In [1]:
```
#for manipulation
import pandas as pd
import numpy as np
#for data visualizations
import seaborn as sns
import matplotlib.pyplot as plt
#for interactivity
from ipywidgets import interact
```

In [2]:
```
#data file as df
#read the data file
df=pd.read_csv("E:\\projects\\DS&ML\\AGRICULTURAL PRODUCTION\\Crop_recommendation.c
```

In [3]:
```
#shape of dataset
print(df.shape)
```

```
(2200, 8)
```

Shape of dataset: (2200, 8) 2200 rows 8 columns

In [4]:
```
#missing value
df.isnull().sum()
```

Out[4]:
```
N              0
P              0
K              0
temperature    0
humidity       0
ph             0
rainfall       0
label          0
dtype: int64
```

There is no missing value in dataset

In [5]:
```
#check the crops present in the dataset
df['label'].value_counts()
```

Out[5]:
```
rice           100
maize          100
jute           100
cotton         100
coconut        100
papaya         100
orange         100
apple          100
muskmelon      100
watermelon     100
grapes         100
mango          100
banana         100
pomegranate    100
lentil         100
blackgram      100
mungbean       100
mothbeans      100
pigeonpeas     100
kidneybeans    100
chickpea       100
coffee         100
Name: label, dtype: int64
```

In [6]:
```
#summary of all crops
```

```
print("Average ratio of Nitrogen in the soil: {0:.2f}".format(df['N'].mean()))
print("Average ratio of Phosphorous in the soil: {0:.2f}".format(df['P'].mean()))
print("Average ratio of Potassium in the soil: {0:.2f}".format(df['K'].mean()))
print("Average ratio of Temperature in celsius: {0:.2f}".format(df['temperature'].m
print("Average ratio of Humidity in %: {0:.2f}".format(df['humidity'].mean()))
print("Average ratio of PH value in the soil: {0:.2f}".format(df['ph'].mean()))
print("Average ratio of Rainfall in mm: {0:.2f}".format(df['rainfall'].mean()))
```

```
Average ratio of Nitrogen in the soil: 50.55
Average ratio of Phosphorous in the soil: 53.36
Average ratio of Potassium in the soil: 48.15
Average ratio of Temperature in celsius: 25.62
Average ratio of Humidity in %: 71.48
Average ratio of PH value in the soil: 6.47
Average ratio of Rainfall in mm: 103.46
```

In [7]:

```
#summary of each crops

@interact
def summary(crops=list(df['label'].value_counts().index)):
    x=df[df['label']==crops]
    print("------------------------------------------------")
    print("Statistics for Nitrogen")
    print("Maximum Nitrogen required:", x['N'].max())
    print("Average Nitrogen required:", x['N'].mean())
    print("Minimum Nitrogen required:", x['N'].min())
    print("------------------------------------------------")
    print("------------------------------------------------")
    print("Statistics for Phosphorous")
    print("Maximum Phosphorous required:", x['P'].max())
    print("Average Phosphorous required:", x['P'].mean())
    print("Minimum Phosphorous required:", x['P'].min())
    print("------------------------------------------------")
    print("------------------------------------------------")
    print("Statistics for Potassium")
    print("Maximum Potassium required:", x['K'].max())
    print("Average Potassium required:", x['K'].mean())
    print("Minimum Potassium required:", x['K'].min())
    print("------------------------------------------------")
    print("------------------------------------------------")
    print("Statistics for Temperature")
    print("Maximum Temperature required:", x['temperature'].max())
    print("Average Temperature required:", x['temperature'].mean())
    print("Minimum Temperature required:", x['temperature'].min())
    print("------------------------------------------------")
    print("Statistics for Humidity")
    print("Maximum Humidity required:", x['humidity'].max())
    print("Average Humidity required:", x['humidity'].mean())
    print("Minimum Humidity required:", x['humidity'].min())
    print("------------------------------------------------")
    print("------------------------------------------------")
    print("Statistics for PH")
    print("Maximum PH required:", x['ph'].max())
    print("Average PH required:", x['ph'].mean())
    print("Minimum PH required:", x['ph'].min())
    print("------------------------------------------------")
    print("------------------------------------------------")
    print("Statistics for Rainfall")
    print("Maximum Rainfall required:", x['rainfall'].max())
    print("Average Rainfall required:", x['rainfall'].mean())
    print("Minimum Rainfall required:", x['rainfall'].min())
    print("------------------------------------------------")
```

```
interactive(children=(Dropdown(description='crops', options=('rice', 'maize', 'jut
e', 'cotton', 'coconut', 'pa…
```

In [8]:
```
#compare the average requirement for each crops with average conditions
@interact
def avg_req(conditions=['N','P','K','temperature','ph','humidity','rainfall']):
    print("Average value for", conditions, "is {0:.2f}".format(df[conditions].mean(
    print("--------------------------------------------")
    for i in df['label'].unique():
        print(i+": {0:.2f}".format(df[(df["label"]==i)][conditions].mean()))
```

```
interactive(children=(Dropdown(description='conditions', options=('N', 'P', 'K',
'temperature', 'ph', 'humidit…
```

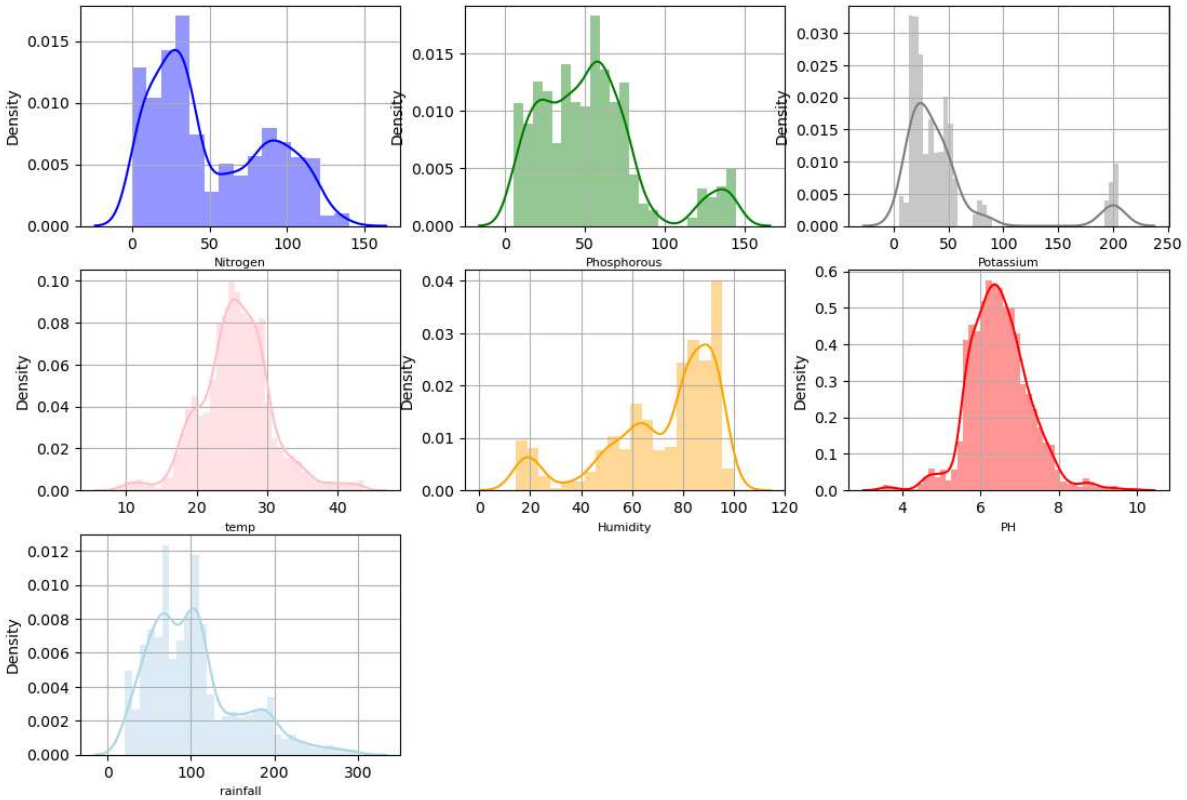
In [9]:
```
#more than average need
@interact
def avg_req(conditions=['N','P','K','temperature','ph','humidity','rainfall']):
    print("crops which require greater than average", conditions,'\n')
    print(df[df[conditions]> df[conditions].mean()]['label'].unique())
    print("----------------------------------------------------")
    print("crops which require less or equal than average", conditions,'\n')
    print(df[df[conditions]<= df[conditions].mean()]['label'].unique())
```

```
interactive(children=(Dropdown(description='conditions', options=('N', 'P', 'K',
'temperature', 'ph', 'humidit…
```

In [10]:
```
#check conditions by graph

plt.figure(figsize=(13, 9))

plt.subplot(3,3,1)
sns.distplot(df['N'], color='blue')
plt.xlabel("Nitrogen", fontsize=8)
plt.grid()

plt.subplot(3,3,2)
sns.distplot(df['P'], color='green')
plt.xlabel("Phosphorous", fontsize=8)
plt.grid()

plt.subplot(3,3,3)
sns.distplot(df['K'], color='grey')
plt.xlabel("Potassium", fontsize=8)
plt.grid()

plt.subplot(3,3,4)
sns.distplot(df['temperature'], color='pink')
plt.xlabel("temp", fontsize=8)
plt.grid()

plt.subplot(3,3,5)
sns.distplot(df['humidity'], color='orange')
plt.xlabel("Humidity", fontsize=8)
plt.grid()

plt.subplot(3,3,6)
sns.distplot(df['ph'], color='red')
plt.xlabel("PH", fontsize=8)
plt.grid()

plt.subplot(3,3,7)
sns.distplot(df['rainfall'], color='lightblue')
plt.xlabel("rainfall", fontsize=8)
plt.grid()
```

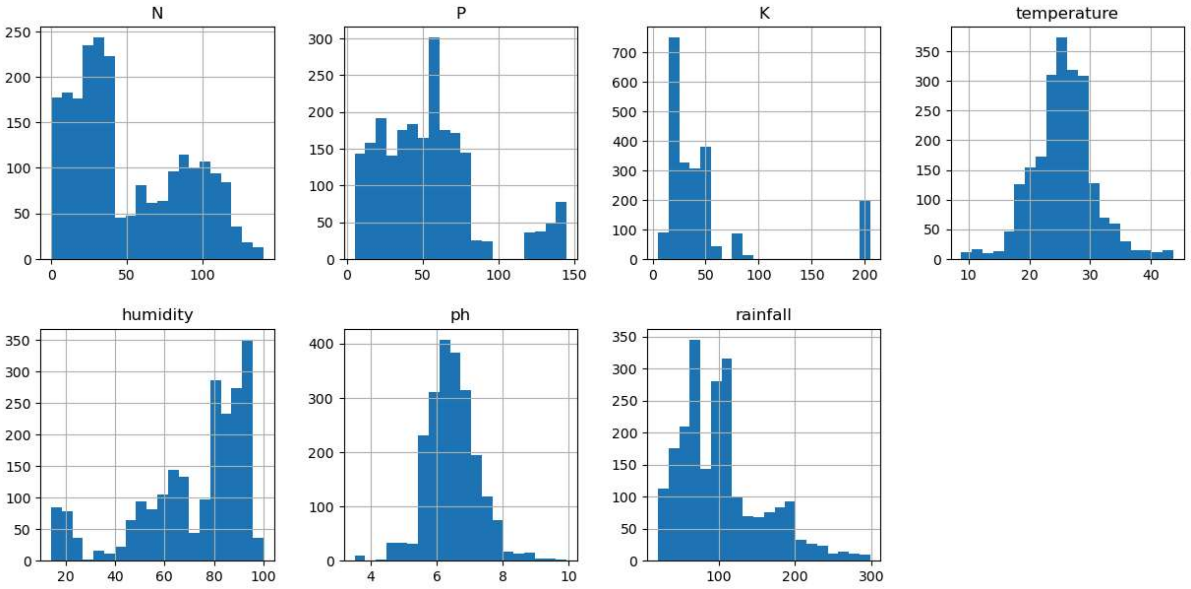
```
C:\Users\abhis\AppData\Local\Temp\ipykernel_7192\205437792.py:6: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['N'], color='blue')
C:\Users\abhis\AppData\Local\Temp\ipykernel_7192\205437792.py:11: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['P'], color='green')
C:\Users\abhis\AppData\Local\Temp\ipykernel_7192\205437792.py:16: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['K'], color='grey')
C:\Users\abhis\AppData\Local\Temp\ipykernel_7192\205437792.py:21: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['temperature'], color='pink')
C:\Users\abhis\AppData\Local\Temp\ipykernel_7192\205437792.py:26: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['humidity'], color='orange')
C:\Users\abhis\AppData\Local\Temp\ipykernel_7192\205437792.py:31: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751
```

```
  sns.distplot(df['ph'], color='red')
C:\Users\abhis\AppData\Local\Temp\ipykernel_7192\205437792.py:36: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['rainfall'], color='lightblue')
```



In [11]:
```
df.hist(figsize=(15,15), layout=(4,4), bins=20)
```

Out[11]:
```
array([[<Axes: title={'center': 'N'}>, <Axes: title={'center': 'P'}>,
        <Axes: title={'center': 'K'}>,
        <Axes: title={'center': 'temperature'}>],
       [<Axes: title={'center': 'humidity'}>,
        <Axes: title={'center': 'ph'}>,
        <Axes: title={'center': 'rainfall'}>, <Axes: >],
       [<Axes: >, <Axes: >, <Axes: >, <Axes: >],
       [<Axes: >, <Axes: >, <Axes: >, <Axes: >]], dtype=object)
```



In [12]:
```
df.skew()
```

```
C:\Users\abhis\AppData\Local\Temp\ipykernel_7192\1665899112.py:1: FutureWarning: T
he default value of numeric_only in DataFrame.skew is deprecated. In a future vers
ion, it will default to False. In addition, specifying 'numeric_only=None' is depr
ecated. Select only valid columns or specify the value of numeric_only to silence
this warning.
  df.skew()
```

Out[12]:
```
N              0.509721
P              1.010773
K              2.375167
temperature    0.184933
humidity      -1.091708
ph             0.283929
rainfall       0.965756
dtype: float64
```

observation:

1. P is right skewed
2. Humidity left skewed

In [13]:
```
#some interesting pattern
print('crops which requires very high ratio of Nitrogen content in soil:', df[df['N
print('crops which requires very high ratio of Phosphorous content in soil:', df[df
print('crops which requires very high ratio of Potassium content in soil:', df[df['
print('crops which requires very high rainfall:', df[df['rainfall']>200]['label'].u
print('crops which requires very low temperature:', df[df['temperature']<10]['label
print('crops which requires very high temperature:', df[df['temperature']>40]['labe
print('crops which requires very low humidity:', df[df['humidity']<20]['label'].uni
print('crops which requires very low ph:', df[df['ph']<4]['label'].unique())
print('crops which requires very high ph:', df[df['ph']>9]['label'].unique())
```

```
crops which requires very high ratio of Nitrogen content in soil: ['cotton']
crops which requires very high ratio of Phosphorous content in soil: ['grapes' 'ap
ple']
crops which requires very high ratio of Potassium content in soil: ['grapes' 'appl
e']
crops which requires very high rainfall: ['rice' 'papaya' 'coconut']
crops which requires very low temperature: ['grapes']
crops which requires very high temperature: ['grapes' 'papaya']
crops which requires very low humidity: ['chickpea' 'kidneybeans']
crops which requires very low ph: ['mothbeans']
crops which requires very high ph: ['mothbeans']
```

observation:

we can see mothbeans grow in very low and very high pH conditions so pH is not a major factor for mothbeans.

In [14]:
```
# Season wise crops can be grown
print('Summer Crops:', df[(df['temperature']>30) & (df['humidity']>50)]['label'].ur
print('Winter Crops:', df[(df['temperature']<20) & (df['humidity']>30)]['label'].ur
print('Rainy Crops:', df[(df['rainfall']>200) & (df['humidity']>30)]['label'].uniqu
```

```
Summer Crops: ['pigeonpeas' 'mothbeans' 'blackgram' 'mango' 'grapes' 'orange' 'pap
aya']
Winter Crops: ['maize' 'pigeonpeas' 'lentil' 'pomegranate' 'grapes' 'orange']
Rainy Crops: ['rice' 'papaya' 'coconut']
```

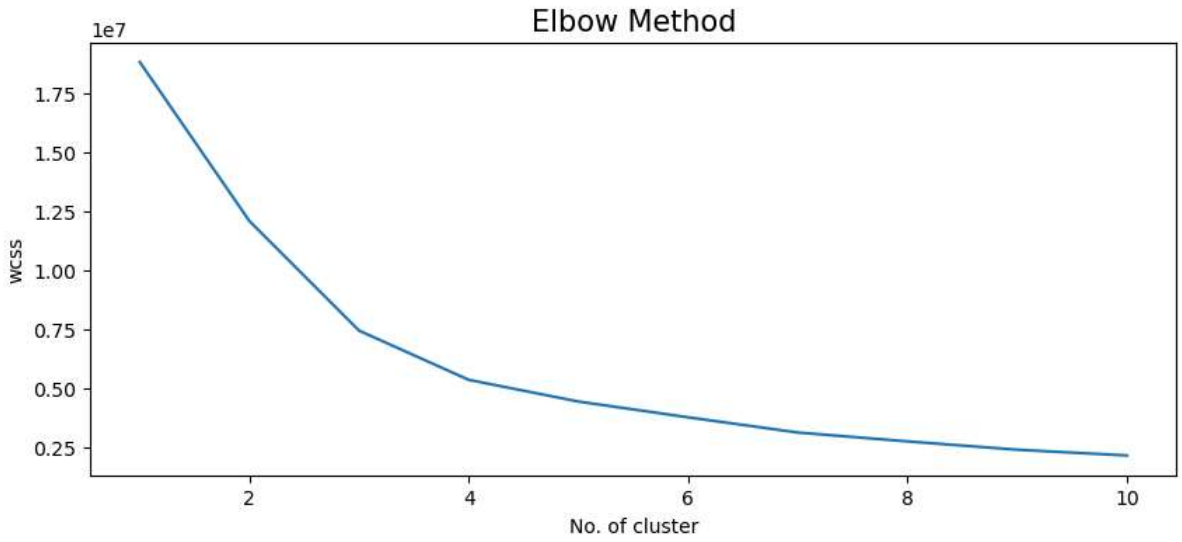
Creating A Model

In [15]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [17]:
```
from sklearn.cluster import KMeans

#removing the labels column
z=df.drop(['label'], axis=1)

#selecting all the values of data
z=z.values

print(z.shape)
```

```
(2200, 7)
```

In [18]:
```
#Determine Optimum number of cluster by elbow method

plt.rcParams['figure.figsize'] = (10,4)
wcss = []
for i in range (1,11):
    km = KMeans(n_clusters =i, init= 'k-means++', max_iter=300, n_init=10, random_s
    km.fit(z)
    wcss.append(km.inertia_)

#plot the results
plt.plot(range(1,11), wcss)
plt.title('Elbow Method', fontsize= 15)
plt.xlabel('No. of cluster')
plt.ylabel('wcss')
plt.show()
```



Observation:

we get two elbows at 3 and 4. As per elbow method definition we take the last one, so our no. of cluster will be 4.

In [20]:
```
km = KMeans(n_clusters =4, init= 'k-means++', max_iter=500, n_init=10, random_state
y_means= km.fit_predict(z)

a = df['label']
y_means = pd.DataFrame(y_means)
w = pd.concat([y_means, a], axis=1)
w =w.rename(columns= {0:'cluster'})

#Check Cluster of each group

for i in range(0,4): #for 4 clusters 0,1,2,3
    print('Crops is cluster', i, w[w['cluster']==i]['label'].unique())
    print('-------------------------------------------------------------------------------
```

```
Crops is cluster 0 ['grapes' 'apple']
--------------------------------------------------------------------------------
-----
Crops is cluster 1 ['maize' 'chickpea' 'kidneybeans' 'pigeonpeas' 'mothbeans' 'mun
gbean'
 'blackgram' 'lentil' 'pomegranate' 'mango' 'orange' 'papaya' 'coconut']
--------------------------------------------------------------------------------
-----
Crops is cluster 2 ['maize' 'banana' 'watermelon' 'muskmelon' 'papaya' 'cotton' 'c
offee']
--------------------------------------------------------------------------------
-----
Crops is cluster 3 ['rice' 'pigeonpeas' 'papaya' 'coconut' 'jute' 'coffee']
--------------------------------------------------------------------------------
-----
```

Observation:

some of the crops are more than in one cluster. This is known as soft clustering

We want our crops to be in single cluster so next step we are going to do hard clustering

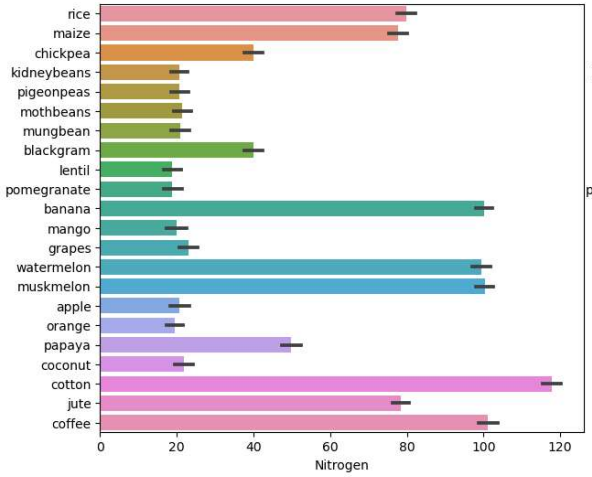
In hard clustering we can able to see each data points are completely in one cluster or not

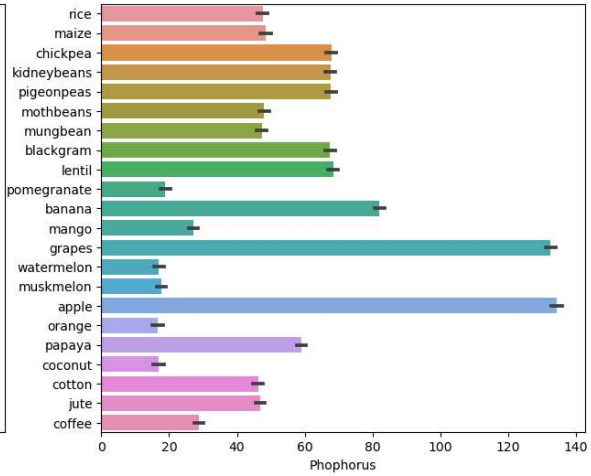
In [21]:
```
for i in range(0,4):
    counts=w[w['cluster']==i]['label'].value_counts()
    d = w.loc[w['label'].isin(counts.index[counts>=50])]
    d=d['label'].value_counts()
    print('Crops in Cluster', i,':', list(d.index))
    print('--------------------------------------------------------------------------------')
```

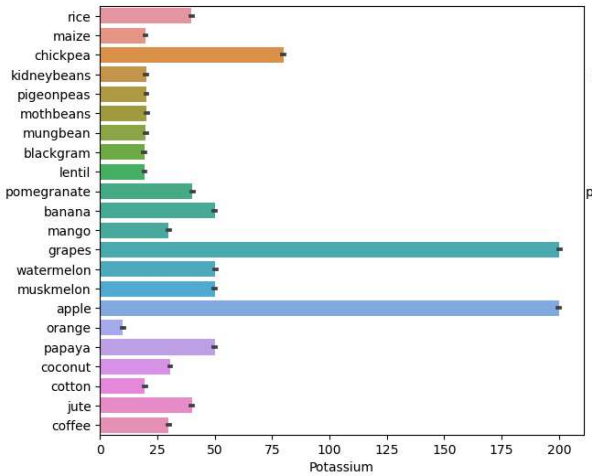
```
Crops in Cluster 0 : ['grapes', 'apple']
--------------------------------------------------------------------------------
Crops in Cluster 1 : ['chickpea', 'kidneybeans', 'mothbeans', 'mungbean', 'blackgr
am', 'lentil', 'pomegranate', 'mango', 'orange']
--------------------------------------------------------------------------------
Crops in Cluster 2 : ['maize', 'banana', 'watermelon', 'muskmelon', 'cotton']
--------------------------------------------------------------------------------
Crops in Cluster 3 : ['rice', 'pigeonpeas', 'papaya', 'coconut', 'jute', 'coffee']
--------------------------------------------------------------------------------
```

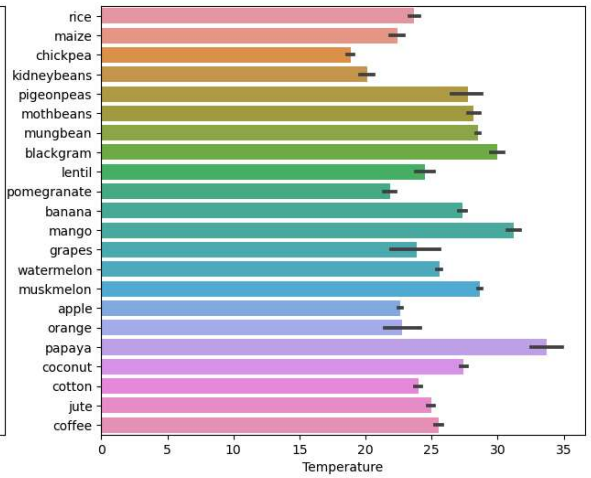
In [22]:
```
plt.figure(figsize=(15, 28))
plt.subplot(4,2,1)
sns.barplot(x=df['N'], y=df['label'])
plt.ylabel(' ')
plt.xlabel('Nitrogen')
plt.subplot(4,2,2)
sns.barplot(x=df['P'], y=df['label'])
plt.ylabel(' ')
plt.xlabel('Phophorus')
plt.subplot(4,2,3)
sns.barplot(x=df['K'], y=df['label'])
plt.ylabel(' ')
plt.xlabel('Potassium')
plt.subplot(4,2,4)
sns.barplot(x=df['temperature'], y=df['label'])
plt.ylabel(' ')
plt.xlabel('Temperature')
plt.subplot(4,2,5)
sns.barplot(x=df['humidity'], y=df['label'])
plt.ylabel(' ')
plt.xlabel('Humidity')
plt.subplot(4,2,6)
sns.barplot(x=df['ph'], y=df['label'])
plt.ylabel(' ')
plt.xlabel('pH')
plt.subplot(4,2,7)
sns.barplot(x=df['rainfall'], y=df['label'])
plt.ylabel(' ')
plt.xlabel('rainfall')
#apply for loop
```

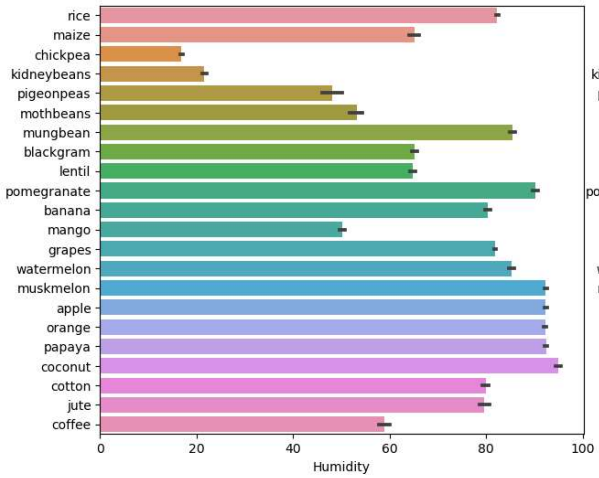
Out[22]:
```
Text(0.5, 0, 'rainfall')
```

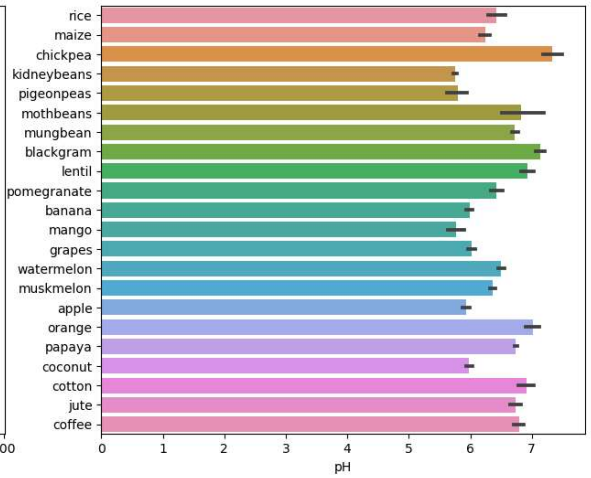

rice
maize
chickpea
kidneybeans
pigeonpeas
mothbeans
mungbean
blackgram
lentil
pomegranate
banana
mango
grapes
watermelon
muskmelon
apple
orange
papaya
coconut
cotton
jute
coffee
0
20
40
60
80
100
120
Nitrogen

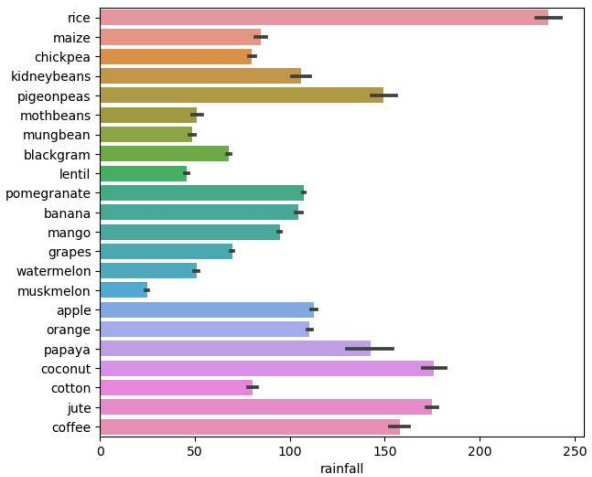

rice
maize
chickpea
kidneybeans
pigeonpeas
mothbeans
mungbean
blackgram
lentil
pomegranate
banana
mango
grapes
watermelon
muskmelon
apple
orange
papaya
coconut
cotton
jute
coffee
0
20
40
60
80
100
120
140
Phophorus


rice
maize
chickpea
kidneybeans
pigeonpeas
mothbeans
mungbean
blackgram
lentil
pomegranate
banana
mango
grapes
watermelon
muskmelon
apple
orange
papaya
coconut
cotton
jute
coffee
0
25
50
75
100
125
150
175
200
Potassium


rice
maize
chickpea
kidneybeans
pigeonpeas
mothbeans
mungbean
blackgram
lentil
pomegranate
banana
mango
grapes
watermelon
muskmelon
apple
orange
papaya
coconut
cotton
jute
coffee
0
5
10
15
20
25
30
35
Temperature


rice
maize
chickpea
kidneybeans
pigeonpeas
mothbeans
mungbean
blackgram
lentil
pomegranate
banana
mango
grapes
watermelon
muskmelon
apple
orange
papaya
coconut
cotton
jute
coffee
0
20
40
60
80
100
Humidity


rice
maize
chickpea
kidneybeans
pigeonpeas
mothbeans
mungbean
blackgram
lentil
pomegranate
banana
mango
grapes
watermelon
muskmelon
apple
orange
papaya
coconut
cotton
jute
coffee
0
1
2
3
4
5
6
7
pH


rice
maize
chickpea
kidneybeans
pigeonpeas
mothbeans
mungbean
blackgram
lentil
pomegranate
banana
mango
grapes
watermelon
muskmelon
apple
orange
papaya
coconut
cotton
jute
coffee
0
50
100
150
200
250
rainfall

Observation

1. Cotton requires high amount of Nitrogen among all
2. Grapes and Apple requires very high amount of phosphorus and Potassium
3. least amount of potassium is the favorable condition of Orange to grow
4. Papaya requires more than 30 degree to grow well whereas others required <= 30 degree
5. chickpea and kidneybeans humidity requires very less humidity to grow
6. All crops require more than pH value of 5 to grow
7. Rice requires very heavy rainfall (more than 200mm) where the muskmelon requires the least

In [23]:
```
#Let split the dataset for predective modelling

x = df.drop(['label'], axis=1)
x.head()
```

Out[23]:

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

In [24]:
```
y = df['label']
y.head()
```

Out[24]:
```
0    rice
1    rice
2    rice
3    rice
4    rice
Name: label, dtype: object
```

In [26]:
```
#create training and testing sets  for validation of results

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_stat
```

80% data will trained and 20% data will use for testing the model

In [27]:
```
print('shape of x_train',x_train.shape )
print('shape of x_test',x_test.shape )
print('shape of y_train',y_train.shape )
print('shape of y_test',y_test.shape )
```

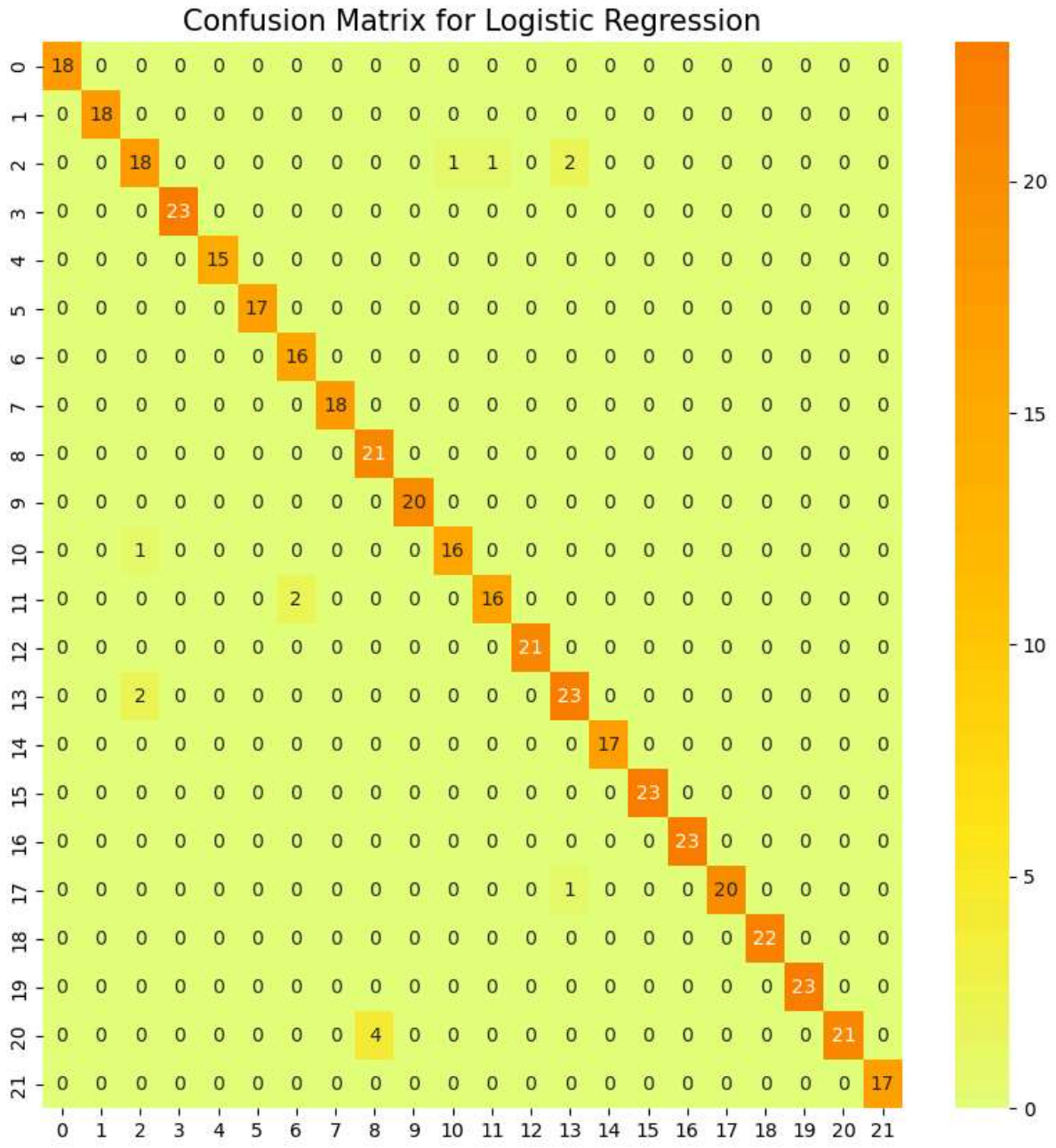
```
shape of x_train (1760, 7)
shape of x_test (440, 7)
shape of y_train (1760,)
shape of y_test (440,)
```

