CROP AND FERTILIZER RECOMMENDATION USING MACHINE LEARNING ALGORITHMS

CREATIVE AND INNOVATIVE PROJECT REPORT

Submitted by

KAVIYA R V (2019103538)

KEERTHANA A (2019103028)

RAMYA S (2019103049)

in partial fulfillment of the requirements for the award of the

degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

COLLEGE OF ENGINEERING, GUINDY

ANNA UNIVERSITY: CHENNAI 600 025

JUNE 2022

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certificate that this project request titled **CROP AND FERTILIZER RECOMMENDATION USING MACHINE LEARNING ALGORITHMS** the bonafide work of **KAVIYA RV (2019103538)** and **KEERTHANA A (2019103028)** and **RAMYA S (2019103049)** who carried out the project work under my supervision, for the fulfillment of the requirements for the award of the degree of Bachelor of Engineering in Computer Science and Engineering. Certified further that to the best of my knowledge, the work reported herein does not form part of any other thesis on the basis of which a degree or an award was conferred on an earlier occasion on these or any other candidates.

Place: Chennai

Date:

Mrs. Lalitha Devi K

Teaching Fellow,

Department of Computer Science and Engineering,

Anna University, Chennai-25.

COUNTERSIGNED

Dr. Valli S,

Head of the Department,

Department of Computer Science and Engineering, Anna University Chennai,

Chennai - 600 025.

ACKNOWLEDGEMENT

Foremost, we would like to express our sincere gratitude to our project guide, **Mrs. Lalitha Devi K**, Teaching Fellow, Department of Computer Science and Engineering, College of Engineering Guindy, Chennai for her constant source of inspiration. We thank her for the continuous support and guidance which was instrumental in taking the project to successful completion.

We are grateful to **Dr.S.Valli,** Professor and Head, Department of Computer Science and Engineering, College of Engineering Guindy, Chennai for her support and for providing necessary facilities to carry out for our project.

We would also like to thank our friends and family for their encouragement and continued support. We would also like to thank the Almighty for giving us the moral strength to accomplish our task.

KAVIYA RV **KEERTHANA A** **RAMYA S**

ABSTRACT

India is known as an agricultural country, where the recommendations are given by traditional methods. While using these traditional methods there is a high level of failures. Variations in weather, climate, and other such environmental conditions have become a major risk for the healthy existence of agriculture. So, by identifying losses and risk in traditional methods of crop and fertilizer recommendation we have to find solution to these problems to increase of crop yield.

We propose crop and fertilizer recommendation system by giving agricultural data using machine learning algorithms.

INTRODUCTION

India is one among the oldest countries which is still practicing agriculture. But in recent times the trends in agriculture have drastically evolved due to globalization. Various factors have affected the health of agriculture in India. Many new technologies have been evolved to regain the health. One such technique is precision agriculture. Precision agriculture is budding in India. Precision agriculture is the technology of "site-specific" farming.

Recommendation of crops is one major domain in precision agriculture. Recommendation of crops is dependent on various parameters. Precision agriculture aims in identifying these parameters in a site-specific manner in order to resolve. The "site-specific" technique has improved the results yet there is a need to supervise the results of such systems. Not all precision agriculture systems provide accurate results. But in agriculture it is important that the recommendations made are accurate and precise because incase of errors it may lead to heavy material and capital loss.

Many research works is being carried out, in order to attain an accurate and efficient model for crop prediction. Ensemble is one such technique that is included in such research works. Among these various machine learning techniques that are being used in this field.

1.1. OBJECTIVE

The Objective of our model is to recommend the Crops based on soil, weather, humidity, rainfall, and other variables to increase agricultural output. In order to increase the yield of crop our model recommend the suitable fertilizer, this recommendation is based on the various soil properties.

1.2. PROBLEM STATEMENT

- Based on predicted rainfall, soil contents and weather parameters the system will recommend the most suitable crop for cultivation.
- Crop prediction is an essential task for the decision-makers at national and regional levels for rapid decision-making.
- It allows the farmers to choose any of the alternative combinations of fertilizers depending on the recommended crop and soil conditions.
- An accurate crop yield prediction model can help farmers to decide on what to grow and when to grow

1.3. CHALLENGES FACED IN THE SYSTEM

- The major limitations of the Neural Network are reduction in the relative error and decreased prediction efficiency of Crop Yield.
- Similarly, supervised learning techniques were incapable to capture the nonlinear bond between input and output variables faced a problem during the selection of
  fruits grading or sorting.

- The existing model used SVM that classified the crop data based on the texture, shape, color of patterns on the diseased surface as it includes an unambiguous perception of the defects
- An existing technique used CNN that reduced the relative error as well as decreased

  the prediction of crop yield.

1.4. SCOPE OF THE SYSTEM

System will check soil quality and predict the crop yield accordingly along with it provide fertilizer recommendation if needed depending upon the quality of soil. a fertilizer Selection Method improves net yield rate of the crop. It suggests a series of weather, soil type, water density, crop type.

2. LITERATURE SURVEY

AUTHOR	PAPER	YEAR	JOURNAL	KEYFINDINGS
Aakunuri Manjula, Dr.G. Narsima	A Flexible and Extensible Framework for Agricultural Crop Yield Prediction', Conference on Intelligent Systems and Control (ISCO)	2018	IEEE Conference on Intelligent Systems and Control (ISCO)	The paper states the necessity for crop yield prediction and its help in a nation's strategic policy making in agriculture. A framework extensible Crop Yield Prediction Framework

				(XCYPF) is developed.
Rakesh Kumar, M.P. Singh, Prabhat Kumar and J.P. Singh	'Crop Selection Method to Maximize Crop Yield Rate using Machine Learning Technique',	2020	International Conference on (ICSTM).	This paper proposes a Crop Selection Method (CSM) which solves the crop selection problem and improves net yield rate of the crop.
Satish Babu	A Software Model for Precision Agriculture for Small and Marginal Farmers', at the International Centre for Free and Open-Source Software (ICFOSS) Trivandrum, India	2013	IEEE Conference on Global Humanitarian Technology, South Asia Satellite (GHTC-SAS)	It deeply analyses the basics of precision farming.
Anshal Savla, Parul Dhawan,	Survey of classification algorithms for	2015	International Conference on Innovations in	The paper makes a comparative study of classification

Himtanaya Bhadada, Nivedita Israni, Alisha Mandholia, Sanya	formulating yield prediction accuracy in precision agriculture', Innovations in Information, Embedded and Communication systems (ICIIECS)		Information, Embedded and Communication Systems (ICIIECS)	algorithms and their performance in yield prediction in precision Agriculture
A.T.M Shakil Ahamed, Navid Tanzeem Mahmood, Nazmul Hossain, Mohammad Tanzir Kabir, Kallal Das, Faridur Rahman,	Applying Data Mining Techniques to Predict Annual Yield of Major Crops and Recommend Planting Different Crops in Different Districts in Bangladesh	2015	IEEE/ACIS International Conference.	Data mining techniques in paper are used to estimate the crop yield for cereal crops in major districts of Bangladesh.

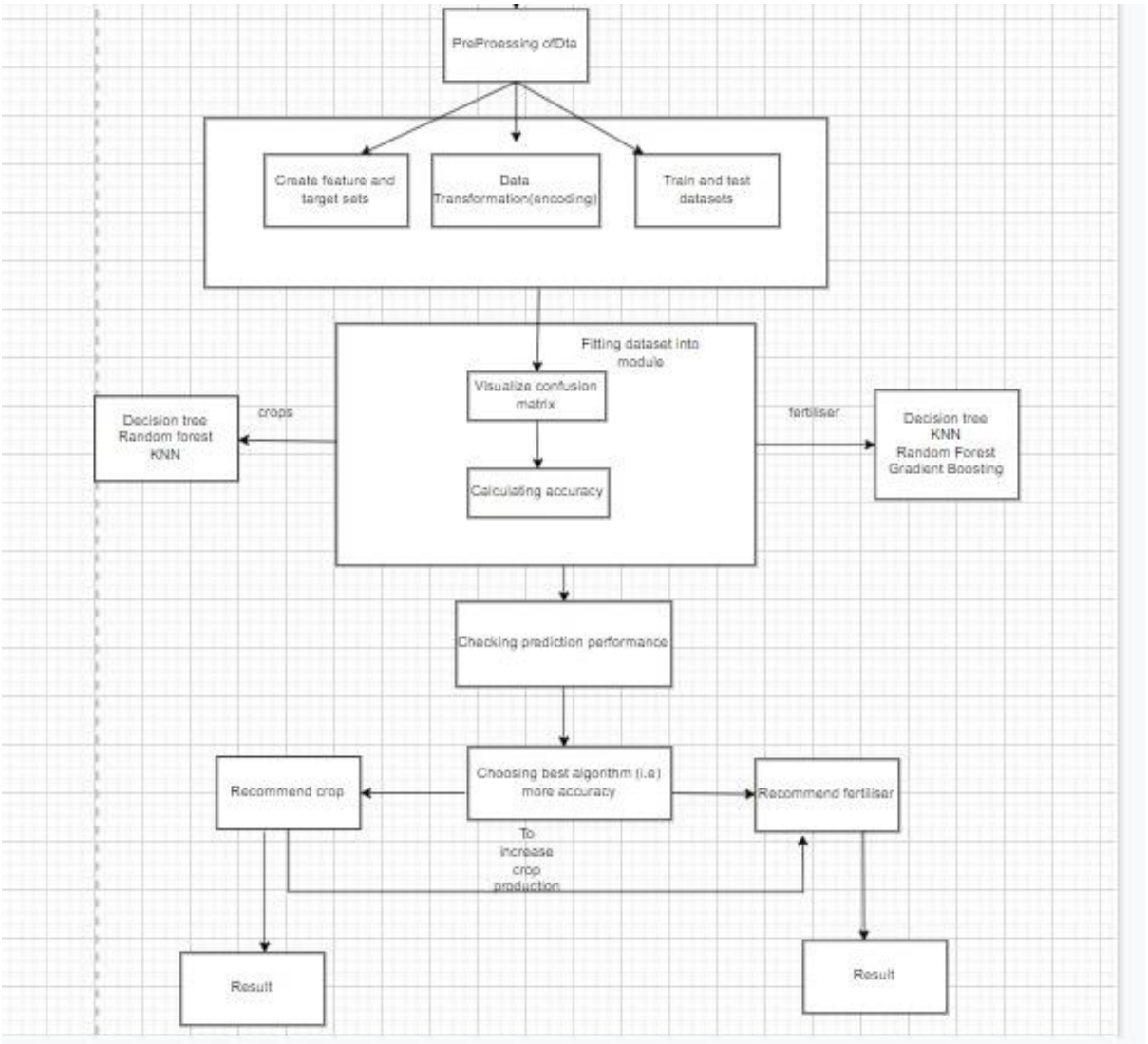
Rashedur M Rahman (2015),, (SNPD) IEEE/ACIS International Conference				

3. PROPOSED APPROACH

This model focuses on predicting the crop type in advance by analysing factors like district (assuming same weather and soil parameters in a particular district), state, season, crop type using various supervised machine learning techniques. This helps the farmers to know the crop yield in advance to plan and choose a crop that would give a better yield**.** In this project we are dealing with a classification problem in fertilizer recommendation. Therefore, we have fitted our dataset into few popular machine learning models such as K-Nearest Neighbours, Decision Tree, Random Forest and Gradient Boosting to predict fertilizer recommendation for rice and coconut. Finally, we will select one or two models based on their accuracy and precision

4. SYSTEM DESIGN

4.1 BLOCK DIAGRAMM



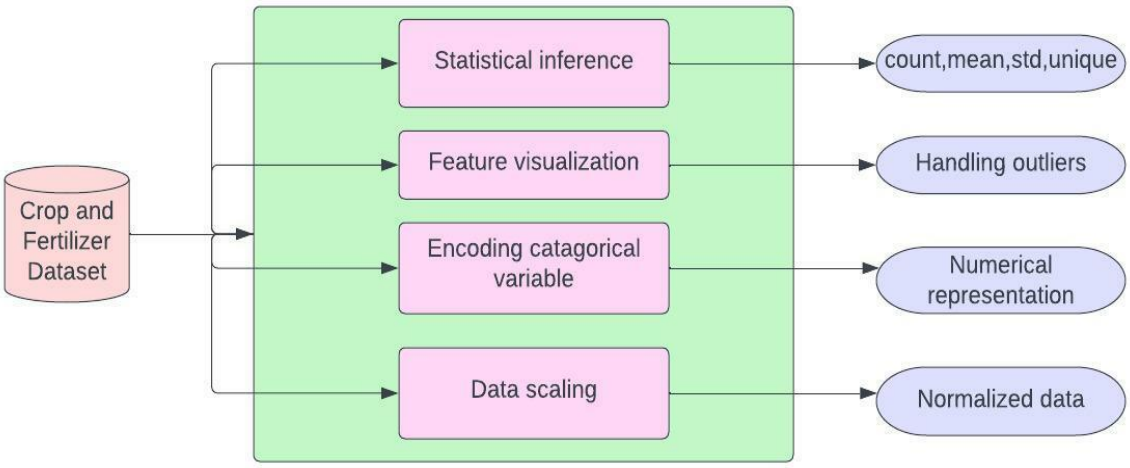
4.2 SYSTEM REQUIREMENTS

- Python, Collab, Jupyter
- Python Packages - Scikit Learn, Pandas

5. DETAILED ARCHITECTURE

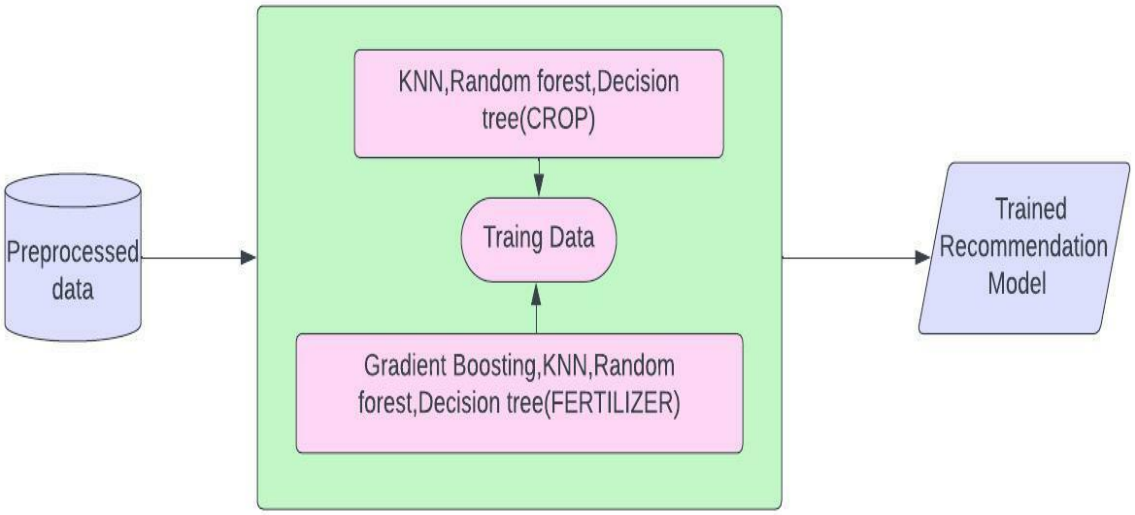
- Exploratory data analysis and pre-processing
- Model development
- Crop recommendation model building
- Fertilizer recommendation model building
- Model evaluation

