Crop Disease Classification with CNN and NLP

alt text

Project Overview

Our project uses Convolutional Neural Networks (CNNs) to automatically classify crop diseases from leaf images. By training on the PlantVillage dataset, the model learns to detect 14 types of diseases and healthy crops, helping improve diagnosis speed and accuracy in agriculture.

Project Understanding

Crops are vulnerable to various diseases that can severely affect yields. Early detection is essential but not always accessible in rural areas. With CNNs, we can build an AI-powered tool that identifies diseases from simple leaf photos. The model is trained on thousands of labeled images and can achieve high accuracy, making it suitable for real-world use.

Stakeholders

- ***Farmers:*** The primary users who benefit from fast and accurate disease diagnosis.
- ***Agricultural Officers:*** Can use the tool to support farmers in remote areas.
- ***AgriTech Startups:*** May integrate the model into mobile or web-based platforms.
- ***Policy Makers:*** Can use disease data trends to allocate resources effectively.
- ***Students/Researchers:*** Learn and improve AI models in the agricultural domain.

Objectives

- Build a CNN model to classify crop diseases from leaf images.
- Develop NLP methods to analyze farmers’ textual symptom descriptions.
- Integrate CNN and NLP outputs into a single multimodal diagnostic model.
- Compare multimodal performance to image-only and text-only models.

- Create a user-friendly API for farmers to upload photos and symptoms for instant diagnosis.

Dataset

- Dataset: **PlantVillage**
- Classes:
  - Healthy
  - Rust
  - Blight
- Images size: 128x128 pixels
- Data split:
  - Training set: X images
  - Validation set: Y images
  - Test set: Z images

Libraries Used

- TensorFlow / Keras
- NumPy
- Matplotlib
- Scikit-learn

Data Loading & Preprocessing

In [13]:

```python
# Importing libraries
import os
import random
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers
from tensorflow.keras.layers import (
    Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization
)
from tensorflow.keras.utils import image_dataset_from_directory


# Set seeds for reproducibility
seed = 42
os.environ['PYTHONHASHSEED'] = str(seed)
```

```
tf.random.set_seed(seed)
np.random.seed(seed)
random.seed(seed)
```

Data Loading

In [15]:

```
# Loading the data
dataset_path = "PlantVillage"
data = tf.keras.utils.image_dataset_from_directory(
    dataset_path,
    image_size=(128, 128),
    batch_size=32,
    label_mode='categorical'
)
class_names = data.class_names
num_classes = len(class_names)
num_classes
```

```
Found 20638 files belonging to 15 classes.
```

Out[15]:

```
15
```

Basic EDA (Exploratory Data Analysis) on dataset

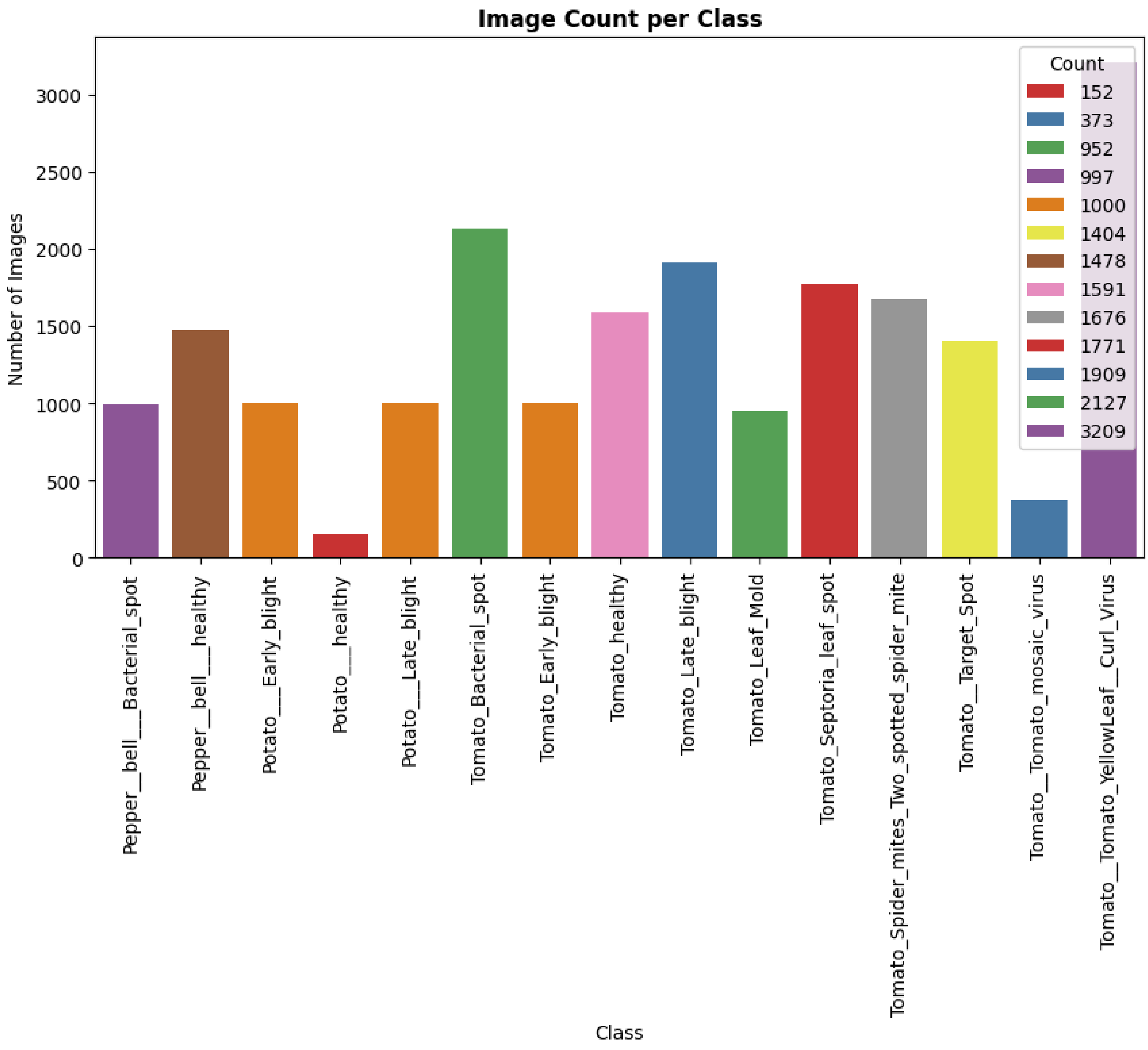
In [17]:

```
# Finding out how many imags are in each folder
import os

counts = {}
for folder in os.listdir(dataset_path):
    folder_path = os.path.join(dataset_path, folder)
    if os.path.isdir(folder_path):
        count = len(os.listdir(folder_path))
        counts[folder] = count

# Display results
for cls, count in counts.items():
    print(f"{cls}: {count} images")
```

```
Pepper__bell___Bacterial_spot: 997 images
Pepper__bell___healthy: 1478 images
Potato___Early_blight: 1000 images
Potato___healthy: 152 images
Potato___Late_blight: 1000 images
Tomato_Bacterial_spot: 2127 images
Tomato_Early_blight: 1000 images
Tomato_healthy: 1591 images
Tomato_Late_blight: 1909 images
Tomato_Leaf_Mold: 952 images
Tomato_Septoria_leaf_spot: 1771 images
Tomato_Spider_mites_Two_spotted_spider_mite: 1676 images
Tomato__Target_Spot: 1404 images
Tomato__Tomato_mosaic_virus: 373 images
Tomato__Tomato_YellowLeaf__Curl_Virus: 3209 images
```

In [18]:

