Module 4 Gaussian Naive Bayes

In [1]:

```
import pandas as p
from sklearn.naive_bayes import GaussianNB
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score,
```

In [2]:

```
import warnings
warnings.filterwarnings('ignore')
```

In [3]:

```
data = p.read_csv('crop.csv')
```

In [4]:

```
df=data.dropna()
```

In [5]:

```
df.columns
```

Out[5]:

```
Index(['nitrogen', 'phosphorus', 'potassium', 'temperature', 'humidity', 'ph',
       'rainfall', 'label'],
      dtype='object')
```

In [6]:

```
var_mod = ['label']
le = LabelEncoder()
for i in var_mod:
    df[i] = le.fit_transform(df[i]).astype(int)
```

In [7]:

```
inputs = df.drop(labels='label', axis=1)
output = df.loc[:,'label']
```

In [8]:

```
X_train, X_test, y_train, y_test = train_test_split(inputs, output, test_size=0.3, r
print("Number of Training Dataset: ", len(X_train))
print("Number of Testing Dataset: ", len(X_test))
print("Total Number of Dataset: ", len(X_train)+len(X_test))
```

```
Number of Training Dataset:  1540
Number of Testing Dataset:  660
Total Number of Dataset:  2200
```

In [9]:

```
#Model Training
nb = GaussianNB()
nb.fit(X_train,y_train)
predicted_nb = nb.predict(X_test)
```

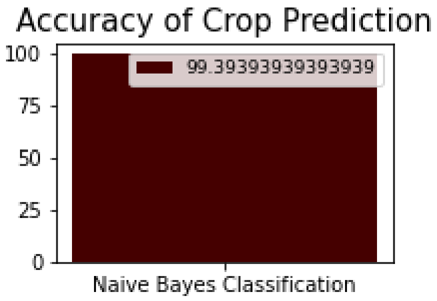
In [10]:

```
#Getting Accuracy
accuracy = accuracy_score(y_test,predicted_nb) # accuracy: (tp + tn) / (p + n)
print('Accuracy of Naive Bayes is: ',accuracy*100)
```

```
Accuracy of Naive Bayes is:  99.39393939393939
```

In [11]:

```
DT=accuracy.mean() *100
def graph():
    data=[DT]
    alg="Naive Bayes Classification"
    plt.figure(figsize=(3,2))
    b=plt.bar(alg,data,color=("#450000"))
    plt.title("Accuracy of Crop Prediction",fontsize=15)
    plt.legend(b,data,fontsize=9)
graph()
```


Accuracy of Crop Prediction

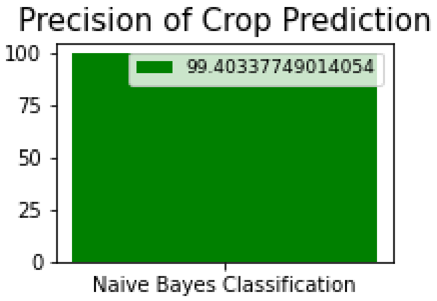
In [12]:

```
#Getting Precision
precision = precision_score(y_test,predicted_nb,average='weighted') # precision tp /
print('Precision of Naive Bayes is: ',precision*100)
```

```
Precision of Naive Bayes is:  99.40337749014054
```

In [13]:

```
DT=precision.mean() *100
def graph():
    data=[DT]
    alg="Naive Bayes Classification"
    plt.figure(figsize=(3,2))
    b=plt.bar(alg,data,color=("green"))
    plt.title("Precision of Crop Prediction",fontsize=15)
    plt.legend(b,data,fontsize=9)
graph()
```


Precision of Crop Prediction

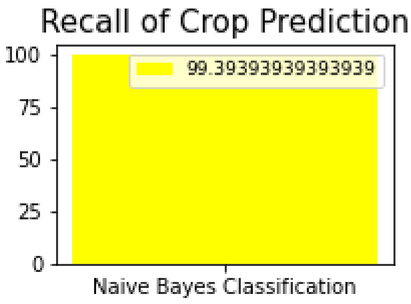
In [14]:

```
#Getting Recall
recall = recall_score(y_test,predicted_nb,average='weighted') # recall: tp / (tp + f
print('Recall of Naive Bayes is: ',recall*100)
```

```
Recall of Naive Bayes is:  99.39393939393939
```

In [15]:

```
DT=recall.mean() *100
def graph():
    data=[DT]
    alg="Naive Bayes Classification"
    plt.figure(figsize=(3,2))
    b=plt.bar(alg,data,color=("yellow"))
    plt.title("Recall of Crop Prediction",fontsize=15)
    plt.legend(b,data,fontsize=9)
graph()
```



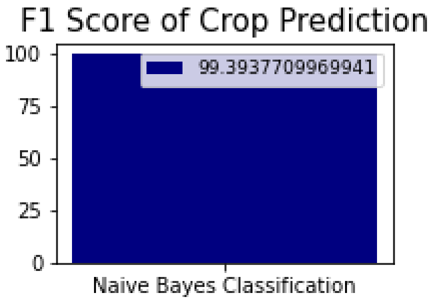
In [16]:

```
#Getting F1 Score
f1 = f1_score(y_test,predicted_nb,average='weighted') # f1: 2 tp / (2 tp + fp + fn)
print('F1 Score of Naive Bayes is: ',f1*100)
```

```
F1 Score of Naive Bayes is:  99.3937709969941
```

In [17]:

```
DT=f1.mean() *100
def graph():
    data=[DT]
    alg="Naive Bayes Classification"
    plt.figure(figsize=(3,2))
    b=plt.bar(alg,data,color=("#000080"))
    plt.title("F1 Score of Crop Prediction",fontsize=15)
    plt.legend(b,data,fontsize=9)
graph()
```



In [18]:

```
#Classification Report
cr = classification_report(y_test,predicted_nb)
print('Classification report\n',cr)
```

```
Classification report
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        30
           1       1.00      1.00      1.00        30
           2       1.00      1.00      1.00        30
           3       1.00      1.00      1.00        30
           4       1.00      1.00      1.00        30
           5       1.00      1.00      1.00        30
           6       0.97      1.00      0.98        30
           7       1.00      1.00      1.00        30
           8       0.97      0.93      0.95        30
           9       1.00      1.00      1.00        30
          10       1.00      1.00      1.00        30
          11       1.00      0.97      0.98        30
          12       1.00      1.00      1.00        30
          13       1.00      1.00      1.00        30
          14       1.00      1.00      1.00        30
          15       1.00      1.00      1.00        30
          16       1.00      1.00      1.00        30
          17       1.00      1.00      1.00        30
          18       1.00      1.00      1.00        30
          19       1.00      1.00      1.00        30
          20       0.94      0.97      0.95        30
          21       1.00      1.00      1.00        30

    accuracy                           0.99       660
   macro avg       0.99      0.99      0.99       660
weighted avg       0.99      0.99      0.99       660
```

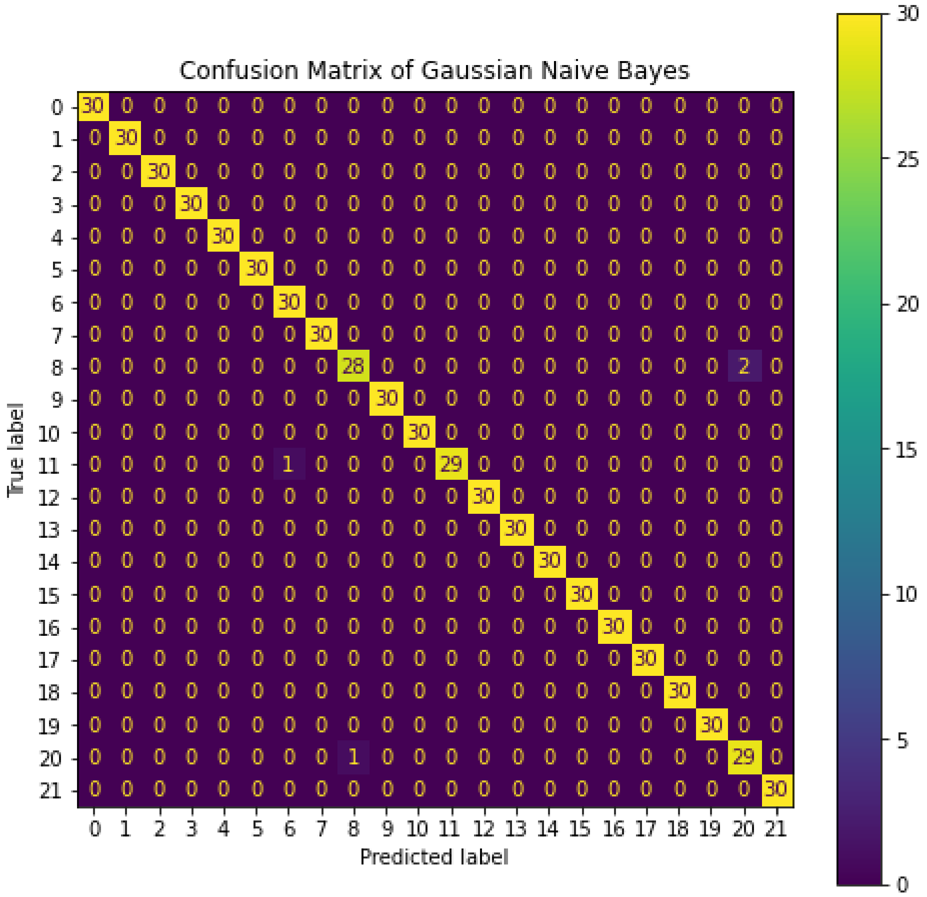
In [19]:

```
#Confusion Matrix
cm = confusion_matrix(y_test,predicted_nb)
print('Confusion matrix\n',cm)
```

```
Confusion matrix
 [[30  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0 30  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0 30  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0 30  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0 30  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0 30  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0 30  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0 30  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0 28  0  0  0  0  0  0  0  0  0  0  0  2  0]
 [ 0  0  0  0  0  0  0  0  0 30  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0 30  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  1  0  0  0  0 29  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0 30  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0 30  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0 30  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 30  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 30  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 30  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 30  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 30  0  0]
 [ 0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0 29  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 30]]
```

In [20]:

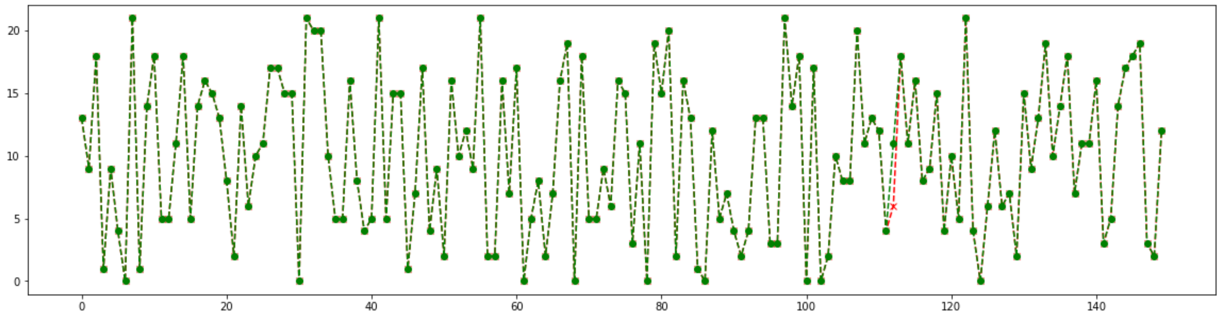
```
fig, ax = plt.subplots(figsize=(8,8))
plot_confusion_matrix(nb, X_test, y_test, ax=ax)
plt.title('Confusion Matrix of Gaussian Naive Bayes')
plt.show()
```



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	30	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	28	0	0	0	0	0	0	0	0	0	0	0	2	0
9	0	0	0	0	0	0	0	0	0	30	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	30	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	1	0	0	0	0	29	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	30	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	30	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	30	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	30	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	30	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	30	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	30	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	30	0	0
20	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	29	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	30

In [21]:

```
DF = p.DataFrame()
DF["y_test"] = y_test
DF["predicted"] = predicted_nb
DF.reset_index(inplace=True)
plt.figure(figsize=(20, 5))
plt.plot(DF["predicted"][:150], marker='x', linestyle='dashed', color='red')
plt.plot(DF["y_test"][:150],  marker='o', linestyle='dashed', color='green')
plt.show()
```



In [22]:

```
#Saving Model
import joblib
joblib.dump(nb,'nb.pkl')
```

Out[22]:

```
['nb.pkl']
```