Sri Sivasubramaniya Nadar College of Engineering, Kalavakkam – 603 110
(An Autonomous Institution, Affiliated to Anna University, Chennai)
Department of Computer Science and Engineering

Assignment – II

Regulations – R2021

Degree B.E. / B. Tech.	B.E.	Branch	CSE
Semester	II	Academic Year	2023-2024
Subject Code & Name	UCS2202 – Foundations of Data Science		
Batch: 2023-2027			Maximum: 30 Marks

Name: Nithyasree K **Name:** SK Shivaanee

Register number: 3122235001092 **Register Number:** 3122235001123

1. **Identify your dataset to perform any one of the data modeling such as:**
   - **Regression**
   - **Classification**
   - **Clustering**

Ans: The dataset we have used for this model building is Crop recommendation dataset. The dataset under consideration embodies a wealth of information encompassing key factors such as Nitrogen, Phosphorus, and Potassium levels, alongside environmental variables like Temperature, Humidity, pH_Value, and Rainfall. Understanding and analyzing this dataset is fundamental to making informed decisions that may enhance agricultural productivity, resource management, and overall crop health.

```
[6] path="/content/drive/MyDrive/Crop_Recommendation.csv"
    df=pd.read_csv(path, delimiter=';')
```

```
[ ] df.head()
```

	Nitrogen	Phosphorus	Potassium	Temperature	Humidity	pH_Value	Rainfall	Crop
0	90.0	42.0	43.0	20,87974371	NaN	6,502985292	202,9355362	Rice
1	85.0	58.0	41.0	21,77046169	80,31964408	7,038096361	226,6555374	Rice
2	60.0	55.0	44.0	23,00445915	82,3207629	7,840207144	263,9642476	Rice
3	NaN	35.0	40.0	26,49109635	80,15836264	6,980400905	242,8640342	Rice
4	78.0	42.0	42.0	20,13017482	81,60487287	7,628472891	262,7173405	Rice

The necessary modules to download:

```python
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O
import seaborn as sns
import matplotlib.pyplot as plt
```

```python
import matplotlib.pyplot as plt # data visualization
import seaborn as sns # statistical data visualization
from scipy import stats # statistical functions
from sklearn import metrics
from sklearn.linear_model import LinearRegression, LogisticRegression  # Import LinearRegression and LogisticRegression
from sklearn.ensemble import RandomForestRegressor  # Import RandomForestRegressor for random forest
from sklearn.svm import SVR  # Import SVR for Support Vector Regression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor  # Import DecisionTree for decision tree
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier # implementing the K-Nearest Neighbors algorithm for continous value predict
from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB # Import naive_bayes
from sklearn.model_selection import train_test_split # splitting data into training and testing sets
from sklearn.metrics import accuracy_score # evaluating the accuracy of the classifier
from sklearn.ensemble import ExtraTreesClassifier # Feature selection
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.model_selection import GridSearchCV
import seaborn as sb
import warnings
warnings.filterwarnings('ignore')  # Ignore warning messages
```

2. Develop Python code to perform the following:

- **To read the dataset**

```python
df.head()
```

	Nitrogen	Phosphorus	Potassium	Temperature	Humidity	pH_Value	Rainfall	Crop
0	90.0	42.0	43.0	20,87974371	NaN	6,502985292	202,9355362	Rice
1	85.0	58.0	41.0	21,77046169	80,31964408	7,038096361	226,6555374	Rice
2	60.0	55.0	44.0	23,00445915	82,3207629	7,840207144	263,9642476	Rice
3	NaN	35.0	40.0	26,49109635	80,15836264	6,980400905	242,8640342	Rice
4	78.0	42.0	42.0	20,13017482	81,60487287	7,628472891	262,7173405	Rice

- **To check for null values**

```python
print(f'Number of null values: {df.isnull().any()}')
print(f'Number of duplicated values: {df.duplicated().any()}')
```

```
Number of null values: Nitrogen      True
Phosphorus     True
Potassium      True
Temperature    True
Humidity       True
pH_Value       True
Rainfall       True
Crop           False
dtype: bool
Number of duplicated values: False
```

```
df.nunique()
```

```
Nitrogen        137
Phosphorus      117
Potassium        73
Temperature    2192
Humidity       2195
pH_Value       2193
Rainfall       2193
Crop             22
dtype: int64
```

```
describe= df.describe().T
describe
```

	count	mean	std	min	25%	50%	75%	max
Nitrogen	2195.0	50.483371	36.905544	0.0	21.0	37.0	84.0	140.0
Phosphorus	2194.0	53.370556	32.948534	5.0	28.0	51.0	68.0	145.0
Potassium	2195.0	48.179043	50.700879	5.0	20.0	32.0	49.0	205.0

- **To handle missing values**

```
# Display data types of columns
print(df.dtypes)
```

```
Nitrogen       float64
Phosphorus     float64
Potassium      float64
Temperature     object
Humidity        object
pH_Value        object
Rainfall        object
Crop            object
dtype: object
```

```python
# Select numeric columns
numeric_cols = df.select_dtypes(include=['number']).columns

# Select non-numeric columns
non_numeric_cols = df.select_dtypes(exclude=['number']).columns
```

```python
[8] # Fill missing values with the mean for numeric columns
df[numeric_cols] = df[numeric_cols].fillna(df[numeric_cols].mean())
```

```python
# Fill missing values with the mode for non-numeric columns
for col in non_numeric_cols:
    df[col].fillna(df[col].mode()[0], inplace=True)
```

```python
# Verify that there are no more missing values
print(df.isnull().sum())

# Display the first few rows of the cleaned DataFrame
print(df.head())
```

OUTPUT:

```
Nitrogen       0
Phosphorus     0
Potassium      0
Temperature    0
Humidity       0
pH_Value       0
Rainfall       0
Crop           0
dtype: int64
    Nitrogen  Phosphorus  Potassium  Temperature     Humidity     pH_Value  \
0  90.000000        42.0       43.0  20,87974371  14,25803981  6,502985292
1  85.000000        58.0       41.0  21,77046169  80,31964408  7,038096361
2  60.000000        55.0       44.0  23,00445915   82,3207629  7,840207144
3  50.483371        35.0       40.0  26,49109635  80,15836264  6,980400905
4  78.000000        42.0       42.0  20,13017482  81,60487287  7,628472891

      Rainfall  Crop
0  202,9355362  Rice
1  226,6555374  Rice
2  263,9642476  Rice
3  242,8640342  Rice
4  262,7173405  Rice
```

3. Examine a few approaches suitable for the identified dataset based on the exploratory data analysis. (4 marks)

These are the exploratory analyses we used to identify the necessary models:

```
# Example check to identify columns with commas
for col in df.columns:
    if df[col].dtype == 'object' and df[col].str.contains(',').any():
        print(f"Column '{col}' contains commas.")
```

```
Column 'Temperature' contains commas.
Column 'Humidity' contains commas.
Column 'pH_Value' contains commas.
Column 'Rainfall' contains commas.
```

```
# Replace commas with periods and convert to numeric
for col in df.columns:
    if df[col].dtype == 'object' and df[col].str.contains(',').any():
        df[col] = df[col].str.replace(',', '.').astype(float)

# Verify changes
print(df.head())
```

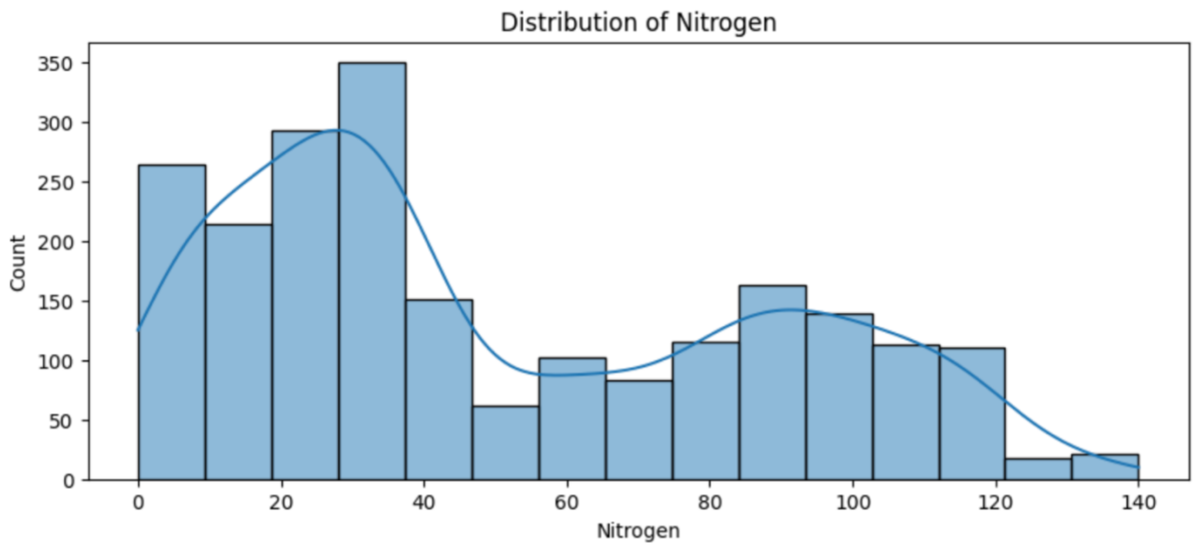
```
    Nitrogen  Phosphorus  Potassium  Temperature   Humidity  pH_Value  \
0  90.000000        42.0       43.0    20.879744  14.258040  6.502985
1  85.000000        58.0       41.0    21.770462  80.319644  7.038096
2  60.000000        55.0       44.0    23.004459  82.320763  7.840207
3  50.483371        35.0       40.0    26.491096  80.158363  6.980401
4  78.000000        42.0       42.0    20.130175  81.604873  7.628473

     Rainfall  Crop
0  202.935536  Rice
1  226.655537  Rice
2  263.964248  Rice
3  242.864034  Rice
4  262.717340  Rice
```

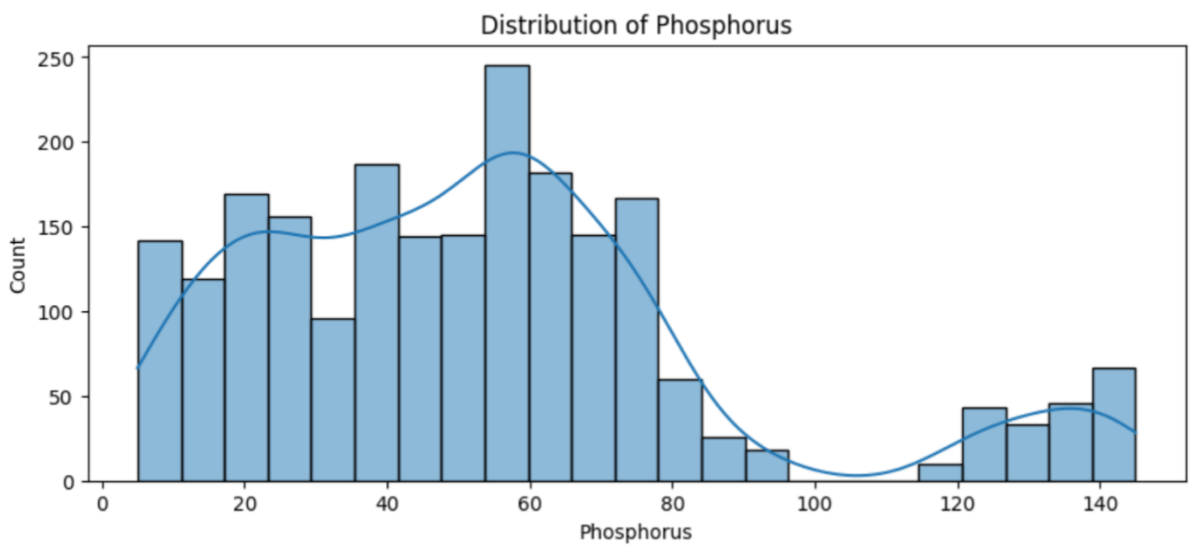
```
# Select only numeric columns for correlation
numeric_df = df.select_dtypes(include=['number'])

# Verify selected columns
print("Numeric Data Types:\n", numeric_df.dtypes)
```

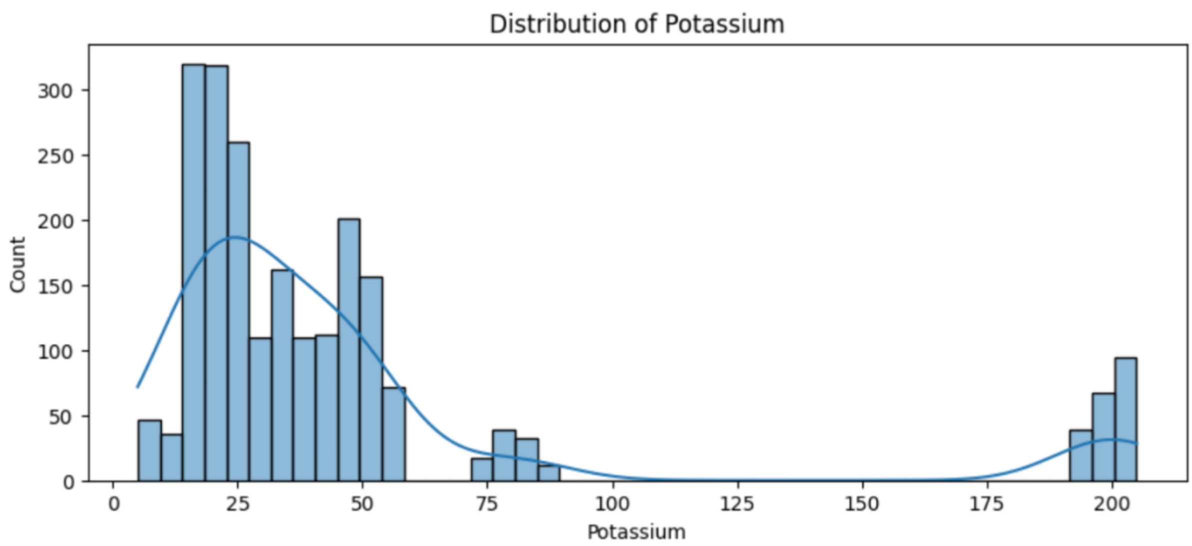
```
Numeric Data Types:
 Nitrogen        float64
Phosphorus      float64
Potassium       float64
Temperature     float64
Humidity        float64
pH_Value        float64
Rainfall        float64
dtype: object
```

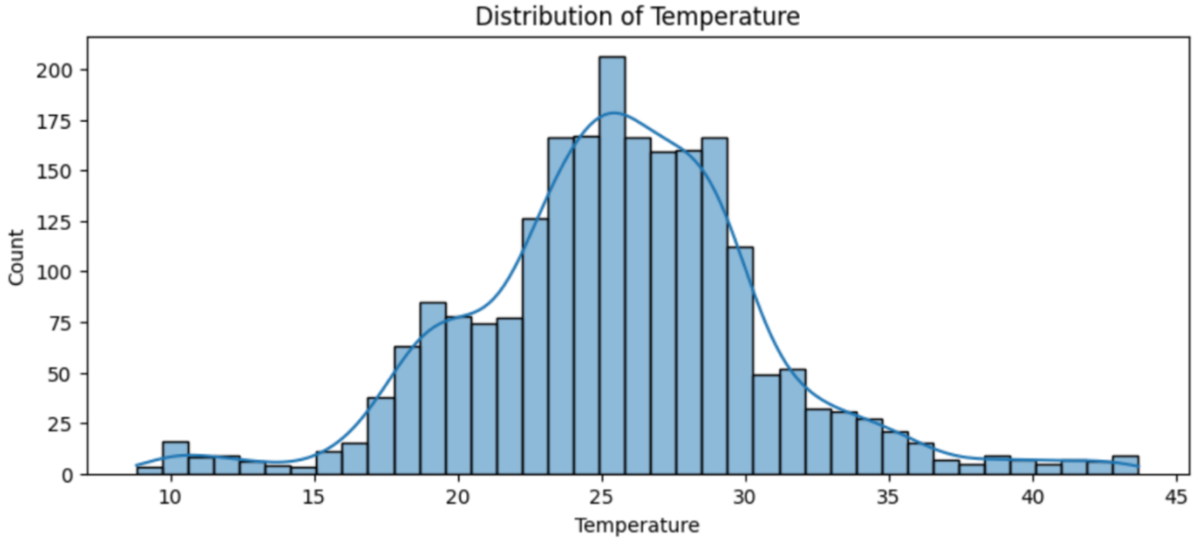
Graphs:

