A Project by Bethlehem Kebede

Building a Predictive Model so as to suggest the most suitable crops to grow based on the available climatic and soil conditions.

7 key features that I've used: Amount of Nitrogen, Phosphorus and Potassium in soil, Temperature in degree celcius, Humidity, pH and Rainfall in mm.

In [7]:
```
#Tasks:
##1. Comparing the averge soil and climiate condition for different Crops?
##2. Visualize the Distribution of agriculture condications and identify crops which require higher and lower agriculture condications? ##3. Find crops which are able
to grow in same soil and climate conditions?
##4. Designe a predictive machine learning model using Logistic Regression, Decision tree and random forest? ##5. Evaluate the
performance of the model?
```

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# for interactivity
from __future__ import print_function
from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets
import warnings
warnings.simplefilter(action="ignore", category=FutureWarning)
pd.options.mode.chained_assignment = None # To omit default='warnings' from
```

In [2]:
```
#reading the data and show top five records
Agri_data=pd.read_csv('Agriculture data.csv')
Agri_data.head()
```

Out[2]:

	Nitrogen	Phosphorus	K(Potassium)	Temperature	Humidity	Ph Value	Rainfall	Crop Name
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

1. Data preprocessing

In [3]:
```
###Total Instance of data
Agri_data.shape
```

Out[3]: (2200, 8)

In [4]:
```
#Checking the number of null values on each columns
Agri_data.isnull().sum()
```

Out[4]:
```
Nitrogen          0
Phosphorus        0
K(Potassium)      0
Temperature       0
Humidity          0
Ph Value          0
Rainfall          0
Crop Name         0
```

In [5]:
```
#checking duplicate records: Indicating has no duplicate records.
Agri_data.duplicated(keep=False)
```

Out[5]:
```
0        False
1        False
2        False
3        False
4        False
         ...
2195     False
2196     False
2197     False
2198     False
2199     False
```

In [6]:
```
#Column Names
Agri_data.columns
```

Out[6]:
```
Index(['Nitrogen', 'Phosphorus', 'K(Potassium)', 'Temperature', 'Humidity', 'Ph Value',
       'Rainfall', 'Crop Name'],
      dtype='object')
```

In [7]:
```
#Checking the percentage of each crop to check whether it's balanced or not.
crop_percentage={}
for crop in Agri_data['Crop Name'].unique():
    crop_percentage[crop]=len(Agri_data[Agri_data['Crop Name']==crop])
for crop,perc in crop_percentage.items():
    print(crop + " Percentage {:.4f} %".format(crop_percentage[crop]/(crop_percentage[crop]+sum(crop_percentage.values()))))
```

```
rice Percentage 0.0435 % maize
Percentage 0.0435 % chickpea
Percentage 0.0435 %
kidneybeans Percentage 0.0435 %
pigeonpeas Percentage 0.0435 %
mothbeans Percentage 0.0435 %
mungbean Percentage 0.0435 %
blackgram Percentage 0.0435 % lentil
Percentage 0.0435 % pomegranate
Percentage 0.0435 % banana Percentage
0.0435 % mango Percentage 0.0435 %
grapes Percentage 0.0435 % watermelon
Percentage 0.0435 % muskmelon
Percentage 0.0435 % apple Percentage
0.0435 % orange Percentage 0.0435 %
papaya Percentage 0.0435 % coconut
Percentage 0.0435 % cotton
Percentage  0.0435  % jute  Percentage
0.0435  % coffee Percentage 0.0435 %
```

2.Which Crop requires higher, min & average soil and climate conditions?

In [8]:
```
# List of target values
Agri_data['Crop Name'].unique()
```

Out[8]:
```
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 [3]:
```
#Comparing the averge soil and climiate condition  for  Some  Specific  Crops
@interact
def  compare(Conditions=['Nitrogen','Phosphorus','K(Potassium)','Humidity','Ph  Value',¥
                                    'Rainfall','Temperature']):
    print("The Average Value for",Conditions,round(Agri_data[Conditions].mean(),2)) print("                ")
    print("The Average Value for Rice",round(Agri_data[Agri_data['Crop Name']=='rice'][Conditions].mean(),2)) print("The Average Value for
    papaya",round(Agri_data[Agri_data['Crop Name']=='papaya'][Conditions].mean(),2)) print("The Average Value for
    cotton",round(Agri_data[Agri_data['Crop Name']=='cotton'][Conditions].mean(),2))
    print("The Average Value for kidneybeans",round(Agri_data[Agri_data['Crop Name']=='kidneybeans'][Conditions].mean(),2))
```