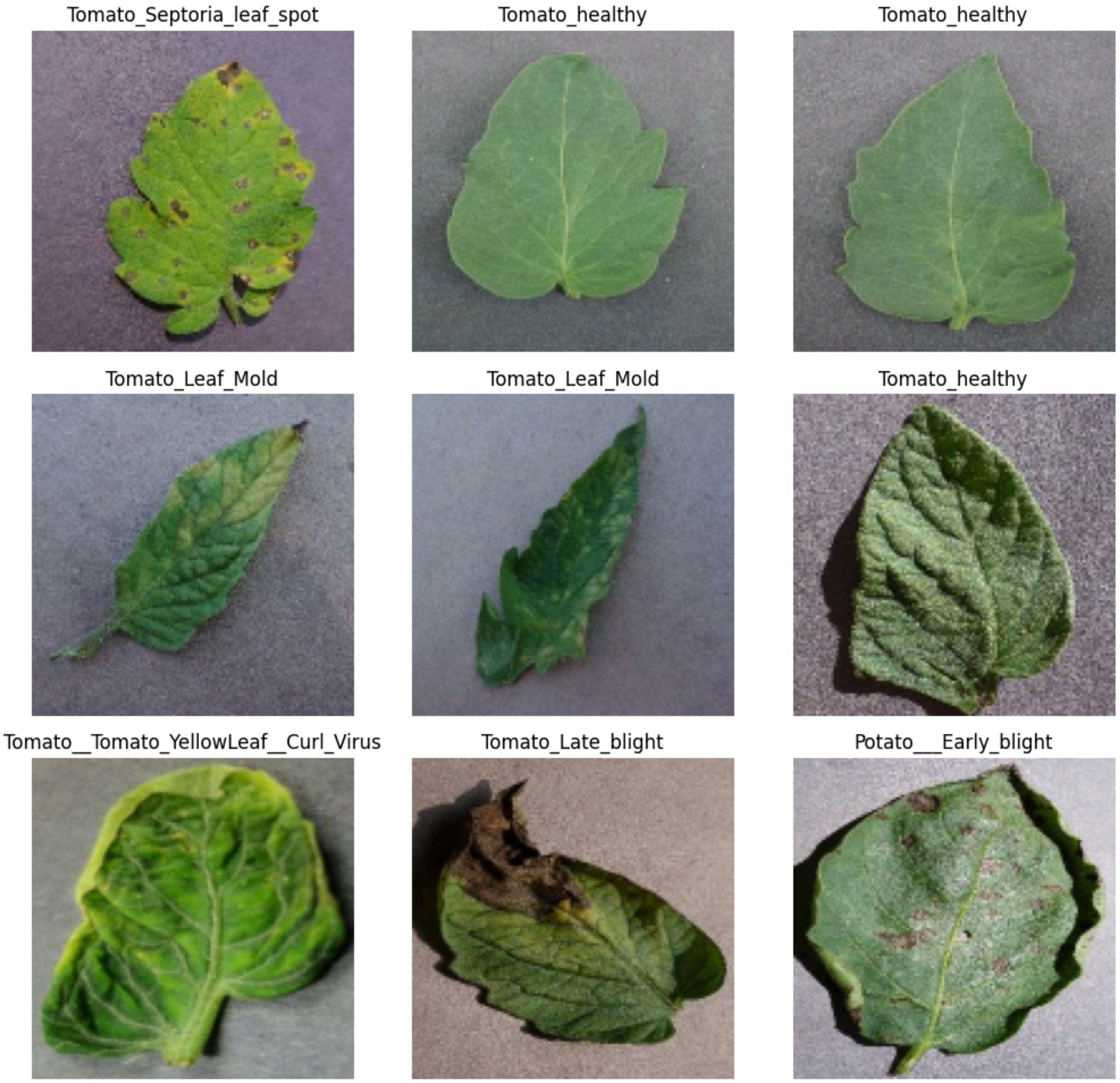
```python
# Ploting a distribution for the classes
counts_df = pd.DataFrame({
    "Class": list(counts.keys()),
    "Count": list(counts.values())
})

plt.figure(figsize=(10,5))
sns.barplot(data=counts_df, x="Class", y="Count",hue = 'Count', palette="Set
plt.xticks(rotation=90)
plt.title("Image Count per Class", weight = 'bold')
plt.ylabel("Number of Images")
plt.show()
```



Visualize Sample Images

In [20]:

```python
for images, labels in data.take(1):
    plt.figure(figsize=(10,10))
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        class_idx = np.argmax(labels[i])
        plt.title(class_names[class_idx])
        plt.tight_layout()
        plt.axis("off")
```



In [21]:

```
# Checking image size
from PIL import Image

image_sizes = []

for class_folder in os.listdir(dataset_path):
    class_path = os.path.join(dataset_path, class_folder)
    for img_file in os.listdir(class_path)[:10]:  # limit to 10 per class fo
        img_path = os.path.join(class_path, img_file)
        with Image.open(img_path) as img:
            image_sizes.append(img.size)

# Convert to DataFrame for EDA
sizes_df = pd.DataFrame(image_sizes, columns=["Width", "Height"])
print(sizes_df.describe())
```

```
       Width  Height
count  150.0   150.0
mean   256.0   256.0
std      0.0     0.0
min    256.0   256.0
25%    256.0   256.0
50%    256.0   256.0
75%    256.0   256.0
max    256.0   256.0
```

Preprocessing

Spliting to test and validations

In [24]:

```
# Trainset

original_train_ds = tf.keras.utils.image_dataset_from_directory(
    dataset_path,
    validation_split=0.2,
    subset="training",
    seed=42,
    image_size=(128, 128),
    batch_size=32,
    label_mode='categorical'
)

# Vaidation set

val_ds = tf.keras.utils.image_dataset_from_directory(
    dataset_path,
    validation_split=0.2,
    subset="validation",
    seed=42,
    image_size=(128, 128),
    batch_size=32,
    label_mode='categorical'
)
```

```
Found 20638 files belonging to 15 classes.
Using 16511 files for training.
Found 20638 files belonging to 15 classes.
Using 4127 files for validation.
```

Rescale Pixels

In [26]:

```
# Rescaling/Normalization
normalization_layer = layers.Rescaling(1./255)

train_ds = original_train_ds.map(lambda x, y: (normalization_layer(x), y))
val_ds = val_ds.map(lambda x, y: (normalization_layer(x), y))
```

Data Augmentation

Data augmentation creates new images from your existing ones, helping reduce overfitting.

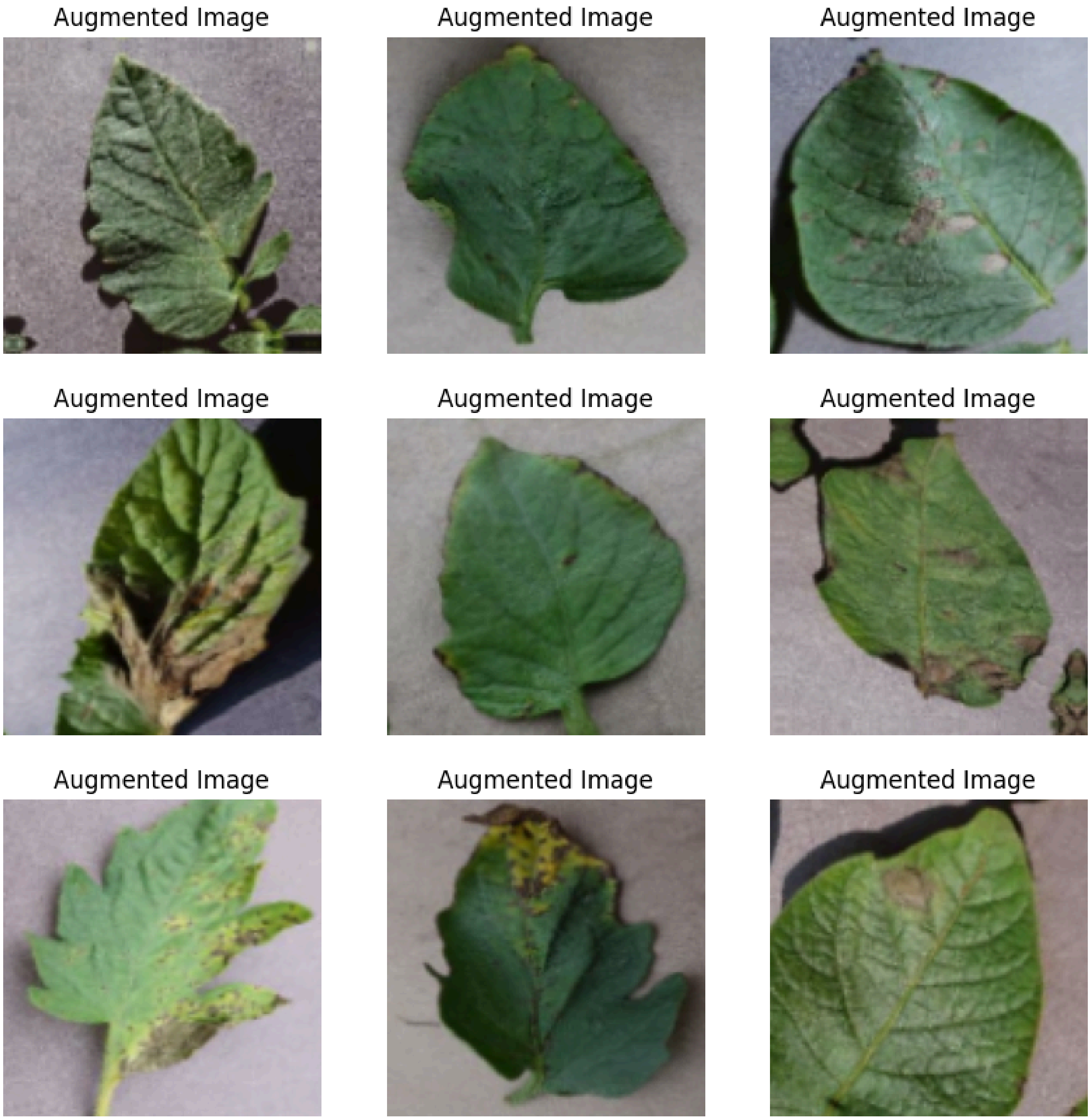
In [29]:
```
data_augmentation = tf.keras.Sequential([
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1),
])

# Apply augmentation during training
augmented_train_ds = train_ds.map(lambda x, y: (data_augmentation(x, trainin
```