5.1 Exploratory data analysis and pre-processing



- **info()** method shows some of the characteristics of the data such as Column Name, No. of non-null values of our columns, Data type of the data, and Memory Usage.
- **describe()** method shows basic statistical characteristics of each numerical feature
- **Drop the missing values**
- Handling the **outliers** in the data, the **extreme values** in the data. We can find the outliers in our data using a Boxplot.
- By using minmaxscaler, put our variable values inside a defined range (like 0-1) so that they all have the same range.
- df.corr() is used to find the pairwise correlation of all columns in the data frame. Any 'nan' values are automatically excluded.

5.2 Model development



- After getting the cleaned data we build the model by using various machine learning algorithms. The two different set of algorithms were used for both crop and fertilizer recommendation respectively.
- Analysing the performance of each and every model we have to fit the model with suitable algorithm which give high accuracy among all of these

5.3 Crop recommendation model building

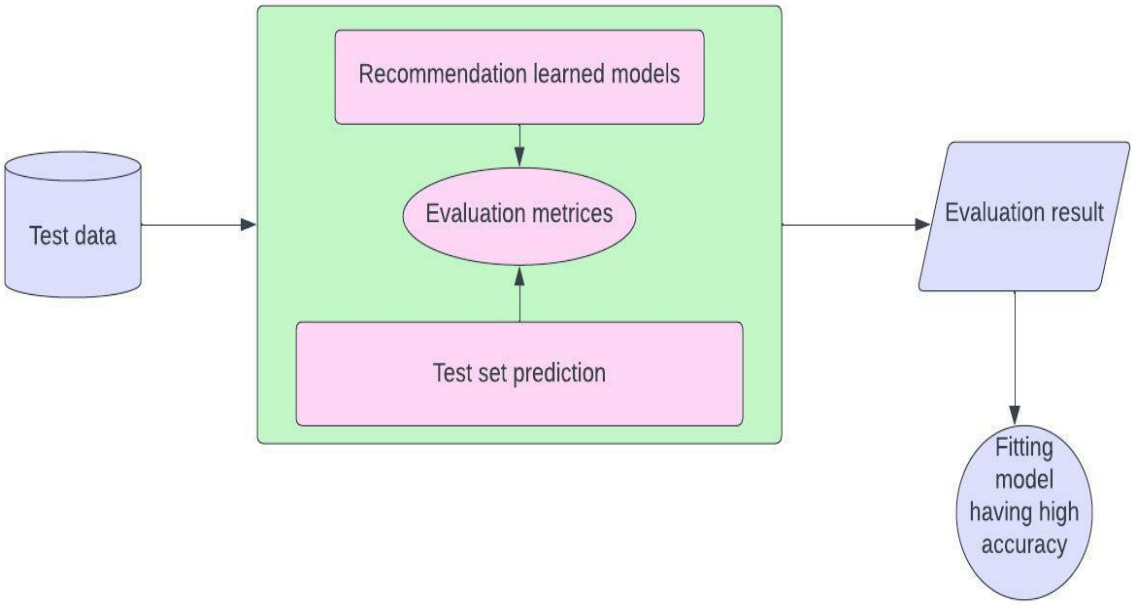
- A call to the function yields an attribute and a target column **(CROP)** of the same length import NumPy as np from scikit learn. datasets import make_classification **X, y = make_classification () print (X. shape, y. shape)**
- generate a multi-output data with a make_multilabel_classification function **multi_target_forest = MultiOutputClassifier (forest, n_jobs=-1)**
- Predict the class labels **(CROP)** based on an existing tree fit and new predictive features. Arguments: fit the result of tree.**DecisionTreeClassifier.fit (); uses** the last fit model if None. features the new X array/ new predictive features to use; should be (p x n), n samples with p features.

- implementing KNN on crop data set by using scikit learn **KneighborsClassifer**. (**knn_clf=KNeighborsClassifier(3))**
- **knn_clf.fit(x_train, y_train), multi_target_decision.fit(x_train, y_train), multi_target_forest.fit (x_train, y_train**)-fit respective models

5.4 Fertilizer recommendation model building

- **X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,random_state=42, stratify=y)-**assign the data into train and test sets randomly to ensure that the train and test sets contain the representative samples from the source dataset
- Importing libraries for classification
- **Knn=KNeighborsClassifier(n_neighbors=k),**
  model_DT=DecisionTreeClassifier(random_state=42),
  model_RF = RandomForestClassifier(random_state=92),
  model_GB=GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,max_depth=1,
  random_state=0).fit(X_train, y_train)

5.5 Model evaluation



- Import accuracy_score, classification_report - to calcutate accuracy of model,import classification_report - to calculte precision, recall, f1-score import plot_confusion_matrix - to draw confusion_matrix
- multi_target_decision.predict(x_test)- Multi-output data contains more than one y label data for a given X input data
- accuracy_score(y_test, y_pred_model))- represents the model's ability to correctly predict both the positives and negatives out of all the predictions
- import GridSearchCV - to best select hyperparameter
- grid_search_DT=GridSearchCV (estimator=model_DT, param_grid=parameters,scoring="accuracy",n_jobs=-1)- All possible permutations of the hyper parameters for a particular model are used to build models. The performance of each model is evaluated and the best performing one is selected
-

IMPLEMENTATIONS:

CROP RECOMMENDATION:

1.Model Pre-processing:

```
In [5]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
```

```
In [6]: crop_data = pd.read_csv(r"C:\Users\kaviy\OneDrive\Desktop\CIP\DATASET\Crop_recommendation.csv")
        crop_data
```

Out[6]:

	N	P	K	temperature	humidity	ph	rainfall	label
0	90	42	43	20.879744	82.002744	6.502985	202.935536	rice
1	85	58	41	21.770462	80.319644	7.038096	226.655537	rice
2	60	55	44	23.004459	82.320763	7.840207	263.964248	rice
3	74	35	40	26.491096	80.158363	6.980401	242.864034	rice
4	78	42	42	20.130175	81.604873	7.628473	262.717340	rice
...	...	...	...	...	...	...	...	...
2195	107	34	32	26.774637	66.413269	6.780064	177.774507	coffee
2196	99	15	27	27.417112	56.636362	6.086922	127.924610	coffee
2197	118	33	30	24.131797	67.225123	6.362608	173.322839	coffee
2198	117	32	34	26.272418	52.127394	6.758793	127.175293	coffee
2199	104	18	30	23.603016	60.396475	6.779833	140.937041	coffee

Printing information about the crop dataset and dataset columns:

2200 rows × 8 columns

```
In [7]: crop_data.shape
```

```
Out[7]: (2200, 8)
```

```
In [8]: crop_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2200 entries, 0 to 2199
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   N            2200 non-null   int64  
 1   P            2200 non-null   int64  
 2   K            2200 non-null   int64  
 3   temperature  2200 non-null   float64
 4   humidity     2200 non-null   float64
 5   ph           2200 non-null   float64
 6   rainfall     2200 non-null   float64
 7   label        2200 non-null   object 
dtypes: float64(4), int64(3), object(1)
memory usage: 137.6+ KB
```

```
In [9]: # dataset columns
        crop_data.columns
```

Statistical interface of dataset:

```
Out[9]: Index(['N', 'P', 'K', 'temperature', 'humidity', 'ph', 'rainfall', 'label'], dtype='object')
```

```
In [10]: # statistical inference of the dataset

         crop_data.describe()
```

Out[10]:

	N	P	K	temperature	humidity	ph	rainfall
count	2200.000000	2200.000000	2200.000000	2200.000000	2200.000000	2200.000000	2200.000000
mean	50.551818	53.362727	48.149091	25.616244	71.481779	6.469480	103.463655
std	36.917334	32.985883	50.647931	5.063749	22.263812	0.773938	54.958389
min	0.000000	5.000000	5.000000	8.825675	14.258040	3.504752	20.211267
25%	21.000000	28.000000	20.000000	22.769375	60.261953	5.971693	64.551686
50%	37.000000	51.000000	32.000000	25.598693	80.473146	6.425045	94.867624
75%	84.250000	68.000000	49.000000	28.561654	89.948771	6.923643	124.267508
max	140.000000	145.000000	205.000000	43.675493	99.981876	9.935091	298.560117

Handling Missing Values:

Handling Missing Values in Columns:

In [11]:
```
# Checking missing values of the dataset in each column
crop_data.isnull().sum()
```

Out[11]:
```
N              0
P              0
K              0
temperature    0
humidity       0
ph             0
rainfall       0
label          0
dtype: int64
```

In [12]:
```
# Dropping missing values
crop_data = crop_data.dropna()
crop_data
```

Out[12]:

	N	P	K	temperature	humidity	ph	rainfall	label
0	90	42	43	20.879744	82.002744	6.502985	202.935536	rice
1	85	58	41	21.770462	80.319644	7.038096	226.655537	rice
2	60	55	44	23.004459	82.320763	7.840207	263.964248	rice
3	74	35	40	26.491096	80.158363	6.980401	242.864034	rice
4	78	42	42	20.130175	81.604873	7.628473	262.717340	rice

2195	107	34	32	26.774637	66.413269	6.780064	177.774507	coffee
2196	99	15	27	27.417112	56.636362	6.086922	127.924610	coffee
2197	118	33	30	24.131797	67.225123	6.362608	173.322839	coffee
2198	117	32	34	26.272418	52.127394	6.758793	127.175293	coffee
2199	104	18	30	23.603016	60.396475	6.779833	140.937041	coffee

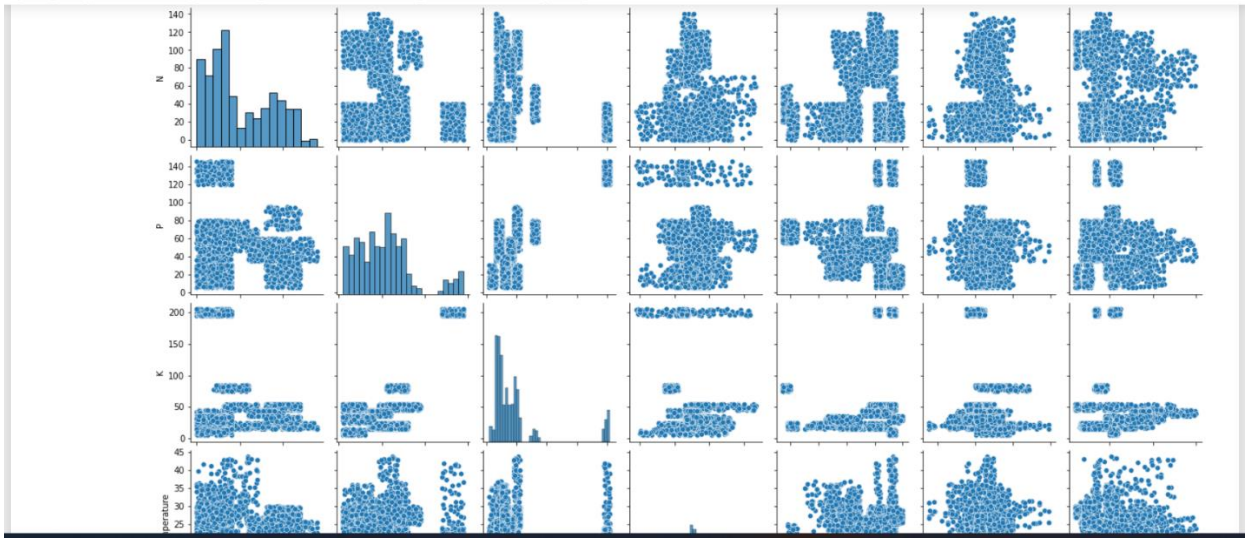
2200 rows × 8 columns

In [13]:
```
#checking
crop_data.isnull().values.any()
```

Out[13]:
```
False
```

In [14]:
```
# Visualizing the features

ax = sns.pairplot(crop_data)
ax
```

Visualizing the Features:



