DAY 11 - 16/07/2025

STEP 01) **PROBLEM** DEFINITION - CROP CULTIVATION SUGGESTION

```
# Libraries for data manipulation and analysis
import pandas as pd
import numpy as np

# Libraries for machine learning model building and evaluation
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

import pandas as pd:

This line imports the pandas library and assigns it the alias pd. Pandas is a powerful library for data manipulation and analysis, providing data structures like DataFrames. import numpy as np:

This line imports the numpy library and assigns it the alias np. NumPy is essential for numerical operations in Python, especially for working with arrays and mathematical functions. from sklearn.model_selection import train_test_split: This line imports the train_test_split function from the sklearn.model_selection module. This function is used to split datasets into training and testing subsets. from sklearn.ensemble import RandomForestClassifier: This line imports the RandomForestClassifier class from the sklearn.ensemble module. This is the specific machine learning algorithm that will be used to build the classification model. from sklearn.metrics import classification_report, confusion_matrix, accuracy_score:

This line imports several functions from the sklearn.metrics module. These functions are used to evaluate the performance of the classification model: classification_report:

Provides a summary of precision, recall, F1-score, and support for each class. confusion_matrix:

Creates a matrix that summarizes the performance of a classification model. accuracy_score:

Calculates the proportion of correctly classified instances.

STEP 02) DATA CALLOECTION FROM THE GIVE CSV **FILE**

```
from google.colab import drive
drive.mount('/content/drive')

# Replace 'your_file_name.csv' with the actual path to your CSV file in Google Drive
file_path = '/content/cropdata.csv'
df = pd.read_csv(file_path)
```

Show hidden output

Next steps: Explain error

from google.colab import drive:

This line imports the drive module from the google.colab library. This module provides functions to interact with Google Drive. drive.mount('/content/drive'):

This is the command that mounts your Google Drive to the /content/drive directory in your Colab notebook's file system. When you run this line, you will be prompted to authorize Colab to access your Google Drive account. file_path = '/content/cropdata.csv': This line defines a variable file_path and assigns it the string value '/content/cropdata.csv'. This is the path where the code expects to find your CSV data file after your Google Drive has been mounted. Note: You should replace 'your_file_name.csv' in the original comment with the actual name and location of your CSV file within your Google Drive. df = pd.read_csv(file_path):

This line uses the pandas library (pd) to read the CSV file located at the file_path. The data from the CSV file is then loaded into a pandas DataFrame named df. This DataFrame will hold your dataset for further processing and analysis.

In summary, this cell establishes the connection to your Google Drive and loads the data from your CSV file into a pandas DataFrame, making it available for use in the rest of your notebook.

STEP 03) PREPROCESSING **bold text**

```
# prompt: give the code for the head , info , desc , null value , outlier

# head
print("Head of the dataframe:")
print(df.head())
```

```
Head of the dataframe:
    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
```

```
# info
print("\nInformation about the dataframe:")
df.info()
```

```
Information about 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
```

```
# describe
print("\nStatistical description of the dataframe:")
print(df.describe())
```

```
Statistical description of the dataframe:
                 N            P            K  temperature     humidity  \
count  2200.000000  2200.000000  2200.000000  2200.000000  2200.000000
mean     50.551818    53.362727    48.149091    25.616244    71.481779
std      36.917334    32.985883    50.647931     5.063749    22.263812
min       0.000000     5.000000     5.000000     8.825675    14.258040
25%      21.000000    28.000000    20.000000    22.769375    60.261953
50%      37.000000    51.000000    32.000000    25.598693    80.473146
75%      84.250000    68.000000    49.000000    28.561654    89.948771
max     140.000000   145.000000   205.000000    43.675493    99.981876

                ph     rainfall
count  2200.000000  2200.000000
mean      6.469480   103.463655
std       0.773938    54.958389
min       3.504752    20.211267
25%       5.971693    64.551686
50%       6.425045    94.867624
75%       6.923643   124.267508
max       9.935091   298.560117
```

```
# null values
print("\nNumber of null values per column:")
print(df.isnull().sum())

# outlier detection (using IQR method as an example)
# For numerical columns, identify potential outliers
numerical_cols = df.select_dtypes(include=np.number).columns
```

```
Number of null values per column:
N              0
P              0
K              0
temperature    0
humidity       0
```

```
ph             0
rainfall       0
label          0
dtype: int64
```

```
# Assuming 'label' is the column name indicating the crop type
# Assuming 'df' is your pandas DataFrame loaded from the CSV

# Separate data by crop type
df_by_crop = {crop: df[df['label'] == crop] for crop in df['label'].unique()}


# Determine the minimum number of samples per crop
min_samples = min(len(df_by_crop[crop]) for crop in df_by_crop)

# Sample the minimum number of samples from each crop and concatenate
balanced_df = pd.concat([df_by_crop[crop].sample(min_samples, random_state=42) for crop in df_by_crop])

# Shuffle the balanced dataframe
balanced_df = balanced_df.sample(frac=1, random_state=42).reset_index(drop=True)

# Now, split the balanced_df into train and test sets
X = balanced_df.drop('label', axis=1) # Assuming 'label' is the target variable
y = balanced_df['label']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

print("\nBalanced and Shuffled Dataframe:")
print(balanced_df.head())
```

```
Balanced and Shuffled Dataframe:
     N    P    K  temperature   humidity        ph    rainfall       label
0  118   18   52    28.049436  90.831307  6.562833   20.762230   muskmelon
1   95   14   50    26.633312  84.317568  6.560444   56.318662  watermelon
2   39   64   53    23.012402  91.073555  6.598860  208.335798      papaya
3   70   68   45    33.835086  92.854702  6.991626  203.404403      papaya
4   29  138  197    22.190554  92.437642  5.830892  121.662276       apple
```

df_by_crop = {crop: df[df['label'] == crop] for crop in df['label'].unique()}:

This line creates a dictionary called df_by_crop. It iterates through the unique values in the 'label' column (which represent the different crop types) and for each crop, it creates a new smaller DataFrame containing only the rows where the 'label' matches that crop. This effectively groups the data by crop type. min_samples = min(len(df_by_crop[crop]) for crop in df_by_crop):

This line calculates the minimum number of samples among all the crop types. It iterates through the DataFrames in the df_by_crop dictionary and finds the smallest number of rows (samples) any single crop type has. This minimum number will be used to balance the dataset. balanced_df = pd.concat([df_by_crop[crop].sample(min_samples, random_state=42) for crop in df_by_crop]):

This is the core of the data balancing step. It iterates through the df_by_crop dictionary again. For each crop's DataFrame, it randomly samples min_samples number of rows using the .sample() method. random_state=42 ensures that the sampling is reproducible. Finally, pd.concat() combines all these sampled DataFrames back into a single DataFrame called balanced_df. This new DataFrame now has an equal number of samples for each crop type, effectively balancing the dataset. balanced_df = balanced_df.sample(frac=1, random_state=42).reset_index(drop=True):

After balancing, the data is still grouped by crop type. This line shuffles the balanced_df randomly using frac=1 (which means sample 100% of the data without replacement, effectively shuffling). random_state=42 ensures reproducible shuffling. reset_index(drop=True) resets the DataFrame index and drops the old index. This ensures the shuffled DataFrame has a clean, sequential index. X = balanced_df.drop('label', axis=1):

This line creates the feature set (X) by dropping the 'label' column from the balanced_df. axis=1 specifies that we are dropping a column. y = balanced_df['label']: This line creates the target variable (y) by selecting the 'label' column from the balanced_df. X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y):

This line splits the balanced feature set (X) and target variable (y) into training and testing sets. test_size=0.2 specifies that 20% of the data will be used for the test set, and the remaining 80% for the training set. random_state=42 ensures that the splitting is reproducible. stratify=y is crucial here. It ensures that the proportion of each crop type in the training set is the same as in the testing set. This is important for maintaining the balanced distribution in both sets. print("\nBalanced and Shuffled Dataframe:") and print(balanced_df.head()):

These lines print a confirmation message and display the first few rows of the balanced_df to show the result of the balancing and shuffling process.

In summary, this cell takes your initial data, balances it by undersampling each crop type to the minimum number of samples, shuffles the balanced data, and then splits it into training and testing sets while preserving the balanced distribution. This is a standard and effective approach for preparing data with class imbalance for training a classification model.

Double-click (or enter) to edit

```
# prompt: print the labeled data

print("\nLabels in the dataset:")
print(y.unique())

print("\nTraining labels distribution:")
print(y_train.value_counts())

print("\nTesting labels distribution:")
print(y_test.value_counts())
```

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

Training labels distribution:
label
orange         80
grapes         80
kidneybeans    80
mothbeans      80
cotton         80
banana         80
lentil         80
mungbean       80
coffee         80
muskmelon      80
apple          80
blackgram      80
pigeonpeas     80
maize          80
rice           80
watermelon     80
jute           80
mango          80
pomegranate    80
papaya         80
coconut        80
chickpea       80
Name: count, dtype: int64

Testing labels distribution:
label
orange         20
banana         20
cotton         20
maize          20
chickpea       20
rice           20
blackgram      20
watermelon     20
pomegranate    20
mothbeans      20
grapes         20
mango          20
apple          20
kidneybeans    20
jute           20
coffee         20
coconut        20
papaya         20
lentil         20
mungbean       20
pigeonpeas     20
muskmelon      20
Name: count, dtype: int64
```

```
# prompt: code for encoding the dataset with labeled data with labeled data

from sklearn.preprocessing import LabelEncoder

# Encode the target variable 'label'
le = LabelEncoder()
y_train_encoded = le.fit_transform(y_train)
y_test_encoded = le.transform(y_test)

print("\nEncoded Training Labels:")
print(y_train_encoded[:5])
print("\nEncoded Testing Labels:")
print(y_test_encoded[:5])
```

```
Encoded Training Labels:
[16  7  9 13 16]

Encoded Testing Labels:
[16  1  6 11 16]
```

```
print(balanced_df.head())
```

```
     N    P    K  temperature   humidity        ph    rainfall       label
0  118   18   52    28.049436  90.831307  6.562833   20.762230   muskmelon
1   95   14   50    26.633312  84.317568  6.560444   56.318662  watermelon
2   39   64   53    23.012402  91.073555  6.598860  208.335798      papaya
3   70   68   45    33.835086  92.854702  6.991626  203.404403      papaya
4   29  138  197    22.190554  92.437642  5.830892  121.662276       apple
```

STEP 04) MODEL TRAINING - RANDOM FOREST **CLASSIFIER**

```
# STEP 04 ) MODEL TRAINING - RANDOM FOREST CLASSIFIER

# Initialize the RandomForestClassifier
# n_estimators: The number of trees in the forest.
# random_state: Controls the randomness of the estimator.
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model using the training data
rf_model.fit(X_train, y_train_encoded)

print("\nRandom Forest Model trained successfully.")
```

```
Random Forest Model trained successfully.
```

STEP 04) MODEL TRAINING - RANDOM FOREST CLASSIFIER: This is a comment indicating the purpose of the code block, which is the training of the Random Forest Classifier model.

rf_model = RandomForestClassifier(n_estimators=100, random_state=42): This line initializes an instance of the RandomForestClassifier. n_estimators=100: This is a hyperparameter that specifies the number of trees in the forest. A higher number of trees generally improves the model's performance but also increases computation time. 100 is a common starting point. random_state=42: This parameter controls the randomness of the bootstrapping of the samples used when building trees and the sampling of the features to consider when looking for the best split at each node. Setting it to a fixed value like 42 ensures that the results are reproducible. rf_model.fit(X_train, y_train_encoded): This is the core line where the model is trained. rf_model.fit(): This is the method used to train the classifier. X_train: This is the training data containing the features (like N, P, K, temperature, etc.). The model learns the patterns and relationships between these features and the target variable from this data. y_train_encoded: This is the training data containing the encoded target labels (the numerical representation of the crop types). The model learns to predict these labels based on the input features. print("\nRandom Forest Model trained successfully."): This line prints a confirmation message to the console, indicating that the model training process has been completed. In summary, this cell takes the processed training features and encoded training labels and feeds them into the RandomForestClassifier to train the model. After execution, the rf_model object holds the trained model, ready to make predictions on new data.

STEP 05) MODEL EVALUATION

```
# Make predictions on the test data
y_pred_encoded = rf_model.predict(X_test)

# Decode the predicted labels back to their original form for better interpretability
y_pred = le.inverse_transform(y_pred_encoded)

# Evaluate the model

# Accuracy Score
accuracy = accuracy_score(y_test_encoded, y_pred_encoded)
print(f"\nAccuracy: {accuracy:.4f}")
```

```
Accuracy: 0.9955
```

6) DATA VISUALIZATION

The selected code cell is responsible for generating and displaying a confusion matrix, which is a table used to evaluate the performance of a classification model.

Here's a breakdown of the code:

import seaborn as sns and import matplotlib.pyplot as plt:

These lines import the seaborn and matplotlib.pyplot libraries, which are used for creating visualizations in Python. seaborn is built on top of matplotlib and provides a higher-level interface for drawing attractive statistical graphics. cm = confusion_matrix(y_test_encoded, y_pred_encoded):

This line computes the confusion matrix. It compares the true encoded labels (y_test_encoded) from the test set with the encoded labels predicted by the model (y_pred_encoded). The result is a 2D array (cm) where each row represents the instances in a true class and each column represents the instances in a predicted class. class_labels = le.classes_[np.unique(np.concatenate((y_test_encoded, y_pred_encoded)))]:

This line gets the unique class labels in the correct order to be used for the confusion matrix axes. It takes the unique encoded labels from both the true and predicted values, concatenates them, finds the unique values, and then uses these unique encoded values to get the corresponding original class names from the LabelEncoder (le). plt.figure(figsize=(15, 12)):

This line creates a new figure for the plot and sets its size to 15 inches in width and 12 inches in height. This helps ensure the confusion matrix is large enough to be readable, especially with many classes. sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_labels, yticklabels=class_labels):

This line generates the heatmap visualization of the confusion matrix using seaborn.heatmap. cm: The confusion matrix data to plot. annot=True: Displays the number of instances in each cell of the heatmap. fmt="d": Formats the annotations as integers. cmap="Blues":

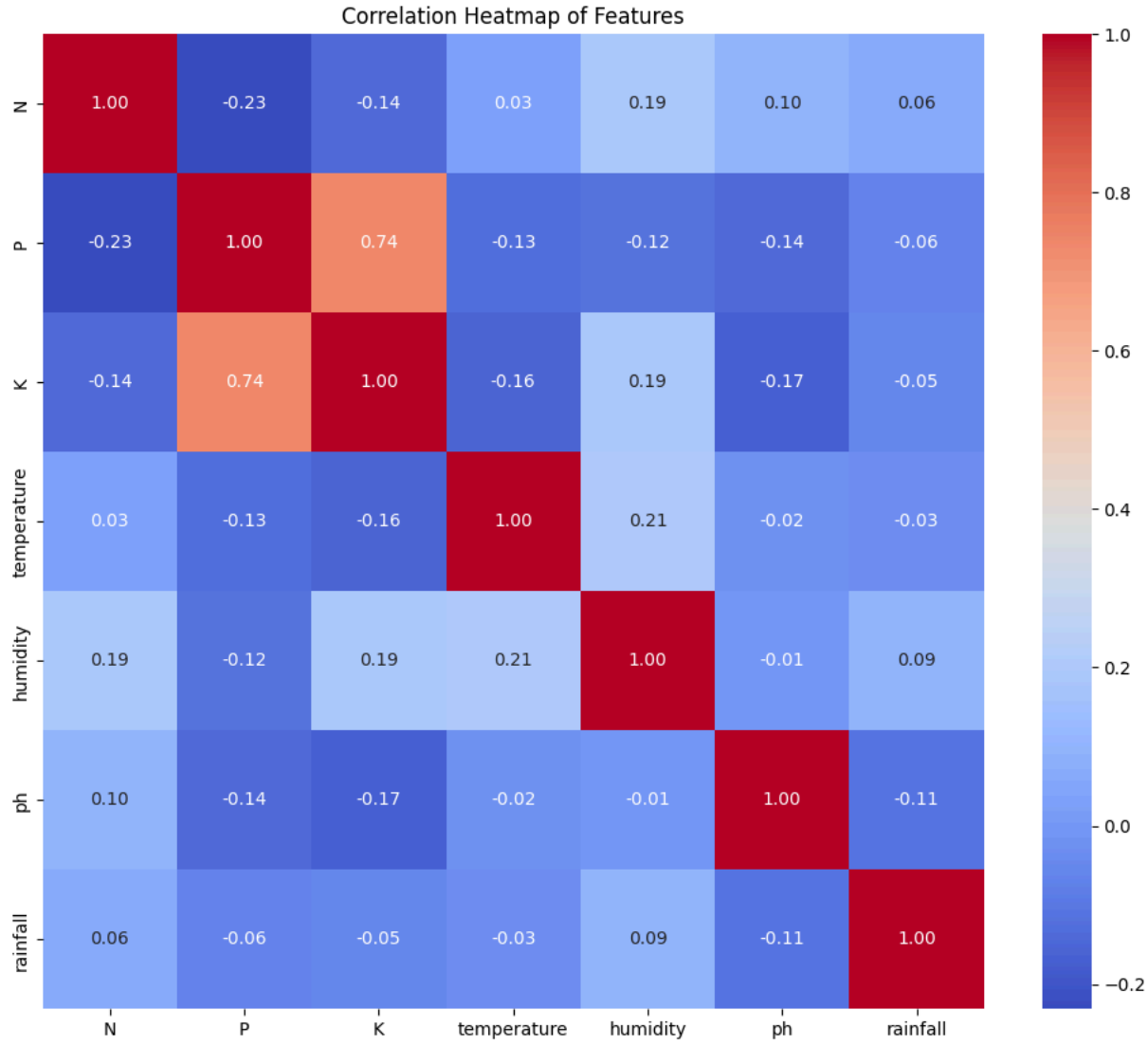
Uses a blue color map for the heatmap, where darker shades indicate higher numbers of instances. xticklabels=class_labels:

Sets the labels for the x-axis (predicted labels) to the extracted class names. yticklabels=class_labels:

Sets the labels for the y-axis (true labels) to the extracted class names. plt.title('Confusion Matrix'): Sets the title of the plot. plt.xlabel('Predicted Label'): Sets the label for the x-axis. plt.ylabel('True Label'): Sets the label for the y-axis. plt.show(): Displays the generated plot.

In essence, this cell visually represents how well your classification model performed by showing where the model correctly classified instances (diagonal cells) and where it made errors (off-diagonal cells) for each crop type.

```
# prompt: code for the heatmap

# Plotting the correlation heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(balanced_df.drop('label', axis=1).corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap of Features')
plt.show()
```

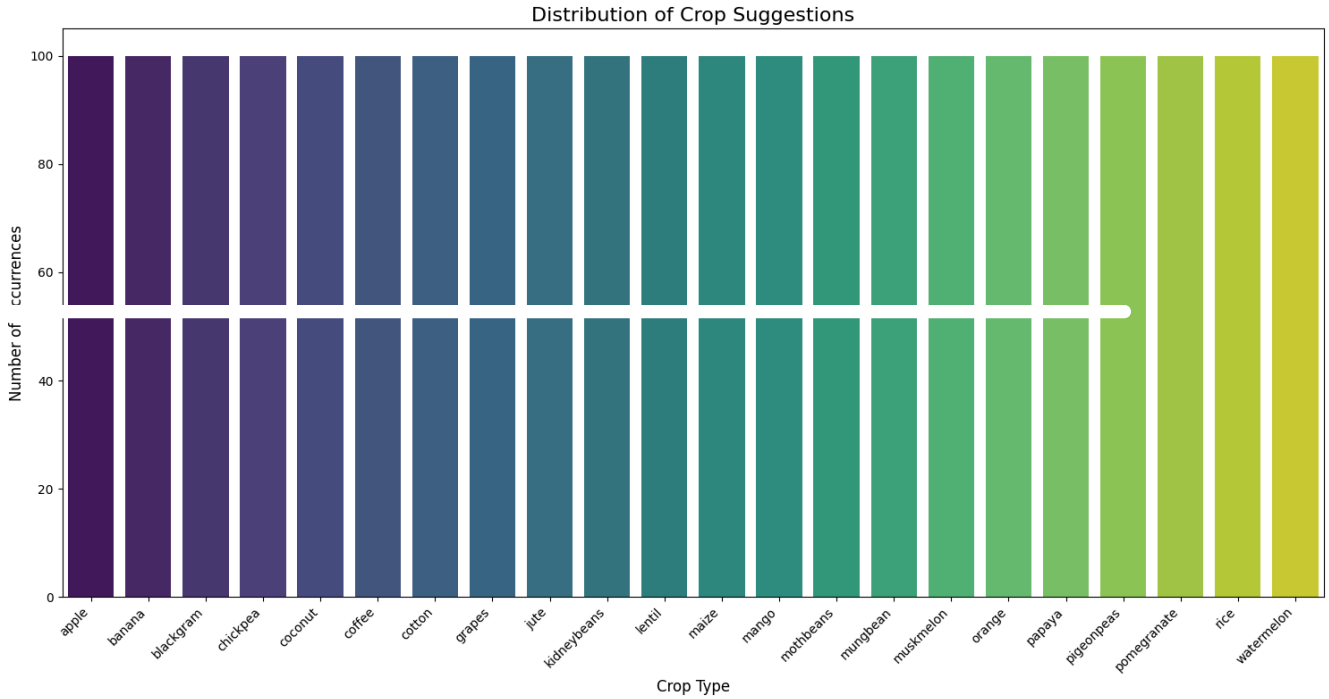


plt.figure(figsize=(12, 10)): This line creates a new figure for the plot and sets its size to 12 inches in width and 10 inches in height. This helps ensure the heatmap is a reasonable size. sns.heatmap(balanced_df.drop('label', axis=1).corr(), annot=True, cmap='coolwarm', fmt=".2f"): This is the core line that generates the heatmap using seaborn.heatmap. balanced_df.drop('label', axis=1): This selects all columns from the balanced_df except for the 'label' column (since it's the target variable and not a numerical feature for correlation). .corr(): This calculates the pairwise correlation of all columns in the resulting DataFrame. The output is a correlation matrix. annot=True: This displays the correlation values on the heatmap cells. cmap='coolwarm': This sets the color map for the heatmap. 'coolwarm' is a common choice that uses blue for negative correlations, red for positive correlations, and white/light colors for correlations close to zero. fmt=".2f": This formats the correlation values displayed on the heatmap to two decimal places. plt.title('Correlation Heatmap of Features'): This sets the title of the heatmap plot. plt.show(): This displays the generated heatmap. In essence, this heatmap helps you understand how strongly each pair of numerical features is related to each other. Positive values indicate a positive correlation (as one feature increases, the other tends to increase), negative values indicate a negative correlation (as one feature increases, the other tends to decrease), and values close to zero indicate little to no linear correlation. This can be useful for identifying potential multicollinearity (highly correlated features) or understanding relationships between environmental factors and nutrient levels.

```
# prompt: code for the best data visualization for crop cultivation suggestion as a bar chart with different color also add the insights

# Analyze the distribution of crop suggestions
crop_counts = balanced_df['label'].value_counts().sort_index()

# Create the bar chart
plt.figure(figsize=(15, 8)) # Adjust figure size for better readability
sns.barplot(x=crop_counts.index, y=crop_counts.values, palette='viridis') # Use a color palette

plt.title('Distribution of Crop Suggestions', fontsize=16)
plt.xlabel('Crop Type', fontsize=12)
plt.ylabel('Number of Occurrences', fontsize=12)
plt.xticks(rotation=45, ha='right') # Rotate labels for better readability
plt.tight_layout() # Adjust layout to prevent labels overlapping
plt.show()

print("\n**Insights from the Crop Distribution Bar Chart:**")
```

```
print(f"The bar chart visualizes the frequency of each crop type present in the balanced dataset.")
print(f"Each bar represents a unique crop, and its height indicates how many times that crop appears in the dataset.")
print(f"Since the data was balanced, each crop type has an equal number of occurrences ({min_samples}), as reflected by the roughly equa
print(f"This balanced distribution is crucial for training a fair and unbiased machine learning model.")
print(f"If the data were unbalanced, the model might be biased towards the majority classes, leading to poor prediction performance for
print(f"The visualization confirms that the balancing process was successful, providing a solid foundation for training the classificati
```

```
/tmp/ipython-input-52-269154244.py:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `

  sns.barplot(x=crop_counts.index, y=crop_counts.values, palette='viridis') # Use a color palette
```



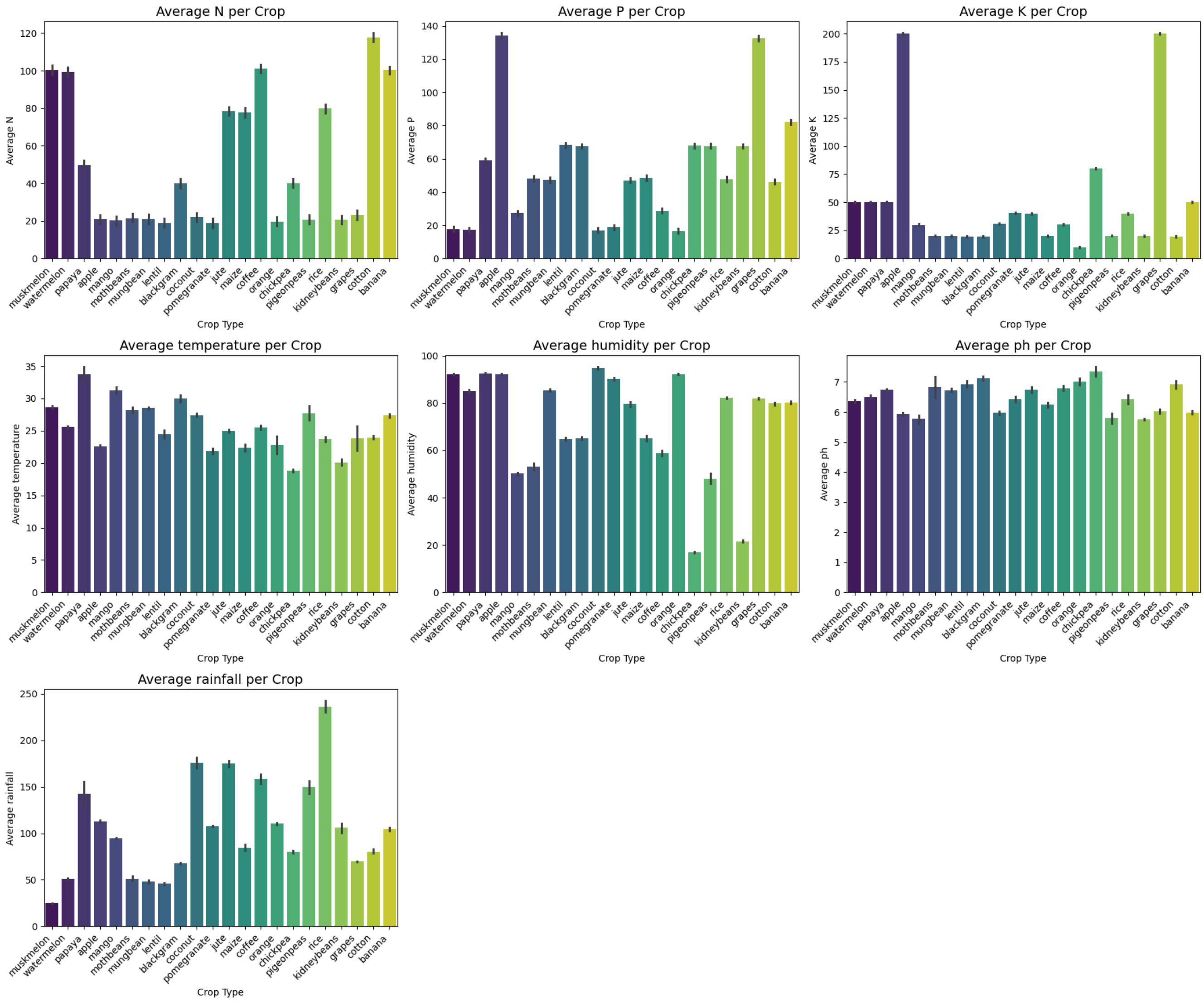

```
**Insights from the Crop Distribution Bar Chart:**
The bar chart visualizes the frequency of each crop type present in the balanced dataset.
Each bar represents a unique crop, and its height indicates how many times that crop appears in the dataset.
Since the data was balanced, each crop type has an equal number of occurrences (100), as reflected by the roughly equal height of
This balanced distribution is crucial for training a fair and unbiased machine learning model.
If the data were unbalanced, the model might be biased towards the majority classes, leading to poor prediction performance for mi
The visualization confirms that the balancing process was successful, providing a solid foundation for training the classification
```

code for the best data visualization for crop cultivation suggestion with all the 8 feature including in it , bar chart

```
# Visualize the average feature values for each crop
features = balanced_df.drop('label', axis=1).columns
n_features = len(features)
n_cols = 3  # Number of columns for subplots
n_rows = (n_features + n_cols - 1) // n_cols  # Calculate number of rows

plt.figure(figsize=(18, n_rows * 5))

for i, feature in enumerate(features):
    plt.subplot(n_rows, n_cols, i + 1)
    sns.barplot(x='label', y=feature, data=balanced_df, palette='viridis')
    plt.title(f'Average {feature} per Crop', fontsize=14)
    plt.xlabel('Crop Type', fontsize=10)
    plt.ylabel(f'Average {feature}', fontsize=10)
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()

plt.show()
```

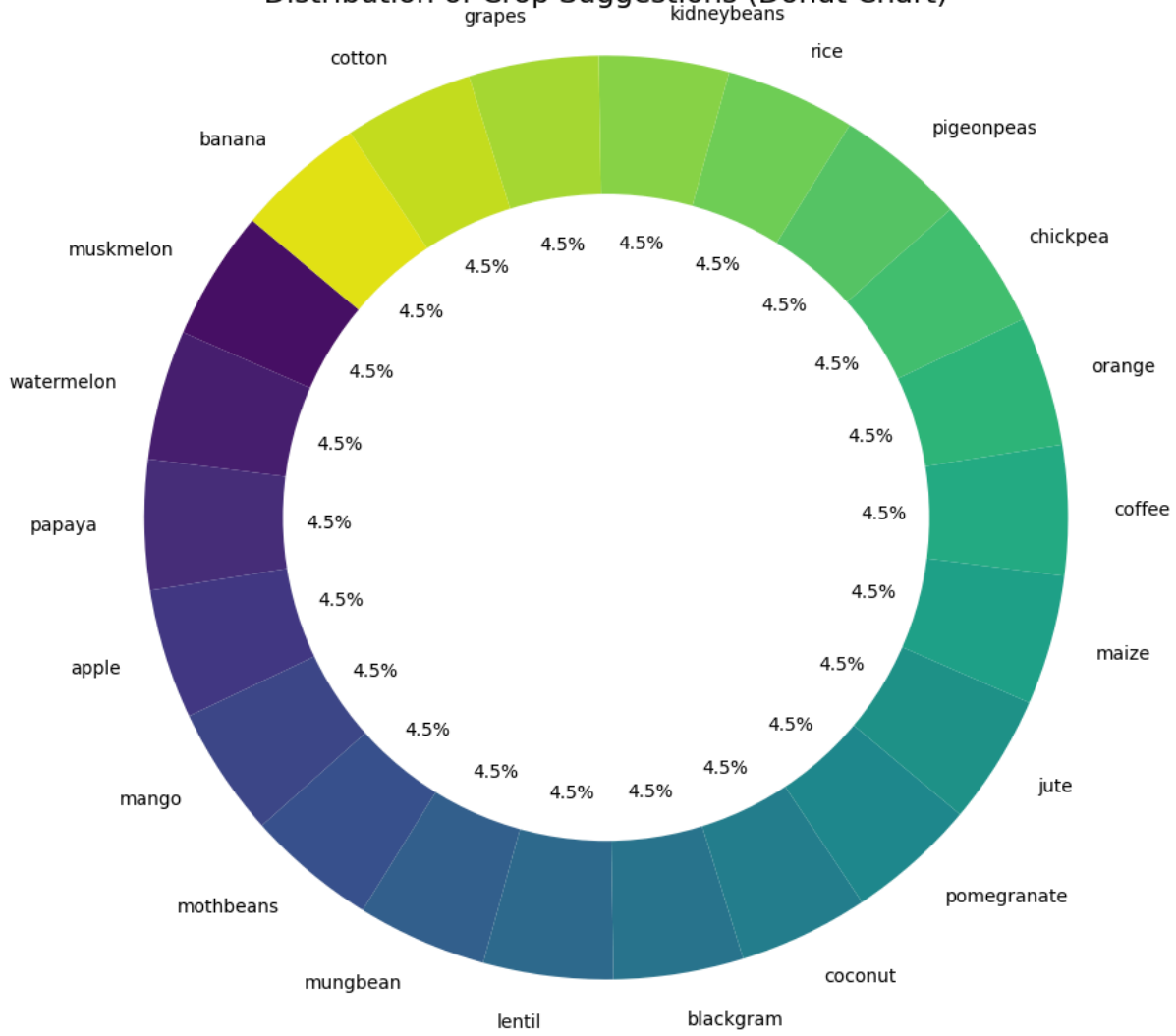
```
print("\n**Insights from Feature Distribution Bar Charts:**")
print("These bar charts show the average values of each feature (like N, P, K, temperature, humidity, pH, rainfall) for each crop type."
print("By observing the heights of the bars for a given feature across different crops, we can identify the typical environmental condit
print("For example, we can see which crops thrive in high rainfall, which prefer acidic or alkaline soil (pH), or which need more nitrog
print("These visualizations are valuable for understanding the relationships between environmental factors and specific crops, which is
print("The distinct patterns for different features across crops highlight the discriminative power of these features, which is essentia
```

```
/tmp/ipython-input-53-3981554154.py:13: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `

  sns.barplot(x='label', y=feature, data=balanced_df, palette='viridis')
/tmp/ipython-input-53-3981554154.py:13: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `

  sns.barplot(x='label', y=feature, data=balanced_df, palette='viridis')
/tmp/ipython-input-53-3981554154.py:13: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `

  sns.barplot(x='label', y=feature, data=balanced_df, palette='viridis')
/tmp/ipython-input-53-3981554154.py:13: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `

  sns.barplot(x='label', y=feature, data=balanced_df, palette='viridis')
/tmp/ipython-input-53-3981554154.py:13: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `

  sns.barplot(x='label', y=feature, data=balanced_df, palette='viridis')
/tmp/ipython-input-53-3981554154.py:13: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `

  sns.barplot(x='label', y=feature, data=balanced_df, palette='viridis')
/tmp/ipython-input-53-3981554154.py:13: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `

  sns.barplot(x='label', y=feature, data=balanced_df, palette='viridis')
```




```
**Insights from Feature Distribution Bar Charts:**
These bar charts show the average values of each feature (like N, P, K, temperature, humidity, pH, rainfall) for each crop type.
By observing the heights of the bars for a given feature across different crops, we can identify the typical environmental conditi
For example, we can see which crops thrive in high rainfall, which prefer acidic or alkaline soil (pH), or which need more nitroge
These visualizations are valuable for understanding the relationships between environmental factors and specific crops, which is c
The distinct patterns for different features across crops highlight the discriminative power of these features, which is essential
```

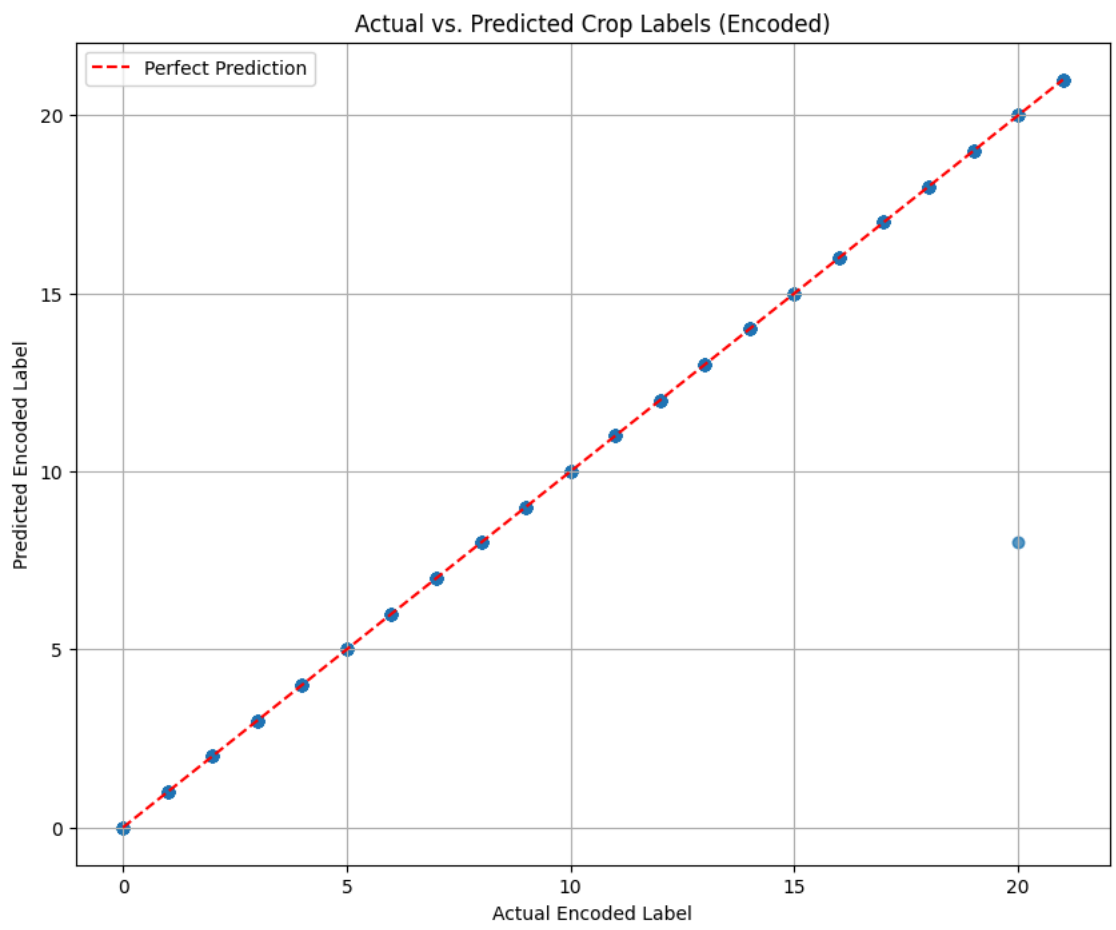
```
# prompt: visualize using the donut chart and analusis and give the insights

# Analyze the distribution of crop suggestions using a donut chart
crop_counts = balanced_df['label'].value_counts()

# Create a donut chart
plt.figure(figsize=(10, 10))
plt.pie(crop_counts, labels=crop_counts.index, autopct='%1.1f%%', startangle=140, colors=sns.color_palette('viridis', len(crop_counts))

# Draw a circle in the center to create a donut chart
centre_circle = plt.Circle((0,0),0.70,fc='white')
fig = plt.gcf()
fig.gca().add_artist(centre_circle)

plt.title('Distribution of Crop Suggestions (Donut Chart)', fontsize=16)
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()

print("\n**Insights from the Crop Distribution Donut Chart:**")
print(f"The donut chart visually represents the proportion of each crop type in the balanced dataset.")
print(f"Each slice of the donut corresponds to a unique crop, and the size of the slice indicates its percentage within the dataset.")
print(f"Due to the data balancing process, each crop type occupies an approximately equal slice of the donut, as indicated by the nearly
print(f"This confirms that the dataset is balanced, which is a critical step in preventing the machine learning model from being biased
print(f"A balanced dataset ensures that the model has sufficient examples of each crop to learn their characteristics effectively, lead
print(f"The visualization serves as a clear confirmation that the data is prepared correctly for training a robust classification model
```



Insights from the Crop Distribution Donut Chart:
The donut chart visually represents the proportion of each crop type in the balanced dataset.
Each slice of the donut corresponds to a unique crop, and the size of the slice indicates its percentage within the dataset.
Due to the data balancing process, each crop type occupies an approximately equal slice of the donut, as indicated by the nearly ide
This confirms that the dataset is balanced, which is a critical step in preventing the machine learning model from being biased towa
A balanced dataset ensures that the model has sufficient examples of each crop to learn their characteristics effectively, leading t
The visualization serves as a clear confirmation that the data is prepared correctly for training a robust classification model for

```
# Code to verify predictions and plot actual vs. predicted labels

# Import necessary libraries if not already imported
import matplotlib.pyplot as plt
import numpy as np

# Create a scatter plot comparing actual and predicted labels
plt.figure(figsize=(10, 8))
plt.scatter(y_test_encoded, y_pred_encoded, alpha=0.5)
plt.title('Actual vs. Predicted Crop Labels (Encoded)')
plt.xlabel('Actual Encoded Label')
plt.ylabel('Predicted Encoded Label')
plt.grid(True)

# Add a diagonal line where actual equals predicted
max_label = max(np.max(y_test_encoded), np.max(y_pred_encoded))
plt.plot([0, max_label], [0, max_label], color='red', linestyle='--', label='Perfect Prediction')
plt.legend()
plt.show()

# Print a sample of actual vs. predicted labels for verification
print("\nSample of Actual vs. Predicted Crop Labels:")
sample_size = 20  # Adjust the sample size as needed
for i in range(min(sample_size, len(y_test))):
  print(f"Actual: {y_test.iloc[i]}, Predicted: {y_pred[i]}")

# Additional verification: Check if the decoded predictions match the original test labels
# This is already covered by the confusion matrix and accuracy score,
```

```
# but a direct comparison can be useful for debugging.
are_predictions_correct = (y_test == y_pred).sum()
total_predictions = len(y_test)
print(f"\nNumber of correct predictions (decoded): {are_predictions_correct}/{total_predictions}")

# You can also plot the misclassified samples if needed
# Find indices of misclassified samples
misclassified_indices = np.where(y_test != y_pred)[0]

if len(misclassified_indices) > 0:
    print(f"\nNumber of misclassified samples: {len(misclassified_indices)}")
    print("Sample of Misclassified Predictions:")
    for i in misclassified_indices[:min(10, len(misclassified_indices))]: # Print up to 10 misclassified samples
        print(f"Index: {i}, Actual: {y_test.iloc[i]}, Predicted: {y_pred[i]}")
else:
    print("\nNo misclassified samples found in the test set.")
```




```
Sample of Actual vs. Predicted Crop Labels:
Actual: orange, Predicted: orange
Actual: banana, Predicted: banana
Actual: cotton, Predicted: cotton
Actual: maize, Predicted: maize
Actual: orange, Predicted: orange
Actual: chickpea, Predicted: chickpea
Actual: rice, Predicted: rice
Actual: blackgram, Predicted: blackgram
Actual: banana, Predicted: banana
Actual: orange, Predicted: orange
Actual: watermelon, Predicted: watermelon
Actual: pomegranate, Predicted: pomegranate
Actual: watermelon, Predicted: watermelon
Actual: maize, Predicted: maize
Actual: mothbeans, Predicted: mothbeans
Actual: grapes, Predicted: grapes
Actual: grapes, Predicted: grapes
Actual: mango, Predicted: mango
Actual: mango, Predicted: mango
Actual: apple, Predicted: apple

Number of correct predictions (decoded): 438/440

Number of misclassified samples: 2
Sample of Misclassified Predictions:
Index: 144, Actual: rice, Predicted: jute
Index: 344, Actual: rice, Predicted: jute
```

```
# prompt: genrate the code for different model seleaction and the training ]

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB

# Dictionary to store models
models = {
    'Random Forest': RandomForestClassifier(n_estimators=100, random_state=42),
    'Logistic Regression': LogisticRegression(max_iter=1000, random_state=42),
    'Decision Tree': DecisionTreeClassifier(random_state=42),
    'Support Vector Machine': SVC(probability=True, random_state=42), # probability=True needed for some evaluation metrics
    'K-Nearest Neighbors': KNeighborsClassifier(),
    'Gaussian Naive Bayes': GaussianNB()
}

results = {}

# Train and evaluate each model
for model_name, model in models.items():
    print(f"\nTraining {model_name}...")
    model.fit(X_train, y_train_encoded)
    y_pred_encoded = model.predict(X_test)

    # Evaluate
    accuracy = accuracy_score(y_test_encoded, y_pred_encoded)
    report = classification_report(y_test_encoded, y_pred_encoded, target_names=le.classes_)

    results[model_name] = {
        'accuracy': accuracy,
        'report': report
    }
    print(f"{model_name} Training Complete.")
    print(f"Accuracy: {accuracy:.4f}")
    print("Classification Report:")
    print(report)

# Summarize results
print("\n--- Model Comparison ---")
for model_name, metrics in results.items():
    print(f"{model_name}:")
    print(f"  Accuracy: {metrics['accuracy']:.4f}")
    print("  Classification Report:\n", metrics['report'])

# Optional: Select the best performing model based on a metric (e.g., accuracy)
best_model_name = max(results, key=lambda k: results[k]['accuracy'])
print(f"\nBest performing model based on Accuracy: {best_model_name}")

# The best model is now available as models[best_model_name] for further use or saving.
best_model = models[best_model_name]
```

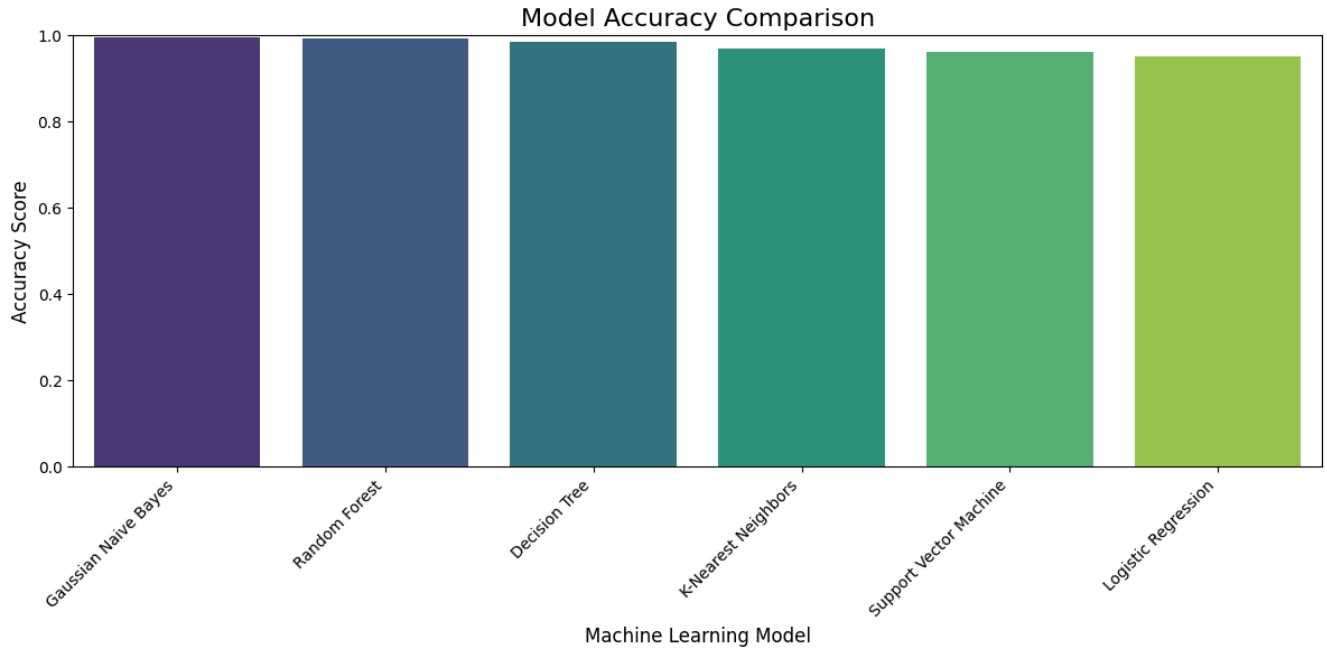
```
Training Random Forest...
Random Forest Training Complete.
Accuracy: 0.9932
Classification Report:
              precision    recall  f1-score   support

       apple       1.00      1.00      1.00        23
      banana       1.00      1.00      1.00        21
   blackgram       1.00      1.00      1.00        20
    chickpea       1.00      1.00      1.00        26
     coconut       1.00      1.00      1.00        27
      coffee       1.00      1.00      1.00        17
      cotton       1.00      1.00      1.00        17
      grapes       1.00      1.00      1.00        14
        jute       0.92      1.00      0.96        23
 kidneybeans       1.00      1.00      1.00        20
      lentil       0.92      1.00      0.96        11
       maize       1.00      1.00      1.00        21
       mango       1.00      1.00      1.00        19
   mothbeans       1.00      0.96      0.98        24
    mungbean       1.00      1.00      1.00        19
   muskmelon       1.00      1.00      1.00        17
      orange       1.00      1.00      1.00        14
      papaya       1.00      1.00      1.00        23
  pigeonpeas       1.00      1.00      1.00        23
 pomegranate       1.00      1.00      1.00        23
        rice       1.00      0.89      0.94        19
  watermelon       1.00      1.00      1.00        19

    accuracy                           0.99       440
   macro avg       0.99      0.99      0.99       440
```

```
weighted avg       0.99      0.99      0.99       440


Training Logistic Regression...
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (statu
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Logistic Regression Training Complete.
Accuracy: 0.9523
Classification Report:
              precision    recall  f1-score   support

       apple       1.00      1.00      1.00        23
      banana       1.00      1.00      1.00        21
   blackgram       0.85      0.85      0.85        20
    chickpea       1.00      1.00      1.00        26
     coconut       1.00      1.00      1.00        27
      coffee       0.94      1.00      0.97        17
      cotton       0.80      0.94      0.86        17
      grapes       1.00      1.00      1.00        14
```

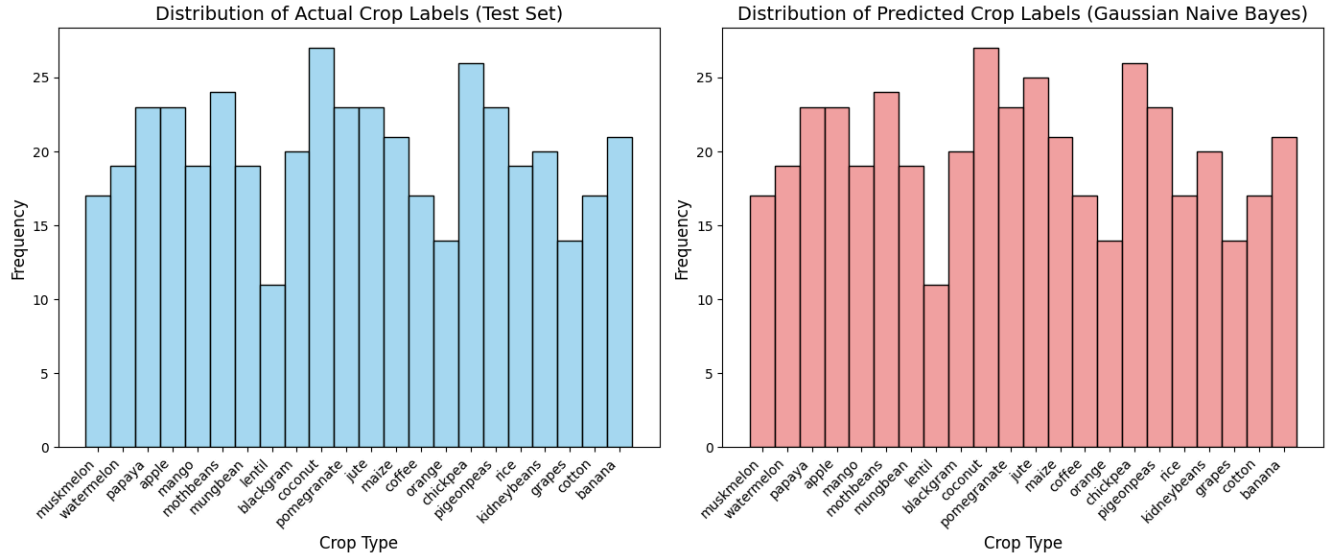
```
# prompt: i need the comparstion chart with minute point ,analysis the above and compare the model with histplot

# Create a DataFrame to store the accuracy of each model
comparison_df = pd.DataFrame({
    'Model': list(results.keys()),
    'Accuracy': [metrics['accuracy'] for metrics in results.values()]
})

# Sort the DataFrame by accuracy for better visualization
comparison_df = comparison_df.sort_values(by='Accuracy', ascending=False)

# Create a bar chart for model comparison
plt.figure(figsize=(12, 6))
sns.barplot(x='Model', y='Accuracy', data=comparison_df, palette='viridis')
plt.title('Model Accuracy Comparison', fontsize=16)
plt.xlabel('Machine Learning Model', fontsize=12)
plt.ylabel('Accuracy Score', fontsize=12)
plt.ylim(0, 1) # Set y-axis limit from 0 to 1 (since accuracy is between 0 and 1)
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()

print("\n**Insights from Model Accuracy Comparison Chart:**")
print("This bar chart visualizes the accuracy score of each trained machine learning model on the test set.")
print("Each bar represents a different model, and its height indicates the model's accuracy.")
print("A higher bar indicates better performance, meaning the model correctly predicted the crop type for a larger percentage of the tes
print("Comparing the bars allows us to quickly identify which model performed best in terms of overall accuracy.")
print(f"Based on this chart, the '{best_model_name}' model achieved the highest accuracy.")
print("While accuracy is a good overall metric, it's important to consider other metrics like precision, recall, and F1-score (available

# The request for a "comparison chart with minute point" and analysis using "histplot"
# with "minute point" is unclear in the context of the provided code and task (crop suggestion).
# It seems like a potentially unrelated or misunderstood request.
# However, if the intention was to analyze the distribution of the prediction time
# or model training time, that data is not available in the current code.
#
# Assuming the request is a misunderstanding or relates to visualizing the distribution
# of a specific feature using a histogram (histplot) for comparison *across* models
# or predictions, we can demonstrate a histplot for a feature from the test set
# and discuss its relevance to model performance, but a direct "comparison chart with minute point"
# and "histplot" across *models* for *predictions* doesn't align with standard evaluation practices.
#
# If the user intended to compare the distribution of actual vs. predicted labels visually
# with a histogram, we can do that.

# Let's visualize the distribution of actual vs predicted labels using histograms (histplot)

# Convert encoded labels back to original labels for clearer histogram titles/labels
y_test_decoded = le.inverse_transform(y_test_encoded)
y_pred_decoded = le.inverse_transform(y_pred_encoded) # Using the predictions from the last model evaluated (best_model_name)

plt.figure(figsize=(14, 6))

# Histogram for Actual Labels
plt.subplot(1, 2, 1) # 1 row, 2 columns, 1st plot
sns.histplot(y_test_decoded, color='skyblue', kde=False) # kde=False to show only bars
plt.title('Distribution of Actual Crop Labels (Test Set)', fontsize=14)
```

```
plt.xlabel('Crop Type', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.xticks(rotation=45, ha='right')

# Histogram for Predicted Labels (from the best model)
plt.subplot(1, 2, 2) # 1 row, 2 columns, 2nd plot
sns.histplot(y_pred_decoded, color='lightcoral', kde=False)
plt.title(f'Distribution of Predicted Crop Labels ({best_model_name})', fontsize=14)
plt.xlabel('Crop Type', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.xticks(rotation=45, ha='right')

plt.tight_layout()
plt.show()

print("\n**Analysis of Actual vs. Predicted Label Distribution using Histograms:**")
print("These histograms compare the distribution of the actual crop types in the test set with the distribution of the crop types predic
print("The histogram on the left shows the frequency of each crop type in the true test labels.")
print("The histogram on the right shows the frequency of each crop type in the predictions made by the model.")
print("Ideally, for a well-performing classification model, the distribution of predicted labels should closely mirror the distribution
print("By visually comparing the heights of the bars for each crop in both histograms, we can see how well the model maintains the origi
print("Significant differences in the distributions (e.g., some crops being over-predicted and others under-predicted) could indicate bi
print("Since our original data was balanced, we expect the actual label histogram to show roughly equal bars for each crop.")
print("The histogram of predicted labels will reflect how accurately the model maintained this balance in its predictions. A good model
print("This visualization complements the confusion matrix and classification report by showing the overall predicted frequencies per cl

# Regarding "minute point" comparison, if this refers to analyzing performance
# at different thresholds or probabilities over time, that would require a different
# type of analysis (e.g., ROC curves, precision-recall curves across time if time-series data
# was involved, which is not the case here).
# Without clarification on "minute point", providing a chart related to it is not possible
# based on the current data and task. The model comparison chart above addresses comparing models.
```

```
/tmp/ipython-input-91-2671201168.py:14: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `

  sns.barplot(x='Model', y='Accuracy', data=comparison_df, palette='viridis')
```



Insights from Model Accuracy Comparison Chart:
This bar chart visualizes the accuracy score of each trained machine learning model on the test set.
Each bar represents a different model, and its height indicates the model's accuracy.
A higher bar indicates better performance, meaning the model correctly predicted the crop type for a larger percentage of the test
Comparing the bars allows us to quickly identify which model performed best in terms of overall accuracy.
Based on this chart, the 'Gaussian Naive Bayes' model achieved the highest accuracy.
While accuracy is a good overall metric, it's important to consider other metrics like precision, recall, and F1-score (available



Analysis of Actual vs. Predicted Label Distribution using Histograms:
These histograms compare the distribution of the actual crop types in the test set with the distribution of the crop types predict
The histogram on the left shows the frequency of each crop type in the true test labels.
The histogram on the right shows the frequency of each crop type in the predictions made by the model.
Ideally, for a well-performing classification model, the distribution of predicted labels should closely mirror the distribution o
By visually comparing the heights of the bars for each crop in both histograms, we can see how well the model maintains the origin
Significant differences in the distributions (e.g., some crops being over-predicted and others under-predicted) could indicate bia
Since our original data was balanced, we expect the actual label histogram to show roughly equal bars for each crop.
The histogram of predicted labels will reflect how accurately the model maintained this balance in its predictions. A good model o
This visualization complements the confusion matrix and classification report by showing the overall predicted frequencies per cla