NICOBARS	Arecanut	2007	Rabi	1626.4	2277.0	1.40
1	Andaman and Nicobar Island	NICOBARS	Arecanut	2008	Autumn	4147.0	3060.0	0.74
2	Andaman and Nicobar Island	NICOBARS	Arecanut	2009	Autumn	4153.0	3120.0	0.75
3	Andaman and Nicobar Island	NICOBARS	Arecanut	2009	Summer	4153.0	2080.0	0.50
4	Andaman and Nicobar Island	NICOBARS	Arecanut	2006	Whole Year	896.0	478.0	0.53

Figure 4 Crop Production Statistics – India Dataset

- Historical Weather Dataset: -

<https://www.kaggle.com/datasets/hiteshsoneji/historical-weather-data-for-indian-cities>

	date_time	maxtempC	mintempC	sunHour	pressure	uvIndex	sunHour	moonset	moonrise	humidity	tempC	winddirDegree	windspeedKmph	City
0	2009-01-01 00:00:00	27	12	11.6	1014	5	11.6	10:03 PM	09:58 AM	91	14	109	8	Bengaluru
1	2009-01-01 01:00:00	27	12	11.6	1014	5	11.6	10:03 PM	09:58 AM	93	14	85	6	Bengaluru
2	2009-01-01 02:00:00	27	12	11.6	1014	5	11.6	10:03 PM	09:58 AM	94	13	61	4	Bengaluru
3	2009-01-01 03:00:00	27	12	11.6	1014	5	11.6	10:03 PM	09:58 AM	96	12	37	3	Bengaluru
4	2009-01-01 04:00:00	27	12	11.6	1015	5	11.6	10:03 PM	09:58 AM	88	14	45	3	Bengaluru

Figure 5 Historical Weather Dataset

- Fertilizer Dataset: -

<https://www.kaggle.com/datasets/gdabhishek/fertilizer-prediction-> Kaggle

	Temperature	Humidity	Moisture	Soil Type	Crop Type	Nitrogen	Potassium	Phosphorous	Fertilizer Name
0	26	52	38	Sandy	Maize	37	0	0	Urea
1	29	52	45	Loamy	Sugarcane	12	0	36	DAP
2	34	65	62	Black	Cotton	7	9	30	14-35-14
3	32	62	34	Red	Tobacco	22	0	20	28-28
4	28	54	46	Clayey	Paddy	35	0	0	Urea

Figure 6 Fertilizer Dataset

## 4 Project Code Files

Four Jupyter Notebook Code Files were used in this study.

- Final Data Set 1 Cleaning – This File contains the code of the Data Cleaning Process of the Crop Production Statistics – India Dataset.
- Final Data Set 2 Cleaning – This File contains the code of the Data Cleaning Process of the Historical Weather Dataset.

- Final Data Set 3 Cleaning – This File contains the code of Data Cleaning Process of the Fertilizer Dataset.
- Model Creation Final – This File contains the code of the Merging of the three data, Model Creation and Evaluation.





Name	Type
 Final Data Set 1 Cleaning.ipynb	IPYNB File
 Final Data Set 2 Cleaning.ipynb	IPYNB File
 Final Data Set 3 Cleaning.ipynb	IPYNB File
 Model Creation Final.ipynb	IPYNB File