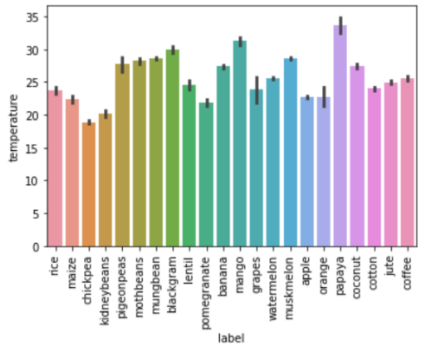
In [15]:
```
crop_data.label.unique()
```

Out[15]:
```
array(['rice', 'maize', 'chickpea', 'kidneybeans', 'pigeonpeas',
       'mothbeans', 'mungbean', 'blackgram', 'lentil', 'pomegranate',
       'banana', 'mango', 'grapes', 'watermelon', 'muskmelon', 'apple',
       'orange', 'papaya', 'coconut', 'cotton', 'jute', 'coffee'],
      dtype=object)
```

In [16]:
```
sns.barplot(crop_data["label"], crop_data["temperature"])
plt.xticks(rotation = 90)
```

```
C:\Users\kaviy\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword ar
gs: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit
keyword will result in an error or misinterpretation.
  warnings.warn(
```

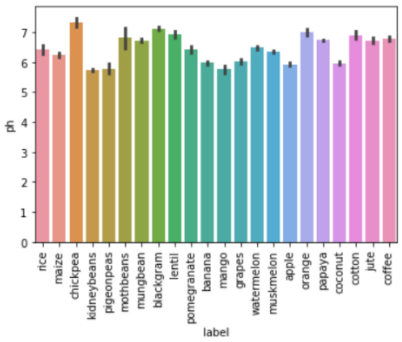
Out[16]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21]),
 [Text(0, 0, 'rice'),
  Text(1, 0, 'maize'),
  Text(2, 0, 'chickpea'),
  Text(3, 0, 'kidneybeans'),
  Text(4, 0, 'pigeonpeas'),
  Text(5, 0, 'mothbeans'),
  Text(6, 0, 'mungbean'),
  Text(7, 0, 'blackgram'),
  Text(8, 0, 'lentil'),
  Text(9, 0, 'pomegranate'),
  Text(10, 0, 'banana'),
  Text(11, 0, 'mango'),
  Text(12, 0, 'grapes'),
  Text(13, 0, 'watermelon'),
  Text(14, 0, 'muskmelon'),
  Text(15, 0, 'apple'),
  Text(16, 0, 'orange'),
  Text(17, 0, 'papaya'),
  Text(18, 0, 'coconut'),
  Text(19, 0, 'cotton'),
  Text(20, 0, 'jute'),
  Text(21, 0, 'coffee')])
```



In [17]:
```
sns.barplot(crop_data["label"], crop_data["ph"])
plt.xticks(rotation = 90)
```

```
C:\Users\kaviy\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword ar
gs: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit
keyword will result in an error or misinterpretation.
  warnings.warn(
```

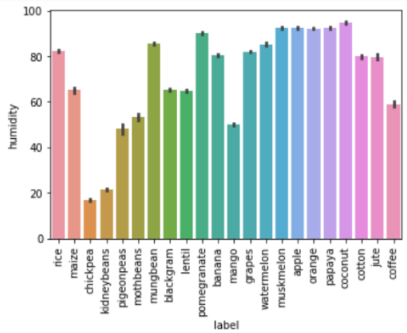
Out[17]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21]),
 [Text(0, 0, 'rice'),
  Text(1, 0, 'maize'),
  Text(2, 0, 'chickpea'),
  Text(3, 0, 'kidneybeans'),
  Text(4, 0, 'pigeonpeas'),
  Text(5, 0, 'mothbeans'),
  Text(6, 0, 'mungbean'),
  Text(7, 0, 'blackgram'),
  Text(8, 0, 'lentil'),
  Text(9, 0, 'pomegranate'),
  Text(10, 0, 'banana'),
  Text(11, 0, 'mango'),
  Text(12, 0, 'grapes'),
  Text(13, 0, 'watermelon'),
  Text(14, 0, 'muskmelon'),
  Text(15, 0, 'apple'),
  Text(16, 0, 'orange'),
  Text(17, 0, 'papaya'),
  Text(18, 0, 'coconut'),
  Text(19, 0, 'cotton'),
  Text(20, 0, 'jute'),
  Text(21, 0, 'coffee')])
```



In [18]:
```
sns.barplot(crop_data["label"], crop_data["humidity"])
plt.xticks(rotation = 90)
```

```
C:\Users\kaviy\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword ar
gs: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit
keyword will result in an error or misinterpretation.
  warnings.warn(
```

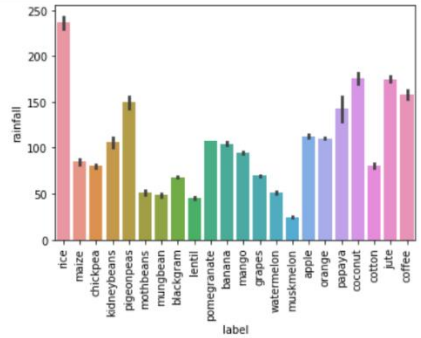
Out[18]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21]),
 [Text(0, 0, 'rice'),
  Text(1, 0, 'maize'),
  Text(2, 0, 'chickpea'),
  Text(3, 0, 'kidneybeans'),
  Text(4, 0, 'pigeonpeas'),
  Text(5, 0, 'mothbeans'),
  Text(6, 0, 'mungbean'),
  Text(7, 0, 'blackgram'),
  Text(8, 0, 'lentil'),
  Text(9, 0, 'pomegranate'),
  Text(10, 0, 'banana'),
  Text(11, 0, 'mango'),
  Text(12, 0, 'grapes'),
  Text(13, 0, 'watermelon'),
  Text(14, 0, 'muskmelon'),
  Text(15, 0, 'apple'),
  Text(16, 0, 'orange'),
  Text(17, 0, 'papaya'),
  Text(18, 0, 'coconut'),
  Text(19, 0, 'cotton'),
  Text(20, 0, 'jute'),
  Text(21, 0, 'coffee')])
```



In [19]:
```
sns.barplot(crop_data["label"], crop_data["rainfall"])
plt.xticks(rotation = 90)
```

```
C:\Users\kaviy\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword ar
gs: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit
keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[19]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21]),
 [Text(0, 0, 'rice'),
  Text(1, 0, 'maize'),
  Text(2, 0, 'chickpea'),
  Text(3, 0, 'kidneybeans'),
  Text(4, 0, 'pigeonpeas'),
  Text(5, 0, 'mothbeans'),
  Text(6, 0, 'mungbean'),
  Text(7, 0, 'blackgram'),
  Text(8, 0, 'lentil'),
  Text(9, 0, 'pomegranate'),
  Text(10, 0, 'banana'),
  Text(11, 0, 'mango'),
  Text(12, 0, 'grapes'),
  Text(13, 0, 'watermelon'),
  Text(14, 0, 'muskmelon'),
  Text(15, 0, 'apple'),
  Text(16, 0, 'orange'),
  Text(17, 0, 'papaya'),
  Text(18, 0, 'coconut'),
  Text(19, 0, 'cotton'),
  Text(20, 0, 'jute'),
  Text(21, 0, 'coffee')])
```

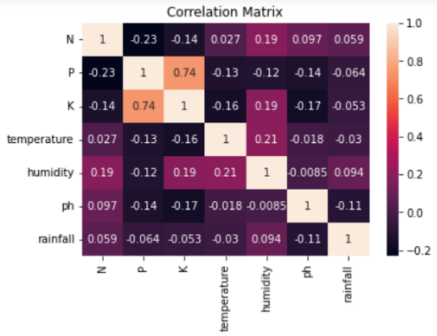


In [21]:
```
crop_data.corr()
```

	N	P	K	temperature	humidity	ph	rainfall
N	1.000000	-0.231460	-0.140512	0.026504	0.190688	0.096683	0.059020
P	-0.231460	1.000000	0.736232	-0.127541	-0.118734	-0.138019	-0.063839
K	-0.140512	0.736232	1.000000	-0.160387	0.190859	-0.169503	-0.053461
temperature	0.026504	-0.127541	-0.160387	1.000000	0.205320	-0.017795	-0.030084
humidity	0.190688	-0.118734	0.190859	0.205320	1.000000	-0.008483	0.094423
ph	0.096683	-0.138019	-0.169503	-0.017795	-0.008483	1.000000	-0.109069
rainfall	0.059020	-0.063839	-0.053461	-0.030084	0.094423	-0.109069	1.000000

In [22]:
```
sns.heatmap(crop_data.corr(), annot =True)
plt.title('Correlation Matrix')
```

Out[22]:
```
Text(0.5, 1.0, 'Correlation Matrix')
```



In [23]:
```
# Shuffling data to remove order effects

# shuffling the dataset to remove order
from sklearn.utils import shuffle

df  = shuffle(crop_data,random_state=5)
df.head()
```

Shuffling the dataset to remove order and Selection of Feature and Target variables:

Out[23]:

	N	P	K	temperature	humidity	ph	rainfall	label
1270	6	140	205	17.665584	82.929034	6.313086	69.867126	grapes
1481	98	22	47	29.072653	91.915332	6.341401	28.835684	muskmelon
1832	38	14	30	26.924495	91.201060	5.570745	194.902214	coconut
293	35	63	76	17.815645	17.607566	7.714153	90.820976	chickpea
1307	85	22	53	25.965342	89.770767	6.849472	59.463386	watermelon

In [24]:
```
# Selection of Feature and Target variables.

x = df[['N', 'P','K','temperature', 'humidity', 'ph', 'rainfall']]
target = df['label']
```

In [93]:
```
y = pd.get_dummies(target)
y
```

Out[93]:

	apple	banana	blackgram	chickpea	coconut	coffee	cotton	grapes	jute	kidneybeans	...	mango	mothbeans	mungbean	muskmelon	orange	pap
1270	0	0	0	0	0	0	0	1	0	0	...	0	0	0	0	0	
1481	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0	
1832	0	0	0	0	1	0	0	0	0	0	...	0	0	0	0	0	
293	0	0	0	1	0	0	0	0	0	0	...	0	0	0	0	0	

Training and Testing the dataset:

In [26]:
```
# Splitting data set - 25% test dataset and 75%


from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25, random_state= 0)

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

```
x_train : (1650, 7)
x_test : (550, 7)
y_train : (1650, 22)
y_test : (550, 22)
```

In [27]:
```
from sklearn.datasets import make_classification
from sklearn.multioutput import MultiOutputClassifier
from sklearn.ensemble import RandomForestClassifier
```

In [28]:
```
 #Training

forest = RandomForestClassifier(random_state=1)
multi_target_forest = MultiOutputClassifier(forest, n_jobs=-1)
multi_target_forest.fit(x_train, y_train)
```

Predicting test results:

```
Out[28]: MultiOutputClassifier(estimator=RandomForestClassifier(random_state=1),
                               n_jobs=-1)
```

```
In [29]: # Predicting test results

         forest_pred = multi_target_forest.predict(x_test)
         forest_pred
```

```
Out[29]: array([[0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 1, 0],
                [0, 0, 0, ..., 0, 0, 0],
                ...,
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 1, 0],
                [0, 0, 0, ..., 0, 0, 0]], dtype=uint8)
```

```
In [30]: from sklearn.metrics import accuracy_score
         a1 = accuracy_score(y_test, forest_pred)
         print('Accuracy score:', accuracy_score(y_test, forest_pred))
```

```
Accuracy score: 0.98
```

```
In [31]: from sklearn.model_selection import cross_val_score
         score = cross_val_score(multi_target_forest,X = x_train, y = y_train,cv=5)
         score
```

```
Out[31]: array([0.97575758, 0.96666667, 0.95454545, 0.96666667, 0.96969697])
```

Training:

```
In [32]: b1 = "{:.2f}".format(score.mean()*100)
         b1 = float(b1)
         b1
```

```
Out[32]: 96.67
```

```
In [33]: c1 = (score.std()*100)
         c1
```

```
Out[33]: 0.6910154091509904
```

```
In [34]: print("Accuracy : {:.2f}%".format (score.mean()*100))
         print("Standard Deviation : {:.2f}%".format(score.std()*100))
```

```
Accuracy : 96.67%
Standard Deviation : 0.69%
```

```
In [35]: # Training
         from sklearn.tree import DecisionTreeClassifier

         clf = DecisionTreeClassifier(random_state=6)
         multi_target_decision = MultiOutputClassifier(clf, n_jobs=-1)
         multi_target_decision.fit(x_train, y_train)
```

```
Out[35]: MultiOutputClassifier(estimator=DecisionTreeClassifier(random_state=6),
                               n_jobs=-1)
```

Predicting the test results and calculating the accuracy:

```
In [36]: # Predicting test results

         decision_pred = multi_target_decision.predict(x_test)
         decision_pred
```

```
Out[36]: array([[0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 1, 0],
                [0, 0, 0, ..., 0, 0, 0],
                ...,
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 1, 0],
                [0, 0, 0, ..., 0, 0, 0]], dtype=uint8)
```

```
In [37]: # Calculating Accuracy

         from sklearn.metrics import accuracy_score
         a2 = accuracy_score(y_test,decision_pred)
         print('Accuracy score:', accuracy_score(y_test,decision_pred))
         a2
```

```
Accuracy score: 0.9436363636363636
```

```
Out[37]: 0.9436363636363636
```

```
In [38]: from sklearn.model_selection import cross_val_score
         score = cross_val_score(multi_target_decision,X = x_train, y = y_train,cv=7)
         score
```

```
Out[38]: array([0.88135593, 0.91525424, 0.90677966, 0.93220339, 0.92372881,
                0.96595745, 0.94468085])

In [39]: b2 = "{:.2f}".format(score.mean()*100)
         b2 = float(b2)
         b2

Out[39]: 92.43

In [40]: c2 = (score.std()*100)
         c2

Out[40]: 2.5203429649690308

In [49]: from sklearn.neighbors import KNeighborsClassifier

         knn_clf=KNeighborsClassifier(3)
         model = MultiOutputClassifier(knn_clf, n_jobs=-1)
         knn_clf.fit(x_train, y_train)

Out[49]: KNeighborsClassifier(n_neighbors=3)

In [50]: knn_pred = knn_clf.predict(x_test)
         knn_pred.shape

Out[50]: (550, 22)
```

Calculating accuracy:

```
In [51]: # Calculating Accuracy

         from sklearn.metrics import accuracy_score
         a3 = accuracy_score(y_test,knn_pred)
         print('Accuracy score:', accuracy_score(y_test,knn_pred))
         a3

         Accuracy score: 0.98

Out[51]: 0.98

In [52]: from sklearn.model_selection import cross_val_score
         score = cross_val_score(model,X = x_train, y = y_train,cv=7)
         score

Out[52]: array([0.99152542, 0.97033898, 0.97033898, 0.97881356, 0.97457627,
                0.96170213, 0.98723404])

In [53]: b3 = "{:.2f}".format(score.mean()*100)
         b3 = float(b3)
         b3

Out[53]: 97.64

In [54]: c3 = (score.std()*100)
         c3

Out[54]: 0.9597581684278791
```

Initialise data and creating the dataset and printing it:

```
In [55]: import pandas as pd

         # initialise data of lists.
         data = {'Algorithms':['Random Forest', 'Decision-tree', 'KNN Classifier'],
                 'Accuracy':[b1, b2, b3],
                 'Standard Deviation':[c1,c2,c3]}

         # Creates pandas DataFrame.
         df = pd.DataFrame(data)

         # print the data
         df
```

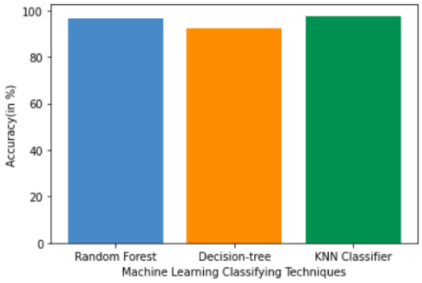
Out[55]:

	Algorithms	Accuracy	Standard Deviation
0	Random Forest	96.67	0.691015
1	Decision-tree	92.43	2.520343
2	KNN Classifier	97.64	0.959758

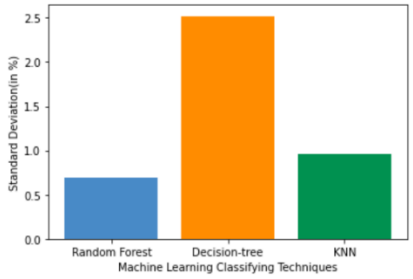
```
In [56]: import numpy as np
         import matplotlib.pyplot as plt

         # create a dataset
         Algorithms = ['Random Forest', 'Decision-tree','KNN Classifier']
         Accuracy = [b1, b2, b3]
```

Showing the graph:

```
# Create names on the x-axis
plt.xticks(x_pos, Algorithms)
plt.ylabel('Accuracy(in %)')
plt.xlabel('Machine Learning Classifying Techniques')

# Show graph
plt.show()
```



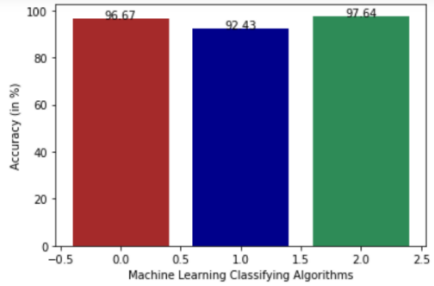
In [57]:
```
import numpy as np
import matplotlib.pyplot as plt

# create a dataset
Algorithms = ['Random Forest', 'Decision-tree','KNN']
Accuracy = [c1, c2, c3]

x_pos = np.arange(len(Accuracy))

# Create bars with different colors
plt.bar(x_pos, Accuracy, color= ['#488AC7','#ff8c00','#009150'])

# Create names on the x-axis
plt.xticks(x_pos, Algorithms)
plt.ylabel('Standard Deviation(in %)')
plt.xlabel('Machine Learning Classifying Techniques')

# Show graph
plt.show()
```



In [58]:
```
def addlabels(x,y):
    for i in range(len(x)):
        plt.text(i,y[i],y[i],ha = 'center')

if __name__ == '__main__':
    # creating data on which bar chart will be plot
    x = ["Random Forest", "Decision tree", "KNN"]
    y = [b1,b2,b3]
```

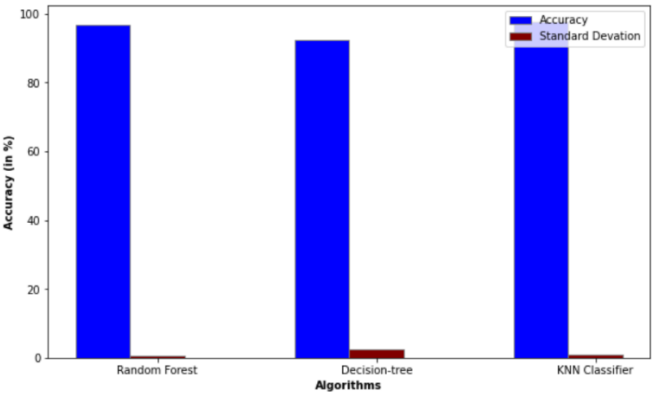
In [58]:
```
def addlabels(x,y):
    for i in range(len(x)):
        plt.text(i,y[i],y[i],ha = 'center')

if __name__ == '__main__':
    # creating data on which bar chart will be plot
    x = ["Random Forest", "Decision tree", "KNN"]
    y = [b1,b2,b3]


    x_pos = np.arange(len(y))

    # Create bars with different colors
    plt.bar(x_pos, y, color= ['#A52A2A','#00008B','#2E8B57'])

    # calling the function to add value labels
    addlabels(x, y)

    # giving X and Y labels
    plt.xlabel("Machine Learning Classifying Algorithms")
    plt.ylabel("Accuracy (in %)")

    # visualizing the plot
    plt.show()
```

Visualising the plots:



In [59]:
```
import numpy as np
import matplotlib.pyplot as plt

# set width of bar
barWidth = 0.25
fig = plt.subplots(figsize =(10, 6))

# set height of bar
Algorithms = ['Random Forest', 'Decision-tree', 'KNN Classifier']
Accuracy = [b1, b2, b3]
Standard_Deviation = [c1,c2,c3]
```

```
# set height of bar
Algorithms = ['Random Forest', 'Decision-tree', 'KNN Classifier']
Accuracy = [b1, b2, b3]
Standard_Deviation = [c1,c2,c3]

# Set position of bar on X axis
br1 = np.arange(len(Accuracy))
br2 = [x + barWidth for x in br1]
br3 = [x + barWidth for x in br2]

# Make the plot
plt.bar(br1, Accuracy, color ='blue', width = barWidth,
        edgecolor ='grey', label ='Accuracy')
plt.bar(br2, Standard_Deviation, color ='maroon', width = barWidth,
        edgecolor ='grey', label ='Standard Devation')

# Adding Xticks
plt.xlabel('Algorithms', fontweight ='bold', fontsize = 10)
plt.ylabel('Accuracy (in %)', fontweight ='bold', fontsize = 10)
plt.xticks([r + barWidth for r in range(len(Accuracy))],
        Algorithms)

plt.legend()
plt.show()
```



```
In [78]: from sklearn.neighbors import KNeighborsClassifier

         knn_clf=KNeighborsClassifier()
         #model = MultiOutputClassifier(knn_clf, n_jobs=-1)
         knn_clf.fit(x_train, y_train)

Out[78]: KNeighborsClassifier()

In [79]: y_pred=model.predict(x_test)
         print(y_pred.shape)

         (550, 22)

In [80]: import pickle

In [81]: # save the knn_model to disk
         filename = 'model.sav'
         pickle.dump(model, open(filename, 'wb'))

In [82]: knn_pred = model.predict(x_test)

In [83]: # load the model from disk
         filename = 'model.sav'
         knn_model_reloaded = pickle.load(open(filename, 'rb'))
```

```
knn_model_reloaded.predict(x_test)

print(x_test)
```

```
        N   P   K  temperature   humidity        ph    rainfall
486    39  77  21    22.997744  60.242186  4.603563  159.689339
42     83  60  36    25.597049  80.145093  6.903986  200.834898
1916  135  43  16    23.479869  81.730491  6.720450   86.762879
1496   82  26  47    28.504164  93.468065  6.565313   24.200072
1482  117  25  54    28.682760  92.509693  6.150686   29.111877
...   ...  ..  ..          ...        ...       ...         ...
1126   38  19  31    34.738239  49.088643  5.855119   90.650222
305    26  65  22    17.848066  18.776220  5.949949  143.098417
1498   90  15  52    27.049275  91.382173  6.448062   23.657475
55     75  38  39    23.446768  84.793524  6.215110  283.933847
1705   70  68  45    33.835086  92.854702  6.991626  203.404403

[550 rows x 7 columns]
```

```
In [84]: KNN_predict = knn_model_reloaded.predict(x_test) #Predictions on Testing data
         print(KNN_predict.shape)

         (550, 22)
```

```
In [85]: #####!!!!!
         x_train.shape
         from sklearn.neighbors import KNeighborsClassifier
         neigh = KNeighborsClassifier(n_neighbors=3)
         neigh.fit(x_train, y_train)
         y_pred=neigh.predict(x_test)

In [86]: y_pred.shape

Out[86]: (550, 22)

In [87]: print(x_train.shape)
         print(y_train.shape)

         (1650, 7)
         (1650, 22)

In [88]: print(y_pred)

         [[0 0 0 ... 0 0 0]
          [0 0 0 ... 0 0 0]
          [0 0 0 ... 0 0 0]
          ...
          [0 0 0 ... 0 0 0]
          [0 0 0 ... 0 1 0]
          [0 0 0 ... 0 0 0]]
```

Printing the predicted crop as array:

```
In [95]: #data = array([[28, 96, 17, 6,77, 4, 4]])
import pandas as pd
df=pd.DataFrame(crop_data)
df.pop('label')
i = df.iloc[45,] #- gets 45th row from dataframe
#i = get_crop('Pulses')
#i = i.drop(['Fertilizer '])
prediction = knn_clf.predict(np.array(i).reshape(1,-1))
#y=pd.get_dummies(np.array(prediction))

#print(y)
print(prediction)
```

```
[[0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0]]
```

Out[95]:

	apple	banana	blackgram	chickpea	coconut	coffee	cotton	grapes	jute	kidneybeans	...	mango	mothbeans	mungbean	muskmelon	orange	papay

Out[95]:

	apple	banana	blackgram	chickpea	coconut	coffee	cotton	grapes	jute	kidneybeans	...	mango	mothbeans	mungbean	muskmelon	orange	papay
1270	0	0	0	0	0	0	0	1	0	0	...	0	0	0	0	0	
1481	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0	
1832	0	0	0	0	1	0	0	0	0	0	...	0	0	0	0	0	
293	0	0	0	1	0	0	0	0	0	0	...	0	0	0	0	0	
1307	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
740	0	0	1	0	0	0	0	0	0	0	...	0	0	0	0	0	
1032	0	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
2121	0	0	0	0	0	1	0	0	0	0	...	0	0	0	0	0	
1424	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0	
1725	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	

2200 rows × 22 columns

FERTILIZER RECOMMENDATION:

Loading the dataset:

```
In [103]: # Importing libraries and packages for basic statistics
import os # To change working directory
import pandas as pd # to read and manipulating data
import numpy as np # to calculate mean and standard deviations

pd.options.display.max_columns = 100
pd.options.display.max_rows = 100

import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=UserWarning)
```

```
In [104]: #Load dataset to start EDA
#Changing working directory
os.chdir(r'C:\Users\kaviy\OneDrive\Desktop\CIP\DATASET')

# To read 'csv' file with panda library
df = pd.read_csv('Fertilizer Prediction.csv')
```

```
In [105]: # To display the first 10 rows of dataset
display(df.head(10))
```

Displaying the dataset and printing the column names:

	Temparature	Humidity	Moisture	Soil Type	Crop Type	Nitrogen	Potassium	Phosphorous	Fertilizer
0	26	52	38	alluvial	Maize	37	0	0	Urea
1	29	52	45	Loamy	Sugarcane	12	0	36	DAP
2	34	65	62	Black	Cotton	7	9	30	14-35-14
3	32	62	34	Red	Tobacco	22	0	20	28-28
4	28	54	46	Clayey	Paddy	35	0	0	Urea
5	26	52	35	alluvial	Barley	12	10	13	17-17-17
6	25	50	64	Red	Cotton	9	0	10	20-20
7	33	64	50	Loamy	Wheat	41	0	0	Urea
8	30	60	42	alluvial	Millets	21	0	18	28-28
9	29	58	33	Black	Oil seeds	9	7	30	14-35-14

In [106]:
```
# To find Column name
df.columns
```

Out[106]:
```
Index(['Temparature', 'Humidity ', 'Moisture', 'Soil Type', 'Crop Type',
       'Nitrogen', 'Potassium', 'Phosphorous', 'Fertilizer '],
      dtype='object')
```

Printing the information:

In [107]:
```
# To find the number of rows and columns
print(df.shape)

# check for the data types, memory usage, etc
display(df.info())
```

```
(99, 9)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99 entries, 0 to 98
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   Temparature  99 non-null     int64 
 1   Humidity     99 non-null     int64 
 2   Moisture     99 non-null     int64 
 3   Soil Type    99 non-null     object
 4   Crop Type    99 non-null     object
 5   Nitrogen     99 non-null     int64 
 6   Potassium    99 non-null     int64 
 7   Phosphorous  99 non-null     int64 
 8   Fertilizer   99 non-null     object
dtypes: int64(6), object(3)
memory usage: 7.1+ KB

None
```

In [108]:
```
# checking the no. of missing values in the dataset
df.isnull().sum()
```

Checking the missing values:

Out[108]:
```
Temparature    0
Humidity       0
Moisture       0
Soil Type      0
Crop Type      0
Nitrogen       0
Potassium      0
Phosphorous    0
Fertilizer     0
dtype: int64
```

In [109]:
```
df.isnull().values.any()
```

Out[109]:
```
False
```

In [110]:
```
# statistics of the numerical variables
display(df.describe().T)
```

	count	mean	std	min	25%	50%	75%	max
Temparature	99.0	30.282828	3.502304	25.0	28.0	30.0	33.0	38.0
Humidity	99.0	59.151515	5.840331	50.0	54.0	60.0	64.0	72.0
Moisture	99.0	43.181818	11.271568	25.0	34.0	41.0	50.5	65.0
Nitrogen	99.0	18.909091	11.599693	4.0	10.0	13.0	24.0	42.0

Potassium	99.0	3.383838	5.814667	0.0	0.0	0.0	7.5	19.0
Phosphorous	99.0	18.606061	13.476978	0.0	9.0	19.0	30.0	42.0

In [111]:
```
# statistics of the category variables
display(df.describe(include='object'))
```

	Soil Type	Crop Type	Fertilizer
count	99	99	99
unique	5	11	7
top	Loamy	Sugarcane	Urea
freq	21	13	22

In [112]:
```
import matplotlib.pyplot as plt # to visualize graph
%matplotlib inline
import seaborn as sns # for better visualization of graph with the help of Matplotlib
%pip install dython
from dython import nominal # to find out correlation and visualize it
```

```
Requirement already satisfied: dython in c:\users\kaviy\anaconda3\lib\site-packages (0.7.1.post3)
Requirement already satisfied: scipy>=1.7.1 in c:\users\kaviy\anaconda3\lib\site-packages (from dython) (1.7.1)
Requirement already satisfied: matplotlib>=3.4.3 in c:\users\kaviy\anaconda3\lib\site-packages (from dython) (3.4.3)
Requirement already satisfied: seaborn>=0.11.0 in c:\users\kaviy\anaconda3\lib\site-packages (from dython) (0.11.2)
Requirement already satisfied: scikit-plot>=0.3.7 in c:\users\kaviy\anaconda3\lib\site-packages (from dython) (0.3.7)
Requirement already satisfied: numpy>=1.19.5 in c:\users\kaviy\anaconda3\lib\site-packages (from dython) (1.20.3)
```

Visualising the graph:

In [113]:
```
# print the unique class of the Fertilizer
print(df['Fertilizer '].unique())
```

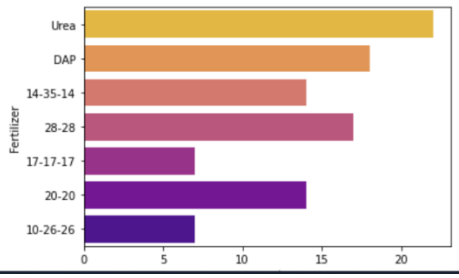
```
['Urea' 'DAP' '14-35-14' '28-28' '17-17-17' '20-20' '10-26-26']
```

In [114]:
```
print("Soil : ", df['Soil Type'].unique())
```

```
Soil :  ['alluvial' 'Loamy' 'Black' 'Red' 'Clayey']
```

In [115]:
```
#Visualization of the class in Fertilizer category with countplot
sns.countplot(y='Fertilizer ',data=df,palette="plasma_r")
```

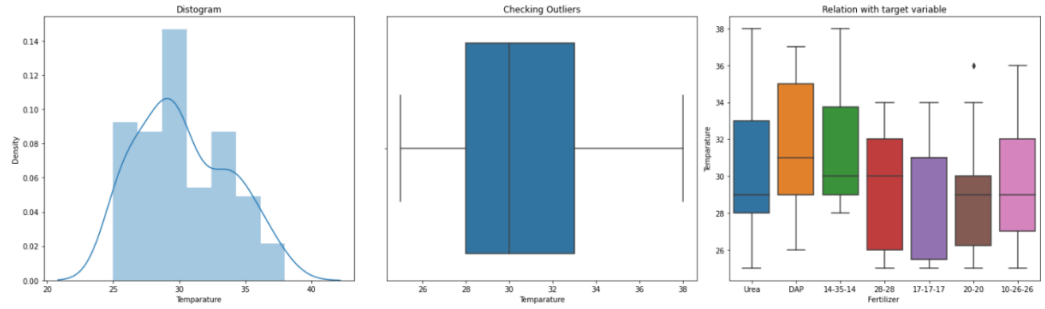
Out[115]: <AxesSubplot:xlabel='count', ylabel='Fertilizer '>



Relationship with target variables:

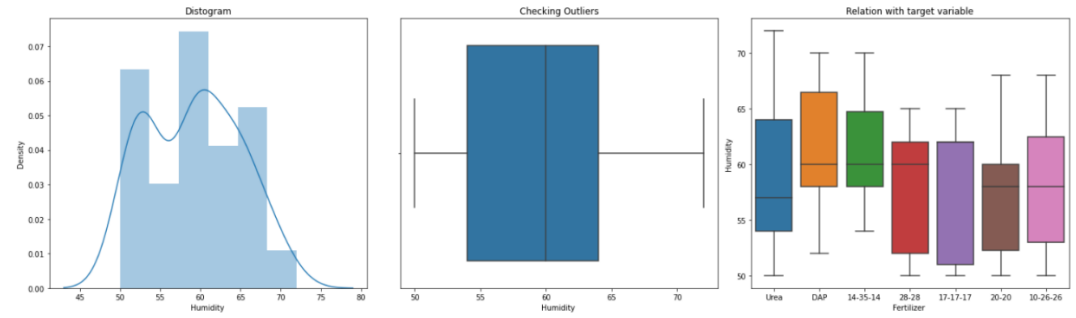
In [116]:
```
# Defining function for Continuous variable and their relationship with target variable
def plot_conti(x):
    fig, axes = plt.subplots(nrows=1,ncols=3,figsize=(20,6),tight_layout=True)
    axes[0].set_title('Distogram')
    sns.distplot(x,ax=axes[0])
    axes[1].set_title('Checking Outliers')
    sns.boxplot(x,ax=axes[1])
    axes[2].set_title('Relation with target variable')
    sns.boxplot(y = x,x = df['Fertilizer '])
```

In [117]:
```
# EDA - Temparature variable
plot_conti(df['Temparature'])
```

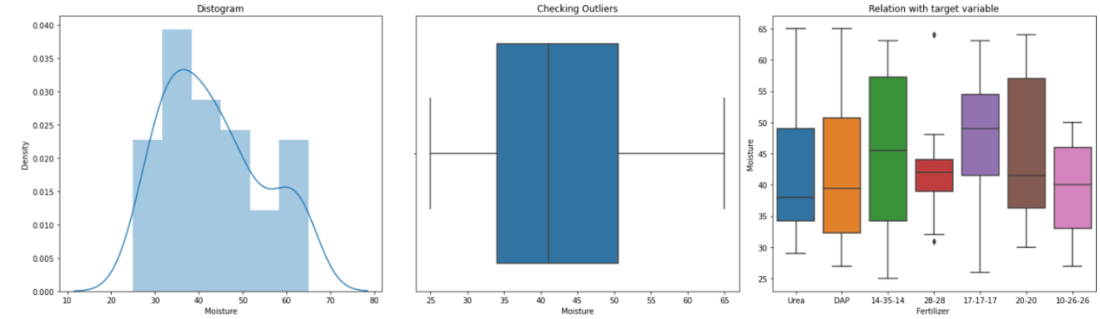


In [118]:
```
#EDA - Humidity variable
plot_conti(df['Humidity '])
```

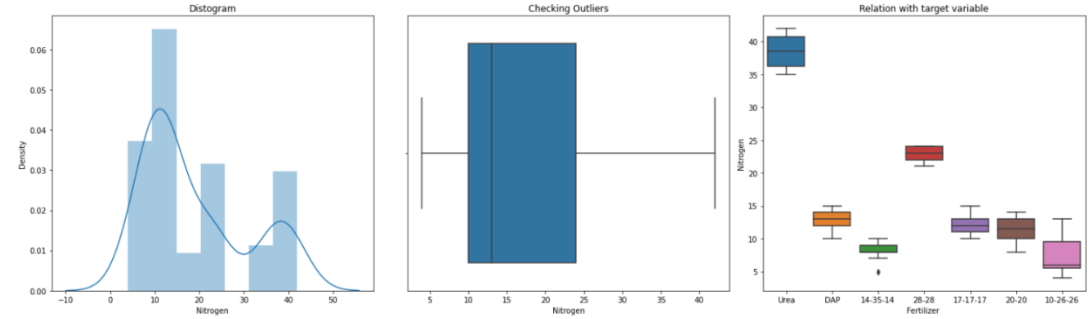
In [118]:
#EDA - Humidity variable
plot_conti(df['Humidity '])



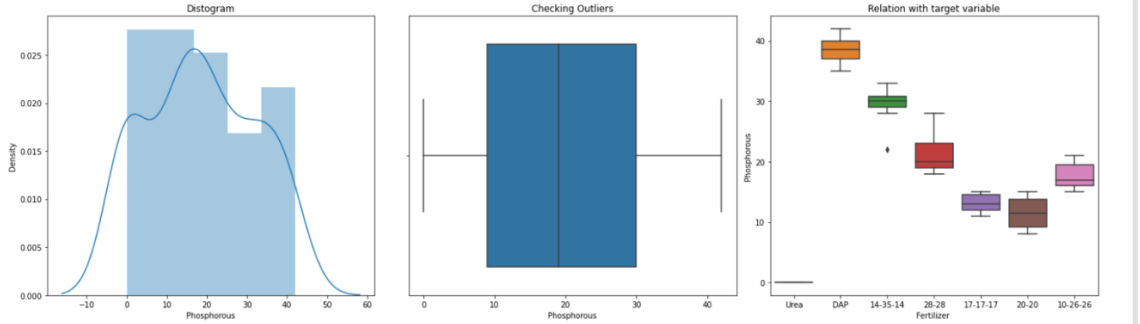
In [119]:
```
#EDA - Rainfall variable
plot_conti(df['Moisture'])
```



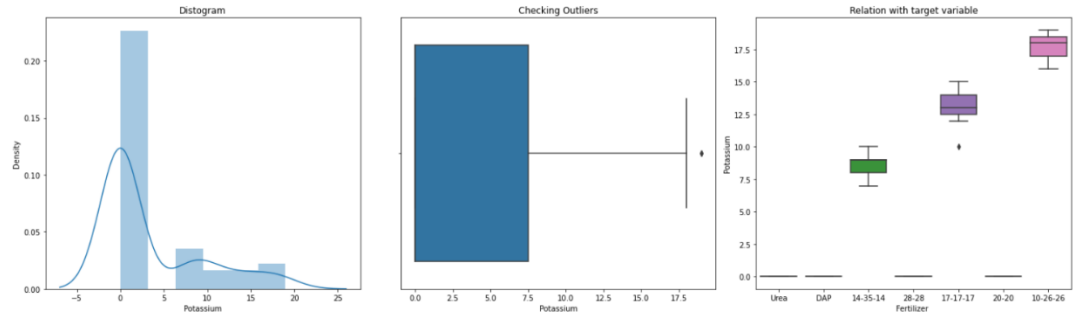
In [120]:
```
plot_conti(df['Nitrogen'])
```



In [121]:
```
# EDA - Phosphorous variable
plot_conti(df['Phosphorous'])
```



In [122]:
```
plot_conti(df['Potassium'])
```

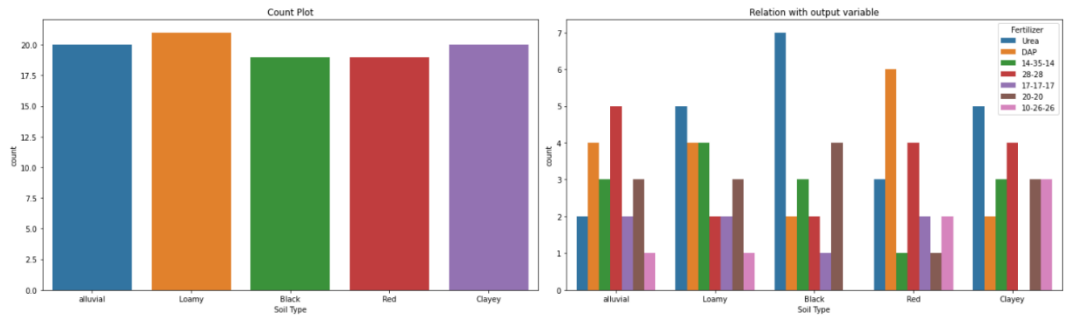


In [123]:
```
# Defining function to visualize catogorical variable and their relationship with target variable
def plot_cato(x):
    fig, axes = plt.subplots(nrows=1,ncols=2,figsize=(20,6),tight_layout=True)
    axes[0].set_title('Count Plot')
    sns.countplot(x,ax=axes[0])
    axes[1].set_title('Relation with output variable')
    sns.countplot(x = x,hue = df['Fertilizer '], ax=axes[1])
```

In [124]:
```
# print the unique types of the Soil
print("Soil Type : ", df['Soil Type'].unique())
```

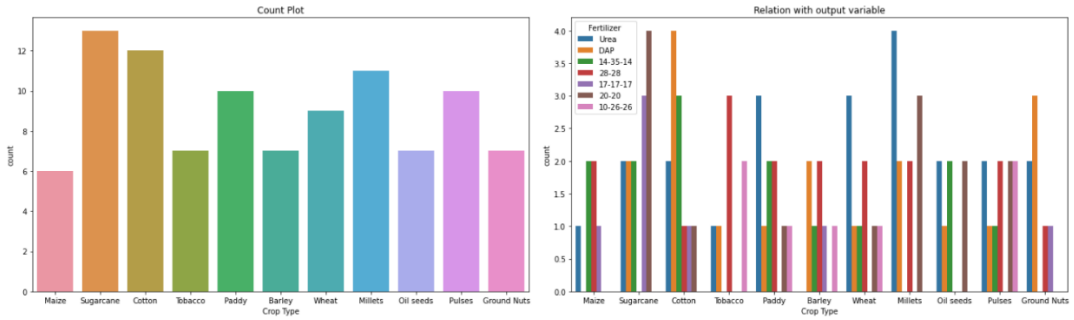
```
Soil Type :  ['alluvial' 'Loamy' 'Black' 'Red' 'Clayey']
```

In [125]:
```
plot_cato(df['Soil Type'])
```



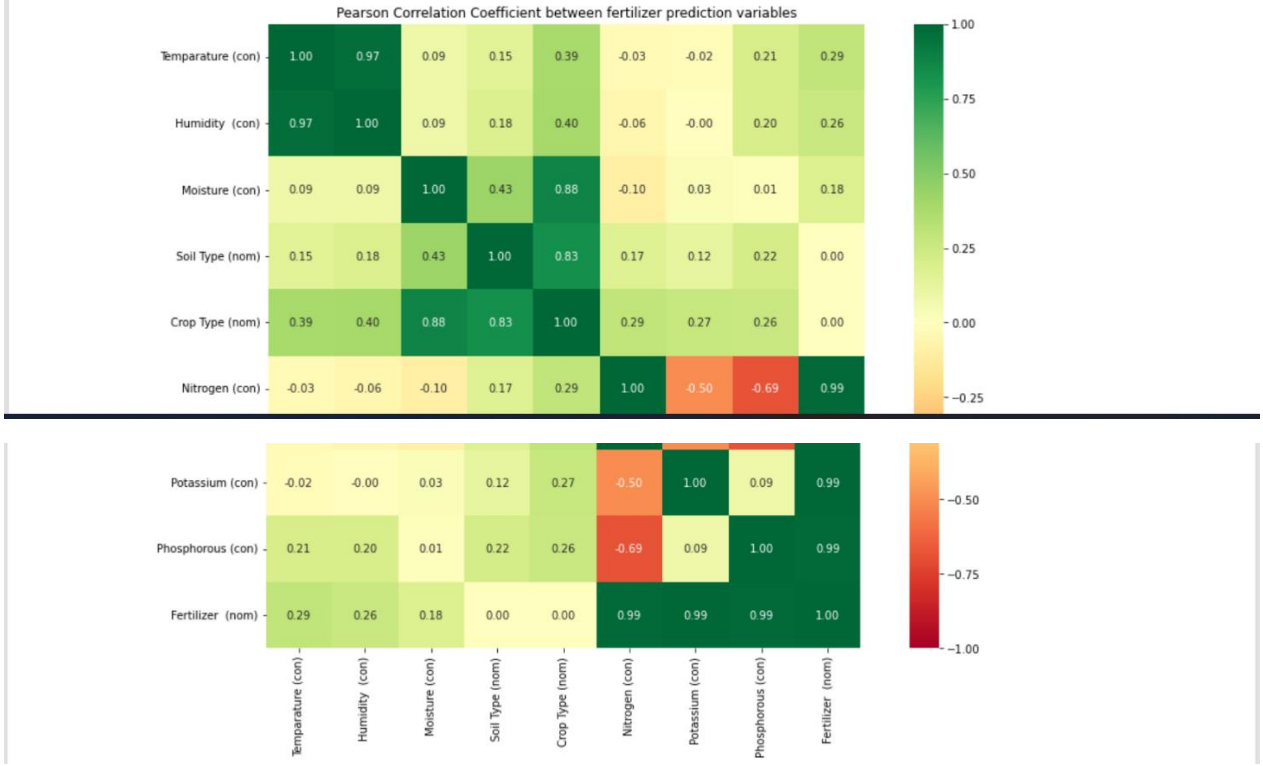
Printing unique types of crops:

```
In [126]: # print the unique types of the Crop
          print("Crop: ", df['Crop Type'].unique())

Crop:  ['Maize' 'Sugarcane' 'Cotton' 'Tobacco' 'Paddy' 'Barley' 'Wheat' 'Millets'
 'Oil seeds' 'Pulses' 'Ground Nuts']

In [127]: plot_cato(df['Crop Type'])
```



Correlation matrix:

```
In [128]: #To find out correlation with both nominal and numeric variables
          nominal.associations(df,figsize=(16,10),
                               mark_columns=True,
                               title="Pearson Correlation Coefficient between fertilizer prediction variables",
                               cmap='RdYlGn')
```



Out[128]:
```
{'corr':                    Temparature (con)  Humidity  (con)  Moisture (con)  \
 Temparature (con)           1.000000         0.973164        0.091222
 Humidity  (con)             0.973164         1.000000        0.091342
 Moisture (con)              0.091222         0.091342        1.000000
 Soil Type (nom)             0.154873         0.182937        0.426168
 Crop Type (nom)             0.385183         0.397787        0.880808
 Nitrogen (con)             -0.033771        -0.060646       -0.095945
 Potassium (con)            -0.023424        -0.003833        0.027727
 Phosphorous (con)           0.207545         0.204044        0.009276
 Fertilizer  (nom)           0.290043         0.258220        0.179475

                     Soil Type (nom)  Crop Type (nom)  Nitrogen (con)  \
 Temparature (con)          0.154873         0.385183       -0.033771
 Humidity  (con)            0.182937         0.397787       -0.060646
 Moisture (con)             0.426168         0.880808       -0.095945
 Soil Type (nom)            1.000000         0.831388        0.168916
 Crop Type (nom)            0.831388         1.000000        0.294960
 Nitrogen (con)             0.168916         0.294960        1.000000
 Potassium (con)            0.118855         0.268462       -0.500087
 Phosphorous (con)          0.215567         0.261762       -0.686971
 Fertilizer  (nom)          0.000000         0.000000        0.987361

                     Potassium (con)  Phosphorous (con)  Fertilizer  (nom)
 Temparature (con)         -0.023424           0.207545           0.290043
 Humidity  (con)           -0.003833           0.204044           0.258220
 Moisture (con)             0.027727           0.009276           0.179475
 Soil Type (nom)            0.118855           0.215567           0.000000
 Crop Type (nom)            0.268462           0.261762           0.000000
```

In [129]:
```
from sklearn.preprocessing import MinMaxScaler # to normalize data
from sklearn.preprocessing import LabelEncoder # to encode object variable to numeric
from sklearn.model_selection import train_test_split # to split data into training and testing sets
```

In [130]:
```
X = df.drop(['Fertilizer '], axis=1) #feature variables
y = df[['Fertilizer ']] #Target variable
print('The shape of feature set, X is ' , X.shape)
print('The shape of target, y is ' , y.shape)
```

```
The shape of feature set, X is  (99, 8)
The shape of target, y is  (99, 1)
```

In [131]:
```
#Label Encoding
le = LabelEncoder()
df['Fertilizer ']= le.fit_transform(df['Fertilizer '])
df['Soil Type']= le.fit_transform(df['Soil Type'])
df['Crop Type']= le.fit_transform(df['Crop Type'])
```

In [132]:
```
display(df.head())
```

	Temparature	Humidity	Moisture	Soil Type	Crop Type	Nitrogen	Potassium	Phosphorous	Fertilizer
0	26	52	38	4	3	37	0	0	6
1	29	52	45	2	8	12	0	36	5
2	34	65	62	0	1	7	9	30	1
3	32	62	34	3	9	22	0	20	4
4	28	54	46	1	6	35	0	0	6

In [77]:
```
#Label Encoding
le = LabelEncoder()
#df['Fertilizer ']= le.fit_transform(df['Fertilizer '])
df['Soil Type']= le.fit_transform(df['Soil Type'])
df['Crop Type']= le.fit_transform(df['Crop Type'])
```

In [78]:
```
display(df.head())
```

	Temparature	Humidity	Moisture	Soil Type	Crop Type	Nitrogen	Potassium	Phosphorous	Fertilizer
0	26	52	38	4	3	37	0	0	6
1	29	52	45	2	8	12	0	36	5
2	34	65	62	0	1	7	9	30	1
3	32	62	34	3	9	22	0	20	4
4	28	54	46	1	6	35	0	0	6

In [133]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(df)
scaled = scaler.fit_transform(df)
scaled_df = pd.DataFrame(scaled, columns=df.columns)
print(scaled_df)
```

```
    Temparature  Humidity   Moisture  Soil Type  Crop Type  Nitrogen  \
0      0.076923   0.090909     0.325       1.00        0.3  0.868421
1      0.307692   0.090909     0.500       0.50        0.8  0.210526
2      0.692308   0.681818     0.925       0.00        0.1  0.078947
3      0.538462   0.545455     0.225       0.75        0.9  0.473684
4      0.230769   0.181818     0.525       0.25        0.6  0.815789
5      0.076923   0.090909     0.250       1.00        0.0  0.210526
6      0.000000   0.000000     0.975       0.75        0.1  0.131579
7      0.615385   0.636364     0.625       0.50        1.0  0.973684
```

In [134]:
```
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.3,
                                                    random_state=42,
                                                    stratify=y)

print('Shape of X_train is', X_train.shape)
print('Shape of X_test is', X_test.shape)
print('Shape of y_train is', y_train.shape)
print('Shape of y_test is',  y_test.shape)
```

```
Shape of X_train is (69, 8)
Shape of X_test is (30, 8)
Shape of y_train is (69, 1)
Shape of y_test is (30, 1)
```

In [135]:
```
X = df[df.columns[:-1]]
Y = df['Fertilizer ']
X_train, X_test, y_train, y_test =train_test_split(X,
                                                   y,
                                                   test_size=0.33,
                                                   random_state=42,
                                                   stratify=y)
print('Shape of X_train is', X_train.shape)
print('Shape of X_test is', X_test.shape)
print('Shape of y_train is', y_train.shape)
print('Shape of y_test is',  y_test.shape)
```

Importing libraries for classification:

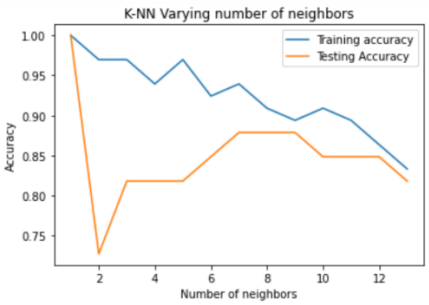
```
Shape of X_train is (66, 8)
Shape of X_test is (33, 8)
Shape of y_train is (66, 1)
Shape of y_test is (33, 1)
```

In [136]:
```
# Importing libraries for classification and performance evaluation
from sklearn.neighbors import KNeighborsClassifier #to build KNeighbors model
from sklearn.model_selection import GridSearchCV # to best select hyperparameter

from sklearn.metrics import accuracy_score, classification_report # to calcutate accuracy of model
from sklearn.metrics import classification_report #to calculte precision, recall, f1-score
from sklearn.metrics import plot_confusion_matrix # to draw confusion_matrix
```

In [137]:
```
# To determine the k value with highest accuracy
neighbors = np.arange(1,14)
train_accuracy = np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))

for i,k in enumerate(neighbors):
        knn = KNeighborsClassifier(n_neighbors=k)
        knn.fit(X_train, y_train)
        train_accuracy[i] = knn.score(X_train, y_train)
        test_accuracy[i] = knn.score(X_test, y_test)
plt.title('K-NN Varying number of neighbors')
plt.plot(neighbors, train_accuracy, label='Training accuracy')
plt.plot(neighbors, test_accuracy, label='Testing Accuracy')
plt.legend()
plt.xlabel('Number of neighbors')
```

In [137]:
```
# To determine the k value with highest accuracy
neighbors = np.arange(1,14)
train_accuracy = np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))

for i,k in enumerate(neighbors):
        knn = KNeighborsClassifier(n_neighbors=k)
        knn.fit(X_train, y_train)
        train_accuracy[i] = knn.score(X_train, y_train)
        test_accuracy[i] = knn.score(X_test, y_test)
plt.title('K-NN Varying number of neighbors')
plt.plot(neighbors, train_accuracy, label='Training accuracy')
plt.plot(neighbors, test_accuracy, label='Testing Accuracy')
plt.legend()
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy')
plt.show()
```

Predict the response for test dataset:



In [138]:
```
# Create K-Nearest Neighbors Classifier
model_knn = KNeighborsClassifier(n_neighbors=3)

# Train the model using the training sets
model_knn = model_knn.fit(X_train, y_train)
```

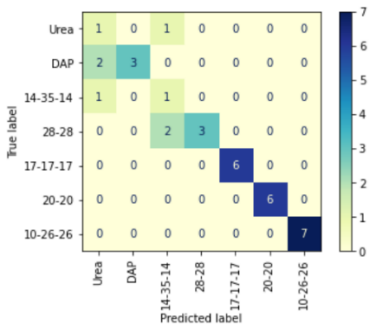
In [139]:
```
#Predict the response for test dataset
y_pred_knn = model_knn.predict(X_test)
```

Visualise the confusion matrix:

In [140]:
```
# Model Accuracy, how often is the classifier correct?
print('Accuracy of K-Nearest Neighbor Model: ', accuracy_score(y_test, y_pred_knn))
```

```
Accuracy of K-Nearest Neighbor Model:  0.8181818181818182
```

In [141]:
```
## To visualize confusion matrix
plot_confusion_matrix(model_knn,
                      X_test,
                      y_test,
                      display_labels=['Urea' ,'DAP', '14-35-14', '28-28', '17-17-17', '20-20', '10-26-26'],
                      xticks_rotation ='vertical',
                      cmap='YlGnBu')
plt.show()
```



Classification report:

In [142]:
```
#Classification report
print(classification_report(y_test,y_pred_knn))
```

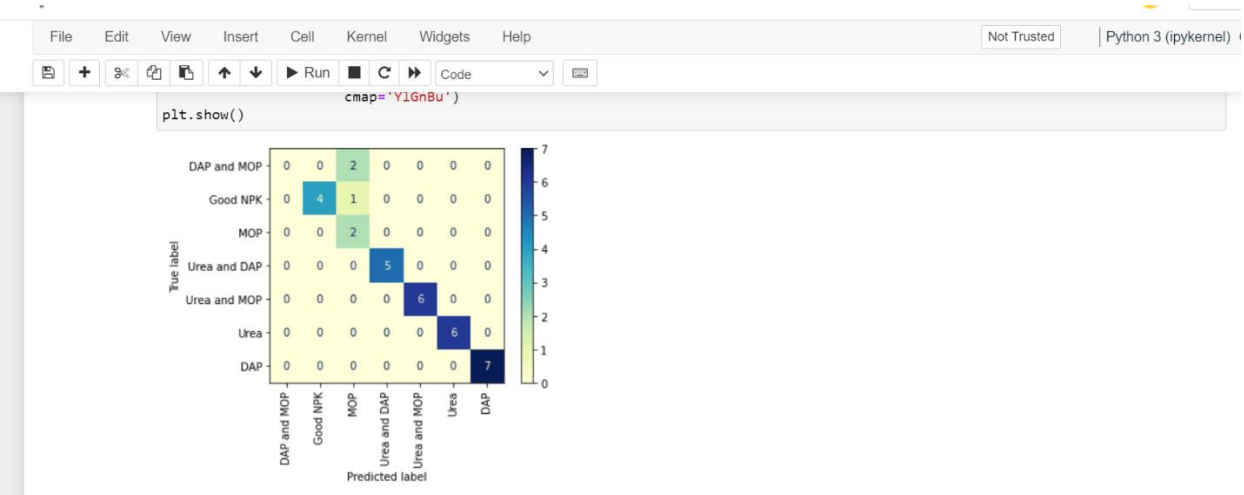
```
              precision    recall  f1-score   support

    10-26-26       0.25      0.50      0.33         2
    14-35-14       1.00      0.60      0.75         5
    17-17-17       0.25      0.50      0.33         2
       20-20       1.00      0.60      0.75         5
       28-28       1.00      1.00      1.00         6
         DAP       1.00      1.00      1.00         6
        Urea       1.00      1.00      1.00         7

    accuracy                           0.82        33
   macro avg       0.79      0.74      0.74        33
weighted avg       0.91      0.82      0.84        33
```

In [143]:
```
from sklearn.ensemble import GradientBoostingClassifier #to build GradientBoosting model
from sklearn.ensemble import RandomForestClassifier #to build RandomForest model
from sklearn.tree import DecisionTreeClassifier #to build a classification tree
from sklearn.tree import plot_tree # to draw a classification tree
from sklearn.model_selection import GridSearchCV # to best select hyperparameter
```

```
In [144]: #Decision Tree model
          #random state (int): Controls the randomness of the estimator for reproducibility
          model_DT = DecisionTreeClassifier(random_state=42)

          # Train the model using the training sets
          model_DT = model_DT.fit(X_train, y_train)

          #Predict the response for test dataset
          y_pred_DT = model_DT.predict(X_test)
```

```
In [145]: # To visualize confusion matrix
          plot_confusion_matrix(model_DT,
                                X_test,
                                y_test,
                                display_labels=['DAP and MOP','Good NPK','MOP','Urea and DAP','Urea and MOP','Urea','DAP'],
                                xticks_rotation ='vertical',
                                cmap='YlGnBu')
          plt.show()
```

Confusion matrix:



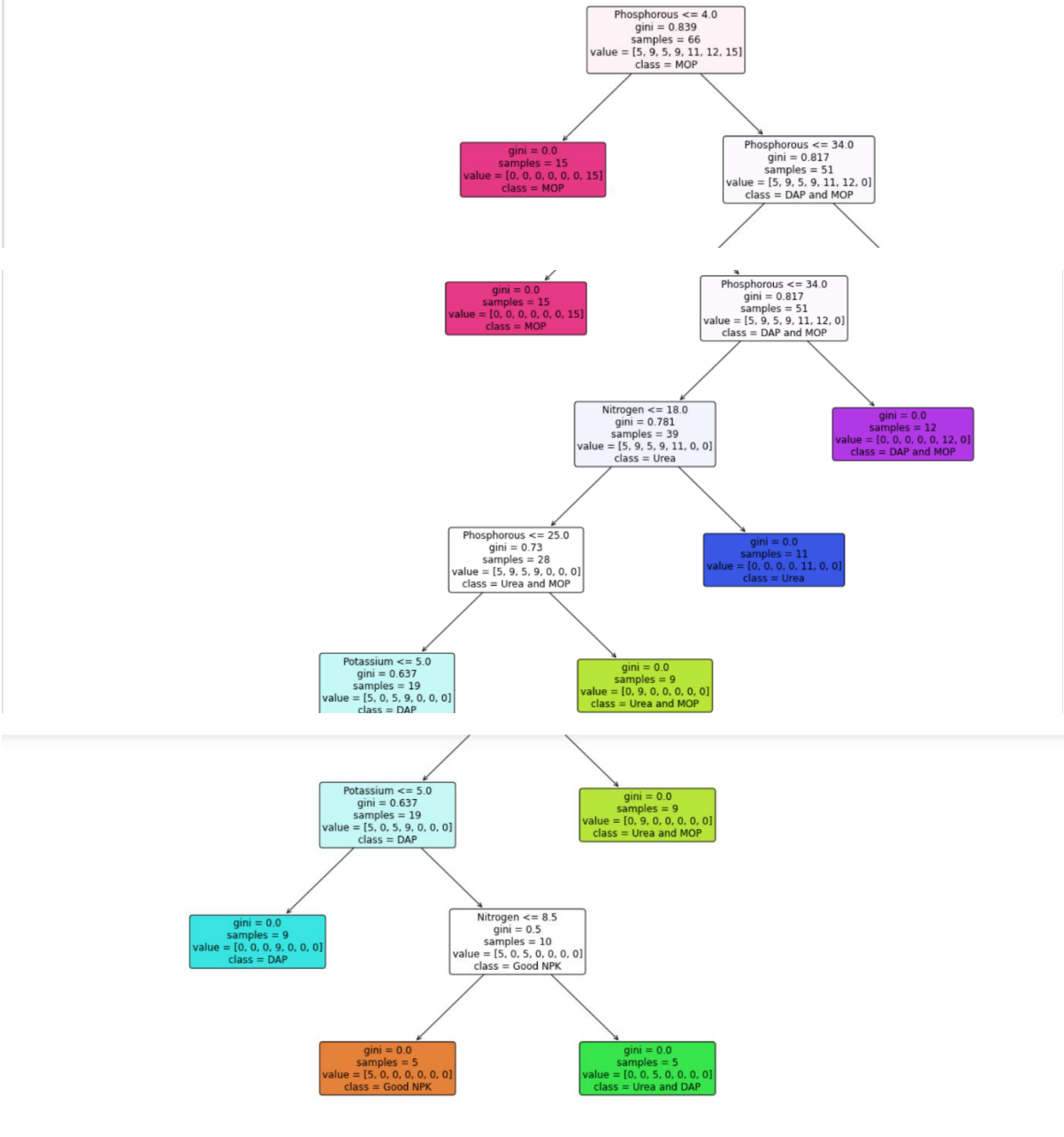
Accuracy of model:

```
In [146]: # Model Accuracy, how often is the classifier correct?
          print('Accuracy: ', accuracy_score(y_test, y_pred_DT))
          #Classification report
          print(classification_report(y_test, y_pred_DT))
```

```
Accuracy:  0.9090909090909091
              precision    recall  f1-score   support

    10-26-26       0.00      0.00      0.00         2
    14-35-14       1.00      0.80      0.89         5
    17-17-17       0.40      1.00      0.57         2
       20-20       1.00      1.00      1.00         5
       28-28       1.00      1.00      1.00         6
         DAP       1.00      1.00      1.00         6
        Urea       1.00      1.00      1.00         7

    accuracy                           0.91        33
   macro avg       0.77      0.83      0.78        33
weighted avg       0.90      0.91      0.90        33
```

Visualising decision tree:

In [147]:
```
# Visualizing Decision Tree
plt.figure(figsize = (20, 20))
plot_tree(model_DT,
        filled=True,
        rounded=True,
        class_names = ['Good NPK', 'Urea and MOP','Urea and DAP', 'DAP','Urea', 'DAP and MOP', 'MOP'],
        feature_names = X.columns,
        fontsize=12)

plt.show()
```



In [148]:
```
# Hyperparameters
parameters = {'criterion': ['gini','entropy'],
              'max_depth': [3, 4, 5, 6, 7]}

# GridSearchCV: to find the best hyperparameters based on the scoring method
#CV: Cross validation
grid_search_DT = GridSearchCV(estimator=model_DT,
                              param_grid=parameters,
                              scoring="accuracy",
                              cv=5,
                              n_jobs=-1)
```

In [149]:
```
# fit the model with the best hyper-parameters
grid_result_DT = grid_search_DT.fit(X_train, y_train)
```

In [150]:
```
# Best hyperparameters in the grid search
grid_result_DT.best_params_
```

Out[150]:
```
{'criterion': 'gini', 'max_depth': 6}
```

In [151]:
```
#Cross validation result
cv_result = pd.DataFrame(grid_result_DT.cv_results_)
cv_result = cv_result[['param_criterion', 'param_max_depth',
                       'split0_test_score',
                       'split1_test_score', 'split2_test_score', 'split3_test_score',
                       'split4_test_score', 'mean_test_score', 'std_test_score',
                       'rank_test_score']]
```

```
cv_result = cv_result.sort_values(by='rank_test_score')
display(cv_result)
print(cv_result.columns)
```

	param_criterion	param_max_depth	split0_test_score	split1_test_score	split2_test_score	split3_test_score	split4_test_score	mean_test_score	std_test_score
3	gini	6	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	0.000000
4	gini	7	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	0.000000
6	entropy	4	0.928571	1.000000	0.923077	1.000000	1.000000	0.970330	0.036380
7	entropy	5	0.928571	1.000000	0.923077	1.000000	1.000000	0.970330	0.036380
8	entropy	6	0.928571	1.000000	0.923077	1.000000	1.000000	0.970330	0.036380
9	entropy	7	0.928571	1.000000	0.923077	1.000000	1.000000	0.970330	0.036380
2	gini	5	0.928571	0.923077	0.923077	0.923077	0.923077	0.924176	0.002198
5	entropy	3	0.928571	0.923077	0.923077	0.923077	0.923077	0.924176	0.002198
1	gini	4	0.857143	0.846154	0.846154	0.846154	0.846154	0.848352	0.004396
0	gini	3	0.714286	0.615385	0.692308	0.692308	0.692308	0.681319	0.034048

```
Index(['param_criterion', 'param_max_depth', 'split0_test_score',
       'split1_test_score', 'split2_test_score', 'split3_test_score',
       'split4_test_score', 'mean_test_score', 'std_test_score',
       'rank_test_score'],
      dtype='object')
```

Accuracy of model:

In [152]:
```
#Prediction
y_pred_Grid = grid_result_DT.predict(X_test)
```

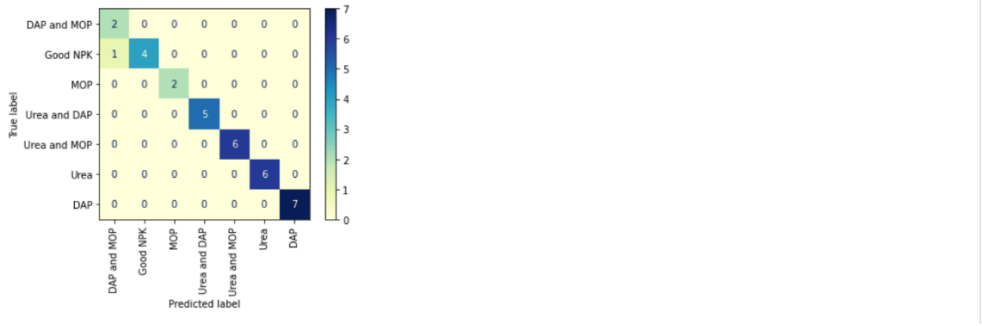
In [153]:
```
# Model Accuracy, how often is the classifier correct?
print('Accuracy: ', accuracy_score(y_test, y_pred_Grid))

#Classification report
print(classification_report(y_test, y_pred_Grid))
```

```
Accuracy:  0.9090909090909091
              precision    recall  f1-score   support

    10-26-26       0.00      0.00      0.00         2
    14-35-14       1.00      0.80      0.89         5
    17-17-17       0.40      1.00      0.57         2
       20-20       1.00      1.00      1.00         5
       28-28       1.00      1.00      1.00         6
         DAP       1.00      1.00      1.00         6
        Urea       1.00      1.00      1.00         7

    accuracy                           0.91        33
   macro avg       0.77      0.83      0.78        33
weighted avg       0.90      0.91      0.90        33
```

In [154]:
```
#Random Forest model
model_RF = RandomForestClassifier(random_state=92)

# Train the model using the training sets
model_RF = model_RF.fit(X_train, y_train)

#Predict the response for test dataset
y_pred_RF = model_RF.predict(X_test)
```

In [155]:
```
# To visualize confusion matrix
plot_confusion_matrix(model_RF,
                      X_test,
                      y_test,
                      display_labels=['DAP and MOP','Good NPK','MOP','Urea and DAP','Urea and MOP','Urea','DAP'],
                      xticks_rotation ='vertical',
                      cmap='YlGnBu')
plt.show()
```

Visualise confusion matrix:



Accuracy of model:

In [156]:
```
# Model Accuracy, how often is the classifier correct?
print('Accuracy: ', accuracy_score(y_test, y_pred_RF))

#Classification report
print(classification_report(y_test, y_pred_RF))
```

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

    10-26-26       0.67      1.00      0.80         2
    14-35-14       1.00      0.80      0.89         5
    17-17-17       1.00      1.00      1.00         2
       20-20       1.00      1.00      1.00         5
       28-28       1.00      1.00      1.00         6
         DAP       1.00      1.00      1.00         6
        Urea       1.00      1.00      1.00         7

    accuracy                           0.97        33
   macro avg       0.95      0.97      0.96        33
weighted avg       0.98      0.97      0.97        33
```

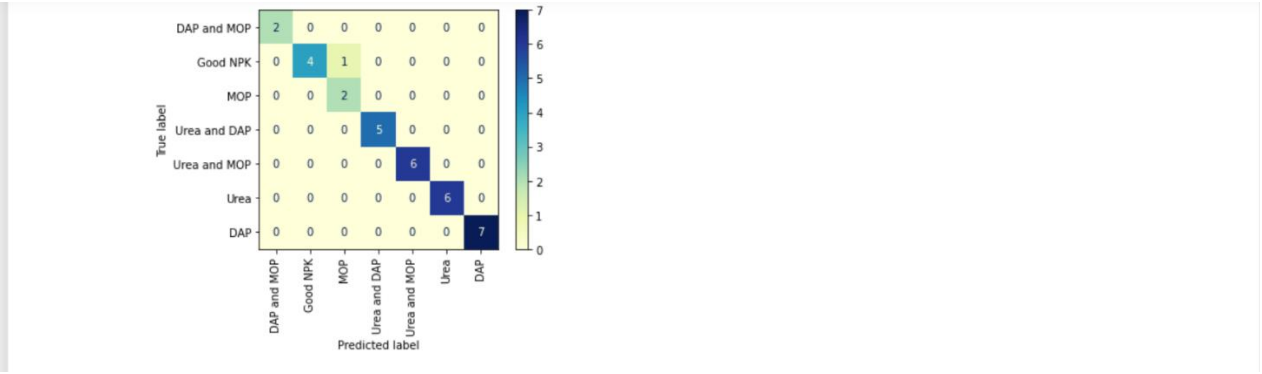
In [157]:
```
#Gradient Boosting model
model_GB = GradientBoostingClassifier()
model_GB= GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,max_depth=1, random_state=0).fit(X_train, y_train)
model_GB.score(X_test, y_test)

# Train the model using the training sets
#model_GB = model_GB.fit(X_train, y_train)
```

Out[157]: 0.9696969696969697

In [158]:
```
#Predict the response for test dataset
y_pred_GB = model_GB.predict(X_test)
```

In [159]:
```
# To visualize confusion matrix
plot_confusion_matrix(model_GB,
                      X_test,
                      y_test,
                      display_labels=['DAP and MOP','Good NPK','MOP','Urea and DAP','Urea and MOP','Urea','DAP'],
                      xticks_rotation ='vertical',
                      cmap='YlGnBu')
plt.show()
```

Confusion matrix:



Accuracy of model:

```
In [160]: # Model Accuracy, how often is the classifier correct?
          print('Accuracy: ', accuracy_score(y_test, y_pred_GB))

          #Classification report
          print(classification_report(y_test, y_pred_GB))
```

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

    10-26-26       1.00      1.00      1.00         2
    14-35-14       1.00      0.80      0.89         5
    17-17-17       0.67      1.00      0.80         2
       20-20       1.00      1.00      1.00         5
       28-28       1.00      1.00      1.00         6
         DAP       1.00      1.00      1.00         6
        Urea       1.00      1.00      1.00         7

    accuracy                           0.97        33
   macro avg       0.95      0.97      0.96        33
weighted avg       0.98      0.97      0.97        33
```

```
In [161]: print("Model\t\t\tAccuracy of the model")
          print("""K-Nearest Neighbors\t\t {:.4f}""".format( accuracy_score(y_test, y_pred_knn)))
          print("""Decision Tree\t\t\t {:.4f}""".format(accuracy_score(y_test, y_pred_DT)))
          print("""Random Forest\t\t\t {:.4f}""".format(accuracy_score(y_test, y_pred_RF)))
          print("""Gradient Boosting\t\t {:.4f}""".format( accuracy_score(y_test, y_pred_GB)))
```

```
Model                   Accuracy of the model
K-Nearest Neighbors              0.8182
Decision Tree                    0.9091
Random Forest                    0.9697
Gradient Boosting                0.9697
```

```
In [162]: df
```

Out[162]:

	Temparature	Humidity	Moisture	Soil Type	Crop Type	Nitrogen	Potassium	Phosphorous	Fertilizer
0	26	52	38	4	3	37	0	0	6
1	29	52	45	2	8	12	0	36	5
2	34	65	62	0	1	7	9	30	1
3	32	62	34	3	9	22	0	20	4
4	28	54	46	1	6	35	0	0	6
5	26	52	35	4	0	12	10	13	2
6	25	50	64	3	1	9	0	10	3
7	33	64	50	2	10	41	0	0	6

```
In [178]: #For K-Nearest Neighbours model
          #data = np.array([[28, 96, 17, 6,77, 4, 4, 4]])
          #i = df.iloc[45,] - gets 45th row from dataframe
          i = get_crop('Pulses')
          i = i.drop(['Fertilizer '])
          prediction =
          (np.array(i).reshape(1,-1))
          print(prediction)

          ['DAP']

In [169]: def get_crop(crop):
              for i, row in df.iterrows():
                  if le.classes_[row['Crop Type']] == crop:
                      return row

In [165]: i = get_paddy()
          print(np.array(i))

          [28 54 46  1  6 35  0  0  6]

In [166]: le.classes_

Out[166]: array(['Barley', 'Cotton', 'Ground Nuts', 'Maize', 'Millets', 'Oil seeds',
                 'Paddy', 'Pulses', 'Sugarcane', 'Tobacco', 'Wheat'], dtype=object)
```

7. RESULTS & DISCUSSIONS:

7.1 RESULT ANALYSIS:

CROP RESULT:

```
In [95]: #data = array([[28, 96, 17, 6,77, 4, 4]])
         import pandas as pd
         df=pd.DataFrame(crop_data)
         df.pop('label')
         i = df.iloc[45,] #- gets 45th row from dataframe
         #i = get_crop('Pulses')
         #i = i.drop(['Fertilizer '])
         prediction = knn_clf.predict(np.array(i).reshape(1,-1))
         #y=pd.get_dummies(np.array(prediction))

         #print(y)
         print(prediction)

         [[0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]]
```

Out[95]:

	apple	banana	blackgram	chickpea	coconut	coffee	cotton	grapes	jute	kidneybeans	...	mango	mothbeans	mungbean	muskmelon	orange	papay
1270	0	0	0	0	0	0	0	1	0	0	...	0	0	0	0	0	
1481	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0	
1832	0	0	0	0	1	0	0	0	0	0	...	0	0	0	0	0	
293	0	0	0	1	0	0	0	0	0	0	...	0	0	0	0	0	
1307	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
740	0	0	1	0	0	0	0	0	0	0	...	0	0	0	0	0	
1032	0	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
2121	0	0	0	0	0	1	0	0	0	0	...	0	0	0	0	0	

INPUTS: N, P, K, temperature, humidity, PH, rainfall

OUTPUT: Label

In this crop recommendation module, we got 1 as the values in array for label of crop. That 1 is nothing but “GRAPES” since in the place of “GRAPES” 1 has appeared all other crop label has 0 as dummies.

FERTILIZER RESULT:

```
In [178]: #For K-Nearest Neighbours model
          #data = np.array([[28, 96, 17, 6,77, 4, 4, 4]])
          #i = df.iloc[45,] - gets 45th row from dataframe
          i = get_crop('Pulses')
          i = i.drop(['Fertilizer '])
          prediction =
          (np.array(i).reshape(1,-1))
          print(prediction)

          ['DAP']

In [169]: def get_crop(crop):
              for i, row in df.iterrows():
                  if le.classes_[row['Crop Type']] == crop:
                      return row

In [165]: i = get_paddy()
          print(np.array(i))

          [28 54 46  1  6 35  0  0  6]

In [166]: le.classes_

Out[166]: array(['Barley', 'Cotton', 'Ground Nuts', 'Maize', 'Millets', 'Oil seeds',
                 'Paddy', 'Pulses', 'Sugarcane', 'Tobacco', 'Wheat'], dtype=object)
```

Here in this fertilizer recommendation module, we have got ‘DAP’ as the result, b for input crop type ‘PULSES’

INPUTS: Temperature, humidity, moisture, soil type, crop type, nitrogen, potassium, phosphorus.

OUTPUT: Fertilizer

8. CONCLUSION & FUTURE WORKS:

The present research work discussed about the variety of features that are mainly dependent on the data availability and each of the research will investigated crop and fertilizer recommendation using ML algorithms that differed from the features. The features were chosen based upon the geological position, scale, and crop features and these choices were mainly dependent upon the data-set availability, but the more features usage was not always giving better results. Most of the exiting models utilized Neural networks, random forests, KNN regression techniques for crop and fertilizer recommendation and a variety of ML techniques were also used for best prediction. From the studies most of the common algorithms used were Random Forest, Decision Tree, Gradient Boosting and KNN algorithms but still improvement was still required further in crop and fertilizer recommendation The present research shows several existing models that consider elements such as temperature, weather condition, performing models for the effective crop and fertilizer recommendation. Ultimately, the experimental study showed the combination of ML with the agricultural domain field for improving the advancement in crop and fertilizer recommendation.

9. REFERENCES:

R. Ghadge, J. Kulkarni, P. More, S. Nene, and R. L. Priya, “Prediction of crop yield using machine learning,” Int. Res. J. Eng. Technology, vol. 5, 2018.

F. H. Tseng, H. H.Cho, and H. T. Wu, “Applying big data for intelligent agriculture-based crop selection analysis,” IEEE Access, vol. 7, pp. 116965-116974, 2019.

[3] A. Suresh, N. Manjunathan, P. Rajesh, and E. Thangadurai, “Crop Yield Prediction Using Linear Support Vector Machine,” European Journal of Molecular & Clinical Medicine, vol. 7, no. 6, pp. 2189- 2195, 2020

[4] M. Alagurajan, and C. Vijayakumaran, “ML Methods for Crop Yield Prediction and Estimation: An Exploration,” International Journal of Engineering and Advanced Technology, vol. 9 no. 3, 2020

[5] P. Kumari, S. Rathore, A. Kalamkar, and T. Kambale, “Predicition of Crop Yeild Using SVM Approch with the Facility of E-MART System” Easychair 2020.

6] S. D. Kumar, S. Esakkirajan, S. Bama, and B. Keerthiveena, “A microcontroller-based machine vision approach for tomato grading and sorting using SVM classifier,” Microprocessors and Microsystems, vol. 76, pp.103090, 2020

[7] P. Tiwari, and P. Shukla, “Crop yield prediction by modified convolutional neural network and geographical indexes,” International Journal of Computer Sciences and Engineering, vol. 6, no. 8, pp. 503-513, 2018.

[8] P. Sivanandhini, and J. Prakash, “Crop Yield Prediction Analysis using Feed Forward and Recurrent Neural Network,” International Journal of Innovative Science and Research Technology, vol. 5, no. 5, pp. 1092-1096, 2020.

[9] N. Nandhini, and J. G. Shankar, “Prediction of crop growth using machine learning based on seed,” Ictact journal on soft computing, vol. 11, no. 01, 2020

[10] A. A. Alif, I. F. Shukanya, and T. N. Afee, “Crop prediction based on geographical and climatic data using machine learning and deep learning”, Doctoral dissertation, BRAC University) 2018.

[11] A. Fuentes, S. Yoon, S. C. Kim, and D. S. Park, “A robust deeplearning-based detector for real-time tomato plant diseases and pests’ recognition,” Sensors, vol. 17, no. 9, pp. 2022, 2017.

[12] J. Sun, L, Di, Z. Sun, Y. Shen, and Z. Lai, “County-level soybean yield prediction using deep CNN-LSTM model,” Sensors, vol. 19, no. 20, pp. 4363, 2019.

[13] K. A. Shastry, and H. A. Sanjay, “Hybrid prediction strategy to predict agricultural information,” Applied Soft Computing, vol. 98, pp. 106811, 2021.

[14] D. A. Bondre, and S. Mahagaonkar, “Prediction of Crop Yield and Fertilizer Recommendation Using Machine Learning Algorithms,” International Journal of Engineering Applied Sciences and Technology, vol. 4, no. 5, pp. 371-376, 2019.

[15] B. Devika, and B. Ananthi, “Analysis of crop yield prediction using data mining technique to predict annual yield of major crops,” International Research Journal of Engineering and Technology, vol. 5, no.12, pp. 1460-1465, 2018.