In [1]:
```
import numpy as np
import pandas as pd
```

In [2]:
```
crop=pd.read_csv('Crop_recommendation.csv')
crop.head()
```

Out[2]:

	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 [3]:
```
crop.shape
```

Out[3]:
```
(2200, 8)
```

In [4]:
```
crop.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 [5]:
```
crop.isnull().sum()
```

Out[5]:
```
N              0
P              0
K              0
temperature    0
humidity       0
ph             0
rainfall       0
label          0
dtype: int64
```

In [6]:
```
crop.duplicated().sum()
```

Out[6]:
```
np.int64(0)
```

In [7]:
```
crop.describe()
```

Out[7]:

	N	P	K	temperature	humidity	ph
count	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
std	36.917334	32.985883	50.647931	5.063749	22.263812	0.773938
min	0.000000	5.000000	5.000000	8.825675	14.258040	3.504752
25%	21.000000	28.000000	20.000000	22.769375	60.261953	5.971693
50%	37.000000	51.000000	32.000000	25.598693	80.473146	6.425045
75%	84.250000	68.000000	49.000000	28.561654	89.948771	6.923643
max	140.000000	145.000000	205.000000	43.675493	99.981876	9.935091

In [8]:
```
crop['label'].value_counts()
```

Out[8]:
```
label
rice           100
maize          100
chickpea       100
kidneybeans    100
pigeonpeas     100
mothbeans      100
mungbean       100
blackgram      100
lentil         100
pomegranate    100
banana         100
mango          100
grapes         100
watermelon     100
muskmelon      100
apple          100
orange         100
papaya         100
coconut        100
cotton         100
jute           100
coffee         100
Name: count, dtype: int64
```

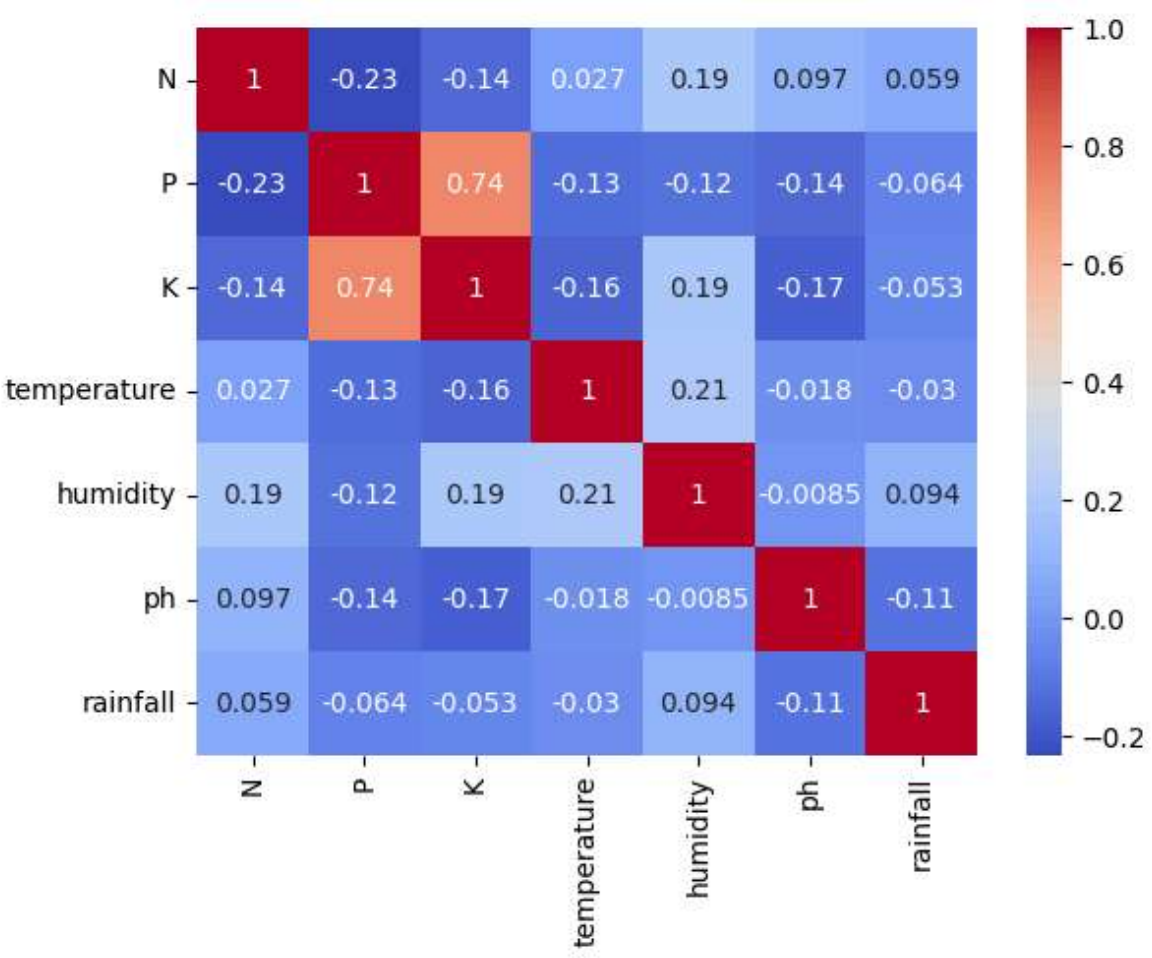
In [9]:
```
corr = crop.select_dtypes(include=np.number).columns
corr = crop[corr].corr()
corr
```

Out[9]:

	N	P	K	temperature	humidity	ph	rainf
N	1.000000	-0.231460	-0.140512	0.026504	0.190688	0.096683	0.059(
P	-0.231460	1.000000	0.736232	-0.127541	-0.118734	-0.138019	-0.063
K	-0.140512	0.736232	1.000000	-0.160387	0.190859	-0.169503	-0.053
temperature	0.026504	-0.127541	-0.160387	1.000000	0.205320	-0.017795	-0.030(
humidity	0.190688	-0.118734	0.190859	0.205320	1.000000	-0.008483	0.094
ph	0.096683	-0.138019	-0.169503	-0.017795	-0.008483	1.000000	-0.109(
rainfall	0.059020	-0.063839	-0.053461	-0.030084	0.094423	-0.109069	1.000(

In [10]:
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.heatmap(corr, annot=True, cmap='coolwarm')
```

Out[10]: <Axes: >



In [11]:
```
sns.distplot(crop['N'])
plt.show()
```

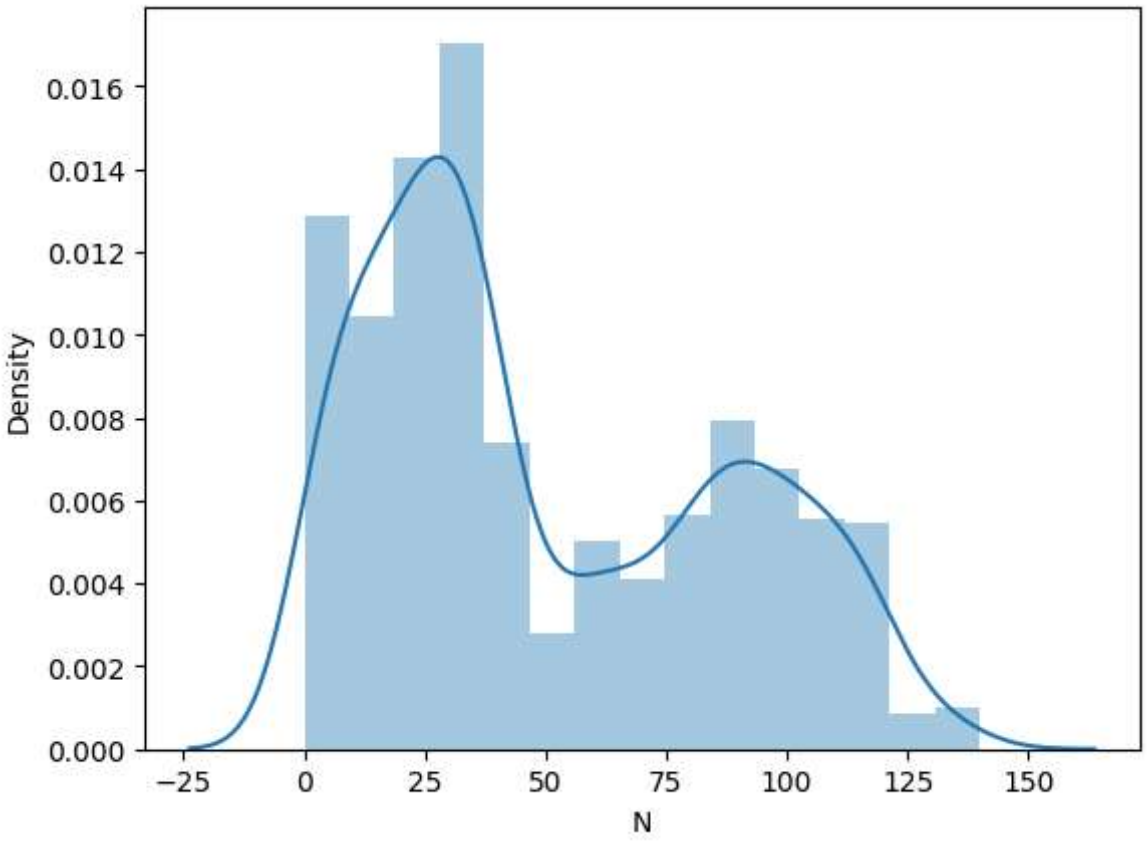
```
C:\Users\91789\AppData\Local\Temp\ipykernel_15516\3669751595.py:1: UserWarning:

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

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

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

  sns.distplot(crop['N'])
```



In [12]:
```
crop['label'].unique()
```

Out[12]:
```
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 [13]:
```
crop_dict={
    'rice':1,
    'maize':2,
    'chickpea':3,
    'kidneybeans':4,
    'pigeonpeas':5,
    'mothbeans':6,
    'mungbean':7,
    'blackgram':8,
    'lentil':9,
    'pomegranate':10,
    'banana':11,
    'mango':12,
    'grapes':13,
```

```
    'watermelon':14,
    'muskmelon':15,
    'apple':16,
    'orange':17,
    'papaya':18,
    'coconut':19,
    'cotton':20,
    'jute':21,
    'coffee':22
}
crop['crop_num']=crop['label'].map(crop_dict)
```

In [14]:
```
crop=crop.drop(['label'], axis=1)
crop.head()
```

Out[14]:

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

In [15]:
```
x=crop.drop('crop_num', axis=1)
y=crop['crop_num']
```

In [16]:
```
x.shape
```

Out[16]: (2200, 7)

In [17]:
```
y.shape
```

Out[17]: (2200,)

In [18]:
```
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_s
```

In [19]:
```
x_train.shape
```

Out[19]: (1760, 7)

In [20]:
```
x_test.shape
```

Out[20]: (440, 7)

In [21]:
```
from sklearn.preprocessing import MinMaxScaler
ms = MinMaxScaler()
x_train = ms.fit_transform(x_train)
x_test = ms.transform(x_test)
```

In [22]:
```
x_train
```

Out[22]:
```
array([[0.12142857, 0.07857143, 0.045     , ..., 0.9089898 , 0.48532225,
        0.29685161],
       [0.26428571, 0.52857143, 0.07      , ..., 0.64257946, 0.56594073,
        0.17630752],
       [0.05      , 0.48571429, 0.1       , ..., 0.57005802, 0.58835229,
        0.08931844],
       ...,
       [0.07857143, 0.22142857, 0.13      , ..., 0.43760347, 0.46198144,
        0.28719815],
       [0.07857143, 0.85      , 0.995     , ..., 0.76763665, 0.44420505,
        0.18346657],
       [0.22857143, 0.52142857, 0.085     , ..., 0.56099735, 0.54465022,
        0.11879596]], shape=(1760, 7))
```

In [23]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_train=sc.fit_transform(x_train)
x_test=sc.transform(x_test)
```

In [24]:
```
x_train
```

Out[24]:
```
array([[-9.03426596e-01, -1.12616170e+00, -6.68506601e-01, ...,
         9.36586183e-01,  1.93473784e-01,  5.14970176e-03],
       [-3.67051340e-01,  7.70358846e-01, -5.70589522e-01, ...,
        -1.00470485e-01,  8.63917548e-01, -6.05290566e-01],
       [-1.17161422e+00,  5.89737842e-01, -4.53089028e-01, ...,
        -3.82774991e-01,  1.05029771e+00, -1.04580687e+00],
       ...,
       [-1.06433917e+00, -5.24091685e-01, -3.35588533e-01, ...,
        -8.98381379e-01, -6.34357580e-04, -4.37358211e-02],
       [-1.06433917e+00,  2.12501638e+00,  3.05234239e+00, ...,
         3.86340190e-01, -1.48467347e-01, -5.69036842e-01],
       [-5.01145154e-01,  7.40255346e-01, -5.11839275e-01, ...,
        -4.18045489e-01,  6.86860180e-01, -8.96531475e-01]],
      shape=(1760, 7))
```

In [25]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
```

In [26]:
```
models={
    'LogisticRegression':LogisticRegression(),
    'DecisionTreeClassifier':DecisionTreeClassifier(),
    'RandomForestClassifier':RandomForestClassifier(),
    'SVC':SVC(),
    'KNeighborsClassifier':KNeighborsClassifier(),
    'GaussianNB':GaussianNB(),
    'GradientBoostingClassifier':GradientBoostingClassifier(),
    'AdaBoostClassifier':AdaBoostClassifier(),
    'BaggingClassifier':BaggingClassifier(),
```

```
    'ExtraTreesClassifier':ExtraTreesClassifier()
}

for name, model in models.items():
  model.fit(x_train, y_train)
  y_pred=model.predict(x_test)

  print(f"{name} with accuracy:{accuracy_score(y_test,y_pred)}")
```

```
LogisticRegression with accuracy:0.9636363636363636
DecisionTreeClassifier with accuracy:0.9886363636363636
RandomForestClassifier with accuracy:0.990909090909091
SVC with accuracy:0.9681818181818181
KNeighborsClassifier with accuracy:0.9659090909090909
GaussianNB with accuracy:0.9954545454545455
GradientBoostingClassifier with accuracy:0.9818181818181818
AdaBoostClassifier with accuracy:0.14545454545454545
BaggingClassifier with accuracy:0.990909090909091
ExtraTreesClassifier with accuracy:0.9863636363636363
```

In [27]:
```
rfc=RandomForestClassifier()
rfc.fit(x_train, y_train)
y_pred=rfc.predict(x_test)
print(f'RandomForestClassifier with accuracy:{accuracy_score(y_test,y_pred)}')

print(f"{name}  with accuracy : {accuracy_score(y_test,y_pred)}")
print("Confusion matrix : ",confusion_matrix(y_test,y_pred))
print("============================================================")
```

```
RandomForestClassifier with accuracy:0.9931818181818182
ExtraTreesClassifier  with accuracy : 0.9931818181818182
Confusion matrix :  [[17  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
2  0]
 [ 0 21  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0 26  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0 20  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0 23  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0 23  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0 19  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0 20  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0 11  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0 23  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0 21  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0 19  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0 14  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0 19  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0 17  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 23  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 14  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 23  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 27  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 17  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 23]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 17]]
============================================================
```

In [28]:
```
def recommendation(N,P,k,temperature,humidity,ph,rainfal):
    features = np.array([[N,P,k,temperature,humidity,ph,rainfal]])
    transformed_features = ms.fit_transform(features)
    prediction = rfc.predict(transformed_features)
```

```
    print(prediction)
    return prediction[0]
```

In [29]:
```
N = 40
P = 50
k = 50
temperature = 40.0
humidity = 20
ph = 100
rainfall = 100

predict = recommendation(N,P,k,temperature,humidity,ph,rainfall)

crop_dict = {1: "Rice", 2: "Maize", 3: "Jute", 4: "Cotton", 5: "Coconut", 6: "Pa
                8: "Apple", 9: "Muskmelon", 10: "Watermelon", 11: "Grapes", 12:
                14: "Pomegranate", 15: "Lentil", 16: "Blackgram", 17: "Mungbean
                19: "Pigeonpeas", 20: "Kidneybeans", 21: "Chickpea", 22: "Coffe

if predict in crop_dict:
    crop = crop_dict[predict]
    print("{} is a best crop to be cultivated ".format(crop))
else:
    print("Sorry are not able to recommend a proper crop for this environment")
```

```
[12]
Mango is a best crop to be cultivated
```

In [30]:
```
import pickle
pickle.dump(rfc, open('model.pkl','wb'))
pickle.dump(ms,open('minmaxScaler.pkl','wb'))
pickle.dump(sc,open('StandardScaler.pkl','wb'))
```