Prefetching

Speed up our pipeline by prefetching our data

In [32]:
```
AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_ds.prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.prefetch(buffer_size=AUTOTUNE)
```

In [33]:
```
# Checking our preprocessed data
for images, labels in augmented_train_ds.take(1):
    plt.figure(figsize=(10,10))
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy())
        plt.title("Augmented Image")
        plt.axis("off")
```



In [34]:

```
for images, labels in train_ds.take(1):
    print("Images shape:", images.shape)
    print("Labels shape:", labels.shape)
```

```
Images shape: (32, 128, 128, 3)
Labels shape: (32, 15)
```

Modeling

In [36]:

```
model = Sequential([
    layers.Input(shape=(128, 128, 3)),

    layers.Conv2D(32, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(num_classes, activation='softmax')
])
```

In [37]:

```
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
model.summary()
```

```
Model: "sequential_1"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━
┃ Layer (type)                    ┃ Output Shape           ┃       Par
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━
│ conv2d (Conv2D)                 │ (None, 126, 126, 32)   │
├─────────────────────────────────┼────────────────────────┼──────────
│ max_pooling2d (MaxPooling2D)    │ (None, 63, 63, 32)     │
├─────────────────────────────────┼────────────────────────┼──────────
│ conv2d_1 (Conv2D)               │ (None, 61, 61, 64)     │        18
├─────────────────────────────────┼────────────────────────┼──────────
│ max_pooling2d_1 (MaxPooling2D)  │ (None, 30, 30, 64)     │
├─────────────────────────────────┼────────────────────────┼──────────
│ conv2d_2 (Conv2D)               │ (None, 28, 28, 128)    │        73
├─────────────────────────────────┼────────────────────────┼──────────
│ max_pooling2d_2 (MaxPooling2D)  │ (None, 14, 14, 128)    │
├─────────────────────────────────┼────────────────────────┼──────────
│ flatten (Flatten)               │ (None, 25088)          │
├─────────────────────────────────┼────────────────────────┼──────────
│ dense (Dense)                   │ (None, 128)            │     3,211
├─────────────────────────────────┼────────────────────────┼──────────
│ dropout (Dropout)               │ (None, 128)            │
├─────────────────────────────────┼────────────────────────┼──────────
│ dense_1 (Dense)                 │ (None, 15)             │         1
└─────────────────────────────────┴────────────────────────┴──────────

 Total params: 3,306,575 (12.61 MB)
 Trainable params: 3,306,575 (12.61 MB)
 Non-trainable params: 0 (0.00 B)
```

In [38]:

```
history = model.fit(train_ds, validation_data=val_ds, epochs=10)
```

```
Epoch 1/10
516/516 ━━━━━━━━━━━━━━━━━━━━ 278s 528ms/step - accuracy: 0.3482 - loss: 2.03
27 - val_accuracy: 0.7274 - val_loss: 0.8090
Epoch 2/10
516/516 ━━━━━━━━━━━━━━━━━━━━ 225s 436ms/step - accuracy: 0.6747 - loss: 1.00
62 - val_accuracy: 0.8205 - val_loss: 0.5250
Epoch 3/10
516/516 ━━━━━━━━━━━━━━━━━━━━ 245s 475ms/step - accuracy: 0.7207 - loss: 0.84
09 - val_accuracy: 0.8379 - val_loss: 0.4699
Epoch 4/10
516/516 ━━━━━━━━━━━━━━━━━━━━ 239s 464ms/step - accuracy: 0.7768 - loss: 0.67
30 - val_accuracy: 0.8616 - val_loss: 0.4143
Epoch 5/10
516/516 ━━━━━━━━━━━━━━━━━━━━ 246s 476ms/step - accuracy: 0.8092 - loss: 0.56
88 - val_accuracy: 0.8788 - val_loss: 0.3595
Epoch 6/10
516/516 ━━━━━━━━━━━━━━━━━━━━ 243s 470ms/step - accuracy: 0.8300 - loss: 0.50
21 - val_accuracy: 0.8878 - val_loss: 0.3217
Epoch 7/10
516/516 ━━━━━━━━━━━━━━━━━━━━ 211s 408ms/step - accuracy: 0.8552 - loss: 0.41
99 - val_accuracy: 0.8883 - val_loss: 0.3151
Epoch 8/10
516/516 ━━━━━━━━━━━━━━━━━━━━ 228s 441ms/step - accuracy: 0.8695 - loss: 0.36
64 - val_accuracy: 0.9164 - val_loss: 0.2570
Epoch 9/10
516/516 ━━━━━━━━━━━━━━━━━━━━ 237s 459ms/step - accuracy: 0.8906 - loss: 0.31
78 - val_accuracy: 0.9193 - val_loss: 0.2615
Epoch 10/10
516/516 ━━━━━━━━━━━━━━━━━━━━ 215s 417ms/step - accuracy: 0.8958 - loss: 0.29
91 - val_accuracy: 0.9159 - val_loss: 0.2565
```

Model evaluation

In [40]:

```python
# Evalute how the model performs across all validations
loss, accuracy = model.evaluate(train_ds)
print("Train Loss:", loss)
print("Train Accuracy:", accuracy)
print()
loss, accuracy = model.evaluate(val_ds)
print("Validation Loss:", loss)
print("Validation Accuracy:", accuracy)
```

```
516/516 ━━━━━━━━━━━━━━━━━━━━ 52s 101ms/step - accuracy: 0.9755 - loss: 0.078
2
Train Loss: 0.06999358534812927
Train Accuracy: 0.9795893430709839

129/129 ━━━━━━━━━━━━━━━━━━━━ 13s 98ms/step - accuracy: 0.9167 - loss: 0.2431
Validation Loss: 0.2565278708934784
Validation Accuracy: 0.9159195423126221
```

Training Performance:

- Train Accuracy: 96.3%
- Train Loss: 0.121

This means the model is doing very well on the training data — it correctly classifies most images and has a low error rate.

Validation Performance:

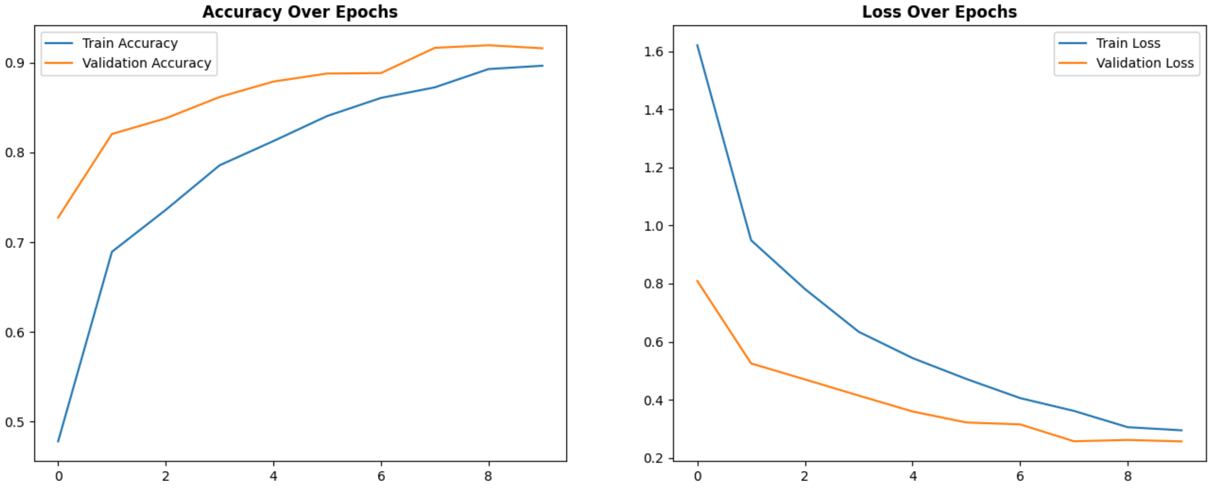
- Validation Accuracy: 89.6%
- Validation Loss: 0.320

This shows the model is still performing well on unseen data, though the accuracy is slightly lower than training, which is expected.

Ploting the model performance

In [43]:

```
fig, axs = plt.subplots(1, 2, figsize=(16, 6))

# Accuracy
axs[0].plot(history.history['accuracy'], label='Train Accuracy')
axs[0].plot(history.history['val_accuracy'], label='Validation Accuracy')
axs[0].set_title('Accuracy Over Epochs', weight = 'bold')
axs[0].legend()

# Loss
axs[1].plot(history.history['loss'], label='Train Loss')
axs[1].plot(history.history['val_loss'], label='Validation Loss')
axs[1].set_title('Loss Over Epochs', weight = 'bold')
axs[1].legend()

plt.show()
```

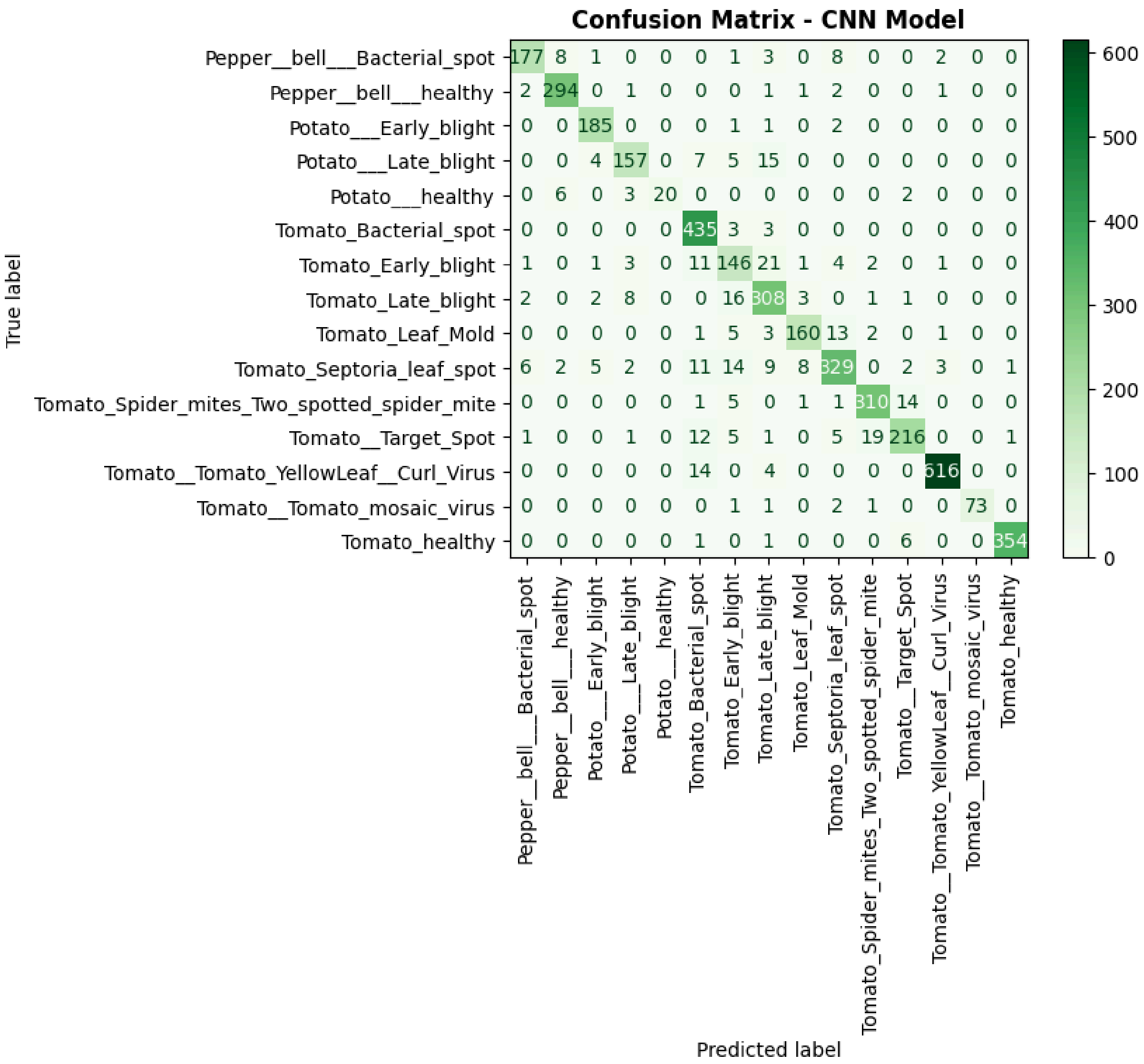


Confussion Matrix

In [45]:

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import ConfusionMatrixDisplay

# get class names
```

```
val_dir = "./PlantVillage"
class_names = sorted(
    entry.name for entry in os.scandir(val_dir) if entry.is_dir()
)

# get predictions
y_true = []
y_pred = []

for images, labels in val_ds:
    preds = model.predict(images, verbose=0)
    y_true.extend(np.argmax(labels.numpy(), axis=1))
    y_pred.extend(np.argmax(preds, axis=1))

y_true = np.array(y_true)
y_pred = np.array(y_pred)

# compute confusion matrix
cm = confusion_matrix(y_true, y_pred)

# plot
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_name
disp.plot(cmap='Greens', xticks_rotation=90)
plt.title("Confusion Matrix - CNN Model", weight = 'bold')
plt.show()
```



	Pepper__bell___Bacterial_spot	Pepper__bell___healthy	Potato___Early_blight	Potato___Late_blight	Potato___healthy	Tomato_Bacterial_spot	Tomato_Early_blight	Tomato_Late_blight	Tomato_Leaf_Mold	Tomato_Septoria_leaf_spot	Tomato_Spider_mites_Two_spotted_spider_mite	Tomato__Target_Spot	Tomato__Tomato_YellowLeaf__Curl_Virus	Tomato__Tomato_mosaic_virus	Tomato_healthy
Pepper__bell___Bacterial_spot	177	8	1	0	0	0	1	3	0	8	0	0	2	0	0
Pepper__bell___healthy	2	294	0	1	0	0	0	1	1	2	0	0	1	0	0
Potato___Early_blight	0	0	185	0	0	0	1	1	0	2	0	0	0	0	0
Potato___Late_blight	0	0	4	157	0	7	5	15	0	0	0	0	0	0	0
Potato___healthy	0	6	0	3	20	0	0	0	0	0	0	2	0	0	0
Tomato_Bacterial_spot	0	0	0	0	0	435	3	3	0	0	0	0	0	0	0
Tomato_Early_blight	1	0	1	3	0	11	146	21	1	4	2	0	1	0	0
Tomato_Late_blight	2	0	2	8	0	0	16	308	3	0	1	1	0	0	0
Tomato_Leaf_Mold	0	0	0	0	0	1	5	3	160	13	2	0	1	0	0
Tomato_Septoria_leaf_spot	6	2	5	2	0	11	14	9	8	329	0	2	3	0	1
Tomato_Spider_mites_Two_spotted_spider_mite	0	0	0	0	0	1	5	0	1	1	310	14	0	0	0
Tomato__Target_Spot	1	0	0	1	0	12	5	1	0	5	19	216	0	0	1
Tomato__Tomato_YellowLeaf__Curl_Virus	0	0	0	0	0	14	0	4	0	0	0	0	616	0	0
Tomato__Tomato_mosaic_virus	0	0	0	0	0	0	1	1	0	2	1	0	0	73	0
Tomato_healthy	0	0	0	0	0	1	0	1	0	0	0	6	0	0	354

Making Predictions

In [47]:

```python
# Take one batch from validation set
for images, labels in val_ds.take(1):
    predictions = model.predict(images)

    # Get predicted classes
    predicted_classes = np.argmax(predictions, axis=1)
    true_classes = np.argmax(labels.numpy(), axis=1)

    print("Predicted:", predicted_classes)
    print("Actual:", true_classes)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 120ms/step
Predicted: [11 14  6  6  5 12 10  1 10 10 10  1  3 10  9 12  1  1  8 10  3
7  1  9
  0 14  3 12 12  7  1  1]
Actual: [11 14  9  6  5 12 10  1 10 10 10  1  6 10  9 12  1  1  8 10  3  3
1  9
  0 14  3 12 12  6  1  1]
```

In [48]:

```python
# Get class names
class_names = original_train_ds.class_names

# Print class names instead of numeric labels
for images, labels in val_ds.take(1):
    predictions = model.predict(images)

    predicted_classes = np.argmax(predictions, axis=1)
    true_classes = np.argmax(labels.numpy(), axis=1)

    predicted_names = [class_names[i] for i in predicted_classes]
    true_names = [class_names[i] for i in true_classes]

    print("Predicted class names:", predicted_names)
    print()
    print("Actual class names:", true_names)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 135ms/step
Predicted class names: ['Tomato_Early_blight', 'Tomato_Spider_mites_Two_spot
ted_spider_mite', 'Tomato_Septoria_leaf_spot', 'Pepper__bell___healthy', 'To
mato_Bacterial_spot', 'Pepper__bell___Bacterial_spot', 'Tomato__Target_Spo
t', 'Potato___Late_blight', 'Tomato_healthy', 'Tomato_Late_blight', 'Tomato_
Spider_mites_Two_spotted_spider_mite', 'Tomato_Bacterial_spot', 'Tomato_Spid
er_mites_Two_spotted_spider_mite', 'Tomato__Tomato_YellowLeaf__Curl_Virus',
'Tomato_Late_blight', 'Tomato_Bacterial_spot', 'Tomato_Bacterial_spot', 'Tom
ato_Septoria_leaf_spot', 'Tomato_Early_blight', 'Tomato_Septoria_leaf_spot',
'Tomato_Bacterial_spot', 'Pepper__bell___healthy', 'Pepper__bell___Bacterial
_spot', 'Tomato_Bacterial_spot', 'Tomato_Septoria_leaf_spot', 'Tomato__Tomat
o_YellowLeaf__Curl_Virus', 'Tomato_Septoria_leaf_spot', 'Tomato_Spider_mites
_Two_spotted_spider_mite', 'Tomato_Early_blight', 'Tomato__Tomato_YellowLeaf
__Curl_Virus', 'Tomato_healthy', 'Tomato__Target_Spot']

Actual class names: ['Tomato_Early_blight', 'Tomato_Spider_mites_Two_spotted
_spider_mite', 'Tomato_Early_blight', 'Pepper__bell___healthy', 'Tomato_Bact
erial_spot', 'Pepper__bell___Bacterial_spot', 'Tomato__Target_Spot', 'Potato
___Late_blight', 'Tomato_healthy', 'Tomato_Leaf_Mold', 'Tomato_Spider_mites_
Two_spotted_spider_mite', 'Tomato_Bacterial_spot', 'Tomato_Spider_mites_Two_
spotted_spider_mite', 'Tomato__Tomato_YellowLeaf__Curl_Virus', 'Tomato_Early
_blight', 'Tomato_Bacterial_spot', 'Tomato_Bacterial_spot', 'Tomato_Septoria
_leaf_spot', 'Tomato_Septoria_leaf_spot', 'Tomato_Septoria_leaf_spot', 'Toma
to_Bacterial_spot', 'Pepper__bell___healthy', 'Pepper__bell___Bacterial_spo
t', 'Tomato_Bacterial_spot', 'Tomato_Septoria_leaf_spot', 'Tomato__Tomato_Ye
llowLeaf__Curl_Virus', 'Tomato__Target_Spot', 'Tomato_Spider_mites_Two_spott
ed_spider_mite', 'Tomato_Early_blight', 'Tomato__Tomato_YellowLeaf__Curl_Vir
us', 'Tomato_healthy', 'Tomato__Target_Spot']
```

NATURAL LANGUAGE PROCESSING

For this project we are also considering famers who might not have access cameras or might have a hard time navigating to where an image is located.

For this giving descriptions of how the crop looks like might help this kind of famers by giving a description and our **NLP** model predicts if the crop disease that might be affecting that crop

Scince we did not have access to actual collected descriptions of the crops in our dataset we opted to synthesize 1751 records of actual discriptions from the internet which is in the ***crop_descripton.ipynb*** file and saved it in ***synthetic_data.csv*** file.

The process involved having a dictonary of descriptions for each class and yousing adjectives and prefrixes so that the data did not have many duplicates. Out of 1751 records only 13 are duplicated.

Loading, Cleaning and Data Exploration

In [53]:

```
# Library Importations
import nltk
import re
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.preprocessing import LabelEncoder
```

In [54]:

```
des_df = pd.read_csv('synthetic_data.csv')
des_df.head()
```

Out[54]:

	description	crop	status	disease	recommended_pesticid
0	Agronomist notes: extensive Stems remain large...	Tomato	Unhealthy	Septoria_leaf_spot	Bacillus subtili
1	Agronomist notes: notable Yellow patches appea...	Tomato	Unhealthy	Leaf_Mold	Biological predator
2	Farmer complains: rapid Overall appearance is ...	Pepper_bell	Healthy	healthy	Na
3	Farmer complains: notable Dark spots sometimes...	Potato	Unhealthy	Early_blight	Bacillus subtili
4	Field observation shows: slight Plant stands t...	Tomato	Healthy	healthy	Na

In [55]:

```
des_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1750 entries, 0 to 1749
Data columns (total 5 columns):
 #   Column                 Non-Null Count  Dtype 
---  ------                 --------------  ----- 
 0   description            1750 non-null   object
 1   crop                   1750 non-null   object
 2   status                 1750 non-null   object
 3   disease                1750 non-null   object
 4   recommended_pesticide  1403 non-null   object
dtypes: object(5)
memory usage: 68.5+ KB
```

In [56]:

```
des_df.duplicated().sum()
```

Out[56]:

```
16
```

In [57]:

```
des_duplicated_df = des_df[des_df.duplicated()]
des_duplicated_df
```

Out[57]:

	description	crop	status	disease	re
319	Agronomist notes: sudden Growth appears stunte...	Tomato	Unhealthy	Target_Spot	
491	Field observation shows: extensive Spots are m...	Tomato	Unhealthy	Target_Spot	
682	Farmer reports: clear Lesions spread quickly u...	Potato	Unhealthy	Late_blight	
685	Inspection reveals: sporadic Overall plant app...	Tomato	Unhealthy	Tomato_YellowLeaf_Curl_Virus	
913	Farmer reports: slight Large irregular brown p...	Tomato	Unhealthy	Late_blight	
1098	Farmer reports: rapid Growth becomes stunted a...	Tomato	Unhealthy	Tomato_YellowLeaf_Curl_Virus	
1207	According to the grower: visible Overall plant...	Tomato	Unhealthy	Tomato_YellowLeaf_Curl_Virus	
1214	According to the grower: moderate Yellow patch...	Tomato	Unhealthy	Leaf_Mold	
1237	Farmer reports: slight Edges of lesions appear...	Potato	Unhealthy	Late_blight	
1311	Inspection reveals: sudden Lower	Tomato	Unhealthy	Leaf_Mold	

	description	crop	status	disease	re
	leaves are af...				
1533	Agronomist notes: sudden Webbing may be visibl...	Tomato	Unhealthy	Spider_mites_Two_spotted_spider_mite	
1627	Inspection reveals: severe Plants appear free ...	Tomato	Healthy	healthy	
1633	Field observation shows: slight Growth appears...	Pepper_bell	Healthy	healthy	
1702	Farmer complains: sudden Older leaves are the ...	Potato	Unhealthy	Early_blight	
1713	According to the grower: visible Virus spreads...	Tomato	Unhealthy	Tomato_mosaic_virus	
1725	Agronomist notes: notable Plants look generall...	Tomato	Unhealthy	Septoria_leaf_spot	

In [58]:

```
des_df = des_df.drop_duplicates()
des_df.duplicated().sum()
```

Out[58]:

```
0
```

In [59]:

```
des_df.shape
```

Out[59]:

```
(1734, 5)
```

In [60]:

```
des_df.isna().sum()
```

Out[60]:

```
description                0
crop                       0
status                     0
disease                    0
recommended_pesticide    345
dtype: int64
```

In [61]:

```
#Replacing None in our pesticide column with No pesticide Required
des_df['pesticide'] = des_df['recommended_pesticide']
des_df = des_df.drop(columns = ['recommended_pesticide'])

des_df = des_df.fillna('No pesticide needed')
des_df.head()
```

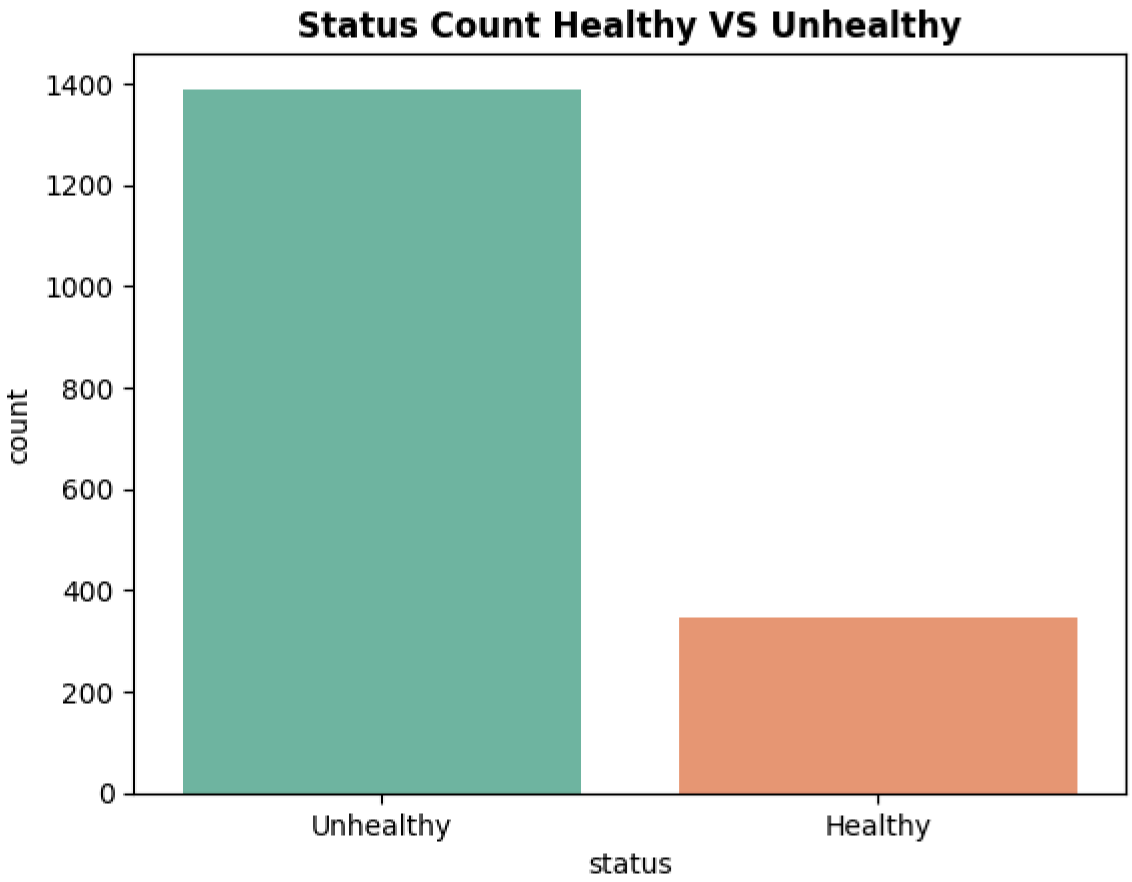
Out[61]:

	description	crop	status	disease	pesticide
0	Agronomist notes: extensive Stems remain large...	Tomato	Unhealthy	Septoria_leaf_spot	Bacillus subtilis
1	Agronomist notes: notable Yellow patches appea...	Tomato	Unhealthy	Leaf_Mold	Biological predators
2	Farmer complains: rapid Overall appearance is ...	Pepper_bell	Healthy	healthy	No pesticide needed
3	Farmer complains: notable Dark spots sometimes...	Potato	Unhealthy	Early_blight	Bacillus subtilis
4	Field observation shows: slight Plant stands t...	Tomato	Healthy	healthy	No pesticide needed

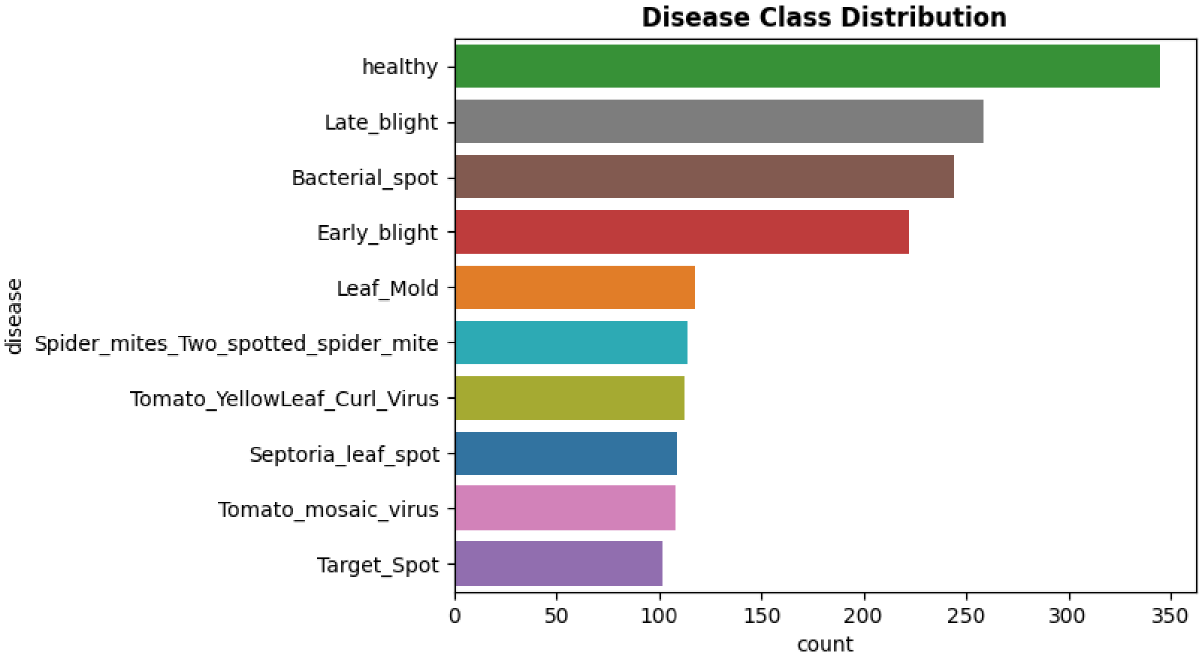
Description Data Analysis

In [63]:

```
# Count of healthy and Unhealthy status
sns.countplot(x = 'status', data = des_df, hue = 'status', palette = 'Set2')
plt.title('Status Count Healthy VS Unhealthy', weight = 'bold')
plt.show()
```

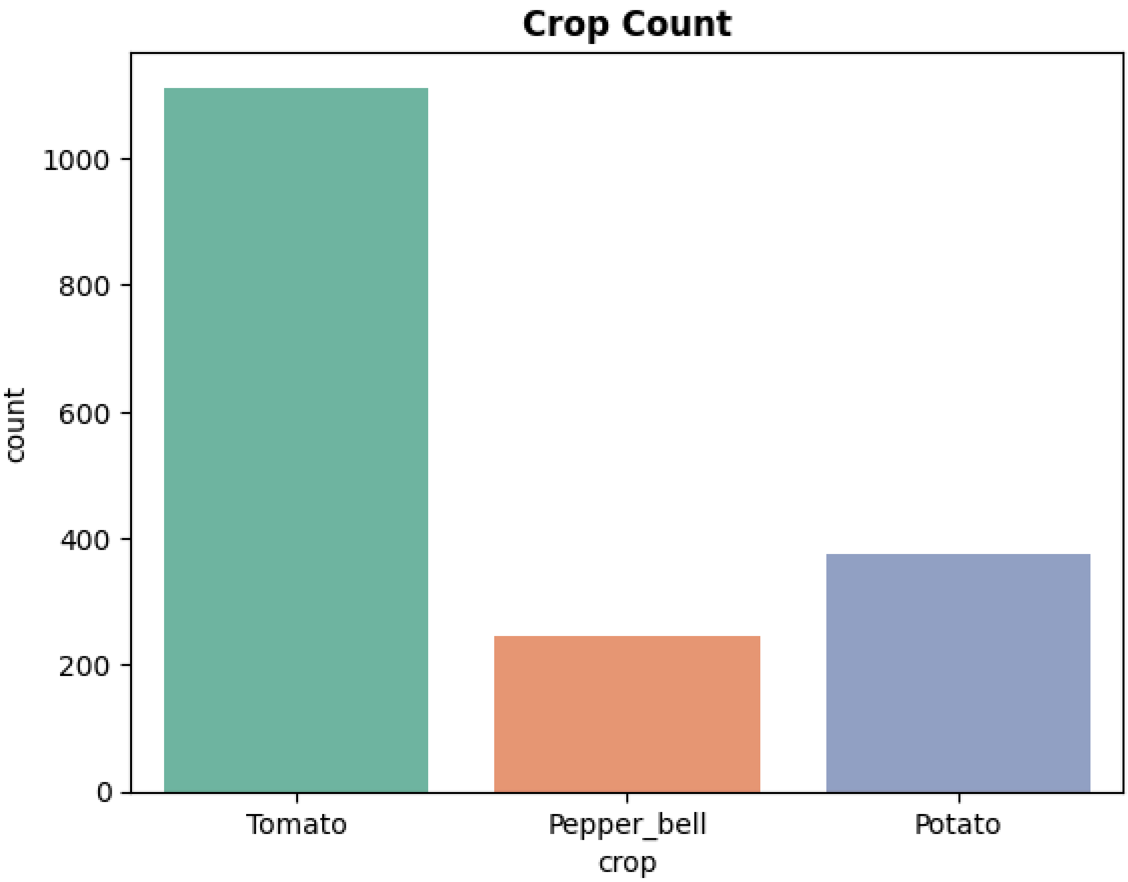


```
In [64]: # Dicease Count
         sns.countplot(y='disease', data=des_df, order=des_df['disease'].value_counts
         plt.title('Disease Class Distribution', weight = 'bold')
         plt.show()
```



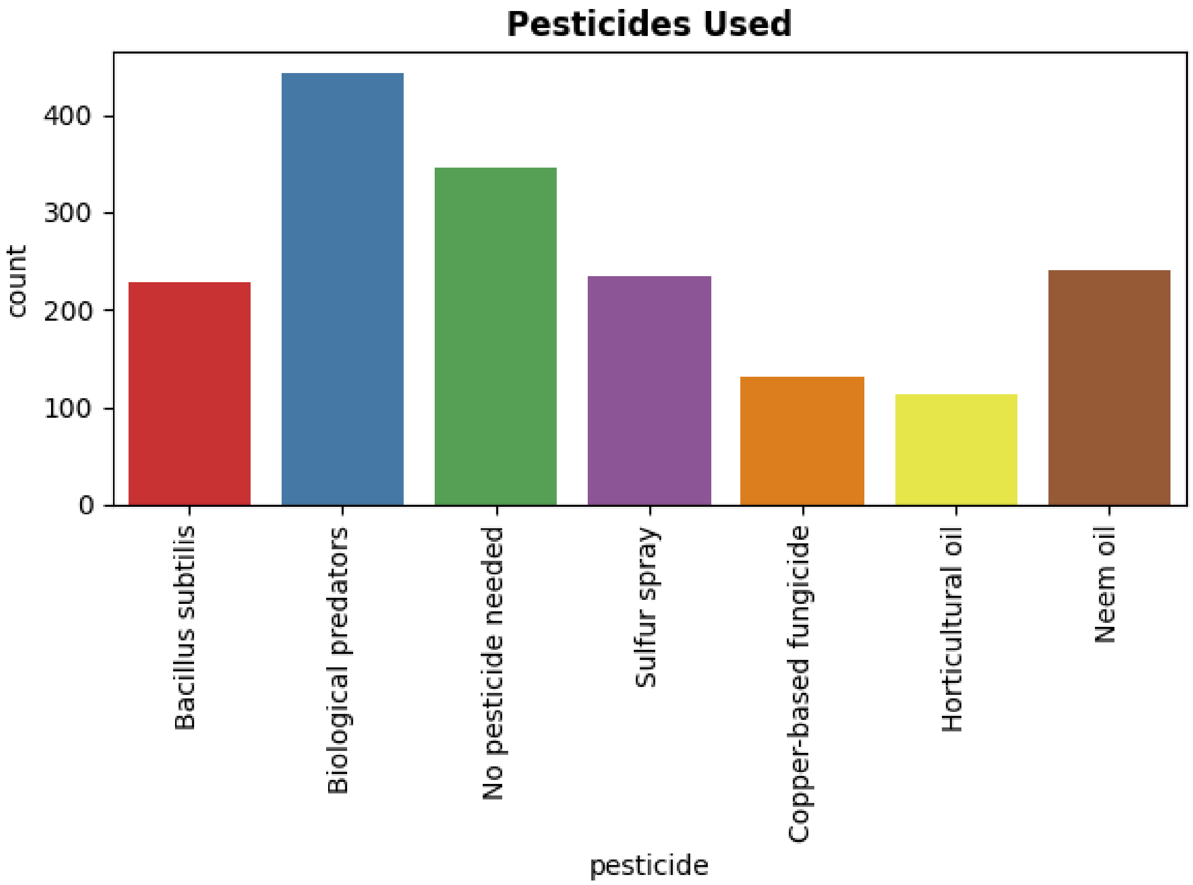
```
In [65]: # Crop count
         sns.countplot(x = 'crop', data = des_df, hue = 'crop', palette = 'Set2')
```

```
plt.title('Crop Count', weight = 'bold')
plt.show()
```



In [66]:

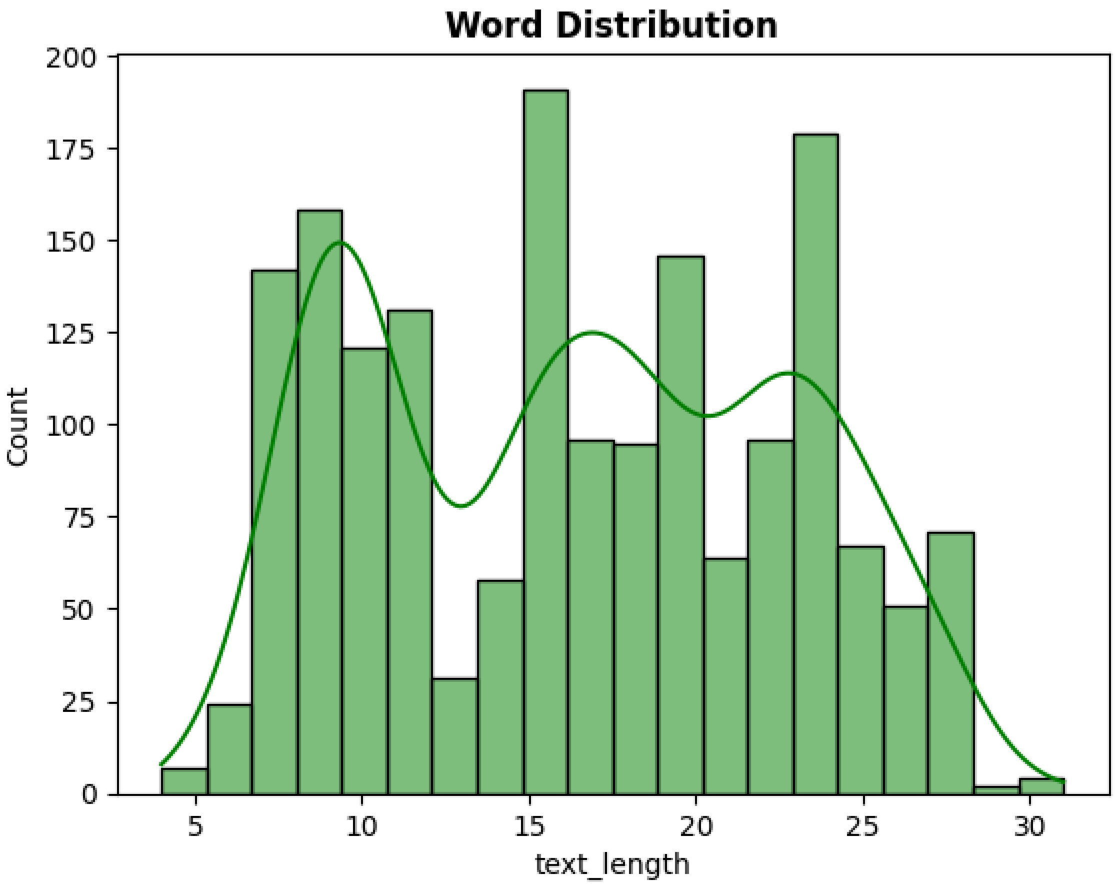
```
# Pesticide Counts
sns.countplot(x = 'pesticide', data = des_df, hue = 'pesticide', palette = '
plt.title('Pesticides Used', weight = 'bold')
plt.xticks(rotation = 90)
plt.tight_layout()
plt.show()
```



Text Analysis

In [68]:

