Importing libraries

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

Data Processing

Removing unnecessary feature

In [ ]:
```
data = pd.read_csv('./crop_dataset.csv')
data  = data.drop(data.columns[0], axis=1) # removing uncessary feature.
```

Data Normalization

In [ ]:
```
from sklearn import preprocessing
# normalization is done only for features
normalized = preprocessing.normalize(data[["N","Fe","Mn","S","Mg","Ca","P","K
data[["N","Fe","Mn","S","Mg","C","P","K","temperature","humidity","ph","rainf
data.head()
```

Out[ ]:

	N	Fe	Mn	S	Mg	C	P	K	ten
0	0.050188	0.012756	0.979098	0.000267	0.055449	0.139434	0.023421	0.023979	
1	0.069924	0.021697	0.953846	0.007388	0.095366	0.209694	0.033077	0.033417	
2	0.053091	0.017731	0.978423	0.006981	0.017121	0.144084	0.025454	0.025380	
3	0.161523	0.036713	0.774581	0.024525	0.188933	0.404721	0.078478	0.077238	
4	0.076145	0.021868	0.952423	0.005924	0.077242	0.204697	0.037487	0.036422	

Dataset preview

In [ ]:
```
data.head()  # 1ast 5 entries of our dataset.
```

Out[ ]:

	N	Fe	Mn	S	Mg	C	P	K	ten
0	0.050188	0.012756	0.979098	0.000267	0.055449	0.139434	0.023421	0.023979	
1	0.069924	0.021697	0.953846	0.007388	0.095366	0.209694	0.033077	0.033417	
2	0.053091	0.017731	0.978423	0.006981	0.017121	0.144084	0.025454	0.025380	
3	0.161523	0.036713	0.774581	0.024525	0.188933	0.404721	0.078478	0.077238	
4	0.076145	0.021868	0.952423	0.005924	0.077242	0.204697	0.037487	0.036422	

In [ ]:
```
data.tail()   #last 5 entries of our data set.
```

Out[ ]:

	N	Fe	Mn	S	Mg	C	P	K
69713	0.291729	0.070051	0.229477	0.012203	0.391048	0.717454	0.054548	0.084239
69714	0.082111	0.021277	0.966353	0.009690	0.021561	0.207856	0.015071	0.023704
69715	0.071872	0.018828	0.977051	0.009072	0.078907	0.147462	0.012943	0.020743
69716	0.235415	0.050005	0.588021	0.039484	0.129464	0.670674	0.041573	0.067926
69717	0.234981	0.060675	0.689271	0.011144	0.135759	0.564005	0.040670	0.067783

Displaying data information

In [ ]:
```
data.info()    #print the information about our dataset.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 69718 entries, 0 to 69717
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   N            69718 non-null  float64
 1   Fe           69718 non-null  float64
 2   Mn           69718 non-null  float64
 3   S            69718 non-null  float64
 4   Mg           69718 non-null  float64
 5   C            69718 non-null  float64
 6   P            69718 non-null  float64
 7   K            69718 non-null  float64
 8   temperature  69718 non-null  float64
 9   humidity     69718 non-null  float64
 10  ph           69718 non-null  float64
 11  rainfall     69718 non-null  float64
 12  label        69718 non-null  object 
dtypes: float64(12), object(1)
memory usage: 6.9+ MB
```

Displaying data shape

In [ ]:
```
data.shape    # we have 69,718 entries in our dataset.
```

Out[ ]:
```
(69718, 13)
```

Data description

In [ ]:
```
data.describe()    # getting information about each column
```

Out[ ]:

	N	Fe	Mn	S	Mg	
count	69718.000000	69718.000000	69718.000000	6.971800e+04	69718.000000	69718.0

mean	0.062451	0.031067	0.862879	1.243721e-02	0.092921	0.3
std	0.064800	0.019773	0.199598	1.156890e-02	0.084016	0.1
min	0.000000	0.009755	0.050732	3.408135e-07	0.000003	0.0
25%	0.018479	0.016081	0.851421	4.532166e-03	0.034272	0.1
50%	0.042192	0.023364	0.953833	9.092562e-03	0.068785	0.2
75%	0.079275	0.040796	0.978432	1.612768e-02	0.121609	0.4
max	0.448990	0.126427	0.993325	8.435813e-02	0.516173	0.9

Displaying data columns

In [ ]:
```
data.columns   # we have 7 features in our dataset
```

Out[ ]:
```
Index(['N', 'Fe', 'Mn', 'S', 'Mg', 'C', 'P', 'K', 'temperature', 'humidity',
       'ph', 'rainfall', 'label'],
      dtype='object')
```

Calculating total unique value

In [ ]:
```
data.nunique()   # printing unique count of each each feature.
```

Out[ ]:
```
N              69692
Fe             69718
Mn             69718
S              69718
Mg             69718
C              69718
P              69718
K              69718
temperature    69718
humidity       69718
ph             69718
rainfall       69718
label             22
dtype: int64
```

Checking for null value in data

In [ ]:
```
data.isnull().sum()  # we don't have any null values
```

Out[ ]:
```
N              0
Fe             0
Mn             0
S              0
Mg             0
C              0
P              0
K              0
temperature    0
humidity       0
ph             0
rainfall       0
```

```
label          0
dtype: int64
```

Checking duplicate values in data

In [ ]:
```
data.duplicated().sum() # find duplicate entries if any
```

Out[ ]:
```
0
```

In [ ]:
```
data['label'].unique()  # printing unique label classes 0
```

Out[ ]:
```
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 [ ]:
```
data['label'].nunique()  # printing count of unique class
```

Out[ ]:
```
22
```

Relationship Analysis

In [ ]:
```
#Relationship analysis?
```

Correlation

In [ ]:
```
correlation = data.corr()  # correlation of features with each other
print(correlation)
```

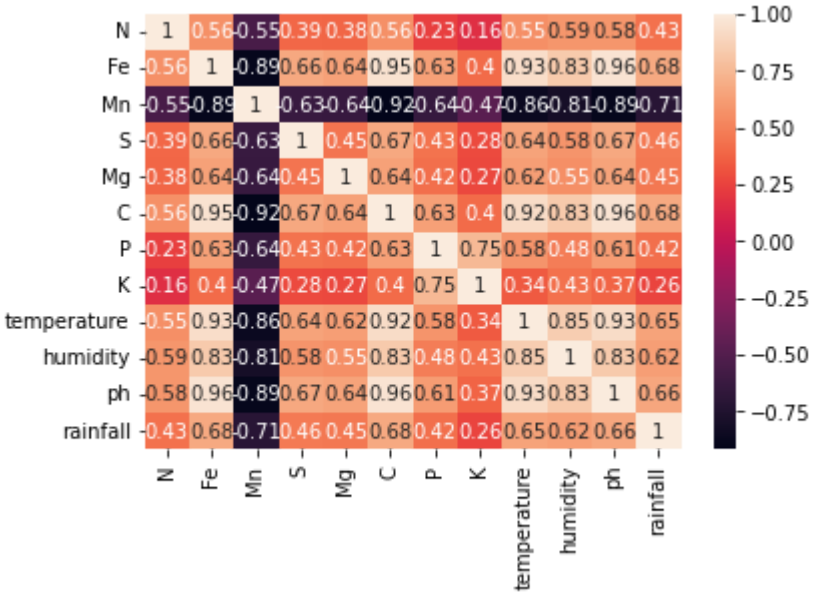
```
                   N        Fe        Mn         S        Mg         C  \
N           1.000000  0.564121 -0.552473  0.389372  0.379942  0.563732
Fe          0.564121  1.000000 -0.890520  0.663477  0.639808  0.952897
Mn         -0.552473 -0.890520  1.000000 -0.625233 -0.638906 -0.916537
S           0.389372  0.663477 -0.625233  1.000000  0.445473  0.665578
Mg          0.379942  0.639808 -0.638906  0.445473  1.000000  0.639326
C           0.563732  0.952897 -0.916537  0.665578  0.639326  1.000000
P           0.232860  0.630610 -0.642107  0.434515  0.421120  0.630855
K           0.163991  0.403427 -0.471210  0.277838  0.273394  0.403707
temperature 0.552408  0.925195 -0.857920  0.644738  0.618857  0.923372
humidity    0.585983  0.831017 -0.806937  0.578004  0.552993  0.830564
ph          0.581168  0.956833 -0.887301  0.666812  0.640751  0.955655
rainfall    0.427346  0.676118 -0.707085  0.462914  0.454070  0.677137

                   P         K  temperature  humidity        ph  rainfall
N           0.232860  0.163991     0.552408  0.585983  0.581168  0.427346
Fe          0.630610  0.403427     0.925195  0.831017  0.956833  0.676118
Mn         -0.642107 -0.471210    -0.857920 -0.806937 -0.887301 -0.707085
S           0.434515  0.277838     0.644738  0.578004  0.666812  0.462914
Mg          0.421120  0.273394     0.618857  0.552993  0.640751  0.454070
C           0.630855  0.403707     0.923372  0.830564  0.955655  0.677137
P           1.000000  0.751659     0.576846  0.477527  0.613030  0.417207
K           0.751659  1.000000     0.340489  0.428696  0.374674  0.258114
temperature 0.576846  0.340489     1.000000  0.846277  0.925075  0.646037
```

humidity	0.477527	0.428696	0.846277	1.000000	0.833555	0.619769
ph	0.613030	0.374674	0.925075	0.833555	1.000000	0.659021
rainfall	0.417207	0.258114	0.646037	0.619769	0.659021	1.000000

Plotting heatmap

In [ ]:
```
sns.heatmap(correlation, xticklabels=correlation.columns, yticklabels=correla
```

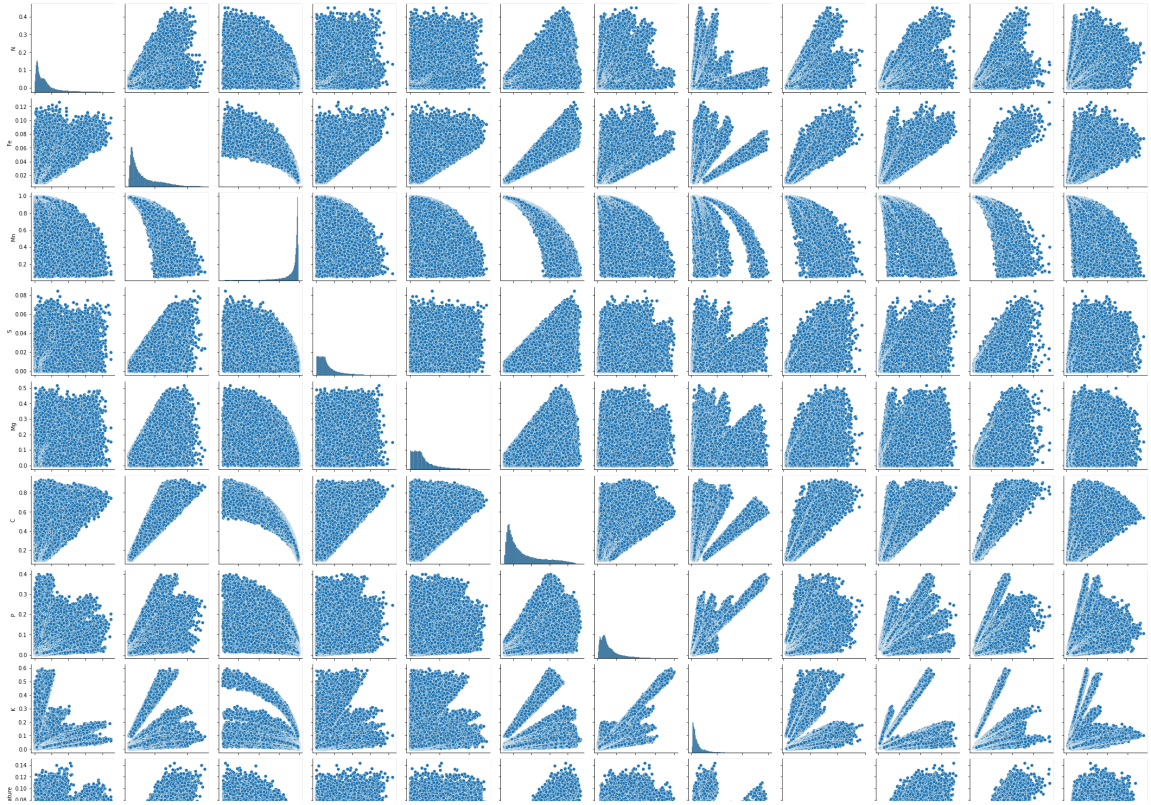
Out[ ]: <AxesSubplot:>

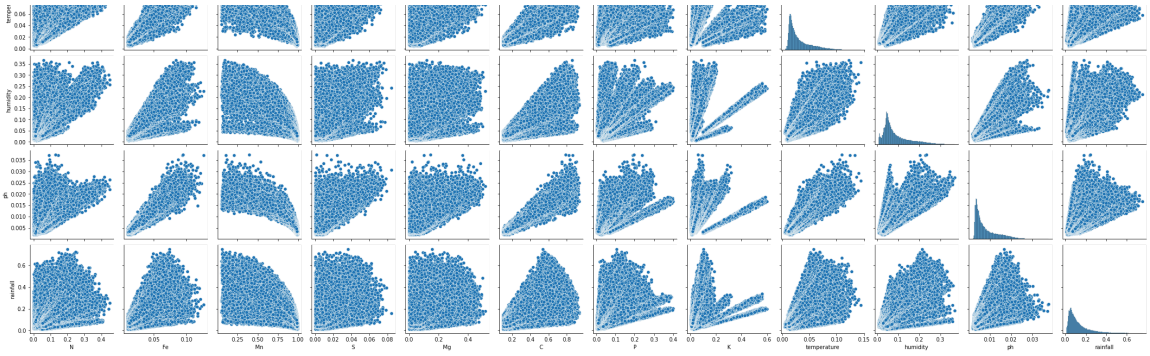


Plotting pairgrid

In [ ]:
```
sns.pairplot(data)
```

Out[ ]: <seaborn.axisgrid.PairGrid at 0x7fe16d3acfa0>

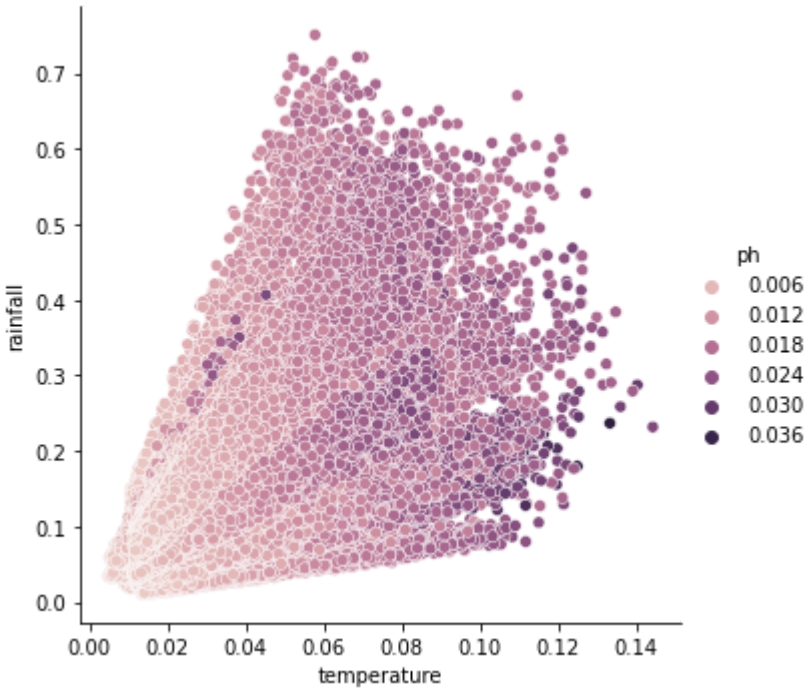




Plotting Relational Plot

In [ ]:
```
sns.relplot(data=data,x='temperature', y='rainfall', hue = 'ph')
```

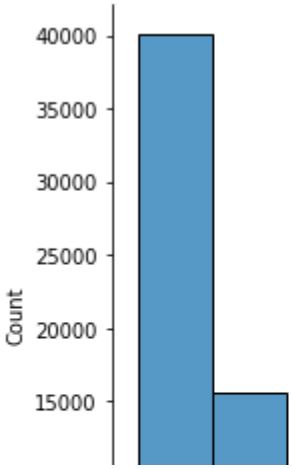
Out[ ]: <seaborn.axisgrid.FacetGrid at 0x7fe17413c0a0>

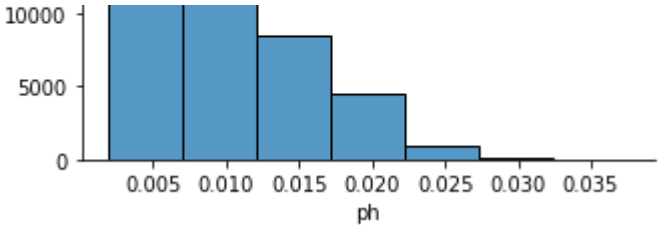


Data distribution plot

In [ ]:
```
sns.displot(data['ph'],bins=7)
```

Out[ ]: <seaborn.axisgrid.FacetGrid at 0x7fe146347ac0>

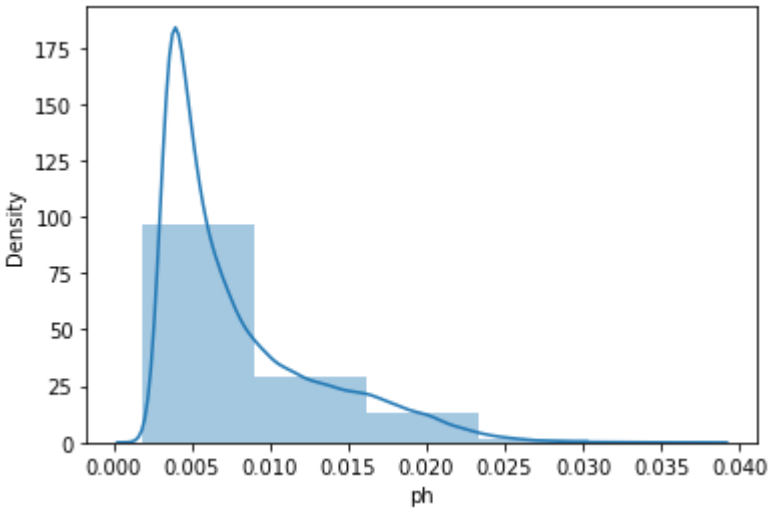




In [ ]:
```
sns.distplot(data['ph'],bins=5)
```

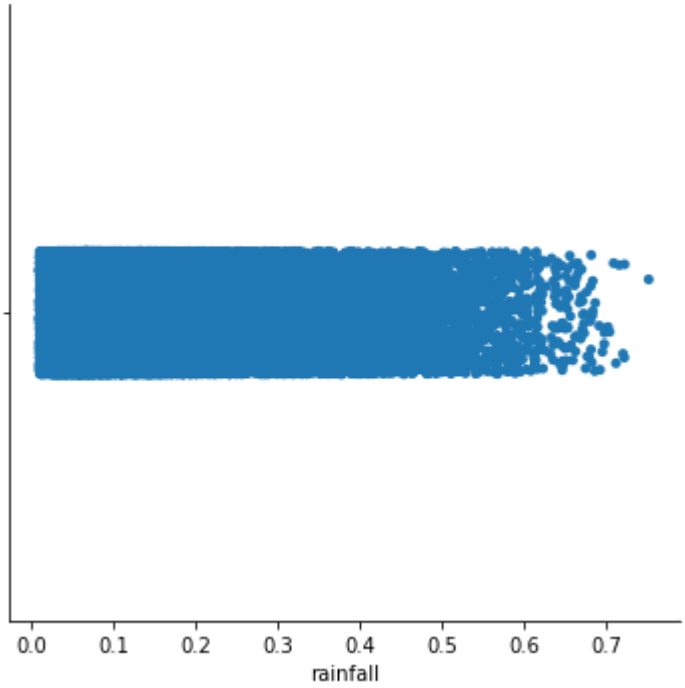
```
/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[ ]: <AxesSubplot:xlabel='ph', ylabel='Density'>



In [ ]:
```
sns.catplot(x='rainfall', data = data)
```

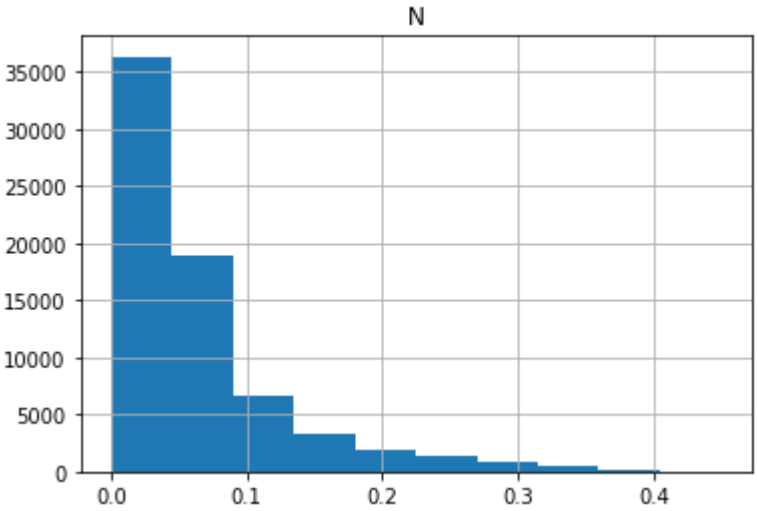
Out[ ]: <seaborn.axisgrid.FacetGrid at 0x7fe16e92fac0>

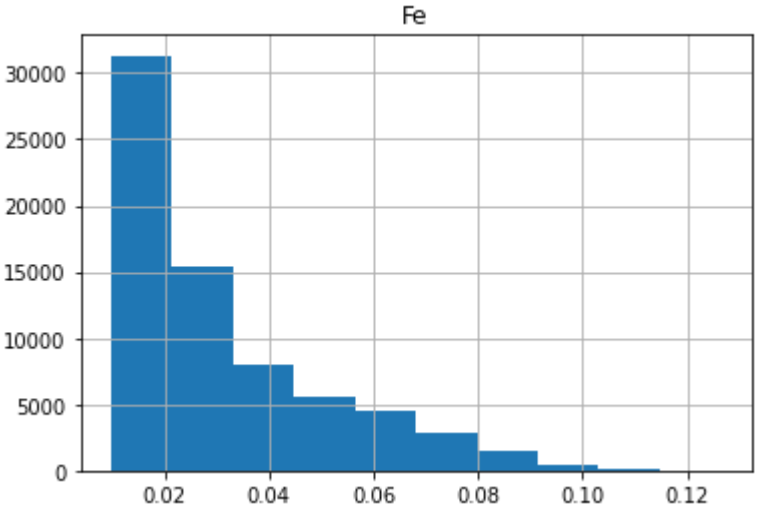


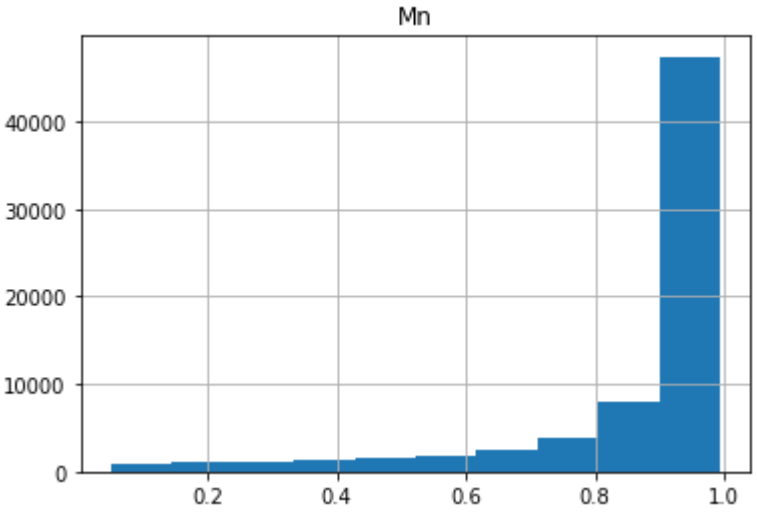
Plotting histogram from each feature to check skewed features if any

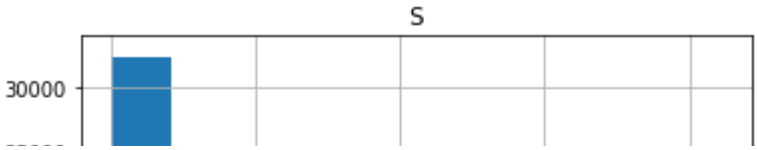
In [ ]:

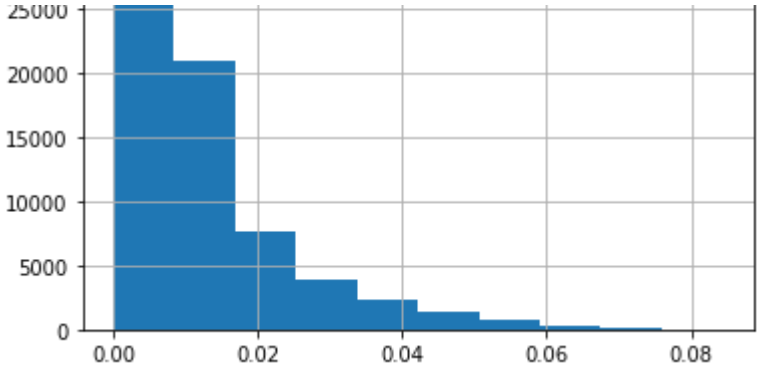
```
feature_col = np.array(data.drop(columns=['label']).columns)
for col in feature_col:
    plt.figure()
    data[col].hist()
    plt.title(col)
```

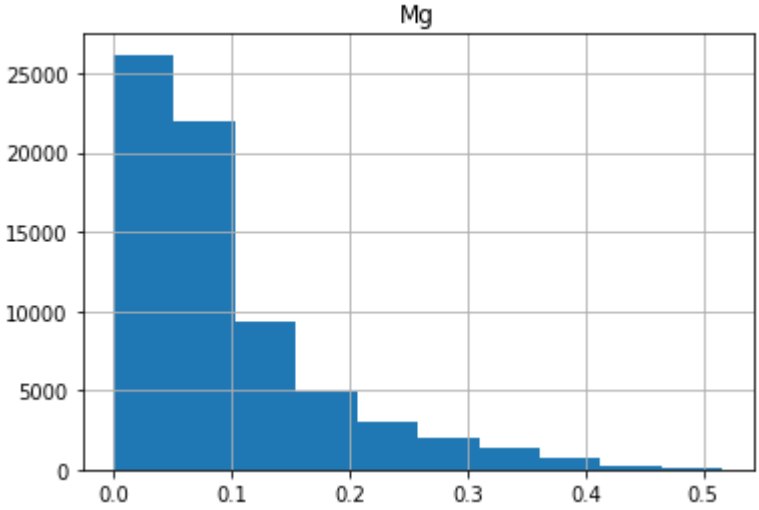


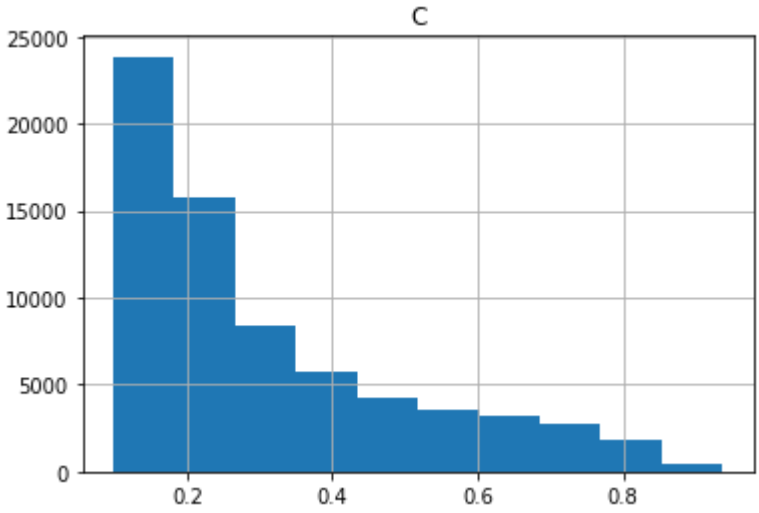


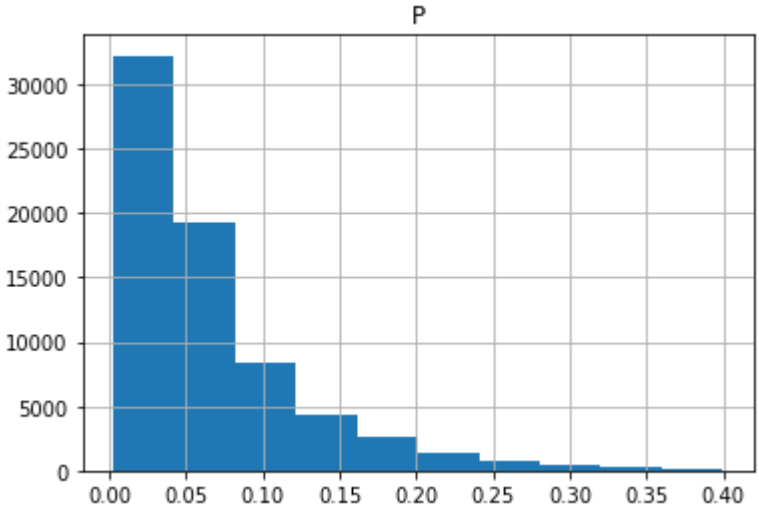




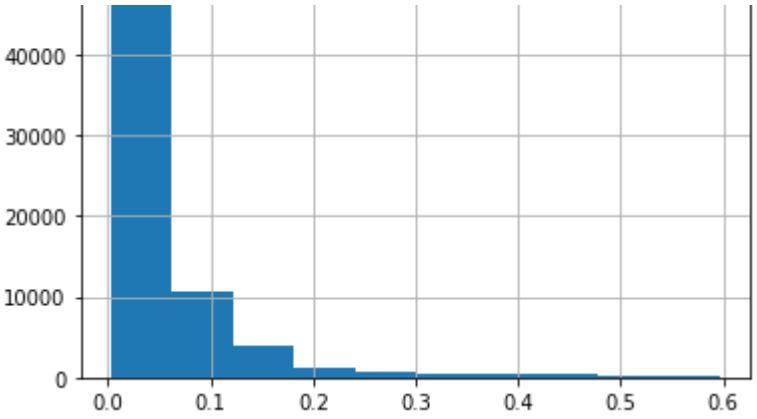

25000
20000
15000
10000
5000
0
0.00
0.02
0.04
0.06
0.08

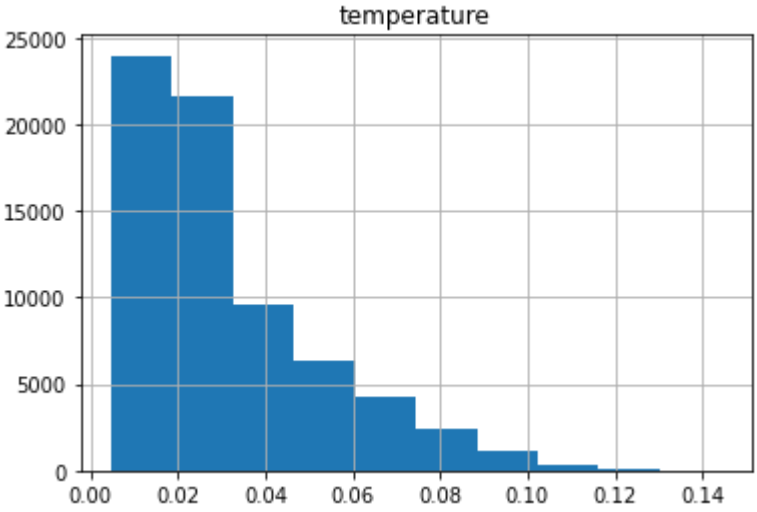

Mg
25000
20000
15000
10000
5000
0
0.0
0.1
0.2
0.3
0.4
0.5

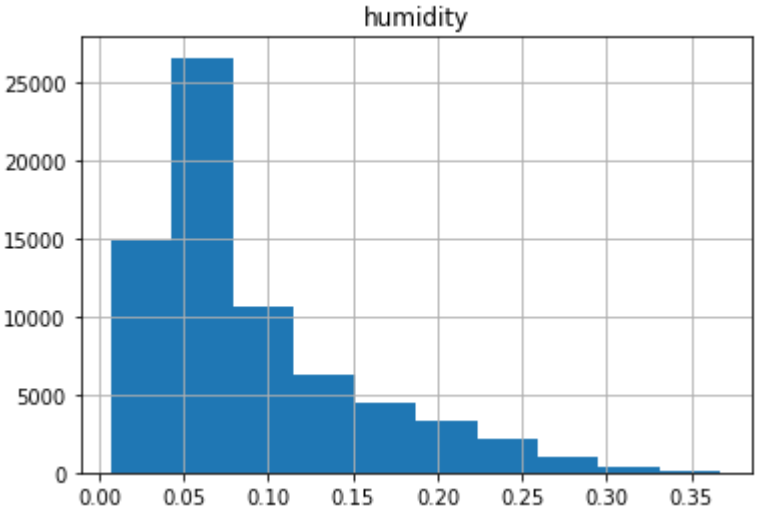

C
25000
20000
15000
10000
5000
0
0.2
0.4
0.6
0.8

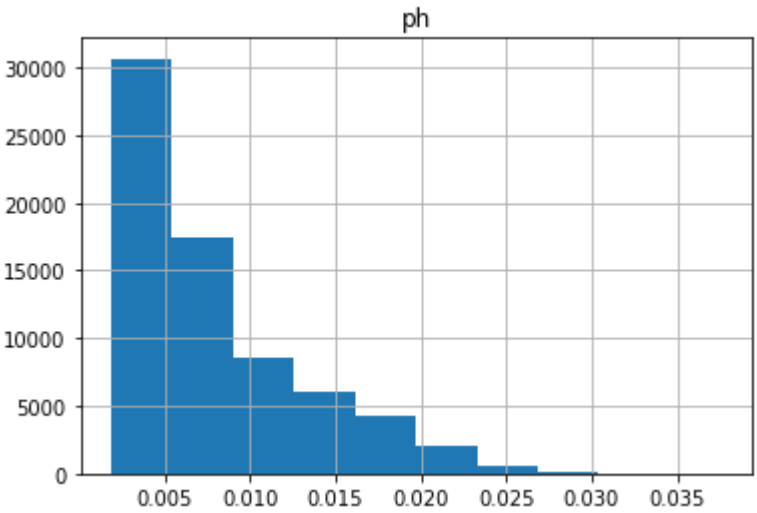

P
30000
25000
20000
15000
10000
5000
0
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40


K
50000

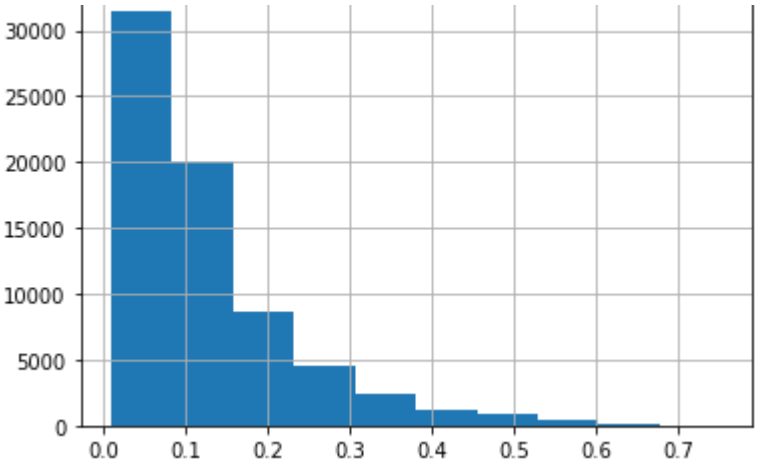

40000
30000
20000
10000
0
0.0
0.1
0.2
0.3
0.4
0.5
0.6


temperature
25000
20000
15000
10000
5000
0
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14


humidity
25000
20000
15000
10000
5000
0
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35


ph
30000
25000
20000
15000
10000
5000
0
0.005
0.010
0.015
0.020
0.025
0.030
0.035


rainfall



Splitting Dataset

Importing Libraries

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import classification_report
```

Spliting datasets in x and y

In [ ]:
```
# feature vector and output label
y_label = data['label']
x_features = data.drop(columns='label')
```

Splitting datasset into training and testing data

In [ ]:
```
# splitting training and testing dataset. 70% dataset is taken as training
#and remaining 30% dataset is taken for testing
x_train,x_test,y_train,y_test=train_test_split(x_features,y_label,test_size=0
```

In [ ]:
```
print("feature shape",x_train.shape)
print("label shape",x_test.shape)
```

```
feature shape (48802, 12)
label shape (20916, 12)
```

Training Model

Defining model vs accuracy variables

In [ ]:
```
model_name = []
accuracy_of_model = []
```

Logistic Regression Model

In [ ]:
```
from sklearn.linear_model import LogisticRegression
# creating LogisticRegression object
logisticRegModel = LogisticRegression(solver = 'saga',random_state=0,penalty=
# training our model
logisticRegModel.fit(x_train,y_train)
# finding predicted labels of testing dataset
predicted_labels = logisticRegModel.predict(x_test)
# calculating accuracy score of our predicted labels
score = metrics.accuracy_score(y_test, predicted_labels)

print("Logistic Regression's Accuracy is: ", score*100, "%")
# printing accuracy of each classes
print(classification_report(y_test,predicted_labels))
model_name.append("Logistic Regression")
accuracy_of_model.append(score*100)
```

```
Logistic Regression's Accuracy is:  77.68215720022948 %
              precision    recall  f1-score   support

       apple       0.96      1.00      0.98       949
      banana       0.92      0.99      0.96       938
   blackgram       0.88      0.70      0.78       962
    chickpea       1.00      1.00      1.00       981
     coconut       0.96      0.80      0.87       994
      coffee       0.75      1.00      0.86      1004
      cotton       0.89      0.91      0.90       967
      grapes       1.00      0.96      0.98       968
        jute       0.90      0.44      0.59       964
  kidneybeans       0.69      0.97      0.81       949
      lentil       0.86      0.65      0.74       970
       maize       0.99      0.60      0.75       916
       mango       0.99      0.89      0.93       993
    mothbeans       0.99      0.22      0.35       918
     mungbean       0.46      1.00      0.63       887
    muskmelon       0.47      1.00      0.64       918
       orange       0.53      1.00      0.69      1011
       papaya       0.96      0.65      0.78       919
   pigeonpeas       0.94      0.71      0.81       905
  pomegranate       1.00      0.37      0.54       938
         rice       0.70      0.93      0.80       939
   watermelon       1.00      0.25      0.40       926

     accuracy                           0.78     20916
    macro avg       0.86      0.77      0.76     20916
 weighted avg       0.86      0.78      0.77     20916
```

Creating Confusion Matrix

In [ ]:
```
from sklearn.metrics import confusion_matrix
# printing confusion matrix for testing dataset
confusion_matrix(y_test, predicted_labels)
```

Out[ ]:
```
array([[ 946,    0,    0,    0,    0,    0,    0,    3,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0],
       [   0,  931,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    7,    0,    0,    0,    0,    0,    0],
       [   0,    6,  676,    0,    0,    0,    0,    0,    0,   29,   34,
           0,    0,    3,  194,   20,    0,    0,    0,    0,    0,    0],
       [   0,    0,    0,  981,    0,    0,    0,    0,    0,    0,    0,
```

```
       0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0],
   [   0,    0,    0,    0,  799,    0,    0,    0,    0,    0,    0,
       0,    0,    0,    0,    0,  195,    0,    0,    0,    0,    0],
   [   0,    0,    0,    0,    0, 1002,    0,    0,    0,    0,    0,
       0,    0,    0,    0,    0,    0,    0,    0,    0,    2,    0],
   [   0,    0,    0,    0,    0,    0,  877,    0,    0,    0,    0,
       4,    0,    0,    0,   86,    0,    0,    0,    0,    0,    0],
   [  35,    0,    0,    0,    0,    0,    0,  933,    0,    0,    0,
       0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0],
   [   0,    0,    0,    0,    1,  267,    0,    0,  425,    0,    0,
       0,    0,    0,    0,    0,    4,   13,    0,    0,  254,    0],
   [   0,    0,    0,    0,    0,    0,    0,    0,    0,  918,    0,
       0,    0,    0,    0,    0,    0,    0,    0,   31,    0,    0,    0],
   [   0,    0,   24,    0,    0,    0,    0,    0,    0,    0,  626,
       0,    0,    0,  318,    2,    0,    0,    0,    0,    0,    0],
   [   0,   53,    0,    0,    0,   42,  110,    0,    0,    4,    0,
     547,    0,    0,    0,  144,   15,    1,    0,    0,    0,    0],
   [   0,    0,    0,    0,    0,    0,    0,    0,    0,   25,    0,
       0,  880,    0,    0,    0,   86,    2,    0,    0,    0,    0],
   [   0,    0,   63,    0,    0,    0,    0,    0,    0,   92,   69,
       0,    9,  198,  466,   19,    2,    0,    0,    0,    0,    0],
   [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
       0,    0,    0,  884,    3,    0,    0,    0,    0,    0,    0],
   [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
       0,    0,    0,    0,  918,    0,    0,    0,    0,    0,    0],
   [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
       0,    0,    0,    0,    0, 1011,    0,    0,    0,    0,    0],
   [   0,   19,    2,    0,   33,    0,    0,    0,    3,    0,    0,
       0,    0,    0,   70,   60,    7,  600,   12,    0,  113,    0],
   [   0,    0,    2,    0,    0,    0,    0,    0,    0,  255,    0,
       0,    1,    0,    0,    0,    0,    1,  646,    0,    0,    0],
   [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
       0,    0,    0,    0,    0,  579,    9,    0,  350,    0,    0],
   [   0,    0,    0,    0,    1,   27,    0,    0,   42,    0,    0,
       0,    0,    0,    0,    0,    0,    0,    0,    0,  869,    0],
   [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
       0,    0,    0,    0,  695,    0,    0,    0,    0,    0,  231]])
```

Decision Tree

In [ ]:

```
from sklearn.tree import DecisionTreeClassifier
# creating Decision Tree model object
decisionTreeModel = DecisionTreeClassifier(criterion="gini",random_state=0,ma
# Training model
decisionTreeModel.fit(x_train,y_train)
# Predicting our test labels
predicted_labels = decisionTreeModel.predict(x_test)
# calculating accuracy score of our predicted labels
score = metrics.accuracy_score(y_test, predicted_labels)
print("Decision Trees's Accuracy is: ", score*100,"%")
print(classification_report(y_test,predicted_labels))
model_name.append("Decision Trees")
accuracy_of_model.append(score*100)
```

```
Decision Trees's Accuracy is:  72.83897494740869 %
              precision    recall  f1-score   support

       apple       0.92      0.96      0.94       949
      banana       0.44      0.98      0.60       938
   blackgram       0.53      0.53      0.53       962
    chickpea       1.00      1.00      1.00       981
     coconut       0.72      0.74      0.73       994
```

```
              coffee       0.75      0.74      0.74      1004
              cotton       0.73      0.87      0.79       967
              grapes       0.97      0.91      0.94       968
                jute       0.63      0.18      0.28       964
          kidneybeans       0.95      0.73      0.83       949
              lentil       0.66      0.66      0.66       970
               maize       0.78      0.45      0.57       916
               mango       0.90      0.64      0.74       993
           mothbeans       0.56      0.35      0.43       918
            mungbean       0.86      0.65      0.74       887
           muskmelon       1.00      0.95      0.97       918
              orange       0.99      0.83      0.90      1011
              papaya       0.87      0.52      0.65       919
           pigeonpeas       0.73      0.70      0.71       905
          pomegranate       0.71      0.87      0.78       938
                rice       0.35      0.79      0.48       939
          watermelon       0.98      0.94      0.96       926

            accuracy                           0.73     20916
           macro avg       0.77      0.73      0.73     20916
        weighted avg       0.77      0.73      0.73     20916
```

Creating confusion matrix

In [ ]:

```
from sklearn.metrics import confusion_matrix
# creating confusion matrix of testing dataset
confusion_matrix(y_test, predicted_labels)
```

Out[ ]:

```
array([[915,   0,  10,   0,   0,   0,   0,  24,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0, 918,   0,   0,   0,   0,   7,   0,   0,   0,   0,   0,   0,
          1,   0,   0,   0,   2,   0,   0,   6,   4],
       [  0, 309, 514,   0,   0,   0,   1,   0,   0,   0,  52,   8,   0,
         39,   7,   0,   0,   0,  10,   0,  22,   0],
       [  0,   0,   0, 979,   0,   0,   0,   0,   0,   0,   0,   0,   1,
          1,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0, 738,  15,   0,   0,   1,   0,   0,   0,   6,
          0,   0,   0,   4,   5,   9, 195,  21,   0],
       [  0,   0,   0,   0,   0, 742,  26,   0,   9,   0,   0,  43,   0,
          0,   0,   0,   0,   0,   0,   0, 184,   0],
       [  0,  58,   0,   0,   0,   7, 843,   0,   0,   0,   0,  57,   0,
          2,   0,   0,   0,   0,   0,   0,   0,   0],
       [ 80,   0,   4,   0,   0,   0,   0, 882,   0,   0,   0,   0,   2,
          0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,  28,   0,   0,   6,  82,   0,   0, 177,   0,   0,   1,   0,
          0,   0,   0,   0,  12,   1,   7, 650,   0],
       [  0,  67,   1,   1,   0,   0,   0,   0,   0, 694,   4,   0,   1,
          2,   0,   0,   0,   0,  77,   0, 102,   0],
       [  0, 100, 124,   0,   0,   0,   0,   0,   0,   0, 637,   0,   0,
         82,  16,   0,   0,   0,  11,   0,   0,   0],
       [  0, 196,   9,   0,   0,  16, 199,   0,   3,   0,   0, 415,   0,
         41,   2,   0,   0,   0,   0,   0,  35,   0],
       [  0,  12,  35,   0,  92,   5,   0,   0,  12,   0,   1,   0, 633,
          4,   0,   0,   2,  30,  56,  69,  42,   0],
       [  0, 119, 222,   0,   1,   0,   0,   0,   0,   0, 141,   0,  16,
        321,  60,   0,   0,   0,  32,   3,   3,   0],
       [  0, 134,   7,   0,   1,   0,   0,   0,   0,   0, 101,   0,   0,
         46, 574,   0,   4,   2,  13,   5,   0,   0],
       [  0,   0,   0,   0,   0,   0,  31,   0,   0,   0,   0,   0,   0,
          0,   0, 876,   0,   0,   0,   0,   0,  11],
       [  0,   0,   0,   0,  62,  13,   1,   0,   0,   0,   0,   9,   0,
          0,   0,   0, 841,   0,   8,  47,  30,   0]
```

```
       0,   0,   0, 841,   0,   0,  47,  36,   0],
  [  0, 136,   6,   0,  37,   0,   0,   0,  13,   0,  10,   0,  30,
    37,   9,   0,   0, 477,  21,  10, 132,   1],
  [  0,  22,  46,   0,   0,   0,   0,   0,   3,  33,  19,   0,  12,
     0,   0,   0,   0,   0, 635,   0, 135,   0],
  [  0,   0,   0,   0,  86,   2,   0,   0,   0,   0,   0,   0,   6,
     0,   3,   0,   0,   6,   1, 816,  18,   0],
  [  0,   1,   0,   0,   6, 112,   0,   0,  61,   0,   0,   0,   0,
     0,   0,   0,   0,  17,   0,   1, 741,   0],
  [  0,   0,   0,   0,   0,   1,  53,   0,   0,   0,   0,   0,   0,
     0,   0,   4,   0,   0,   0,   1,   0, 867]])
```

Ada Boost

In [ ]:

```
from sklearn.ensemble import AdaBoostClassifier
clf = AdaBoostClassifier()
clf.fit(x_train,y_train)
y_predict=clf.predict(x_test)
score = metrics.accuracy_score(y_test, y_predict)
print("AdaBoost Accuracy is: ", score*100,"%")
print(classification_report(y_test,y_predict))
model_name.append("AdaBoost")
accuracy_of_model.append(score*100)
```

```
AdaBoost Accuracy is:  11.713520749665328 %
/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1248: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
              precision    recall  f1-score   support

       apple       0.29      0.99      0.45       949
      banana       0.00      0.00      0.00       938
   blackgram       0.00      0.00      0.00       962
    chickpea       1.00      0.08      0.15       981
     coconut       0.00      0.00      0.00       994
      coffee       0.00      0.00      0.00      1004
      cotton       0.00      0.00      0.00       967
      grapes       0.00      0.00      0.00       968
        jute       0.00      0.00      0.00       964
 kidneybeans       0.44      0.57      0.50       949
      lentil       0.00      0.00      0.00       970
       maize       0.00      0.00      0.00       916
       mango       0.00      0.00      0.00       993
   mothbeans       0.00      0.00      0.00       918
    mungbean       0.05      1.00      0.10       887
   muskmelon       0.00      0.00      0.00       918
      orange       0.00      0.00      0.00      1011
      papaya       0.00      0.00      0.00       919
  pigeonpeas       0.00      0.00      0.00       905
 pomegranate       0.00      0.00      0.00       938
        rice       0.06      0.00      0.01       939
  watermelon       0.00      0.00      0.00       926

    accuracy                           0.12     20916
   macro avg       0.08      0.12      0.05     20916
weighted avg       0.09      0.12      0.05     20916

/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1248: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
```

```
  _warn_prf(average, modifier, msg_start, len(result))
/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/s
klearn/metrics/_classification.py:1248: UndefinedMetricWarning: Precision and F
-score are ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [ ]:

```
model_boost=AdaBoostClassifier(DecisionTreeClassifier(max_depth=9),n_estimato
model_boost.fit(x_train.values,y_train.values)
y_predict=model_boost.predict(x_test.values)
score = metrics.accuracy_score(y_test, y_predict)
predicted_labels = model_boost.predict(x_test)
print("AdaBoost with Decision Tree(Base) Accuracy is: ", score*100,"%")
print(classification_report(y_test,predicted_labels))
model_name.append("AdaBoost with Decision Tree(Base)")
accuracy_of_model.append(score*100)
```

```
AdaBoost with Decision Tree(Base) Accuracy is:  89.57257601835916 %
              precision    recall  f1-score   support

       apple       1.00      0.99      0.99       949
      banana       0.99      0.97      0.98       938
   blackgram       0.67      0.86      0.76       962
    chickpea       1.00      1.00      1.00       981
     coconut       0.95      0.96      0.96       994
      coffee       0.97      0.85      0.91      1004
      cotton       0.96      0.75      0.84       967
      grapes       0.99      1.00      0.99       968
        jute       0.63      0.87      0.73       964
 kidneybeans       1.00      0.95      0.97       949
      lentil       0.92      0.57      0.70       970
       maize       0.77      0.94      0.85       916
       mango       0.98      0.93      0.96       993
   mothbeans       0.62      0.83      0.71       918
    mungbean       0.96      0.78      0.86       887
   muskmelon       1.00      0.99      1.00       918
      orange       1.00      0.97      0.98      1011
      papaya       0.86      0.93      0.89       919
  pigeonpeas       0.92      0.94      0.93       905
 pomegranate       0.97      0.98      0.98       938
        rice       0.88      0.64      0.74       939
  watermelon       0.99      1.00      1.00       926

    accuracy                           0.90     20916
   macro avg       0.91      0.90      0.90     20916
weighted avg       0.91      0.90      0.90     20916
```

In [ ]:

```
from sklearn.metrics import confusion_matrix
# creating confusion matrix of testing dataset
confusion_matrix(y_test, predicted_labels)
```

Out[ ]:

```
array([[936,   0,   0,   0,   0,   0,   0,  10,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   3,   0,   0,   0,   0],
       [  0, 910,   1,   0,   0,   0,   0,   0,   9,   0,   0,   2,   0,
          2,   0,   0,   0,  14,   0,   0,   0,   0],
       [  0,   0, 830,   0,   0,   0,   1,   0,   0,   0,   6,   5,   0,
        113,   2,   0,   0,   4,   1,   0,   0,   0],
       [  0,   0,   0, 981,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0, 956,   0,   0,   0,   0,   0,   0,   0,   1,
```

```
       0,   0,   0,   2,   9,   0,  26,   0,   0],
   [  0,   0,   0,   0,   0, 857,   0,   0, 134,   0,   0,   4,   0,
      0,   0,   0,   0,   0,   0,   0,   9,   0],
   [  0,   0,   0,   0,   0,   1, 724,   0,   0,   0,   0, 241,   0,
      0,   0,   0,   0,   1,   0,   0,   0,   0],
   [  2,   0,   0,   0,   0,   0,   0, 966,   0,   0,   0,   0,   0,
      0,   0,   0,   0,   0,   0,   0,   0,   0],
   [  0,   0,   0,   0,   0,  13,   0,   0, 837,   0,   0,   8,   0,
      0,   0,   0,   0,  33,   0,   0,  73,   0],
   [  0,   0,   0,   0,   0,   0,   0,   0,   0, 904,   0,   0,   0,
      0,   0,   0,   0,   0,  45,   0,   0,   0],
   [  0,   0, 215,   0,   0,   0,   0,   0,   0,   0, 550,   0,   0,
    185,  20,   0,   0,   0,   0,   0,   0,   0],
   [  0,   6,   9,   0,   0,   8,  26,   0,   5,   0,   0, 862,   0,
      0,   0,   0,   0,   0,   0,   0,   0,   0],
   [  0,   0,   0,   0,   7,   0,   0,   0,   0,   0,   0,   0, 927,
      5,   0,   0,   0,  44,  10,   0,   0,   0],
   [  0,   0, 102,   0,   0,   0,   0,   0,   0,   0,  22,   0,   4,
    765,   5,   0,   0,   5,  15,   0,   0,   0],
   [  0,   0,  68,   0,   0,   0,   0,   0,   0,   0,  18,   0,   0,
    104, 693,   0,   0,   4,   0,   0,   0,   0],
   [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
      0,   0, 912,   0,   0,   0,   0,   0,   6],
   [  0,   0,   1,   0,  33,   0,   0,   0,   0,   0,   0,   0,   0,
      0,   0,   0, 977,   0,   0,   0,   0,   0],
   [  0,   3,   0,   0,   0,   0,   0,   0,  31,   0,   2,   0,  12,
     15,   2,   0,   0, 852,   1,   0,   1,   0],
   [  0,   0,   8,   0,   0,   0,   0,   0,   0,   4,   0,   0,   2,
     40,   0,   0,   0,   4, 847,   0,   0,   0],
   [  0,   0,   0,   0,  12,   0,   0,   0,   0,   0,   0,   0,   1,
      0,   0,   0,   0,   2,   0, 923,   0,   0],
   [  0,   0,   0,   0,   0,   8,   0,   0, 310,   0,   0,   0,   0,
      0,   0,   0,   0,  21,   0,   0, 600,   0],
   [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
      0,   0,   0,   0,   0,   0,   0,   0, 926]])
```

Random Forest

In [ ]:

```
from sklearn.ensemble import RandomForestClassifier
# creating Random Forest Model
randomForestModel = RandomForestClassifier(n_estimators=20, random_state=2,cr
# Training model
randomForestModel.fit(x_train,y_train)
# Predicting label of testing dataset.
predicted_labels = randomForestModel.predict(x_test)
# calcualting accuracy score of RF model
score = metrics.accuracy_score(y_test, predicted_labels)

print("Random Forest Accuracy is: ", score*100, "%")

print(classification_report(y_test,predicted_labels))

model_name.append("Random Forest")
accuracy_of_model.append(score*100)
```

```
Random Forest Accuracy is:  92.60374832663989 %
              precision    recall  f1-score   support

       apple       0.99      1.00      1.00       949
      banana       0.96      1.00      0.98       938
   blackgram       0.74      0.88      0.80       962
    chickpea       1.00      1.00      1.00       981
```

coconut	0.94	0.95	0.95	994
coffee	0.96	0.97	0.96	1004
cotton	0.87	0.99	0.92	967
grapes	1.00	0.99	1.00	968
jute	0.84	0.77	0.80	964
kidneybeans	0.98	0.99	0.99	949
lentil	0.83	0.83	0.83	970
maize	0.97	0.82	0.89	916
mango	0.98	0.96	0.97	993
mothbeans	0.93	0.59	0.72	918
mungbean	0.82	0.92	0.87	887
muskmelon	1.00	1.00	1.00	918
orange	1.00	0.99	1.00	1011
papaya	0.93	0.93	0.93	919
pigeonpeas	0.93	0.98	0.95	905
pomegranate	0.96	0.93	0.94	938
rice	0.81	0.87	0.84	939
watermelon	1.00	1.00	1.00	926
accuracy			0.93	20916
macro avg	0.93	0.93	0.92	20916
weighted avg	0.93	0.93	0.92	20916

Creating confusion matrix

In [ ]:
```
from sklearn.metrics import confusion_matrix
# creating confusion matrix of RF model
confusion_matrix(y_test, predicted_labels)
```

Out[ ]:
```
array([[ 949,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0],
       [   0,  938,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0],
       [   0,    0,  847,    0,    0,    0,    0,    0,    0,    0,   66,
           11,    0,   20,   14,    0,    0,    0,    4,    0,    0,    0],
       [   0,    0,    0,  981,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,  942,    2,    0,    0,    1,    0,    0,
            0,    8,    0,    0,    0,    0,    2,    0,   38,    1,    0],
       [   0,    1,    0,    0,    0,  973,    1,    0,   22,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    5,    2],
       [   0,    1,    0,    0,    0,    0,  954,    0,    0,    0,    0,
           12,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0],
       [   7,    0,    0,    0,    0,    0,    0,  961,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0],
       [   0,    7,    0,    0,    1,   24,    0,    0,  740,    0,    0,
            3,    0,    0,    0,    0,    0,   22,    0,    0,  166,    1],
       [   0,    0,    0,    1,    0,    0,    0,    0,    0,  941,    0,
            0,    0,    0,    0,    0,    0,    0,    7,    0,    0,    0],
       [   0,    0,  112,    0,    0,    0,    0,    0,    0,    0,  806,
            0,    0,   12,   40,    0,    0,    0,    0,    0,    0,    0],
       [   0,    6,    3,    0,    0,    2,  146,    0,    4,    0,    0,
          755,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    1,    3,    0,    0,    0,    0,    0,
            0,  956,    0,    0,    0,    0,   21,    9,    1,    1,    1],
       [   0,    0,  134,    0,    0,    0,    0,    0,    0,    2,   84,
            0,    1,  538,  119,    0,    0,    3,   37,    0,    0,    0],
       [   0,    0,   45,    0,    0,    0,    0,    0,    0,    0,   20,
            0,    0,    6,  815,    0,    0,    0,    1,    0,    0,    0],
       [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,  918,    0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    4,    0,    0,    0,    0,    0,    0,
```

```
       0,    1,    0,    0,    0, 1004,    0,    2,    0,    0,    0],
    [   0,   24,    3,    0,    0,    0,    0,    0,   10,    0,    0,
        0,    2,    0,    4,    0,    0,  854,    4,    0,   18,    0],
    [   0,    0,    4,    0,    0,    0,    0,    0,    0,   16,    0,
        0,    0,    0,    0,    0,    0,    1,  884,    0,    0,    0],
    [   0,    0,    0,    0,   51,    0,    0,    0,    1,    0,    0,
        0,    6,    0,    0,    0,    0,    5,    0,  875,    0,    0],
    [   0,    0,    0,    0,    0,    9,    0,    0,  105,    0,    0,
        0,    0,    0,    0,    0,    0,   10,    0,    0,  815,    0],
    [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
        0,    0,    0,    0,    3,    0,    0,    0,    0,    0,  923]])
```

Naive bayes

In [ ]:

```
from sklearn.naive_bayes import GaussianNB
# creating object of gaussina Naive Bayes
gaussianNaiveBayesModel = GaussianNB(var_smoothing=0.09)
# training our model
gaussianNaiveBayesModel.fit(x_train,y_train)
# predicting labels of testing set
predicted_labels = gaussianNaiveBayesModel.predict(x_test)
# calculating score of our testing set
score = metrics.accuracy_score(y_test, predicted_labels)

print("Naive Bayes's Accuracy is: ", score*100, "%")

print(classification_report(y_test,predicted_labels))

model_name.append("Naive Bayes")
accuracy_of_model.append(score*100)
```

```
Naive Bayes's Accuracy is:  39.06100592847581 %
              precision    recall  f1-score   support

       apple       0.82      0.54      0.65       949
      banana       0.62      0.38      0.47       938
   blackgram       0.33      0.25      0.29       962
    chickpea       0.38      0.52      0.44       981
     coconut       0.68      0.34      0.46       994
      coffee       0.64      0.31      0.42      1004
      cotton       0.69      0.30      0.42       967
      grapes       0.93      0.47      0.62       968
        jute       0.45      0.20      0.28       964
  kidneybeans       0.71      0.44      0.54       949
      lentil       0.72      0.42      0.53       970
       maize       0.25      0.21      0.23       916
       mango       0.15      0.37      0.22       993
   mothbeans       0.11      0.82      0.19       918
    mungbean       0.79      0.35      0.49       887
   muskmelon       0.76      0.62      0.68       918
      orange       0.79      0.42      0.55      1011
      papaya       0.47      0.16      0.24       919
  pigeonpeas       0.64      0.31      0.42       905
 pomegranate       0.80      0.30      0.43       938
        rice       0.49      0.45      0.47       939
  watermelon       0.88      0.41      0.56       926

    accuracy                           0.39     20916
   macro avg       0.60      0.39      0.44     20916
weighted avg       0.60      0.39      0.44     20916
```

Creating Confusion matrix

In [ ]:
```
from sklearn.metrics import confusion_matrix
# creating confusion matrix of our testing set
confusion_matrix(y_test, predicted_labels)
```

Out[ ]:
```
array([[509,   0,   0, 414,   0,   0,   0,  26,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0, 357, 203,   0,   0,   0,   0,   0,   1,   0,   0,  93,   0,
        284,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,  98, 242,   0,   0,   0,  18,   0,   0,   0,  33,   0,   0,
        551,   1,   0,   0,  19,   0,   0,   0,   0],
       [ 30,   0,   0, 515,   0,   0,   0,   3,   0,   0,   0,   0,   0,
        433,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0, 341,   0,   0,   0,   7,   0,   0,   0, 408,
         54,   0,   0, 109,   3,   0,   2,  70,   0],
       [  0,   0,   0,   0,   0, 316,   0,   0,  80,   0,   0, 131, 267,
        135,   0,   0,   0,   0,   0,   0,  75,   0],
       [  0,   2,  23,   0,   0,   0, 293,   0,   0,   0,   0, 314,   0,
        335,   0,   0,   0,   0,   0,   0,   0,   0],
       [ 81,   0,   0, 433,   0,   0,   0, 454,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0,   0,  80,   0,   0, 192,   0,   0,  31, 395,
        105,   0,   0,   0,   0,   2,   0, 159,   0],
       [  0,   1,   0,   0,   0,   0,   0,   0,   0, 418,   4,   0,   0,
        408,   0,   0,   0,   0, 117,   0,   1,   0],
       [  0,  15,  44,   0,   0,   0,   2,   6,   0,   0, 411,   0,   0,
        450,  40,   0,   0,   2,   0,   0,   0,   0],
       [  0,  59, 102,   0,   0,   2,  86,   0,  23,   0,   0, 189,   0,
        455,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0,   6,   0,   0,   0,  39,   0,   0,   0, 371,
        483,   0,   0,   0,  53,   8,  33,   0,   0],
       [  0,   4,  49,   0,   0,   0,   6,   0,   0,   0,  41,   0,   2,
        752,  32,   0,   0,  26,   5,   1,   0,   0],
       [  0,   1,   1,   0,   0,   0,   7,   0,   0,   0,  73,   0,   0,
        478, 311,   0,   0,  15,   0,   1,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        298,   0, 568,   0,   0,   0,   0,   0,  52],
       [  0,   0,   0,   0, 102,   0,   0,   0,  15,   0,   0,   0,  24,
        411,   0,   0, 423,   3,   0,  33,   0,   0],
       [  0,  40,  56,   0,  18,   0,   1,   0,  43,   0,   5,   0, 216,
        265,   8,   0,   0, 148,  23,   0,  94,   2],
       [  0,   0,   3,   0,   0,   0,   0,   0,   7, 173,   0,   0,  97,
        294,   0,   0,   0,   5, 285,   0,  41,   0],
       [  0,   0,   0,   0,  19,   0,   0,   0,   3,   0,   0,   0, 249,
        347,   0,   0,   1,  42,   0, 277,   0,   0],
       [  0,   0,   0,   0,  13,  96,   0,   0,  17,   0,   0,   3, 383,
          0,   0,   0,   0,   0,   7,   0, 420,   0],
       [  0,   0,   0,   0,   0,   0,   9,   0,   0,   0,   0,   0,   0,
        356,   0, 183,   0,   0,   0,   0,   0, 378]])
```

In [ ]:
```
from sklearn.neural_network import MLPClassifier
# creating an object of MLP Classifier , tol = 0.001 not max iter
mlpClassifier = MLPClassifier(hidden_layer_sizes = 200,activation='relu',batc
# training our model
mlpClassifier.fit(x_train,y_train)
# predicting labels of testing set
predicted_labels = mlpClassifier.predict(x_test)
# accuaray of our model on testing set
score = metrics.accuracy_score(y_test, predicted_labels)

print("Neural Network Accuracy is: ", score*100, "%")
```

```
print(classification_report(y_test,predicted_labels))

model_name.append("Neural Network")
accuracy_of_model.append(score*100)
```

```
Neural Network Accuracy is:  98.38879326831133 %
              precision    recall  f1-score   support

       apple       1.00      1.00      1.00       949
      banana       1.00      1.00      1.00       938
   blackgram       0.96      0.96      0.96       962
    chickpea       1.00      1.00      1.00       981
     coconut       0.99      0.99      0.99       994
      coffee       0.99      1.00      1.00      1004
      cotton       0.99      0.98      0.98       967
      grapes       1.00      1.00      1.00       968
        jute       0.91      0.96      0.93       964
  kidneybeans       0.99      1.00      0.99       949
      lentil       0.97      0.98      0.98       970
       maize       0.97      0.98      0.98       916
       mango       1.00      1.00      1.00       993
    mothbeans       0.95      0.94      0.94       918
     mungbean       1.00      1.00      1.00       887
    muskmelon       1.00      1.00      1.00       918
      orange       1.00      1.00      1.00      1011
      papaya       0.99      0.98      0.98       919
  pigeonpeas       1.00      0.98      0.99       905
 pomegranate       0.99      1.00      0.99       938
        rice       0.96      0.90      0.93       939
  watermelon       1.00      1.00      1.00       926

    accuracy                           0.98     20916
   macro avg       0.98      0.98      0.98     20916
weighted avg       0.98      0.98      0.98     20916
```

In [ ]:

```
from sklearn.metrics import confusion_matrix
# creating confusion matrix of our testing set
confusion_matrix(y_test, predicted_labels)
```

Out[ ]:

```
array([[ 949,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0],
       [   0,  938,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0],
       [   0,    0,  921,    0,    0,    0,    0,    0,    0,    0,    5,
            0,    0,   35,    0,    0,    0,    1,    0,    0,    0,    0],
       [   0,    0,    0,  981,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,  985,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    1,    0,    8,    0,    0],
       [   0,    0,    0,    0,    0, 1003,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    1,    0],
       [   0,    0,    0,    0,    0,    0,  943,    0,    0,    0,    0,
           24,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    0,    0,    0,  968,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    0,    8,    0,    0,  923,    0,    0,
            0,    0,    0,    0,    0,    0,    4,    0,    0,   29,    0],
       [   0,    0,    0,    0,    0,    0,    0,    0,    0,  948,    0,
            0,    0,    0,    0,    0,    0,    0,    1,    0,    0,    0],
       [   0,    0,    4,    0,    0,    0,    0,    0,    0,    0,  952,
            0,    0,   14,    0,    0,    0,    0,    0,    0,    0,    0],
```

```
array([[  0,   1,   0,   0,   0,   0,  14,   0,   0,   0,   0,
        901,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0, 993,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,  33,   0,   0,   0,   0,   0,   0,   0,  21,
          0,   0, 863,   0,   0,   0,   0,   1,   0,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0, 887,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0, 918,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0, 1011,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0,   5,   0,   0,   0,   1,   0,   4,
          0,   0,   0,   3,   0,   0, 897,   0,   2,   7,   0],
       [  0,   0,   3,   0,   0,   0,   0,   0,   0,  14,   0,
          0,   0,   0,   0,   0,   0,   0, 888,   0,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0, 938,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  93,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0, 846,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 926]])
```

In [ ]:

```
from sklearn.svm import SVC
svmClassifier = SVC(kernel='poly',degree=2,tol = 0.001,random_state=0)
# training our model
svmClassifier.fit(x_train,y_train)
# predicting labels of testing set
predicted_labels = svmClassifier.predict(x_test)
# accuaray of our model on testing set
score = metrics.accuracy_score(y_test, predicted_labels)

print("SVM accuray is: ", score*100, "%")

print(classification_report(y_test,predicted_labels))

model_name.append("SVM")
accuracy_of_model.append(score*100)
```

```
SVM accuray is:  96.91145534519029 %
              precision    recall  f1-score   support

       apple       1.00      1.00      1.00       949
      banana       0.99      1.00      1.00       938
   blackgram       0.90      0.91      0.91       962
    chickpea       1.00      1.00      1.00       981
     coconut       1.00      0.96      0.98       994
      coffee       0.99      1.00      1.00      1004
      cotton       0.97      0.98      0.97       967
      grapes       1.00      1.00      1.00       968
        jute       0.86      0.96      0.91       964
 kidneybeans       0.96      0.99      0.97       949
      lentil       0.95      0.95      0.95       970
       maize       0.97      0.96      0.97       916
       mango       0.99      1.00      1.00       993
   mothbeans       0.88      0.87      0.88       918
    mungbean       0.97      1.00      0.99       887
   muskmelon       0.99      1.00      1.00       918
      orange       1.00      1.00      1.00      1011
      papaya       1.00      0.97      0.98       919
  pigeonpeas       0.98      0.93      0.96       905
 pomegranate       0.97      1.00      0.98       938
        rice       0.96      0.84      0.90       939
  watermelon       1.00      0.99      1.00       926
```

```
              accuracy                           0.97     20916
             macro avg       0.97      0.97      0.97     20916
          weighted avg       0.97      0.97      0.97     20916
```

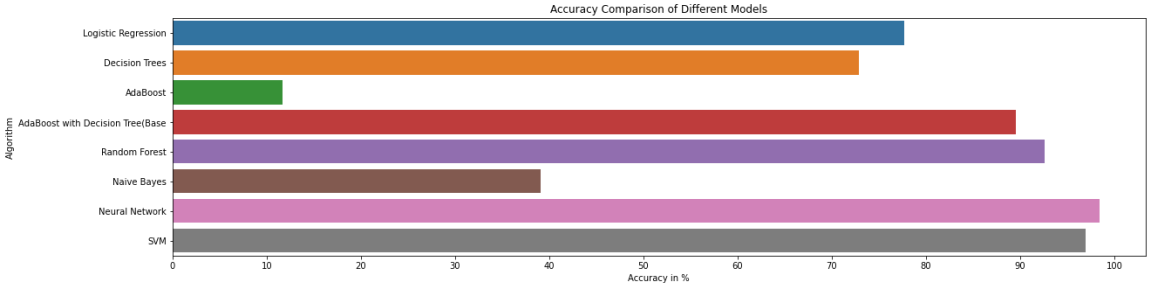
In [ ]:
```
from sklearn.metrics import confusion_matrix
# creating confusion matrix of our testing set
confusion_matrix(y_test, predicted_labels)
```

Out[ ]:
```
array([[ 949,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0],
       [   0,  936,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            2,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0],
       [   0,    1,  878,    0,    0,    0,    0,    0,    0,    0,   13,
            0,    0,   70,    0,    0,    0,    0,    0,    0,    0,    0],
       [   0,    0,    0,  981,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,  958,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    5,    0,    0,   31,    0,    0],
       [   0,    0,    0,    0,    0, 1003,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    1,    0],
       [   0,    0,    0,    0,    0,    0,  944,    0,    0,    0,    0,
           23,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    0,    0,    0,  968,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    0,    8,    0,    0,  927,    0,    0,
            0,    0,    0,    0,    0,    0,    2,    0,    0,   27,    0],
       [   0,    0,    0,    0,    0,    0,    0,    0,    0,  935,    0,
            0,    0,    0,    0,    0,    0,    0,   14,    0,    0,    0],
       [   0,    0,   14,    0,    0,    0,    0,    0,    0,    0,  921,
            0,    0,   35,    0,    0,    0,    0,    0,    0,    0,    0],
       [   0,    3,    0,    0,    0,    1,   31,    0,    0,    0,    0,
          881,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,  993,    0,    0,    0,    0,    0,    0,    0,    0,    0],
       [   0,    0,   71,    0,    0,    0,    0,    0,    0,    0,   36,
            0,    2,  800,    9,    0,    0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,  887,    0,    0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,  918,    0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0, 1011,    0,    0,    0,    0,    0],
       [   0,    1,    1,    0,    0,    0,    0,    0,    5,    0,    0,
            0,    0,    0,   17,    0,    0,  894,    0,    0,    1,    0],
       [   0,    0,   14,    0,    0,    0,    0,    0,    0,   44,    0,
            0,    4,    0,    0,    0,    0,    1,  842,    0,    0,    0],
       [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,  938,    0,    0],
       [   0,    0,    0,    0,    0,    0,    0,    0,  152,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,  787,    0],
       [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    7,    0,    0,    0,    0,    0,  919]])
```

In [ ]:
```
import matplotlib.pyplot as plt
fig = plt.figure(figsize = (20, 5))
# plotting accuracy of different model in same graph
plt.title('Accuracy Comparison of Different Models')
plt.ylabel('Algorithm')
plt.xlabel('Accuracy in %')
plt.xticks(range(0,101,10))
sns.barplot(y = model_name,x = accuracy_of_model)
```

```
print(model_name)
print(accuracy_of_model)
```

```
['Logistic Regression', 'Decision Trees', 'AdaBoost', 'AdaBoost with Decision T
ree(Base', 'Random Forest', 'Naive Bayes', 'Neural Network', 'SVM']
[77.68215720022948, 72.83897494740869, 11.713520749665328, 89.57257601835916, 9
2.60374832663989, 39.06100592847581, 98.38879326831133, 96.91145534519029]
```



In [ ]:

```
from sklearn import tree
fig = plt.figure(figsize=(100,100))  # updating figure size
tree.plot_tree(decisionTreeModel, filled=True)  # plotting DT
print()
```