In [55]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

from sklearn.model_selection import train_test_split
```

In [16]:
```
crop = pd.read_csv(r"C:\Users\mayur\OneDrive\Desktop\kissan\Crop_recommendation
crop .head()
```

Out[16]:

	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 [17]:
```
crop.shape
```

Out[17]:
```
(2200, 8)
```

In [18]:
```
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 [19]:
```
crop.isnull().sum()
```

Out[19]:
```
N              0
P              0
K              0
temperature    0
humidity       0
ph             0
rainfall       0
label          0
dtype: int64
```

In [20]:
```
crop.duplicated().sum()
```

Out[20]:
```
0
```

In [21]:
```
crop.describe()
```

Out[21]:

	N	P	K	temperature	humidity	ph	rainfa
count	2200.000000	2200.000000	2200.000000	2200.000000	2200.000000	2200.000000	2200.00000
mean	50.551818	53.362727	48.149091	25.616244	71.481779	6.469480	103.46365
std	36.917334	32.985883	50.647931	5.063749	22.263812	0.773938	54.95838
min	0.000000	5.000000	5.000000	8.825675	14.258040	3.504752	20.21126
25%	21.000000	28.000000	20.000000	22.769375	60.261953	5.971693	64.55168
50%	37.000000	51.000000	32.000000	25.598693	80.473146	6.425045	94.86762
75%	84.250000	68.000000	49.000000	28.561654	89.948771	6.923643	124.26750
max	140.000000	145.000000	205.000000	43.675493	99.981876	9.935091	298.56011

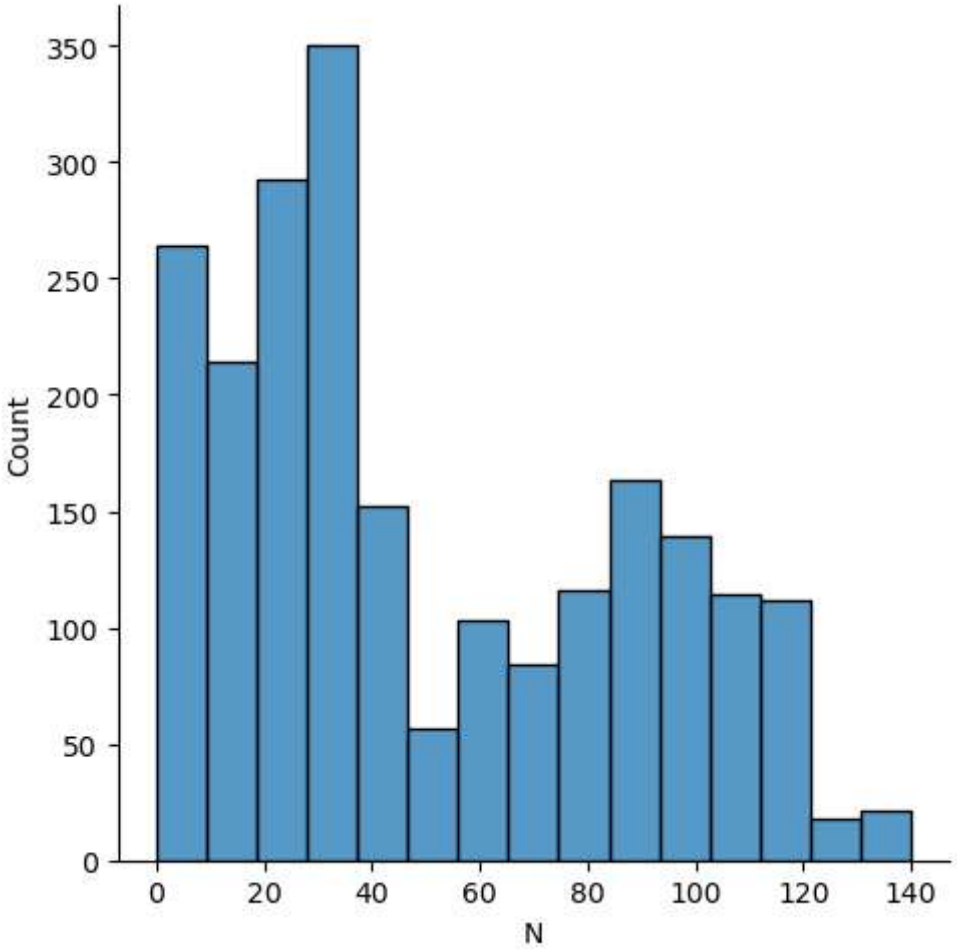
In [22]:
```
crop['label'].value_counts()
```

Out[22]:
```
label
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: count, dtype: int64
```

In [23]:
```
sns.displot(crop['N'])
```

```
C:\Users\mayur\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarni
ng: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

Out[23]:
```
<seaborn.axisgrid.FacetGrid at 0x19febab6b10>
```



In [24]:
```
crop_dict = {
    'rice': 1,
    'maize': 2,
    'jute': 3,
    'cotton': 4,
    'coconut': 5,
    'papaya': 6,
    'orange': 7,
    'apple': 8,
    'muskmelon': 9,
    'watermelon': 10,
    'grapes': 11,
    'mango': 12,
    'banana': 13,
    'pomegranate': 14,
    'lentil': 15,
    'blackgram': 16,
    'mungbean': 17,
    'mothbeans': 18,
    'pigeonpeas': 19,
    'kidneybeans': 20,
    'chickpea': 21,
    'coffee': 22
}

crop['crop_num']=crop['label'].map(crop_dict)
```

In [25]:
```
crop['crop_num'].value_counts()
```

Out[25]:
```
crop_num
1     100
2     100
3     100
4     100
5     100
6     100
7     100
8     100
9     100
10    100
11    100
12    100
13    100
14    100
15    100
16    100
17    100
18    100
19    100
20    100
21    100
22    100
Name: count, dtype: int64
```

In [33]:
```
crop.drop('label',axis=1,inplace=True)
crop.head()
```

Out[33]:

	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 [34]:
```
X =crop.drop('crop_num',axis=1)
y = crop['crop_num']
```

In [35]:
```
X.shape
```

Out[35]: (2200, 7)

In [36]:
```
y.shape
```

Out[36]: (2200,)

In [40]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state
```

In [42]:
```
X_train.shape
```

Out[42]: (1760, 7)

In [46]:
```
from sklearn.preprocessing import MinMaxScaler
ms=MinMaxScaler()

X_train = ms.fit_transform(X_train)
X_test = ms.transform(X_test)
```

In [47]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()


sc.fit(X_train)
X_train = sc.transform(X_train)
X_test = sc.transform(X_test)
```

In [48]:
```
X_train
```

Out[48]:
```
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]])
```

In [49]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import ExtraTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score
```

In [50]:
```
models = {
    'Logistic Regression': LogisticRegression(),
    'Naive Bayes': GaussianNB(),
    'Support Vector Machine': SVC(),
    'K-Nearest Neighbors': KNeighborsClassifier(),
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'Bagging': BaggingClassifier(),
    'AdaBoost': AdaBoostClassifier(),
    'Gradient Boosting': GradientBoostingClassifier(),
    'Extra Trees': ExtraTreeClassifier(),
}


for name, md in models.items():
    md.fit(X_train,y_train)
    ypred = md.predict(X_test)

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

```
Logistic Regression  with accuracy : 0.9636363636363636
Naive Bayes  with accuracy : 0.9954545454545455
Support Vector Machine  with accuracy : 0.9681818181818181
K-Nearest Neighbors  with accuracy : 0.9590909090909091
Decision Tree  with accuracy : 0.9840909090909091
Random Forest  with accuracy : 0.9931818181818182
Bagging  with accuracy : 0.9886363636363636
AdaBoost  with accuracy : 0.1409090909090909
Gradient Boosting  with accuracy : 0.9818181818181818
Extra Trees  with accuracy : 0.8727272727272727
```

In [51]:
```
rfc = RandomForestClassifier()
rfc.fit(X_train,y_train)
ypred = rfc.predict(X_test)
accuracy_score(y_test,ypred)
```

Out[51]:
```
0.9931818181818182
```

In [52]:
```
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)
    transformed_features = sc.fit_transform(transformed_features)
    prediction = rfc.predict(transformed_features).reshape(1,-1)

    return prediction[0]
```

In [56]:
```
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: "P
                8: "Apple", 9: "Muskmelon", 10: "Watermelon", 11: "Grapes", 12
                14: "Pomegranate", 15: "Lentil", 16: "Blackgram", 17: "Mungbea
                19: "Pigeonpeas", 20: "Kidneybeans", 21: "Chickpea", 22: "Coff

if predict[0] in crop_dict:
    crop = crop_dict[predict[0]]
    print("{} is a best crop to be cultivated ".format(crop))
else:
    print("Sorry are not able to recommend a proper crop for this environment")
```

```
Papaya is a best crop to be cultivated
```

In [ ]: