

# Predictive Modelling for Soil-Crop Compatibility Using Machine Learning Algorithms

A  
DISSERTATION SUBMITTED  
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS  
OF THE DEGREE OF  
MASTER OF SCIENCE  
IN  
MATHEMATICS & SCIENTIFIC COMPUTING

BY  
*Shrikant Sushilrao Deshmukh*  
(23MAC2R25)

Under the supervision of  
*Dr. J. Pranitha*



DEPARTMENT OF MATHEMATICS  
NATIONAL INSTITUTE OF TECHNOLOGY WARANGAL  
TELANGANA - 506004, INDIA

MAY 2025

## **ABSTRACT**

Agriculture serves as the backbone of maintaining economies, and maximizing the choice of crop depending on soil and climate can enhance yield and profitability remarkably. This project, "Predictive Modeling for Soil-Crop Compatibility Using ML Algorithms," utilizes machine learning models to help farmers choose the best crop for their land. The model is created using primary soil and climate parameters like Nitrogen (N), Phosphorus (P), Potassium (K), Temperature, Humidity, Soil pH, and Rainfall.

Different machine learning algorithms, such as Decision Tree, Support Vector Classifier, K-Nearest Neighbors, Bagging, and Stacking, were experimented upon to find the best accuracy for prediction. Among them, Random Forest Classifier gave the maximum accuracy of 99.09%, which is to be deployed as the final model. The system gives data-driven suggestions to the farmers that can help them gain maximum yield, maximize profit, and minimize crop failure risks. This study demonstrates the efficiency of machine learning in precision agriculture and encourages sustainable agriculture by making decisions based on data.

## TABLE OF CONTENTS

S.N	Name of The Chapter	Page No.
1.	<b>Introduction To</b>	
	1.1 Machine Learning	
	1.2 Project	
2.	<b>Definitions</b>	
	2.1 Library	
	2.2 Modulus	
	2.3 Classes	
3.	<b>Process Diagram</b>	
4.	<b>Data Collection</b>	
5.	<b>Data Preprocessing</b>	
	5.1 Exploring The Data	
	5.2 Handling Missing Values	
	5.3 Outliers Detection	
6.	<b>Exploratory Data Analysis (EDA)</b>	
	6.1 Data Visualization	
	6.2 Data Splitting	
	6.3 Feature Sealing	
	6.4 Handling Categorical Data	
7.	<b>Model Training, Testing &amp; Selection</b>	
	<b>7.1 Processes For Every Models</b>	
	7.1.1 Object Creation	
	7.1.2 Hyperparameter Tuning Using Grid Search	
	7.1.3 Selection Of Best Parameters	
	7.1.4 Update Parameters	
	7.1.5 Fit the Model Base on Training Data	
	7.1.6 Prediction	
	7.1.7 Accuracy	
	<b>7.2 List of Models</b>	
	7.2.1 Decision Tree Classifier	
	7.2.2 SVC	
	7.2.3 KNN	
	7.2.4 Bagging	
	7.2.5 Stacking	
8.	<b>Confusion Matrix, Heat Map</b>	
9.	<b>Selection Of Best Model</b>	
10.	<b>Conclusion</b>	
11.	<b>GitHub Link to Code</b>	
12.	<b>References</b>	

## **Chapter 1 : Introduction**

### **1. Machine Learning :-**

Machine Learning is a branch of Artificial Intelligence that enables computers to learn from data and make decisions or predictions without using computer programming language.

### **2. Project :-**

- Predictive Modeling for Soil-Crop Compatibility Using ML Algorithms is a Machine Learning-based Project for Identifying Suitable Crops for Specific Soil Types.
- Which aims to provide the farmer or agricultural expert with a decision of the best crop to be planted according to the conditions of the soil, weather, and geographical location.
- We build the model using these input features : Nitrogen (N Kg/ha), Phosphorus (P Kg/ha), Potassium (K Kg/ha), Temperature (°C), humidity, PH of soil and Rainfall (cm).
- To explore various machine learning algorithms (e.g. Decision Tree Classifier, Support Vector Classifier, Random Forest Classifier, KNeighbors Classifier, Bagging Classifier, etc) to identify the best accuracy model.
- Built a Random Forest Classifier model which achieved an highest accuracy of 99.09% .

## Chapter 2 : Definitions

1. **Library :-** A library is a collection of set of modules or packages, pre-written code, functions, and this tools help for developers build, training model and deploy Machine Learning models efficiently.
  - **NumPy** – Numerical computing library.
  - **Pandas** – Data manipulation and analysis.
  - **Matplotlib** – A powerful, low-level plotting library for creating graphs, animations and interactive visualizations.
  - **Seaborn** – Built on top of Matplotlib, it provides a higher-level of visualizations, animations and graphs as compare to Matplotlib.
  - **Scikit-Learn** – Classical ML algorithms (e.g., regression, clustering, classification).
  - **XGBoost** – Optimized gradient boosting library for structured data.
2. **Modulus :-** A module in Python is a single file with extension ‘.py’. That contains reusable code such as Functions, Classes and Variables. Models helps to organize code, improve reusability and make programming more efficient.

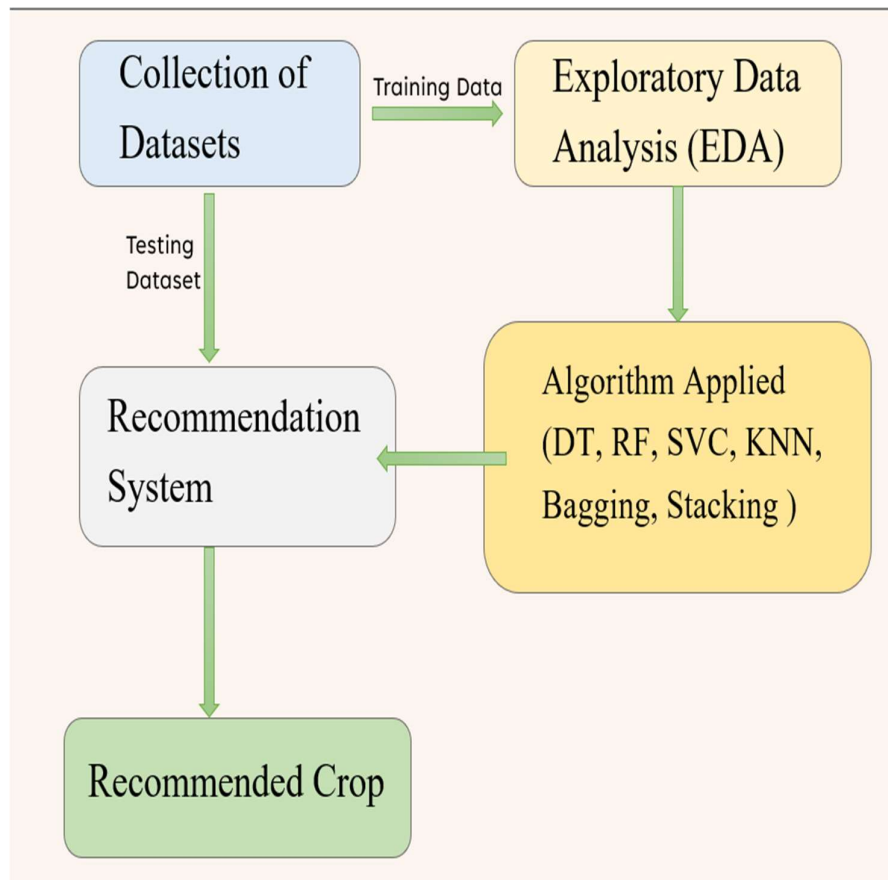
Examples :

  - `matplotlib.pyplot`, `sklearn.preprocessing`, `sklearn.tree`, `sklearn.ensemble`, `sklearn.svm`, `sklearn.metrics`, `sklearn.model_selection`.
3. **Package :-** A package in Python is a collection of multiple modules are organized in a directory data type. Package helps group related modules together for better code organization and reuse. A package contain `__init__.py` file to be recognized as a package  
Examples :
  - `sklearn.preprocessing`, `sklearn.model_selection`, `sklearn.tree`, `sklearn.ensemble`, `sklearn.svm`, `sklearn.neighbors`, `sklearn.linear_model`, `sklearn.metrics`.
4. **Classes :-** In ML, classes are used to structure models, preprocess data, and build reusable components by creating object of classes.

Examples :

  - `LabelEncoder`, `normalize`, `DecisionTreeClassifier`, `BaggingClassifier`, `SVC`, `GridSearchCV`, `KNeighborsClassifier`, `StackingClassifier`, `RandomForestClassifier`, `GradientBoostingClassifier`, `LogisticRegression`, `r2_score`, `accuracy_score`.

### Chapter 3 : Process Diagram



## Chapter 4 : Data Collection

- The dataset consists of parameters like Nitrogen(N kg/ha), Phosphorous(P kg/ha), Pottasium (K kg/ha), PH value of soil, Humidity, Temperature (°C) and Rainfall (cm).
- The datasets have been obtained from the ICAR stands for Indian Council of Agricultural Research (14 crop data) and kaggle website (11 crops data),
- Total 25 crops and each has 100 rows. Total size of data is 2500 rows and 8 columns.
- Code for uploading dataset :-  

```
df = pd.read_csv("/content/Crop_recommendation.csv", encoding="ISO-8859-1")
```

## Chapter 5 : Data Preprocessing

### 5. Data Preprocessing :-

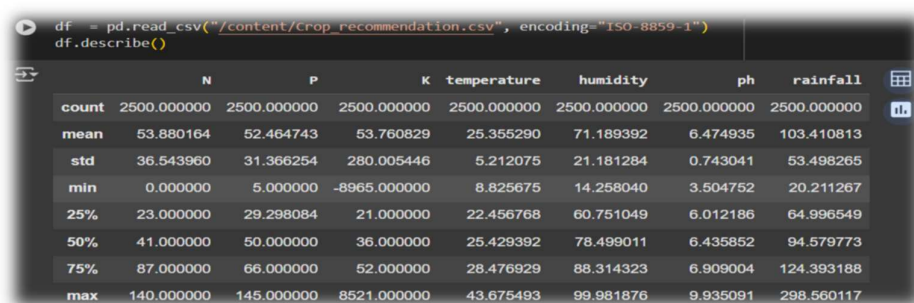
Data preprocessing is a second most important step in the data science that used for transforming raw data into a clean structured format for analysis. It includes various tasks like handling missing values, normalizing data and Outliers Detection. Using Data Preprocessing model performs well.

#### 5.1 Exploring The Data :-

Exploring the data refers to the process of understanding the structure, quality, and key characteristics of a dataset before performing further analysis or building machine learning models.

In which I did following steps

- `df.head()` :- Shows the first 5 rows of input dataset and it helps get an initial understanding of the dataset format and values.
- `df.shape` :- Check Dimensions (Number of rows, Number of columns) of give dataset, Understand the dataset size.
- `df.info()` :- Summary of data types and missing values, Helps differentiate between numerical and categorical variables.
- `df.describe()` :- Quickly understand data distribution, this function in pandas that provides a summary of statistical measures for numerical columns in a DataFrame.



```
df = pd.read_csv("/content/Crop_recommendation.csv", encoding="ISO-8859-1")
df.describe()
```

	N	P	K	temperature	humidity	ph	rainfall
count	2500.000000	2500.000000	2500.000000	2500.000000	2500.000000	2500.000000	2500.000000
mean	53.880164	52.464743	53.760829	25.355290	71.189392	6.474935	103.410813
std	36.543960	31.366254	280.005446	5.212075	21.181284	0.743041	53.498265
min	0.000000	5.000000	-8965.000000	8.825675	14.258040	3.504752	20.211267
25%	23.000000	29.298084	21.000000	22.456768	60.751049	6.012186	64.996549
50%	41.000000	50.000000	36.000000	25.429392	78.499011	6.435852	94.579773
75%	87.000000	66.000000	52.000000	28.476929	88.314323	6.909004	124.393188
max	140.000000	145.000000	8521.000000	43.675493	99.981876	9.935091	298.560117

#### 5.2 Handling Missing Values :-

In machine learning, handling missing values is a crucial preprocessing step. Missing values, represented as NaNs (Not a Number) or blanks, can significantly impact the performance of your models.

Code for counting missing values in each column.

- `df.isnull().sum()` :- Count missing values in each column.
- `df.isna().mean() * 100` :- Overall percentage of missing values in dataset.



By default, the given data has no missing values, so there is no need to handle missing values. Otherwise, the 'SimpleImputer()' class can be used for handling them.

### 5.3 Outliers Detection :-

Outlier detection is a process in data preprocessing that identifies data points that deviate significantly from the normal or expected pattern within a dataset. In simple terms, it is a process of finding those data points that are 'anomalous' or 'unusual' in dataset.

#### Why is outlier detection important ?

- Outlier can skew the statistical analyses and affect machine learning models to producing unreliable results.
- Detecting outliers can help identify errors in data collection.
- Due to outlier model predict wrong prediction it is harmful.
- It is important for fraud detection, spam detection etc.

#### How to detected outliers ?

Detected outliers by using visual and statistical methods, to identify data points that deviate significantly from the rest of your dataset.

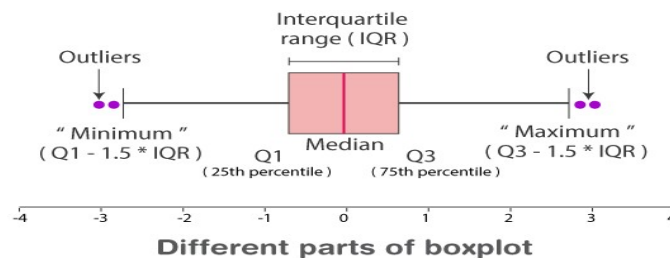
#### Visual Methods using Box-plot :-

- Box plots (box-and-whisker plots) visually represent the distribution of your data.
- In visual methods, outliers are shown as points that fall outside the minimum and maximum boundaries of box-plot.
- This method is effective for identifying outliers in a single variable.

#### Statistical Method using Interquartile Range (IQR) :-

- The IQR (Interquartile Range) it is range between 1<sup>st</sup> quartile (25<sup>th</sup> percentile, Q1) and 3<sup>rd</sup> quartile (75<sup>th</sup> percentile, Q3)
- Outliers are defined as data points that fall before minimum boundary (  $Q1 - 1.5 * IQR$  ) or after maximum boundary (  $Q3 + 1.5 * IQR$  ).
- This method is robust for datasets with non-normal distributions.

#### Diagram For Box-plot and IQR



## Chapter 6 : Exploratory Data Analysis (EDA)

### 6. Exploratory Data Analysis (EDA) :-

Exploratory Data Analysis (EDA) is the process of visualization and summarizing datasets to understand their main characteristics, by using this process we standardized all features into a stander format, and it is about gaining insights into the data before modelling or hypothesis testing.

#### 6.1 Data Visualization :-

Data Visualization is the process of graphically representating of information and data. It represents the information and data using visual elements like charts, graphs, maps and other graphical tools to identify outliers, patterns, trends in datasets.

#### 6.2 Data Splitting :-

Data splitting is the process of dividing a dataset into two or more subsets for training and evaluating machine learning models. In machine learning process, data are split into two subset : training set and testing set.

- **Training Set :-** The training set is used to train the machine learning models. The model learns form training set, identify some patters and finds relationships within training data. Then model predict the prediction based on training data.
- **Testing Set :-** The testing set is used to evaluate the performance of the trained model. It provides unseen data to model to assess accuracy of model.

And We can do this by using 'train\_test\_split()' class from 'sklearn.model\_selection' this modulus.

**Code :-**

```
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
X = df.drop("label",axis = 1)
Y = df["label"]
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size = 0.2,
random_state = 42)
```

#### 6.3 Feature Scaling :-

Feature scaling is the process of normalizing or standardizing the range of features of data. In simple terms, it is a process that brings all the features to a similar scale.

Here's a breakdown :-

- **Why is it important ? :-** Many machine learning algorithms make incorrect predictions when features are not on the same scale. when all features are on a similar scale, these machine learning algorithms perform better or

converge faster. Algorithms like Gradient Descent, KNN and SVM are sensitive to the scale of features.

- **Common methods :-**

- **Normalization (Min-Max Scaling) :-**

- Scales all feature to a specific range, typically [0, 1].
    - Formula :-  $(x - \text{min\_value}) / (\text{max\_value} - \text{min\_value})$

- **Standardization (Z-score Normalization) :-**

- Feature scaling adjusts features to a specific range, such as a mean of 0 and a standard deviation of 1.
    - Formula:  $(x - \text{mean}) / \text{standard deviation}$ .

- **But I used Standardization, Code :-**

```
scaler = StandardScaler()
X_train_scaler = scaler.fit_transform(X_train)
X_test_scaler = scaler.transform(X_test)
```

#### 6.4 Handling Categorical Data :-

Handling categorical data in machine learning involves transforming non-numerical features into a numerical format that algorithms can understand. Categorical data represents qualities or characteristics, and it comes in two main types :-

- **Nominal Data:** Categories without any inherent order (e.g., colors: red, blue, green).
- **Ordinal Data:** Categories with a meaningful order (e.g., education levels: high school, bachelor's, master's).

`Y = df["label"]` This is output categorical column

For this I used Ordinal Data type and Code is :-

```
le = LabelEncoder()
Y_train_le = le.fit_transform(Y_train)
Y_test_le = le.transform(Y_test)
```

Following code snippet demonstrates how to create a dictionary that maps categorical labels to their corresponding numerical representations after using a LabelEncoder,

```
label_mapping = dict(zip(le.classes_, range(len(le.classes_))))
print("Label Mapping:", label_mapping)
```

### **Ordinal Encoder output :-**

Label Mapping: { 'Corn' : 0, 'apple' : 1, 'banana' : 2, 'blackgram' : 3, 'chickpea' : 4, 'coconut' : 5, 'coffee' : 6, 'cotton' : 7, 'grapes' : 8, 'jute' : 9, 'kidneybeans' : 10, 'lentil' : 11, 'mango' : 12, 'mothbeans' : 13, 'mungbean' : 14, 'muskmelon' : 15, 'orange' : 16, 'papaya' : 17, 'pigeonpeas' : 18, 'pomegranate' : 19, 'rice' : 20, 'soybean' : 21, 'turmeric' : 22, 'watermelon' : 23, 'wheat' : 24 }

## Chapter 7 : Model Training, Testing & Selection

### 7. Model Training, Testing & Selection

#### 7.1 Processes For Every Models

- 7.1.1 Object Creation
- 7.1.2 Hyperparameter Tuning Using Grid Search
- 7.1.3 Selection Of Best Parameters
- 7.1.4 Update Parameters
- 7.1.5 Training Model Base on Training Data
- 7.1.6 Prediction
- 7.1.7 Accuracy

#### 7.2 List of Models

##### 7.2.1 Decision Tree Classifier :-

A Decision Tree Classifier is a supervised machine learning algorithm used for classification tasks. It works by creating a tree-like structure where,

- **Nodes** : Represent features or attributes in the dataset.
- **Branches** : Represent decision rules or conditions based on those features.
- **Leaves** : Represent the predicted class labels

##### a) Object Creation :-

Create an object for Decision Tree using `DecisionTreeClassifier()` class from scikit-learn. Using this object, we train the model and perform all classification tasks.

Creating a Decision Tree Classifier object :

```
clf = DecisionTreeClassifier()
```

##### b) Hyperparameter Tuning Using Grid Search :-

Hyperparameter tuning is a technique in machine learning used to find optimal combination of hyperparameters for a model using Grid Search.

##### What is Grid Search?

Grid Search is a complete search method.

##### It involves :-

- Defining a "grid" of possible values for each hyperparameter.
- Training and evaluating the model for every possible combination of these values.
- Selecting the combination that produces the best performance.

The GridSearchCV class from sklearn.model\_selection is used to perform grid search.

grid\_search.best\_params\_ :- It stores the optimal combination of hyperparameters that were found during the grid search process. It returns a dictionary.

The keys of this dictionary are the names of the hyperparameters.

The values are the corresponding optimal values that were determined by the grid search.

```
Return      { 'criterion' : 'gini',
               'max_depth' : 20,
               'max_features' : 5,
               'min_samples_leaf' : 2,
               'min_samples_split' : 2,
               'splitter' : 'best' }
```

But some time model not give best score using this parameter, so you try different values for parameter and select best values of parameters.

### c) Selection Of Best Parameters :-

- **criterion = 'gini' :-** By using 'gini' criterion we measured the impurity of node. It helps the tree decide which feature to split on.
  - Gini is calculated for every feature in the dataset.
  - The best feature and split are chosen based on minimum weighted Gini.
  - Lower Gini means better splits (more purity).
  - Formula :-

$$Gini = 1 - \sum_{i=1}^N (P_i)^2 \quad \& \quad Gini(split, P) = \sum_{i=1}^k \frac{n_i}{n} GINI(i)$$

- Where :-

$P_i$  : Probability of a data point belonging to class i.

N : Number of classes.

k : Number of subsets after splitting.

$n_i$  : Number of samples in subset i.

n : Total number of samples before splitting.

GINI(i) : Gini impurity of subset i.

- **splitter = 'best' :-** The splitter parameter controls how the decision tree chooses the feature to split at each node.  
'best' :- Selects the best feature based on the chosen criterion (gini, entropy, or log\_loss). Tries all possible splits and picks the one that reduces impurity the most.

- **max\_depth = 14** :- The maximum depth of the tree, Controls how deep the tree can grow, default value is 'None'. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min\_samples\_split samples.
- **max\_features = 5** :- The number of features to consider when looking for the best split.
- **min\_samples\_split = 5** :- The minimum number of samples required to split an internal node.
- **min\_samples\_leaf = 2** :- The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min\_samples\_leaf training samples in each of the left and right branches.
- **random\_state = 42** :- The **random\_state** parameter in DecisionTreeClassifier controls the **randomness** of the tree-building process. It ensures **consistent results** every time you train the model.

**d) Update Parameters :-**

Using following values of parameters mode get best accuracy score

```
clf = DecisionTreeClassifier(criterion = 'gini', splitter = 'best',
max_depth = 14, max_features = 5, min_samples_split = 5,
min_samples_leaf = 2, random_state=42)
```

**e) Fit the Model Base on Training Data :-**

Train or fit the model using **clf** object of **DecisionTreeClassifier**, using following code

```
clf.fit(X_train_scaler, Y_train_le)
```

X\_train\_scaler :- input features, Y\_train\_le :- output features

**f) Prediction :-**

Once a model is trained, .predict() takes **new input data** (X\_test\_scaler ) and returns **predicted outputs (labels or values)** based on the learned patterns.

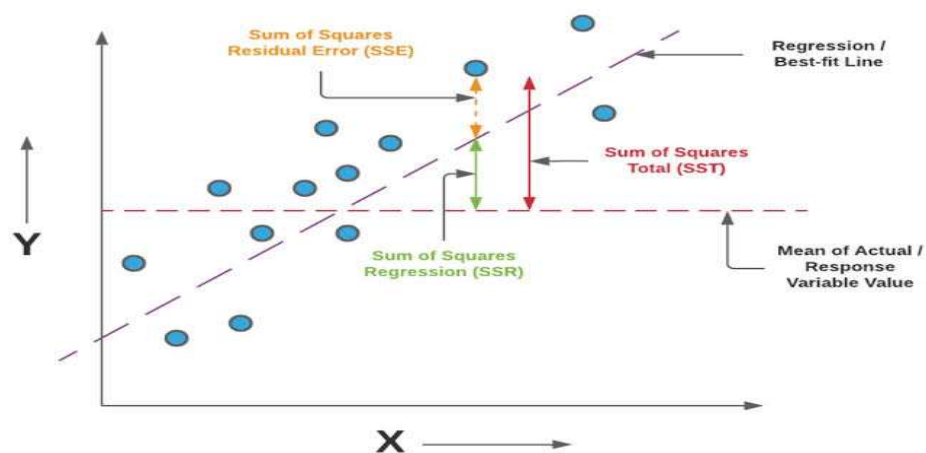
```
Code :- y_pred = clf.predict(X_test_scaler)
```

y\_pred is predicted outputs values.

**g) Accuracy :-**

- R-squared, or the coefficient of determination, is a statistical factor employed in machine learning to measure the goodness of a regression model. It calculates the fit of the model to the data by determining the fraction of variance in the dependent variable accounted for by the independent variables.
- Maximum score is 1.0 and can be negative (since the model can be arbitrarily worse). A constant model predicting always the expected value of y, without paying attention to the input features, would obtain a  $R^2$  score of 0.
- Code :- `r2_score(Y_test_le, y_pred)` we got 0.98 `r2_score`  
Formula :-

$$R^2 = 1 - \frac{SSE}{SST} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$



**7.2.2 SVC :-**

SVC stands for Support Vector Classification, which is a part of Support Vector Machines (SVM) used for classification tasks. It is implemented in Scikit-Learn as **from sklearn.svm import SVC**

- SVC is a classification algorithm that finds the best decision boundary (hyperplane) to separate different classes in the dataset.
- It works well for binary classification and multi-class classification problems.



**a. Object Creation :-**

Object creation in a Support Vector Classifier primarily involves instantiating the SVC class from scikit-learn. This creates a model that can then be trained and used for classification tasks.

Creating a Support Vector Classifier object

```
clf_svc = SVC()
```

**b. Hyperparameter Tuning Using Grid Search :-**

The values are the corresponding optimal values that were determined by the grid search.

```
Return { 'C' : 100,  
        'coef0' : 5,  
        'degree' : 3,  
        'gamma' : 0.1,  
        'kernel' : 'poly' }
```

**c. Selection Of Best Parameters :-**

- **C = 100 :-** C is a Regularization Parameter,
  - Controls the trade-off between maximizing the margin and minimizing classification errors.
  - High C (Small Margin): Model focuses more on correctly classifying every training example but may overfit.
  - Low C (Large Margin): Model allows more misclassifications, leading to better generalization.
- **kernel = 'poly' :-** Determines how the data is transformed into a higher-dimensional space.
  - Formula :-  $K(x_1, x_2) = (\gamma \cdot x_1^T x_2 + c)^d$
  - Where d is the polynomial degree,  $\gamma$  (gamma) controls the influence of each individual training sample on the decision boundary and c (coef0) is the bias term which moves the data up or down.
- **coef0 = 7 :-** Controls the influence of higher-order terms in the polynomial and sigmoid kernels.
- **Gamma = 0.8 :-** Controls how much influence a single training example has.
- **degree = 3 :-** (Only for poly Kernel) Specifies the degree of the polynomial kernel function.

**d. Update Parameters :-**

Using following values of parameters mode get best accuracy score

```
clf1_svc = SVC(C = 100 ,kernel = 'poly', coef0 = 7, degree = 3,  
gamma = 0.8)
```

**e. Fit the Model Base on Training Data :-**

Train or fit the model using `clf1_svc` object of SVC, using following code

```
clf1_svc.fit(X_train_scaler, Y_train_le)
```

`X_train_scaler` :- input features, `Y_train_le` :- output features

**f. Prediction :-**

Once a model is trained, predict the new input data using `clf1_svc.predict()` and return predicted output based on the learning patterns.

```
Code :- y_prd1_svc = clf1_svc.predict(X_test_scaler)
```

`y_prd1` is predicted outputs values.

**g. Accuracy :-**

Using `r2_score` we can check the accuracy of model

```
Code :- r2_score(Y_test_le,y_prd1_svc) we got 0.95 r2_score
```

Where `Y_test_le` : original outputs of data and `y_prd1_svc` : predicted outputs of data.

### 7.2.3 KNN :-

In the k-Nearest Neighbours (k-NN) algorithm k is just a number that tells the algorithm how many nearby points (neighbours) to look at when it makes a decision.

K-Nearest Neighbors is referred to as a lazy learner algorithm because it doesn't learn from the training set directly rather it saves the dataset and at classification time, it does something on the dataset.

**a. Object Creation :-**

Creating a KNN object

```
clf_knn = KNeighborsClassifier()
```

**b. Hyperparameter Tuning Using Grid Search :-**

The values are the corresponding optimal values that were determined by the grid search.

```
Return { 'algorithm' : 'ball_tree',  
        'leaf_size' : 10,  
        'n_neighbors' : 4,
```

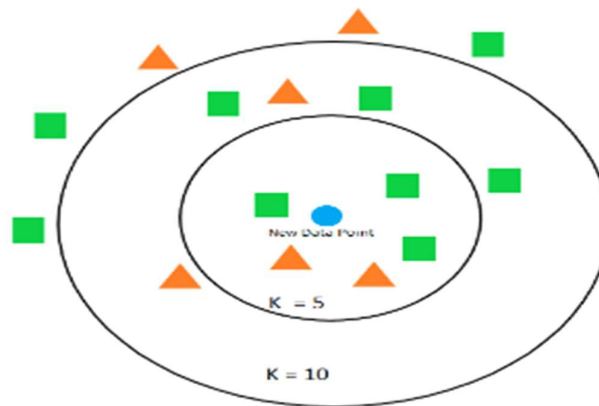
'p' : 1,  
'weights' : 'distance' }

**c. Selection Of Best Parameters :-**

- **algorithm = 'ball\_tree'** :- Algorithm used to compute the nearest neighbors

Ball Trees in KNN

- **Tree Construct** :- Each non-leaf node divided into two sub-node. Base on some criterion like minimizing variance. Each node in the tree represents a ball, which contains a subset of the dataset, and each ball is associated with a centre and a radius.
- **Test Point** :- 1st goes to root node then check nearest distance from centre of sub node then goes to nearest sub-node, and do this recursively finally test point fall in some sub-node, after that check majority point and predict



- **leaf\_size = 10** :- Leaf size passed to BallTree or KDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem.
- **n\_neighbors = 4** :- Number of neighbors required for each sample for predicting input sample point . The default is the value passed to the constructor.

- **p = 1** :- The parameter p in KNeighborsClassifier is used to define the Minkowski distance, which generalizes different distance metrics. It determines how distances between points are calculated. The Minkowski distance formula is :

$$d(x, y) = \left( \sum_{k=0}^n |x_k - y_k|^p \right)^{\frac{1}{p}}$$

Where,

$x_i$  and  $y_i$  are the coordinates of two points in n-dimensional space.

Power  $\frac{1}{p}$  for normalize the distance.

- **weights = 'distance'** :- The weights parameter in KNeighborsClassifier controls how the neighbors contribute to the prediction. if closer ones have more influence.
  - uniform weights determines, all points with in every neighborhood are given equal weight.
  - distance weight controls, nearest neighbors of a query point will have a stronger impact than neighbors that are further away.

#### d. Update Parameters :-

Using following values of parameters mode get best accuracy score

```
clf_knn = KNeighborsClassifier(algorithm = 'ball_tree', leaf_size = 10, n_neighbors = 4, p = 1, weights = 'distance')
```

#### e. Fit the Model Base on Training Data :-

Train or fit the model using clf\_knn object of KNeighborsClassifier, using following code

```
clf_knn.fit(X_train_scaler, Y_train_le)
```

#### f. Prediction :-

Once a model is trained, predict the new input data using `clf_knn.predict()`

and return predicted output based on the learning patterns.

Code :- `y_pred_knn = clf_knn.predict(X_test_scaler)`

`y_pred_knn` is predicted outputs values.

#### g. Accuracy :-

Using `r2_score` we can check the accuracy of model

Code :- `r2_score(Y_test_le, y_pred_knn)` we got 0.82 `r2_score`

And Using `accuracy_score()` we got 0.954 accuracy

Code :- `accuracy_score(Y_test_le,y_pred_knn)`

#### 7.2.4 **Bagging (Random Forest) :-**

Bagging, or bootstrap aggregating, is an ensemble machine learning method that improves model stability and accuracy by training multiple models on different subsets of the data, and then aggregating their predictions. This helps reduce variance and overfitting.

When all multiple models are Decision Trees then it called Random Forest

##### a. Object Creation :-

Create object of bagging classifier using `class BaggingClassifier()`

```
bag_clf = BaggingClassifier()
```

##### b. Hyperparameter Tuning Using Grid Search :-

The values are the corresponding optimal values that were determined by the grid search.

```
Return { 'bootstrap': False,  
        'max_features': 4,  
        'max_samples': 500,  
        'n_estimators': 30 }
```

##### c. Selection Of Best Parameters :-

- `estimator = best_model` :- An estimator in machine learning refers to any model or algorithm that learns from data and makes predictions.
- `n_estimators = 63` :- The number of base estimators in the ensemble.
- `max_features = 3` :- The number of features to draw from X to train each base estimator.
- `max_samples = 500` :- The number of samples to draw from X to train each base estimator.
- `bootstrap = False` :- Whether samples are drawn with replacement. If False, sampling without replacement is performed.

##### d. Update Parameters :-

Using following values of parameters mode get best accuracy score

```
bag_clf = BaggingClassifier(estimator = best_model, n_estimators = 63,  
max_features = 3, max_samples = 500, bootstrap = False,  
random_state=42)
```

**e. Fit the Model Base on Training Data :-**

Train or fit the model using bag\_clf object of BaggingClassifier(), using following code

```
bag_clf.fit(X_train_scaler, Y_train_le)
```

**f. Prediction :-**

Once a model is trained, predict the new input data using bag\_clf.predict() and return predicted output based on the learning patterns.

```
Code :- y1_pred = bag_clf.predict(X_test_scaler)
```

y1\_pred is predicted outputs values.

**g. Accuracy :-**

Using r2\_score we can check the accuracy of model

```
Code :- r2_score(Y_test_le, y1_pred) we got 0.99 r2_score
```

**7.2.5 Stacking :-**

Stacking is an ensemble learning technique that contain multiple classifications or regression model, also called as Stacking Generalization. The stacking algorithm train different model(Base Model) independently on training dataset, then combine the output of base model and predicts the final output using other model, called the meta-model. The idea is that different models can learn different aspects of the problem, improving overall performance.

## Chapter 8 & 9 : Selection Of Best Model & Confusion Matrix and Heat Map

Above I Used 5 type of different ML algorithms that is Decision Tree Classifier, SVC, KNN, Bagging and Stacking.

Algorithms with their accuracy result in percentage are as follows.

Algorithms	Accuracy
Decision Tree Classifier (DT)	98.16 %
<b>Random Forest</b>	<b>99.02 %</b>
SVC (Support Vector Classifier)	94.80 %
KNN (k-Nearest Neighbors classifier)	82.50 %
Bagging Classifier for KNN	98 %
Stacking (DT, SVC, KNN, LR)	98 %

**But I achieved highest accuracy of 0.99 with Random Forest, which is Bagging Classifier.**

So, I finally selected Random Forest for this 'Predictive Modeling for Soil-Crop Compatibility Using ML Algorithms' project.

### Confusion Matrix & Heat Map

A Confusion Matrix is a table used to evaluate the performance of a classification model. It compares the actual target values with the predicted values

Code :- `disp = confusion_matrix(Y_test_le, y1_pred)`

Confusion Matrix Structure

Actual / Predicted	Predicted: Positive (1)	Predicted: Negative (0)
<b>Actual : Positive (1)</b>	True Positive (TP)	False Negative (FN)
<b>Actual : Negative (0)</b>	False Positive (FP)	True Negative (TN)

- TP (True Positive) : Model correctly predicted Positive
- TN (True Negative) : Model correctly predicted Negative
- FP (False Positive) : Model incorrectly predicted Positive (Type I error)
- FN (False Negative) : Model incorrectly predicted Negative (Type II error)

**Heat Map :-** A heatmap is a graphical representation of data where individual values are represented by color intensity. It is commonly used to visualize correlations, confusion matrices, feature importance, and large datasets.

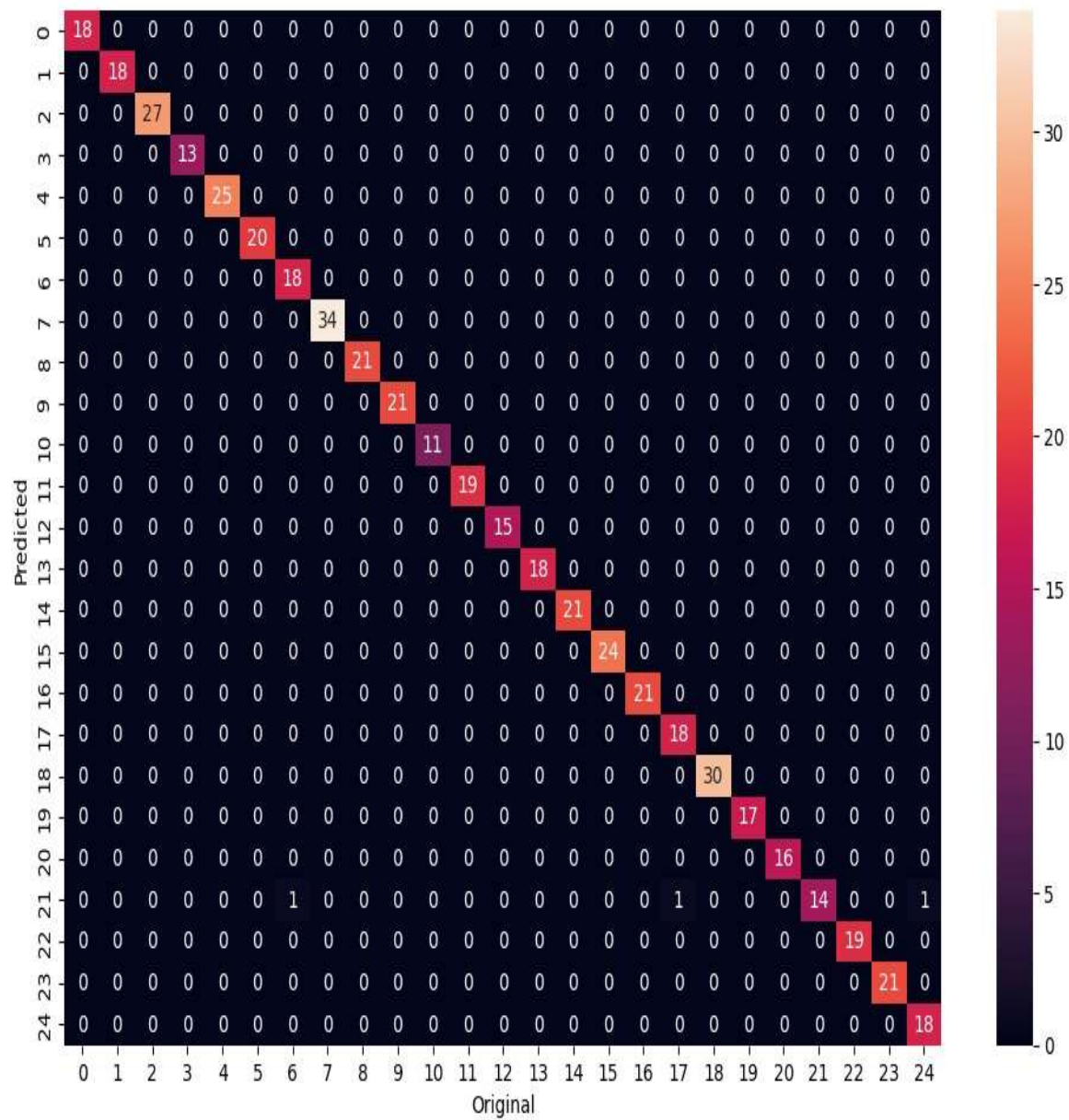
Code :- `sns.heatmap(dis,annot=True)`

`plt.xlabel("Original")`

`plt.ylabel("Predicted")`

`plt.show()`

### Heat Map :-





## **Chapter 10 : Conclusion**

In this project, I worked on different ML algorithms like Decision Tree (DT), Random Forest (RF), K-Nearest Neighbors (KNN), Support Vector Classifier (SVC), Bagging, and Stacking. For getting best accuracy model for this project, finally I achieved highest accuracy of 99% using Random Forest (RF).

Now, this project is ready for farmers and farmer will choose the best crop for their farm based on input parameter such as N(kg/ha), P (kg/ha), K (ka/ha), Temperature (°C), humidity, PH of soil and Rainfall (cm). The ML Algorithm takes these input values and suggests the suitable crop for the farmer's farm.

By using this project, we can increases their profits and reduce the risk of losses..

My final goal is to develop an ML algorithm that helps farmers choose the best crop for their farm based on given input parameters.

**GitHub Link of this code :-**

## Chapter 11 : References

- 1) International Journal of Scientific Research in Computer Science, Engineering and Information Technology, “Crop Recommendation System using Machine Learning” ( Dhruvi Gosai, Chintal Raval, Rikin Nayak, Hardik Jayswal, Axat Patel), Article Volume 7, Issue 3 Page Number 554-557, Publication Issue : May-June-2021
- 2) Manual Soil-Site Suitability Criteria for Major Crops, National Bureau of Soil Survey and Land Use Planning (ICAR) Amravati Road Nagpur- 440 010, ( L.G.K. Naidu, V. Ramamurthy, O. Challa, Rajendra Hegde, P. Krishnan), March 2006
- 3) International Journal of Advanced Trends in Computer Science and Engineering, “Hybrid Method for Improving Accuracy of Crop-Type Detection using Machine Learning”, (Ankit.Arun.Mahule , Dr.A.J.Agrawal ) Volume 9 No.2, March - April 2020.
- 4) International Journal of Engineering Applied Sciences and Technology, “Prediction of crop yield using machine learning”, Prof. Farhana Kausar, Vol. 4, Issue 9, ISSN No. 2455-2143, Pages 153-156, January 2020 in IJEAST.
- 5) International Conference on Communication and Signal Processing, “Recommendation System for Crop Identification and Pest Control Technique in Agriculture”, (Avinash Kumar, Sobhangi Sarkar and Chittaranjan Pradhan), pp. 0185-0189. IEEE, April 4-6, 2019, India