```
import numpy as np
import pandas as pd
```

In

Insert code cell below
Ctrl+M B

```
crop = pd.read_csv("Crop_recommendation.csv")
crop.head()
```

	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

Next steps: Generate code with crop View recommended plots

```
crop.shape #dimensions of dataset
```

```
(2200, 8)
```

```
crop.info() #summary of the DataFrame
```

```
<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
```

```
crop.isnull().sum() #check for any missing values   #.sum() This will count the number of True values (missing values)
```

```
N              0
P              0
K              0
temperature    0
humidity       0
ph             0
rainfall       0
label          0
dtype: int64
```

```
crop.duplicated().sum()
```

```
0
```

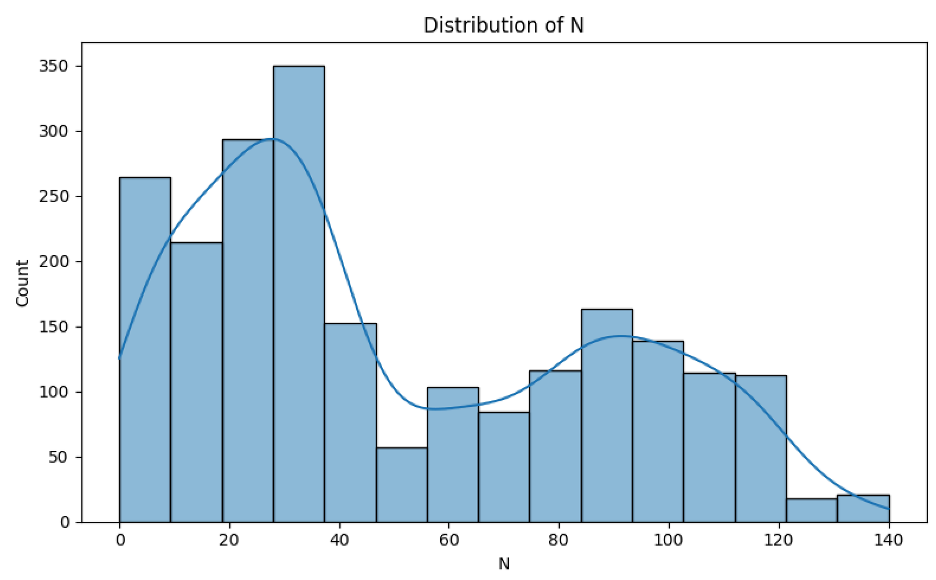
```
crop.describe() #summary statistics of the data
```

	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

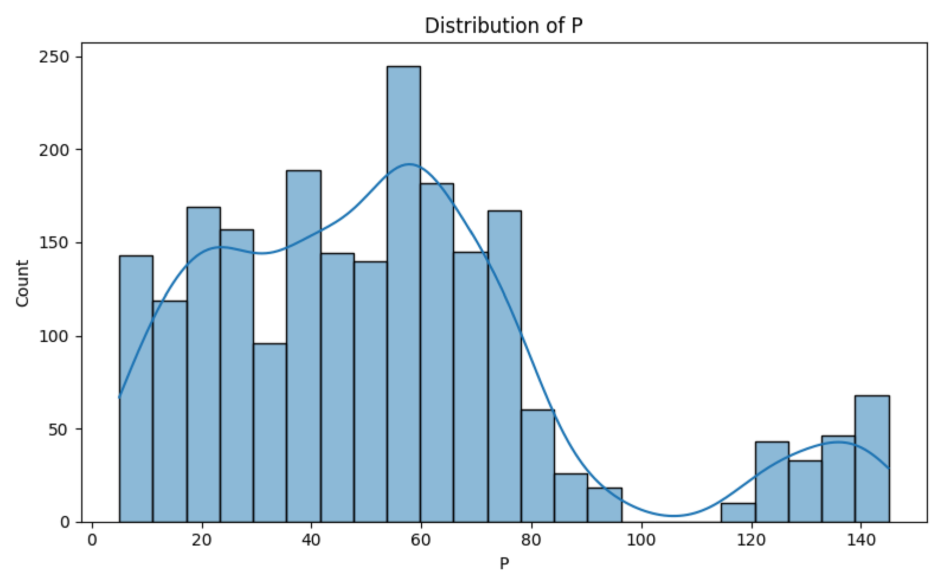
Data Visulazation

```
import seaborn as sns
import matplotlib.pyplot as plt
```

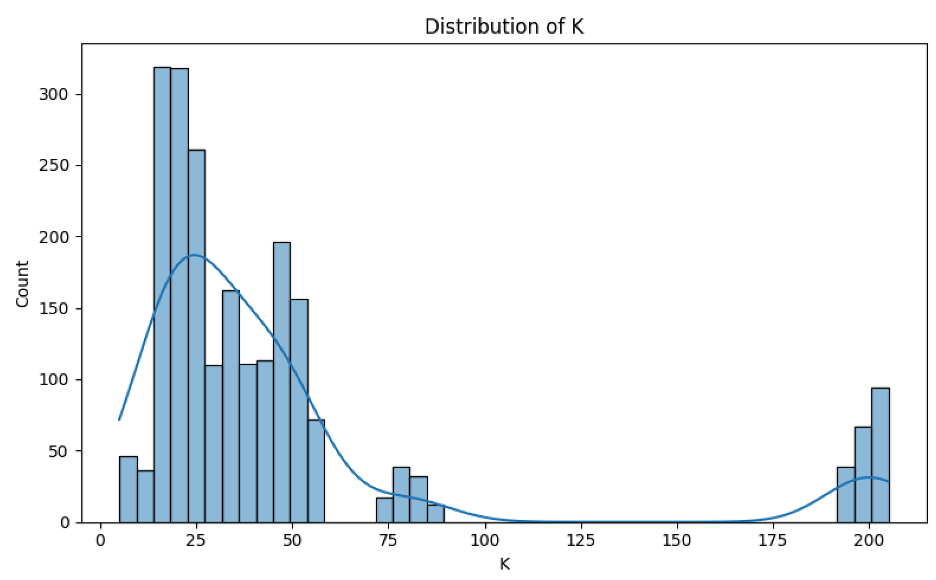
```
# Plotting histograms for feature distributions
plt.figure(figsize=(8, 5)) # Displays the frequency of data points
sns.histplot(crop['N'], kde=True)      #function from the Seaborn library that plots a histogram. The kde=True parameter adds a kernel density estimate line ove
plt.title('Distribution of N')
plt.tight_layout()
plt.show()
```



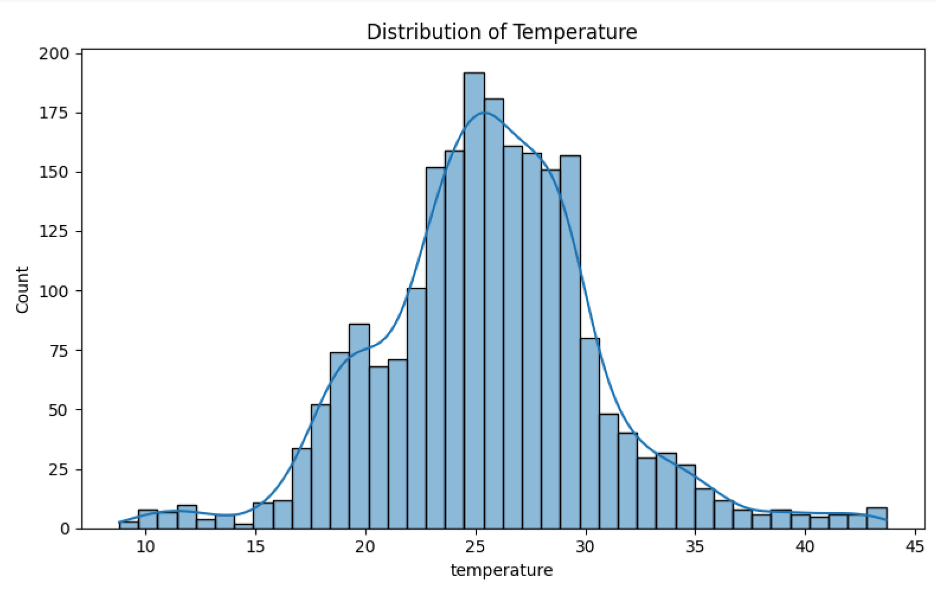

```
plt.figure(figsize=(8, 5))
sns.histplot(crop['P'], kde=True)
plt.title('Distribution of P')
plt.tight_layout()
plt.show()
```



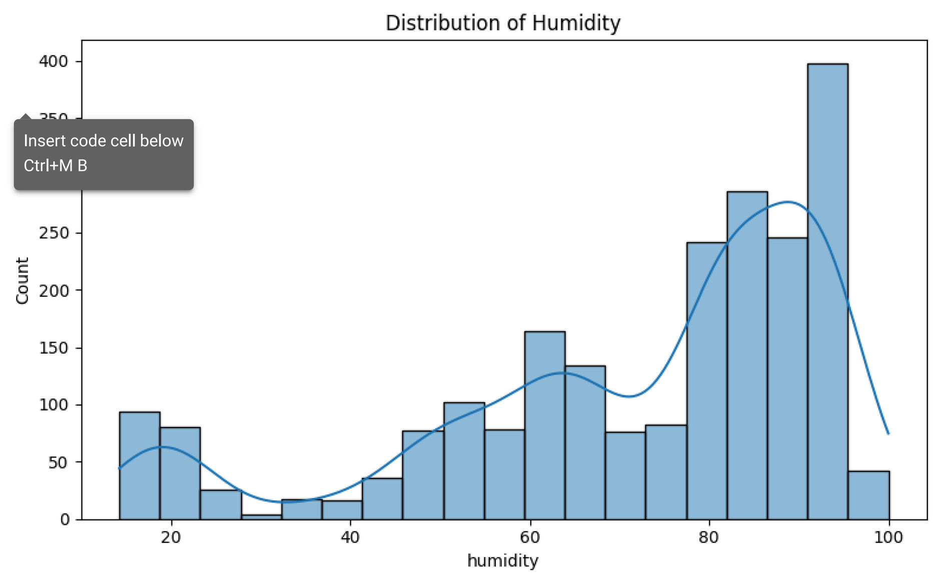

```
plt.figure(figsize=(8, 5))
sns.histplot(crop['K'], kde=True)
plt.title('Distribution of K')
plt.tight_layout()
plt.show()
```



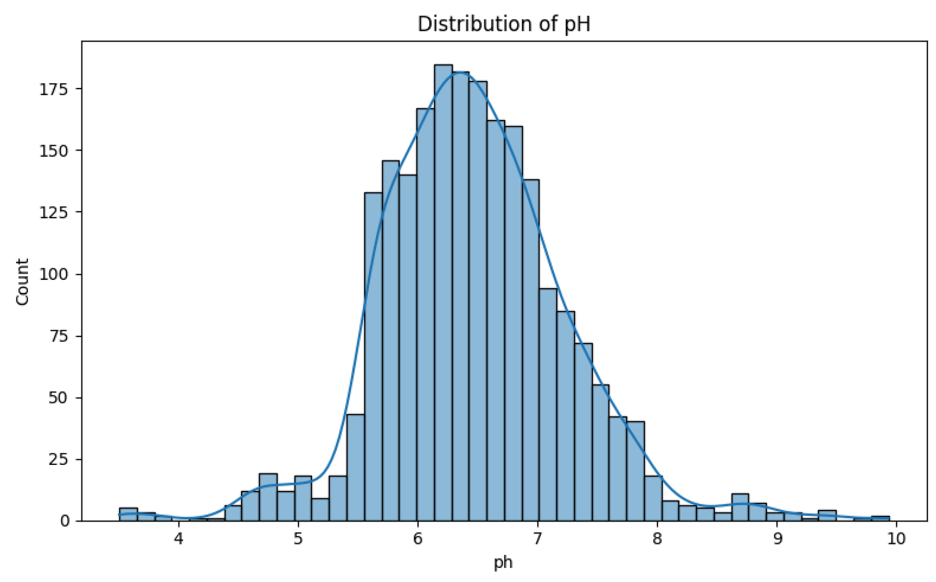

```
plt.figure(figsize=(8, 5))
sns.histplot(crop['temperature'], kde=True)
plt.title('Distribution of Temperature')
plt.tight_layout()
plt.show()
```



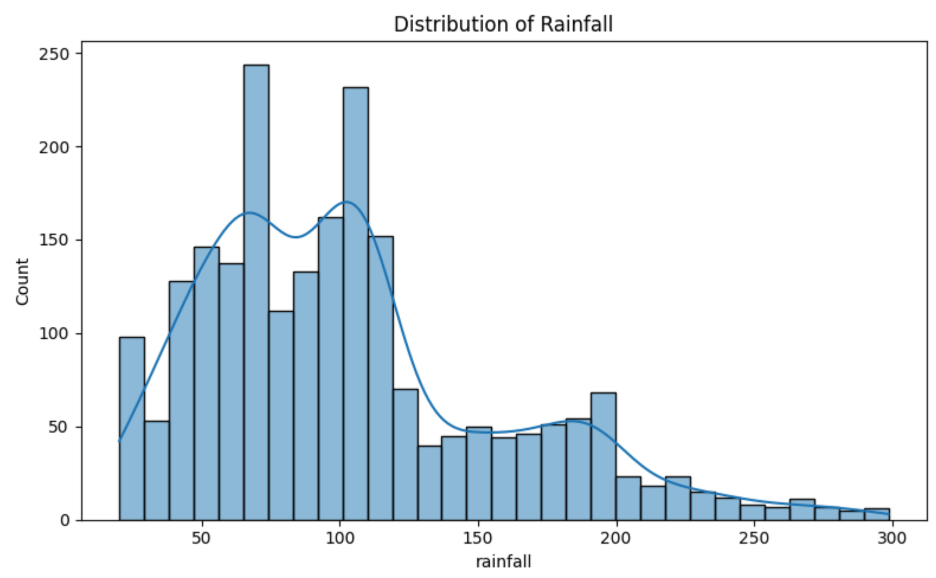

```
plt.figure(figsize=(8, 5))
sns.histplot(crop['humidity'], kde=True)
plt.title('Distribution of Humidity')
plt.tight_layout()
plt.show()
```



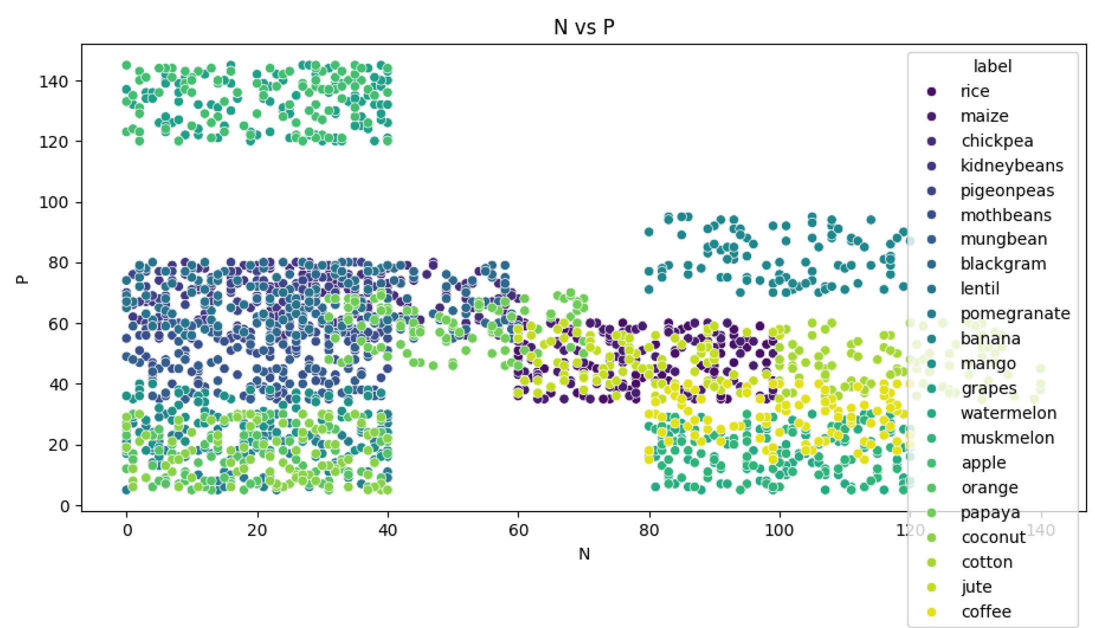

```
plt.figure(figsize=(8, 5))
sns.histplot(crop['ph'], kde=True)
plt.title('Distribution of pH')
plt.tight_layout()
plt.show()
```



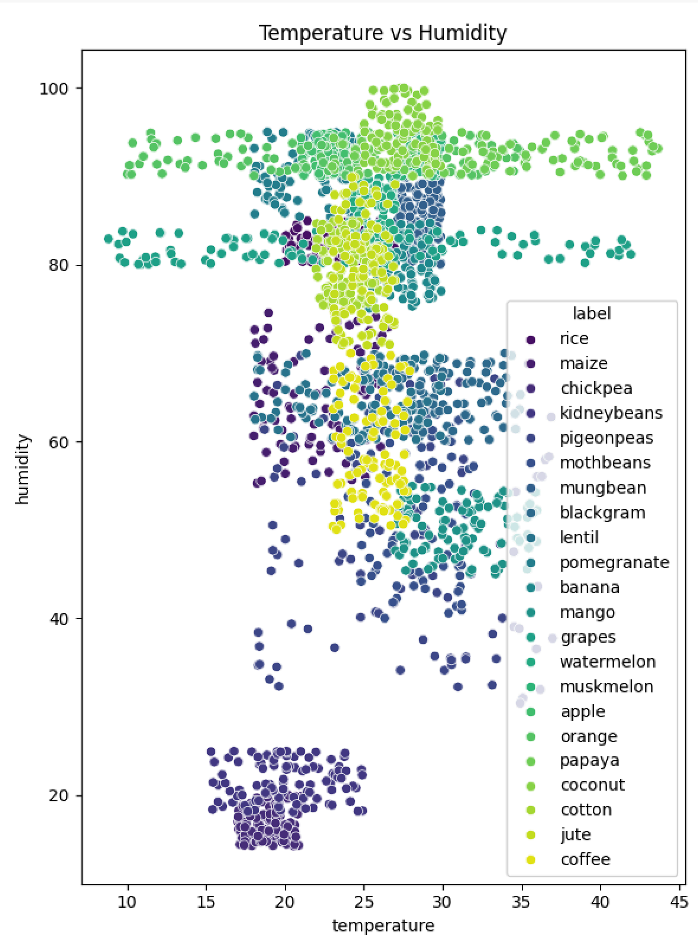

```
plt.figure(figsize=(8, 5))
sns.histplot(crop['rainfall'], kde=True)
plt.title('Distribution of Rainfall')
plt.tight_layout()
plt.show()
```



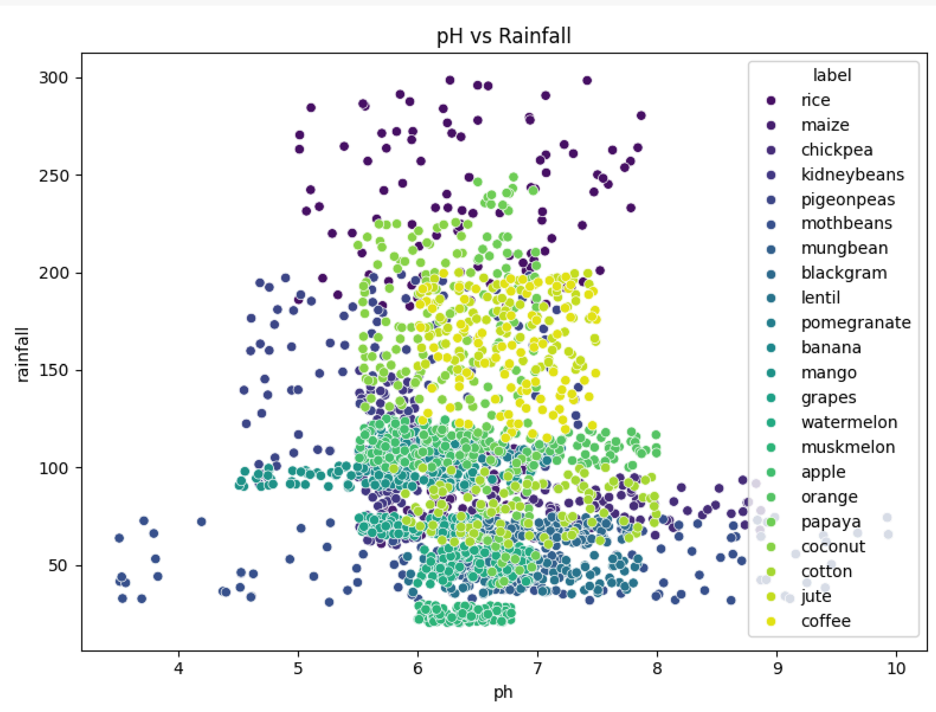

```
# Plotting scatter plots to show relationships between features
plt.figure(figsize=(18, 10))

plt.subplot(2, 2, 2)
sns.scatterplot(x='N', y='P', hue='label', data=crop, palette='viridis')
plt.title('N vs P')

plt.tight_layout()
plt.show()
```




```
# Plotting scatter plots to show relationships between features
plt.figure(figsize=(6, 8))

sns.scatterplot(x='temperature', y='humidity', hue='label', data=crop, palette='viridis')
plt.title('Temperature vs Humidity')

plt.tight_layout()
plt.show()
```




```
# Plotting scatter plots to show relationships between features
plt.figure(figsize=(8, 6))

sns.scatterplot(x='ph', y='rainfall', hue='label', data=crop, palette='viridis')
plt.title('pH vs Rainfall')

plt.tight_layout()
plt.show()
```



Exploring Data

```
# Select only numeric columns for correlation calculation
numeric_crop = crop.select_dtypes(include=['number'])

# Now calculate the correlation
numeri[illegible] Pearson correlation coefficient by default, which measures the linear relationship between pairs of variables
#The c[illegible]lps identify which variables are strongly related to each other, which can inform feature selection and engineering steps in the machi
```

Insert code cell below
Ctrl+M B

	N	P	K	temperature	humidity	ph	rainfall
N	1.000000	-0.231460	-0.140512	0.026504	0.190688	0.096683	0.059020
P	-0.231460	1.000000	0.736232	-0.127541	-0.118734	-0.138019	-0.063839
K	-0.140512	0.736232	1.000000	-0.160387	0.190859	-0.169503	-0.053461
temperature	0.026504	-0.127541	-0.160387	1.000000	0.205320	-0.017795	-0.030084
humidity	0.190688	-0.118734	0.190859	0.205320	1.000000	-0.008483	0.094423
ph	0.096683	-0.138019	-0.169503	-0.017795	-0.008483	1.000000	-0.109069
rainfall	0.059020	-0.063839	-0.053461	-0.030084	0.094423	-0.109069	1.000000

```
crop['label'].value_counts()
```

```
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
```

```
crop.label.unique()
```

```
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)
```

```
crop['label'].unique().size
```

```
22
```

```
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)
```

```
crop['crop_num'].value_counts()
```

```
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
```

```
crop.head(600)
```

	N	P	K	temperature	humidity	ph	rainfall	label	crop_num
0	90	42	43	20.879744	82.002744	6.502985	202.935536	rice	1
1	85	58	41	21.770462	80.319644	7.038096	226.655537	rice	1
2	60	55	44	23.004459	82.320763	7.840207	263.964248	rice	1
3	74	35	40	26.491096	80.158363	6.980401	242.864034	rice	1
4	78	42	42	20.130175	81.604873	7.628473	262.717340	rice	1
...	...	...	...	...	...	...	...	...	...
595	4	59	22	29.337434	49.003231	8.914075	42.440543	mothbeans	18
596	22	51	16	27.965837	61.349001	8.639586	70.104721	mothbeans	18
597	33	47	17	24.868040	48.275320	8.621514	63.918765	mothbeans	18
598	2	51	17	25.876823	45.963419	5.838509	38.532547	mothbeans	18
599	16	51	21	31.019636	49.976752	3.532009	32.812965	mothbeans	18

600 rows × 9 columns

Next steps: Generate code with crop | View recommended plots

Train Test Split

Prepare Features and Labels

```
#Feature Selection: Dropping unnecessary columns ('crop_num' and 'label') ensures that only relevant features are used for training, improving model performance
X = crop.drop(['crop_num','label'],axis=1) #independent variables
y = crop['crop_num'] #target variable for the machine learning model  #dependent
```

```
X
```

	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
...	...	...	...	...	...	...	...
2195	107	34	32	26.774637	66.413269	6.780064	177.774507
2196	99	15	27	27.417112	56.636362	6.086922	127.924610
2197	118	33	30	24.131797	67.225123	6.362608	173.322839
2198	117	32	34	26.272418	52.127394	6.758793	127.175293
2199	104	18	30	23.603016	60.396475	6.779833	140.937041

2200 rows × 7 columns

Next steps: Generate code with X | View recommended plots

```
from sklearn.model_selection import train_test_split
```

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42) #we get the same train and test sets across different executions
```

```
X_train.shape
```

```
(1760, 7)
```

```
X_test.shape
```

```
(440, 7)
```

```
X_train
```

	N	P	K	temperature	humidity	ph	rainfall
1656	17	16	14	16.396243	92.181519	6.625539	102.944161
752	37	79	19	27.543848	69.347863	7.143943	69.408782
892	7	73	25	27.521856	63.132153	7.288057	45.208411
1041	101	70	48	25.360592	75.031933	6.012697	116.553145
1179	0	17	30	35.474783	47.972305	6.279134	97.790725
...	...	...	...	...	...	...	...
1638	10	5	5	21.213070	91.353492	7.817846	112.983436
1095	108	94	47	27.359116	84.546250	6.387431	90.812505
1130	11	36	31	27.920633	51.779659	6.475449	100.258567
1294	11	124	204	13.429886	80.066340	6.361141	71.400430
860	32	78	22	23.970814	62.355576	7.007038	53.409060

1760 rows × 7 columns

Next steps: Generate code with X_train | View recommended plots

```
X_test
```

	N	P	K	temperature	humidity	ph	rainfall
1451	101	17	47	29.494014	94.729813	6.185053	26.308209
1334	98	8	51	26.179346	86.522581	6.259336	49.430510
[illegible]	[illegible]	[illegible]	[illegible]	43.360515	93.351916	6.941497	114.778071
[illegible]	[illegible]	[illegible]	[illegible]	34.280461	90.555616	6.825371	98.540477
1576	30	137	200	22.914300	90.704756	5.603413	118.604465
...	...	...	...	...	...	...	...
59	99	55	35	21.723831	80.238990	6.501698	277.962619
71	67	45	38	22.727910	82.170688	7.300411	260.887506
1908	121	47	16	23.605640	79.295731	7.723240	72.498009
1958	116	52	19	22.942767	75.371706	6.114526	67.080226
482	5	68	20	19.043805	33.106951	6.121667	155.370562

440 rows × 7 columns

Next steps: Generate code with X_test View recommended plots

Scale the features using MinMaxScaler

```
from sklearn.preprocessing import MinMaxScaler
ms = MinMaxScaler()

X_train = ms.fit_transform(X_train) # to scale your training data,
X_test = ms.transform(X_test) #This method both fits the scaler to your training data and transforms it.
```

```
X_train
```

```
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]])
```

Standarization

```
from sklearn.preprocessing import StandardScaler

# Scale the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
X_train
```

```
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]])
```

Training Models

LogisticRegression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Train the logistic regression model
logistic_model = LogisticRegression(max_iter=200) #initializes the logistic regression model with a maximum of 200 iterations
logistic_model.fit(X_train, y_train) # trains the model using the training data

# Predict the test set results
y_pred = logistic_model.predict(X_test) #uses the trained model to predict labels for the test data

# Calculate the accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Logistic Regression model accuracy: {accuracy}")
```

```
Logistic Regression model accuracy: 0.9636363636363636
```

KNN MODEL

```
from sklearn.neighbors import KNeighborsClassifier

# Train the KNN model
knn_model = KNeighborsClassifier(n_neighbors=5)  # initializes the KNN model 5 nearest neighbors
knn_model.fit(X_train, y_train)

# Predict the test set results
y_pred = knn_model.predict(X_test) #trains the KNN model using the training data

# Calculate the accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"KNN model accuracy: {accuracy}")
```

```
KNN model accuracy: 0.9590909090909091
```

Predictive System

```
# logistic_model
#Function to recommend crop based on input features
def recommend_crop(N, P, K, temperature, humidity, ph, rainfall):
    input_features = np.array([[N, P, K, temperature, humidity, ph, rainfall]])
    scaled_features = scaler.transform(input_features)
    prediction = logistic_model.predict(scaled_features)
    for crop, label in crop_dict.items():
        if label == prediction[0]:
            return crop

#User Input for Prediction
N = int(input("Enter N value: "))
P = int(input("Enter P value: "))
k = int(input("Enter k value: "))
temperature = int(input("Enter temperature value: "))
humidity = int(input("Enter humidity value: "))
ph = int(input("Enter ph value: "))
rainfall = int(input("Enter rainfall value: "))

recommended_crop = recommend_crop(N,P,k,temperature,humidity,ph,rainfall)
recommended_crop1 = recommend_crop(101, 17, 47, 29.494014,  94.729813,  6.185053, 26.308209)

print(f"Recommended crop: {recommended_crop}")
print(f"Recommended crop: {recommended_crop1}")
```

```
Enter N value: 30
Enter P value: 66
Enter k value: 58
Enter temperature value: 47
Enter humidity value: 89
Enter ph value: 47
Enter rainfall value: 994
Recommended crop: rice
Recommended crop: papaya
```

```
# KN
def recommend_crop_knn(N, P, K, temperature, humidity, ph, rainfall):
    input_features = np.array([[N, P, K, temperature, humidity, ph, rainfall]])
    scaled_features = scaler.transform(input_features)
    predicted_label = knn_model.predict(scaled_features)[0]

    # Reverse map the crop type to its name
    recommended_crop_name = [key for key, value in crop_dict.items() if value == predicted_label][0]
    return recommended_crop_name  #The predicted label (numeric) is mapped back to the crop name using crop_dict, which is a dictionary mapping crop names to th

# Example usage of the recommendation function
N = int(input("Enter N value: "))
P = int(input("Enter P value: "))
k = int(input("Enter k value: "))
temperature = int(input("Enter temperature value: "))
humidity = int(input("Enter humidity value: "))
ph = int(input("Enter ph value: "))
rainfall = int(input("Enter rainfall value: "))

recommended_crop = recommend_crop_knn(N,P,k,temperature,humidity,ph,rainfall)
print(f"Recommended crop: {recommended_crop}")
```

```
Enter N value: 100
Enter P value: 68
Enter k value: 44
Enter temperature value: 98
Enter humidity value: 68
Enter ph value: 229
Enter rainfall value: 58
Recommended crop: mothbeans
```