Import The following packages

In [8]:

```
import numpy as np # for complex mathematical problems
import pandas as pd # for data analysis and manipulation
import seaborn as sns # for data visualization
from matplotlib import pyplot as plt
from ipywidgets import interact # for interactivity
```

Retrieving the dataset

In [9]:

```
data =pd.read_csv('E:/archive/data.csv')
```

In [10]:

```
# lets check the head of the Dataset
data.head()
```

Out[10]:

	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

In [11]:

```
print("The shape of the Dataset is:",data.shape)
```

```
The shape of the Dataset is: (2200, 8)
```

In [12]:

```
# To check the null value in the dataset
data.isnull().sum()
```

Out[12]:

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

- Fill-Na function is sued to replace these missing values such as mean, median, mode.
- Na means not available
- Pandas have function like fill_na, drop-Na to treat missing values

In [13]:

```
# Lets check the crops present in ths dataset
```

```
data['label'].value_counts()
```

Out[13]:
```
grapes         100
mungbean       100
pigeonpeas     100
coconut        100
blackgram      100
lentil         100
coffee         100
banana         100
chickpea       100
mango          100
papaya         100
cotton         100
mothbeans      100
muskmelon      100
watermelon     100
rice           100
orange         100
pomegranate    100
apple          100
maize          100
jute           100
kidneybeans    100
Name: label, dtype: int64
```

In [14]:
```
# Lets check the summary
print("Average Ratio of Nitrogen in the soil: {0:.2f}".format(data['N'].mean()))
print("Average Ratio of Phosphorous in the soil: {0:.2f}".format(data['P'].mean()))
print("Average Ratio of Potassium in the soil: {0:.2f}".format(data['K'].mean()))
print("Average Temperature in celcius: {0:.2f}".format(data['temperature'].mean()))
print("Average Relative Humidity in % : {0:.2f}".format(data['humidity'].mean()))
print("Average Ph value of the soil: {0:.2f}".format(data['ph'].mean()))
print("Average Rainfall in mm: {0:.2f}".format(data['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 Temperature in celcius: 25.62
Average Relative Humidity in % : 71.48
Average Ph value of the soil: 6.47
Average Rainfall in mm: 103.46
```

In [15]:
```
# Lets check the summary Statistics for each of the crops
@interact
def summary(crops = list(data['label'].value_counts().index)):
    x =data[data['label'] ==crops]
    print("---------------------------")
    print("Statistics for Nitrogen")
    print("Minimum Nitrigen required:",x['N'].min())
    print("Average Nitrigen required:",x['N'].mean())
    print("Maximum Nitrigen required:",x['N'].max())
    print("---------------------------")
    print("Statistics for Phosphorous")
    print("Minimum Phosphorous required:",x['P'].min())
    print("Average Phosphorous required:",x['P'].mean())
    print("Maximum Phosphorous required:",x['P'].max())
    print("---------------------------")
    print("Statistics for Potassium")
    print("Minimum Potassium required:",x['K'].min())
    print("Average Potassium required:",x['K'].mean())
```

```
    print("Maximum Potassium required:",x['K'].max())
    print("---------------------------")
    print("Statistics for Temperature")
    print("Minimum Temperature required:",x['temperature'].min())
    print("Average Temperature required:",x['temperature'].mean())
    print("Maximum Temperature required:",x['temperature'].max())
    print("---------------------------")
    print("Statistics for Humidity")
    print("Minimum Humidity required:",x['humidity'].min())
    print("Average Humidityrequired:",x['humidity'].mean())
    print("Maximum Humidity required:",x['humidity'].max())
    print("---------------------------")
    print("Statistics for PH")
    print("Minimum PH required:",x['ph'].min())
    print("Average PHrequired:",x['ph'].mean())
    print("Maximum PH required:",x['ph'].max())
    print("---------------------------")
    print("Statistics for Rainfall")
    print("Minimum Rainfall required:",x['rainfall'].min())
    print("Average Rainfall required:",x['rainfall'].mean())
    print("Maximum rainfall required:",x['rainfall'].max())
    print("---------------------------")
```

In [16]:

```
data['label'].value_counts()
```

Out[16]:

