VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY
UNIVERSITY OF INFORMATION TECHNOLOGY
FACULTY OF INFORMATION SYSTEMS

PROJECT FINAL REPORT

DATA MINING

CROP YIELD PREDICTION

Class Code: IS502.N11.HTCL

Instructors: PhD. Cao Thị Nhạn, Ths. Nguyễn Hồ Duy Trí

20521643 – Bùi Thị Thanh Ngân

20521285 – Hồ Thị Hằng

20521292 – Lê Thị Ngọc Hảo

19522325 – Lê Trịnh Thanh Thúy

Ho Chi Minh City, October 2022

Table of Contents

1 Introduction

Agriculture in India is livelihood for a majority of the population and can never be underestimated. However, the agricultural sector is going through most stressed phase in the last three decades. Indian agriculture is plagued by several problems; some of them are natural and some others are manmade. Nearly three-quarters of India's families depend on rural incomes. Every solution if rightly executed can make a huge difference.

In this paper, a machine learning model is proposed to get a crop recommendation that guarantees high yield based on many parameters (such as atmospheric conditions, type of fertilizer, soil, and seed, etc.). During the development and evaluation of our model, we will show the code used for each step followed by its output. This will facilitate the reproducibility of our work. In this study, Python programming language with a number of Python packages will be used.

1.1 Goal of the Study

The main objectives of this study are as follows:

- To apply data preprocessing and preparation techniques in order to obtain clean data
- To build machine learning models able to recommend a crop based on crop features
- To analyze and compare models' performance in order to choose the best model

1.2 Paper Organization

This paper is organized as follows: in the next section, section 3, we go through data preparation including data cleaning, outlier removal, and feature engineering. Next in section 4, we discuss the type of our problem and the type of machine-learning prediction that should be applied; we also list the prediction techniques that will be used. In section 5, we choose algorithms to implement the techniques. In section 4, we build models based on these algorithms; we also train and test each model. In section 6, we analyze and compare the results we got from section 5 and conclude the paper.

2 Literature Review

Given the significance of crop prediction, numerous suggestions have been proposed in the past with the goal of improving crop prediction accuracy. In this paper feed-forward

back propagation Artificial Neural Network methodology has been approached to model and forecast various crop yields at rural areas based on parameters of soil(PH, nitrogen, potassium, etc.) and parameters related to the atmosphere (rainfall, humidity, etc.) [1].

This paper looks at five of Tamil Nadu's most important crops- rice, maize, ragi, sugarcane, and tapioca during a five-year period beginning in 2005. [2]. In order to get the maximum crop productivity, various factors such as rainfall, groundwater, and cultivation area, and soil type were used in the analysis. K-Means technique was used for the clustering, and for the classification, the study looked at three different types of algorithms: fuzzy, KNN, and Modified KNN. After the analysis, MKNN gave the best prediction result of the three algorithms.

An application for farmers can be created that will aid in the reduction of many problems in the agriculture sector [3]. In this application, farmers perform single/multiple testing by providing input such as crop name, season, and location. As soon as one provides the input, the user can choose a method and mine the outputs. The outputs will show you the crop's yield rate. The findings of the previous year's data are included in the datasets and transformed into a supported format. The machine learning models used are Naïve Bayes and KNN.

To create the dataset, information about crops over the previous ten years was gathered from a variety of sources, such as government websites. An IoT device was setup to collect the atmospheric data using the components like Soil sensors, Dht11 sensor for humidity and temperature, and Arduino Uno with Atmega as a processor. Naive Bayes, a supervised learning algorithm obtaining an accuracy of 97% was further improved by using boosting algorithm, which makes use of weak rule by an iterative process to bring higher accuracy [5]. To anticipate the yield, the study employs advanced regression techniques such as ENet, Kernel Ridge, and Lasso algorithms [4]. The three regression techniques are improved by using Stacking Regression for better prediction.

However, when a comparison study is conducted between the existing system and the proposed system employing Naive Bayes and Random Forest, respectively. The proposed system comes out on top. Because it is a bagging method, the random forest algorithm has

a high accuracy level, but the Naïve Bayes classifier's accuracy level is lower as the algorithm is probability based. [6].

This paper contributes to the following aspects- (a) Crop production prediction utilizing a range of Machine Learning approaches and a comparison of error rate and accuracy for certain regions. (b) An easy-to-use mobile app that recommends the most gainful crop. (c) A GPS-based location identifier that can be used to obtain rainfall estimates for a specific location. (d) A system that recommends the prime time to apply fertilizers [7]. On the given datasets from Karnataka and Maharashtra, different machine learning algorithms such as KNN, SVM, MLR, Random Forest, and ANN were deployed and assessed for yield to accuracy [9]. The accuracy of the above algorithms is compared. The results show that Decision Tree is the most accurate of the standard algorithms used on the given datasets, with a 99.87% accuracy rate.

Regression Analysis is applied to determine the relationship between the three factors: Area Under Cultivation, Food Price Index, and Annual Rainfall and their impact on crop yield. The above three factors are taken as independent variables, and for the dependent variable, crop yield is taken into consideration. The R2 obtained after the implementation of RA shows these three factors showed slight differences indicating their impact on the crop yield [8].

In the proposed paper, the dataset is collected from the government websites such as APMC website, VC Farm Mandya, which contains data related to climatic conditions and soil nutrients [10]. Two machine learning models were used; the model was trained using the Support Vector Machine model with Radial Basis Function kernel for rainfall prediction and Decision Tree for the crop prediction.

A comparative study of various machine learning can be applied on a dataset with a view to determine the best performing methodology. The prediction is found by applying the Regression Based Techniques such as Linear, Random Forest, Decision Tree, Gradient Boosting, Polynomial and Ridge on the dataset containing details about the types of crops, different states, and climatic conditions under different seasons [11]. The parameters used to estimate the efficiency of these techniques were mean absolute error, root mean square

error, mean squared error, R-square, and cross validation. Gradient Boosting gave the best accuracy- 87.9% for the target variable 'Yield' and Random Forest- 98.9% gave the best accuracy for the target value 'Production'.

The DHT22 sensor is recommended for monitoring live temperature and humidity [12]. The surrounding air is measured with a thermistor and a capacitive humidity sensor and outputs a digital signal on the data pin to the Arduino Uno port pin. The humidity value ranges from 0-100% RH and - 40 to 80 degrees Celsius to read the temperature. The above two parameters and soil characteristics are considered as input to three different machine learning models: Support Vector Machine, Decision Tree, and KNN. The Decision Tree gave better accuracy results.

Author	Proposed Model	Accuracy
M.Kalimuthu et.al(2020)[5]	Naïve Bayes	97%
V.Geetha et.al(2020)[6]	Naïve Bayes and Random Forest Classifier	95% Random Forest Classifier
Shilpa Mangesh P et.al(2021)[7]	Support Vector Machine, K-Nearest Neighbor, Multivariate Linear Regression, Artificial Neural Network, Random Forest	95% Random Forest Classifier
S Bharah et.al(2020([9]	Support Vector Machine, Decision Tree, K-Nearest Neighbor, Random Forest	99.87% Decision Tree Classifier
Payal Gulati Suman Kumar(2020)[11]	Linear Regression, Random Forest, Decision Tree, Gradient Boosting Regression, Ridge Regression, Polynomial Regression	98.9% Random Forest Classifier
Archana Gupta et.al(2020)[12]	K-Nearest Neighbor, Support Vector Machine, Decision Tree	91.03% Decision Tree Classifier

3 Data Preparation

In this study, we will use a crop growth dataset. This dataset was built by augmenting datasets of rainfall, climate, and fertilizer data available for India. The dataset contains a large number of variables that are involved in growing a crop based on various parameters.

We obtained a .csv copy of the data from https://www.kaggle.com/datasets/atharvaingle/crop-recommendation-dataset.

2.1 Data Description

The dataset contains 2200 records (rows) and 8 features (columns). Here, we will provide a brief description of dataset features.

Feature	Description
N	ratio of Nitrogen content in soil
P	ratio of Phosphorous content in soil
K	ratio of Potassium content in soil
temperature	temperature in degree Celsius
humidity	relative humidity in %
ph	ph value of the soil
rainfall	rainfall in mm
label	crop name

2.2 Reading the dataset

After importing some essential libraries, the next step is reading the dataset from the .csv file we downloaded. We will use the read_csv() function from Pandas Python package:

```
df = pd.read_csv('plant.csv')
```

Let's display the first few rows of the dataset to get a feel of it:

```
pd.options.display.float_format ='{:20.2f}'.format
df.head(n=5)
```

	N	P	K	temperature	humidity	ph	rainfall	label
0	90	42.00	43.00	20.88	82.00	6.50	203.98	rice
1	85	58.00	41.00	22.00	80.32	7.04	226.66	rice
2	60	55.00	44.00	23.00	82.32	7.84	263.96	rice
3	74	35.00	40.00	26.49	80.16	6.98	242.86	rice
4	78	42.00	42.00	20.13	81.60	7.63	262.72	rice

Now, let's get statistical information about the numeric columns in our dataset. We want to know the mean, the standard deviation, the minimum, the maximum, and the 50th percentile (the median) for each numeric column in the dataset.

```
df.describe(include=[np.number], percentiles=[.5]) \
.transpose().drop("count", axis=1)
```

	mean	std	min	50%	max
N	50.56	36.92	0.00	37.00	140.00
P	53.33	32.97	5.00	51.00	145.00
K	48.18	50.67	5.00	32.00	205.00
temperature	25.63	5.07	8.83	25.60	43.68
humidity	71.53	22.26	14.26	80.52	99.98
ph	6.47	0.77	3.50	6.43	9.94
rainfall	103.66	54.98	20.21	95.09	298.56

Then, we move to see statistical information about the non-numerical columns in our dataset.

```
df.describe(include=[np.object]).transpose() \
.drop("count", axis=1)
```

	unique	top	freq
label	22	rice	100

2.3 Data Cleaning

2.3.1 Dealing with Missing Values

We should deal with the problem of missing values because some machine learning models don't accept data with missing values. Firstly, let's see the number of missing values in our dataset. We want to see the number of missing values for each column that actually contains missing values.

```
# Getting the number of missing values in each column
num_missing = df.isna().sum()

# Excluding columns that contains 0 missing values
num_missing = num_missing[num_missing > 0]

# Getting the percentages of missing values
percent_missing = num_missing * 100 / df.shape[0]

# Concatenating the number and perecentage of missing values
# into one dataframe and sorting it
pd.concat([num_missing, percent_missing], axis=1,
keys=['Missing Values', 'Percentage']).\
sort_values(by="Missing Values", ascending=False)
```

	Missing Values	Percentage
humidity	32	1.45
temperature	30	1.36
ph	30	1.36
rainfall	26	1.18
label	20	0.91
K	3	0.14
P	1	0.05

Now we start dealing with these missing values.

We will replace the values of P and K, we will fill in missing values with 0 since this is a numerical column:

```
df.P = df.P.fillna(value = 0)
df.K = df.K.fillna(value = 0)
```

With temperature, humidity, ph, and rainfall, we will fill in this value with the mean of these column:

```
df.temperature = df.temperature.fillna(value = df.temperature.mean())
df.humidity = df.humidity.fillna(value = df.humidity.mean())
df.ph = df.ph.fillna(value = df.ph.mean())
df.rainfall = df.rainfall.fillna(value = df.rainfall.mean())
```

With label attribute, we will fill in this value with the mode of this column:

```
df.label = df.label.fillna(value = df.label.mode()[0])
```

Now let's check if there is any remaining missing value in our dataset:

```
df.isna().sum()
```

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

This means that our dataset is now complete; it doesn't contain any missing values anymore.

4 Exploratory Data Analysis

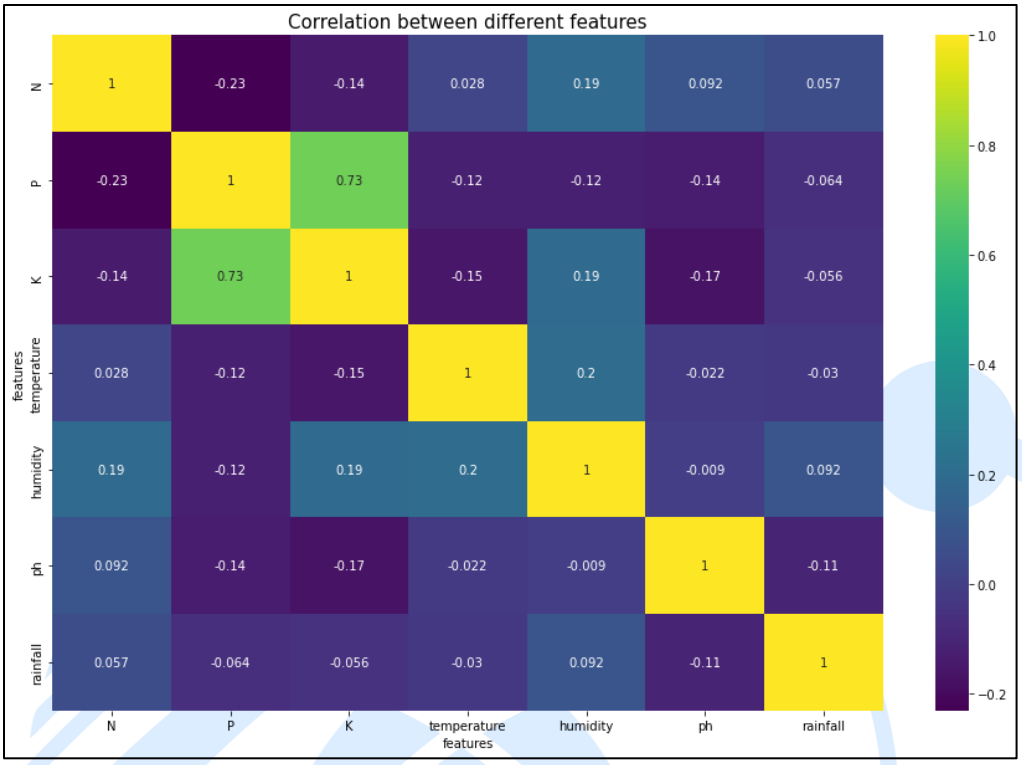
In this section, we will explore the data using visualizations. This will allow us to understand the data and the relationships between variables better, which will help us build a better model.

4.1 Correlation Between Variables

We want to see how the dataset variables are correlated with each other and how predictor variables are correlated with the target variable. For example, we would like to see how humidity and label are correlated: Do they increase and decrease together (positive correlation)? Does one of them increase when the other decrease or vice versa (negative correlation)? Or are they not correlated?

Correlation is represented as a value between -1 and +1 where +1 denotes the highest positive correlation, -1 denotes the highest negative correlation, and 0 denotes that there is no correlation. We will show the correlation between our dataset variables (numerical and boolean variables only) using a heatmap graph.

```
fig, ax = plt.subplots(1, 1, figsize=(15, 10))
correlation = df.corr()
sns.heatmap(correlation , annot = True , cmap = 'viridis')
ax.set(xlabel='features')
ax.set(ylabel='features')

plt.title('Correlation between different features', fontsize = 15, c='black')
plt.show()
```

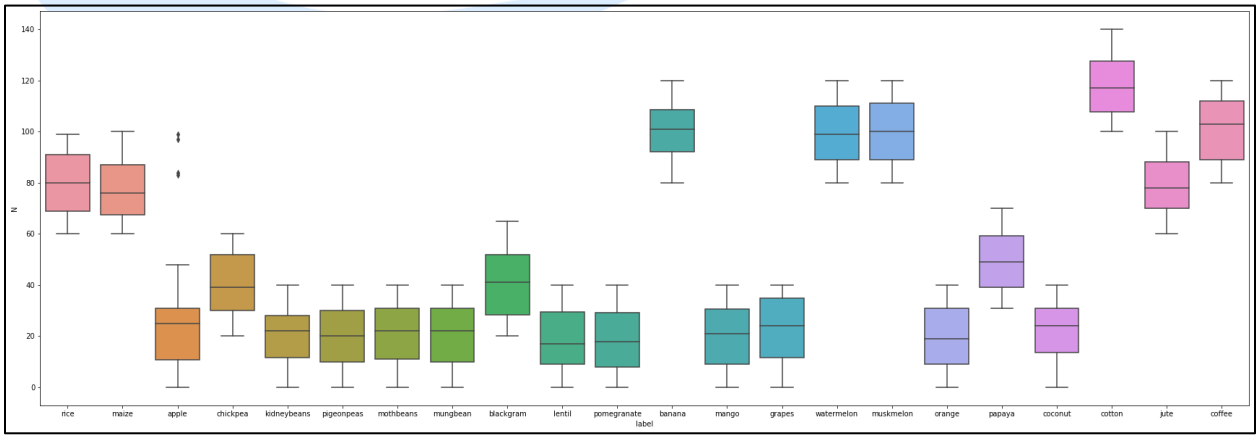


We can see that there is just a few correlated variables in our dataset. We notice that P and K have high positive correlation which is reasonable because both P and K are help form new roots, make seeds, fruit and flowers and help plants to fight desease.

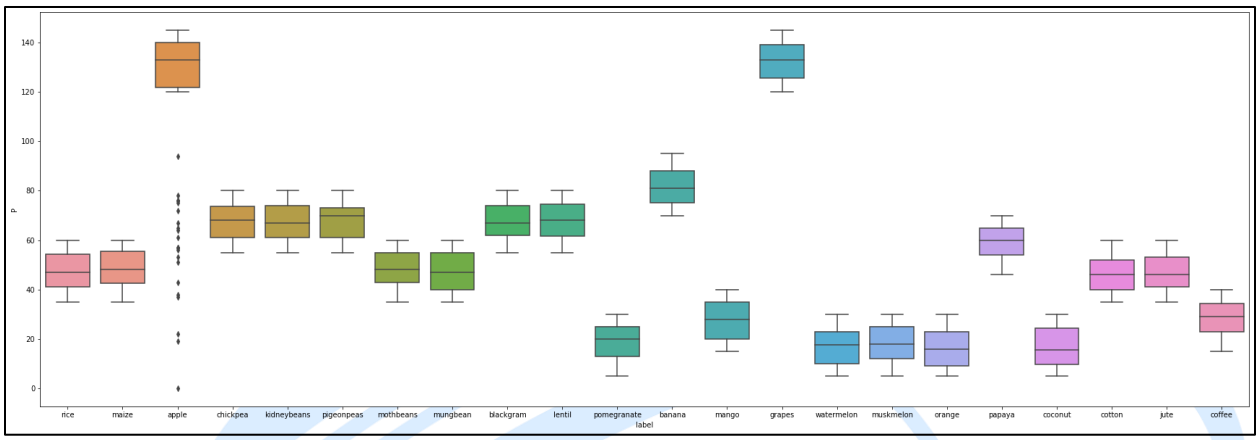
Most importantly, we want to look at the predictor variables that are correlated with the target variable (label).

4.2 Relationships Between the Target Variable and Other Varibles

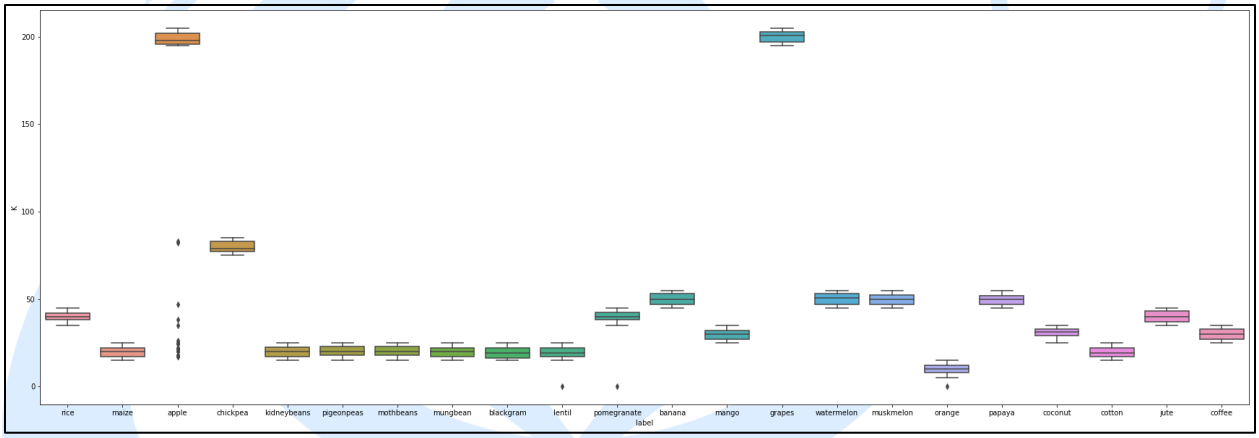
```
plt.figure(figsize = (30 , 10))
sns.boxplot(data = df , x = 'label' , y = 'N')
```



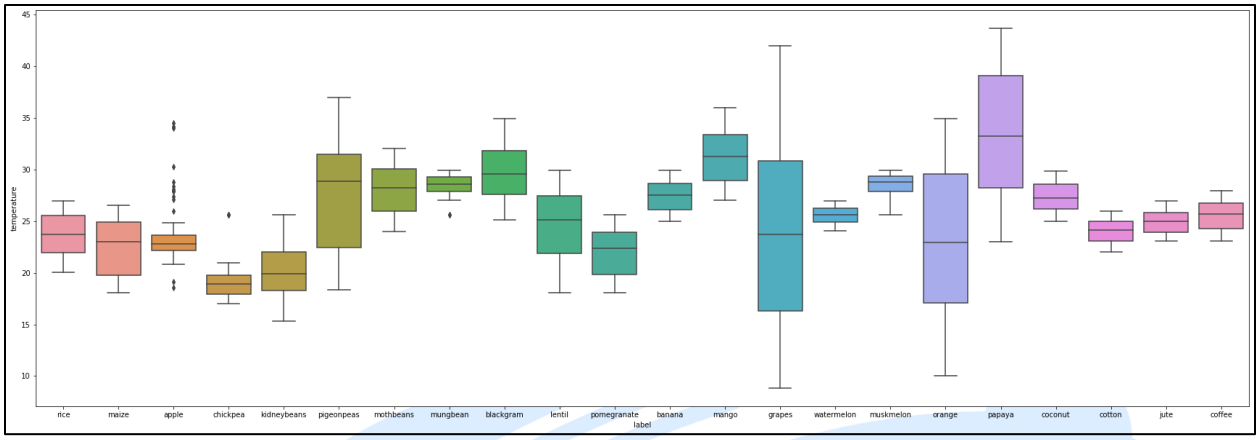
```
plt.figure(figsize = (30 , 10))
sns.boxplot(data = df , x = 'label' , y = 'P')
```



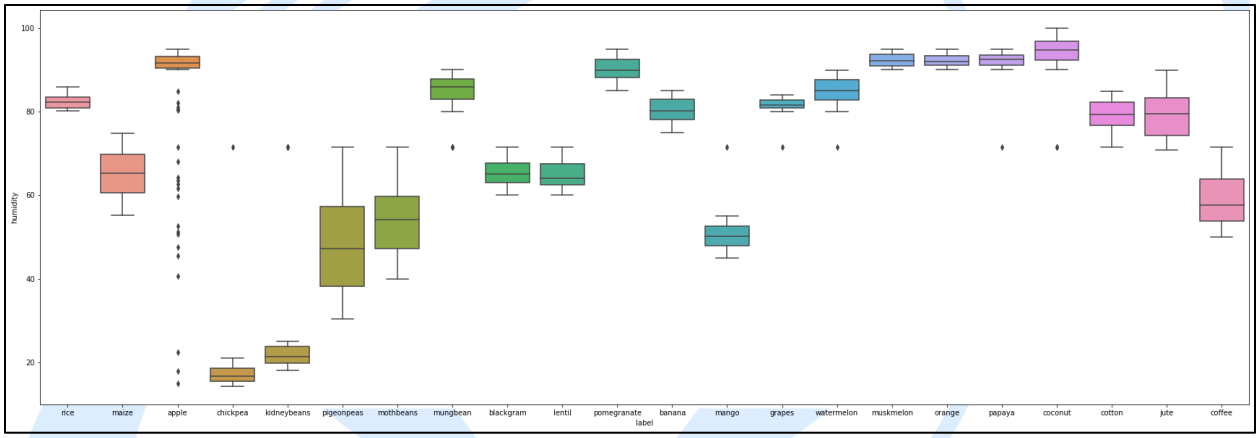

```
plt.figure(figsize = (30 , 10))
sns.boxplot(data = df , x = 'label' , y = 'K')
```



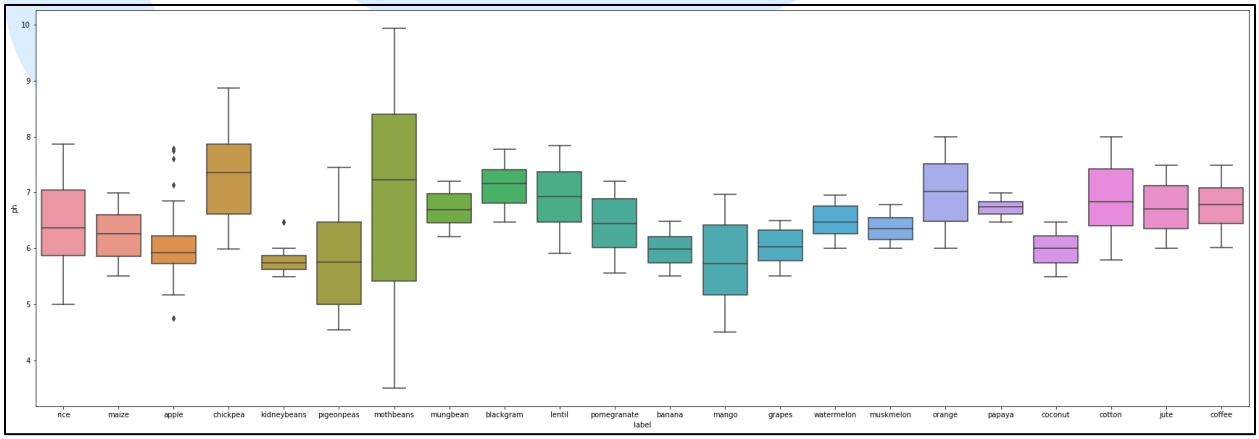

```
plt.figure(figsize = (30 , 10))
sns.boxplot(data = df , x = 'label' , y = 'temperature')
```



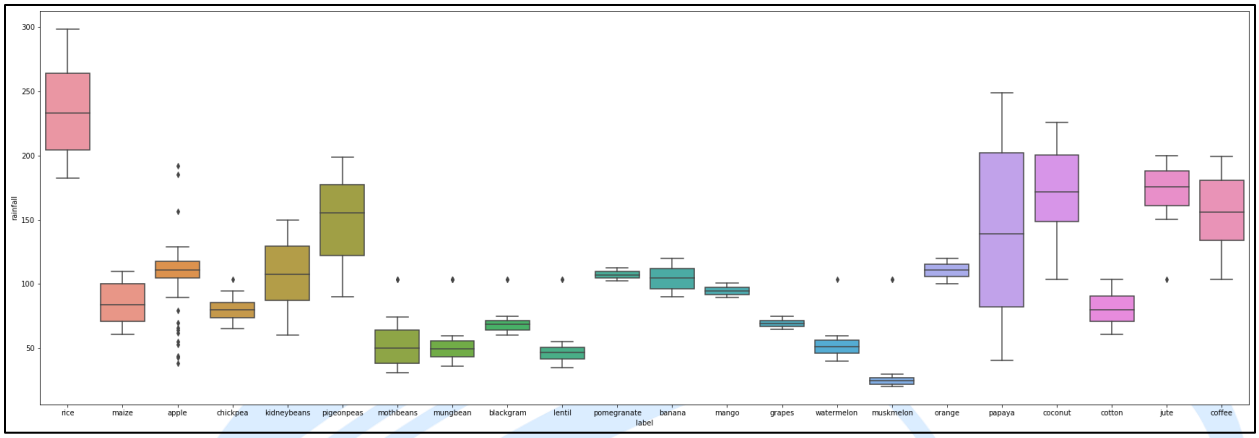

```
plt.figure(figsize = (30 , 10))
sns.boxplot(data = df , x = 'label' , y = 'humidity')
```




```
plt.figure(figsize = (30 , 10))
sns.boxplot(data = df , x = 'label' , y = 'ph')
```



```
plt.figure(figsize = (30 , 10))
sns.boxplot(data = df , x = 'label' , y = 'rainfall')
```



4.3 Feature Engineering

4.3.1 Creating New Derived Features

According to the original dataset, we can see that rainfall is in mm parameter so to have a clearly analyze in this attribute, we add another attribute which name "Water Usage". We can distributed rainfall attribute into 3 categories: high, medium, low to represent for water scarcity, if the mm of rainfall <= 150 → the water usage is low, if the mm of rainfall > 250 → the water usage is high, otherwise, the water usage is medium.

```
categor_condn = [(df['rainfall'] <= 150),
                 (df['rainfall'] > 250)]

rating = ['low','high']
df['Water Usage'] = np.select(categor_condn,rating,default = 'medium')

df.head()
```

	N	P	K	temperature	humidity	ph	rainfall	label	Water Usage
0	90	42.00	43.00	20.88	82.00	6.50	203.98	rice	medium
1	85	58.00	41.00	22.00	80.32	7.04	226.66	rice	medium
2	60	55.00	44.00	23.00	82.32	7.84	263.96	rice	high
3	74	35.00	40.00	26.49	80.16	6.98	242.86	rice	medium
4	78	42.00	42.00	20.13	81.60	7.63	262.72	rice	high

4.3.2 Dealing with Ordinal Variable

There is only one ordinal feature left in our dataset. For example, the Water Useage feature has the following possible values with the total numbers of each values:

```
df['Water Usage'].value_counts()
```

```
low       1779
medium     383
high        38
Name: Water Usage, dtype: int64
```

But the problem is that machine learning models will not know that this feature represents a ranking; it will be treated as other categorical features. So to solve this issue, we will map each one of the possible values of this feature to a number. We will map "high" to 0, "low" to 1, and "medium" to 2.

```
label_encoder = LabelEncoder()
```

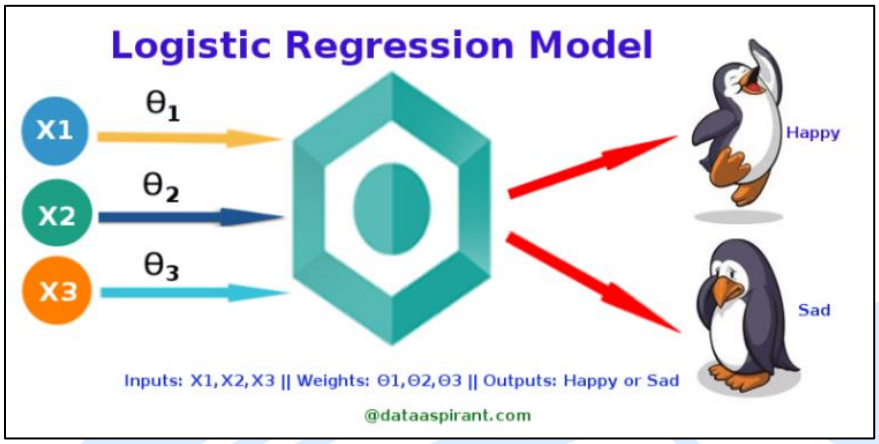
```
df['Water Usage'] = label_encoder.fit_transform(df['Water Usage'])
```

```
df.head(10)
```

	N	P	K	temperature	humidity	ph	rainfall	label	Water Usage
0	90	42.00	43.00	20.88	82.00	6.50	203.98	rice	2
1	85	58.00	41.00	22.00	80.32	7.04	226.66	rice	2
2	60	55.00	44.00	23.00	82.32	7.84	263.96	rice	0
3	74	35.00	40.00	26.49	80.16	6.98	242.86	rice	2
4	78	42.00	42.00	20.13	81.60	7.63	262.72	rice	0
5	69	37.00	42.00	23.06	83.37	7.07	251.05	rice	0
6	69	55.00	38.00	22.71	82.64	5.70	271.32	rice	0
7	94	53.00	40.00	20.28	82.89	5.72	241.97	rice	2
8	89	54.00	38.00	24.52	83.54	6.69	230.45	rice	2
9	68	58.00	38.00	23.22	83.03	6.34	221.21	rice	2

5 Prediction Type and Modeling Techniques

5.1 Logistic Regression

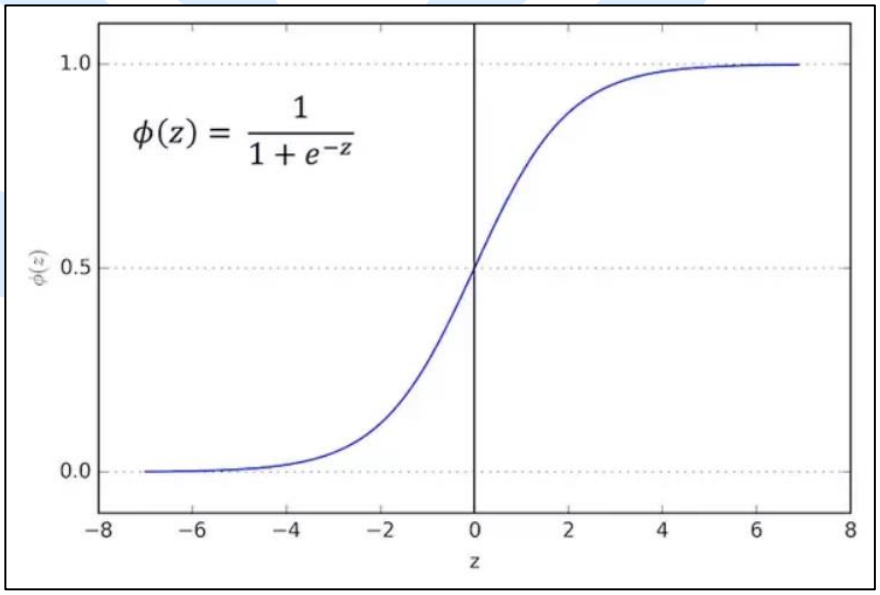


- **Definition**

Logistic regression is a statistical analysis model to predict a binary outcome division (1 or 0, Yes or No, True or False) for a set of independent variables.

Logistic regression can be considered a special case of linear regression when the outcome variable is categorical, where we are using the log of odds as a dependent variable. Simply put, it predicts the probability of an event occurring by how to match data with a logit function.

Sigmoid function is a function that always has a value in the range [0, 1], continuous and easy to use.



Sigmoid formula:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Loss function is a function that allows to determine the degree of deviation of the predicted result from the actual price to be predicted. It is a method of measuring the quality of a predictive model on an observational data set. If the prediction model is wrong, the value of the loss function will be larger and conversely, the more correct it is, the lower the value of the loss function will be.

$$J = -\sum_{i=1}^{N} (y_i * log(\hat{y}_i) + (1 - y_i) * log(1 - \hat{y}_i))$$

Gradient Descent algorithm gives us a way to find these local minima approximately after some iterations.

$$\theta_n = \theta_{n-1} - \eta \frac{\partial}{\partial \theta} L(\theta_{n-1})$$

Logistic Regression we have the following prediction function:

$$\hat{y}_i = \sigma(w_0 + w_1 * x_1^{(i)} + w_2 * x_2^{(i)}) = \frac{1}{1+e^{-(w_0 + w_1 * x_1^{(i)} + w_2 * x_2^{(i)})}}$$

We then use some probability threshold to classify the observation as either 1 or 0.

For example, we might say that observations with a probability greater than or equal to 0.5 will be classified as "1" and all other observations will be classified as "0."

- **Logistic Regression Algorithm**

Step 1: Initialize the parameter w_1, w_2, …w_n and β

Step 2: Use sigmoid function to convert the result into rank [0,1]

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Step 3: Evaluate the weight vector w

$$J = -\sum_{i=1}^{N} (y_i * log(\hat{y}_i) + (1 - y_i) * log(1 - \hat{y}_i))$$

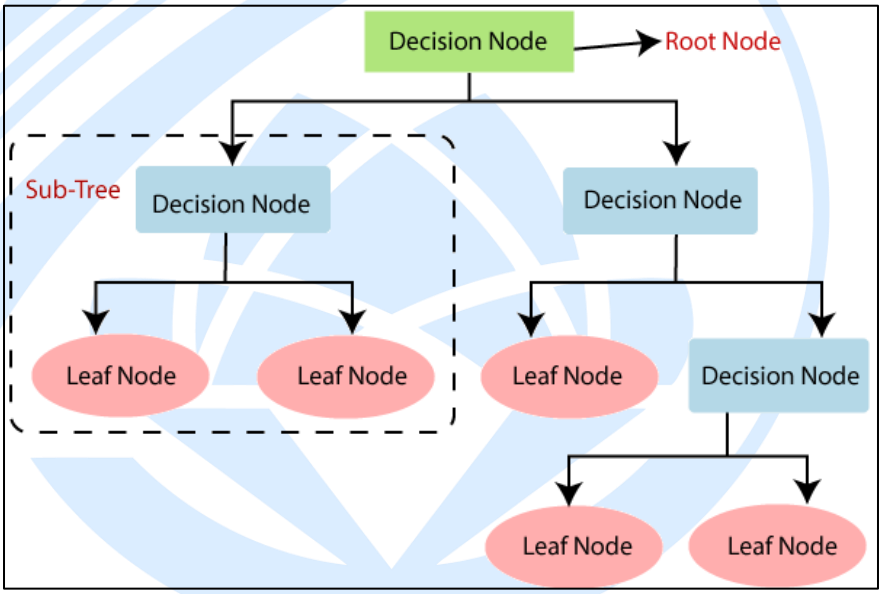
Step 4: Calculate the cost function

$$J(w) = \frac{1}{n} \sum_{i=1}^{n} L^{(i)}(w)$$

Step 5: Using gradient descent to minimize. If the value of cost function is small enough, break the iteration, else repeat the process.

Step 6: End the algorithm.

5.2 Decision Tree Classifier



- **Definition**

Decision Tree is a supervised learning technique used for both classification and regression problems where each path is a set of decisions leading to a class. A sequence of questions is asked by taking an instance from the training set. The non-terminal node such as root and internal nodes has decision attributes.

A tree can be "learned" by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called recursive partitioning. The recursion is completed when the subset at a node all has the

same value of the target variable, or when splitting no longer adds value to the predictions.

- **ID3 Algorithm:** The criteria for measuring Information Gain are the Gini index and Entropy. Information gain is a measurement of how much information is gained about an attribute and the reduction in entropy. Entropy and Gini Index are the metrics that measure the impurity of the nodes. A node is considered impure if it has multiple classes else, it is considered pure.
    - **Entropy**: Entropy is a metric that gives the degree of impurity in a specified attribute. The following formula can be used to compute entropy:

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

        - In a dataset with C of type label, p_i is % or probability of S of type i
        - If the entropy value is lower, the algorithm will be better
        - The value of entropy is in the range (0, $\log_2 C$)
        - P purity: $p_i = 0$ or $p_i = 1$
        - P remains opaque: $p_i = 1/C$ (label types have equal probability of occurrence)
    - **Gini**: Gini is estimated by deducting the sum of squared probabilities of each class from one. The lower Gini Index value is preferred rather than a higher value. Scikit-learn takes "Gini" as the default value and supports "Gini" criteria for Gini Index.

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$

        - In a dataset with C of type label, pi is % or probability of S of type i
        - If Gini value is lower, the algorithm is better
        - The values of Gini is in the range (0, 0.5)
        - Gini's computation speed is faster than entropy because entropy has

log

- **Decision Tree Algorithm**

Step 1: Starting with the root node of the tree, which consists of the entire dataset, says S.

Step 2: The most appropriate attribute is obtained from the dataset by applying the Attribute Selection Measure (ASM).

Step 3: The S is divided into subdivisions that enclose feasible values for the most appropriate attributes.

Step 4: the node is formed in the decision tree with the most appropriate attribute.

Step 5: The tree information is set up by iteratively repeating this method for each child until one of the following requirements is met:

- The tuples are entirely correlated with the same attribute value.
- There are no further attributes accessible.
- There aren't any more instances.

- **Steps to Split**

The dataset used in teh project has numerical values. The decision tree works with the numerical values in the following ways:

Step 1: Sorting all the values.

Step 2: It will consider a threshold value from the feature values.

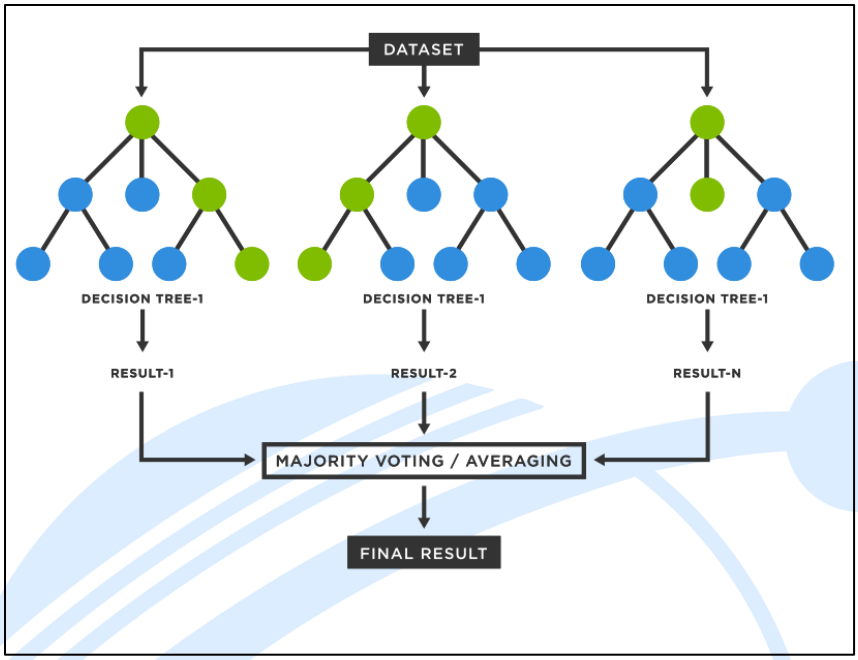
Step 3: Feature value will split into two parts such that the left node contains feature values less than a threshold value, and the right node contains feature values greater than a threshold value.

Step 4: The next feature value will consider as a threshold value and again create the same split as Step 3.

Step 5: Entropy/Gini and Information Gain are calculated of each split, and from the two splits, the split with better information gain is considered.

Step 6: Repeat from Step 2 to Step 5. In this way, it will get branches for the decision tree.

5.3 Random Forest Classifier



- **Definition**

The Random Forest method consists of multiple decision tree classifiers to enhance the model's performance. Here ensemble learning is the principle used to resolve complicated problems. It is a supervised learning algorithm. Decision trees are created at random using the instances from the training set. Each of the decision trees gives out predictions as their outcome. The final prediction for the model is decided by majority voting. One of the reasons for its popularity as a machine learning approach is that it can handle the issue of overfitting, and accuracy can be increased by using more trees.

- **Random Forest Algorithm**

Step 1: K instances are chosen at random from the given training dataset.

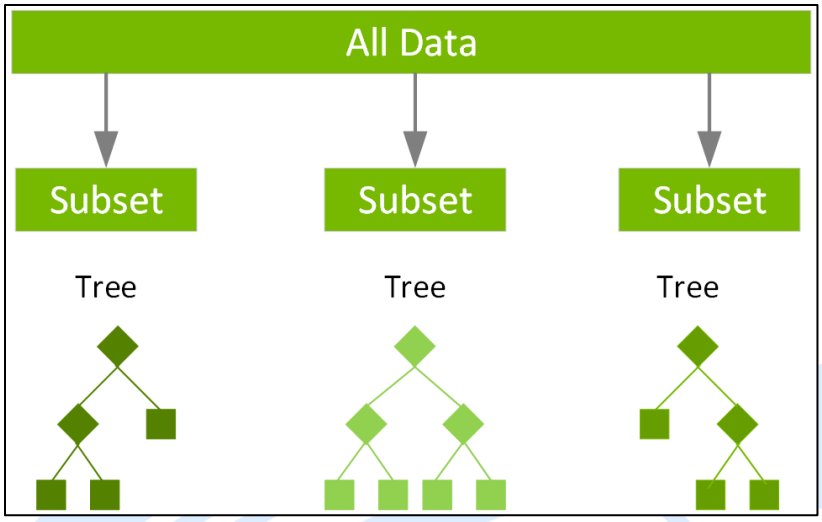
Step 2: Decision trees are created for the chosen instances.

Step 3: The N is selected for the number of estimators to be created.

Step 4: Step 1 & Step 2 is repeated.

Step 5: For the new instance, the predictions of each estimator is determined, the category with the highest vote is assigned.

5.4 XGBoost



- **Definition**

XGBoost regression is short form for extreme gradient boost regression. It works well compared to others machine learning algorithms. XGBoost is one of the best supervised learning algorithms which can inferred by the way it flows, it consists of objective function and base learners. Loss function is present in objective function which shows the difference between actual values and predicted values whereas regularisation term is used for showing how far is actual value away from predicted value. Ensemble learning used in XGBoost considers many models which are known as base learners for predicting a single value.

Boosting will create a series of weak models, learning to complement each other. In other words, in Boosting, the following models will try to learn to limit the mistakes of the previous models.

To limit mistakes from previous models, Boosting weights the newly added models based on different optimization methods. Depending on the weighting method (how the models fit sequentially) and how the models are aggregated, two types of Boosting are formed.

- Adaptive Boosting
- Gradient Boosting (XGBoost based on this algorithm)

- **XGBoost Algorithm**

Step 1: Initialize pseudo-residuals to be equal for each data point

Step 2: In the loop i, new train model will ne added to fit existing pseudo-residuals values

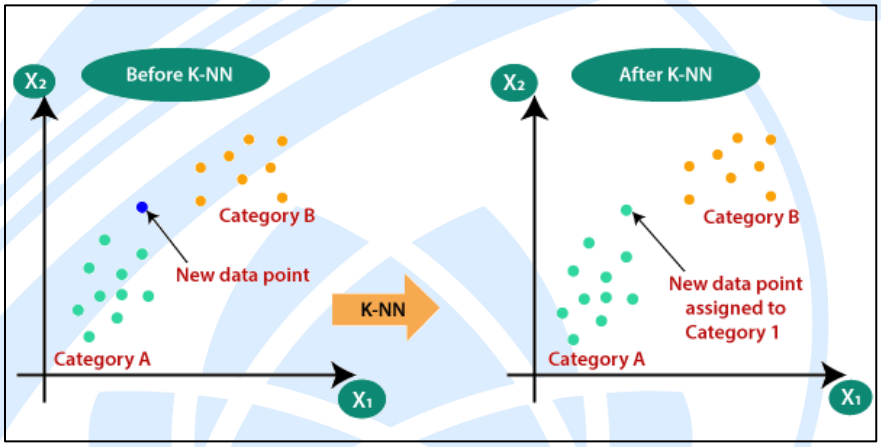
Calculate confidence score c_i of the model just trained

Update main model: $$W = W + c_i * w_i$$

Finally, calculate the value pseudo-residuals to make the label for the next model

Step 3: Then repeat with i + 1 loop.

5.5 K-Nearest Neighbours



- **Definition**

Nearest Neighbors is a type of instance-based learning. For this technique, the model tries to find a number (k) of training examples closest in distance to a new point, and predict the output for this new point from these closest neighbors. k can be a user-defined number (k-nearest neighbors) or vary based on the local density of points (radius-based neighbors). The distance metric used to measure the closeness is mostly the Euclidean distance.

- **KNN Algorithm**

Step 1: Select the number K of the neighbors

Step 2: Calculate the Euclidean distance of K number of neighbors

Step 3: Take the K nearest neighbors as per the calculated Euclidean distance.

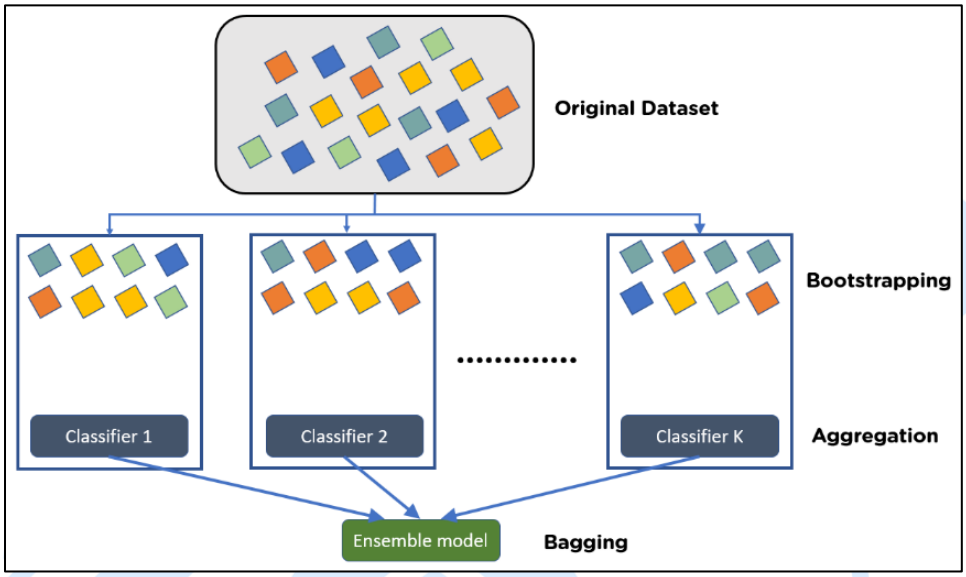
Step 4: Among these k neighbors, count the number of the data points in each category.

Step 5: Assign the new data points to that category for which the number of the

neighbor is maximum.

Step 6: Our model is ready.

5.6 Bagging Classifier



- **Definition**

A Bagging classifier is an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction. Such a meta-estimator can typically be used as a way to reduce the variance of a black-box estimator (e.g., a decision tree), by introducing randomization into its construction procedure and then making an ensemble out of it.

Each base classifier is trained in parallel with a training set which is generated by randomly drawing, with replacement, N examples(or data) from the original training dataset – where N is the size of the original training set. Training set for each of the base classifiers is independent of each other. Many of the original data may be repeated in the resulting training set while others may be left out.

- **Bagging Algorithm**
    - Consider there are n observations and m features in the training set. You need to select a random sample from the training dataset without replacement
    - A subset of m features is chosen randomly to create a model using sample

observations

- The feature offering the best split out of the lot is used to split the nodes.
- The tree is grown, so you have the best root nodes
- The above steps are repeated n times. It aggregates the output of individual decision trees to give the best prediction

6 Model Building and Evaluation

6.1 Splitting the Dataset

As usual for supervised machine learning problems, we need a training dataset to train our model and a test dataset to evaluate the model. So we will split our dataset randomly into two parts, one for training and the other for testing. For that, we will use another function from Scikit-Learn called train_test_split()

```
x_train , x_test , y_train , y_test = train_test_split(x,y, test_size = 0.30 , random_state = 10)
```

```
print("Dimension of x_train :",x_train.shape)
print("Dimension of x_test :",x_test.shape)
print("Dimension of y_train :",y_train.shape)
print("Dimension of y_test :",y_test.shape)
```

```
Dimension of x_train : (1540, 8)
Dimension of x_test : (660, 8)
Dimension of y_train : (1540,)
Dimension of y_test : (660,)
```

We specified the size of the test set to be 30% of the whole dataset. This leaves 70% for the training dataset. Now we have four subsets: x_train, x_test, y_train, and y_test. Later we will use x_train and y_train to train our model, and x_test and y_test to test and evaluate the model. x_train and x_test represent features (predictors); y_train and y_test represent the target. From now on, we will refer to x_train and y_train as the training dataset, and to x_test and y_test as the test dataset.

6.2 Modeling Approach

For each one of the techniques mentioned in the previous section (Logistic Regression, Nearest Neighbor, etc.), we will follow these steps to build a model:

- Choose an algorithm that implements the corresponding technique
- Search for an effective parameter combination for the chosen algorithm
- Create a model using the found parameters
- Train (fit) the model on the training dataset
- Test the model on the test dataset and get the results

6.3 Modeling

6.3.1 Logistic Regression

For Logistic Regression (LR), we will use an implementations provided by the Scikit-Learn package.

The Logistic Regression model has the following syntax:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

Firstly, we will use *GridSearchCV()* to search for the best model parameters in a parameter space provided by us.

```
# define models and parameters
logmodel = lm()
solvers = ['newton-cg', 'lbfgs', 'liblinear']
penalty = ['l2']
c_values = [100, 10, 1.0, 0.1, 0.01]

# define grid search
grid = dict(solver=solvers,penalty=penalty,C=c_values)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=logmodel, param_grid=grid, n_jobs=-1, cv=cv, scoring='accuracy',error_score=0)
grid_result = grid_search.fit(x_train, y_train)

# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.958225 using {'C': 0.01, 'penalty': 'l2', 'solver': 'newton-cg'}
0.947403 (0.017929) with: {'C': 100, 'penalty': 'l2', 'solver': 'newton-cg'}
0.926623 (0.018378) with: {'C': 100, 'penalty': 'l2', 'solver': 'lbfgs'}
0.941558 (0.021796) with: {'C': 100, 'penalty': 'l2', 'solver': 'liblinear'}
0.948918 (0.017959) with: {'C': 10, 'penalty': 'l2', 'solver': 'newton-cg'}
0.926190 (0.018781) with: {'C': 10, 'penalty': 'l2', 'solver': 'lbfgs'}
0.937879 (0.022032) with: {'C': 10, 'penalty': 'l2', 'solver': 'liblinear'}
0.954545 (0.018519) with: {'C': 1.0, 'penalty': 'l2', 'solver': 'newton-cg'}
0.928139 (0.020045) with: {'C': 1.0, 'penalty': 'l2', 'solver': 'lbfgs'}
0.935498 (0.021532) with: {'C': 1.0, 'penalty': 'l2', 'solver': 'liblinear'}
0.955195 (0.016543) with: {'C': 0.1, 'penalty': 'l2', 'solver': 'newton-cg'}
0.928571 (0.021471) with: {'C': 0.1, 'penalty': 'l2', 'solver': 'lbfgs'}
0.925974 (0.020438) with: {'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}
0.958225 (0.016314) with: {'C': 0.01, 'penalty': 'l2', 'solver': 'newton-cg'}
0.927489 (0.021542) with: {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
0.911472 (0.020567) with: {'C': 0.01, 'penalty': 'l2', 'solver': 'liblinear'}
```

Now we build our Logistic Regression model with the best parameters found:

```
logmodel1 = lm(solver='newton-cg', penalty='l2', C=0.01)
```

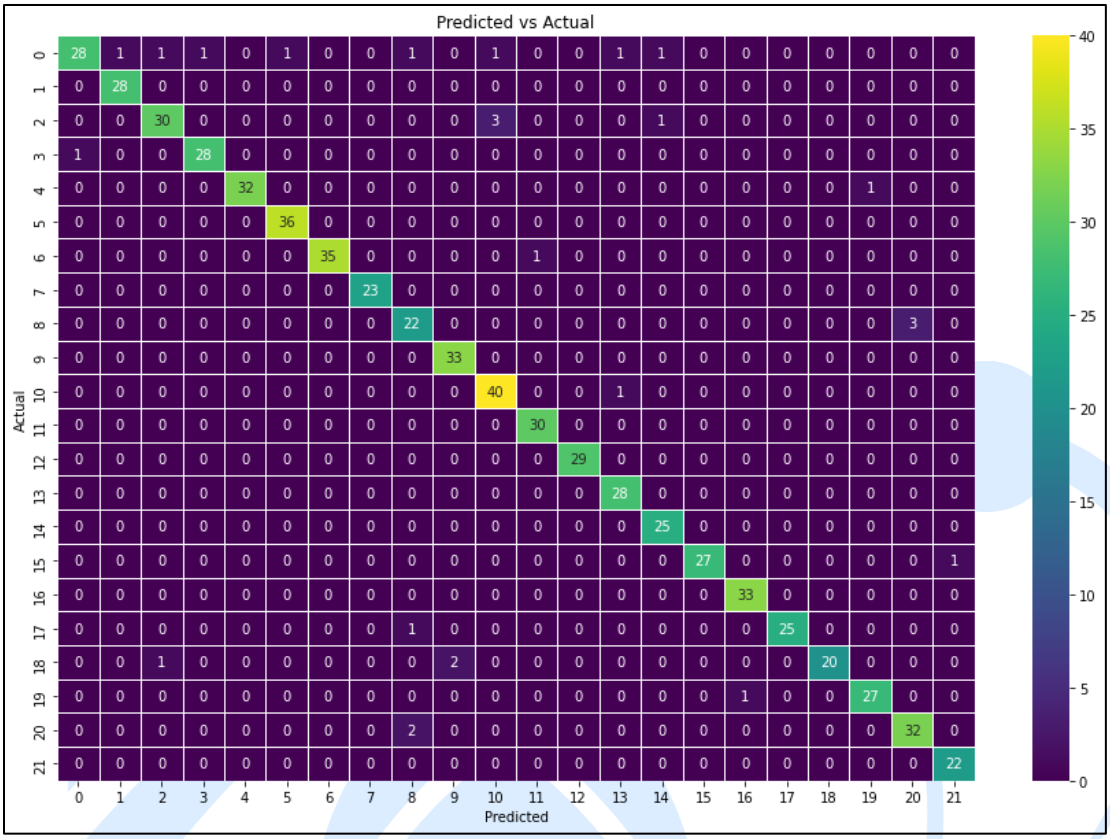
Next, we train our model using training set (x_train and y_train). Then, we test our model on x_test. Finally, we evaluate the model performance by getting the accuracy score:

```
logmodel1.fit(x_train,y_train)

pred_log1 = logmodel1.predict(x_test)

x = metrics.accuracy_score(y_test, pred_log1)
acc.append(x)
model.append('Logistic Regression')
print("LR Accuracy is: ", x*100)

print(classification_report(y_test,pred_log1))
```

```
LR Accuracy is:  95.9090909090909
              precision    recall  f1-score   support

       apple       0.97      0.78      0.86        36
      banana       0.97      1.00      0.98        28
   blackgram       0.94      0.88      0.91        34
    chickpea       0.97      0.97      0.97        29
     coconut       1.00      0.97      0.98        33
      coffee       0.97      1.00      0.99        36
      cotton       1.00      0.97      0.99        36
      grapes       1.00      1.00      1.00        23
        jute       0.85      0.88      0.86        25
 kidneybeans       0.94      1.00      0.97        33
      lentil       0.91      0.98      0.94        41
       maize       0.97      1.00      0.98        30
       mango       1.00      1.00      1.00        29
   mothbeans       0.93      1.00      0.97        28
    mungbean       0.93      1.00      0.96        25
   muskmelon       1.00      0.96      0.98        28
      orange       0.97      1.00      0.99        33
      papaya       1.00      0.96      0.98        26
  pigeonpeas       1.00      0.87      0.93        23
 pomegranate       0.96      0.96      0.96        28
        rice       0.91      0.94      0.93        34
  watermelon       0.96      1.00      0.98        22

    accuracy                           0.96       660
   macro avg       0.96      0.96      0.96       660
weighted avg       0.96      0.96      0.96       660
```

Then, we will draw a heatmap to review the confusion matrix between Predicted and Actual Values.

```
y_pred = logmodel1.predict(x_test)
y_true = y_test

cm_knn = confusion_matrix(y_true,y_pred)

f, ax = plt.subplots(figsize=(15,10))
sns.heatmap(cm_knn, annot=True, linewidth=0.5, fmt=".0f",cmap='viridis', ax = ax)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Predicted vs Actual')
plt.show()
```



6.3.2 Decision Tree

For Decision Tree (DT), we will use an implementations provided by the Scikit-Learn package.

The Decision Tree model has the following syntax:

```
DecisionTreeRegressor(criterion=mse, splitter=best, max_depth=None,
                      min_samples_split=2, min_samples_leaf=1,
                      min_weight_fraction_leaf=0.0, max_features=None,
                      random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, presort=False)
```

Firstly, we will use *GridSearchCV()* to search for the best model parameters in a parameter space provided by us. The parameter *min_samples_split* determines the minimum number of samples required to split an internal node, *min_samples_leaf* determines the minimum number of samples required to be at a leaf node, and *max_features* controls the number of features to consider when looking for the best split.

```
# Hyper parameters range intialization for tuning
dtree = DecisionTreeClassifier()

splitter = ["best","random"]
max_depth = [1,3,5,7,9,11,12]
min_samples_split= [2, 5, 10, 15, 100]
min_samples_leaf=[1,2,3,4,5,6,7,8,9,10]
max_features=["auto","log2","sqrt",None]
max_leaf_nodes=[None,10,20,30,40,50,60,70,80,90]

#Creating a dictionary for the hyper parameters
hyperT = dict(splitter=splitter, max_depth = max_depth, min_samples_split = min_samples_split,
              min_samples_leaf = min_samples_leaf, max_features=max_features, max_leaf_nodes=max_leaf_nodes)
```

We defined the parameter space above using reasonable values for chosen parameters. Then we used GridSearchCV() with 3 folds (cv=3).

```
#Applying GridSearchCV to get the best value for hyperparameters
gridT = GridSearchCV(dtree, hyperT, cv = 3, verbose = 1, n_jobs = -1)
gridT.fit(x_train, y_train)
```

```
#Printing the best hyperparameters
print('The best hyper parameters are: \n',gridT.best_params_)
```

```
The best hyper parameters are:
 {'max_depth': 12, 'max_features': None, 'max_leaf_nodes': 40, 'min_samples_leaf': 1, 'min_samples
_split': 2, 'splitter': 'best'}
```

Now we build our Decision Tree model with the best parameters found:

```
dtree1 = DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=12, min_samples_split=2,
                                min_samples_leaf=1, max_features=None, max_leaf_nodes=40)
```

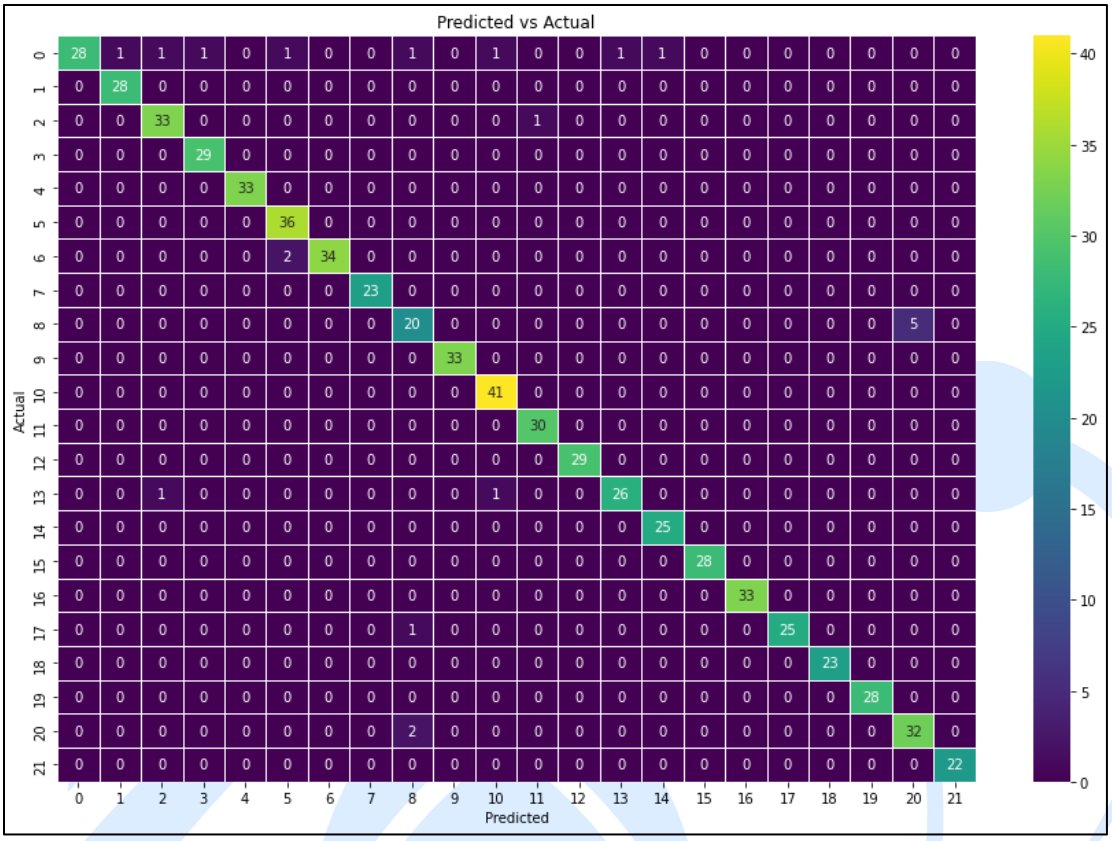
Next, we train our model using training set (x_train and y_train). Then, we test our model on x_test. Finally, we evaluate the model performance by getting the accuracy score:

```
dtree1.fit(x_train,y_train)

y_predictions1 = dtree1.predict(x_test)

x = metrics.accuracy_score(y_test, y_predictions1)
acc.append(x)
model.append('Decision Tree')
print("Decision Tree's Accuracy is: ", x*100)

print(classification_report(y_test,y_predictions1))
```

```
Decision Tree's Accuracy is:  96.81818181818181
              precision    recall  f1-score   support

       apple       1.00      0.78      0.88        36
      banana       0.97      1.00      0.98        28
   blackgram       0.94      0.97      0.96        34
    chickpea       0.97      1.00      0.98        29
     coconut       1.00      1.00      1.00        33
      coffee       0.92      1.00      0.96        36
      cotton       1.00      0.94      0.97        36
      grapes       1.00      1.00      1.00        23
        jute       0.83      0.80      0.82        25
 kidneybeans       1.00      1.00      1.00        33
      lentil       0.95      1.00      0.98        41
       maize       0.97      1.00      0.98        30
       mango       1.00      1.00      1.00        29
   mothbeans       0.96      0.93      0.95        28
    mungbean       0.96      1.00      0.98        25
   muskmelon       1.00      1.00      1.00        28
      orange       1.00      1.00      1.00        33
      papaya       1.00      0.96      0.98        26
  pigeonpeas       1.00      1.00      1.00        23
 pomegranate       1.00      1.00      1.00        28
        rice       0.86      0.94      0.90        34
  watermelon       1.00      1.00      1.00        22

    accuracy                           0.97       660
   macro avg       0.97      0.97      0.97       660
weighted avg       0.97      0.97      0.97       660
```

Then, we will draw a heatmap to review the confusion matrix between Predicted and Actual Values.

```
y_pred = dtree1.predict(x_test)
y_true = y_test


cm_dt = confusion_matrix(y_true,y_pred)

f, ax = plt.subplots(figsize=(15,10))
sns.heatmap(cm_dt, annot=True, linewidth=0.5, fmt=".0f",  cmap='viridis', ax = ax)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title('Predicted vs Actual')
plt.show()
```



6.3.3 Random Forest

For Random Forest (RF), we will use an implementations provided by the Scikit-Learn package.

The Random Forest model has the following syntax:

```
RandomForestRegressor(n_estimators=warn, criterion=mse, max_depth=None,
                      min_samples_split=2, min_samples_leaf=1,
                      min_weight_fraction_leaf=0.0, max_features=auto,
                      max_leaf_nodes=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, bootstrap=True, oob_score=False,
                      n_jobs=None, random_state=None, verbose=0, warm_start=False)
```

Firstly, we will use *GridSearchCV()* to search for the best model parameters in a parameter space provided by us. The parameter *n_estimators* specifies the number of trees in the forest, bootstrap determines whether bootstrap samples are used when building trees. *max_depth*, *min_samples_split*, *min_samples_leaf* are the same as those of the decision tree model.

```
rfc = RandomForestClassifier()

#Setting values for the parameters
n_estimators = [100, 300, 500, 800, 1200]
max_depth = [5, 10, 15, 25, 30]
min_samples_split = [2, 5, 10, 15, 100]
min_samples_leaf = [1, 2, 5, 10]

#Creating a dictionary for the hyper parameters
hyper_rf = dict(n_estimators = n_estimators, max_depth = max_depth,
              min_samples_split = min_samples_split, min_samples_leaf = min_samples_leaf)
```

We defined the parameter space above using reasonable values for chosen parameters. Then we used GridSearchCV() with 3 folds (cv=3).

```
#Applying GridSearchCV to get the best value for hyperparameters
gridrf = GridSearchCV(rfc, hyper_rf, cv = 3, verbose = 1, n_jobs = -1)
gridrf.fit(x_train, y_train)
```

Now we build our Random Forest model with the best parameters found:

```
#Printing the best hyperparameters
print('The best hyper parameters are:\n',gridrf.best_params_)
```

```
The best hyper parameters are:
 {'max_depth': 25, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 500}
```

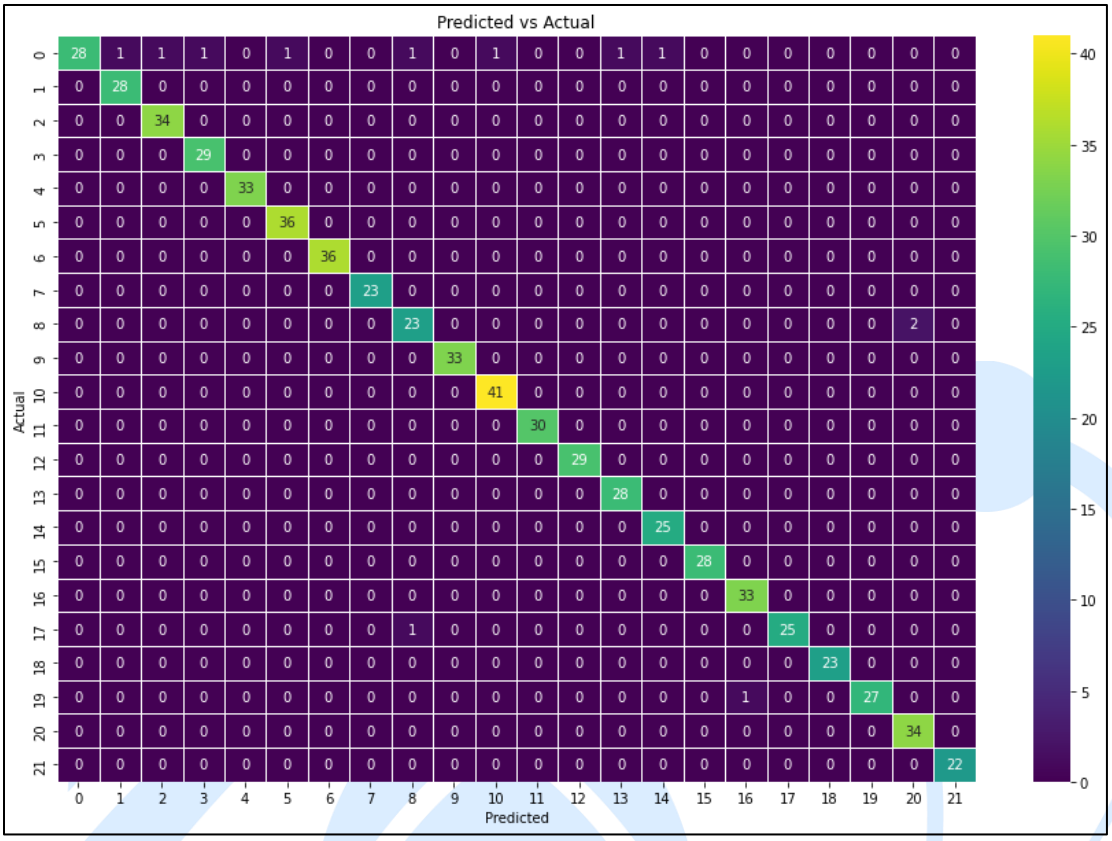
Next, we train our model using training set (x_train and y_train). Then, we test our model on x_test. Finally, we evaluate the model performance by getting the accuracy score:

```
rfc1 = RandomForestClassifier(n_estimators=500, max_depth=25, min_samples_split=2, min_samples_leaf=1)
rfc1.fit(x_train,y_train)

predict_r1 = rfc1.predict(x_test)

x = metrics.accuracy_score(y_test, predict_r1)
acc.append(x)
model.append('Random Forest')
print("Random Forest Accuracy is: ", x*100)

print(classification_report(y_test,predict_r1))
```

```
Random Forest Accuracy is:  98.18181818181819
              precision    recall  f1-score   support

       apple       1.00      0.78      0.88        36
      banana       0.97      1.00      0.98        28
   blackgram       0.97      1.00      0.99        34
    chickpea       0.97      1.00      0.98        29
     coconut       1.00      1.00      1.00        33
      coffee       0.97      1.00      0.99        36
      cotton       1.00      1.00      1.00        36
      grapes       1.00      1.00      1.00        23
        jute       0.92      0.92      0.92        25
 kidneybeans       1.00      1.00      1.00        33
      lentil       0.98      1.00      0.99        41
       maize       1.00      1.00      1.00        30
       mango       1.00      1.00      1.00        29
   mothbeans       0.97      1.00      0.98        28
    mungbean       0.96      1.00      0.98        25
   muskmelon       1.00      1.00      1.00        28
      orange       0.97      1.00      0.99        33
      papaya       1.00      0.96      0.98        26
  pigeonpeas       1.00      1.00      1.00        23
 pomegranate       1.00      0.96      0.98        28
        rice       0.94      1.00      0.97        34
  watermelon       1.00      1.00      1.00        22

    accuracy                           0.98       660
   macro avg       0.98      0.98      0.98       660
weighted avg       0.98      0.98      0.98       660
```

Then, we will draw a heatmap to review the confusion matrix between Predicted and Actual Values.

```
y_pred = rfc1.predict(x_test)
y_true = y_test

cm_rf = confusion_matrix(y_true,y_pred)

f, ax = plt.subplots(figsize=(15,10))
sns.heatmap(cm_rf, annot=True, linewidth=0.5, fmt=".0f",  cmap='viridis', ax = ax)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title('Predicted vs Actual')
plt.show()
```



6.3.4 XGBoost

For Gradient Boosting (GB), we will use the renowned XGBoost implementations. XGBoost model has the following syntax:

```
XGBRegressor(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True,
             objective='reg:linear', booster='gbtree', n_jobs=1, nthread=None,
             gamma=0, min_child_weight=1, max_delta_step=0, subsample=1,
             colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_lambda=1,
             scale_pos_weight=1, base_score=0.5, random_state=0, seed=None,
             missing=None, importance_type='gain', **kwargs)
```

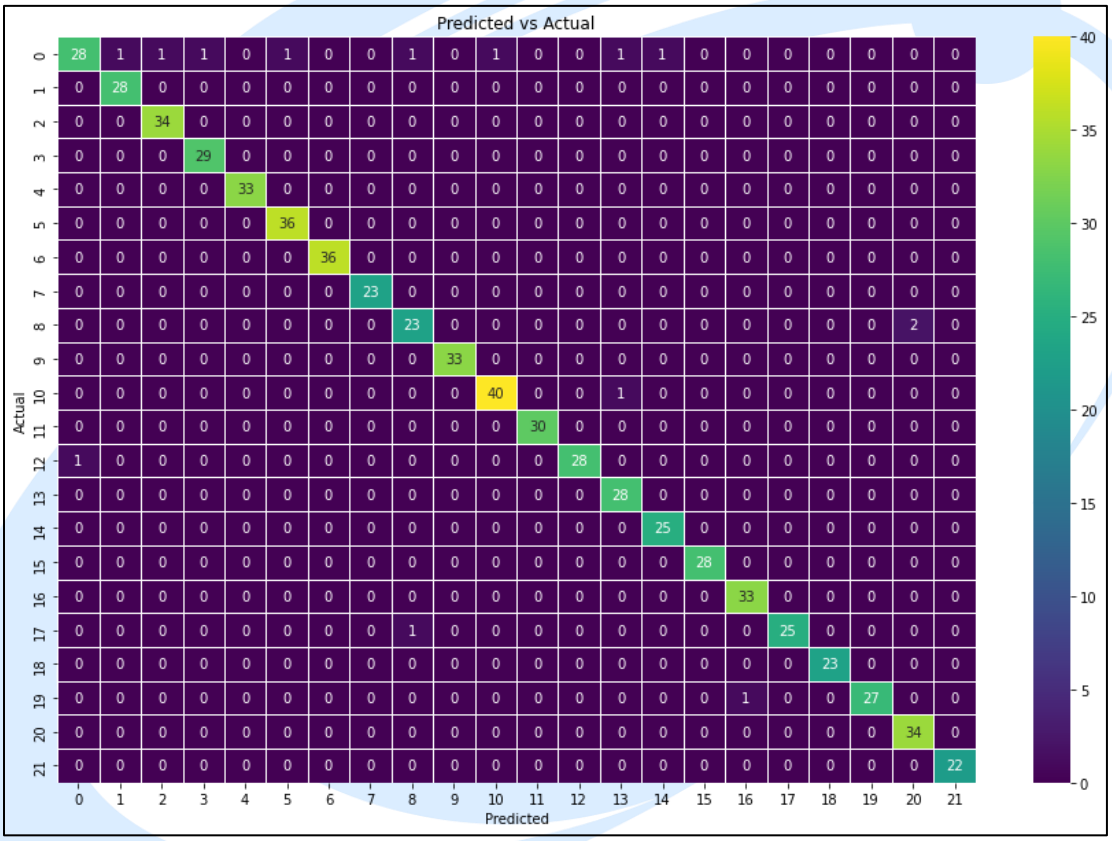
Firstly, we train our model using training set (x_train and y_train). Then, we test our model on x_test. Finally, we evaluate the model performance by getting the accuracy score:

```
XB = xgb.XGBClassifier()
XB.fit(x_train,y_train)

predicted_values = XB.predict(x_test)

x = metrics.accuracy_score(y_test, predicted_values)
acc.append(x)
model.append('XGBoost')
print("XGBoost Accuracy is: ", x)

print(classification_report(y_test, predicted_values))
```

```
XGBoost Accuracy is:  0.9787878787878788
              precision    recall  f1-score   support

       apple       0.97      0.78      0.86        36
      banana       0.97      1.00      0.98        28
   blackgram       0.97      1.00      0.99        34
    chickpea       0.97      1.00      0.98        29
     coconut       1.00      1.00      1.00        33
      coffee       0.97      1.00      0.99        36
      cotton       1.00      1.00      1.00        36
      grapes       1.00      1.00      1.00        23
        jute       0.92      0.92      0.92        25
 kidneybeans       1.00      1.00      1.00        33
      lentil       0.98      0.98      0.98        41
       maize       1.00      1.00      1.00        30
       mango       1.00      0.97      0.98        29
   mothbeans       0.93      1.00      0.97        28
    mungbean       0.96      1.00      0.98        25
   muskmelon       1.00      1.00      1.00        28
      orange       0.97      1.00      0.99        33
      papaya       1.00      0.96      0.98        26
  pigeonpeas       1.00      1.00      1.00        23
 pomegranate       1.00      0.96      0.98        28
        rice       0.94      1.00      0.97        34
  watermelon       1.00      1.00      1.00        22

    accuracy                           0.98       660
   macro avg       0.98      0.98      0.98       660
weighted avg       0.98      0.98      0.98       660
```

Then, we will draw a heatmap to review the confusion matrix between Predicted and Actual Values.

```
y_pred = XB.predict(x_test)
y_true = y_test

cm_nb = confusion_matrix(y_true,y_pred)

f, ax = plt.subplots(figsize=(15,10))
sns.heatmap(cm_nb, annot=True, linewidth=0.5, fmt=".0f",  cmap='viridis', ax = ax)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title('Predicted vs Actual')
plt.show()
```



6.3.5 K-Nearest Neighbours

For Nearest Neighbors, we will use an implementation of the k-nearest neighbors (KNN) algorithm provided by Scikit-Learn package.

The KNN model has the following syntax:

```
KNeighborsRegressor(n_neighbors=5, weights=uniform, algorithm=auto,
                    leaf_size=30, p=2, metric=minkowski, metric_params=None,
                    n_jobs=None, **kwargs)
```

Firstly, we will use GridSearchCV() to search for the best model parameters in a parameter space provided by us. The parameter *n_neighbors* represents k which is the number of neighbors to use, *weights* determines the weight function used in prediction: uniform or distance, *metric* specifies the algorithm used to compute the nearest neighbors. It can affect the speed of the construction and query, as well as the memory required to store the tree.

```
grid_params = { 'n_neighbors' : [12,13,14,15,16,17,18],
                'weights' : ['uniform','distance'],
                'metric' : ['minkowski','euclidean','manhattan']}
```

We defined the parameter space above using reasonable values for chosen parameters. Then we used GridSearchCV() with 3 folds (cv=3). Now we build our KNN model with the best parameters found:

```
gs = GridSearchCV(KNeighborsClassifier(), grid_params, verbose = 1, cv=3, n_jobs = -1)
```

```
gs.fit(x_train, y_train)
```

```
gs.best_score_
```

```
0.9584398378855338
```

```
gs.best_params_
```

```
{'metric': 'manhattan', 'n_neighbors': 12, 'weights': 'distance'}
```

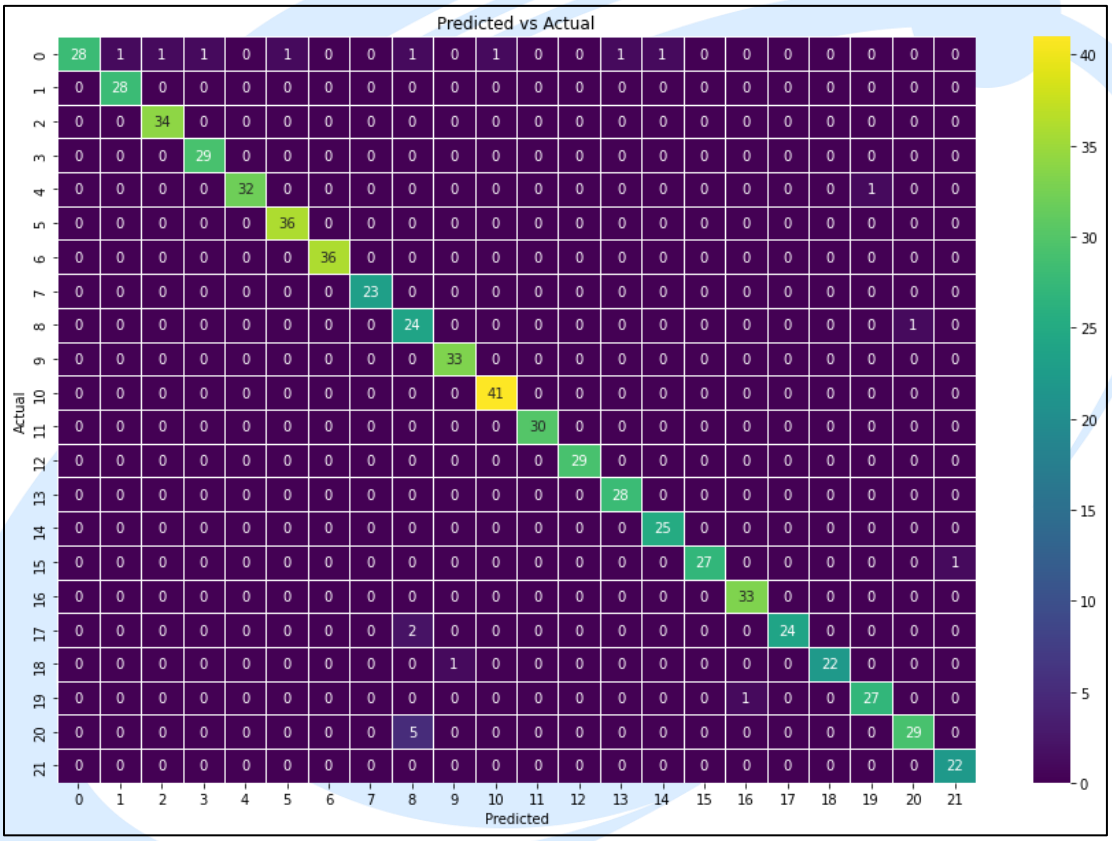
Next, we train our model using training set (x_train and y_train). Then, we test our model on x_test. Finally, we evaluate the model performance by getting the accuracy score:

```python
# Using the best hyperparameters
knn_1 = KNeighborsClassifier(n_neighbors = 12, weights = 'distance',algorithm = 'brute',metric = 'manhattan')
knn_1.fit(x_train, y_train)

predicted_values = knn_1.predict(x_test)

x = metrics.accuracy_score(y_test, predicted_values)
acc.append(x)
model.append('K Nearest Neighbours')
print("KNN Accuracy is: ", x)

print(classification_report(y_test,predicted_values))
```

```
KNN Accuracy is:  0.9696969696969697
              precision    recall  f1-score   support

       apple       1.00      0.78      0.88        36
      banana       0.97      1.00      0.98        28
   blackgram       0.97      1.00      0.99        34
    chickpea       0.97      1.00      0.98        29
     coconut       1.00      0.97      0.98        33
      coffee       0.97      1.00      0.99        36
      cotton       1.00      1.00      1.00        36
      grapes       1.00      1.00      1.00        23
        jute       0.75      0.96      0.84        25
 kidneybeans       0.97      1.00      0.99        33
      lentil       0.98      1.00      0.99        41
       maize       1.00      1.00      1.00        30
       mango       1.00      1.00      1.00        29
   mothbeans       0.97      1.00      0.98        28
    mungbean       0.96      1.00      0.98        25
   muskmelon       1.00      0.96      0.98        28
      orange       0.97      1.00      0.99        33
      papaya       1.00      0.92      0.96        26
  pigeonpeas       1.00      0.96      0.98        23
 pomegranate       0.96      0.96      0.96        28
        rice       0.97      0.85      0.91        34
  watermelon       0.96      1.00      0.98        22

    accuracy                           0.97       660
   macro avg       0.97      0.97      0.97       660
weighted avg       0.97      0.97      0.97       660
```

Then, we will draw a heatmap to review the confusion matrix between Predicted and Actual Values.

```
y_pred = knn_1.predict(x_test)
y_true = y_test

cm_nb = confusion_matrix(y_true,y_pred)

f, ax = plt.subplots(figsize=(15,10))
sns.heatmap(cm_nb, annot=True, linewidth=0.5, fmt=".0f",  cmap='viridis', ax = ax)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title('Predicted vs Actual')
plt.show()
```



6.3.6 Bagging Classifier

For Bagging Classifier, we will use an implementations provided by the Scikit-Learn package.

Firstly, we will use *GridSearchCV()* to search for the best model parameters in a parameter space provided by us.

```
#Setting values for the parameters
n_estimators = [100, 300, 500, 800, 1200]
#max_depth = [5, 10, 15, 25, 30]
max_samples = [5, 10, 25, 50, 100]
max_features = [1, 2, 5, 10, 13]

#Creating a dictionary for the hyper parameters
hyperbag = dict(n_estimators = n_estimators, max_samples = max_samples,
              max_features = max_features)
```

We defined the parameter space above using reasonable values for chosen parameters. Then we used GridSearchCV() with 3 folds (cv=3).

```
#Applying GridSearchCV to get the best value for hyperparameters
gridbag = GridSearchCV(bagg, hyperbag, cv = 3, verbose = 1, n_jobs = -1)
gridbag.fit(x_train, y_train)
```

```
#Printing the best hyperparameters
print('The best hyper parameters are: \n',gridT.best_params_)
```

```
The best hyper parameters are:
 {'max_depth': 12, 'max_features': None, 'max_leaf_nodes': 40, 'min_samples_leaf': 1, 'min_samples
_split': 2, 'splitter': 'best'}
```

Now we build our Bagging model with the best parameters found:

```
# Using the best hyperparameters
bagg1 = BaggingClassifier(max_features=5, max_samples=100,n_estimators= 800)
```

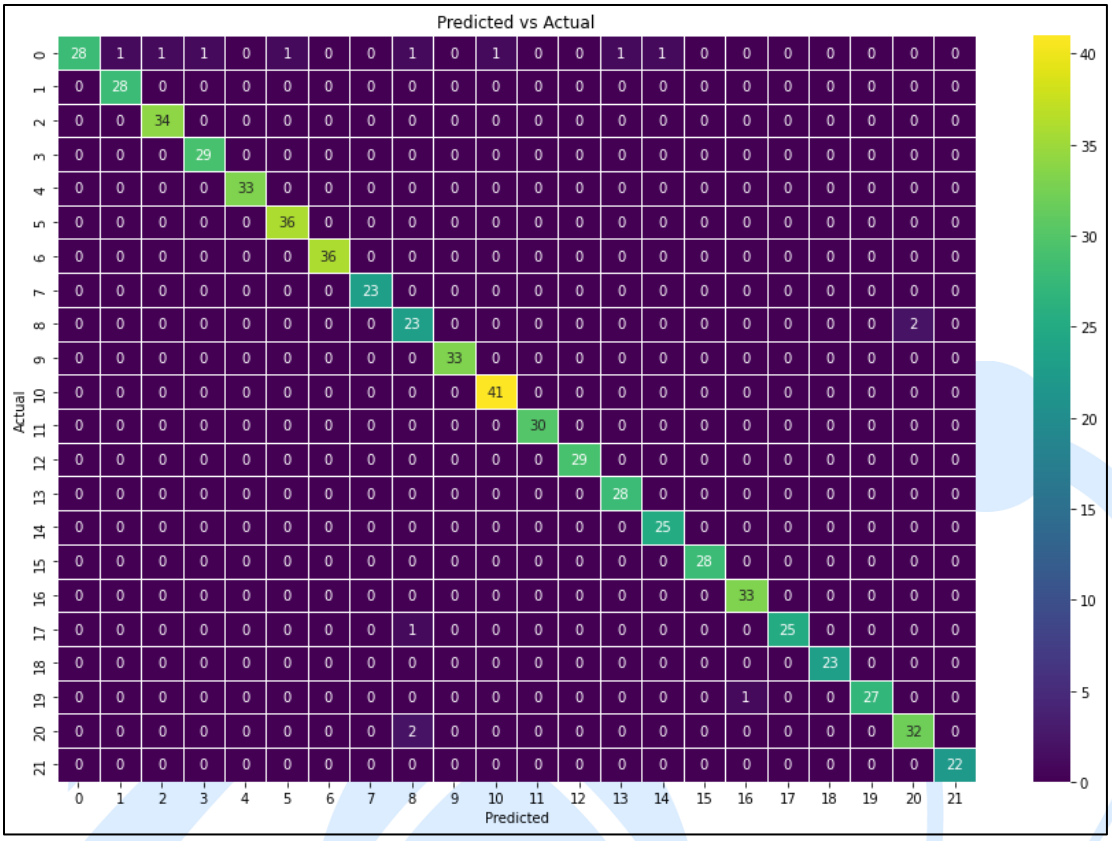
Firstly, we train our model using training set (x_train and y_train). Then, we test our model on x_test. Finally, we evaluate the model performance by getting the accuracy score:

```
bagg1.fit(x_train,y_train)

pred_bagg1 = bagg1.predict(x_test)

x = metrics.accuracy_score(y_test, pred_bagg1)
acc.append(x)
model.append('Bagging Classifier')
print("Bagging Accuracy is: ", x)

print(classification_report(y_test,pred_bagg1))
```

```
Bagging Accuracy is:  0.9787878787878788
              precision    recall  f1-score   support

       apple       1.00      0.78      0.88        36
      banana       0.97      1.00      0.98        28
   blackgram       0.97      1.00      0.99        34
    chickpea       0.97      1.00      0.98        29
     coconut       1.00      1.00      1.00        33
      coffee       0.97      1.00      0.99        36
      cotton       1.00      1.00      1.00        36
      grapes       1.00      1.00      1.00        23
        jute       0.85      0.92      0.88        25
 kidneybeans       1.00      1.00      1.00        33
      lentil       0.98      1.00      0.99        41
       maize       1.00      1.00      1.00        30
       mango       1.00      1.00      1.00        29
   mothbeans       0.97      1.00      0.98        28
    mungbean       0.96      1.00      0.98        25
   muskmelon       1.00      1.00      1.00        28
      orange       0.97      1.00      0.99        33
      papaya       1.00      0.96      0.98        26
  pigeonpeas       1.00      1.00      1.00        23
 pomegranate       1.00      0.96      0.98        28
        rice       0.94      0.94      0.94        34
  watermelon       1.00      1.00      1.00        22

    accuracy                           0.98       660
   macro avg       0.98      0.98      0.98       660
weighted avg       0.98      0.98      0.98       660
```

Then, we will draw a heatmap to review the confusion matrix between Predicted and Actual Values.

```
y_pred = bagg1.predict(x_test)
y_true = y_test

cm_rf = confusion_matrix(y_true,y_pred)

f, ax = plt.subplots(figsize=(15,10))
sns.heatmap(cm_rf, annot=True, linewidth=0.5, fmt=".0f",  cmap='viridis', ax = ax)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title('Predicted vs Actual')
plt.show()
```



7 Analysis and Comparision

In the previous section, we created many models: for each model, we trained (fitted) the model to our training data (x_train and y_train), then tested the model on our test data (x_test), and finally, we evaluated the model performance by comparing the model predictions with the true values in y_test. We used accuracy score to evaluate model performance.

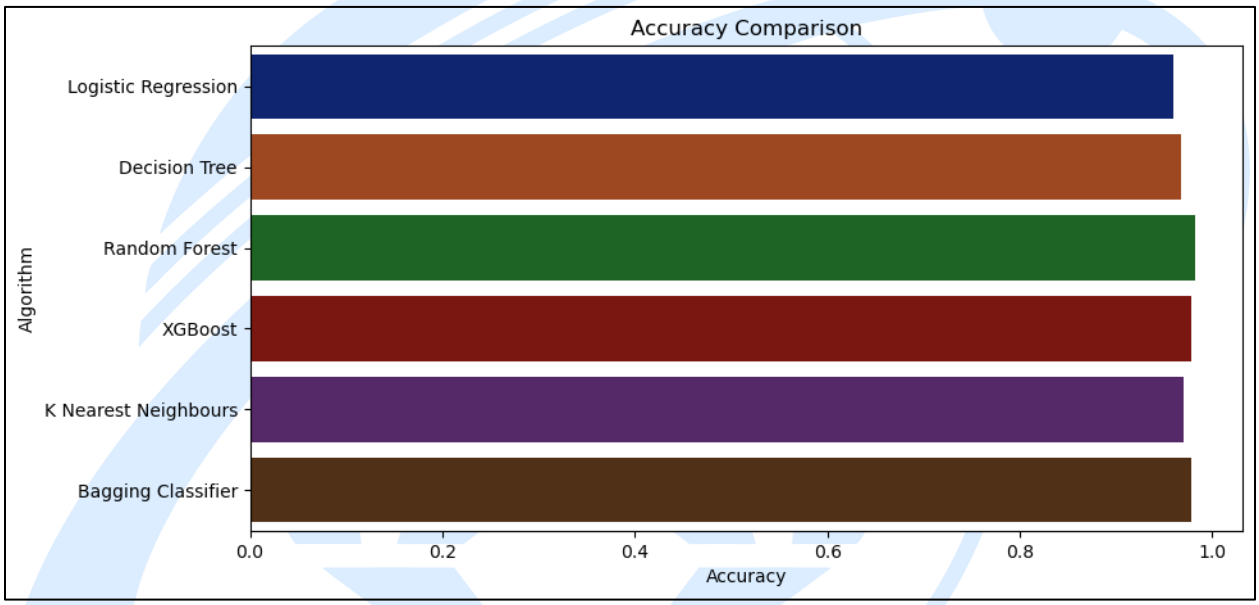
Using the results we got in the previous section, we present a table that shows the Accuracy Score for each model when applied to the test set x_test.

Model	**Accuracy Score**
Logistic Regression	95.91%
Decision Tree	96.81%
Random Forest	98.18%
XGBoost	97.87%

K-Nearest Neighbours	96.97%
Bagging	97.88%

We also present a graph that visualizes the table contents:

```
plt.figure(figsize=[10,5],dpi = 100)
plt.title('Accuracy Comparison')
plt.xlabel('Accuracy')
plt.ylabel('Algorithm')
sns.barplot(x = acc,y = model,palette='dark')
```



By looking at the table and the graph, we can see that Logistic Regression model has the smallest accuracy score, 95.91% followed by Decision Tree model with a little score of 96.81%. After that, K-Nearest Neighbors has accuracy score at 96.97%. Then come XGBoost and Bagging models with close errors: 97.87% and 97.88% respectively. And at last, Random Forest model with an accuracy score at 98.18%.

So, in our experiment, the best model is Random Forest and the worst model is Logistic Regression.

8 Conclusion and Future Work

In this paper, we built serveral regression models to predict the best-suited crop for the particular land that can help farmers to grow crops more efficiently. We eveluated and compared each model to determine the one with highest performance. We also looked at

how some models rank the features according to their importance. In this paper, we followed the data science process starting with getting the data, then cleaning and preprocessing the data, followed by exploring the data and building models, then evaluating the results and communicating them with visualizations.

In the future, new data from the fields can be collected to get a clear image of the soil and incorporate other machine learning algorithms and deep learning algorithms such as ANN or CNN to classify more varieties of crops.

9 References

[1] Dahikar S and Rode S V 2014 Agricultural crop yield prediction using artificial neural network approach International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering vol 2 Issue 1 pp 683-6.

[2] Suresh A, Ganesh P and Ramalatha M 2018 Prediction of major crop yields of Tamilnadu using K-means and Modified KNN 2018 3rd International Conference on Communication and Electronics Systems (ICCES) pp 88-93 doi:10.1109/CESYS.2018.8723956.

[3] Medar R, Rajpurohit V S and Shweta S 2019 Crop yield prediction using machine learning techniques IEEE 5th International Conference for Convergence in Technology (I2CT) pp 1-5 doi: 10.1109/I2CT45611.2019.9033611.

[4] Nishant P S, Venkat P S, Avinash B L and Jabber B 2020 Crop yield prediction based on Indian agriculture using machine learning 2020 International Conference for Emerging Technology (INCET) pp 1-4 doi: 10.1109/INCET49848.2020.9154036.

[5] Kalimuthu M, Vaishnavi P and Kishore M 2020 Crop prediction using machine learning 2020 Third International Conference on Smart Systems and Inventive Technology (ICSSIT) pp 926-32 doi: 10.1109/ICSSIT48917.2020.9214190.

[6] Geetha V, Punitha A, Abarna M, Akshaya M, Illakiya S and Janani A P 2020 An effective crop prediction using random forest algorithm 2020 International Conference on System, Computation, Automation and Networking (ICSCAN) pp 1-5 doi: 10.1109/ICSCAN49426.2020.9262311.

[7] Pande S M, Ramesh P K, Anmol A, Aishwaraya B R, Rohilla K and Shaurya K 2021 Crop recommender system using machine learning approach 2021 5th International Conference on Computing Methodologies and Communication (ICCMC) pp 1066-71 doi: 10.1109/ICCMC51019.2021.9418351.

[8] Sellam V, and Poovammal E 2016 Prediction of crop yield using regression analysis Indian Journal of Science and Technology vol 9(38) pp 1-5.

[9] Bharath S, Yeshwanth S, Yashas B L and Vidyaranya R Javalagi 2020 Comparative Analysis of Machine Learning Algorithms in The Study of Crop and Crop yield Prediction International Journal of Engineering Research & Technology (IJERT) NCETESFT – 2020 vol 8 Issue 14.

[10] Mahendra N, Vishwakarma D, Nischitha K, Ashwini and Manjuraju M. R 2020 Crop prediction using machine learning approaches, International Journal of Engineering Research & Technology (IJERT) vol 9 Issue 8 (August 2020).

[11] Gulati P and Jha S K 2020 Efficient crop yield prediction in India using machine learning techniques International Journal of Engineering Research & Technology (IJERT) ENCADEMS – 2020 vol 8 Issue 10.

[12] Gupta A, Nagda D, Nikhare P, Sandbhor A, 2021, Smart crop prediction using IoT and machine learning International Journal of Engineering Research & Technology (IJERT) NTASU – 2020 vol 9 Issue 3.