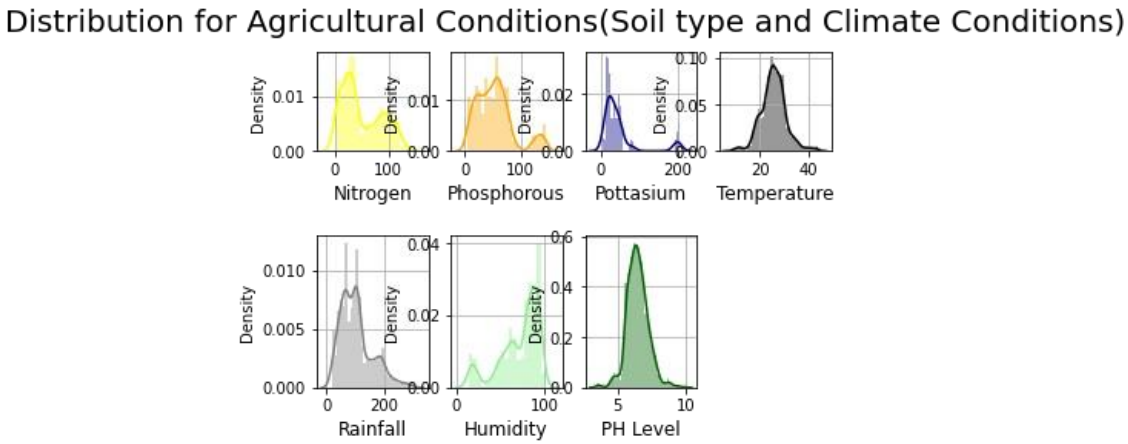
```
interactive(children=(Dropdown(description='Conditions', options=('Nitrogen', 'Phosphorus', 'K(Potassium)', 'H…
```

In [4]:
```
# A statistics which shows crops which require < the average,& above the average of the given soil & Climate condication
@interact
def compare(Conditions=['Nitrogen','Phosphorus','K(Potassium)','Humidity','Ph Value','Rainfall','Temperature']): print("Crops which
    requires greater than Average:",Conditions, '¥n') print(Agri_data[Agri_data[Conditions]>Agri_data[Conditions].mean()]['Crop
    Name'].unique())
    print("                                                                                ")
    print("Crops which requires less than Average:",Conditions, '¥n') print(Agri_data[Agri_data[Conditions]<=
    Agri_data[Conditions].mean()]['Crop Name'].unique())
```

```
interactive(children=(Dropdown(description='Conditions', options=('Nitrogen', 'Phosphorus', 'K(Potassium)', 'H…
```

In [6]:
```
# Visualizing the Distribution of agriculture condications crops
plt.subplot(3,4,1)
```

```
sns.distplot(Agri_data['Nitrogen'], color="yellow")
plt.xlabel('Nitrogen', fontsize = 12)
plt.grid()

plt.subplot(3,4,2) sns.distplot(Agri_data['Phosphorus'],
color="orange") plt.xlabel('Phosphorous', fontsize = 12)
plt.grid()

plt.subplot(3,4,3) sns.distplot(Agri_data['K(Potassium)'],
color="darkblue") plt.xlabel('Pottasium', fontsize = 12)
plt.grid()

plt.subplot(3,4,4) sns.distplot(Agri_data['Temperature'],
color="black") plt.xlabel('Temperature', fontsize = 12)
plt.grid()

plt.subplot(2,4,5) sns.distplot(Agri_data['Rainfall'],
color="grey") plt.xlabel('Rainfall', fontsize = 12)
plt.grid()

plt.subplot(2,4,6)
sns.distplot(Agri_data['Humidity'], color="lightgreen")
plt.xlabel('Humidity', fontsize = 12)
plt.grid()

plt.subplot(2,4,7)
sns.distplot(Agri_data['Ph Value'], color="darkgreen") plt.xlabel('PH
Level', fontsize = 12)
plt.grid()

plt.suptitle('Distribution for Agricultural Conditions(Soil type and Climate Conditions)', fontsize = 20) plt.show()
```



In [12]:
```
# Base on the above distirbutiion plot , we can Identifying crops with some specific climate and soil conditions

print("Crops which requries very High Ratio of Nitrogen content:",Agri_data[Agri_data['Nitrogen']>100]['Crop Name'].unique()
)
print("crops which requries very High Ratio of Phosphorus:",Agri_data[Agri_data['Phosphorus']>100]['Crop Name'].unique())
```

```
print("crops which requries very High Ratio of Potassium:",Agri_data[Agri_data['K(Potassium)']>200]['Crop Name'].unique()) print("crops which requries very High
Rainfall:",Agri_data[Agri_data['Rainfall']>200]['Crop Name'].unique())
print("crops which requries very high Temprature:",Agri_data[Agri_data['Temperature']>40]['Crop Name'].unique()) print("crops which
requries very low Temprature:",Agri_data[Agri_data['Temperature']<10]['Crop Name'].unique()) print("crops which requries very low
Humidity:",Agri_data[Agri_data['Humidity']<20]['Crop Name'].unique()) print("crops which requries very High Ph
Value:",Agri_data[Agri_data['Ph Value']>9]['Crop Name'].unique()) print("crops which requries very low Ph
value:",Agri_data[Agri_data['Ph Value']<5]['Crop Name'].unique())
```

```
Crops which requries very High Ratio of Nitrogen content: ['banana' 'watermelon' 'muskmelon' 'cotton' 'coffee'] crops which requries very
High Ratio of Phosphorus: ['grapes' 'apple']
crops which requries very High Ratio of Potassium: ['grapes' 'apple'] crops which
requries very High Rainfall: ['rice' 'papaya' 'coconut'] crops which requries very
high Temprature: ['grapes' 'papaya']
crops which requries very low Temprature: ['grapes']
crops which requries very low Humidity: ['chickpea' 'kidneybeans'] crops which
requries very High Ph Value: ['mothbeans']
crops which requries very low Ph value: ['pigeonpeas' 'mothbeans' 'mango']
```

3. Find crops which are able to grow in same soil and climate conditions

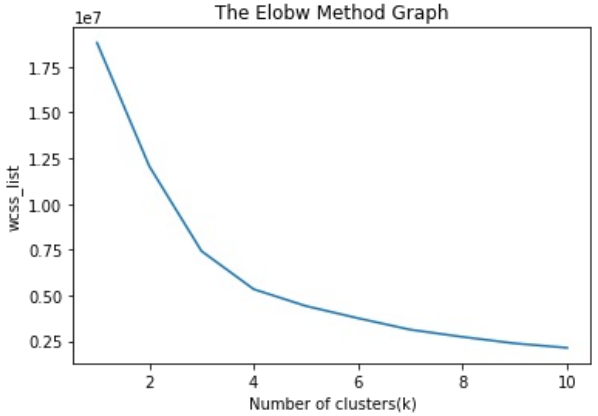
In [8]:

```
##Applying Kmeans Clustering analysis:It is an iterative algorithm that divides the unlabeled dataset into k different clusters ##in such a way that each
dataset belongs only one group that has similar properties.
from sklearn.cluster import KMeans
#removing the target  column
x = Agri_data.drop(['Crop Name'], axis=1)
#selecting all the values of data
x = x.values #checking
the shape print(x.shape)
```

```
(2200, 7)
```

In [9]:

```
##Finding the optimal number of clusters using the elbow method
##Elbow Method: To choose the value of "K number of clusters": A point of the plot looks like an arm, ##then that point is
considered as the best value of K.
##With random_state=None , we get different train and test sets across different executions. ##With random_state=0 ,
we get the same train and test sets across different executions. wcss_list= []  #Initializing the list for the
values of WCSS
#Using for loop for iterations from 1 to 10.
for i in range(1, 11):
     kmeans = KMeans(n_clusters=i, init='k-means++', max_iter = 2000, n_init = 10,random_state= 0) kmeans.fit(x)
     wcss_list.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss_list) plt.title('The Elobw
Method Graph') plt.xlabel('Number of clusters(k)')
plt.ylabel('wcss_list')
plt.show()

#I'm choosing K=4 which is a point of plot looks like an arm
```



In [10]:
```
### Implementing the K Means algortiham  to perform clustering analysis
#n_init (default as 10): Represents the number of time the k-means algorithm will be run independently.

#training the K-means model on a dataset
km = KMeans(n_clusters = 4, init = 'k-means++',  max_iter = 2000, n_init = 10, random_state = 0) y_means =
km.fit_predict(x)

#Finding the results
a = Agri_data['Crop Name'] y_means =
pd.DataFrame(y_means)
z = pd.concat([y_means, a], axis = 1)
```

In [11]:
```
# Checking the clusters for each crop
print("..........")
print("Results after applying K Means Clustering Analysis ¥n") print("
..........")
print("Crops in First Cluster:", z[z['Cluster_Group'] == 0]['Crop Name'].unique()) print("      ")
print("Crops in Second Cluster:", z[z['Cluster_Group'] == 1]['Crop Name'].unique()) print("     ")
print("Crops in Third Cluster:", z[z['Cluster_Group'] == 2]['Crop Name'].unique()) print("      ")
print("Crops in Fourth Cluster:", z[z['Cluster_Group'] == 3]['Crop Name'].unique())
#Group of crops which are able to grow in same soil and climate conditions
```

```
.....
Results after applying K Means Clustering Analysis

.....
Crops in First Cluster: ['maize' 'chickpea' 'kidneybeans' 'pigeonpeas' 'mothbeans' 'mungbean' 'blackgram' 'lentil'
 'pomegranate' 'mango' 'orange' 'papaya' 'coconut']
.....
Crops in Second Cluster: ['maize' 'banana' 'watermelon' 'muskmelon' 'papaya' 'cotton' 'coffee']
.....
Crops in Third Cluster: ['grapes' 'apple']
.....
Crops in Fourth Cluster: ['rice' 'pigeonpeas' 'papaya' 'coconut' 'jute' 'coffee']
```

4. Designing a predictive model (Classification Algorithms)

In [ ]:
```
#1. Logistic Regression algorithm
```

In [12]:
```
#Splitting the Dataset (80 to 20) #Dependant
variable
y = Agri_data['Crop Name']
#Independant variables
x = Agri_data.drop(['Crop Name'], axis=1)
print("Shape of x:", x.shape) print("Shape of
y:", y.shape)
```

```
Shape of x: (2200, 7) Shape of
y: (2200,)
```

In [13]:
```
#Creating training and testing sets for results validation
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 0) print("The Shape Of x
train:", x_train.shape)
print("The Shape Of x test:", x_test.shape) print("The
Shape Of y train:", y_train.shape) print("The Shape Of
y test:", y test.shape)
```

```
The Shape Of x train: (1760, 7) The
Shape Of x test: (440, 7) The Shape
Of y train: (1760,) The Shape Of y
test: (440,)
```

In [ ]:
```
#Chosing the Model

#Three models we are going to try out: #Logistic
Regression
#Decision Tree Regressor #Random
Forest Classifier
```

In [16]:
```
# Put models in a dictionary
from sklearn.linear_model import LogisticRegression from
sklearn.neighbors import KNeighborsClassifier from
sklearn.ensemble import RandomForestClassifier models =
{'Logistic Regression': LogisticRegression(),
           'KNN': KNeighborsClassifier(),
           'Random Forest': RandomForestClassifier()}

# Create function to fit and score models

def fit_and_score(models, x_train, x_test, y_train, y_test): '''
     Fits and evaluates given machine learning models
     models : a dict of different classification sklearn models X_train :
     training data , no labels
     X_test : testing data, no labels
     y_train : training labels y_test: test
     labels
     '''
     # set random seed
     np.random.seed(42)
     # make a dict to keep model scores
     model_scores = {}
     # loop through models
     for name, model in models.items():
          # fit model to data
          model.fit(x_train.values, y_train)
          # Evaluate model and append score to model score
```

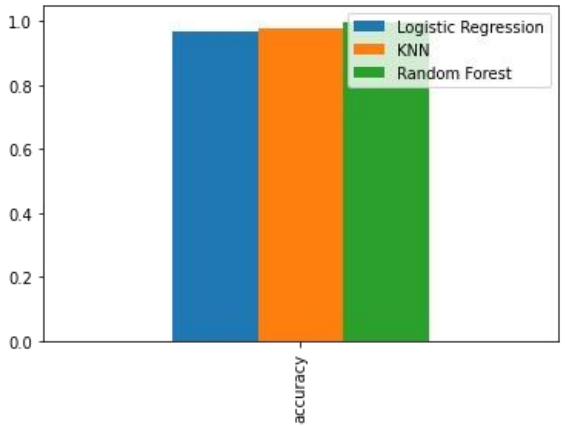
```
    return model_scores
```

In [17]:
```
model_scores = fit_and_score(models=models,
                             x_train=x_train, x_test=x_test,
                             y_train=y_train, y_test=y_test)
model_scores
```

```
C:¥Users¥anduamlak.yitayeh.ETHIO.000¥AppData¥Local¥Programs¥Python¥Python310¥lib¥site-packages¥sklearn¥linear_model¥_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in: https://scikit-
    learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options: https://scikit-
    learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[17]:
```
{'Logistic Regression': 0.9681818181818181,
 'KNN': 0.9772727272727273,
 'Random Forest': 0.9977272727272727}
```

In [18]:
```
# Visualize
model_compare = pd.DataFrame(model_scores, index=['accuracy']) model_compare.plot.bar();
```

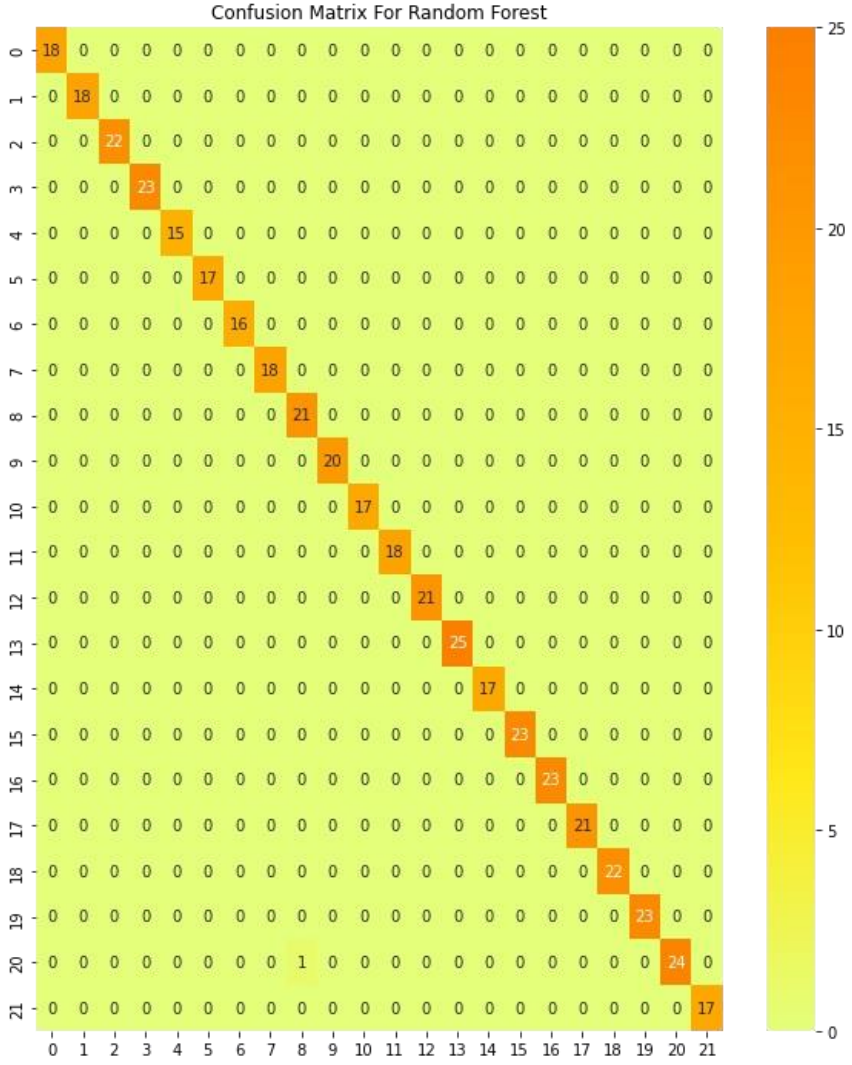


In [19]:
```
y_pred = {}
for name,model in models.items():
    y_pred[name]= model.predict(x_test)
```

```
C:¥Users¥anduamlak.yitayeh.ETHIO.000¥AppData¥Local¥Programs¥Python¥Python310¥lib¥site-packages¥sklearn¥base.py:402: UserWarning: X has feature names, but LogisticRegr ession was fitted without feature
names
  warnings.warn(
C:¥Users¥anduamlak.yitayeh.ETHIO.000¥AppData¥Local¥Programs¥Python¥Python310¥lib¥site-packages¥sklearn¥base.py:402: UserWarning: X has feature names, but KNeighborsCl assifier was fitted without
feature names
  warnings.warn(
C:¥Users¥anduamlak.yitayeh.ETHIO.000¥AppData¥Local¥Programs¥Python¥Python310¥lib¥site-packages¥sklearn¥base.py:402: UserWarning: X has feature names, but RandomForest Classifier was fitted without
feature names
```

In [20]:
```
#Evaluating the model performance
from sklearn.metrics import confusion_matrix
#Printing the Confusing Matrix
```

```
plt.rcParams['figure.figsize'] = (10,12)
cm = confusion_matrix(y_test, y_pred['Random Forest']) sns.heatmap(cm, annot =
True, cmap = 'Wistia') plt.title('Confusion Matrix For Random Forest', fontsize
= 12) plt.show()
```



In [21]:

```
#Defining the classification Report for measuring the precision ,recall and f1-score for all target values
from sklearn.metrics import classification_report
cr = classification_report(y_test, y_pred['Random Forest']) print(cr)
```

	precision	recall	f1-score	support
apple	1.00	1.00	1.00	18
banana	1.00	1.00	1.00	18
blackgram	1.00	1.00	1.00	22
chickpea	1.00	1.00	1.00	23
coconut	1.00	1.00	1.00	15
coffee	1.00	1.00	1.00	17
cotton	1.00	1.00	1.00	16
grapes	1.00	1.00	1.00	18
jute	0.95	1.00	0.98	21
kidneybeans	1.00	1.00	1.00	20
lentil	1.00	1.00	1.00	17
maize	1.00	1.00	1.00	18
mango	1.00	1.00	1.00	21
mothbeans	1.00	1.00	1.00	25
mungbean	1.00	1.00	1.00	17
muskmelon	1.00	1.00	1.00	23
orange	1.00	1.00	1.00	23
papaya	1.00	1.00	1.00	21
pigeonpeas	1.00	1.00	1.00	22
pomegranate	1.00	1.00	1.00	23
rice	1.00	0.96	0.98	25
watermelon	1.00	1.00	1.00	17
accuracy			1.00	440
macro avg	1.00	1.00	1.00	440
weighted avg	1.00	1.00	1.00	440

In [22]:
```
#Sanple tests for preiction
test_input=Agri_data.columns[:7] arr=[]
counter=0
for col_nam in test_input:
    user_input = float(input('Enter '+col_nam+' Value:')) arr.append(user_input)
    counter=counter+1
    if(counter>8): break
for name, model in models.items():
    prediction = model.predict(np.array([arr]))
    if (name=='Logistic Regression'):
        print("The Suggested Crop for given climatic and soil condition is using " +name,prediction) elif (name=='KNN'):
        print("The Suggested Crop for given climatic and soil condition is using "+name,prediction)
    else:
        print("The Suggested Crop for given climatic and soil condition is using "+name,prediction)
```

```
Enter Nitrogen Value:40 Enter
Phosphorus Value:40 Enter
K(Potassium) Value:50 Enter
Temperature Value:56 Enter
Humidity Value:34 Enter Ph Value
Value:2 Enter Rainfall Value:234
The Suggested Crop for given climatic and soil condition is using Logistic Regression ['mango'] The Suggested Crop
for given climatic and soil condition is using KNN ['papaya']
The Suggested Crop for given climatic and soil condition is using Random Forest ['mango']
```

In [ ]:
```
# Thank you!
```