Figure 7 Jupyter Notebook Files

## 5 Data Preparation

### 5.1 Extracting Data:

Extracting the Data from the CSV file (Final Data Set 1, 2, 3 notebook files)

```
# Reading the file as Crop Production Data Set
df = pd.read_csv(r'D:\Assignment\Final Research\Dataset\Main Datasets\APY.csv')

# Importing the City Weather Data CSV Files
csv_files = [
    (r'D:\Assignment\Final Research\Dataset\Main Datasets\bengaluru.csv', 'Bengaluru'),
    (r'D:\Assignment\Final Research\Dataset\Main Datasets\bombay.csv', 'Bombay'),
    (r'D:\Assignment\Final Research\Dataset\Main Datasets\delhi.csv', 'Delhi'),
    (r'D:\Assignment\Final Research\Dataset\Main Datasets\hyderabad.csv', 'Hyderabad'),
    (r'D:\Assignment\Final Research\Dataset\Main Datasets\jaipur.csv', 'Jaipur'),
    (r'D:\Assignment\Final Research\Dataset\Main Datasets\kanpur.csv', 'Kanpur'),
    (r'D:\Assignment\Final Research\Dataset\Main Datasets\nagpur.csv', 'Nagpur'),
    (r'D:\Assignment\Final Research\Dataset\Main Datasets\pune.csv', 'Pune')
]

# Concatenating the CSV files and Key 'City' column
dataframes = []
for file, city in csv_files:
    df = pd.read_csv(file)
    df['City'] = city
    dataframes.append(df)

# Reading the file as Fertilizer Prediction Data frame
df = pd.read_csv('D:\Assignment\Final Research\Dataset\Main Datasets\Fertilizer Prediction.csv')
```

Figure 8: Extracting Data from CSV File

### 5.2 Data Pre-Processing:

Checking for the missing data and performing the cleaning process, data transformation, data reduction and feature engineering and merging dataset. (Final Data Set 1, 2, 3 Notebook files)

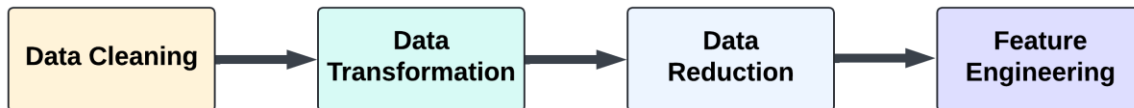


Figure 9: Data Pre-Processing Flowchart

```

# Checking if there are any missing values in the data
missing_values = df.isnull().sum()
print("Checking for missing values:",missing_values)

```

Figure 10: Checking Missing Values

- **Cluster Interpolation**

Cluster Interpolation technique was used for the Production Column. (Final Data Set 1 Notebook File)

```

# Determine the Optimal number of clusters using the elbow method
sse = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(scaled_features)
    sse.append(kmeans.inertia )
# Applying the interpolation within each cluster
df['Production'] = df.groupby('Cluster')['Production'].apply(cluster_interpolation).reset_index(level=0, drop=True)

```

Figure 11: Applying Cluster Interpolation

- **Datatype Conversion**

The below image shows the process of the Data Type Conversion. (Final Data Set 1, 2, 3 Notebook files)

```

# Convert numerical columns to proper data types
df['Area'] = pd.to_numeric(df['Area'], errors='coerce')
df['Production'] = pd.to_numeric(df['Production'], errors='coerce')
df['Yield'] = pd.to_numeric(df['Yield'], errors='coerce')
df['State'] = df['State'].astype('category')
df['District'] = df['District'].astype('category')
df['Crop'] = df['Crop'].astype('category')
df['Season'] = df['Season'].astype('category')
df['Crop_Year'] = pd.to_numeric(df['Crop_Year'], errors='coerce')