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

In [17]:

```
# Lets compare the Average for eachcrops with average conditions
@ interact
def compare(conditions =['N','P','K','temperature','ph','humidity','rainfall']):
    print("Average Value for",conditions, "is {0:.2f}".format(data[conditions].mean()))
    print("----------------------------------------------")
    print("Maize:{0:.2f}".format(data[(data['label'] =='maize')][conditions].mean()))
    print("Mothbeans:{0:.2f}".format(data[(data['label'] =='mothbeans')][conditions].me
```

```
    print("Apple:{0:.2f}".format(data[(data['label'] =='apple')][conditions].mean()))
    print("Grapes:{0:.2f}".format(data[(data['label'] =='grapes')][conditions].mean()))
    print("Muskmelon:{0:.2f}".format(data[(data['label'] =='muskmelon')][conditions].me
    print("Lentil:{0:.2f}".format(data[(data['label'] =='lentil')][conditions].mean()))
    print("Banana:{0:.2f}".format(data[(data['label'] =='banana')][conditions].mean()))
    print("Jute:{0:.2f}".format(data[(data['label'] =='jute')][conditions].mean()))
    print("Rice:{0:.2f}".format(data[(data['label'] =='rice')][conditions].mean()))
    print("Pigeonpeas:{0:.2f}".format(data[(data['label'] =='pigeonpeas')][conditions].
    print("Kidneybeans:{0:.2f}".format(data[(data['label'] =='kidneybeans')][conditions
    print("Coffee:{0:.2f}".format(data[(data['label'] =='coffee')][conditions].mean()))
    print("Papaya:{0:.2f}".format(data[(data['label'] =='papaya')][conditions].mean()))
    print("Mango:{0:.2f}".format(data[(data['label'] =='mango')][conditions].mean()))
    print("Chickpea:{0:.2f}".format(data[(data['label'] =='chickpea')][conditions].mean
    print("Orange:{0:.2f}".format(data[(data['label'] =='orange')][conditions].mean()))
    print("Mungbean:{0:.2f}".format(data[(data['label'] =='mungbean')][conditions].mean
    print("Blackgram:{0:.2f}".format(data[(data['label'] =='blackgram')][conditions].me
    print("Pomegrenate:{0:.2f}".format(data[(data['label'] =='pomegranate')][conditions
    print("Cotton:{0:.2f}".format(data[(data['label'] =='cotton')][conditions].mean()))
    print("Watermelon:{0:.2f}".format(data[(data['label'] =='watermelon')][conditions].
```

In [18]:

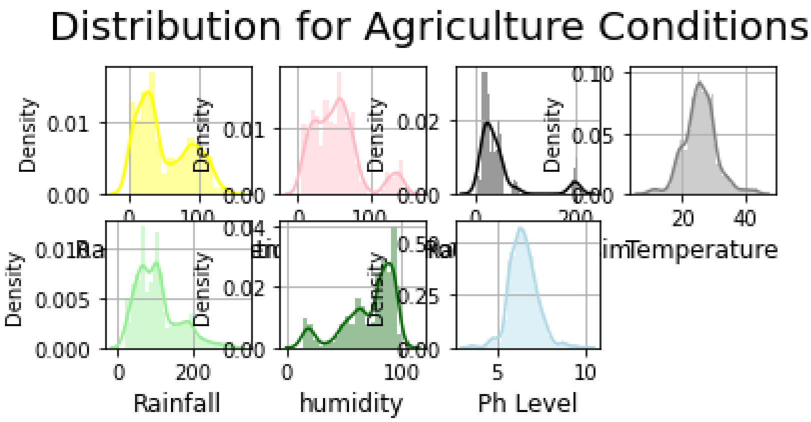
```
#   Lets make this function more promptive
@interact
def compare(conditions=['N','P','K','temperature','ph','humidity','rainfall']):
    print("     Average value of",conditions,"is: {0:.2f}".format(data[conditions].mean
    print("Crops which require greater than average",conditions,'\n')
    print(data[data[conditions] > data[conditions].mean()]['label'].unique())
    print("---------------------------------------")
    print("Crops which require less than average",conditions,'\n')
    print(data[data[conditions] <data[conditions].mean()]['label'].unique())
```

In [19]:

```
plt.subplot(3,4,1)
sns.distplot(data['N'], color ='yellow')
plt.xlabel('Ratio of Nitrogen',fontsize =12)
plt.grid()

plt.subplot(3,4,2)
sns.distplot(data['P'],color ='lightpink')
plt.xlabel('Ratio of Phosphorous',fontsize =12)
plt.grid()

plt.subplot(3,4,3)
sns.distplot(data['K'],color ='black')
plt.xlabel('Ratio of Potassim',fontsize =12)
plt.grid()

plt.subplot(3,4,4)
sns.distplot(data['temperature'],color ='grey')
plt.xlabel('Temperature',fontsize =12)
plt.grid()

plt.subplot(3,4,5)
sns.distplot(data['rainfall'],color ='lightgreen')
plt.xlabel('Rainfall',fontsize=12)
plt.grid()
```

```
plt.subplot(3,4,6)
sns.distplot(data['humidity'],color ='darkgreen')
plt.xlabel('humidity',fontsize=12)
plt.grid()


plt.subplot(3,4,7)
sns.distplot(data['ph'],color ='lightblue')
plt.xlabel('Ph Level',fontsize=12)
plt.grid()


plt.suptitle('Distribution for Agriculture Conditions',fontsize =20)
plt.show()
```

```
C:\Users\asus\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\asus\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\asus\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\asus\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\asus\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\asus\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\asus\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



In [20]:

```
# Lets find out something more
print("Some intresting fact:")
print("---------------------------------------")
print("Crops which require very High Ratio of Nitrogen Content in soil:",data[data['N']
print("Crops which requires very High Ratio of Phosphorous in the soil:",data[data['P']
print("Crops which requires Very High Ratio of Potassiu in the soil",data[data['K'] > 2
print("Crops which requires very High Rainfall:",data[data['rainfall'] >200]['label'].u
print("Crops which requires very low Rainfall:",data[data['rainfall'] <60]['label'].uni
print("Crops which rquires very High Temperature",data[data['temperature'] >40]['label'
print("Crops which requires very Low Temperature",data[data['temperature']<10]['label']
```

```
Some intresting fact:
---------------------------------------
Crops which require very High Ratio of Nitrogen Content in soil: ['cotton']
Crops which requires very High Ratio of Phosphorous in the soil: ['grapes' 'apple']
Crops which requires Very High Ratio of Potassiu in the soil ['grapes' 'apple']
Crops which requires very High Rainfall: ['rice' 'papaya' 'coconut']
Crops which requires very low Rainfall: ['mothbeans' 'mungbean' 'lentil' 'watermelon' 'm
uskmelon' 'papaya']
Crops which rquires very High Temperature ['grapes' 'papaya']
Crops which requires very Low Temperature ['grapes']
```

In [21]:

```
### Lets understand which crops can only be grown in summer, winter and rainy season
print("SUMMER CROPS")
print(data[(data['temperature'] >40) & (data['humidity'] > 50)]['label'].unique())
print("--------------------------------------------------")
print("WINTER CROPS")
print(data[(data['temperature'] < 20) & (data['humidity'] > 30)]['label'].unique())
print("--------------------------------------------------")
print("RAINY CROPS")
print(data[(data['rainfall'] > 200) & (data['humidity']> 30)]['label'].unique())
```

```
SUMMER CROPS
['grapes' 'papaya']
--------------------------------------------------
WINTER CROPS
['maize' 'pigeonpeas' 'lentil' 'pomegranate' 'grapes' 'orange']
--------------------------------------------------
RAINY CROPS
['rice' 'papaya' 'coconut']
```

In [22]:

```
from sklearn.cluster import KMeans
# removing the labels column
x=data.drop(['label'], axis =1)
# selcting all the values of data
```

```
x=x.values
# checking the shape
print(x.shape)
```

(2200, 7)

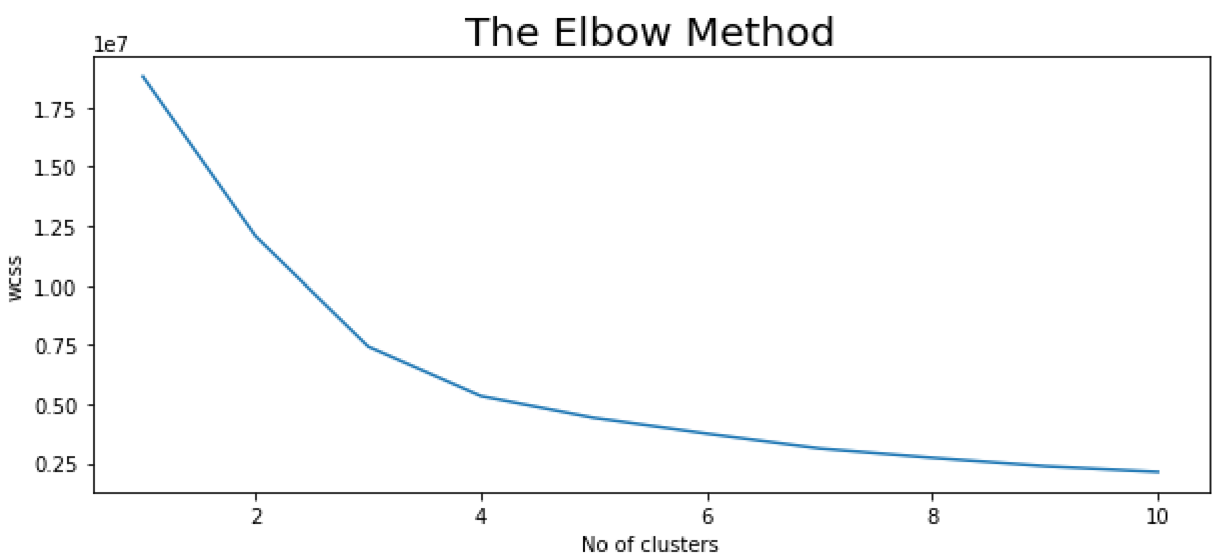
In [23]:

```
x.shape
```

Out[23]: (2200, 7)

In [34]:

```
# Lets determine the optimum no of clusters within the clusters
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_state=0)
    km.fit(x)
    wcss.append(km.inertia_)

# Lets plot the result
plt.plot(range(1,11),wcss)
plt.title('The Elbow Method',fontsize =20)
plt.xlabel('No of clusters')
plt.ylabel('wcss')
plt.show()
```



In [37]:

```
# Lets implement the K Means algroithm to perform clustering analysis
km =KMeans(n_clusters = 4,init ='k-means++',max_iter =300,n_init =10,random_state =0)
y_means =km.fit_predict(x)
# Lets find out the Results
a =data['label']
y_means =pd.DataFrame(y_means)
z =pd.concat([y_means,a], axis =1)
z =z.rename(columns = {0:'cluster'})
# Lets check the clusters of each crops
print("Lets check the results after applying the K means clustering analysis \n")
print("Crops in First clustering:",z[z['cluster'] == 0]['label'].unique())
print("---------------------------------------------------")
print("Crops in Second Cluster:",z[z['cluster'] == 1]['label'].unique())
print("---------------------------------------------------")
```

```
print("Crops in Third Cluster:",z[z['cluster'] == 2]['label'].unique())
print("---------------------------------------------------")
print("Crops in Fourth Cluster:",z[z['cluster'] == 3]['label'].unique())
```

```
Lets check the results after applying the K means clustering analysis

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

In [39]:

```
# Lets split the dataset for predectvive Modeling
y=data['label']
x =data.drop(['label'],axis =1)
print("shape of x:",x.shape)
print("shape of y:",y.shape)
```

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

In [41]:

```
# Lets try to 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_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 [43]:

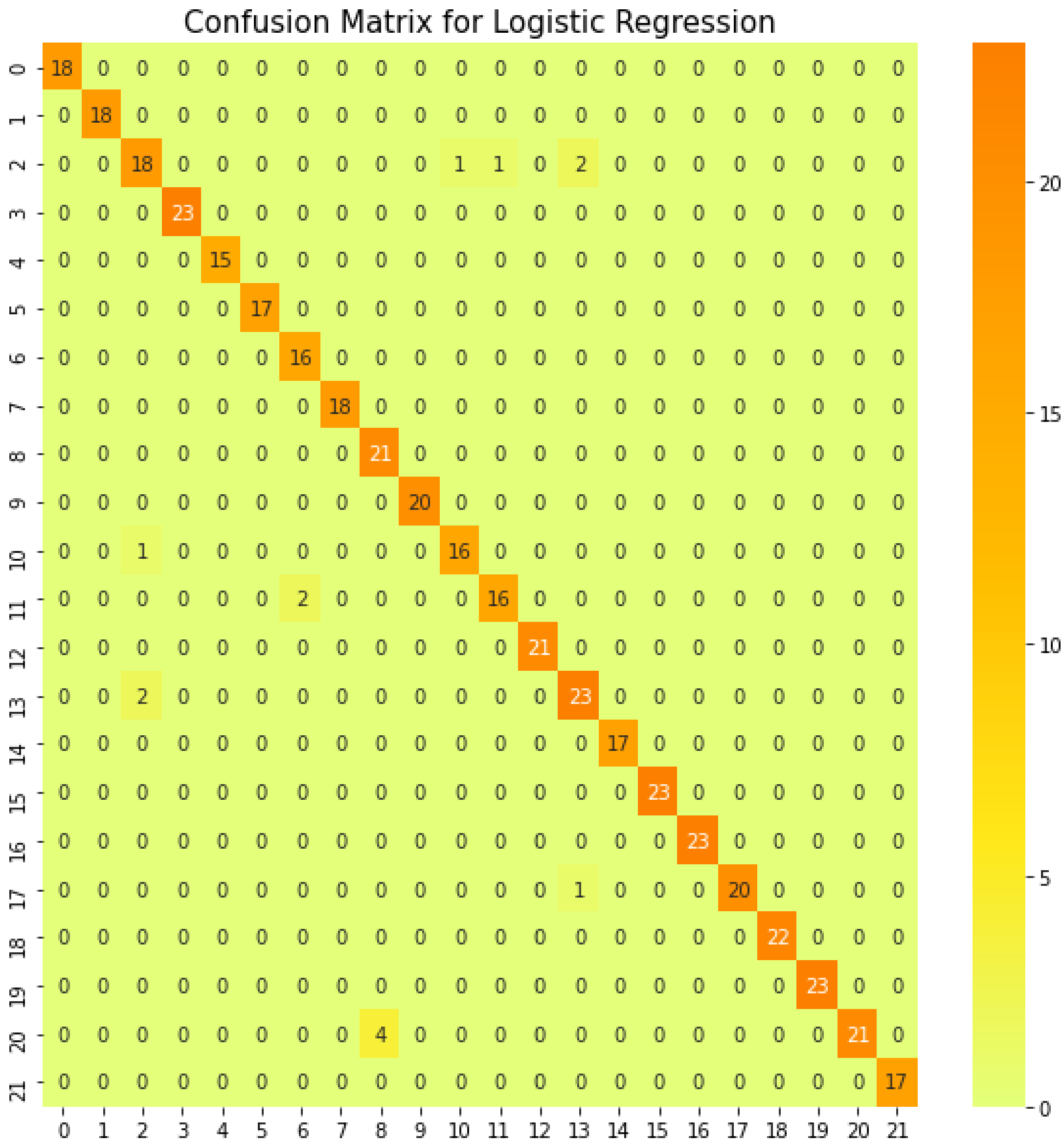
```
# Lets create a Predective Model
from sklearn.linear_model import LogisticRegression
model =LogisticRegression()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
```

```
C:\Users\asus\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converg
enceWarning: 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(
```

In [49]:

```
# Lets evaluate the Model Performance
from sklearn.metrics import confusion_matrix
```

```
# Lets 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 [51]:

```
# Lets print the 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 [55]:

```
data.head(100)
```

Out[55]:

	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
...	...	...	...	...	...	...	...	...
95	88	46	42	22.683191	83.463583	6.604993	194.265172	rice
96	93	47	37	21.533463	82.140041	6.500343	295.924880	rice
97	60	55	45	21.408658	83.329319	5.935745	287.576694	rice
98	78	35	44	26.543481	84.673536	7.072656	183.622266	rice
99	65	37	40	23.359054	83.595123	5.333323	188.413665	rice

100 rows × 8 columns

In [54]:

```
prediction = model.predict((np.array([[70,
                                       40,
                                       40,
                                       25,
                                       80,
                                       7,
                                       250]])))
print("The suggested crop for given cliatic condition is:",prediction)
```

```
The suggested crop for given cliatic condition is: ['rice']
```

In [ ]: