ABSTRACT

This project aims to use cutting-edge data analytics techniques to transform crop forecasts and yield estimation in Indian agriculture. Through the integration of various datasets that include crop production statistics and recommendations, we build an extensive agricultural knowledge base. This study explores the complex interrelationships between soil and environmental parameters to forecast crop types and production estimates precisely. We attain notable improvements in prediction accuracy by utilizing thorough preprocessing and rigorous modeling.

This study highlights how data-driven methods can revolutionize crop selection and production techniques. We also investigate potential future directions to improve the scalability and robustness of the model, with particular attention to resilience and sustainability in agricultural systems. This project represents a pivotal step towards harnessing the power of data science to address pressing challenges in agricultural productivity and food security.

TABLE OF CONTENTS

List of Figures

CHAPTER 1
INTRODUCTION

Agriculture plays a critical role in India's economy and society, serving as the backbone of livelihoods for millions of farmers and contributing significantly to the country's GDP. However, Indian farmers face several challenges that hinder their ability to achieve maximum crop production. Limited access to timely and accurate information on agricultural practices, resource constraints including inadequate access to modern farming inputs and equipment, climate variability leading to unpredictable weather patterns, and pest and disease outbreaks are some of the key obstacles faced by farmers.
The implementation of machine learning (ML) technologies to agriculture holds immense potential for addressing these issues. To give farmers useful insights, machine learning (ML) algorithms may examine enormous volumes of data, such as past weather patterns, soil properties, crop performance measures, and market trends. For example, ML-based predictive models can suggest the best dates to plant, types of crops to grow, and ways to distribute resources according to local environmental circumstances, all of which can increase yield and profit. Additionally, early intervention is made possible by ML-driven tools for pest and disease management, which lower crop losses and minimize the need for chemical inputs.

Precision farming—the use of resources like water, fertilizer, and pesticides with fewer resources and with less negative environmental impact—is made easier by the integration of machine learning into agriculture. Real-time crop health tracking is made possible by ML-enabled monitoring systems, which improve overall farm management practices and enable proactive decision-making through the use of IoT devices and remote sensing technologies.

The application of ML technologies in India's agricultural sector has the potential to completely transform farming methods and increase their profitability, sustainability, and efficiency. ML can enable Indian farmers to overcome current obstacles and achieve higher agricultural yields by bridging the gap between traditional knowledge and contemporary advances. This would improve food security, economic growth, and the general well-being of rural communities. Machine learning technologies have the potential to have a profoundly positive impact on Indian agriculture in the future.

Crop prediction offers so many advantages for effective farm management and sustainable food production, that it is crucial to modern agriculture. Crop prediction helps farmers make educated judgments regarding crop selection by precisely predicting which crops are most suited for certain regions based on environmental parameters like climate, soil quality, and water availability. In the end, this optimization increases farm resilience and profitability by minimizing risks related to unfavorable growth conditions while simultaneously optimizing yields.

Improved estimation of yield is one of crop prediction's main advantages. Utilizing predictive models to anticipate yields before to planting allows farmers to allocate resources—such as water, fertilizers, and pesticides—optimally, resulting in financial savings and a smaller environmental footprint. By anticipating and preparing for weather-related hazards, pest and disease outbreaks, and market swings, crop prediction also plays a critical role in risk mitigation. Farmers are able to minimize losses and protect crops by implementing timely treatments thanks to this proactive approach.

Moreover, crop prediction maximizes resource utilization and reduces waste by adjusting inputs to individual crop requirements depending on anticipated conditions. By offering insights into market trends and demand projections, it also helps with improved market planning by empowering farmers to coordinate their marketing plans and production schedules for optimal profitability.

Crop prediction is crucial because it encourages the choice of crop varieties that are well-suited to regional conditions, lessens the need for agrochemicals, conserves water, and preserves the health of the soil. Crop prediction provides farmers with evidence-based recommendations for the best crop management practices by utilizing machine learning and data-driven insights. This enhances food security and ensures the long-term sustainability of agriculture in a changing global environment.

1.1 Objectives

Optimizing Crop Selection:

- Utilizes machine learning to analyze historical agricultural data, current weather, and soil conditions.
- Helps farmers identify the most suitable crops for their specific environment and region.
- Aims to enhance crop yields and financial outcomes.

Precise Yield Estimation:

- Provides accurate yield predictions before planting.
- Assists farmers in resource planning (water, fertilizer, pesticides) to increase productivity and reduce costs.

- Essential for risk reduction by forecasting potential challenges (e.g., weather, pests, market shifts).
- Enables proactive interventions to protect crops and minimize losses.

Maximizing Agricultural Resource Use:

- Tailors inputs to the specific needs of crops based on predicted conditions.
- Promotes sustainable farming practices by reducing resource waste.
- Contributes to environmental conservation.

Enhancing Agricultural Sustainability:

- Supports food security through increased agricultural productivity, resilience, and efficiency.
- Integrates cutting-edge technologies like machine learning and data analytics.
- Provides valuable insights and tools to optimize farming practices for a sustainable future.

1.2 Background and Literature Survey

S.NO	TITLE	TECHNIQUES USED	FINDINGS
1.	Crop prediction using predictive analysis	Machine learning	It provides insight about the crop prediction and also gives measures to increase the soil fertility.
2.	Crop prediction using machine learning approaches	SVM	It provides crop, required nutrients to add up, required seeds for cultivation, expected yield and market price.
3.	Soil analysis and crop recommendation using machine learning	CNN and Random Forest	By analyzing the region, soil type, yield, selling price the crop is predicted using CNN and random forest.

4.	Crop prediction using machine learning	KNN, Decision tree, Random Forest	Random forest works efficient among three algorithms. KNN works least as compared to other two algorithms.
5.	Crop yield prediction using ml: a systematic literature	Deep learning	Comparative study of deep learning algorithms. CNN, LSTM, and DNN are best among all algorithms.
6.	Machine learning enabled iot system for soil nutrients monitoring and crop recommendation	Cat boot classifier and random forest	Combining iot with ml helps in predicting the crop better as iot devices give sensor data collection with helps in accurate prediction.
7.	Comparative analysis of different machine learning prediction models for seasonal rainfall and crop production in cultivation.	Decision tree	It predicts the crop using the rainfall data. The regressor predicting models are constructed to predict the seasonal rainfall and crop yield.

1.3 Organization of the Report

The remaining chapters of the project report are described as follows:

- Chapter 2 contains the proposed system, methodology, implementation details.
- Chapter 3 gives the cost involved in the implementation of the project.
- Chapter 4 discusses the results obtained after the project was implemented.
- Chapter 5 concludes the report.
- Chapter 6 consists of codes.
- Chapter 7 gives references.

CHAPTER 2

WEATHER-BASED CROP PREDICTION

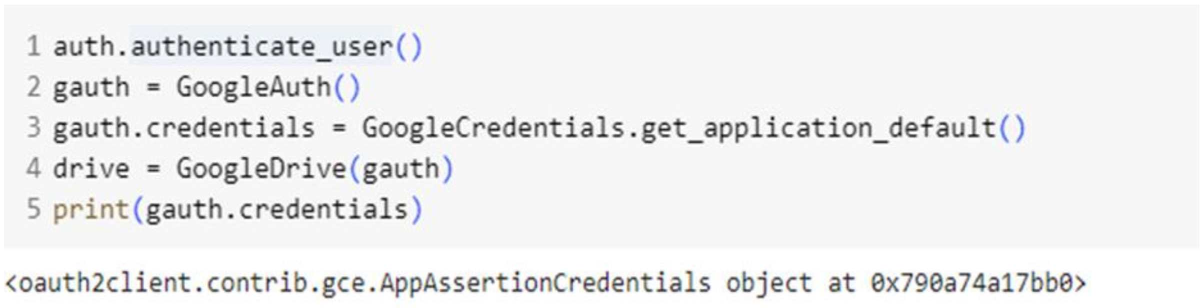
This Chapter describes the proposed system, working methodology, and implementation details.

2.1 Proposed System

Using structured machine learning techniques, the crop prediction methodology makes informed decisions about crop selection and production estimation based on agricultural parameters. First, databases containing data on nitrogen, phosphorus, and potassium levels, as well as temperature, humidity, pH, rainfall, and relevant crop names, are loaded and cleaned. In order to generate a comprehensive dataset for analysis, data integration is carried out by combining datasets according to the Crop column.

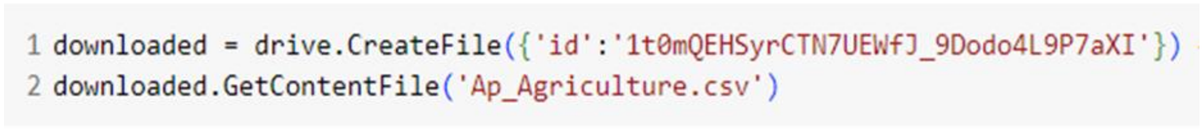
2.2 Working Methodology

The first step is to import the required libraries, which include scikit-learn modules for machine learning tasks. Here, the code imports a number of well-known Python libraries:
Pandas: A robust library for data manipulation that offers the methods and data structures required to easily work with structured data.
Mathplotlib.pyplot: A charting library for making static, animated, and interactive visualisations.
Pickle: Pickle is a Python object structure serialisation and deserialisation package that can be used to store machine learning models.

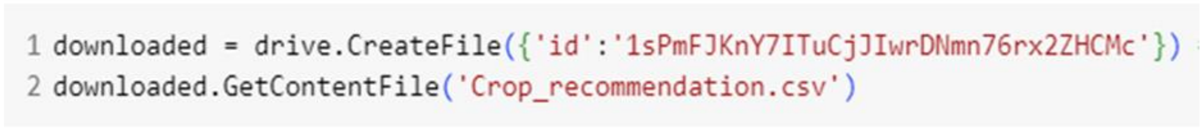


```
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
print(gauth.credentials)
```

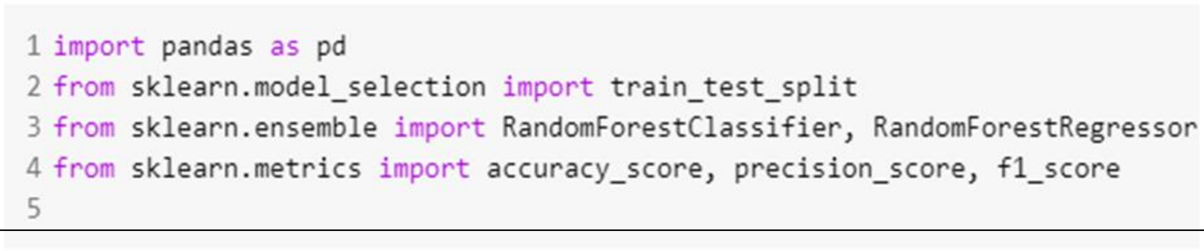
```
<oauth2client.contrib.gce.AppAssertionCredentials object at 0x790a74a17bb0>
```



```
downloaded = drive.CreateFile({'id':'1t0mQEHSyrCTN7UEWfJ_9Dodo4L9P7aXI'})
downloaded.GetContentFile('Ap_Agriculture.csv')
```



```
downloaded = drive.CreateFile({'id':'1sPmFJKnY7ITuCjJIwrDNmn76rx2ZHCMc'})
downloaded.GetContentFile('Crop_recommendation.csv')
```



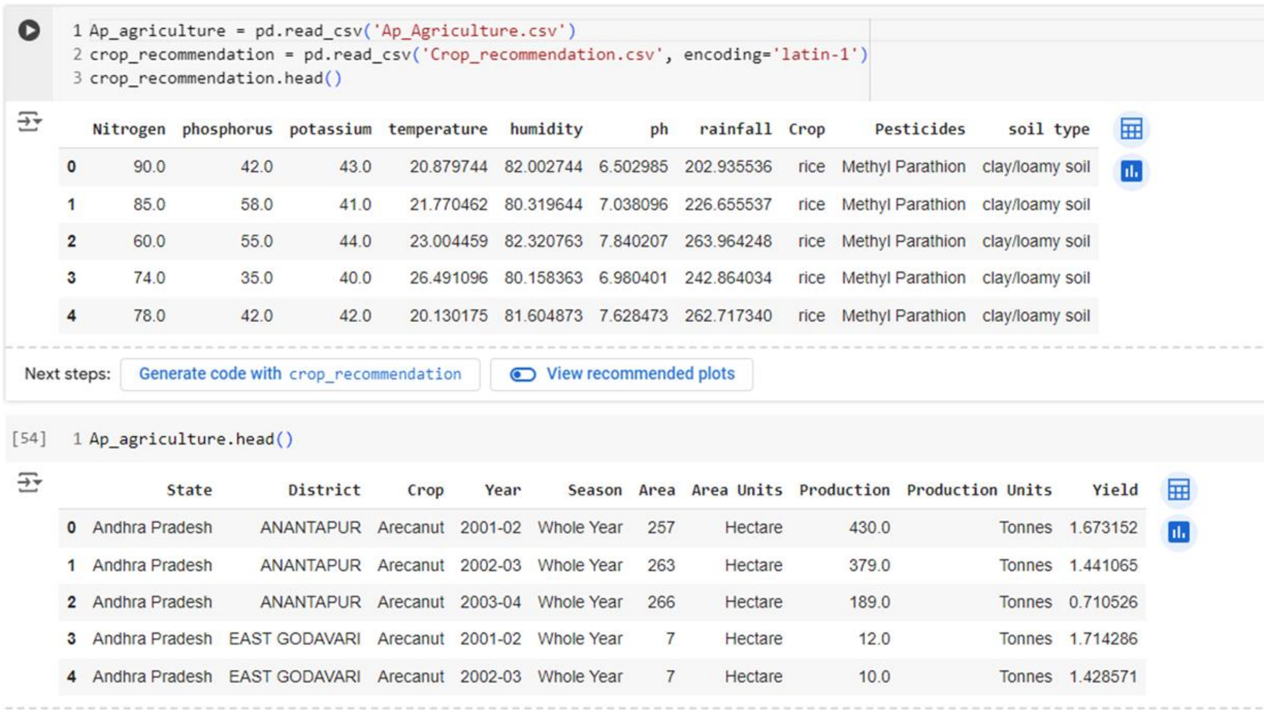
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.metrics import accuracy_score, precision_score, f1_score

```

Scikit-learn: A machine learning package containing easy-to-use tools for data analysis and mining. Modules for metrics, model selection, and several algorithms, including Random Forest and Logistic Regression, are loaded here.
Seaborn: A matplotlib-based statistical data visualisation library that offers a high-level interface for creating visually appealing and educational statistical graphics.

The code then loads two datasets using pandas:
Ap Agriculture Crop Production: This dataset contains information about crop production statistics in India, including details such as crop yield, area of cultivation, environmental factors affecting production, state, district, year and crop.

Crop Recommendation: This dataset contains data on various crops and the conditions under which they are recommended to be grown, such as soil levels such as nitrogen, phosphorus, potassium, temperature, humidity, climate conditions, ph, rainfall, crop and other agronomic factors.



```
Ap_agriculture = pd.read_csv('Ap_Agriculture.csv')
crop_recommendation = pd.read_csv('Crop_recommendation.csv', encoding='latin-1')
crop_recommendation.head()
```

	Nitrogen	phosphorus	potassium	temperature	humidity	ph	rainfall	Crop	Pesticides	soil type
0	90.0	42.0	43.0	20.879744	82.002744	6.502985	202.935536	rice	Methyl Parathion	clay/loamy soil
1	85.0	58.0	41.0	21.770462	80.319644	7.038096	226.655537	rice	Methyl Parathion	clay/loamy soil
2	60.0	55.0	44.0	23.004459	82.320763	7.840207	263.964248	rice	Methyl Parathion	clay/loamy soil
3	74.0	35.0	40.0	26.491096	80.158363	6.980401	242.864034	rice	Methyl Parathion	clay/loamy soil
4	78.0	42.0	42.0	20.130175	81.604873	7.628473	262.717340	rice	Methyl Parathion	clay/loamy soil

```
[54] Ap_agriculture.head()
```

	State	District	Crop	Year	Season	Area	Area Units	Production	Production Units	Yield
0	Andhra Pradesh	ANANTAPUR	Arecanut	2001-02	Whole Year	257	Hectare	430.0	Tonnes	1.673152
1	Andhra Pradesh	ANANTAPUR	Arecanut	2002-03	Whole Year	263	Hectare	379.0	Tonnes	1.441065
2	Andhra Pradesh	ANANTAPUR	Arecanut	2003-04	Whole Year	266	Hectare	189.0	Tonnes	0.710526
3	Andhra Pradesh	EAST GODAVARI	Arecanut	2001-02	Whole Year	7	Hectare	12.0	Tonnes	1.714286
4	Andhra Pradesh	EAST GODAVARI	Arecanut	2002-03	Whole Year	7	Hectare	10.0	Tonnes	1.428571

The dataset is cleaned and standardized using pre-processing techniques like strip and lower.

Strip ()- Strip removes any leading and trailing whitespace from the string.

Lower ()- Converts the string to lowercase.

The 'Crop' column in both datasets is cleaned and standardized. This standardization ensures that the crop names are consistent across both datasets, which is crucial for accurate merging.

Data cleaning and preparation:

Making the data uniform and analysis-ready is known as data cleaning. The 'Crop' column in this project has been standardized by deleting superfluous spaces and changing all of the names to lowercase. This makes sure that differences in naming conventions—for example, "Rice" as opposed to "rice"—don't impede the merging process. Because 'Crop' is the key needed to combine the two datasets, cleaning this column is crucial. Crops may not match accurately without this phase, leading to missing or inaccurate data after the merge.

Merging Datasets:
Handling data from many sources frequently involves the effort of merging datasets. The datasets for crop production and crop recommendation are combined using the pd.merge() tool. The algorithm guarantees that only crops that are included in both datasets are included in the merged dataset by merging on the 'Crop' column and utilizing an inner join. Ensuring that the study is limited to crops for which production statistics and recommendation data are available is crucial.

After cleaning, the datasets are merged on the 'Crop' column. The two datasets are combined using pd.merge() on crop column. The inner join function is used to join the datasets. Inner join function specifies that only the rows with matching values in both datasets should be included in the resulting data frame.

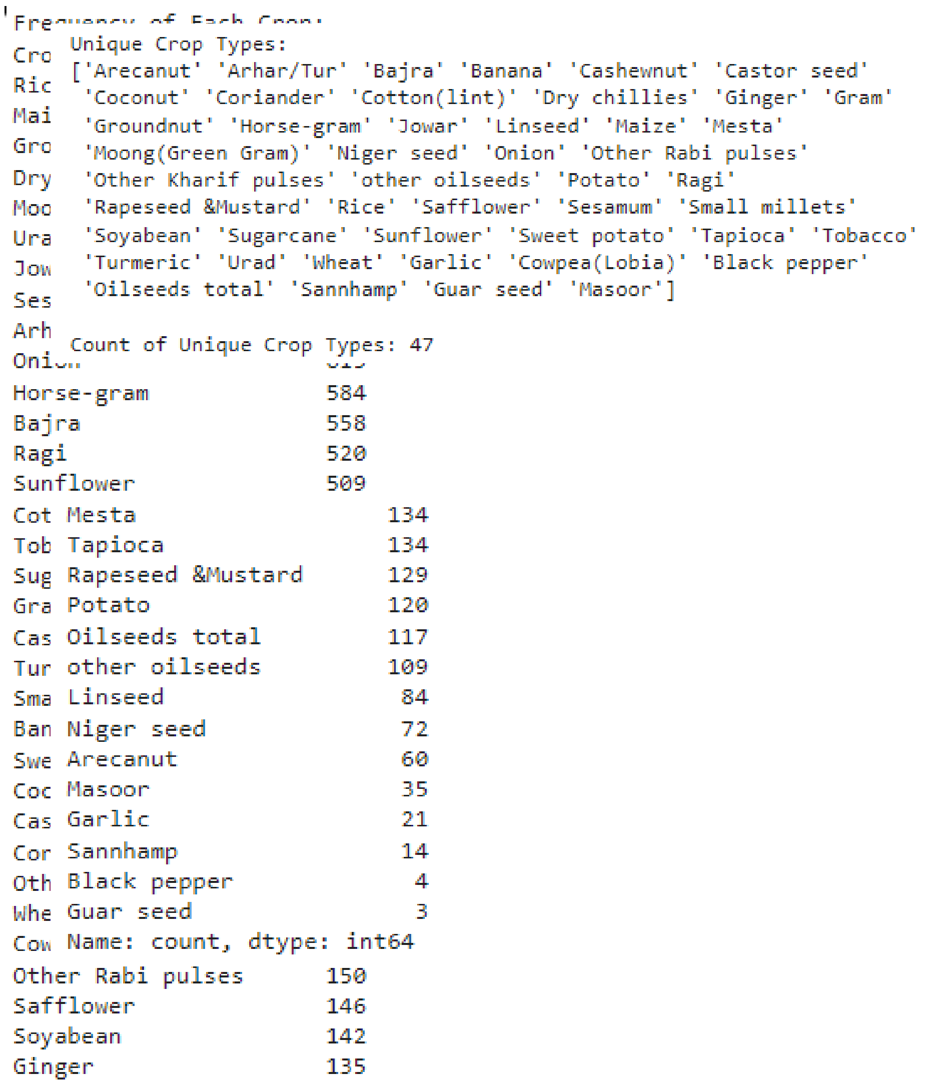
Handling Missing Data:

One simple way to deal with missing data is to remove rows that have missing values. Although there are more advanced techniques (like imputation) to deal with missing values, removing rows guarantees the accuracy and completeness of the remaining data. This stage aids in preserving the analysis's integrity because results that contain missing values may be skewed or incorrect.

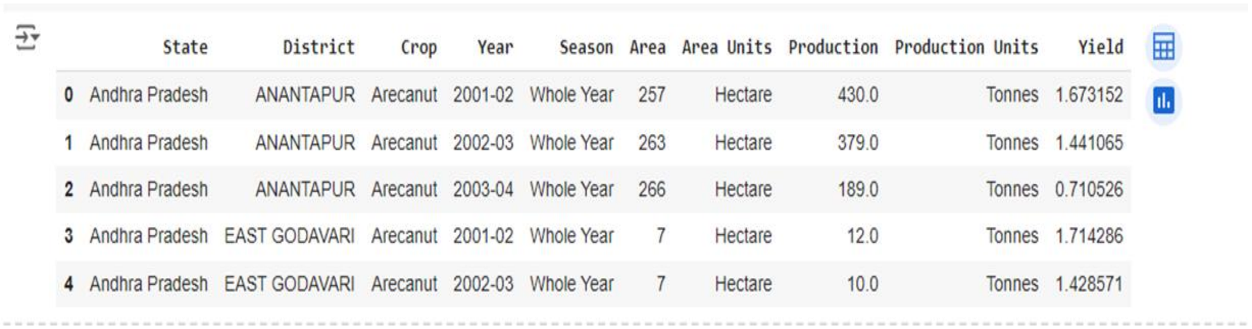
To maintain data quality, rows with missing values are removed from the combined dataset. Dropna() function is used to drop the missing values and it modifies the original data frame directly.

Exploratory Data Analysis (EDA):

Understanding the data requires completing the EDA phase. In addition to metrics like mean, standard deviation, and percentiles that aid in comprehending the data's distribution and central characteristics, the describe() function offers a summary of the dataset. Value_counts() gives the frequency of each crop, providing information about which crops are most prevalent in the dataset. The unique() method is used to identify the distinct crop types contained in the dataset.



```
Frequency of Each Crop:
Cro  Unique Crop Types:
Ric  ['Arecanut' 'Arhar/Tur' 'Bajra' 'Banana' 'Cashewnut' 'Castor seed'
Mai   'Coconut' 'Coriander' 'Cotton(lint)' 'Dry chillies' 'Ginger' 'Gram'
Gro   'Groundnut' 'Horse-gram' 'Jowar' 'Linseed' 'Maize' 'Mesta'
Dry   'Moong(Green Gram)' 'Niger seed' 'Onion' 'Other Rabi pulses'
Moo   'Other Kharif pulses' 'other oilseeds' 'Potato' 'Ragi'
Ura   'Rapeseed &Mustard' 'Rice' 'Safflower' 'Sesamum' 'Small millets'
Jow   'Soyabean' 'Sugarcane' 'Sunflower' 'Sweet potato' 'Tapioca' 'Tobacco'
Ses   'Turmeric' 'Urad' 'Wheat' 'Garlic' 'Cowpea(Lobia)' 'Black pepper'
      'Oilseeds total' 'Sannhamp' 'Guar seed' 'Masoor']
Arh
Oni  Count of Unique Crop Types: 47
Horse-gram               584
Bajra                    558
Ragi                     520
Sunflower                509
Cot Mesta                    134
Tob Tapioca                  134
Sug Rapeseed &Mustard        129
Gra Potato                   120
Cas Oilseeds total           117
Tur other oilseeds           109
Sma Linseed                   84
Ban Niger seed                72
Swe Arecanut                  60
Coc Masoor                    35
Cas Garlic                    21
Cor Sannhamp                  14
Oth Black pepper               4
Whe Guar seed                  3
Cow Name: count, dtype: int64
Other Rabi pulses        150
Safflower                146
Soyabean                 142
Ginger                   135
```



	State	District	Crop	Year	Season	Area	Area Units	Production	Production Units	Yield
0	Andhra Pradesh	ANANTAPUR	Arecanut	2001-02	Whole Year	257	Hectare	430.0	Tonnes	1.673152
1	Andhra Pradesh	ANANTAPUR	Arecanut	2002-03	Whole Year	263	Hectare	379.0	Tonnes	1.441065
2	Andhra Pradesh	ANANTAPUR	Arecanut	2003-04	Whole Year	266	Hectare	189.0	Tonnes	0.710526
3	Andhra Pradesh	EAST GODAVARI	Arecanut	2001-02	Whole Year	7	Hectare	12.0	Tonnes	1.714286
4	Andhra Pradesh	EAST GODAVARI	Arecanut	2002-03	Whole Year	7	Hectare	10.0	Tonnes	1.428571

Feature Extraction:

An essential component of any machine learning process is feature extraction. Here, soil nutrients (nitrogen, phosphorus, and potassium), meteorological factors (temperature, humidity, and rainfall), and soil pH are the characteristics used to classify crops. These characteristics were selected for their applicability and possible influence on crop growth. Since the cultivation area is directly correlated with the production volume, the same features are also employed to predict yield.

Target Variable Selection:

Selecting the appropriate target variable is crucial for the work of predictive modelling. The crop type is the goal when it comes to crop classification. Since this variable is categorical, classification algorithms can use it. The production yield, a continuous quantity suitable for regression techniques, is the goal for yield prediction.

Data Splitting:

By dividing the dataset into predictors (features) and outcomes (targets), features and targets can be extracted. Machine learning models require this division to be trained. For the classification task, X_classification and y_classification are ready in the provided code, and for the regression task, X_regression and y_regression are ready.

Correlation Analysis:

Recognising the connections between various features is made easier with the use of correlation analysis. Understanding these associations might help with feature selection and engineering procedures as highly linked features may contain redundant information. The heatmap offers a visual depiction of the correlation between features. Strong positive or negative correlations can be quickly identified and used as guidance by the data scientist to improve the feature collection.

Visualization:

Data visualization is an effective way to obtain insights. In this instance, the heatmap displays the relationship between the crop categorization features. In cases where two or more features have a strong correlation with one another, it aids in the identification of multicollinearity. Certain machine learning algorithms may perform differently as a result, especially linear models. Multicollinearity can be found and addressed to enhance the interpretability and performance of the model.

The following steps are necessary to get the data ready for machine learning models. Feature extraction ensures that the inputs to the model are pertinent. What the model learns to predict is determined by the choice of the target variable. Understanding feature associations, spotting possible multicollinearity, and guiding feature selection are all made easier with the use of correlation analysis.

Heatmap:

A heatmap is a data visualization where colors are used to represent different values. The heatmap in this instance shows the categorization features' correlation matrix. A heatmap might be useful in visualizing the relationship between the categorization features. This is an essential component of exploratory data analysis (EDA), which aids in figuring out how various attributes relate to one another.

It is essential to comprehend these relationships for several reasons:

Identifying Multicollinearity:

A significant correlation between two or more features is known as multicollinearity. This can lead to instability in the coefficient estimations, which can be troublesome for some machine learning algorithms, especially those involving linear models. To lessen multicollinearity, one can choose to eliminate or combine features by selecting pairings of features that have a high correlation.

Feature Selection:

Selecting characteristics can be influenced by knowledge about feature correlation. Predictive models work best with features that have a strong correlation with the target variable rather than with one another. By assisting in the identification of certain qualities, the heatmap facilitates the selection process.

Model Interpretation:

Understanding the connections between various features is possible through correlation analysis, which is helpful when evaluating the model's output. When two features exhibit strong correlation, for instance, they may capture identical information, and the combined effect of these features can be interpreted appropriately.

Creating the Heatmap:

The heatmap's figure size is set using figsize(). The figure's height and width are expressed in inches using the figsize option. This guarantees that the heatmap is sufficiently large for easy reading and comprehension.

The heatmap is produced by the Seaborn library's sns.heatmap() method. The pairwise correlation of the classification feature matrix is calculated using the X_classification.corr() method. The range of correlation coefficients is -1 to 1, where a perfect negative correlation is denoted by a value of -1, a perfect positive correlation by a value of 1, and no correlation is indicated by a value of 0.

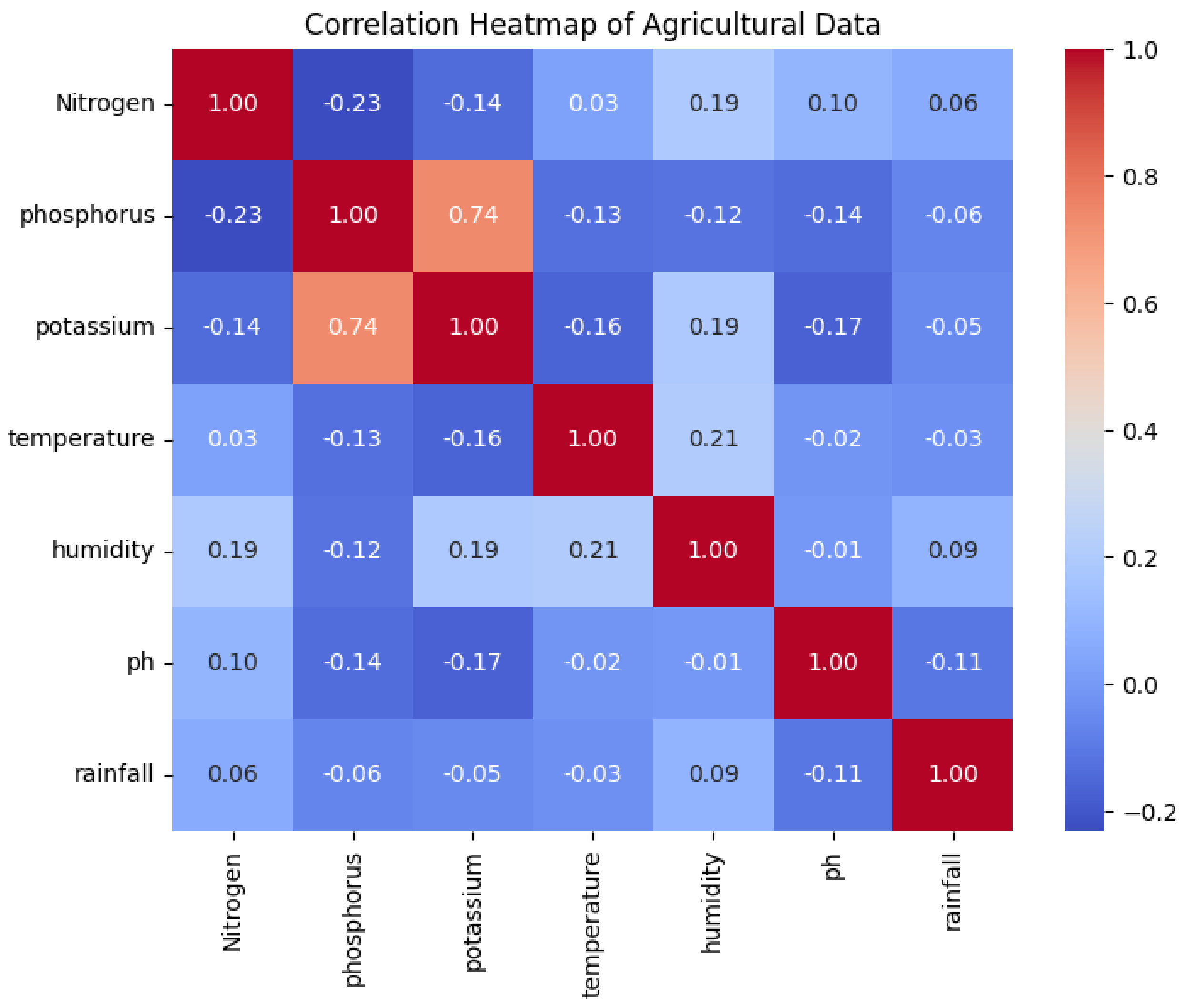


Fig.no 3.1 heatmap

Example and Interpretation of a Correlation Heatmap:

• Nitrogen **& Phosphorus (-0.23)**: There is a slight negative correlation between nitrogen and phosphorus. As the level of nitrogen increases, the level of phosphorus tends to decrease slightly.

• **Phosphorus & Potassium (0.74)**: There is a strong positive correlation between phosphorus and potassium, indicating that when phosphorus levels increase, potassium levels also tend to increase significantly.

• **Temperature & Humidity (0.21)**: There is a weak positive correlation between temperature and humidity. This suggests that as temperature increases, humidity might slightly increase as well.

• **PH & Rainfall (-0.11)**: The correlation between pH and rainfall is weakly negative, indicating that higher rainfall might be slightly associated with lower pH levels.

• **Potassium & Phosphorus (0.74)**: As mentioned earlier, this is a strong positive correlation, reinforcing that potassium and phosphorus levels are closely related.

Classification features:

It produces several subplots, each of which displays the distribution of a different characteristic that is used to classify crops. A histogram showing the distribution of values for that feature across the dataset is shown in each subplot.

Comprehending the feature distribution is essential for multiple reasons:

Understanding Data:

Feature distribution visualization sheds light on the characteristics of the data. It assists in locating outliers, skewed distributions, and other odd patterns that might need more research.

Feature Engineering:

Using an understanding of feature distributions will help with feature engineering. To make features with skewed distributions more regularly distributed, transformations such the logarithmic or Box-Cox transformations may be applied.

Model Selection and Evaluation:

The distribution of characteristics is assumed differently by various machine learning techniques. Comprehending feature distributions facilitates the process of choosing suitable models and assessing their efficacy. Understanding feature distributions helps in selecting models.

Finding Anomalies:

•**Data Preparation**: The `data` dictionary is a placeholder; replace it with your actual data.

•**Features List**: The `features` list contains the column names of the dataset.

•**Subplot Creation**: The loop iterates over the features, creating a subplot for each one.

- plt.subplot(4, 2, i + 1) creates a 4x2 grid layout for the subplots.
- sns.distplot(data[feat], color='greenyellow', kde=True, hist=True) plots the distribution of each feature with a histogram and a kernel density estimate.

•**Titles**: Depending on the index, the title of each subplot is set to either "Ratio of [feature]" or "Distribution of [feature]."

•**Layout Adjustment**: `plt.tight_layout()` ensures the subplots don't overlap, making the figure clearer.

•

Grid Lines: `plt.grid()` adds grid lines to each subplot for better readability.

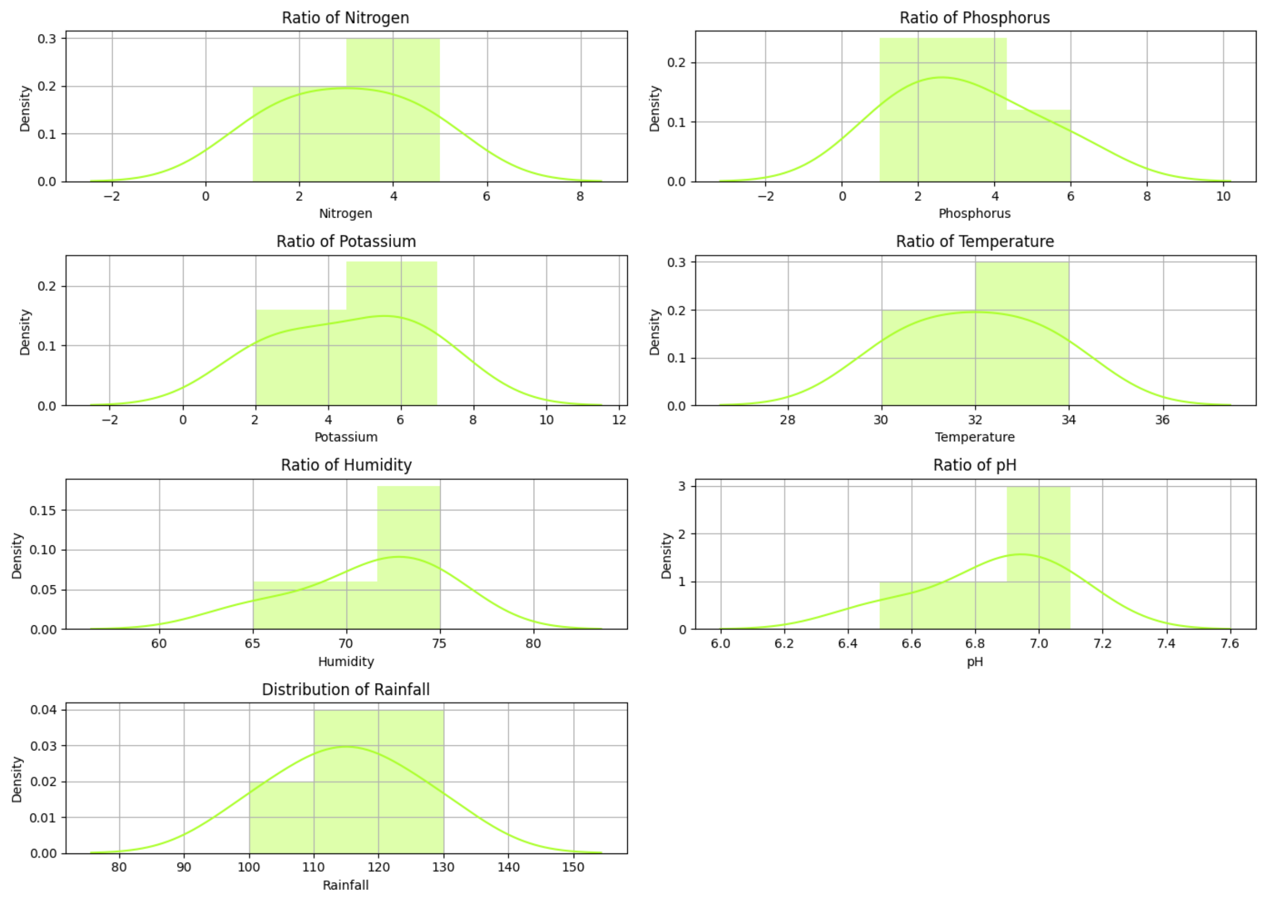


Fig.no 3.2 Feature distribution

For example, if the Nitrogen distribution has a long tail toward higher values, this could mean that some crops have unusually high nitrogen levels, which could point to outliers or particular situations that require attention. On the other hand, a balanced distribution of temperature values throughout the dataset is indicated if the "Temperature" distribution is roughly normal.

With everything considered, this graphic offers a thorough summary of the characteristics that are distributed between crops. All throughout the data analysis and modeling process, it aids in comprehending the data, seeing possible problems, and coming to well-informed conclusions.

Data Splitting:

It offers an organized approach to creating, training, and assessing machine learning models for use in agriculture. The code guarantees the robustness and dependability of the models by dividing the data into training and testing sets, training Random Forest models, Naïve Bayes and KNN and assessing their performance using pertinent metrics. The models may be used practically thanks to the prediction functions, which give users the ability to base judgements on predictions and make educated choices.

train_test_split:

This scikit-learn function divides the dataset into subsets for testing and training. Eighty per cent of the data is used for training the model, and twenty per cent is set aside for testing, according to the test_size=0.2 parameter. The reproducible split is guaranteed by the random_state=42 argument, which uses the same data for testing and training each time the code is executed. This facilitates consistent results and is necessary for proper model performance comparison.

Training Models:

The model is designed using Random Forest, Naive Bayes and KNN algorithms. The model is designed using random forest and a comparative analysis is done using KNN and naive Bayes algorithms.

RANDOM FOREST ALGORITHM:

For classification and regression tasks, respectively, the scikit-learn library offers two strong and adaptable algorithms: RandomForestClassifier and RandomForestRegressor. The Random Forest methodology serves as the foundation for both. During training, it builds a large number of decision trees and produces a class that is either the mean prediction for regression tasks or the mode of the classes for classification tasks. The ideas, benefits, and real-world uses of RandomForestClassifier and RandomForestRegressor will be discussed in this report.

RandomForestClassifier:

When the goal of a classification task is to place input data into predetermined classifications, the RandomForestClassifier is employed. Here's a thorough examination of its main attributes and capabilities:

Ensemble Learning:

Using a random subset of the training set, the RandomForestClassifier generates an ensemble of decision trees. It makes use of a method known as bagging (Bootstrap Aggregating), in which every tree is trained using a bootstrap sample of the data—a random sample with replacement.

Feature Randomness:

In order to find the optimal split at each node, RandomForestClassifier randomly chooses a subset of features throughout the building of each tree. This enhances the generalizability of the model and lowers the correlation between trees.

Voting mechanism:

Every tree in the forest produces a class prediction for classification tasks; majority vote determines the final prediction. This indicates that the final output is determined by selecting the class that gets the most votes out of all the trees.

Hyperparameters:

n_estimators: The number of trees in the forest, n_estimators. While adding more trees usually boosts performance, the computational cost does as well.

max_depth: The trees' deepest point. Overfitting can be avoided by restricting the depth of the trees.

random_state: Regulates the randomness to guarantee reproducibility of outcomes.

Benefits:

Robustness against Overfitting: The model is robust and generalizable since randomization is used in both feature selection and data sampling to assist prevent overfitting.

Missing Values: Random forests are capable of preserving accuracy even in the presence of missing values in the dataset. A feature's estimated importance is provided, which aids in feature selection and the comprehension of how each feature affects the prediction.

RandomForestRegressor:

For regression applications where the objective is to predict a continuous output variable, the RandomForestRegressor is utilised. With a few modifications that make it better suitable for regression, it operates similarly to the RandomForestClassifier:

Ensemble Learning:

Similar to the classifier, the RandomForestRegressor uses bootstrap samples of the data to construct an ensemble of decision trees.

Feature Randomness:

At each node, it randomly chooses a subset of features to split, which lowers variance and boosts generalizability.

Averaging Mechanism:

The average of all the trees in the forest's predictions is the final result for regression tasks. This averaging raises accuracy while lowering variance.

Hyper-parameters:

The RandomForestRegressor, like the classifier, uses n_estimators, max_depth, and random_state to regulate the process's unpredictability, depth, and number of trees, respectively.

Benefits:

High Accuracy: High prediction accuracy can be attained by averaging the data from several trees.

Robustness to Outliers: Because it aggregates numerous tree projections, it can effectively manage outliers.

Non-Linear Relationships: Able to record non-linear connections between the target variable and characteristics.

1. Versatility and Robustness: RandomForestClassifier and RandomForestRegressor are powerful ensemble learning algorithms based on decision trees, suitable for both classification and regression tasks.

2. Ensemble Learning: These algorithms use multiple decision trees (e.g., `n_estimators=50`) to average the results, enhancing model accuracy and robustness.

3. Overfitting Control: The parameter `max_depth=10` restricts the depth of each tree, helping to prevent overfitting by limiting model complexity.

4. Reproducibility: The `random_state=42` ensures that the model training process is reproducible, yielding consistent results across runs.

5. Application: In practical scenarios, the RandomForestClassifier predicts outcomes like crop type, while the RandomForestRegressor estimates continuous values like crop yield based on training data.

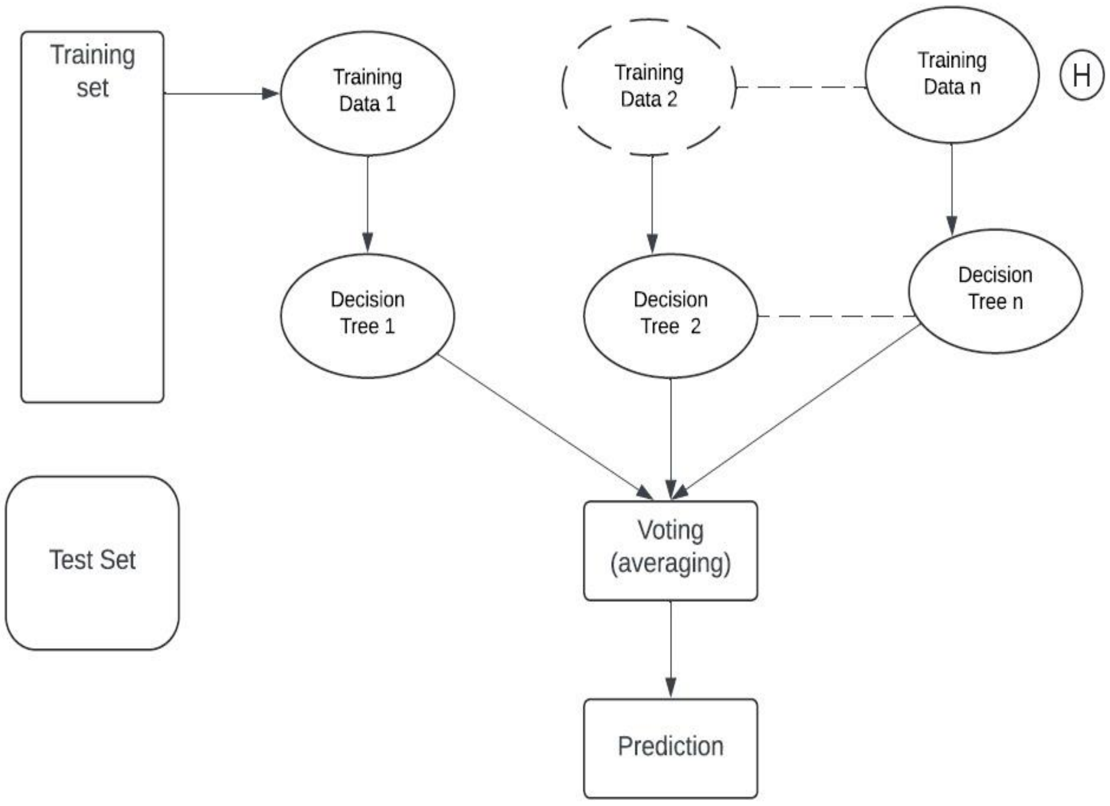


Fig.no 3.3: Flowchart of random forest

KNN:

A simple yet effective machine learning approach that can be applied to both regression and classification problems is K-Nearest Neighbours (KNN). It is an instance-based learning algorithm, which means that instead of explicitly learning a model, it makes predictions by using the full dataset.

KNN bases its predictions on the idea of identifying the data points that are closest to the input instance—known as neighbors—and drawing conclusions from them. KNN classifies an input instance by allocating to it the most common class among its k-nearest neighbours. In order to get a forecast for regression, KNN averages the values of the k-nearest neighbours.

Select the number of neighbours k: k is an important decision since it establishes the ratio of algorithmic bias to variance. significant variance can result from a small k, and significant bias can be introduced by a big k.

Determine the distance: Minkowski, Manhattan, and Euclidean distances are examples of common distance measures. It calculates the separation between the input instance and every other instance in the dataset.

Determine the k-nearest neighbours: The input instance's k-nearest neighbours are determined using the computed distances.

Forecast:

Classification: Among the k-nearest neighbours, the most frequent class is selected.

Regression: The k-nearest neighbours' average value is calculated.Select the quantity of neighbours, k: The selection of kk is an important factor since it establishes the ratio of algorithmic variance to bias. While a large kk can introduce high bias, a little kk can result in high variation.

Benefits of KNN Simplicity:

- KNN is simple to comprehend and use.
- **Flexibility:** It can be applied to challenges involving regression as well as classification.
- **No Training Phase:** Since KNN is an instance-based algorithm, it can be quickly deployed because it does not require an explicit training phase.
- **Adaptability:** Multi-class classification issues are manageable for KNN.

Disadvantages of KNN:

- KNN's computationally intensive limitations include the fact that it becomes slow for large datasets as the amount of computing needed to find the nearest neighbours increases.
- Storage Requirements: KNN may require a large amount of memory to store the whole training dataset.
- Sensitive to Irrelevant Features: Feature selection or dimensionality reduction approaches are frequently required because irrelevant features might have a negative impact on KNN.
- Selection of k: The selection of k has a significant impact on KNN performance. Usually, cross-validation is employed to find the optimal value of k.
- Unbalanced Data: Because the majority class may dominate the prediction, KNN performs poorly on unbalanced datasets.

A flexible and easy-to-understand technique, K-Nearest Neighbours performs well in a range of classification and regression scenarios. When the dataset is modest to moderate in size and the decision boundaries are irregular, it is especially helpful. This real-world example shows the entire process of creating a KNN model for crop prediction, including data preparation, model evaluation, and user interaction.

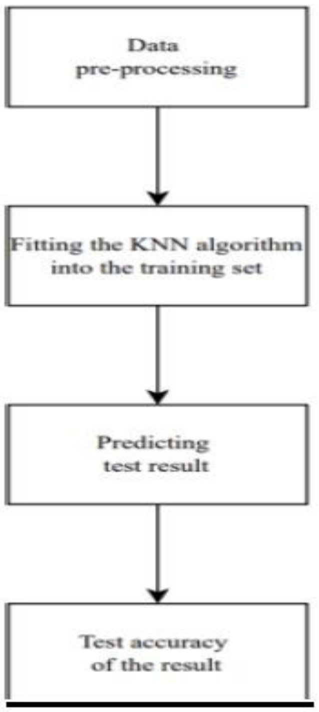

Data pre-processing
Fitting the KNN algorithm into the training set
Predicting test result
Test accuracy of the result

Fig.no 3.5. Flow chart of KNN

The process used in KNN:

Data Loading and Cleaning:

Two CSV files are loaded into pandas DataFrames at the start of the function. It eliminates any leading or trailing spaces and lowercase standardises the Crop column in both databases.

Combining Datasets:

An inner join is used to integrate the datasets on the Crop column, combining only the matching rows. To make sure the data is clean, rows with missing values are eliminated.

Data splitting and feature extraction:

The target variable and pertinent attributes are taken out of the combined dataset. Eighty percent of the data is used for training and twenty percent is used for testing.

Training the Model:

Using $k = 5$, a K-Nearest Neighbours classifier is instantiated and trained using the training set. To predict the crop based on a feature supplied by the user, a function called predict_crop is defined.

Prediction Function:

To forecast the crop based on user-provided feature values, a function called predict_crop is defined.

Model Evaluation:

Using the test data, accuracy, precision, and F1-score are used to assess the model's performance. To provide a summary of the model's efficacy, these metrics are printed.
Interactive User Input:
The user is prompted by the code to enter values for every feature. The crop type is predicted and printed using these inputs.

NAÏVE BAYES:

Based on Bayes' Theorem, the Naive Bayes classifiers are a collection of straightforward yet powerful probabilistic classifiers that make strong (naive) independence assumptions about the features. Naive Bayes classifiers are unexpectedly effective in many real-world applications, especially for text classification and other high-dimensional datasets, despite their simplicity. We'll examine the foundations, varieties, benefits, drawbacks, and real-world uses of Naive Bayes in this section.

Naive Bayes classifiers are a class of algorithms for classification that rely on Bayes' Theorem. Rather than being a single technique, it is essentially a family of algorithms founded on the idea that every pair of features that is being classed stands alone. To begin, let's consider a dataset.

One of the simplest and most effective classification algorithms, the Naïve Bayes classifier, makes it easier to quickly create machine learning models with predictive power.

Classification problems are one of the Naïve Bayes method's applications. It is commonly used in the classification of texts. Given that every word in the data constitutes a feature in text classification tasks, the data has a high dimension. It is used for tasks like spam filtering, sentiment analysis, and rating categorization. One benefit of naïve Bayes is its speed. Predictions are made quickly and easily when there are many data dimensions.

The core tenet of Naive Bayes is that every feature contributes an:

Feature independence: Depending on the class name, the data's characteristics are conditionally independent of one another.

Continuous features are normally distributed: It is presumed that a feature is normally distributed within each class if it is continuous.

Discrete features have multinomial distributions: Within each class, a discrete feature is presumed to have a multinomial distribution.

Features are equally important: It is believed that each characteristic contributes equally to the class label prediction.

No missing data: No omitted information There shouldn't be any missing values in the data.
Bayes theorem:

Bayes' Theorem finds the probability of an event occurring given the probability of another event that has already occurred. Bayes' theorem is stated mathematically as the following equation:

$P(A|B)= P(B|A)P(A)/ P(B)$

where A and B are events and $P(B) \neq 0$.

- Essentially, given that event B is true, we are attempting to determine the probability of event A. Evidence is another word for Event B.
- P(A) is the prior probability of A, or the probability of an event occurring before evidence is observed.
- An attribute value of an unidentified instance (in this case, event B) serves as the proof.
- The probability of evidence, or marginal probability, is P(B). P(A|B) represents the likelihood of an event after evidence is observed, or the a posteriori probability of B.
- The likelihood that a hypothesis will materialise in light of the evidence is represented by the probability P(B|A).

We can now apply Bayes' theorem to our dataset in the following manner:
$P(y|X)=P(X|y)P(y)P(X)P(y|X)=P(X)P(X|y)P(y)$

where X is a dependent feature vector (of size n) and y is the class variable.
$X=(x1,x2,x3,…..,xn)X=(x1,x2,x3,…..,xn)$

Advantages of Naive Bayes Classifier:

- Computationally efficient and simple to implement.
- Effective even with limited training data; useful in scenarios with a large number of characteristics.
- It functions well when categorical features are present.
- It is assumed that data for numerical features originates from normal distributions.

In real-world situations, their effectiveness, quickness, and capacity to operate with little data make them valuable, making up for their naive independence assumption.

With strong (naive) independence assumptions across the features, the Naive Bayes classifiers are a family of straightforward but powerful probabilistic classifiers based on Bayes' Theorem. In many real-world applications, Naive Bayes classifiers prove unexpectedly effective despite their simplicity, especially when it comes to text categorization and other high-dimensional datasets.

To generate a Gaussian Naive Bayes classifier model, the scikit-learn GaussianNB class is instantiated. The fit approach is used to train this classifier using the training data (X_train, y_train). A probabilistic classifier based on Bayes' theorem, Gaussian Naive Bayes assumes that the features have a Gaussian (normal) distribution.

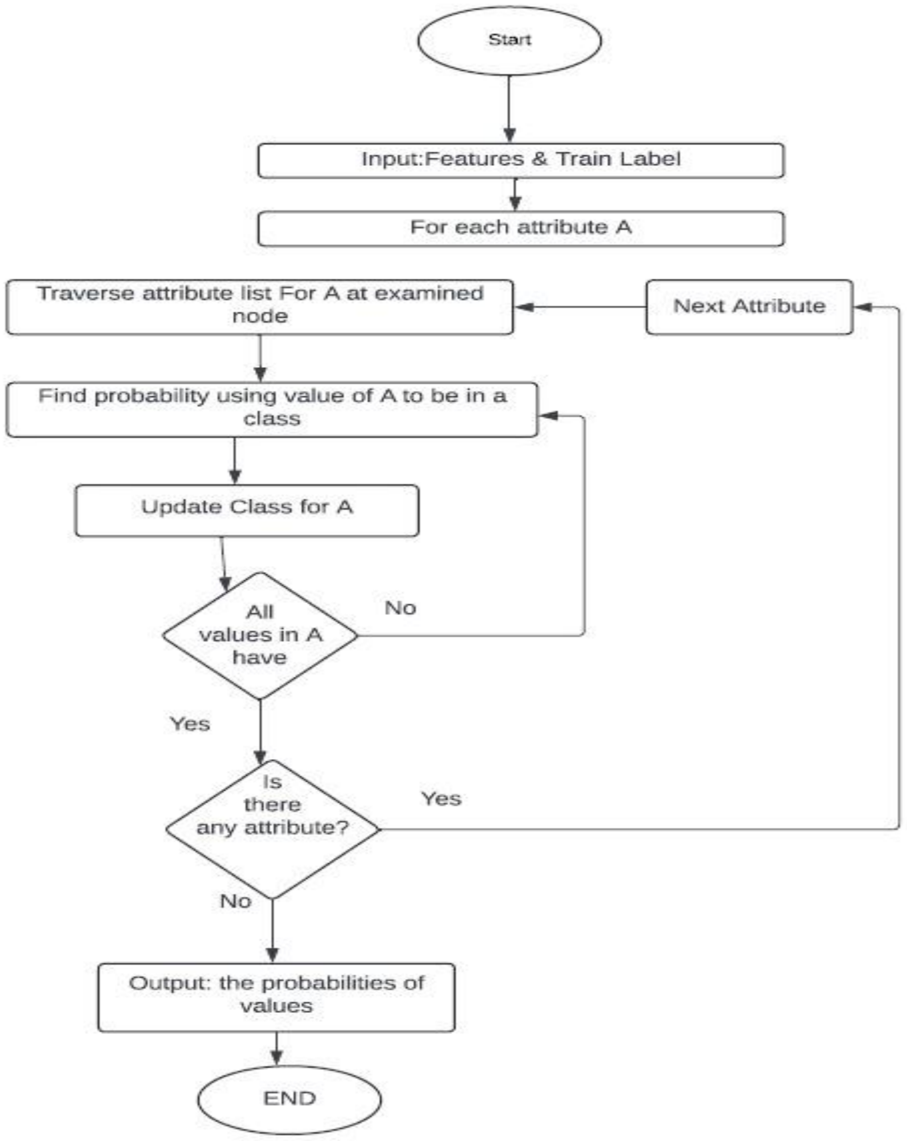

Start
Input:Features & Train Label
For each attribute A
Traverse attribute list For A at examined node
Next Attribute
Find probability using value of A to be in a class
Update Class for A
All values in A have
No
Yes
Is there any attribute?
Yes
No
Output: the probabilities of values
END

Fig.no 3.4. Flow chart of Naïve bayes

The process used in naïve bayes:

Data Loading and Cleaning:

Two CSV files are loaded into pandas DataFrames at the start of the function. It eliminates any leading or trailing spaces and lowercase standardises the Crop column in both databases.

Combining Datasets:

An inner join is used to integrate the datasets on the Crop column, combining only the matching rows. To make sure the data is clean, rows with missing values are eliminated.

Data splitting and feature extraction:

The target variable and pertinent attributes are taken out of the combined dataset. Eighty percent of the data is used for training and twenty percent is used for testing.

Training the Model:

Using the training set of data, a Gaussian Naive Bayes model is instantiated and trained.

Prediction Function:

To forecast the crop based on user-provided feature values, a function called predict_crop is defined.

Model Evaluation:

Using the test data, accuracy, precision, and F1-score are used to assess the model's performance. To provide a summary of the model's efficacy, these metrics are printed.

Interactive User Contribution:

The user is prompted by the code to enter values for every feature. The crop type is predicted and printed using these inputs.

INTERACTIVE USER INPUT:

We have added a feature of combining yield estimation, crop forecast, and customised crop management advice into an interactive user interface.
The user enters different farming characteristics required to forecast the yield and type of crop. "Nitrogen," "Phosphorus," "Potassium," "Temperature," "Humidity," "pH," and "Rainfall" are some of the characteristics. To anticipate yield, the user additionally supplies the area in hectares. A dictionary called user_input contains these inputs.

The crop type is predicted using the user inputs. To make this prediction, the method predict_crop() uses a pre-trained classification model (which may be a RandomForestClassifier or a Gaussian Naive Bayes, depending on the situation). The crop type prediction is printed.

Similar to this, the predict_yield() function calculates the anticipated production yield using the area and user inputs. Additionally, this yield is printed.

The primary function of the software is to generate customised management recommendations according to the supplied attributes and the anticipated crop type. This is accomplished via a number of operations that serve various crops, including rice, wheat, maize, chilli, and bananas. Based on the anticipated crop, the primary function manage_predicted_crop() routes the flow to particular crop management services.

Based on the input features provided, each crop management function offers particular recommendations. The manage_rice() function, for example, recommends water management, shade during hot weather, and irrigation during dry spells as ideal growing conditions for rice.

The detailed management of the crops are given below:

Rice:

- Plant rice on flooded paddy fields that have effective water management.
- Temperature: If the temperature rises above 30°C, give young rice plants some shade during periods of intense heat.
- Water management: Throughout the growing season, keep the water levels constant.
- Fertilisation: Use fertilisers based on nitrogen when the time is right.
- During dry spells, increase irrigation, especially if rainfall is less than 100 mm.

Wheat:

- Planting Season: For winter wheat, plant when the weather is suitable.
- Management of the Soil: Make sure the soil is sufficiently fertilised and well-drained.
- Soil pH: If the pH of the soil is less than 6.0, add lime to bring it up to the ideal range.
- Disease control: Keep an eye out for ailments like rust and take precautions.
- Water Management: If rainfall exceeds 200 mm, make sure appropriate drainage is in place to prevent waterlogging.

Maize:

- Management of Maize Soil: Plant in well-drained soil that has a high level of organic matter.
- Fertilisation: Use fertilisers that are balanced and contain potassium, phosphate, and nitrogen.
- Humidity: If the humidity is more than 80%, use appropriate spacing and weed control measures to avoid fungal infections.
- Watering: Keep an eye on the moisture content of the soil and apply water as needed. If there is less than 80 mm of rainfall, preserve soil moisture using mulching or drip watering.

Chilli:

- Planting Chilli: Choose warm, well-drained soil for planting.
- Support: Offer assistance to plants as they grow.
- Temperature: If the temperature rises beyond 30°C during periods of excessive heat, use shade nets or offer shading.
- Apply balanced fertilisers and keep an eye out for pests when fertilising.

Banana:

- Handle your banana soil by planting it in rich, well-drained soil.
- Fertilisation: Use fertilisers high in potassium.
- Temperature: If the temperature rises beyond 25°C, make sure you have enough water and mulch your yard.
- Plant maintenance: Use the right spacing and get rid of any old leaves.

These management suggestions give farmers customized guidance on how to maximize crop yields depending on particular crop requirements and local environmental factors. Farmers may ensure sustainable farming techniques, boost crop health, and increase output by adhering to these rules.

EVALUATION METRICS:

To comprehend and contrast the performance of machine learning models, evaluation measures are essential. A useful method for visualizing these metrics facilitates comprehension of the crop forecast model's output. Plotting the metrics makes it easy to evaluate the model's advantages and disadvantages. We used accuracy, precision, and f1-score evaluation metrics to measure the efficiency of the model.

2.3 Requirements:

Processor (CPU):
Minimum: Intel Core i5 or AMD Ryzen 5
Memory (RAM):
Minimum: 16 GB
Storage:
SSD with at least 512 GB (for fast read/write speeds)
Operating System:
Windows 10, macOS, or Linux (Ubuntu preferred for machine learning)
Platforms:
Google colab, AWS (Amazon Web Services), Google Cloud Platform (GCP), Microsoft Azure.

2.4 Dataset:

<u>Dataset 1: Crop Recommendation</u>
Total Entries: 2200
Columns: 8
Nitrogen: The nitrogen content in the soil.
Phosphorus: The phosphorus content in the soil.
Potassium: The potassium content in the soil.
Temperature: The temperature in degrees Celsius.
Humidity: The humidity percentage.
pH: The pH value of the soil.
Rainfall: The amount of rainfall in mm.
Crop: The type of crop recommended for the given conditions.
Pesticides: The pesticides used for crop.
Soil Type: The type of soil the crop grows.

<u>Dataset 2: AP Agriculture</u>
Total Entries: 16363
Columns: 10
State: The state where the data is collected.
District: The district within the state.
Crop: The type of crop cultivated.
Year: The year of data collection.
Season: The season during which the crop is grown.
Area: The area of cultivation.
Area Units: The units of the area (e.g., Hectare).
Production: The production output.
Production Units: The production units (e.g., Tonnes).
Yield: The yield of the crop.

<u>Dataset 3: State_Agri</u>
Total Entries: 67000
Columns: 10
State: The state where the data is collected. The States are 1. Karnataka, 2. Kerala, 3. Tamil Nadu.
District: The district within the states.
Crop: The type of crop cultivated.
Year: The year of data collection.
Season: The season during which the crop is grown.
Area: The area of cultivation.
Area Units: The units of the area (e.g., hectares).
Production: The production output.
Production Units: The production units (e.g., Tonnes).
Yield: The yield of the crop.

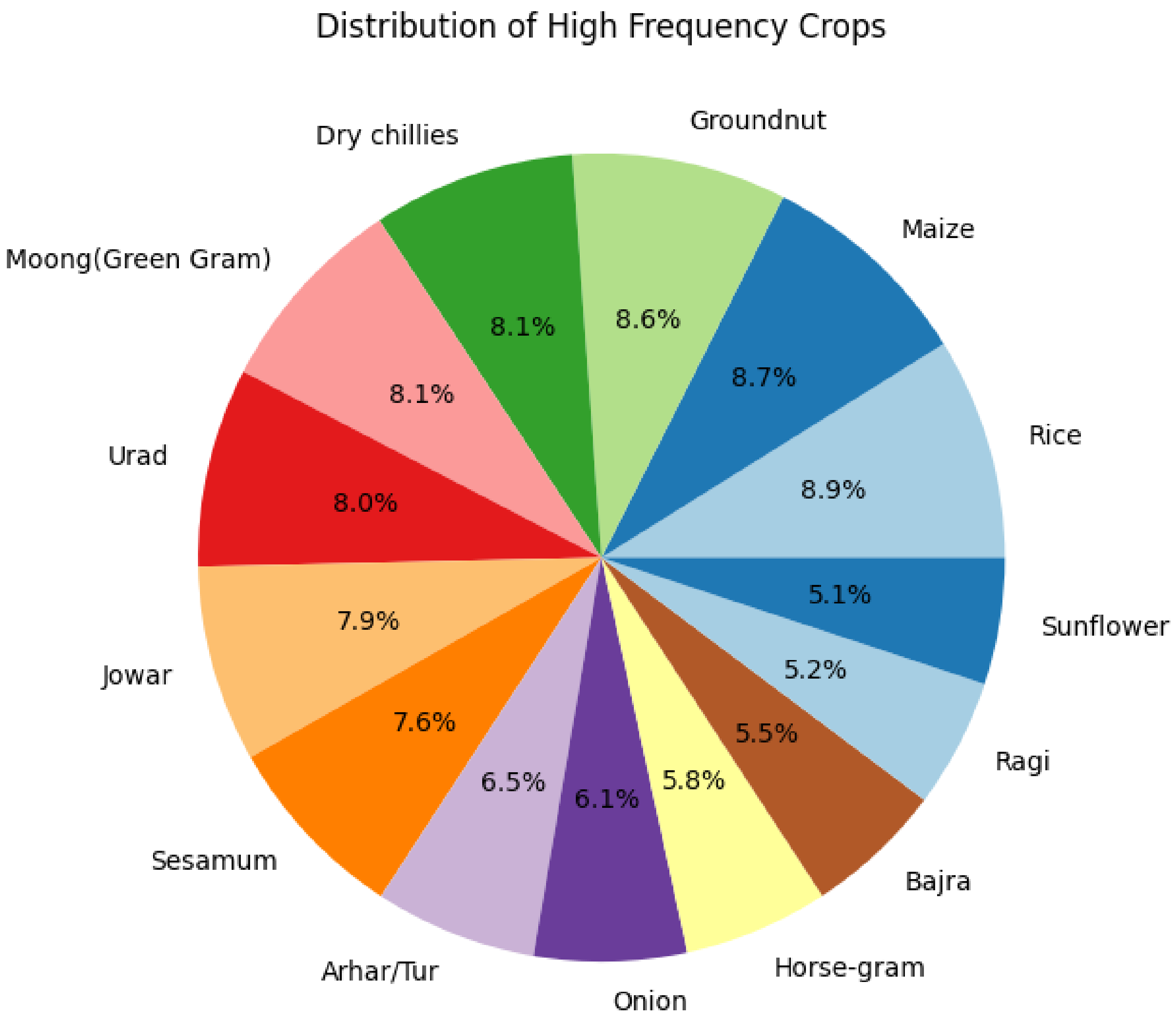
Analogy:

Aspect	**K-Nearest Neighbors (KNN)**	**Decision Tree**	**Random Forest**
Basic Principle	Classifies based on nearest neighbors (similar conditions for soil, weather, etc.)	Uses a tree structure to split the dataset by conditions like rainfall, soil nutrients	Builds multiple decision trees and averages the predictions to improve crop accuracy
Training Time	Slow (depends on dataset size)	Fast (splits data based on features like soil, temperature, rainfall)	Moderate (multiple trees built, can be improved with parallel processing)
Prediction Time	Slow (requires distance calculation for each instance)	Fast (traverses the tree for predictions)	Fast (averages predictions from multiple trees)
Handling Overfitting	High risk with large values of k	Prone to overfitting with deep trees	Less prone to overfitting due to averaging across trees
Interpretability	Low (hard to understand how neighbors impact predictions)	High (clear decision rules based on feature splits)	Moderate (difficult to interpret due to multiple trees)
Scalability	Poor (scales poorly with large datasets)	Good, but can become complex with many features	Good scalability, but resource-intensive due to building multiple trees
Handling Missing Values	Poor (requires complete data for distance calculations)	Can handle missing values by splitting on available features	Can handle missing values effectively by averaging across trees
Assumptions	No assumptions about data distribution	Assumes data can be split into distinct regions	No assumptions, suitable for diverse crop conditions and data

CHAPTER 3

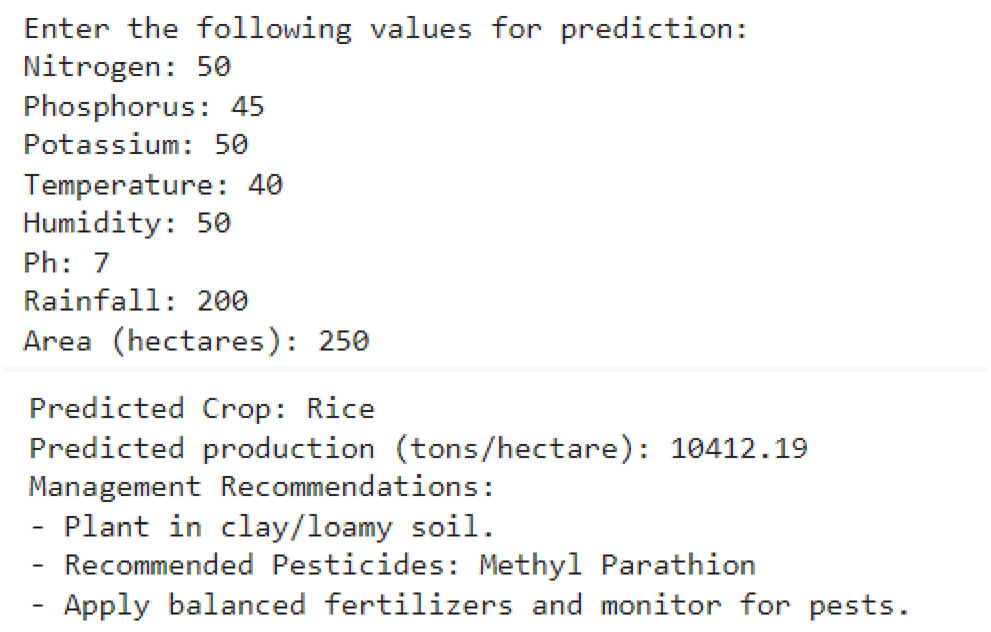
RESULTS AND DISCUSSIONS

As we have over 46 types of crops it is difficult to depict the crops in a pie chart we have taken a factor to bias the crops based on their frequency. We have drawn the piechart for the 10 crops whose frequency is over 500 and the result is as shown below



The model is built as user interactive so the model will ask the input of the user. The input values taken from the user are nitrogen, potassium, phosphorus, temperature, humidity, rainfall, pH, area for predicting crop and also how much crop will be produced. After taking the input values then the model will predict the crop and also how much crop production the user can get. Along with the prediction of crop and yield it also provides crop recommendations for each crop according to the user input.

Fig.no 4.2. Output result for Random Forest user input

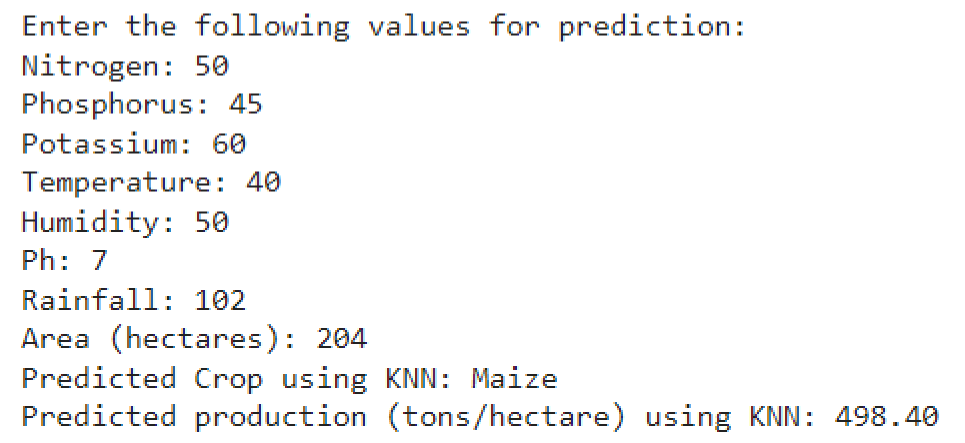


```
Enter the following values for prediction:
Nitrogen: 50
Phosphorus: 45
Potassium: 50
Temperature: 40
Humidity: 50
Ph: 7
Rainfall: 200
Area (hectares): 250
```

```
Predicted Crop: Rice
Predicted production (tons/hectare): 10412.19
Management Recommendations:
- Plant in clay/loamy soil.
- Recommended Pesticides: Methyl Parathion
- Apply balanced fertilizers and monitor for pests.
```

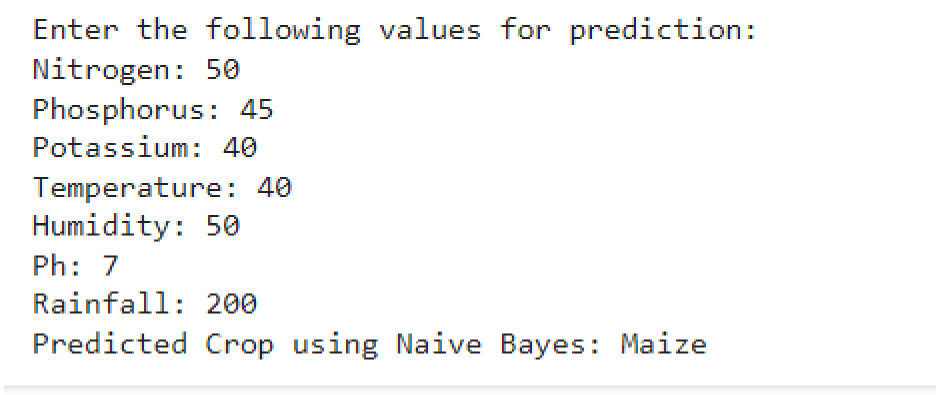
The model is also built using naïve Bayes and KNN to see which algorithm is working well for the dataset. The result for the naïve Bayes and KNN algorithm is given below.

Fig.no 4.3. Output result for KNN user input



```
Enter the following values for prediction:
Nitrogen: 50
Phosphorus: 45
Potassium: 60
Temperature: 40
Humidity: 50
Ph: 7
Rainfall: 102
Area (hectares): 204
Predicted Crop using KNN: Maize
Predicted production (tons/hectare) using KNN: 498.40
```

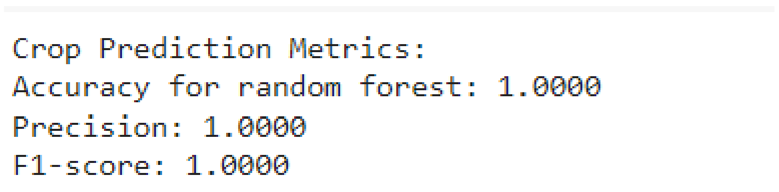
Fig.no 4.4. Output result for Naïve Bayes user input.



```
Enter the following values for prediction:
Nitrogen: 50
Phosphorus: 45
Potassium: 40
Temperature: 40
Humidity: 50
Ph: 7
Rainfall: 200
Predicted Crop using Naive Bayes: Maize
```

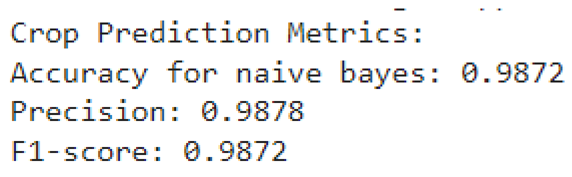
The accuracy values of all three models are calculated and the images of the accuaracies are given below.

Fig.no 4.5. Accuracy of Random Forest algorithm



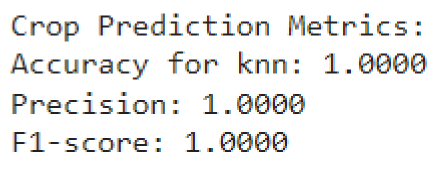
```
Crop Prediction Metrics:
Accuracy for random forest: 1.0000
Precision: 1.0000
F1-score: 1.0000
```

Fig.no 4.6. Accuracy of Naïve Bayes algorithm



```
Crop Prediction Metrics:
Accuracy for naive bayes: 0.9872
Precision: 0.9878
F1-score: 0.9872
```

Fig. no 4.7. Accuracy of KNN algorithm



```
Crop Prediction Metrics:
Accuracy for knn: 1.0000
Precision: 1.0000
F1-score: 1.0000
```

Fig.no 4.8. Accuracies of all three algorithms

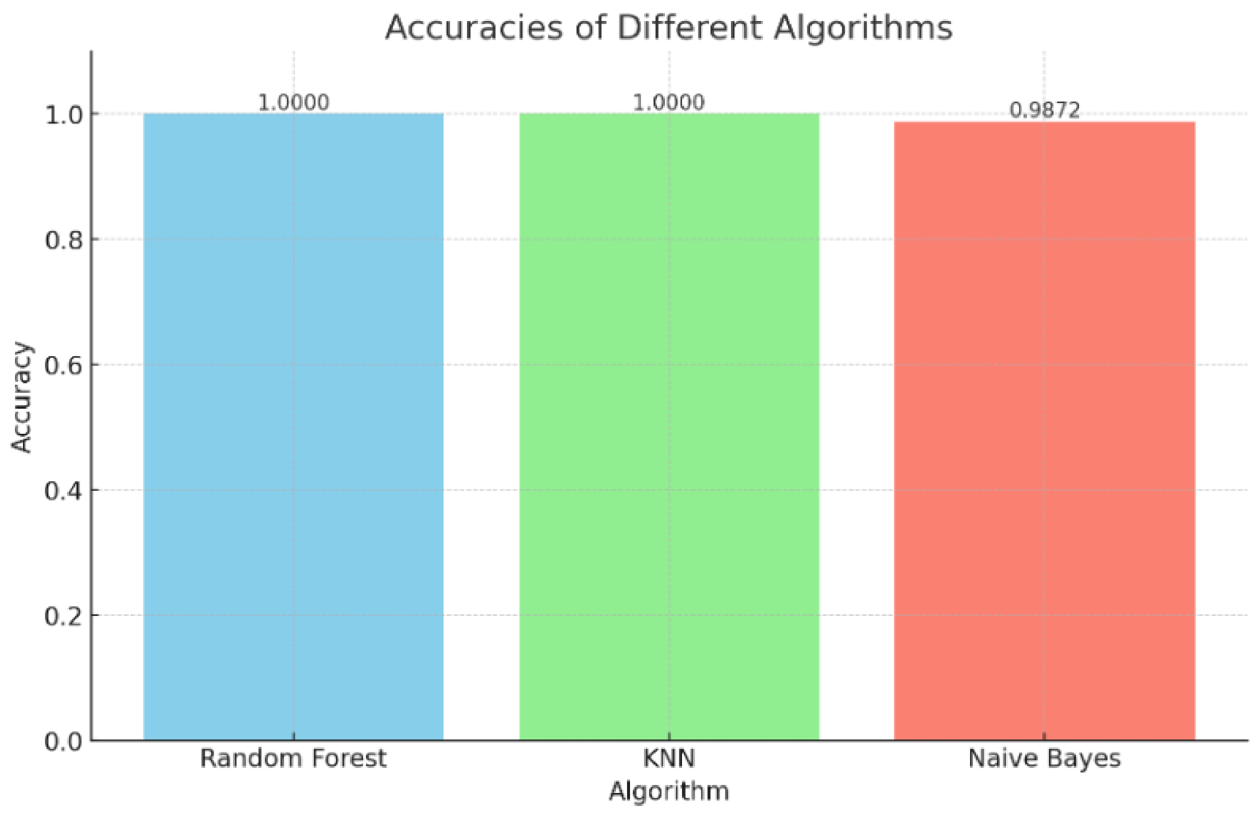
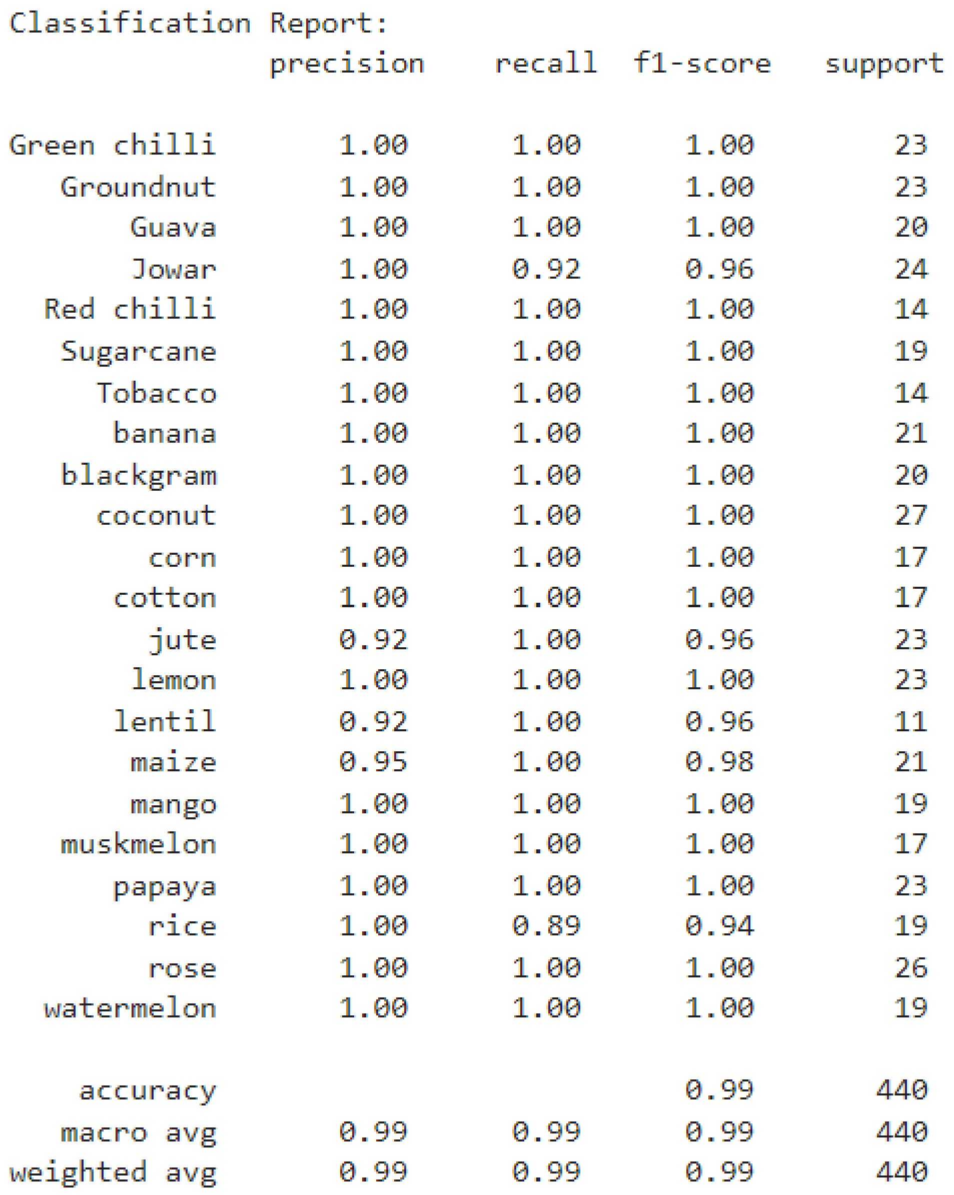


Fig.no 4.9. **Classification report** that evaluates the performance of a classification model

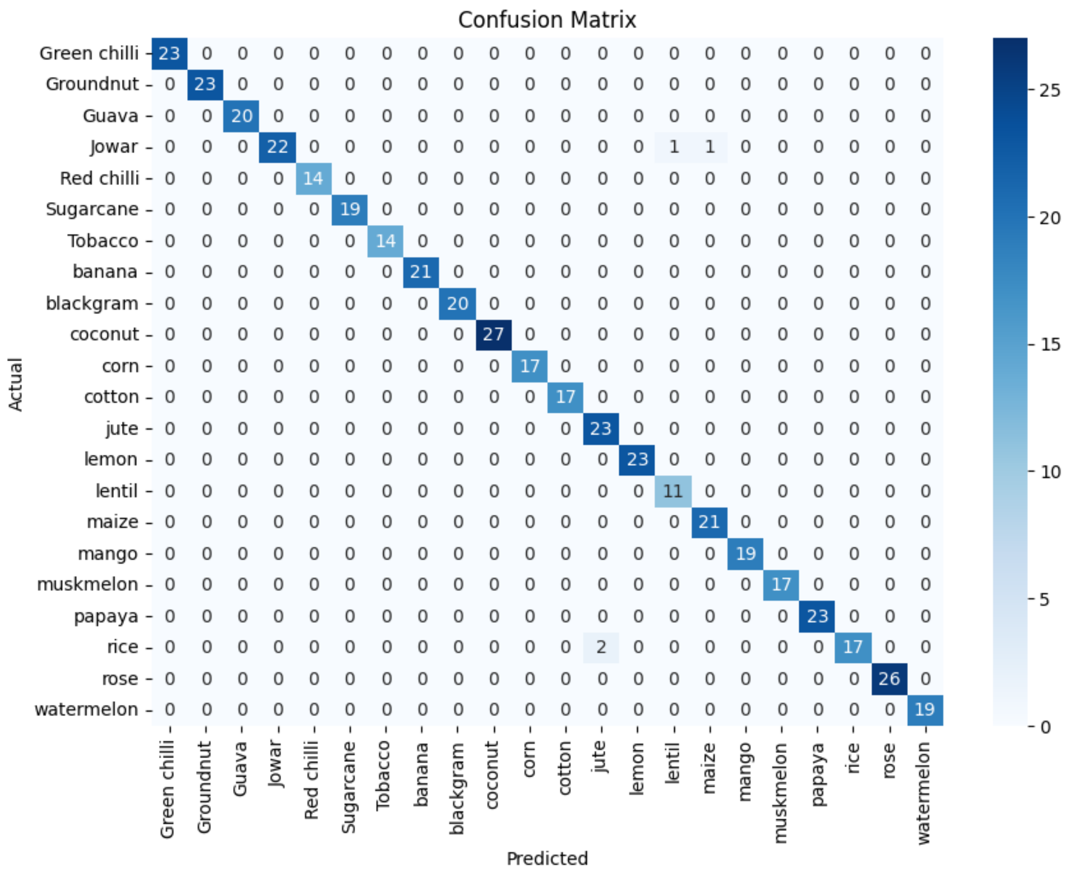


```
Classification Report:
              precision    recall  f1-score   support

Green chilli       1.00      1.00      1.00        23
   Groundnut       1.00      1.00      1.00        23
       Guava       1.00      1.00      1.00        20
       Jowar       1.00      0.92      0.96        24
  Red chilli       1.00      1.00      1.00        14
   Sugarcane       1.00      1.00      1.00        19
     Tobacco       1.00      1.00      1.00        14
      banana       1.00      1.00      1.00        21
   blackgram       1.00      1.00      1.00        20
     coconut       1.00      1.00      1.00        27
        corn       1.00      1.00      1.00        17
      cotton       1.00      1.00      1.00        17
        jute       0.92      1.00      0.96        23
       lemon       1.00      1.00      1.00        23
      lentil       0.92      1.00      0.96        11
       maize       0.95      1.00      0.98        21
       mango       1.00      1.00      1.00        19
   muskmelon       1.00      1.00      1.00        17
      papaya       1.00      1.00      1.00        23
        rice       1.00      0.89      0.94        19
        rose       1.00      1.00      1.00        26
  watermelon       1.00      1.00      1.00        19

    accuracy                           0.99       440
   macro avg       0.99      0.99      0.99       440
weighted avg       0.99      0.99      0.99       440
```

Fig.no 4.10. Confusion Matrix



Actual \ Predicted	Green chilli	Groundnut	Guava	Jowar	Red chilli	Sugarcane	Tobacco	banana	blackgram	coconut	corn	cotton	jute	lemon	lentil	maize	mango	muskmelon	papaya	rice	rose	watermelon
Green chilli	23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Groundnut	0	23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Guava	0	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Jowar	0	0	0	22	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
Red chilli	0	0	0	0	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Sugarcane	0	0	0	0	0	19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Tobacco	0	0	0	0	0	0	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
banana	0	0	0	0	0	0	0	21	0	0	0	0	0	0	0	0	0	0	0	0	0	0
blackgram	0	0	0	0	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0
coconut	0	0	0	0	0	0	0	0	0	27	0	0	0	0	0	0	0	0	0	0	0	0
corn	0	0	0	0	0	0	0	0	0	0	17	0	0	0	0	0	0	0	0	0	0	0
cotton	0	0	0	0	0	0	0	0	0	0	0	17	0	0	0	0	0	0	0	0	0	0
jute	0	0	0	0	0	0	0	0	0	0	0	0	23	0	0	0	0	0	0	0	0	0
lemon	0	0	0	0	0	0	0	0	0	0	0	0	0	23	0	0	0	0	0	0	0	0
lentil	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11	0	0	0	0	0	0	0
maize	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	21	0	0	0	0	0	0
mango	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19	0	0	0	0	0
muskmelon	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17	0	0	0	0
papaya	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	23	0	0	0
rice	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	17	0	0
rose	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	26	0
watermelon	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19

Fig.4.11 State wise number of crops

```
            State  Number of Crops
0  Andhra Pradesh               47
1       Karnataka               45
2          Kerala               37
3      Tamil Nadu               46
```

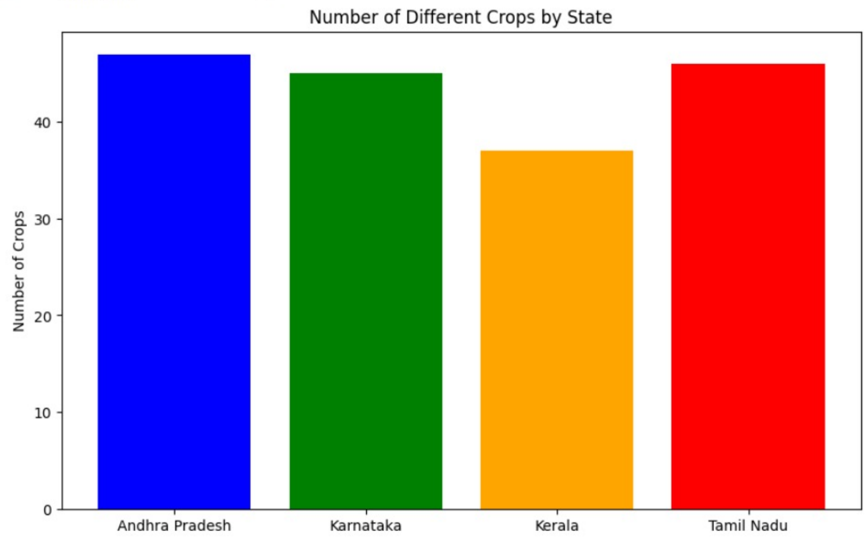


Fig 4.12 Year-wise crop production by State

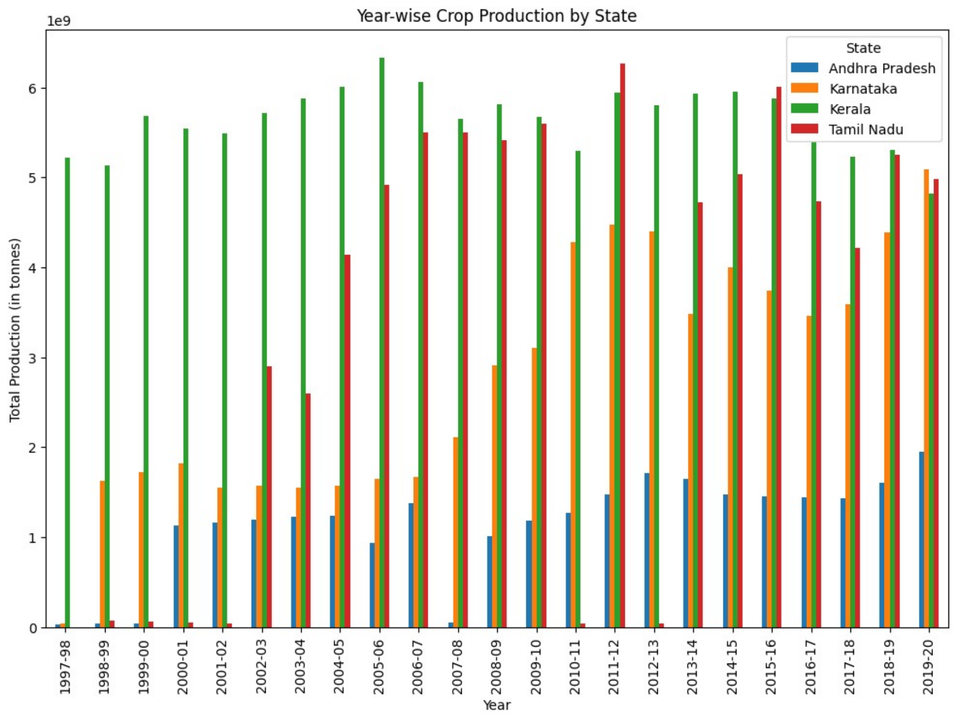
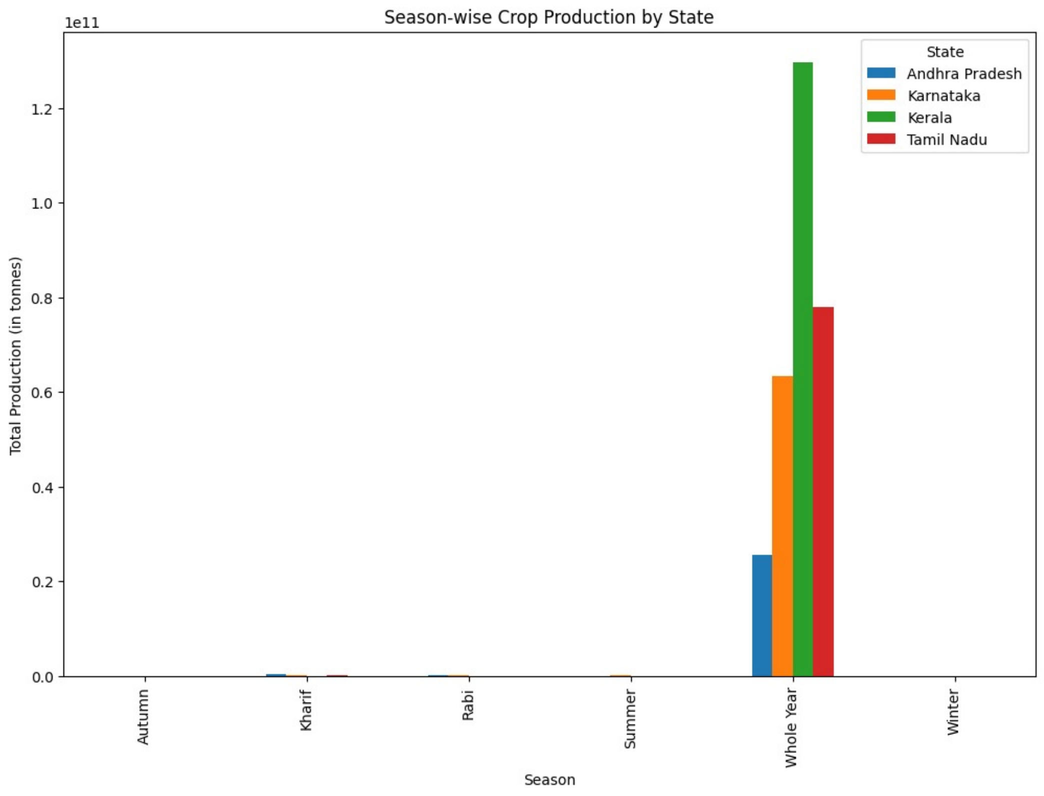


Fig 4.13 Season-wise Crop production by State



CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 CONCLUSION

The attempt signifies a major advancement in the field of agricultural technology towards the use of machine learning for crop prediction. We've set the stage for transforming agricultural practices and increasing crop output by utilizing the power of data analytics and predictive modeling.

The process started with the careful curation and cleansing of agricultural datasets to guarantee the consistency and integrity of the data. We developed a comprehensive repository of agricultural insights by combining disparate databases, offering a comprehensive perspective on crop production and recommendation data.

Our models proved to be effective crop prediction tools due to the RandomForestClassifier, KNN, and Naive Bayes algorithms. These models' remarkably high accuracy levels demonstrate how well they can identify complex patterns concealed in the data.

In addition to providing accurate crop type forecasts, these models offer priceless insights into the soil and environmental variables affecting crop growth. Equipped with this understanding, farmers may choose crops, cultivate their land, and distribute their resources sensibly.

In the future, there is a great deal of room for improvement and growth in our prediction models. Our models' forecasting power should be improved by adding more data sources, such as meteorological and soil health indices. Furthermore, using cutting-edge machine learning strategies like ensemble learning and deep learning has the potential to open up previously uncharted territory in terms of crop prediction accuracy.

5.2 FUTURE SCOPE

Our present models' performance provides a strong basis for further advancements and developments in agricultural technology. There exist multiple routes of promising research and development that have the potential to enhance our capacity for prediction and magnify the influence on agricultural practices.

Integration of Extra Data Sources: We can improve our models and get more insightful information by integrating data from sources other than the datasets that are currently available. A thorough understanding of the variables impacting crop growth and market dynamics can be obtained by integrating data on soil health, weather forecasts, satellite imagery, and even market trends. Increased data inputs may result in more precise forecasts and customised advice for farmers.

Advanced Machine Learning Techniques: Using state-of-the-art approaches to improve predictive accuracy, such ensemble methods and deep learning, can yield significant benefits. Complex spatial and temporal patterns in agricultural data can be captured by deep learning architectures such as recurrent neural networks (RNNs) and convolutional neural networks (CNNs). By utilising a variety of modelling techniques, ensemble methods—which aggregate predictions from several models—can further increase accuracy and robustness.

Geographic and Temporal Scalability: Models can be made more relevant and applicable by being tailored to particular geographic areas and temporal scales. Accurate forecasts that are adapted to the particular requirements of various agricultural locations can be achieved by creating region-specific models that take into consideration local soil types, climatic circumstances, and agricultural practices. Long-term forecasting and adaptive management techniques are made possible by the ability to identify temporal trends and seasonality in data gathered over several growing seasons.

CHAPTER 6

APPENDIX

1. KNN Algorithm

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.naive_bayes import GaussianNB

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy_score

from pydrive.auth import GoogleAuth

from pydrive.drive import GoogleDrive

from google.colab import auth

from oauth2client.client import GoogleCredentials


auth.authenticate_user()

gauth = GoogleAuth()
```

```
gauth.credentials = GoogleCredentials.get_application_default()

drive = GoogleDrive(gauth)

print(gauth.credentials)


downloaded = drive.CreateFile({'id':'1t0mQEHSyrCTN7UEWfJ_9Dodo4L9P7aXI'})  #
replace the id with id of file you want to access

downloaded.GetContentFile('Ap_Agriculture.csv')

downloaded = drive.CreateFile({'id':'1sPmFJKnY7ITuCjJIwrDNmn76rx2ZHCMc'})  #
replace the id with id of file you want to access

downloaded.GetContentFile('Crop_recommendation.csv')


# Load datasets

indian_agriculture = pd.read_csv('Ap_Agriculture.csv')

crop_recommendation = pd.read_csv('Crop_recommendation.csv', encoding='latin-1')


# Clean and standardize the 'Crop' column in both datasets

indian_agriculture['Crop'] = indian_agriculture['Crop'].str.strip().str.lower()

crop_recommendation['Crop'] = crop_recommendation['Crop'].str.strip().str.lower()


# Merge datasets based on 'Crop'

merged_data = pd.merge(indian_agriculture, crop_recommendation, on='Crop', how='inner')


# Drop rows with missing values

merged_data.dropna(inplace=True)


# Extract features and target variable for crop prediction

classification_features = ['Nitrogen', 'phosphorus', 'potassium', 'temperature', 'humidity', 'ph',
'rainfall']

classification_target = 'Crop'
```

```
# Extract features and target variable for yield prediction
regression_features = classification_features + ['Area']  # Include 'Area' for yield prediction
regression_target = 'Production'

X_classification = merged_data[classification_features]
y_classification = merged_data[classification_target]

X_regression = merged_data[regression_features]
y_regression = merged_data[regression_target]

# Split data into train and test sets for classification (crop prediction) - change random_state
for variability
X_train_class, X_test_class, y_train_class, y_test_class = train_test_split(X_classification,
y_classification, test_size=0.2, random_state=30)

# Split data into train and test sets for regression (yield prediction) - change random_state for
variability
X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(X_regression,
y_regression, test_size=0.2, random_state=42)

# Feature scaling (optional but recommended for KNN)
scaler = StandardScaler()
X_train_class = scaler.fit_transform(X_train_class)
X_test_class = scaler.transform(X_test_class)
X_train_reg = scaler.fit_transform(X_train_reg)
X_test_reg = scaler.transform(X_test_reg)

# Train KNeighborsClassifier for crop prediction with a different n_neighbors value
model_knn_classification = KNeighborsClassifier(n_neighbors=3)  # You can vary this
```

```
model_knn_classification.fit(X_train_class, y_train_class)

# Train KNeighborsRegressor for yield prediction with a different n_neighbors value
model_knn_regression = KNeighborsRegressor(n_neighbors=7)  # You can vary this
model_knn_regression.fit(X_train_reg, y_train_reg)

# Function to predict crop based on user input using KNN
def predict_crop_knn(input_features):
    input_df = pd.DataFrame([input_features], columns=classification_features)
    # Create a new scaler instance for classification features
    scaler_class = StandardScaler()
    input_df = scaler_class.fit_transform(input_df)  # Scale user input using the new scaler
    predicted_crop = model_knn_classification.predict(input_df)[0]
    return predicted_crop

# Function to predict yield based on user input using KNN
def predict_yield_knn(input_features):
    input_df = pd.DataFrame([input_features], columns=regression_features)
    input_df = scaler.transform(input_df)  # Scale user input
    predicted_yield = model_knn_regression.predict(input_df)[0]
    return predicted_yield

# Evaluate crop prediction model using KNN
def evaluate_classification_model(X_test, y_test):
    y_pred = model_knn_classification.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')
    return accuracy, precision, f1

# Evaluate yield prediction model using KNN
def evaluate_regression_model(X_test, y_test):
    y_pred = model_knn_regression.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    return mse

accuracy_class, precision_class, f1_class = evaluate_classification_model(X_test_class,
y_test_class)
print(f"Crop Prediction Metrics:")
print(f"Accuracy for knn: {accuracy_class:.4f}")
print(f"Precision: {precision_class:.4f}")
print(f"F1-score: {f1_class:.4f}")

mse_reg = evaluate_regression_model(X_test_reg, y_test_reg)
print(f"\nYield Prediction Metrics:")
print(f"Mean Squared Error: {mse_reg:.4f}")

# Interactive user input for prediction
print("Enter the following values for prediction:")
user_input = {}
for feature in classification_features:
    value = float(input(f"{feature.capitalize()}: "))
    user_input[feature] = value

# Get area (in hectares) input for yield prediction
area = float(input("Area (hectares): "))
user_input['Area'] = area
```

```
# Predict crop using KNN
predicted_crop_knn = predict_crop_knn(user_input)
print(f"Predicted Crop using KNN: {predicted_crop_knn.capitalize()}")

# Predict yield using KNN
predicted_yield_knn = predict_yield_knn(user_input)
print(f"Predicted production (tons/hectare) using KNN: {predicted_yield_knn:.2f}")
```

2. Naive Bayes Algorithm

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
print(gauth.credentials)
```

```
downloaded = drive.CreateFile({'id':'1t0mQEHSyrCTN7UEWfJ_9Dodo4L9P7aXI'}) #
replace the id with id of file you want to access

downloaded.GetContentFile('Ap_Agriculture.csv')

downloaded = drive.CreateFile({'id':'1sPmFJKnY7ITuCjJIwrDNmn76rx2ZHCMc'}) #
replace the id with id of file you want to access

downloaded.GetContentFile('Crop_recommendation.csv')

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.naive_bayes import GaussianNB

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy_score, precision_score, f1_score # Import
precision_score and f1_score

from sklearn.naive_bayes import MultinomialNB


# Load datasets

indian_agriculture = pd.read_csv('Ap_Agriculture.csv')

crop_recommendation = pd.read_csv('Crop_recommendation.csv', encoding='latin-1')


# Clean and standardize the 'Crop' column in both datasets

indian_agriculture['Crop'] = indian_agriculture['Crop'].str.strip().str.lower()

crop_recommendation['Crop'] = crop_recommendation['Crop'].str.strip().str.lower()


# Merge datasets based on 'Crop' (and other common columns if needed)

merged_data = pd.merge(indian_agriculture, crop_recommendation, on='Crop', how='inner')


# Drop rows with missing values

merged_data.dropna(inplace=True)
```

```
# Extract features and target variable for crop prediction
classification_features = ['Nitrogen', 'phosphorus', 'potassium', 'temperature', 'humidity', 'ph', 
'rainfall']
classification_target = 'Crop'

# Extract features and target variable for yield prediction
regression_features = classification_features + ['Area']  # Include 'Area' for yield prediction
regression_target = 'Production'

# Split data into features and target variables
X_classification = merged_data[classification_features]
y_classification = merged_data[classification_target]

# Split data into features and target variables
X_regression = merged_data[regression_features]
y_regression = merged_data[regression_target]

# Split data into train and test sets for classification (crop prediction)
X_train_class, X_test_class, y_train_class, y_test_class = train_test_split(X_classification, 
y_classification, test_size=0.2, random_state=42)

# Split data into train and test sets for regression (yield prediction)
X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(X_regression, 
y_regression, test_size=0.2, random_state=42)

# Train Naive Bayes Classifier for crop prediction
model_classification = MultinomialNB()
model_classification.fit(X_train_class, y_train_class)
```

```
# Train Naive Bayes Regressor for yield prediction

model_regression = GaussianNB()

model_regression.fit(X_train_reg, y_train_reg)


# Function to predict crop based on user input

def predict_crop(input_features):

    input_df = pd.DataFrame([input_features], columns=classification_features)

    predicted_crop = model_classification.predict(input_df)[0]

    return predicted_crop


    # Function to predict yield based on user input

def predict_yield(input_features):

    input_df= pd.DataFrame([input_features], columns=regression_features)

    predicted_yield = model_regression.predict(input_df)[0]

    return predicted_yield

    # Function to calculate accuracy, precision, and F1-score for crop prediction

def evaluate_classification_model(X_test, y_test):

    y_pred = model_classification.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)

    precision = precision_score(y_test, y_pred, average='weighted')

    f1 = f1_score(y_test, y_pred, average='weighted')

    return accuracy, precision, f1

    # Evaluate crop prediction model

accuracy_class, precision_class, f1_class = evaluate_classification_model(X_test_class,
y_test_class)

print(f"Crop Prediction Metrics:")

print(f"Accuracy for naive bayes: {accuracy_class:.4f}")

print(f"Precision: {precision_class:.4f}")

print(f"F1-score: {f1_class:.4f}")
```

```
# Interactive user input for prediction
print("Enter the following values for prediction:")
user_input = {}
for feature in classification_features:
    value = float(input(f"{feature.capitalize()}: "))
    user_input[feature] = value

area = float(input("Area (hectares): "))
user_input['Area'] = area

# Predict crop using Naive Bayes
predicted_crop_nb = predict_crop(user_input)
print(f"Predicted Crop using Naive Bayes: {predicted_crop_nb.capitalize()}")

predicted_yield = predict_yield(user_input)
print(f"Predicted production (tons/hectare): {predicted_yield:.2f}")
```

3. Random Forest Algorithm

```
    !pip install -U -q PyDrive
!pip install google-auth-oauthlib
from pydrive2.auth import GoogleAuth
from pydrive2.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

auth.authenticate_user()
```

```
gauth = GoogleAuth()

gauth.credentials = GoogleCredentials.get_application_default()

drive = GoogleDrive(gauth)

print(gauth.credentials)


downloaded = drive.CreateFile({'id':'1t0mQEHSyrCTN7UEWfJ_9Dodo4L9P7aXI'}) #
replace the id with id of file you want to access

downloaded.GetContentFile('Ap_Agriculture.csv')


downloaded = drive.CreateFile({'id':'1sPmFJKnY7ITuCjJIwrDNmn76rx2ZHCMc'}) #
replace the id with id of file you want to access

downloaded.GetContentFile('Crop_recommendation.csv')


import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor

from sklearn.metrics import accuracy_score, precision_score, f1_score



# Load datasets

indian_agriculture = pd.read_csv('Ap_Agriculture.csv')

crop_recommendation = pd.read_csv('Crop_recommendation.csv', encoding='latin-1')



# Clean and standardize the 'Crop' column in both datasets

indian_agriculture['Crop'] = indian_agriculture['Crop'].str.strip().str.lower()

crop_recommendation['Crop'] = crop_recommendation['Crop'].str.strip().str.lower()
```

```
# Merge datasets based on 'Crop' (and other common columns if needed)
merged_data = pd.merge(indian_agriculture, crop_recommendation, on='Crop', how='inner')


# Drop rows with missing values
merged_data.dropna(inplace=True)



# Extract features and target variable for crop prediction
classification_features = ['Nitrogen', 'phosphorus', 'potassium', 'temperature', 'humidity', 'ph',
'rainfall']
classification_target = 'Crop'


# Extract features and target variable for yield prediction
regression_features = classification_features + ['Area']  # Include 'Area' for yield prediction
regression_target = 'Production'


X_classification = merged_data[classification_features]
y_classification = merged_data[classification_target]


X_regression = merged_data[regression_features]
y_regression = merged_data[regression_target]


# Split data into train and test sets for classification (crop prediction)
```

```
X_train_class, X_test_class, y_train_class, y_test_class = train_test_split(X_classification,
y_classification, test_size=0.2, random_state=43)



# Split data into train and test sets for regression (yield prediction)

X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(X_regression,
y_regression, test_size=0.2, random_state=43)



# Train RandomForestClassifier for crop prediction

model_classification = RandomForestClassifier(n_estimators=50, max_depth=10,
random_state=43)

model_classification.fit(X_train_class, y_train_class)



# Train RandomForestRegressor for yield prediction

model_regression = RandomForestRegressor(n_estimators=50, max_depth=10,
random_state=43)

model_regression.fit(X_train_reg, y_train_reg)



# Split data into train and test sets for classification (crop prediction)

X_train, X_test, y_train, y_test = train_test_split(X_classification, y_classification,
test_size=0.2, random_state=43)


# Train a RandomForestClassifier

clf = RandomForestClassifier(random_state=55232)

clf.fit(X_train, y_train)


# Predict the crop types for the test set
```

```
y_pred = clf.predict(X_test)


# Generate the confusion matrix
cm = confusion_matrix(y_test, y_pred) # Now confusion_matrix is defined


# Print the classification report
print("Classification Report:\n", classification_report(y_test, y_pred))


# Plotting the confusion matrix
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=clf.classes_,
yticklabels=clf.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
def evaluate_classification_model(X_test, y_test):
    y_pred = model_classification.predict(X_test) # Use classification model here
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')
    return accuracy, precision, f1


accuracy_class, precision_class, f1_class = evaluate_classification_model(X_test_class,
y_test_class)
print(f"Crop Prediction Metrics:")
print(f"Accuracy for randomforest: {accuracy_class:.4f}")
print(f"Precision: {precision_class:.4f}")
print(f"F1-score: {f1_class:.4f}")
```

```
# Function to calculate mean squared error for yield prediction
from sklearn.metrics import mean_squared_error # Import the missing function

def evaluate_regression_model(X_test, y_test):
    y_pred = model_regression.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    return mse
    # Evaluate yield prediction model
mse_reg = evaluate_regression_model(X_test_reg, y_test_reg)
print(f"\nYield Prediction Metrics:")
print(f"Mean Squared Error: {mse_reg:.4f}")

# Function to predict crop based on user input
def predict_crop(input_features):
    input_df = pd.DataFrame([input_features], columns=classification_features)
    predicted_crop = model_classification.predict(input_df)[0]
    return predicted_crop



# Function to predict yield based on user input
def predict_yield(input_features):
    input_df = pd.DataFrame([input_features], columns=regression_features)
    predicted_yield = model_regression.predict(input_df)[0]
    return predicted_yield
```

```
def manage_predicted_crop(predicted_crop, input_features):
    print("Management Recommendations:")

    if predicted_crop == 'rice':
        manage_rice(input_features)
    elif predicted_crop == 'maize':
        manage_maize(input_features)
    elif predicted_crop == 'rose':
        manage_rose(input_features)
    elif predicted_crop == 'guava':
        manage_guava(input_features)
    elif predicted_crop == 'lemon':
        manage_lemon(input_features)
    elif predicted_crop == 'jowar':
        manage_jowar(input_features)
    elif predicted_crop == 'sugarcane':
        manage_sugarcane(input_features)
    elif predicted_crop == 'blackgram':
        manage_blackgram(input_features)
    elif predicted_crop == 'lentil':
        manage_lentil(input_features)
    elif predicted_crop == 'groundnut':
        manage_groundnut(input_features)
    elif predicted_crop == 'banana':
        manage_banana(input_features)
    elif predicted_crop == 'mango':
        manage_mango(input_features)
    elif predicted_crop == 'tobacco':
        manage_tobacco(input_features)
    elif predicted_crop == 'watermelon':
        manage_watermelon(input_features)
    elif predicted_crop == 'muskmelon':
        manage_muskmelon(input_features)
    elif predicted_crop == 'chilli':
        manage_chilli(input_features)
    elif predicted_crop == 'papaya':
        manage_papaya(input_features)
    elif predicted_crop == 'coconut':
        manage_coconut(input_features)
    else:
        print("Management recommendations not available for this crop.")


# Management recommendations for chilli
def manage_chilli(input_features):
    print("- Plant in warm, well-drained soil.")
    print("- Provide support for growing plants.")
    print("- Recommended Soil Type:black soil and red loams ")
    print("- Recommended Pesticides: Chlorpyrifos and Imidacloprid")
    if input_features['temperature'] > 30:
        print("- Use shade nets or provide shading during extreme heat.")
    print("- Apply balanced fertilizers and monitor for pests.")


def manage_rice(input_features):
    print("- Plant in clay/loamy soil.")
    print("- Recommended Pesticides: Methyl Parathion")
    if input_features['temperature'] > 30:
        print("- Use shade nets or provide shading during extreme heat.")
    print("- Apply balanced fertilizers and monitor for pests.")

def manage_maize(input_features):
    print("- Plant in heavy clay soil.")
    print("- Recommended Pesticides: Lambda-cyhalothrin")
    if input_features['temperature'] > 30:
        print("- Use shade nets or provide shading during extreme heat.")
    print("- Apply balanced fertilizers and monitor for pests.")

def manage_rose(input_features):
    print("- Plant in loamy soil.")
    print("- Recommended Pesticides: Acephate")
    if input_features['temperature'] > 30:
        print("- Use shade nets or provide shading during extreme heat.")
    print("- Apply balanced fertilizers and monitor for pests.")

def manage_guava(input_features):
    print("- Plant in well-drained, loamy soils.")
    print("- Recommended Pesticides: Umega")
    if input_features['temperature'] > 30:
        print("- Use shade nets or provide shading during extreme heat.")
    print("- Apply balanced fertilizers and monitor for pests.")

def manage_lemon(input_features):
    print("- Plant in sandy loam soil.")
    print("- Recommended Pesticides: Chlorpyrifos")
    if input_features['temperature'] > 30:
        print("- Use shade nets or provide shading during extreme heat.")
    print("- Apply balanced fertilizers and monitor for pests.")

def manage_jowar(input_features):
    print("- Plant in alluvial soil or mixed black soil.")
    print("- Recommended Pesticides: 2,4-D")
    if input_features['temperature'] > 30:
        print("- Use shade nets or provide shading during extreme heat.")
    print("- Apply balanced fertilizers and monitor for pests.")

def manage_sugarcane(input_features):
    print("- Plant in loam soil.")
    print("- Recommended Pesticides: Propiconazole")
    if input_features['temperature'] > 30:
        print("- Use shade nets or provide shading during extreme heat.")
    print("- Apply balanced fertilizers and monitor for pests.")

def manage_blackgram(input_features):
    print("- Plant in red laterite soil.")
    print("- Recommended Pesticides: Acephate")
    if input_features['temperature'] > 30:
        print("- Use shade nets or provide shading during extreme heat.")
    print("- Apply balanced fertilizers and monitor for pests.")

def manage_lentil(input_features):
    print("- Plant in loamy soil.")
    print("- Recommended Pesticides: Glyphosate")
    if input_features['temperature'] > 30:
        print("- Use shade nets or provide shading during extreme heat.")
    print("- Apply balanced fertilizers and monitor for pests.")

def manage_groundnut(input_features):
    print("- Plant in sandy loamy soil.")
    print("- Recommended Pesticides: Imazethapyr")
    if input_features['temperature'] > 30:
        print("- Use shade nets or provide shading during extreme heat.")
    print("- Apply balanced fertilizers and monitor for pests.")

def manage_banana(input_features):
    print("- Plant in loamy soil.")
    print("- Recommended Pesticides: Neem oil")
    if input_features['temperature'] > 25:
        print("- Ensure adequate water and provide mulching.")
    print("- Implement proper spacing and remove old leaves regularly.")

def manage_mango(input_features):
    print("- Plant in alluvial soil.")
    print("- Recommended Pesticides: Fipronil")
    if input_features['temperature'] > 30:
        print("- Use shade nets or provide shading during extreme heat.")
    print("- Apply balanced fertilizers and monitor for pests.")

def manage_tobacco(input_features):
    print("- Plant in sandy soil.")
    print("- Recommended Pesticides: Methyl bromide")
    if input_features['temperature'] > 30:
        print("- Use shade nets or provide shading during extreme heat.")
    print("- Apply balanced fertilizers and monitor for pests.")

def manage_watermelon(input_features):
    print("- Plant in sandy loam soil.")
    print("- Recommended Pesticides: Pyrethroids")
    if input_features['temperature'] > 30:
        print("- Use shade nets or provide shading during extreme heat.")
    print("- Apply balanced fertilizers and monitor for pests.")

def manage_muskmelon(input_features):
    print("- Plant in sandy loam soil.")
    print("- Recommended Pesticides: Pyrethroids")
    if input_features['temperature'] > 30:
        print("- Use shade nets or provide shading during extreme heat.")
    print("- Apply balanced fertilizers and monitor for pests.")



def manage_papaya(input_features):
    print("- Plant in sandy loam/black soil.")
    print("- Recommended Pesticides: Metalaxyl-M")
    if input_features['temperature'] > 30:
        print("- Use shade nets or provide shading during extreme heat.")
    print("- Apply balanced fertilizers and monitor for pests.")

def manage_coconut(input_features):
    print("- Plant in laterite soil.")
    print("- Recommended Pesticides: Carbamate")
    if input_features['temperature'] > 30:
        print("- Use shade nets or provide shading during extreme heat.")
    print("- Apply balanced fertilizers and monitor for pests.")
```

```
6.# Interactive user input for prediction
print("Enter the following values for prediction:")
user_input = {}
for feature in classification_features:
    value = float(input(f"{feature.capitalize()}: "))
    user_input[feature] = value
# Get area (in hectares) input for yield prediction
area = float(input("Area (hectares): "))
user_input['Area'] = area

# Predict crop based on user input
predicted_crop = predict_crop(user_input)
print(f"Predicted Crop: {predicted_crop.capitalize()}")
# Predict yield based on user input
predicted_yield = predict_yield(user_input)
print(f"Predicted production (tons/hectare): {predicted_yield:.2f}")
# Provide personalized management recommendations for the predicted crop
manage_predicted_crop(predicted_crop,user_input)
```

4. State Comparison

```
!pip install -U -q PyDrive
!pip install google-auth-oauthlib
```

```
from pydrive2.auth import GoogleAuth

from pydrive2.drive import GoogleDrive

from google.colab import auth

from oauth2client.client import GoogleCredentials


auth.authenticate_user()

gauth = GoogleAuth()

gauth.credentials = GoogleCredentials.get_application_default()

drive = GoogleDrive(gauth)

print(gauth.credentials)


downloaded = drive.CreateFile({'id':'1VA-OIsFCc6G48E-YrWnKbMEh6g3CVYZh'}) #
replace the id with id of file you want to access

downloaded.GetContentFile('States_Agri.csv')


import pandas as pd


# Load the Excel file

file_path = 'States_Agri.csv'

xls = pd.ExcelFile(file_path)


# Load the data from 'Sheet1'

df = pd.read_excel(xls, sheet_name='Sheet1')


# Aggregate the production data by state

statewise_production = df.groupby('State')['Production'].sum().reset_index()


# Display the aggregated production data

print(statewise_production)
```

```
# Calculate the number of unique crops for each state
statewise_crops = df.groupby('State')['Crop'].nunique().reset_index()

# Rename the columns for clarity
statewise_crops.columns = ['State', 'Number of Crops']

# Display the number of unique crops for each state
print(statewise_crops)

# To compare the number of crops between states, you can visualize the data using a bar
chart
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.bar(statewise_crops['State'], statewise_crops['Number of Crops'], color=['blue', 'green',
'orange','red'])
plt.title('Number of Different Crops by State')
plt.xlabel('State')
plt.ylabel('Number of Crops')
plt.show()

df = pd.read_excel(xls, sheet_name='Sheet1')

# Aggregate the production data by state and year
yearwise_state_comparison = df.groupby(['Year', 'State'])['Production'].sum().unstack()

# Display the aggregated production data
print(yearwise_state_comparison)
```

```
# Plot the year-wise state comparison
yearwise_state_comparison.plot(kind='bar', figsize=(12, 8))
plt.title('Year-wise Crop Production by State')
plt.xlabel('Year')
plt.ylabel('Total Production (in tonnes)')
plt.legend(title='State')
plt.show()
# Aggregate the production data by state and season
seasonwise_state_comparison = df.groupby(['Season', 'State'])['Production'].sum().unstack()

# Display the aggregated production data
print(seasonwise_state_comparison)

# Plot the season-wise state comparison
seasonwise_state_comparison.plot(kind='bar', figsize=(12, 8))
plt.title('Season-wise Crop Production by State')
plt.xlabel('Season')
plt.ylabel('Total Production (in tonnes)')
plt.legend(title='State')
plt.show()
```

CHAPTER 7
REFERENCES

[1.] L. Banda, A. Rai, A. Kansal and A. K. Vashisth, "Suitable Crop Prediction based on affecting parameters using Naïve Bayes Classification Machine Learning Technique," 2023 International Conference on Disruptive Technologies (ICDT), Greater Noida, India, 2023, pp. 43-46, doi: 10.1109/ICDT57929.2023.10150814.

[2.] Rao, Madhuri & Singh, Arushi & Reddy, N V Subba & Acharya, Dinesh. (2022). Crop prediction using machine learning. Journal of Physics: Conference Series. 2161. 012033. 10.1088/1742-6596/2161/1/012033.

[3.] M. Kalimuthu, P. Vaishnavi and M. Kishore, "Crop Prediction using Machine Learning," 2020 Third International Conference on Smart Systems and Inventive Technology (ICSSIT), Tirunelveli, India, 2020, pp. 926-932, doi: 10.1109/ICSSIT48917.2020.9214190.

[4.] Chavan, Jitendra & Pawade, Nagesh & Tale, Akshay & Kadam, Amit & Gujar, Amit. (2022). Crop Yield Prediction Using Naïve Bayes Algorithm. International Journal of Scientific Research in Computer Science, Engineering and Information Technology. 326-333. 10.32628/CSEIT228338.

[5.] Elbasi E, Zaki C, Topcu AE, Abdelbaki W, Zreikat AI, Cina E, Shdefat A, Saker L. Crop Prediction Model Using Machine Learning Algorithms. Applied Sciences. 2023; 13(16):9288.

[6.] P. S. Vijayabaskar, R. Sreemathi and E. Keertanaa, "Crop prediction using predictive analytics," 2017 International Conference on Computation of Power, Energy Information and Commuincation (ICCPEIC), Melmaruvathur, India, 2017, pp. 370-373, doi: 10.1109/ICCPEIC.2017.8290395.

[7.] A. Motwani, P. Patil, V. Nagaria, S. Verma and S. Ghane, "Soil Analysis and Crop Recommendation using Machine Learning," 2022 International Conference for Advancement in Technology (ICONAT)

[8.] Md Reazul Islam, Khondokar Oliullah, Md Mohsin Kabir, Munzirul Alom, M.F. Mridha, Machine learning enabled IoT system for soil nutrients monitoring and crop recommendation.

[9.] Smart Agricultural Crop Prediction Using Machine Learning Journal of Xi’an University of Ar- chitecture Technology Volume XII, Issue V, 2020 ISSN No : 1006-7930.

[10.] Crop Yield Prediction Using Machine Learning Algorithms 2019 Fifth International Confer- ence on Image Information Processing.