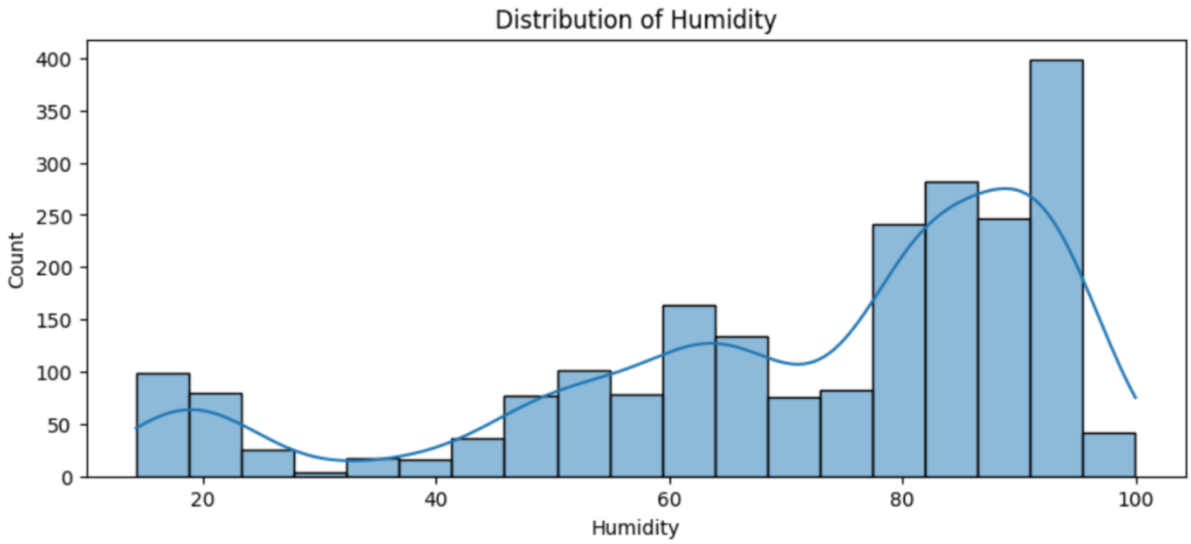
```
# Select only numeric columns for visualization
numeric_df = df.select_dtypes(include=['number'])

# Visualize the distribution of numeric features
numeric_cols = numeric_df.columns

for col in numeric_cols:
    plt.figure(figsize=(10, 4))
    sns.histplot(numeric_df[col], kde=True)
    plt.title(f'Distribution of {col}')
    plt.show()

# Visualize relationships between numeric features
sns.pairplot(numeric_df)
plt.show()

# Generate correlation heatmap for numeric columns
plt.figure(figsize=(5, 3))
sns.heatmap(numeric_df.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```

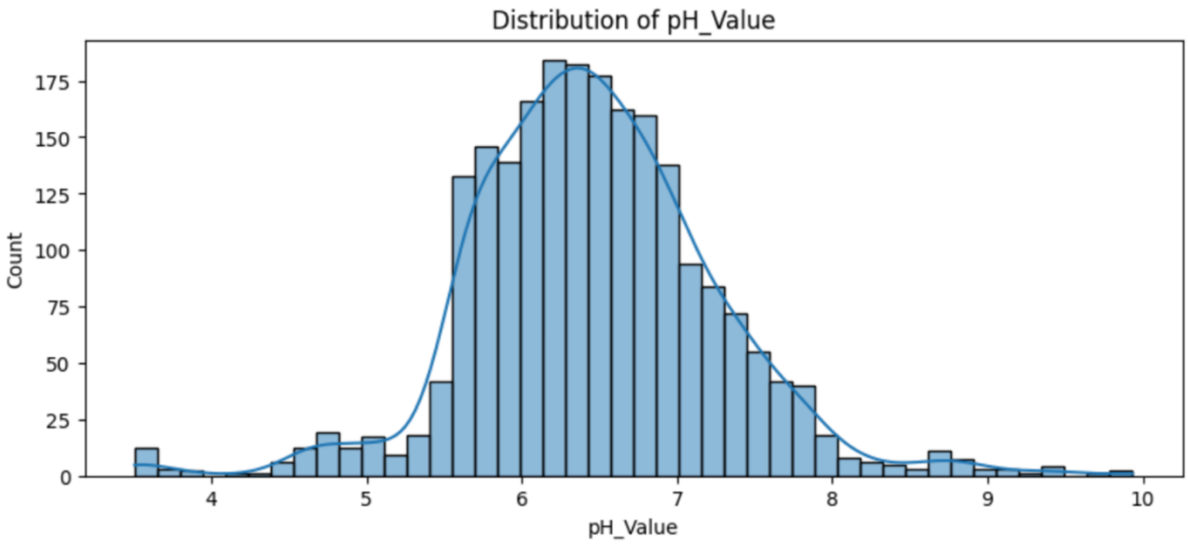

Distribution of Nitrogen
Count
Nitrogen

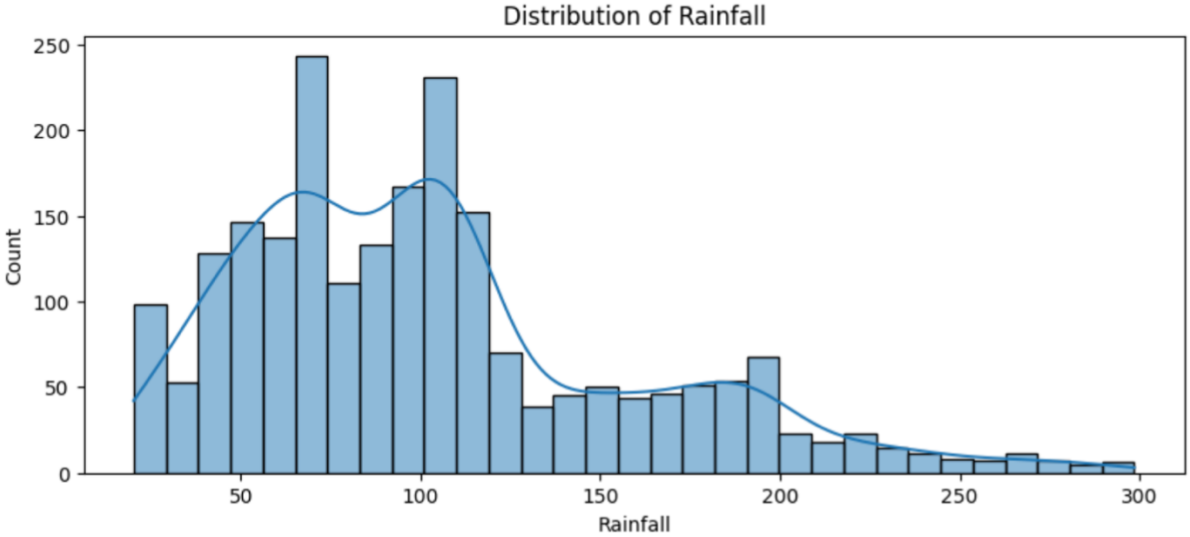

Distribution of Phosphorus
Count
Phosphorus


Distribution of Potassium
Count
Potassium

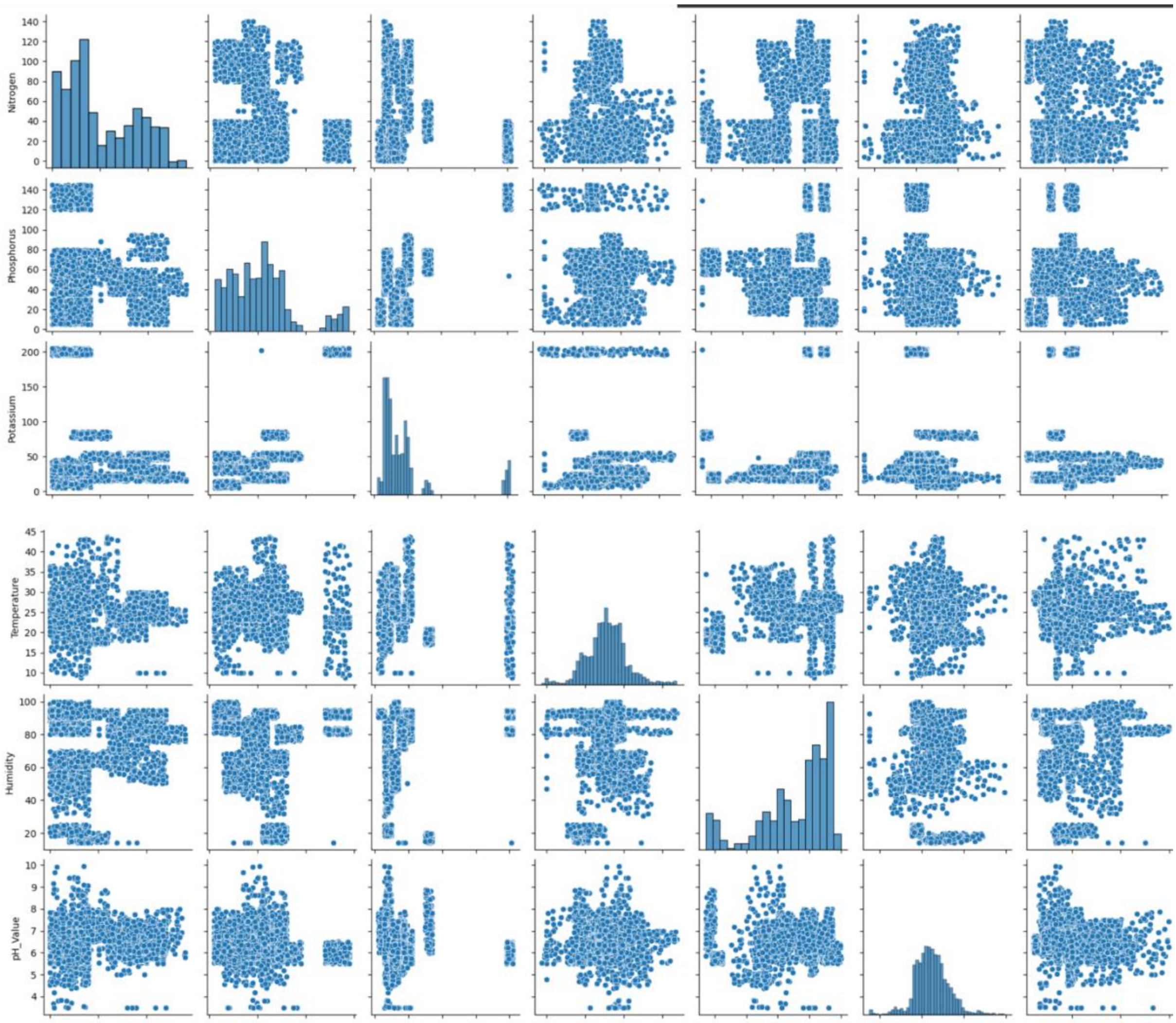

Distribution of Temperature
Count
Temperature
0
25
50
75
100
125
150
175
200
10
15
20
25
30
35
40
45


Distribution of Humidity
Count
Humidity
0
50
100
150
200
250
300
350
400
20
40
60
80
100

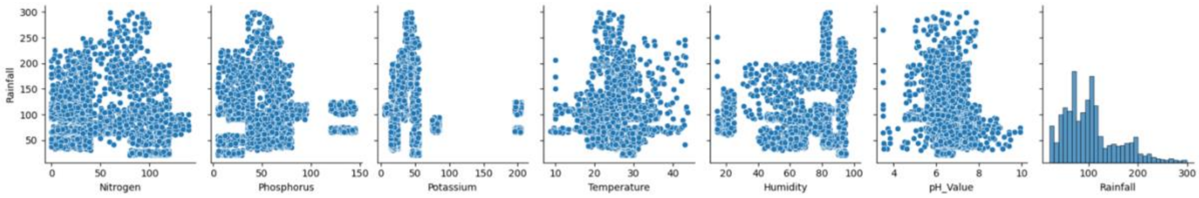

Distribution of pH_Value
Count
pH_Value
0
25
50
75
100
125
150
175
4
5
6
7
8
9
10



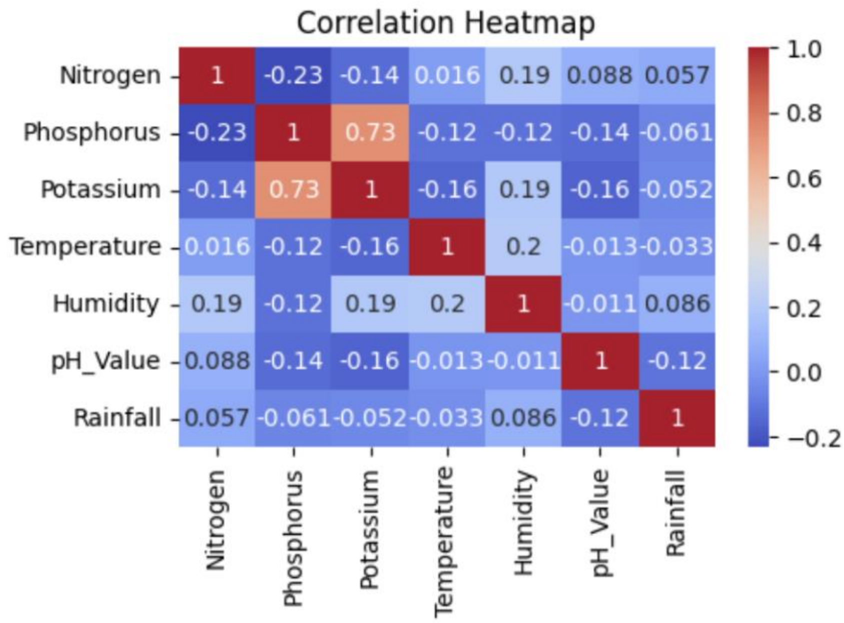
Distribution of Rainfall
Count
Rainfall
0
50
100
150
200
250
300



Nitrogen
Phosphorus
Potassium
Temperature
Humidity
pH_Value



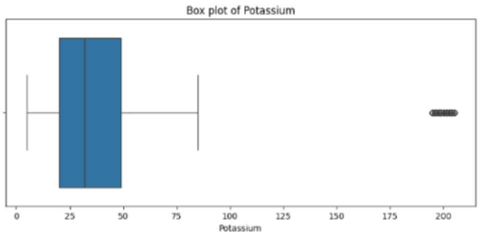
Correlation map:

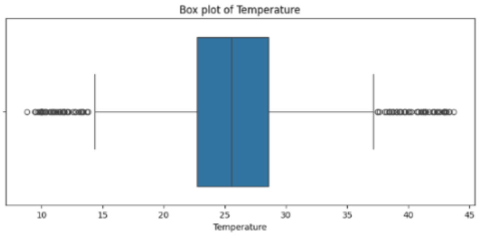


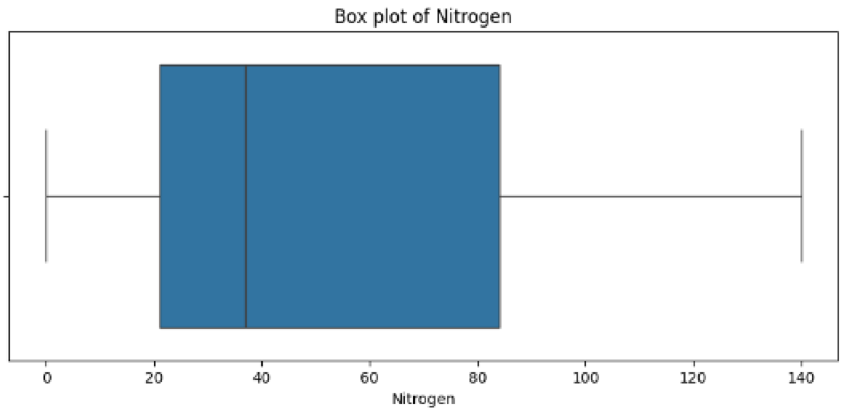
Identify patterns and anomalies

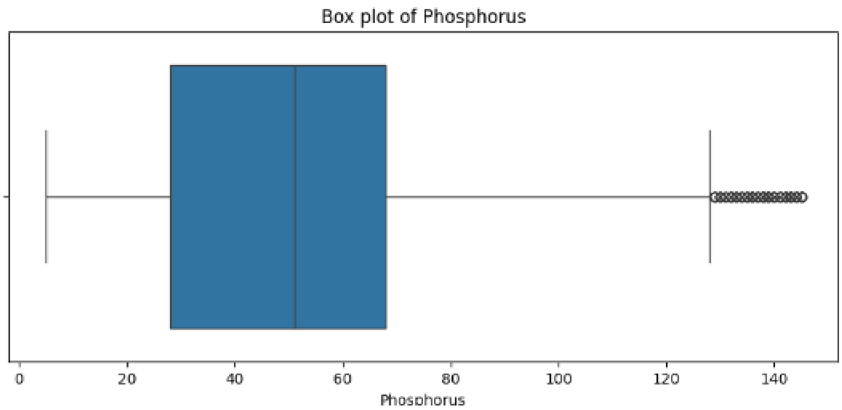
```
[ ]  # Check for missing values
     print("Missing Values:\n", df.isnull().sum())

     # Check for outliers using box plots
     for col in numeric_cols:
         plt.figure(figsize=(10, 4))
         sns.boxplot(x=df[col])
         plt.title(f'Box plot of {col}')
         plt.show()
```

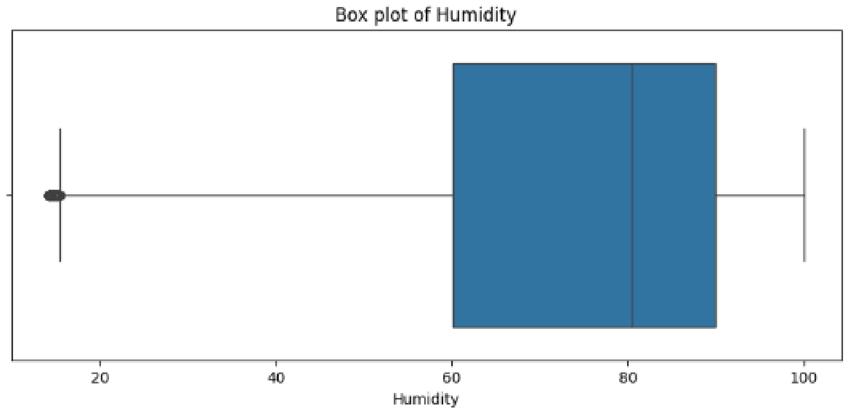
```
Missing Values:
 Nitrogen       0
Phosphorus      0
Potassium       0
Temperature     0
Humidity        0
pH_Value        0
Rainfall        0
Crop            0
dtype: int64
```

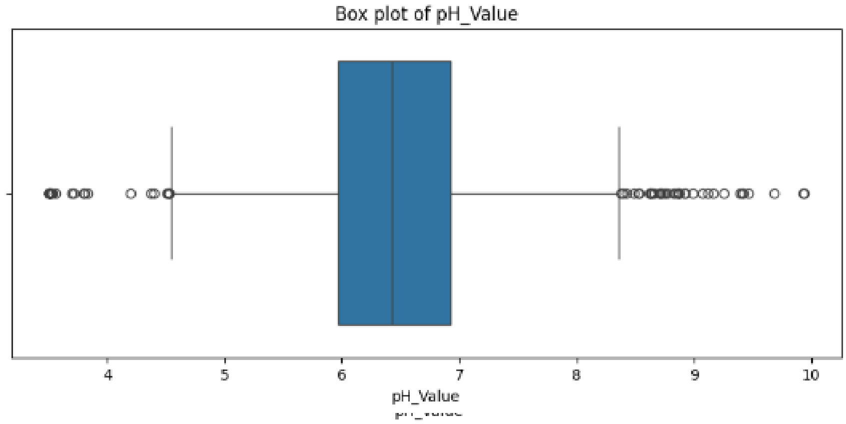


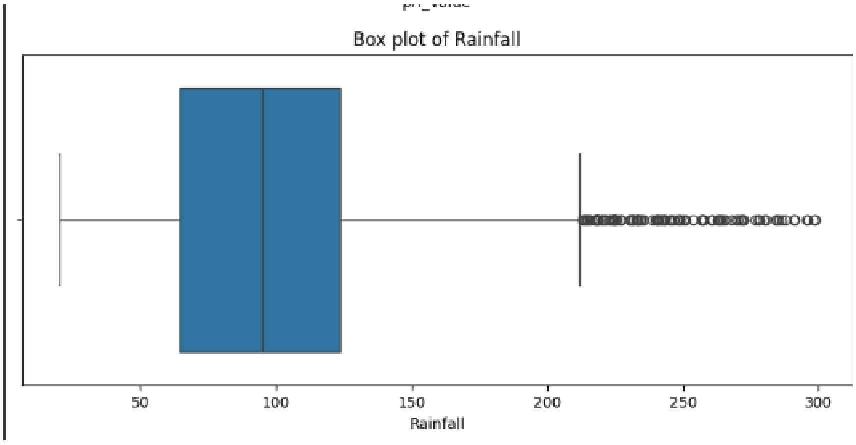






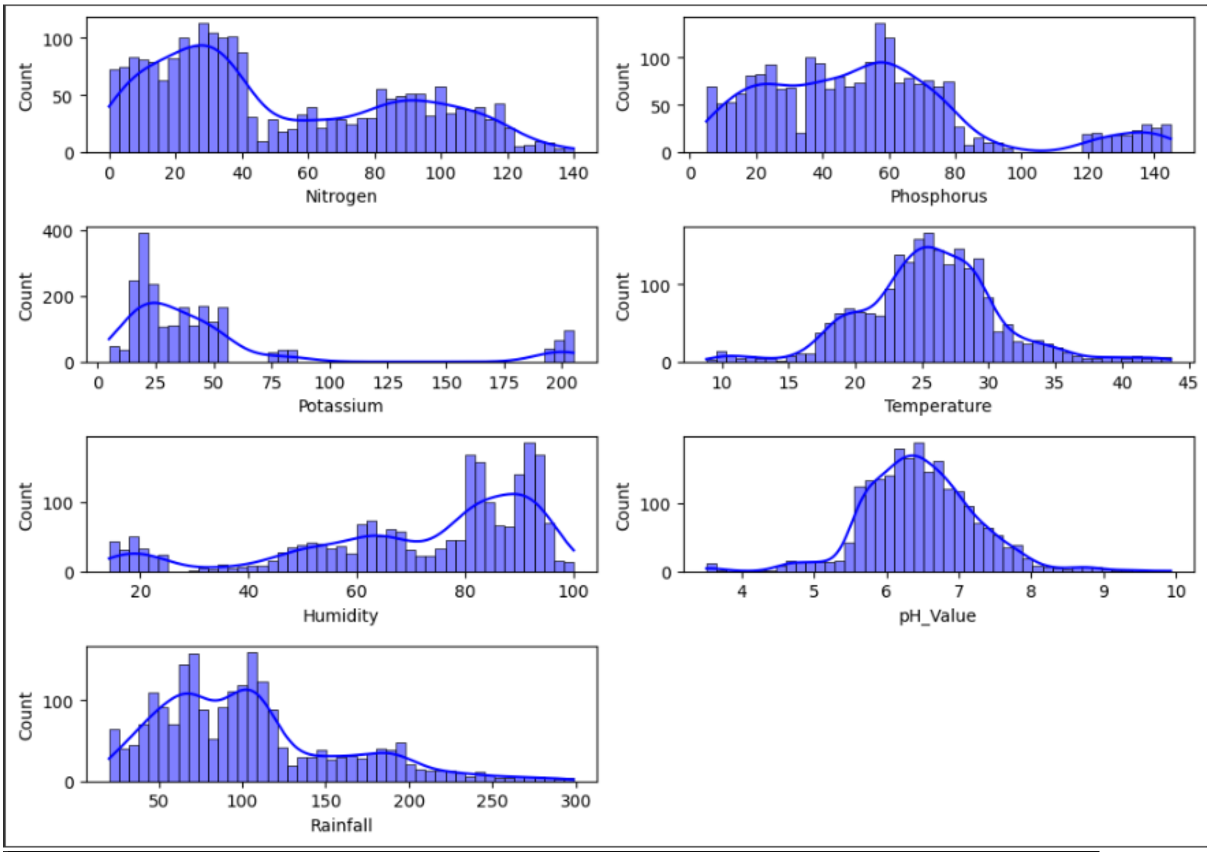




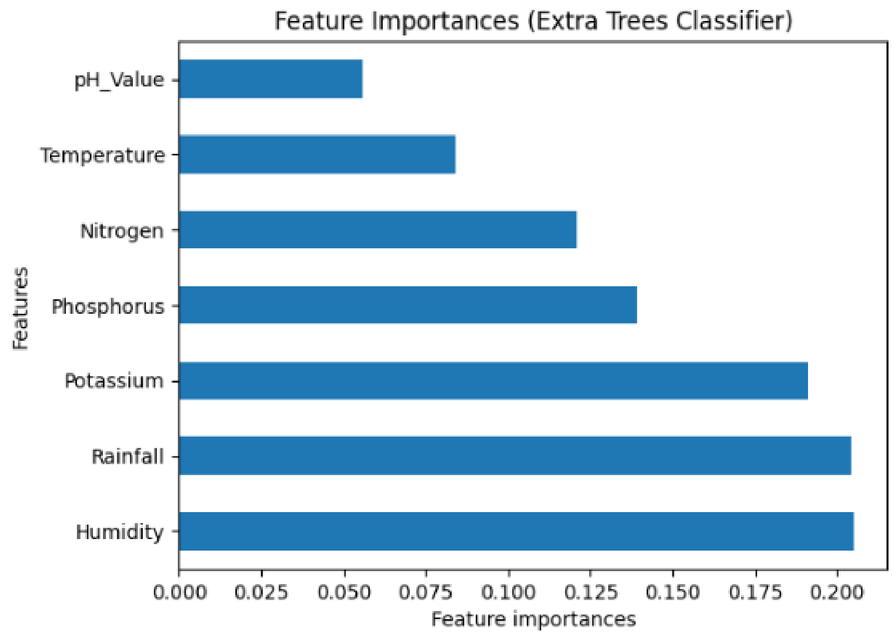


Univariate Analysis

```
[ ]  # Select specific columns from your DataFrame
     selected_columns = ['Nitrogen', 'Phosphorus', 'Potassium', 'Temperature', 'Humidity', 'pH_Value', 'Rainfall']

     # Determine the number of rows and columns for the subplot grid
     num_rows = (len(selected_columns) + 1) // 2  # Round up to the nearest integer
     num_cols = 2

     plt.figure(figsize=(10, 7))
     for i, col in enumerate(selected_columns):
         plt.subplot(num_rows, num_cols, i + 1)
         sns.histplot(data=df, x=col, kde=True, bins=round(np.sqrt(len(df))), color='b')

     plt.tight_layout()  # Adjust spacing between subplots
     plt.show()
```




```python
X= df.drop("Crop",axis= 1) #assume X is indepedendent feature

y= df['Crop'] #assume y is a target variable

feat_selection = ExtraTreesClassifier()
feat_selection.fit(X, y)

feat_importances = pd.Series(feat_selection.feature_importances_, index=X.columns)  # Corrected variable name
feat_importances.nlargest(len(df.columns)).plot(kind='barh')
plt.ylabel("Features")
plt.xlabel("Feature importances")
plt.title("Feature Importances (Extra Trees Classifier)")
plt.show()
```



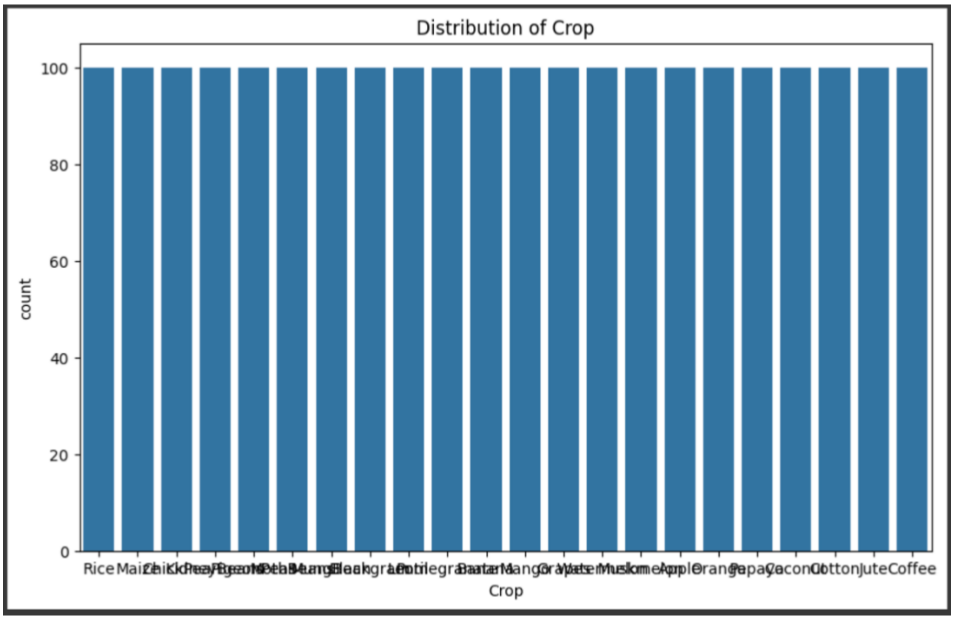
4. Choose an appropriate approach that can be used for model building

After analysing our database we understand that our database follows a classification approach. Where our target variable is ‘Crop’ as we decide the crop to be planted based on various factors like pH_Value, Temperature, Nitrogen, Phosphorus, Potassium, Rainfall, Humidity.

```
target_variable = 'Crop'

# Display unique values in the target variable
unique_values = df[target_variable].unique()
print(f"Unique values in target variable '{target_variable}':")
print(unique_values)

# Display the distribution of the target variable
distribution = df[target_variable].value_counts()
print(f"\nDistribution of target variable '{target_variable}':")
print(distribution)
```

```
Unique values in target variable 'Crop':
['Rice' 'Maize' 'ChickPea' 'KidneyBeans' 'PigeonPeas' 'MothBeans'
 'MungBean' 'Blackgram' 'Lentil' 'Pomegranate' 'Banana' 'Mango' 'Grapes'
 'Watermelon' 'Muskmelon' 'Apple' 'Orange' 'Papaya' 'Coconut' 'Cotton'
 'Jute' 'Coffee']

Distribution of target variable 'Crop':
Crop
Rice           100
Maize          100
Jute           100
Cotton         100
Coconut        100
Papaya         100
Orange         100
Apple          100
Muskmelon      100
Watermelon     100
Grapes         100
Mango          100
Banana         100
Pomegranate    100
Lentil         100
Blackgram      100
MungBean       100
MothBeans      100
PigeonPeas     100
KidneyBeans    100
ChickPea       100
Coffee         100
Name: count, dtype: int64
```

```python
import seaborn as sns
import matplotlib.pyplot as plt

# Plot the distribution of the target variable
plt.figure(figsize=(10, 6))
sns.countplot(x=target_variable, data=df)
plt.title(f'Distribution of {target_variable}')
plt.show()
```



```python
# Summary statistics of the target variable
summary_stats = df[target_variable].describe()
print(f"\nSummary statistics of target variable '{target_variable}':")
print(summary_stats)
```

```
Summary statistics of target variable 'Crop':
count      2200
unique       22
top        Rice
freq        100
Name: Crop, dtype: object
```

5. **Develop Python code to build the model**

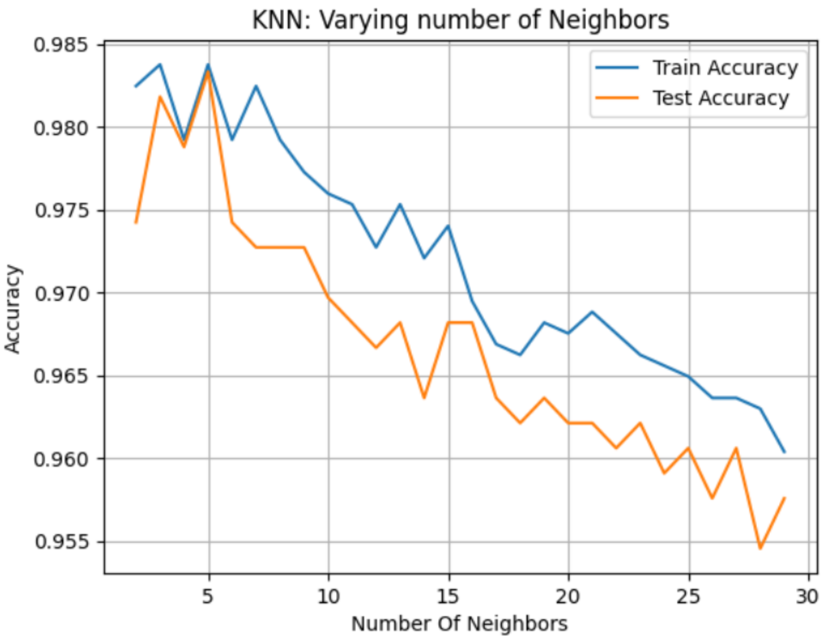
   The models we used in our analysis are:

   - **KNN Classifier**:

```python
# Correctly split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=20)

train_score = {}
test_score = {}
n_neighbors = np.arange(2, 30, 1)
for neighbor in n_neighbors:
    knn = KNeighborsClassifier(n_neighbors=neighbor)
    knn.fit(X_train, y_train)
    train_score[neighbor]=knn.score(X_train, y_train)
    test_score[neighbor]=knn.score(X_test, y_test)

print(f'Train Accuracies: \n{train_score}\n\nTest Accuracies:\n{test_score}')
```

```
Train Accuracies:
{2: 0.9863636363636363, 3: 0.9896103896103896, 4: 0.9837662337662337, 5: 0.9889
610389610389, 6: 0.9837662337662337, 7: 0.9876623376623377, 8: 0.98311688311688
32, 9: 0.9824675324675325, 10: 0.9805194805194806, 11: 0.9805194805194806, 12:
0.9792207792207792, 13: 0.9805194805194806, 14: 0.9772727272727273, 15: 0.97922
07792207792, 16: 0.9753246753246754, 17: 0.9733766233766233, 18: 0.971428571428
5714, 19: 0.9727272727272728, 20: 0.9714285714285714, 21: 0.9733766233766233, 2
2: 0.9727272727272728, 23: 0.9714285714285714, 24: 0.9707792207792207, 25: 0.97
07792207792207, 26: 0.9707792207792207, 27: 0.9701298701298702, 28: 0.967532467
5324676, 29: 0.9655844155844155}

Test Accuracies:
{2: 0.9712121212121212, 3: 0.9818181818181818, 4: 0.9803030303030303, 5: 0.9848
484848484849, 6: 0.9742424242424242, 7: 0.9757575757575757, 8: 0.97272727272727
28, 9: 0.9742424242424242, 10: 0.9696969696969697, 11: 0.9712121212121212, 12:
0.9666666666666667, 13: 0.9696969696969697, 14: 0.9666666666666667, 15: 0.96969
69696969697, 16: 0.9666666666666667, 17: 0.9651515151515152, 18: 0.963636363636
3636, 19: 0.9666666666666667, 20: 0.9651515151515152, 21: 0.9651515151515152, 2
2: 0.9621212121212122, 23: 0.9636363636363636, 24: 0.9590909090909091, 25: 0.96
06060606060606, 26: 0.9575757575757575, 27: 0.9606060606060606, 28: 0.954545454
5454546. 29: 0.9575757575757575}
```

```python
plt.plot(n_neighbors, train_score.values(), label="Train Accuracy")
plt.plot(n_neighbors, test_score.values(), label="Test Accuracy")
plt.xlabel("Number Of Neighbors")
plt.ylabel("Accuracy")
plt.title("KNN: Varying number of Neighbors")
plt.legend()

plt.grid()
plt.show()
```



```
#Create a KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors = 20)

# Fit the classifier on the training data
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
train_accuracy = model.score(X_train, y_train)
test_accuracy = model.score(X_test, y_test)

print(f'Train Accuracy: {train_accuracy}')
print(f'test accuracy: {test_accuracy}')
```

```
Train Accuracy: 0.9675324675324676
test accuracy: 0.9621212121212122
```

- **GAUSSIAN NAÏVE BAYES**

```
# Create a Gaussian Naive Bayes classifier
GNB = GaussianNB()

# Fit the classifier on the training data
GNB.fit(X_train, y_train)
y_pred_gnb = GNB.predict(X_test)


print(f'Train accuracy for GNB: {GNB.score(X_train, y_train)}')
print(f'Test accuracy for GNB: {GNB.score(X_test, y_test)}')
```

```
Train accuracy for GNB: 0.9915584415584415
Test accuracy for GNB: 0.9863636363636363
```

- **MULTINOMIAL NAÏVE BAYES**

```
[ ]  ## Create a MultinomialNB  classifier
     MNB = MultinomialNB()

     MNB.fit(X_train, y_train)
     y_pred_mnb = MNB.predict(X_test)

     # accuracy
     print(f'Train accuracy for GNB: {MNB.score(X_train, y_train)}')
     print(f'Test accuracy for GNB: {MNB.score(X_test, y_test)}')

     Train accuracy for GNB: 0.887012987012987
     Test accuracy for GNB: 0.8772727272727273
```

- **DESCION TREE**

```
[ ]  # Initialize and train the Decision Tree model
     model = DecisionTreeClassifier()
     model.fit(X_train, y_train)

     # Calculate the training and testing accuracy
     train_accuracy = model.score(X_train, y_train)
     test_accuracy = model.score(X_test, y_test)

     print("Training Accuracy: ", train_accuracy)
     print("Testing Accuracy: ", test_accuracy)

     Training Accuracy:  1.0
     Testing Accuracy:  0.9772727272727273
```

As the mode wasn't well fitted we tuned the best parameters

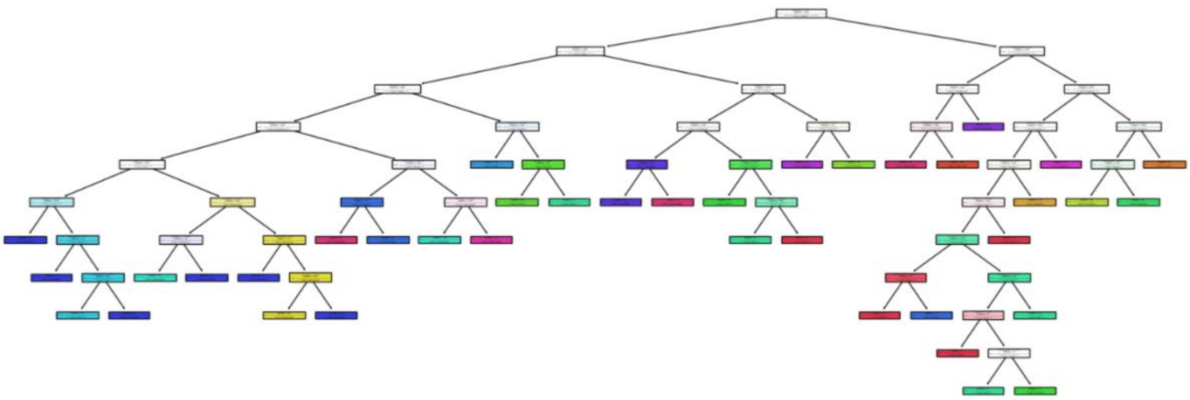
```
[ ]  # Define the parameter grid
     param_grid = {
         'criterion': ['gini', 'entropy'],
         'max_depth': [None, 50, 20, 15],
         'min_samples_split': [2, 7, 10],
         'min_samples_leaf': [1, 2, 3]
     }

     # Initialize the model
     model = DecisionTreeClassifier()

     # Initialize GridSearchCV
     grid_search = GridSearchCV(model, param_grid, cv=5, scoring='accuracy')

     # Fit the GridSearchCV to the training data
     grid_search.fit(X_train, y_train)

     # Get the best parameters and estimator
     best_params = grid_search.best_params_
     best_estimator = grid_search.best_estimator_

     print('Best Hyperparameters:', best_params)

     Best Hyperparameters: {'criterion': 'gini', 'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 2}
```

```
# Initialize the model with specified hyperparameters
model = DecisionTreeClassifier(criterion='entropy', max_depth=50, min_samples_leaf=1, min_samples_split=2)

# Fit the model to the training data
model.fit(X_train, y_train)
y_pred_dc= model.predict(X_test)

# Calculate and print the training and testing accuracy
print("Training Accuracy:", model.score(X_train, y_train))
print("Testing Accuracy:", model.score(X_test, y_test))
```

```
Training Accuracy: 1.0
Testing Accuracy: 0.9803030303030303
```

Plotting the desicon tree

```
#plotting the decision tree

plt.figure(figsize=(20,7))
plot_tree(model, filled=True, feature_names=X.columns, class_names= model.classes_)
plt.show()
```



- **Random Forest Classifier**

```
rf= RandomForestClassifier()


# Fit the model to the training data
rf.fit(X_train,y_train)

y_pred_rf= rf.predict(X_test)

# Calculate and print the training and testing accuracy
print("Training Accuracy:", model.score(X_train, y_train))
print("Testing Accuracy:", model.score(X_test, y_test))
```

```
Training Accuracy: 1.0
Testing Accuracy: 0.9803030303030303
```

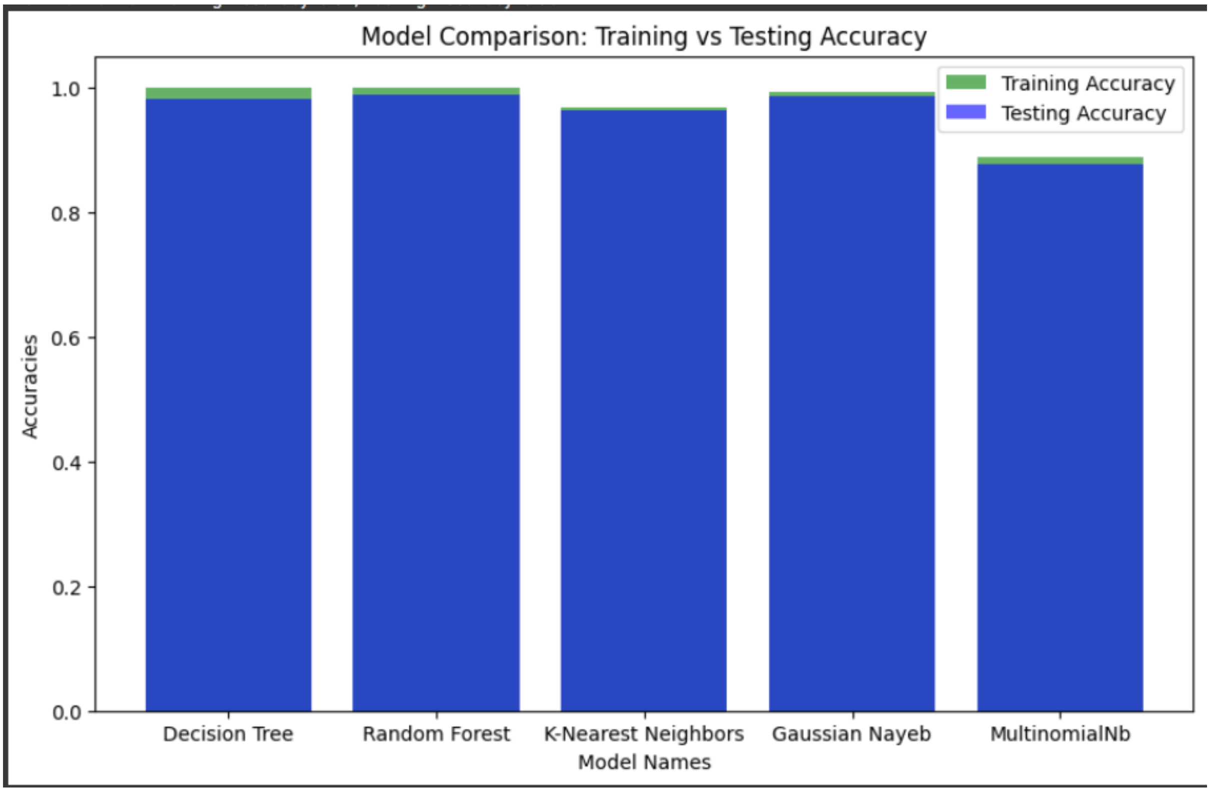
6. **Evaluate the built model using appropriate evaluation metrics.**

```python
# Initialize the models
models = {
    'Decision Tree': DecisionTreeClassifier(criterion='entropy', max_depth=50, min_samples_leaf=1, min_samples_split=2),
    'Random Forest': RandomForestClassifier(),
    'K-Nearest Neighbors': KNeighborsClassifier(n_neighbors=20),
    'Gaussian Nayeb' : GaussianNB(),
    'MultinomialNb': MultinomialNB()
}

# Train the models and calculate accuracies
accuracies = {'Model': [], 'Training Accuracy': [], 'Testing Accuracy': []}

for model_name, model in models.items():
    model.fit(X_train, y_train)
    train_acc = model.score(X_train, y_train)
    test_acc = model.score(X_test, y_test)
    accuracies['Model'].append(model_name)
    accuracies['Training Accuracy'].append(train_acc)
    accuracies['Testing Accuracy'].append(test_acc)
    print(f'{model_name} -Training Accuracy: {train_acc:.2f}, Testing Accuracy: {test_acc:.2f}')
```

```python
# Convert accuracies dictionary to DataFrame for better visualization
acc_df = pd.DataFrame(accuracies)

# Plot the accuracies
plt.figure(figsize=(10, 6))
plt.bar(acc_df['Model'], acc_df['Training Accuracy'], alpha=0.6, color='g', label='Training Accuracy')
plt.bar(acc_df['Model'], acc_df['Testing Accuracy'], alpha=0.6, color='b', label='Testing Accuracy')
plt.xlabel('Model Names')
plt.ylabel('Accuracies')
plt.title('Model Comparison: Training vs Testing Accuracy')
plt.legend()
plt.show()
```

Output:

```
Random Forest -Training Accuracy: 1.00, Testing Accuracy: 0.99
K-Nearest Neighbors -Training Accuracy: 0.97, Testing Accuracy: 0.96
Gaussian Nayeb -Training Accuracy: 0.99, Testing Accuracy: 0.99
MultinomialNb -Training Accuracy: 0.89, Testing Accuracy: 0.88
```



Based on the above comparison of different models we have concluded that Descison tree classifier and Random forest classifier have highest accuracy to predict new crops.

Hence computing classification reports only for those models

Random forest classifier:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix

# Fit the model to the training data
rf = RandomForestClassifier()
rf.fit(X_train, y_train)

# Make predictions on the test data
y_pred_rf = rf.predict(X_test)

# Calculate and print the training and testing accuracy
print("Training Accuracy:", rf.score(X_train, y_train))
print("Testing Accuracy:", rf.score(X_test, y_test))

# Generate and print the classification report
print("Classification Report:")
print(classification_report(y_test, y_pred_rf))

# Generate and print the confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_rf))
```

```
Training Accuracy: 1.0
Testing Accuracy: 0.9878787878787879
Classification Report:
              precision    recall  f1-score   support

       Apple       1.00      1.00      1.00        29
      Banana       1.00      1.00      1.00        25
   Blackgram       1.00      0.97      0.99        36
    ChickPea       1.00      1.00      1.00        32
     Coconut       1.00      1.00      1.00        31
      Coffee       1.00      1.00      1.00        34
      Cotton       1.00      1.00      1.00        34
      Grapes       1.00      1.00      1.00        23
        Jute       0.81      1.00      0.90        30
 KidneyBeans       1.00      1.00      1.00        29
      Lentil       1.00      1.00      1.00        33
       Maize       0.96      1.00      0.98        27
       Mango       1.00      1.00      1.00        29
   MothBeans       1.00      1.00      1.00        33
    MungBean       1.00      1.00      1.00        33
   Muskmelon       1.00      1.00      1.00        25
      Orange       1.00      1.00      1.00        34
      Papaya       1.00      1.00      1.00        21
  PigeonPeas       1.00      1.00      1.00        37
 Pomegranate       1.00      1.00      1.00        23
        Rice       1.00      0.77      0.87        30
  Watermelon       1.00      1.00      1.00        32

    accuracy                           0.99       660
   macro avg       0.99      0.99      0.99       660
weighted avg       0.99      0.99      0.99       660
```

```
Confusion Matrix:
[[29  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0 25  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0 35  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0 32  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0 31  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0 34  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0 34  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0 23  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0 30  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0 29  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0 33  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0 27  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0 29  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0 33  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0 33  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 25  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 34  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 21  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 37  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 23  0  0]
 [ 0  0  0  0  0  0  0  0  7  0  0  0  0  0  0  0  0  0  0  0 23  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 32]]
```

Decision tree:

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix

# Initialize the model with specified hyperparameters
model = DecisionTreeClassifier(criterion='entropy', max_depth=50, min_samples_leaf=1, min_samples_split=2)

# Fit the model to the training data
model.fit(X_train, y_train)

# Predict the labels for the test set
y_pred_dc = model.predict(X_test)

# Calculate and print the training and testing accuracy
print("Training Accuracy:", model.score(X_train, y_train))
print("Testing Accuracy:", model.score(X_test, y_test))

# Generate and print the classification report
report = classification_report(y_test, y_pred_dc)
print("Classification Report:\n", report)

# Generate and print the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred_dc)
print("Confusion Matrix:\n", conf_matrix)
```

```
Training Accuracy: 1.0
Testing Accuracy: 0.9803030303030303
Classification Report:
              precision    recall  f1-score   support

       Apple       1.00      1.00      1.00        29
      Banana       1.00      1.00      1.00        25
   Blackgram       1.00      0.97      0.99        36
    ChickPea       1.00      1.00      1.00        32
     Coconut       0.94      1.00      0.97        31
      Coffee       1.00      1.00      1.00        34
      Cotton       1.00      0.97      0.99        34
      Grapes       1.00      1.00      1.00        23
        Jute       0.78      0.97      0.87        30
 KidneyBeans       1.00      1.00      1.00        29
      Lentil       1.00      1.00      1.00        33
       Maize       0.93      1.00      0.96        27
       Mango       1.00      1.00      1.00        29
   MothBeans       1.00      1.00      1.00        33
    MungBean       1.00      1.00      1.00        33
   Muskmelon       1.00      1.00      1.00        25
      Orange       1.00      1.00      1.00        34
      Papaya       1.00      1.00      1.00        21
  PigeonPeas       1.00      1.00      1.00        37
 Pomegranate       1.00      0.91      0.95        23
        Rice       0.96      0.73      0.83        30
  Watermelon       1.00      1.00      1.00        32

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

```
Confusion Matrix:
 [[29  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0 25  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0 35  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0 32  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0 31  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0 34  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0 33  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0 23  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0 29  0  0  0  0  0  0  0  0  0  0  0  1  0]
 [ 0  0  0  0  0  0  0  0  0 29  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0 33  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0 27  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0 29  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0 33  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0 33  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 25  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 34  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 21  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 37  0  0  0]
 [ 0  0  0  0  2  0  0  0  0  0  0  0  0  0  0  0  0  0  0 21  0  0]
 [ 0  0  0  0  0  0  0  0  8  0  0  0  0  0  0  0  0  0  0  0 22  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 32]]
```

Conclusion:

- Potassium and phosphorus are highly correlated with each other. so production can be effected by these features.
- Decision tree classifier and Random forest classifier gets the great training and test accuracy to predict crops with new data(evaluation metric scores mentioned above).

Links:

➔ [Link to database]

➔ [Link to Colab file]