In [28]:
```
# creating a predective model

from sklearn.linear_model import LogisticRegression
```

```
model=LogisticRegression()
model.fit(x_train,y_train)
y_pred=model.predict(x_test)
```

In [29]:
```
from sklearn.metrics import accuracy_score,confusion_matrix
```

In [30]:
```
#print the confusion matrix first

plt.rcParams['figure.figsize']=(10,10)
cm=confusion_matrix(y_test,y_pred)
sns.heatmap(cm, annot=True, cmap='Wistia')
plt.title("Confusion Matrix for Logistic Regression", fontsize=15)
plt.show()
```



In [31]:
```
#Classification  report
from sklearn.metrics import classification_report
cr=classification_report(y_test,y_pred)
print(cr)
```

	precision	recall	f1-score	support
apple	1.00	1.00	1.00	18
banana	1.00	1.00	1.00	18
blackgram	0.86	0.82	0.84	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	0.89	1.00	0.94	16
grapes	1.00	1.00	1.00	18
jute	0.84	1.00	0.91	21
kidneybeans	1.00	1.00	1.00	20
lentil	0.94	0.94	0.94	17
maize	0.94	0.89	0.91	18
mango	1.00	1.00	1.00	21
mothbeans	0.88	0.92	0.90	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	0.95	0.98	21
pigeonpeas	1.00	1.00	1.00	22
pomegranate	1.00	1.00	1.00	23
rice	1.00	0.84	0.91	25
watermelon	1.00	1.00	1.00	17
accuracy			0.97	440
macro avg	0.97	0.97	0.97	440
weighted avg	0.97	0.97	0.97	440

In [32]:
```
#predict

pred=model.predict((np.array([[90,
                               50,
                               60,
                               20,
                               90,
                               6,
                               220]])))
print('The suggested crop for given climatic condition:',pred)
```

```
The suggested crop for given climatic condition: ['jute']
```

In [33]:
```
y_pred1 = model.predict(x_test)
print("Accuracy Score of LogisticRegression:",accuracy_score(y_test,y_pred1))
```

```
Accuracy Score of LogisticRegression: 0.9681818181818181
```

In [34]:
```
#Let check other Algorithm to check which model is best

from sklearn.neighbors import KNeighborsClassifier
clf_knn = KNeighborsClassifier(n_neighbors=3)
clf_knn.fit(x_train,y_train)
y_pred1 = clf_knn.predict(x_test)
print("Accuracy Score of KNN:",accuracy_score(y_test,y_pred1))

from sklearn.svm import SVC
clf_svc = SVC()
clf_svc.fit(x_train,y_train)
y_pred2 = clf_svc.predict(x_test)
print("Accuracy Score of SVC:",accuracy_score(y_test,y_pred2))

from sklearn.tree import DecisionTreeClassifier
clf_dtc = DecisionTreeClassifier(criterion='entropy',random_state=7)
clf_dtc.fit(x_train,y_train)
```

```
y_pred3 = clf_dtc.predict(x_test)
print("Accuracy Score of decision tree:",accuracy_score(y_test,y_pred3))

from sklearn.ensemble import RandomForestClassifier
clf_rfc = RandomForestClassifier(random_state=1)
clf_rfc.fit(x_train, y_train)
y_pred4 = clf_rfc.predict(x_test)
print("Accuracy Score of Random Forest:",accuracy_score(y_test,y_pred4))
```

```
Accuracy Score of KNN: 0.9772727272727273
Accuracy Score of SVC: 0.9772727272727273
Accuracy Score of decision tree: 0.9931818181818182
Accuracy Score of Random Forest: 0.9977272727272727
```

Random forest gave us the best results

In [35]:
```
y_train_pred = clf_rfc.predict(x_train)
print("Accuracy Score of Random Forest:",accuracy_score(y_train,y_train_pred))
```

```
Accuracy Score of Random Forest: 1.0
```

In [36]:
```
output = pd.DataFrame({'Real_class': y_test, 'Predicted_class': y_pred4})
output.head()
```

Out[36]:

	Real_class	Predicted_class
1320	watermelon	watermelon
1367	watermelon	watermelon
1291	grapes	grapes
264	chickpea	chickpea
728	blackgram	blackgram

In [37]:
```
input = np.array([[90,42,43,20.879744,82.002744,6.502985,202.935536]])
clf_rfc.predict(input)
```

Out[37]:
```
array(['rice'], dtype=object)
```

In [38]:
```
output.to_csv('Optimizing Agricultural Production1.csv', index=False)
print("Submission was successfully saved!")
```

```
Submission was successfully saved!
```

In [ ]: