In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import model_selection
from sklearn import preprocessing
from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import make_pipeline
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.ensemble import BaggingClassifier
```

In [2]:
```
df = pd.read_csv('Crop_recommendation (3).csv')
```

In [3]:
```
df.head()
```

Out[3]:

	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 [4]:
```
print("Shape of the dataframe: ",df.shape)
df.isna().sum()
```

```
Shape of the dataframe:  (2200, 8)
```

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

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2200 entries, 0 to 2199
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   N            2200 non-null   int64  
 1   P            2200 non-null   int64  
 2   K            2200 non-null   int64  
 3   temperature  2200 non-null   float64
 4   humidity     2200 non-null   float64
 5   ph           2200 non-null   float64
 6   rainfall     2200 non-null   float64
 7   label        2200 non-null   object 
dtypes: float64(4), int64(3), object(1)
memory usage: 137.6+ KB
```

In [6]:
```
df.describe()
```

Out[6]:

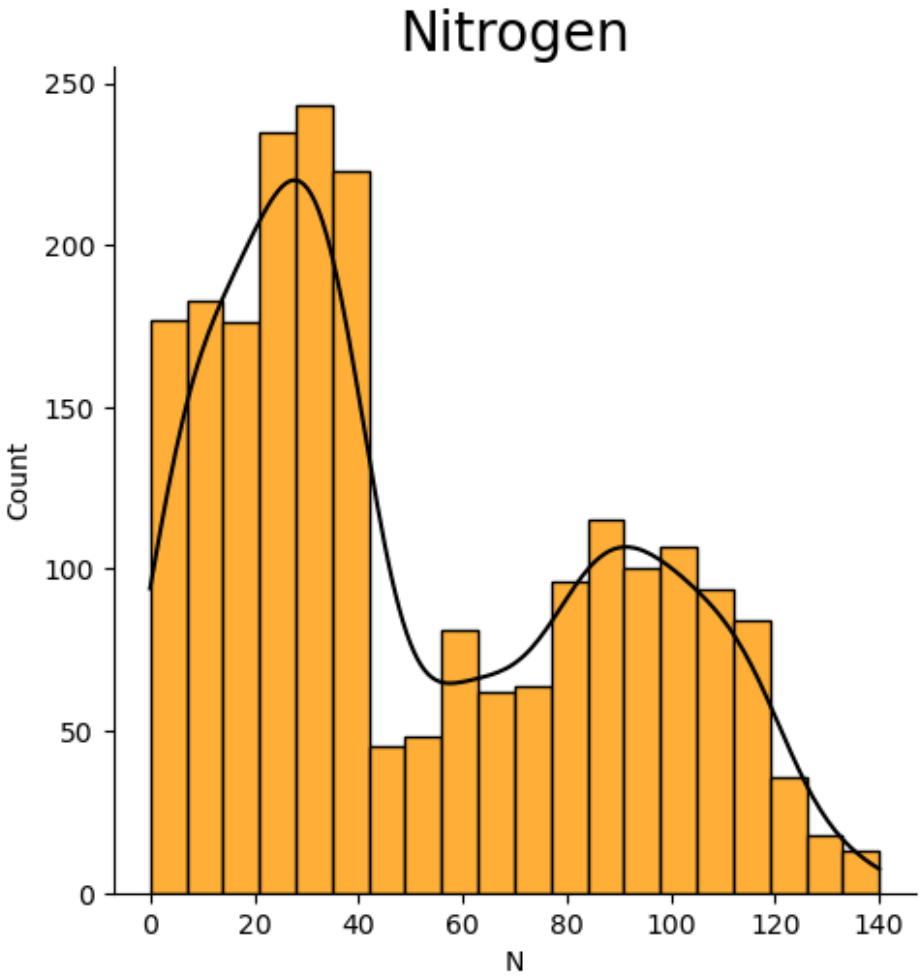
	N	P	K	temperature	humidity	ph	rainfall
count	2200.000000	2200.000000	2200.000000	2200.000000	2200.000000	2200.000000	2200.000000
mean	50.551818	53.362727	48.149091	25.616244	71.481779	6.469480	103.463655
std	36.917334	32.985883	50.647931	5.063749	22.263812	0.773938	54.958389
min	0.000000	5.000000	5.000000	8.825675	14.258040	3.504752	20.211267
25%	21.000000	28.000000	20.000000	22.769375	60.261953	5.971693	64.551686
50%	37.000000	51.000000	32.000000	25.598693	80.473146	6.425045	94.867624
75%	84.250000	68.000000	49.000000	28.561654	89.948771	6.923643	124.267508
max	140.000000	145.000000	205.000000	43.675493	99.981876	9.935091	298.560117

In [7]:
```
df.dtypes
```

Out[7]:
```
N                int64
P                int64
K                int64
temperature    float64
humidity       float64
ph             float64
rainfall       float64
label           object
dtype: object
```

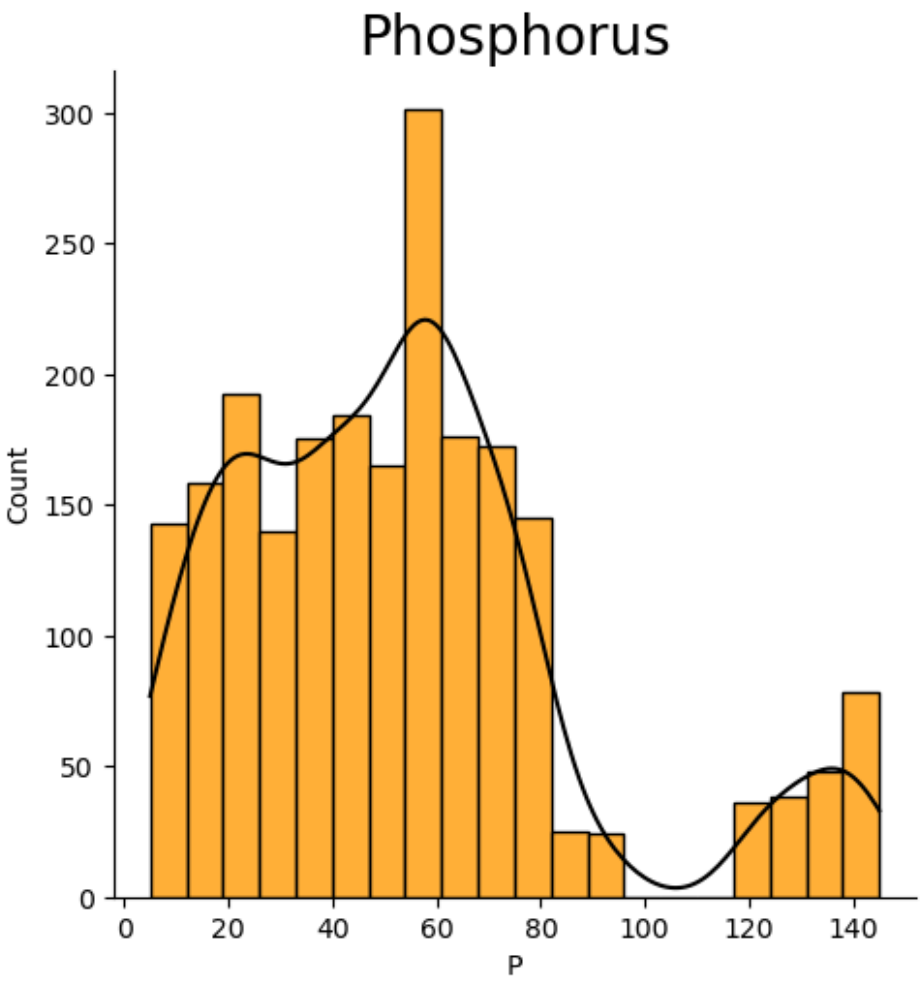
In [9]:

```
sns.displot(x=df['N'], bins=20,kde=True,edgecolor="black",color='black',facecolor='#ffb
plt.title("Nitrogen",size=20)
plt.show()
```



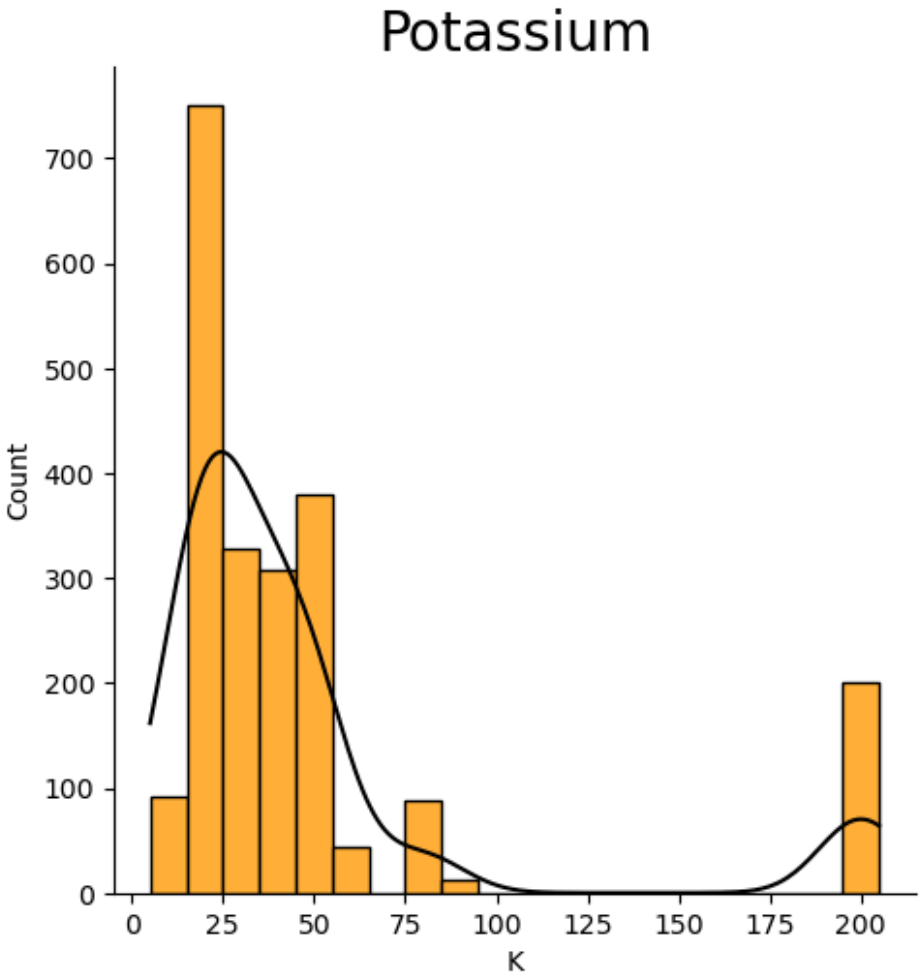
In [10]:

```
sns.displot(x=df['P'],bins=20,color='black',edgecolor='black',kde=True,facecolor='#ffb0
plt.title("Phosphorus", size=20)
plt.xticks(range(0,150,20))
plt.show()
```



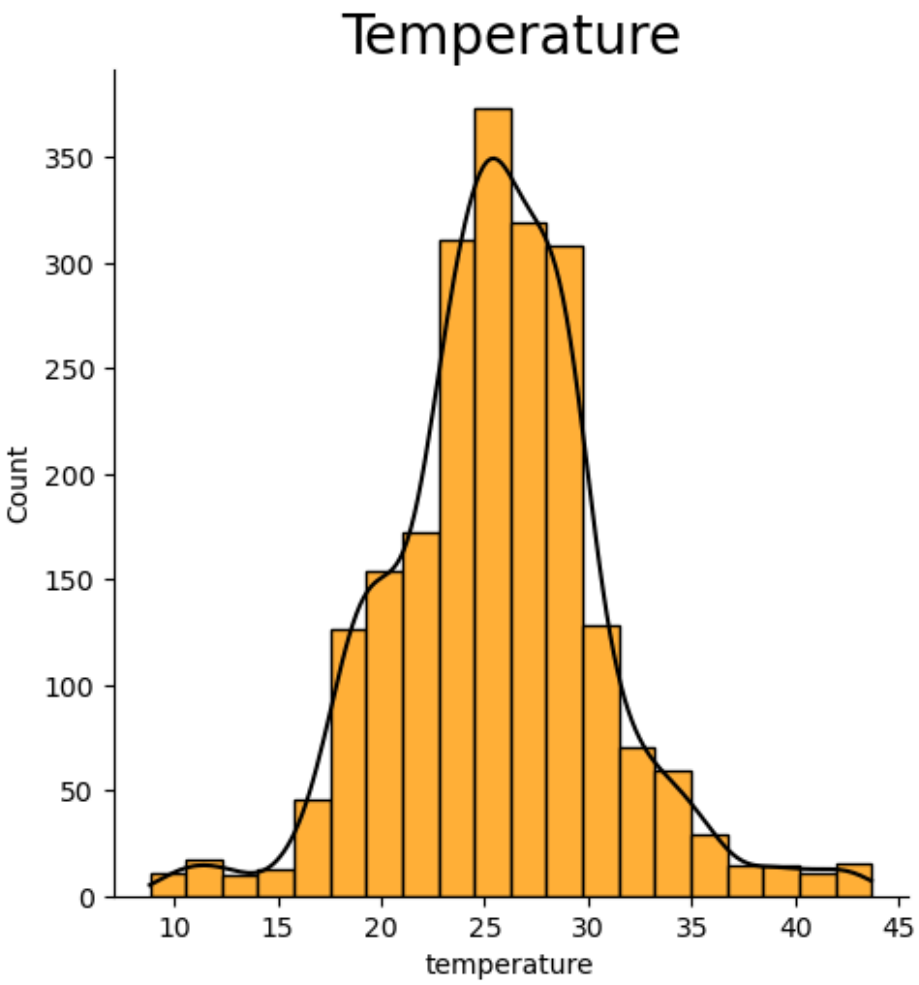
In [11]:

```
sns.displot(x=df['K'],kde=True, bins=20, facecolor='#ffb03b',edgecolor='black', color='
plt.title("Potassium",size=20)
plt.show()
```



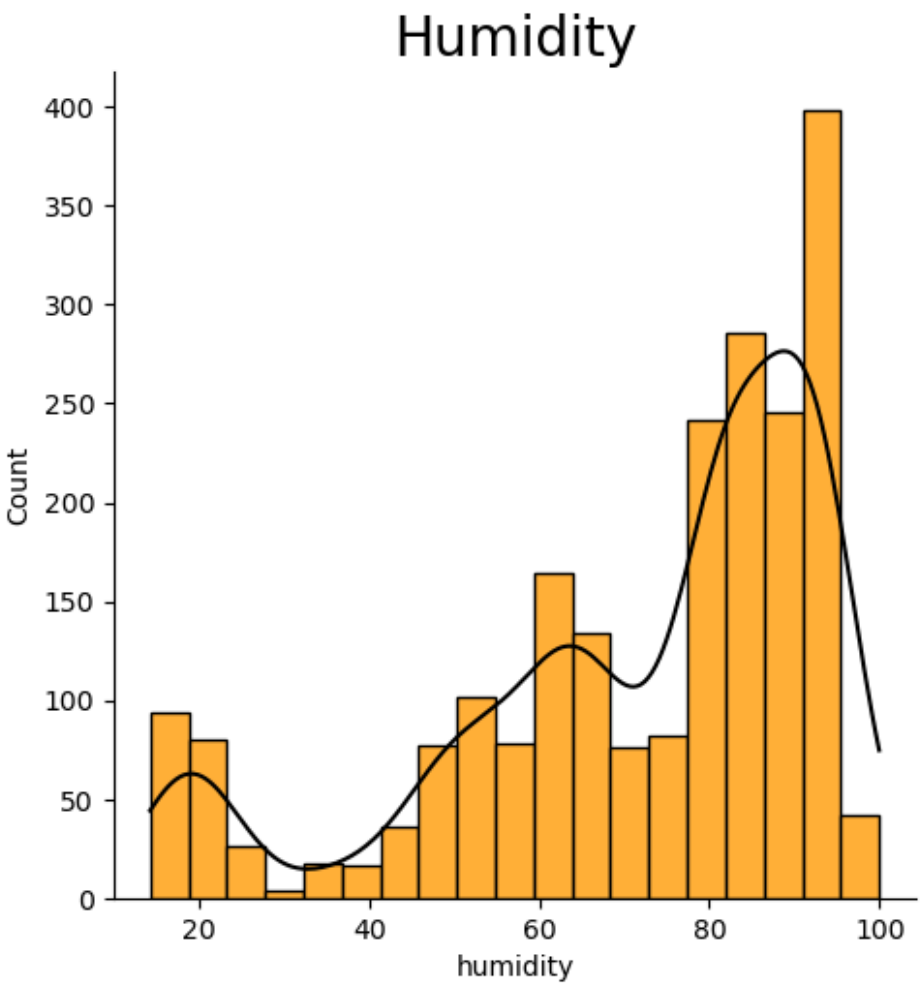
In [12]:

```
sns.displot(x=df['temperature'], bins=20,kde=True,edgecolor="black",color='black',facec
plt.title("Temperature",size=20)
plt.show()
```



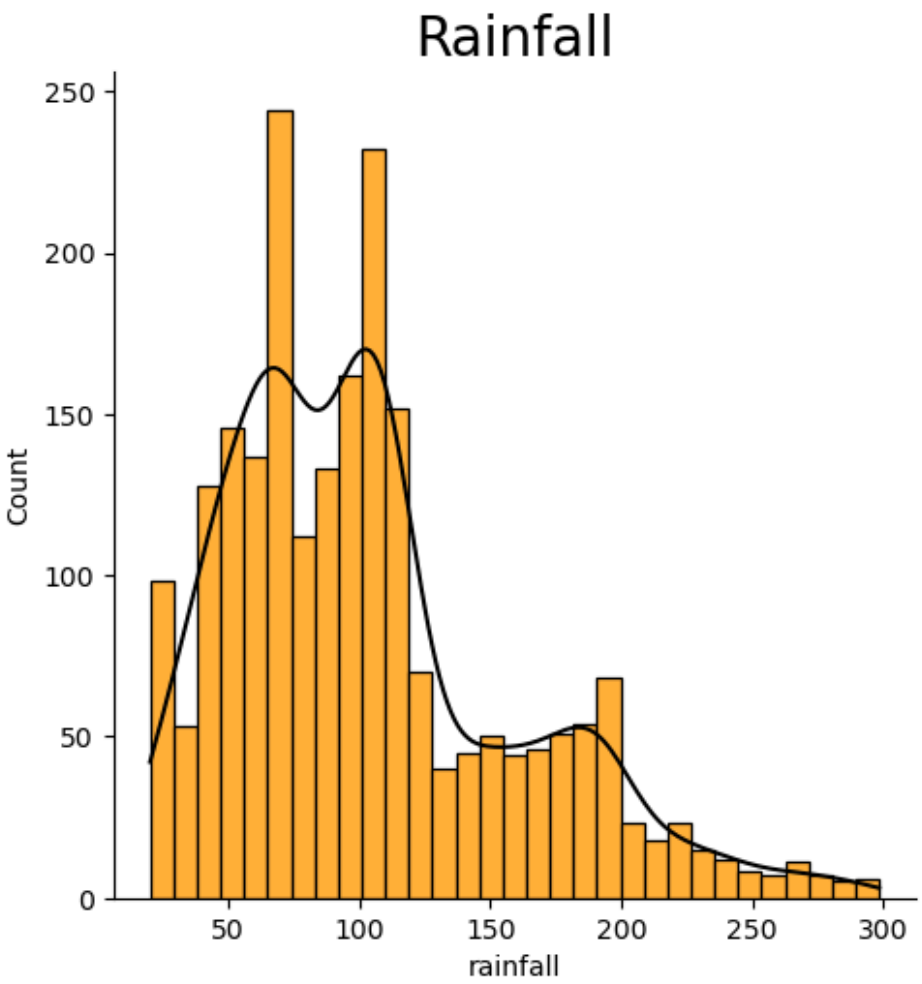
In [13]:

```
sns.displot(x=df['humidity'], color='black',facecolor='#ffb03b',kde=True,edgecolor='bla
plt.title("Humidity",size=20)
plt.show()
```



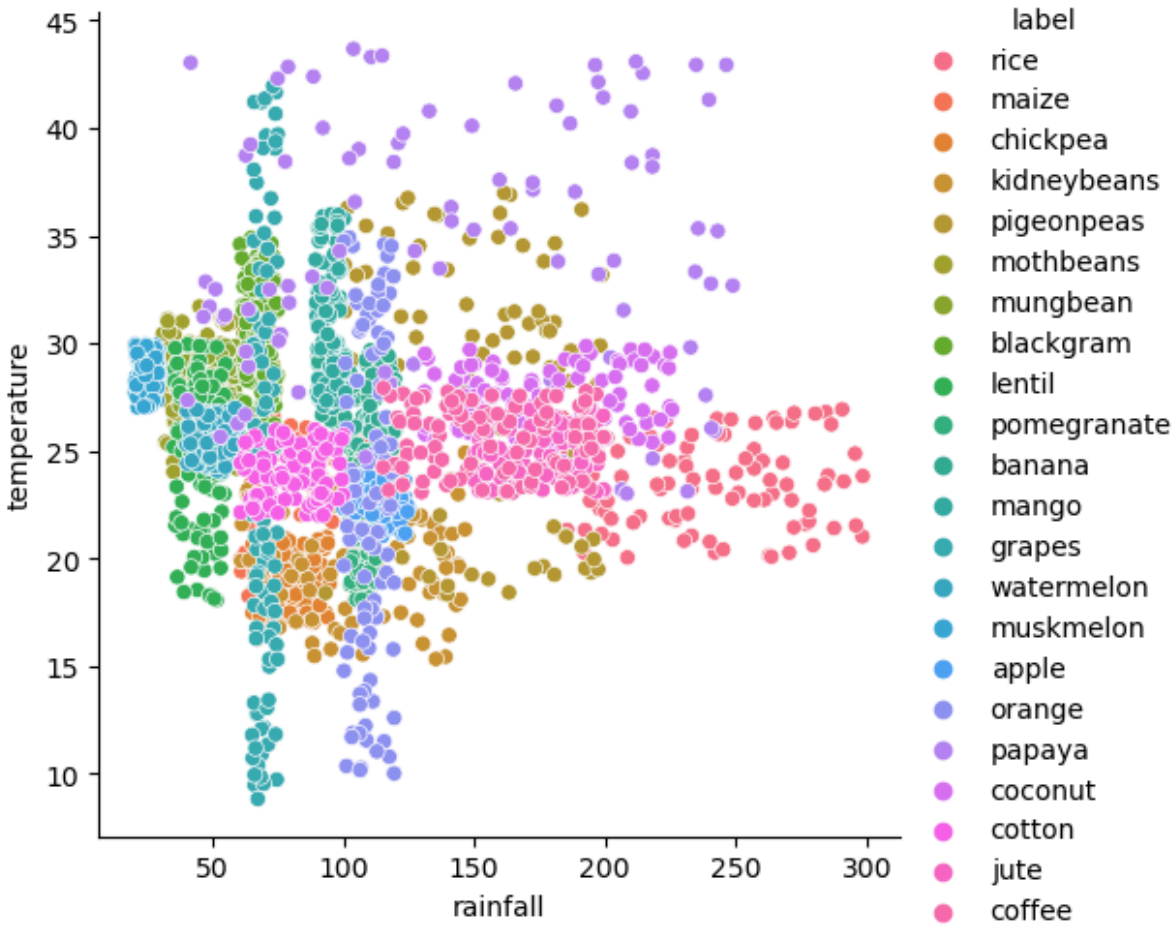
In [14]:

```
sns.displot(x=df['rainfall'], color='black',facecolor='#ffb03b',kde=True,edgecolor='bla
plt.title("Rainfall",size=20)
plt.show()
```



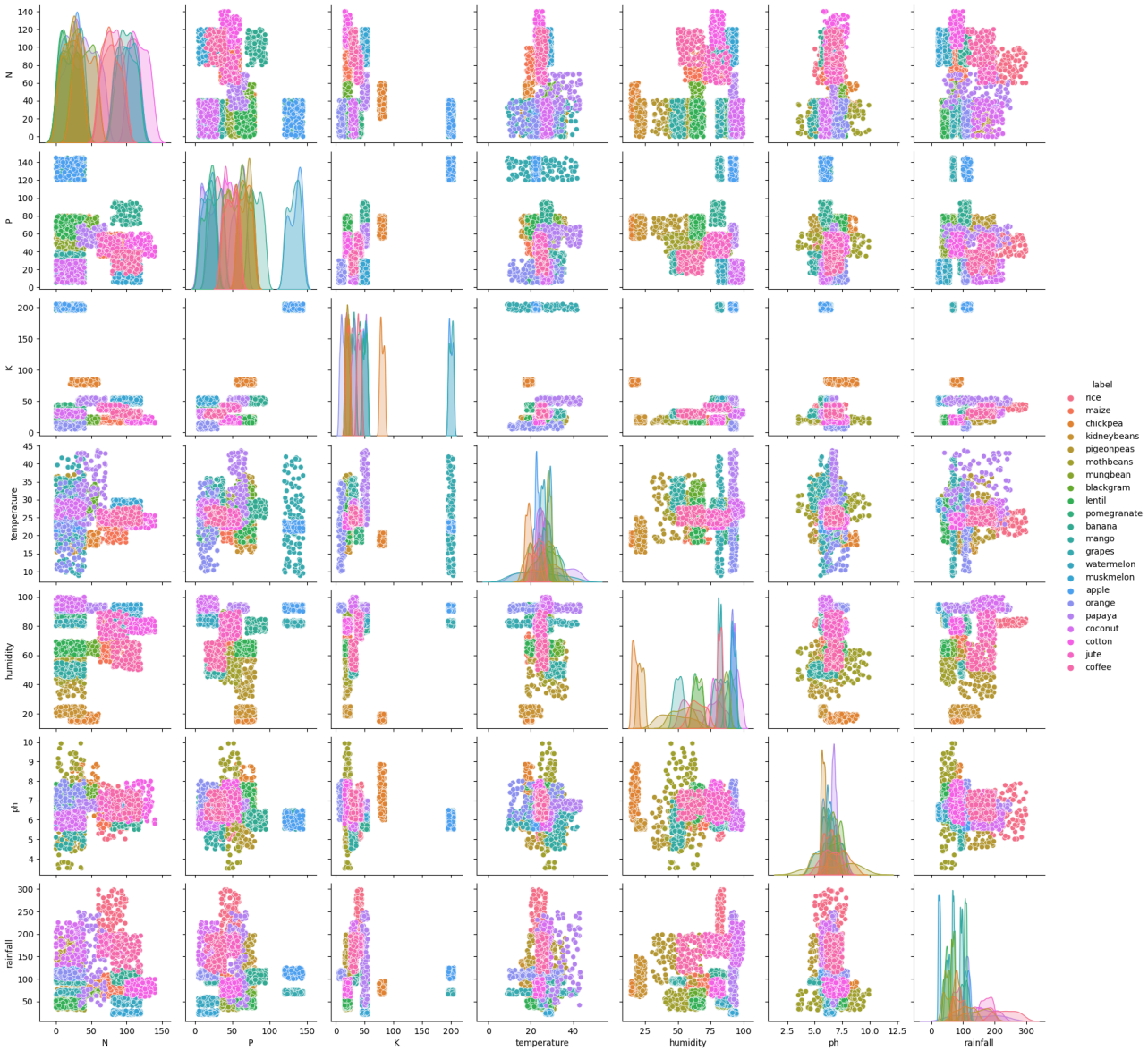
In [15]:

```
sns.relplot(x='rainfall',y='temperature',data=df,kind='scatter',hue='label',height=5)
plt.show()
```



In [20]:

```
sns.pairplot(data=df,hue='label')
plt.show()
```



In [21]:
```
# Unique values in the label column

crops = df['label'].unique()
print(len(crops))
print(crops)
print(pd.value_counts(df['label']))
22
```

```
22
['rice' 'maize' 'chickpea' 'kidneybeans' 'pigeonpeas' 'mothbeans'
 'mungbean' 'blackgram' 'lentil' 'pomegranate' 'banana' 'mango' 'grapes'
 'watermelon' 'muskmelon' 'apple' 'orange' 'papaya' 'coconut' 'cotton'
 'jute' 'coffee']
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
```

Out[21]: 22

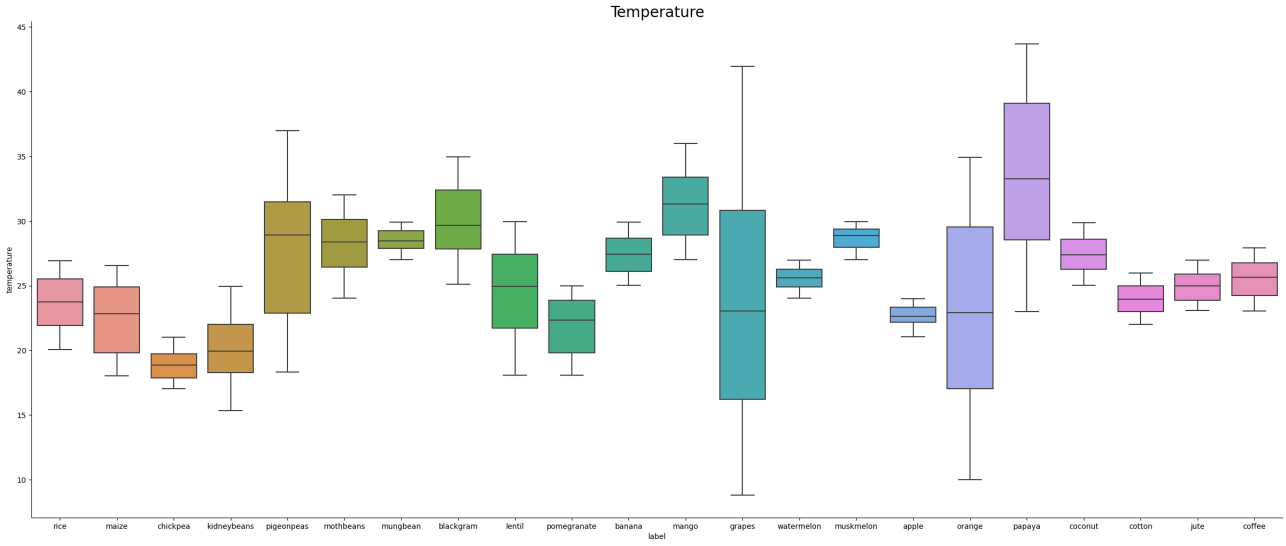
In [22]:
```
# Filtering each unique label and store it in a list df2 for to plot the box plot

df2=[]
for i in crops:
    df2.append(df[df['label'] == i])
df2[1].head()
```

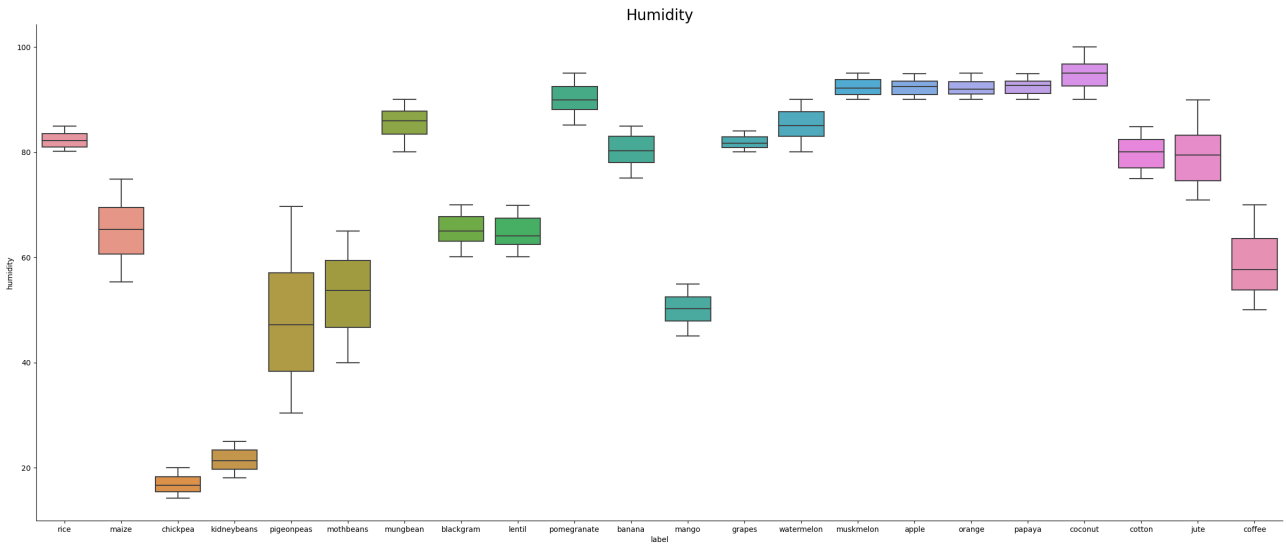
Out[22]:

	N	P	K	temperature	humidity	ph	rainfall	label
100	71	54	16	22.613600	63.690706	5.749914	87.759539	maize
101	61	44	17	26.100184	71.574769	6.931757	102.266244	maize
102	80	43	16	23.558821	71.593514	6.657965	66.719955	maize
103	73	58	21	19.972160	57.682729	6.596061	60.651715	maize
104	61	38	20	18.478913	62.695039	5.970458	65.438354	maize

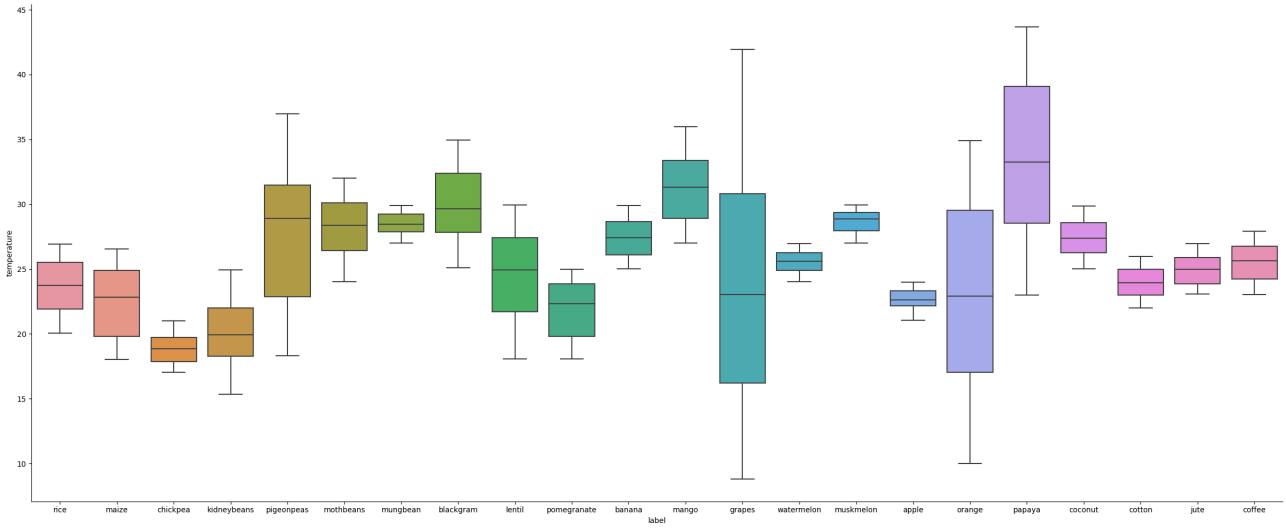
In [23]:
```
sns.catplot(data=df, x='label', y='temperature', kind='box', height=10, aspect=20/8.27)
# plt.xticks(rotation='vertical')
plt.title("Temperature", size=20)
plt.show()
```



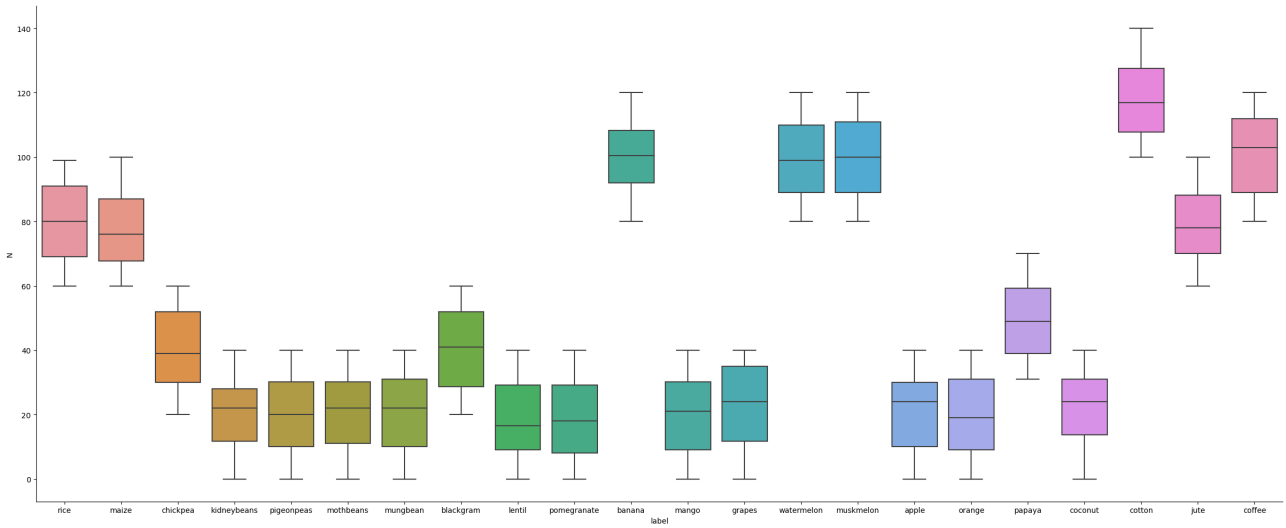
In [24]:
```
sns.catplot(data=df, x='label', y='humidity', kind='box', height=10, aspect=20/8.27)
# plt.xticks(rotation='vertical')
plt.title("Humidity", size=20)
plt.show()
```



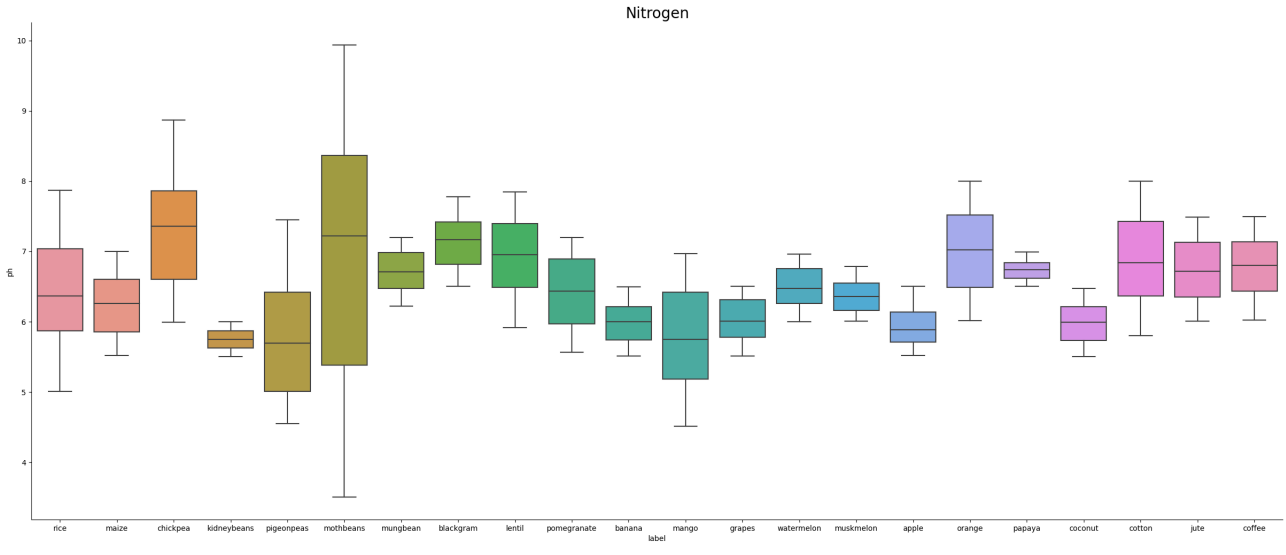
In [25]:
```
sns.catplot(data=df, x='label', y='temperature', kind='box', height=10, aspect=20/8.27)
plt.show()
```



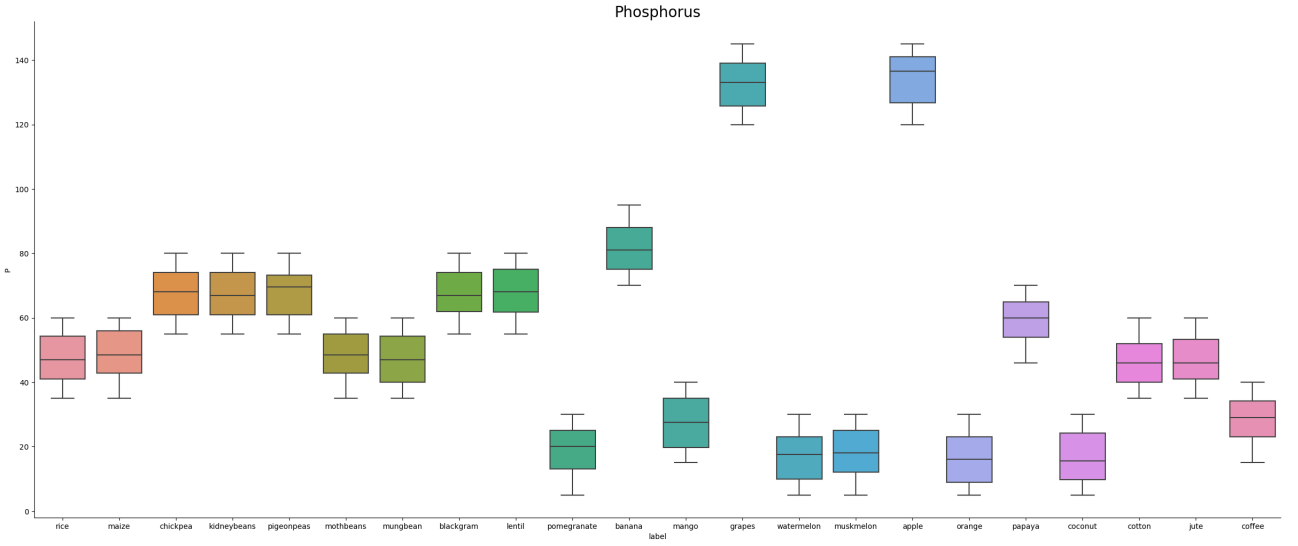
In [26]:
```
sns.catplot(data=df, x='label', y='N', kind='box', height=10, aspect=20/8.27)
# plt.xticks(rotation='vertical')
plt.show()
```



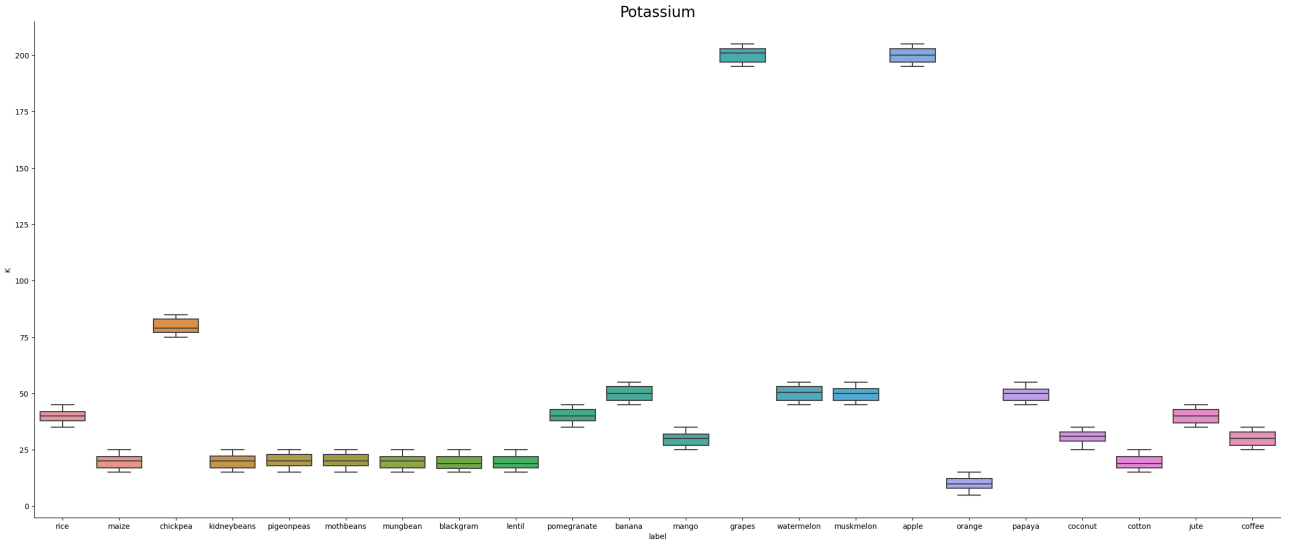
In [27]:
```
sns.catplot(data=df, x='label', y='ph', kind='box', height=10, aspect=20/8.27)
# plt.xticks(rotation='vertical')
plt.title("Nitrogen",size=20)
plt.show()
```



In [28]:
```
sns.catplot(data=df, x='label', y='P', kind='box', height=10, aspect=20/8.27)
# plt.xticks(rotation='vertical')
plt.title("Phosphorus",size=20)
plt.show()
```



In [29]:
```
sns.catplot(data=df, x='label', y='K', kind='box', height=10, aspect=20/8.27)
# plt.xticks(rotation='vertical')
plt.title("Potassium",size=20)
plt.show()
```



In [30]:

```
def detect_outlier(x):
    q1 = x.quantile(0.25)
    q3 = x.quantile(0.75)
    IQR = q3-q1
    lower_limit = q1 - (1.5*IQR)
    upper_limit = q3 + (1.5*IQR)
    print(f"Lower limit: {lower_limit} Upper limit: {upper_limit}")
    print(f"Minimum value: {x.min()}   MAximum Value: {x.max()}")
    for i in [x.min(),x.max()]:
        if i == x.min():
            if lower_limit > x.min():
                print("Lower limit failed - Need to remove minimum value")
            elif lower_limit < x.min():
                print("Lower limit passed - No need to remove outlier")
        elif i == x.max():
            if upper_limit > x.max():
                print("Upper limit passed - No need to remove outlier")
            elif upper_limit < x.max():
                print("Upper limit failed - Need to remove maximum value")
detect_outlier(df['K'][df['label']=='grapes'])
```

```
Lower limit: 188.0 Upper limit: 212.0
Minimum value: 195   MAximum Value: 205
Lower limit passed - No need to remove outlier
Upper limit passed - No need to remove outlier
```

In [31]:

```
for i in df['label'].unique():
    detect_outlier(df['K'][df['label']==i])
    print('---------------------------------------------')
```

```
Lower limit: 23.0 Upper limit: 39.0
Minimum value: 25   MAximum Value: 35
Lower limit passed - No need to remove outlier
Upper limit passed - No need to remove outlier
---------------------------------------------
Lower limit: 9.5 Upper limit: 29.5
Minimum value: 15   MAximum Value: 25
Lower limit passed - No need to remove outlier
Upper limit passed - No need to remove outlier
---------------------------------------------
Lower limit: 28.0 Upper limit: 52.0
Minimum value: 35   MAximum Value: 45
Lower limit passed - No need to remove outlier
Upper limit passed - No need to remove outlier
---------------------------------------------
Lower limit: 18.0 Upper limit: 42.0
Minimum value: 25   MAximum Value: 35
Lower limit passed - No need to remove outlier
Upper limit passed - No need to remove outlier
---------------------------------------------
```

In [32]:
```
x = df.drop(['label'], axis=1)
x.head()
```

Out[32]:

	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 [33]:
```
Y = df['label']
encode = preprocessing.LabelEncoder()
y = encode.fit_transform(Y)
print("Label length: ",len(y))
```

```
Label length:  2200
```

In [34]:
```
x_train,x_test,y_train,y_test = model_selection.train_test_split(x,y)
print(len(x_train),len(y_train),len(x_test),len(y_test))
```

```
1650 1650 550 550
```

In [35]:
```
a={'decision tree' : {
        'model' : DecisionTreeClassifier(criterion='gini'),
        'params':{'decisiontreeclassifier__splitter':['best','random']}
    },
    'svm': {
        'model': SVC(gamma='auto',probability=True),
        'params' : {
            'svc__C': [1,10,100,1000],
            'svc__kernel': ['rbf','linear']
        }
    },
    'random_forest': {
        'model': RandomForestClassifier(),
        'params' : {
            'randomforestclassifier__n_estimators': [1,5,10]
        }
    },
   'k classifier':{
       'model':KNeighborsClassifier(),
       'params':{'kneighborsclassifier__n_neighbors':[5,10,20,25],'kneighborsclassifier
   }
}
```

In [37]:

```
score=[]
details = []
best_param = {}
for mdl,par in a.items():
    pipe = make_pipeline(preprocessing.StandardScaler(),par['model'])
    res = model_selection.GridSearchCV(pipe,par['params'],cv=5)
    res.fit(x_train,y_train)
    score.append({
        'Model name':mdl,
        'Best score':res.best_score_,
        'Best param':res.best_params_
    })
    details.append(pd.DataFrame(res.cv_results_))
    best_param[mdl]=res.best_estimator_
pd.DataFrame(score)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:22
8: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the
default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.
11.0, this behavior will change: the default value of `keepdims` will become Fals
e, the `axis` over which the statistic is taken will be eliminated, and the value
None will no longer be accepted. Set `keepdims` to True or False to avoid this war
ning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:22
8: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the
default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.
11.0, this behavior will change: the default value of `keepdims` will become Fals
e, the `axis` over which the statistic is taken will be eliminated, and the value
None will no longer be accepted. Set `keepdims` to True or False to avoid this war
ning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:22
8: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the
default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.
```

In [38]:

```
details[0]
```

Out[38]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_decisiontreeclassifier__splitter	
0	0.018634	0.005033	0.003656	0.000798	best	{'decis
1	0.007073	0.000298	0.002580	0.000599	random	{'decis

In [39]:
```
details[1]
```

Out[39]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_svc__C	param_svc__kernel	pa
0	0.370423	0.008335	0.056366	0.000980	1	rbf	{'svc__ 'svc__k
1	0.174564	0.006260	0.014651	0.001874	1	linear	{'svc__ 'svc__k 'li
2	0.311651	0.006451	0.040585	0.001002	10	rbf	{'svc__C 'svc__k
3	0.173808	0.011086	0.013291	0.001478	10	linear	{'svc__C 'svc__k 'li
4	0.349348	0.019494	0.045662	0.003793	100	rbf	{'svc 'svc__k
5	0.149991	0.019179	0.010725	0.000851	100	linear	{'svc 'svc__k 'li
6	0.236037	0.003388	0.029210	0.000880	1000	rbf	{'svc 'svc__k
7	0.146957	0.010354	0.009754	0.001122	1000	linear	{'svc 'svc__k 'li

In [40]:
```
details[2]
```

Out[40]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_randomforestclassifier__n_estimators
0	0.006342	0.000524	0.001600	0.000378	1
1	0.015448	0.001420	0.002744	0.000655	5
2	0.027306	0.001027	0.003235	0.000296	10

In [41]:
```
details[3]
```

Out[41]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_kneighborsclassifier__n_neighbors	p
0	0.004031	0.000570	0.013564	0.002943	5	
1	0.004054	0.000097	0.005203	0.000768	5	
2	0.004040	0.000326	0.012161	0.000672	10	
3	0.005100	0.001430	0.006823	0.000700	10	
4	0.004549	0.001040	0.015157	0.002799	20	
5	0.004486	0.000945	0.009369	0.002889	20	
6	0.005139	0.002240	0.015303	0.002844	25	
7	0.004256	0.000711	0.008106	0.000564	25	

In [42]:
```
score
```

Out[42]:
```
[{'Model name': 'decision tree',
  'Best score': 0.9818181818181818,
  'Best param': {'decisiontreeclassifier__splitter': 'best'}},
 {'Model name': 'svm',
  'Best score': 0.9903030303030302,
  'Best param': {'svc__C': 100, 'svc__kernel': 'linear'}},
 {'Model name': 'random_forest',
  'Best score': 0.990909090909091,
  'Best param': {'randomforestclassifier__n_estimators': 10}},
 {'Model name': 'k classifier',
  'Best score': 0.9763636363636363,
  'Best param': {'kneighborsclassifier__n_neighbors': 5,
   'kneighborsclassifier__weights': 'distance'}}]
```

In [43]:
```
pd.DataFrame(score)
```

Out[43]:

	Model name	Best score	Best param
0	decision tree	0.981818	{'decisiontreeclassifier__splitter': 'best'}
1	svm	0.990303	{'svc__C': 100, 'svc__kernel': 'linear'}
2	random_forest	0.990909	{'randomforestclassifier__n_estimators': 10}
3	k classifier	0.976364	{'kneighborsclassifier__n_neighbors': 5, 'knei...

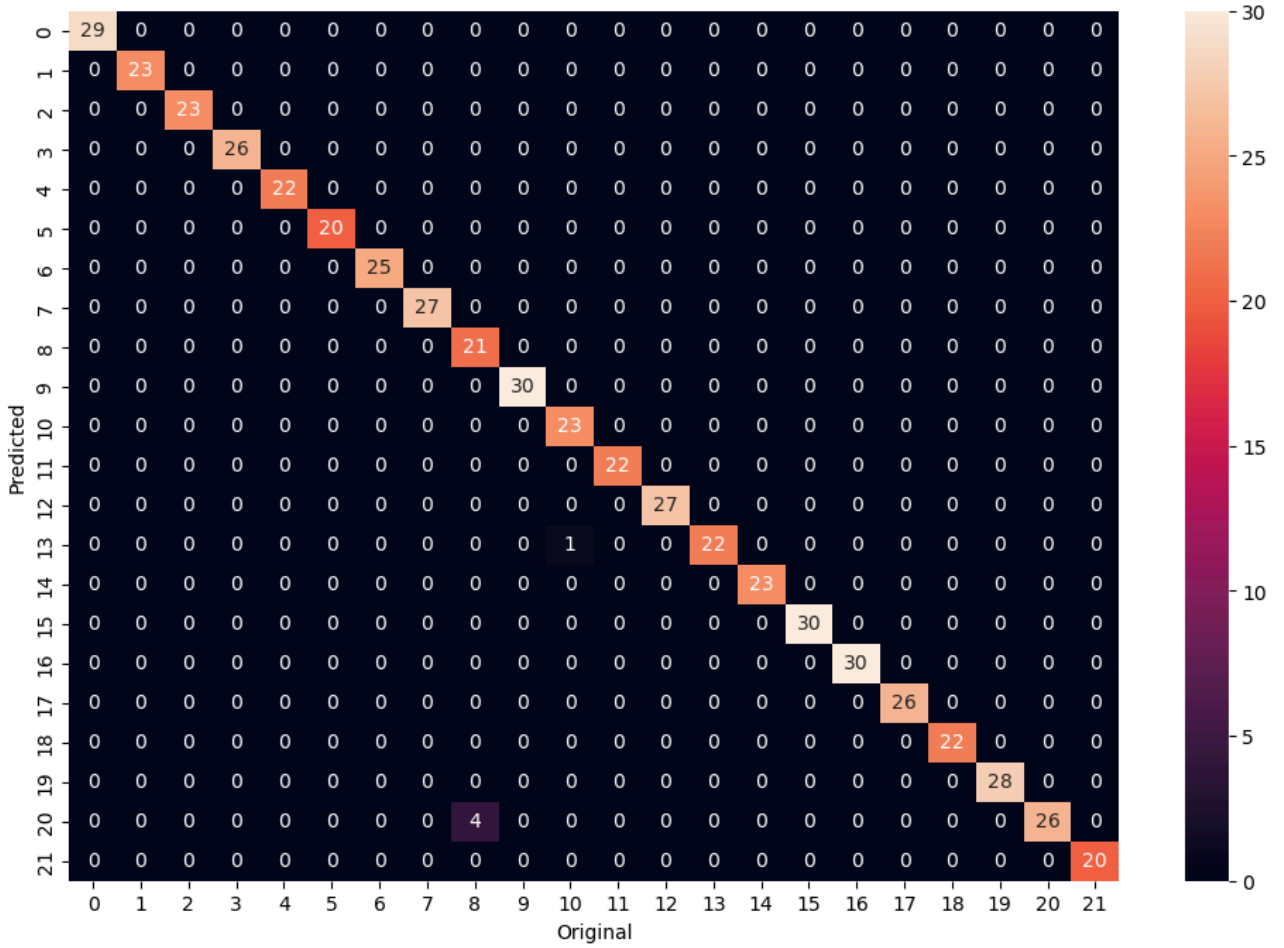
In [44]:
```
for i in best_param.keys():
    print(f'{i} : {best_param[i].score(x_test,y_test)}')
```

```
decision tree : 0.990909090909091
svm : 0.9818181818181818
random_forest : 0.990909090909091
k classifier : 0.9690909090909091
```

In [45]:
```
predicted = best_param['random_forest'].predict(x_test)
predicted
```

Out[45]:
```
array([20,  9,  3,  3, 12,  7, 17,  8, 18,  1,  8, 15,  4,  1, 11,  6, 12,
       19,  8,  5,  1,  8,  7,  2, 13, 20, 15,  6,  7,  5, 12, 21,  6, 21,
        8, 11, 21, 16, 15, 16, 16,  0, 12,  0,  6, 12,  0,  6, 11, 12, 20,
       21, 10,  4,  6, 13, 14, 13, 11,  6,  0, 20,  3,  1,  2, 10,  2,  6,
       17,  6,  8,  7,  9,  0, 11, 14, 16,  2,  5,  7, 20,  6,  2, 20,  7,
       13,  7, 20, 20,  3, 12,  6,  1, 11, 15, 13, 14, 20,  0,  7,  4, 15,
        9,  6, 10, 20, 10, 12, 20, 16, 15,  7, 13,  2, 18,  4,  7,  1,  9,
        9,  5, 12, 13, 16,  3, 16,  9, 17, 19, 10, 15, 21, 10,  0, 21, 17,
       11, 10,  0, 20, 18, 19, 14,  8, 14, 14, 13,  5,  4, 15,  4, 12,  9,
       18,  1, 15, 17, 19,  8,  5,  3, 16, 10,  8, 11,  6,  0, 14, 19, 16,
       10,  4,  5, 10, 14,  4, 15,  2,  9, 18,  4, 16, 19,  1, 18,  8, 14,
       17, 19,  8, 17, 20,  7,  8, 19, 16,  5, 19,  8,  3, 15, 14, 13, 10,
       17, 10,  2,  3, 18, 15, 12, 16, 14, 16, 11,  9, 17, 15, 14,  7,  5,
        0,  7, 12, 14,  4, 20, 11, 18,  0,  6,  9,  3,  1,  1, 12, 20,  8,
        2, 17, 12, 19,  2,  2, 21,  6,  6, 20, 16,  7, 19, 16,  8, 12, 21,
        0,  8,  7, 18,  7, 16, 14,  7,  4,  1, 15,  4, 15,  2, 21,  0, 13,
        3, 11, 10,  4, 15, 19, 16, 21,  2,  4, 16,  5, 21, 19, 20, 15, 16,
       15,  8, 18, 10, 13,  2,  4, 21,  9, 15,  0,  0, 18,  2,  0, 12, 17,
        8, 17,  3, 19, 17, 12,  4, 11,  5,  0, 18, 19,  6,  1, 20,  5,  7,
       21, 20,  3, 18, 12, 12,  3, 19, 18, 10,  0, 19, 19, 20,  9,  1,  3,
       12,  7,  9,  9,  6,  5,  7,  6, 11, 10,  2,  3,  6, 19,  6,  5,  9,
        3, 11, 19,  5, 18,  7, 13,  0,  2,  0,  3,  2, 17, 19,  3,  5,  9,
        4, 10,  1,  2,  7,  1,  2, 12, 21,  9,  3, 17,  9, 21, 17,  0, 17,
        5, 11,  3, 16, 13,  1, 20, 21, 20, 11,  7, 19, 17, 20, 13, 17, 14,
       14, 19,  0, 19, 14, 17, 13, 13,  0,  8,  1, 10, 15,  3,  3, 17, 17,
       15, 19, 11, 15,  9,  0,  9, 19, 10, 17,  7, 16, 17,  3,  6, 21, 15,
       10,  4,  2, 20,  9, 13, 13,  7,  9, 16,  8, 16,  6, 21,  9, 10, 13,
       16,  1, 12,  5, 12, 14,  8,  3,  8,  8, 14, 14,  9, 16,  0, 13,  7,
       18, 16,  0,  8, 12,  7,  8, 19,  3,  6, 20,  9,  1,  5, 18, 10,  0,
       11,  9, 15, 13, 11, 11,  5,  0, 15,  4, 12,  9, 15,  1, 14, 18,  1,
       11, 18,  4, 14, 18,  1,  4, 16, 15, 12,  2, 14,  3, 15, 12,  4,  9,
       21,  6,  9, 16, 17, 10, 13, 15, 16,  2, 15, 18, 21, 11, 16, 18, 10,
       20, 17,  1,  9, 19,  0])
```

In [46]:
```
plt.figure(figsize=(12,8))
sns.heatmap(confusion_matrix(y_test,predicted),annot=True)
plt.xlabel("Original")
plt.ylabel("Predicted")
plt.show()
```



In [47]:
```
pipe1 = make_pipeline(preprocessing.StandardScaler(),RandomForestClassifier(n_estimators
bag_model = BaggingClassifier(base_estimator=pipe1,n_estimators=100,
                              oob_score=True,random_state=0,max_samples=0.8)
```

In [48]:
```
bag_model.fit(x_train,y_train)
```

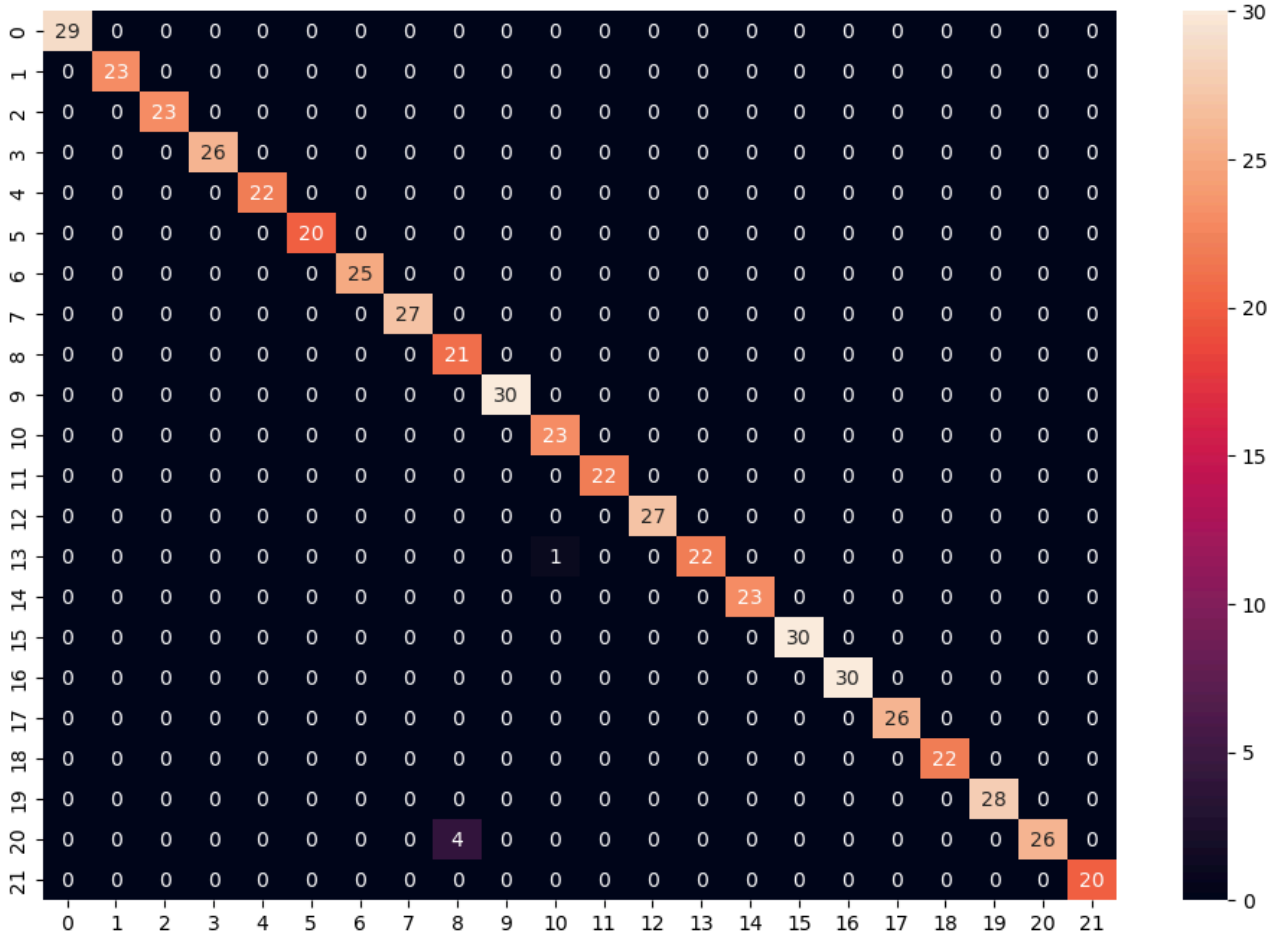
Out[48]:
```
BaggingClassifier(base_estimator=Pipeline(steps=[('standardscaler',
                                                  StandardScaler()),
                                                 ('randomforestclassifier',
                                                  RandomForestClassifier(n_estimators=
10))]),
                  max_samples=0.8, n_estimators=100, oob_score=True,
                  random_state=0)
```

In [49]:
```
bag_model.score(x_test,y_test)
```

Out[49]:
```
0.990909090909091
```

In [50]:
```
predict = bag_model.predict(x_test)
```

In [51]:
```
plt.figure(figsize=(12,8))
sns.heatmap(confusion_matrix(y_test,predict),annot=True)
plt.show()
```



In [52]:
```
dha2 =pd.DataFrame(Y)
code = pd.DataFrame(dha2['label'].unique())
```

In [53]:

```
dha = pd.DataFrame(y)
encode = pd.DataFrame(dha[0].unique())
refer = pd.DataFrame()
refer['code']=code
refer['encode']=encode
refer
```

Out[53]:

	code	encode
0	rice	20
1	maize	11
2	chickpea	3
3	kidneybeans	9
4	pigeonpeas	18
5	mothbeans	13
6	mungbean	14
7	blackgram	2
8	lentil	10
9	pomegranate	19
10	banana	1
11	mango	12
12	grapes	7
13	watermelon	21
14	muskmelon	15
15	apple	0
16	orange	16
17	papaya	17
18	coconut	4
19	cotton	6
20	jute	8
21	coffee	5

In [54]:
```
print(classification_report(y_test,predict))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        29
           1       1.00      1.00      1.00        23
           2       1.00      1.00      1.00        23
           3       1.00      1.00      1.00        26
           4       1.00      1.00      1.00        22
           5       1.00      1.00      1.00        20
           6       1.00      1.00      1.00        25
           7       1.00      1.00      1.00        27
           8       0.84      1.00      0.91        21
           9       1.00      1.00      1.00        30
          10       0.96      1.00      0.98        23
          11       1.00      1.00      1.00        22
          12       1.00      1.00      1.00        27
          13       1.00      0.96      0.98        23
          14       1.00      1.00      1.00        23
          15       1.00      1.00      1.00        30
          16       1.00      1.00      1.00        30
          17       1.00      1.00      1.00        26
          18       1.00      1.00      1.00        22
          19       1.00      1.00      1.00        28
          20       1.00      0.87      0.93        30
          21       1.00      1.00      1.00        20

    accuracy                           0.99       550
   macro avg       0.99      0.99      0.99       550
weighted avg       0.99      0.99      0.99       550
```