```

```

#Convert columns to appropriate data types
combined_df['DateTime'] = pd.to_datetime(combined_df['DateTime'])
combined_df['MaxTempC'] = combined_df['MaxTempC'].astype(float)
combined_df['MinTempC'] = combined_df['MinTempC'].astype(float)
combined_df['totalSnow_cm'] = combined_df['totalSnow_cm'].astype(float)
combined_df['SunHour'] = combined_df['SunHour'].astype(float)
combined_df['UVIndex'] = combined_df['UVIndex'].astype(int)
combined_df['uvIndex.1'] = combined_df['uvIndex.1'].astype(int)
combined_df['moon_illumination'] = combined_df['moon_illumination'].astype(int)
combined_df['WindGustKmph'] = combined_df['WindGustKmph'].astype(int)
combined_df['cloudcover'] = combined_df['cloudcover'].astype(int)
combined_df['Humidity'] = combined_df['Humidity'].astype(int)
combined_df['precipMM'] = combined_df['precipMM'].astype(float)
combined_df['Pressure'] = combined_df['Pressure'].astype(int)
combined_df['TempC'] = combined_df['TempC'].astype(float)
combined_df['visibility'] = combined_df['visibility'].astype(int)
combined_df['WindDirDegree'] = combined_df['WindDirDegree'].astype(int)
combined_df['WindSpeedKmph'] = combined_df['WindSpeedKmph'].astype(int)
# Convert numeric columns
numeric_columns = ['Temperature', 'Humidity', 'Moisture', 'Nitrogen', 'Potassium', 'Phosphorous']
for col in numeric_columns:
    df[col] = pd.to_numeric(df[col], errors='coerce')

```

Figure 12: Converting date datatype

- **Stripping Negative Space**

The below image shows the process of the Stripping the whitespace in the column names. (Final Data Set 1, 2, 3 Notebook files)

```

# Strip whitespace from the column names
df.columns = df.columns.str.strip()

```

Figure 13: Stripping Space in the Column

- **Data Labelling**

The below image shows the process of the data labelling of the Crop Dataset. (Final Dataset 1 Notebook File)

```

# Data Transformation, Data Label Correction
df['Crop'] = df['Crop'].replace('Cotton(lint)', 'Cotton')
df['Crop'] = df['Crop'].replace('Small millets', 'Millets')
df['Crop'] = df['Crop'].replace('Oilseeds total, other oilseeds', 'Oil seeds')
df['Crop'] = df['Crop'].replace('other oilseeds', 'Oil seeds')
df['District'] = df['District'].replace('BENGALURU URBAN', 'BENGALURU')
df['District'] = df['District'].replace('Mumbai', 'BOMBAY')
df['District'] = df['District'].replace('DELHI_TOTAL', 'DELHI')
df['District'] = df['District'].replace('Mumbai', 'BOMBAY')
df['District'] = df['District'].replace('KANPUR DEHAT', 'KANPUR')

```

Figure 14: Data Label Correction



- **Feature Extraction**

The below image shows the process of creation of new seasonal feature (Final Dataset 2 Notebook File)

```
# Convert DateTime column to datetime format
new_df['DateTime'] = pd.to_datetime(new_df['DateTime'])

# Extracting the Year and Month
new_df['Year'] = new_df['DateTime'].dt.year
new_df['Month'] = new_df['DateTime'].dt.month

print(new_df[['Year', 'Month']])
```

Figure 15: Extracting Year, Month from Data Time Column

```
# Label Function
def label_season(row):
    if row['Month'] in [10, 11]:
        return 'Autumn'
    else:
        return 'Not Autumn'

# New Feature Season is created using the months
result['Season'] = result.apply(label_season, axis=1)

autumn_data = result[result['Season'] == 'Autumn']

# Group by Year, City, and Season Autumn
grouped_data_autumn = autumn_data.groupby(['Year', 'City', 'Season']).agg({
    'MaxTempC': 'mean',
    'MinTempC': 'mean',
    'Humidity': 'mean',
    'SunHour': 'mean',
    'Pressure': 'mean',
    'TempC': 'mean',
    'WindSpeedKmph': 'mean'
}).reset_index()
```

Figure 16: Creating New Feature Season based on the months and Grouping the Season Data

```
combined_data = pd.concat([grouped_data_kharif, grouped_data_rabi, grouped_data_autumn, grouped_data_yearly,
                           grouped_data_summer], ignore_index=True)
```

Figure 17: Combining the Group Data of the Season

- **Merging the Dataset**

Merging the Crop Production Data with the Weather Data using the common key columns 'Year', 'City', and 'Season' and then this merged data frame is merged with fertilizer data using the 'City' key column. (Model Creation Final Notebook File)

```
# Merging the Dataset 1 and Dataset 2
merged_df = pd.merge(crop_df, weath_df, on=['Year', 'City', 'Season'], how='inner')
# Merging the merged dataframe and the Fertilizer
merged_df = df1.merge(df2, left_on='Crop', right_on='Crop Type', how='left')
```

Figure 18: Merging the Dataset

- **Outliers Removal**

Outliers in the data was removed using the Interquartile Range.

```
# Remove Outliers using IQR
Q1 = sorted_df[numeric_columns].quantile(0.25)
Q3 = sorted_df[numeric_columns].quantile(0.75)
IQR = Q3 - Q1
sorted_df = sorted_df[~((sorted_df[numeric_columns] < (Q1 - 1.5 * IQR)) | (sorted_df[numeric_columns] > (Q3 + 1.5 * IQR))).any(a
```

Figure 19: Removing the Outliers using the IQR

- **Label Encoding**

Label Encoding is applied to the Categorical Columns.

```
label_encoders = {}

categorical_columns = ['Season', 'Soil Type', 'Crop', 'State', 'City', 'Fertilizer Name']

for column in categorical_columns:
    le = LabelEncoder()
    sorted_df[column] = le.fit_transform(sorted_df[column])
    label_encoders[column] = le
```

Figure 20: Applying Label Encoding

- **Standardizing Features**

```
# Standardizing features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Figure 21: Standardizing Features

- **Feature Selection**

Recursive Feature Elimination is used to extract the Features.

```
# Recursive Feature Elimination
rfe = RFE(estimator=RandomForestClassifier(n_estimators=50, random_state=42), n_features_to_select=10)
X_rfe = rfe.fit_transform(X_scaled, y_crop)
```

Figure 22: Feature Selection using the Recursive Feature Elimination

## 6 Exploratory data analysis

This section is about the exploratory data analysis of the Crop Production, Weather Data and Fertilizer Data and Merged Data (Final Data Set 1, 2, 3 and Model Creation Final Notebook files).

```

bars = plt.bar(yearly_production['Crop_Year'], yearly_production['Production'], color='skyblue', edgecolor='black')
plt.xlabel('Year', fontsize=14, fontweight='bold', color='black')
plt.ylabel('Total Production', fontsize=14, fontweight='bold', color='black')
plt.title('Production Over the Years', fontsize=16, fontweight='bold')
plt.xticks(yearly_production['Crop_Year'], rotation=45, ha='right')
plt.grid(axis='y', linestyle='--', alpha=0.7)

# Plotting
plt.figure(figsize=(12, 8))

plt.tight_layout()
plt.show()

```

Figure 23: Total Crop Production Over the Years

```

# Histograms
plt.figure(figsize=(14, 10))
plt.subplot(2, 2, 1)
sns.histplot(combined_df['MaxTempC'], kde=True, bins=10)
plt.title('Histogram of Max Temperature')

```

Figure 24: Histogram of the Max Temperature

```

df_melted = df.melt(id_vars=['Fertilizer Name'], value_vars=['Nitrogen', 'Potassium', 'Phosphorous'], var_name='Nutrient', value_name='Amount')

plt.figure(figsize=(10, 6))
sns.barplot(x='Fertilizer Name', y='Amount', hue='Nutrient', data=df_melted)
plt.title('Nutrient Content in Different Fertilizers')
plt.xlabel('Fertilizer Name')
plt.ylabel('Amount')
plt.legend(title='Nutrient')
plt.show()

```

Figure 25: Nutrient Content of the Fertilizer

```

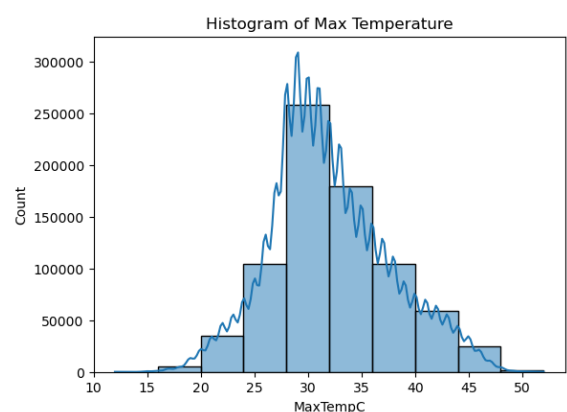
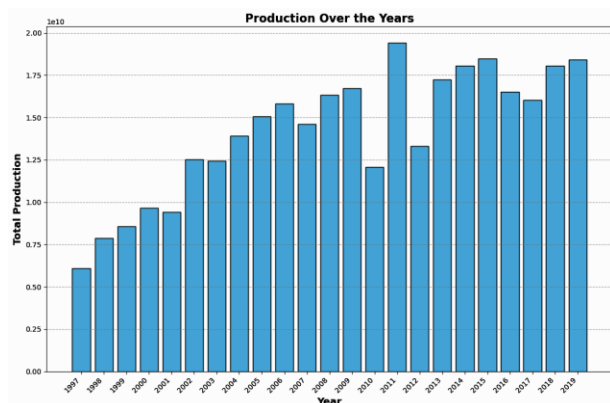
# Count of each city
city_counts = sorted_df['City'].value_counts()

colors = plt.cm.tab20c(range(len(city_counts))) # 'tab20c' colormap provides a good variety of colors

plt.figure(figsize=(10, 8))
plt.pie(city_counts, labels=city_counts.index, colors=colors, autopct='%1.1f%%', startangle=140, textprops={'fontsize': 12, 'fontweight': 'bold'})
plt.title('Crop Production of City', fontsize=14, fontweight='bold')
plt.show()

```

Figure 26: Count of the Cities



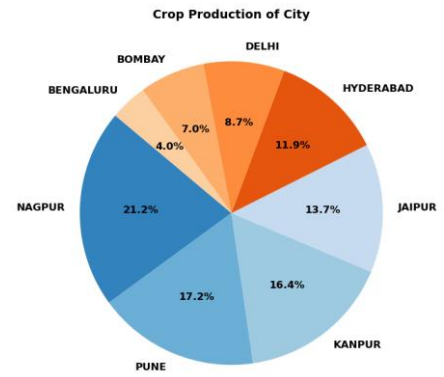
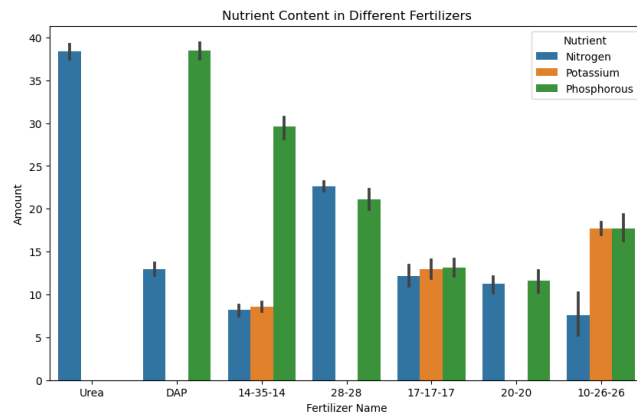


Figure 27: Different Types of EDA Graph

## 7 Model Building

This Section is the overview of splitting the data, model creation of the Individual and the Stacked Models, Evaluation Metrics.

- **Splitting the Data**

```
# Train-Test Split
X_train, X_test, y_train_crop, y_test_crop = train_test_split(X_rfe, y_crop, test_size=0.3, random_state=42)
X_train_fert, X_test_fert, y_train_fert, y_test_fert = train_test_split(X_rfe, y_fertilizer, test_size=0.3, random_state=42)
```

Figure 28: Dataset Splitting

- **Model Initialization**

```
# Defining the Models
models = {
    'rf': RandomForestClassifier(n_estimators=50, max_depth=5, random_state=42),
    'dt': DecisionTreeClassifier(max_depth=5, random_state=42),
    'knn': KNeighborsClassifier(n_neighbors=5),
    'nn': MLPClassifier(hidden_layer_sizes=(50,), max_iter=1000, random_state=42)
}
```

Figure 29: Initializing the models

- **Individual Model Training**

```
# Training of the Crop Recommendation Model
crop_predictions = {}
for name, model in models.items():
    print(f"Training {name} model for crop recommendation.")
    model.fit(X_train, y_train_crop)
    crop_predictions[name] = model.predict(X_test)
    with open(f"model_checkpoints/{name}_crop_model.pkl", 'wb') as file:
        pickle.dump(model, file)
```

Training rf model for crop recommendation.  
 Training dt model for crop recommendation.  
 Training knn model for crop recommendation.  
 Training nn model for crop recommendation.

```
# Training of the Fertilizer Recommendation Model
fertilizer_predictions = {}
for name, model in models.items():
    print(f"Training {name} model for fertilizer recommendation.")
    model.fit(X_train_fert, y_train_fert)
    fertilizer_predictions[name] = model.predict(X_test_fert)
    with open(f"model_checkpoints/{name}_fertilizer_model.pkl", 'wb') as file:
        pickle.dump(model, file)
```

Training rf model for fertilizer recommendation.  
 Training dt model for fertilizer recommendation.  
 Training knn model for fertilizer recommendation.  
 Training nn model for fertilizer recommendation.

Figure 30: Training the Individual Models for Stacking

- **Creation of the Meta Model**

```
# Creation of the meta model
def evaluate_stacking_model(base_predictions, y_meta, meta_model, meta_model_name):
    meta_X = np.column_stack(list(base_predictions.values()))
    meta_model.fit(meta_X, y_meta)
    meta_predictions = meta_model.predict(meta_X)
    return evaluate_model(y_meta, meta_predictions, meta_model_name)
```

Figure 31: Creation of the Meta Model

- **Model Evaluation**

```
# Model Evaluation
def evaluate_model(y_true, y_pred, model_name):
    accuracy = accuracy_score(y_true, y_pred)
    mae = mean_absolute_error(y_true, y_pred)
    rmse = mean_squared_error(y_true, y_pred, squared=False)
    recall = recall_score(y_true, y_pred, average='macro')
    precision = precision_score(y_true, y_pred, average='macro', zero_division=1)
    f1 = f1_score(y_true, y_pred, average='macro')
    roc_auc = plot_roc_curve(y_true, y_pred, model_name)
    print(f'{model_name} - Accuracy: {accuracy}, MAE: {mae}, RMSE: {rmse}, Recall: {recall}, Precision: {precision}, F1 Score: {f1}')
    return accuracy, mae, rmse, recall, precision, f1, roc_auc
```

Figure 32: Model Evaluation

- **Prediction of the Individual model and Evaluation of Crop and Fertilizer**

```
# Random Forest
rf_model = RandomForestClassifier(n_estimators=50, max_depth=5, random_state=42)
print("Training Random Forest model for crop recommendation.")
rf_model.fit(X_train, y_train_crop)
rf_crop_pred = rf_model.predict(X_test)
evaluate_model(y_test_crop, rf_crop_pred, 'Random Forest - Crop Recommendation')
```

```

# Decision Tree
dt_model = DecisionTreeClassifier(max_depth=5, random_state=42)
print("Training Decision Tree model for crop recommendation.")
dt_model.fit(X_train, y_train_crop)
dt_crop_pred = dt_model.predict(X_test)
evaluate_model(y_test_crop, dt_crop_pred, 'Decision Tree - Crop Recommendation')

# K-Nearest Neighbors
knn_model = KNeighborsClassifier(n_neighbors=5)
print("Training K-Nearest Neighbors model for crop recommendation.")
knn_model.fit(X_train, y_train_crop)
knn_crop_pred = knn_model.predict(X_test)
evaluate_model(y_test_crop, knn_crop_pred, 'K-Nearest Neighbors - Crop Recommendation')

# Multi-Layer Perceptron
mlp_model = MLPClassifier(hidden_layer_sizes=(50,), max_iter=1000, random_state=42)
print("Training Multi-Layer Perceptron model for crop recommendation.")
mlp_model.fit(X_train, y_train_crop)
mlp_crop_pred = mlp_model.predict(X_test)
evaluate_model(y_test_crop, mlp_crop_pred, 'Multi-Layer Perceptron - Crop Recommendation')

```

Figure 33: Training the Prediction individual models and Evaluation for the Crop Recommendation

```

print("Training Random Forest model for fertilizer recommendation.")
rf_model.fit(X_train_fert, y_train_fert)
rf_fert_pred = rf_model.predict(X_test_fert)
evaluate_model(y_test_fert, rf_fert_pred, 'Random Forest - Fertilizer Recommendation')

print("Training Decision Tree model for fertilizer recommendation.")
dt_model.fit(X_train_fert, y_train_fert)
dt_fert_pred = dt_model.predict(X_test_fert)
evaluate_model(y_test_fert, dt_fert_pred, 'Decision Tree - Fertilizer Recommendation')

print("Training Multi-Layer Perceptron model for fertilizer recommendation.")
mlp_model.fit(X_train_fert, y_train_fert)
mlp_fert_pred = mlp_model.predict(X_test_fert)
evaluate_model(y_test_fert, mlp_fert_pred, 'Multi-Layer Perceptron - Fertilizer Recommendation')

```

Figure 34: Training and Prediction the individual models and Evaluation for the Fertilizer Recommendation

- **Stacking Base Model Prediction**

```

# Model Prediction of the Base Model
crop_predictions = {}
fertilizer_predictions = {}

for model_name, model in crop_models.items():
    crop_predictions[model_name] = model.predict(X_test)

for model_name, model in fertilizer_models.items():
    fertilizer_predictions[model_name] = model.predict(X_test_fert)

```

Figure 35: Prediction of the Base Models

- Stacking Meta Model Prediction of Crop and Fertilizer

```
# Case 1: Random Forest as meta model
meta_model_rf = RandomForestClassifier(n_estimators=50, max_depth=5, random_state=42)
evaluate_stacking_model(crop_predictions, y_test_crop, meta_model_rf, 'Crop Recommendation - Random Forest as Meta Model')
evaluate_stacking_model(fertilizer_predictions, y_test_fert, meta_model_rf, 'Fertilizer Recommendation - Random Forest as Meta Model')

# Case 2: Decision Tree as meta model
meta_model_dt = DecisionTreeClassifier(max_depth=5, random_state=42)
evaluate_stacking_model(crop_predictions, y_test_crop, meta_model_dt, 'Crop Recommendation - Decision Tree as Meta Model')
evaluate_stacking_model(fertilizer_predictions, y_test_fert, meta_model_dt, 'Fertilizer Recommendation - Decision Tree as Meta Model')

# Case 3: KNN as meta model
meta_model_knn = KNeighborsClassifier(n_neighbors=5)
evaluate_stacking_model(crop_predictions, y_test_crop, meta_model_knn, 'Crop Recommendation - KNN as Meta Model')
evaluate_stacking_model(fertilizer_predictions, y_test_fert, meta_model_knn, 'Fertilizer Recommendation - KNN as Meta Model')

# Case 4: Neural Network as meta model
meta_model_nn = MLPClassifier(hidden_layer_sizes=(50, 50), max_iter=1000, random_state=42)
evaluate_stacking_model(crop_predictions, y_test_crop, meta_model_nn, 'Crop Recommendation - Neural Network as Meta Model')
evaluate_stacking_model(fertilizer_predictions, y_test_fert, meta_model_nn, 'Fertilizer Recommendation - Neural Network as Meta Model')
```

Figure 36: Stacking Model of Crop and Fertilizer Recommendation

- Hyperparameter Tuning

```
#Tuning of the Hyperparameter
param_grids = {
    'rf': {
        'n_estimators': [50, 100, 200],
        'max_depth': [5, 10, 20]
    },
    'dt': {
        'max_depth': [5, 10, 20],
        'min_samples_split': [2, 5, 10]
    },
    'knn': {
        'n_neighbors': [3, 5, 7, 9],
        'weights': ['uniform', 'distance']
    },
    'nn': {
        'hidden_layer_sizes': [(50, 50), (100, 100), (50, 50)],
        'max_iter': [1000, 2000],
        'activation': ['relu', 'tanh']
    }
}
models = ['rf', 'dt', 'knn', 'nn']

# Finding the best parameters for the tuned models
tuned_models = {}
for name, model in models.items():
    print(f"Tuning {name} model.")
    grid_search = GridSearchCV(estimator=model, param_grid=param_grids[name], scoring='accuracy', cv=5, n_jobs=-1)
    grid_search.fit(X_train, y_train_crop)
    tuned_models[name] = grid_search.best_estimator_
    print(f"Best parameters for {name}: {grid_search.best_params_}")

Tuning rf model.
Best parameters for rf: {'max_depth': 20, 'n_estimators': 200}
Tuning dt model.
Best parameters for dt: {'max_depth': 20, 'min_samples_split': 10}
Tuning knn model.
Best parameters for knn: {'n_neighbors': 3, 'weights': 'distance'}
Tuning nn model.
Best parameters for nn: {'activation': 'tanh', 'hidden_layer_sizes': (50, 50), 'max_iter': 1000}
```

Figure 37: Hyperparameter Tuning

- Stacking the Model of the Crop and Fertilizer with the Hyperparameter Tuning

```
# Case 1: Random Forest as meta model with hypertuning
meta_model_rf = RandomForestClassifier(n_estimators=50, max_depth=5, random_state=42)
evaluate_stacking_model(crop_predictions, y_test_crop, meta_model_rf, 'Crop Recommendation - Random Forest as Meta Model')
evaluate_stacking_model(fertilizer_predictions, y_test_fert, meta_model_rf, 'Fertilizer Recommendation - Random Forest as Meta Model')

# Case 2: Decision Tree as meta model with hypertuning
meta_model_dt = DecisionTreeClassifier(max_depth=5, random_state=42)
evaluate_stacking_model(crop_predictions, y_test_crop, meta_model_dt, 'Crop Recommendation - Decision Tree as Meta Model')
evaluate_stacking_model(fertilizer_predictions, y_test_fert, meta_model_dt, 'Fertilizer Recommendation - Decision Tree as Meta Model')
```

```

# Case 3: KNN as meta model with hypertunning
meta_model_knn = KNeighborsClassifier(n_neighbors=5)
evaluate_stacking_model(crop_predictions, y_test_crop, meta_model_knn, 'Crop Recommendation - KNN as Meta Model')
evaluate_stacking_model(fertilizer_predictions, y_test_fert, meta_model_knn, 'Fertilizer Recommendation - KNN as Meta Model')

# Case 4: Neural Network as meta model with hypertunning
meta_model_nn = MLPClassifier(hidden_layer_sizes=(50,), max_iter=1000, random_state=42)
evaluate_stacking_model(crop_predictions, y_test_crop, meta_model_nn, 'Crop Recommendation - Neural Network as Meta Model')
evaluate_stacking_model(fertilizer_predictions, y_test_fert, meta_model_nn, 'Fertilizer Recommendation - Neural Network as Meta

```

Figure 38: Stacking Model of Crop and Fertilizer Recommendation with Hyperparameter Tunning