```
# Histogram of Words
des_df.loc[:, 'text_length'] = des_df['description'].apply(lambda x: len(x.s

sns.histplot(des_df['text_length'], bins = 20, kde = True, color = 'green')
plt.title('Word Distribution', weight = 'bold')
plt.show()
```



In [69]:

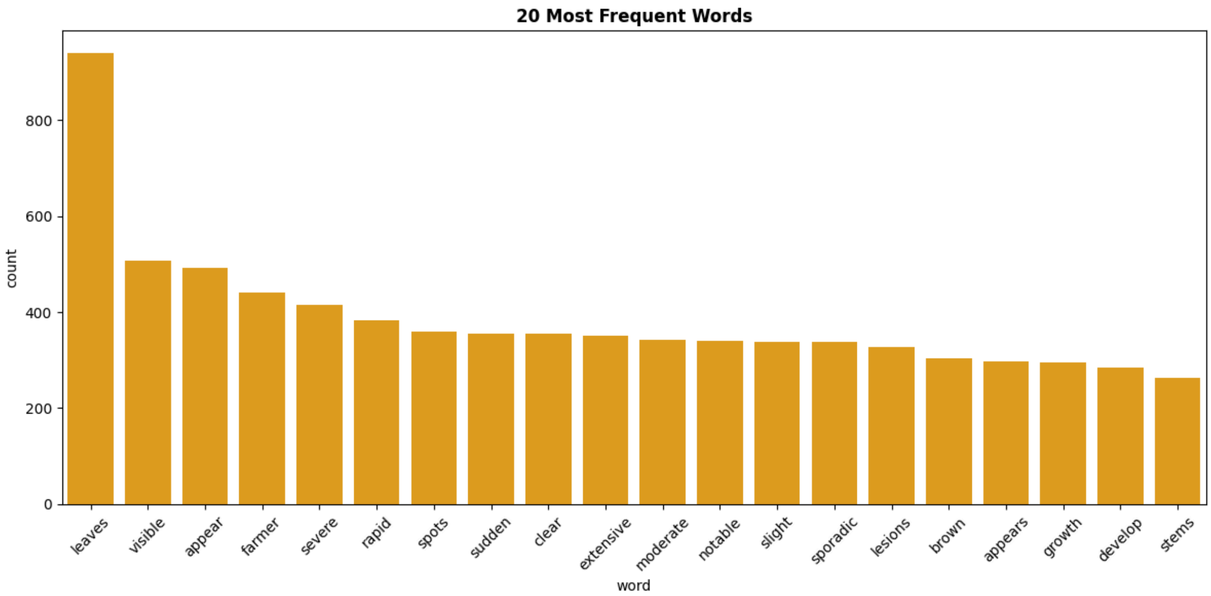
```
import spacy

# Load tokenizer
nlp = spacy.load("en_core_web_sm")

text = "This is a test sentence."
tokens = [token.text for token in nlp(text)]
print(tokens)
```

```
['This', 'is', 'a', 'test', 'sentence', '.']
```

In [70]:

```
# Most Frequent Words
from collections import Counter

all_words = []

for desc in des_df['description']:
    doc = nlp(desc.lower())
    tokens = [token.text for token in doc if token.is_alpha and not token.is
    all_words.extend(tokens)

word_freq = Counter(all_words)
top_20_words = word_freq.most_common(20)

top_words_df = pd.DataFrame(top_20_words, columns=['word', 'count'])

# 5. Plotting the result
plt.figure(figsize=(12,6))
```

```
sns.barplot(data=top_words_df, x='word', y='count', color='orange')
plt.title('20 Most Frequent Words', weight='bold')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Feature Engineering

Since our target is the original 15 classes we need to make a new column with those classe with everything falling to where it is supposed to be.

For this we will need to combine our crop and disease column

In [73]:
```
des_df['target_class'] = des_df['crop'] + '___' + des_df['disease']
des_df['target_class'].head()
```

Out[73]:
```
0    Tomato___Septoria_leaf_spot
1             Tomato___Leaf_Mold
2          Pepper_bell___healthy
3          Potato___Early_blight
4               Tomato___healthy
Name: target_class, dtype: object
```

In [74]:
```
des_df.head()
```

Out[74]:

	description	crop	status	disease	pesticide	text_length
0	Agronomist notes: extensive Stems remain large...	Tomato	Unhealthy	Septoria_leaf_spot	Bacillus subtilis	20
1	Agronomist notes: notable Yellow patches appea...	Tomato	Unhealthy	Leaf_Mold	Biological predators	10
2	Farmer complains: rapid Overall appearance is ...	Pepper_bell	Healthy	healthy	No pesticide needed	24
3	Farmer complains: notable Dark spots sometimes...	Potato	Unhealthy	Early_blight	Bacillus subtilis	9
4	Field observation shows: slight Plant stands t...	Tomato	Healthy	healthy	No pesticide needed	21

In [75]:

```
print(f'Null Count: {des_df.isna().sum().sum()}')

print(f'Duplicate Count: {des_df.duplicated().sum()}')
```

```
Null Count: 0
Duplicate Count: 0
```

Text Preprocessing

In [77]:

```
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\PINCHEZZ\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\PINCHEZZ\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\PINCHEZZ\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[77]:

```
True
```

In [78]:

```
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

# Text cleaner function
def clean_text_spacy(text):
    # Lowercase and remove non-letter characters
    text = re.sub(r'[^a-zA-Z\s]', '', text.lower())

    # Process the text
    doc = nlp(text)

    # Lemmatize, remove stop words and non-alphabetic tokens
    tokens = [token.lemma_ for token in doc if token.is_alpha and not token.

    return " ".join(tokens)
```

In [79]:

```
# Applying our function to our DF
des_df['clean_description'] = des_df['description'].apply(clean_text_spacy)
```

In [80]:

```
des_df.head()
```

Out[80]:

	description	crop	status	disease	pesticide	text_length
0	Agronomist notes: extensive Stems remain large...	Tomato	Unhealthy	Septoria_leaf_spot	Bacillus subtilis	20
1	Agronomist notes: notable Yellow patches appea...	Tomato	Unhealthy	Leaf_Mold	Biological predators	10
2	Farmer complains: rapid Overall appearance is ...	Pepper_bell	Healthy	healthy	No pesticide needed	24
3	Farmer complains: notable Dark spots sometimes...	Potato	Unhealthy	Early_blight	Bacillus subtilis	9
4	Field observation shows: slight Plant stands t...	Tomato	Healthy	healthy	No pesticide needed	21

In [81]:

```python
# Repositioning our target class to be last in our data frame
des_df = des_df[[col for col in des_df.columns if col != 'target_class'] + [

#Checking
des_df.head()
```

Out[81]:

	description	crop	status	disease	pesticide	text_length
0	Agronomist notes: extensive Stems remain large...	Tomato	Unhealthy	Septoria_leaf_spot	Bacillus subtilis	20
1	Agronomist notes: notable Yellow patches appea...	Tomato	Unhealthy	Leaf_Mold	Biological predators	10
2	Farmer complains: rapid Overall appearance is ...	Pepper_bell	Healthy	healthy	No pesticide needed	24
3	Farmer complains: notable Dark spots sometimes...	Potato	Unhealthy	Early_blight	Bacillus subtilis	9
4	Field observation shows: slight Plant stands t...	Tomato	Healthy	healthy	No pesticide needed	21

TF-IDF Vectorization

In [83]:

```python
# Initialize vectorizer
tfidf = TfidfVectorizer(max_features=2000, ngram_range=(1,2))

#Fit with clean_description
x_tfidf = tfidf.fit_transform(des_df['clean_description'])

x_tfidf.shape
```

Out[83]:

```
(1734, 2000)
```

Label Encoding

In [85]:

```python
le = LabelEncoder()
y_encoded = le.fit_transform(des_df['target_class'])
```

```
# Check for correct mapping
class_mapping = dict(zip(le.classes_, le.transform(le.classes_)))
print(class_mapping)
```

```
{'Pepper_bell___Bacterial_spot': 0, 'Pepper_bell___healthy': 1, 'Potato___Ea
rly_blight': 2, 'Potato___Late_blight': 3, 'Potato___healthy': 4, 'Tomato___
Bacterial_spot': 5, 'Tomato___Early_blight': 6, 'Tomato___Late_blight': 7,
'Tomato___Leaf_Mold': 8, 'Tomato___Septoria_leaf_spot': 9, 'Tomato___Spider_
mites_Two_spotted_spider_mite': 10, 'Tomato___Target_Spot': 11, 'Tomato___To
mato_YellowLeaf_Curl_Virus': 12, 'Tomato___Tomato_mosaic_virus': 13, 'Tomato
___healthy': 14}
```

Train/Test Split

In [87]:

```
# Spliting our data into train test and split for our model
X_train, X_test, y_train, y_test = train_test_split(x_tfidf, y_encoded, test
```

Modeling

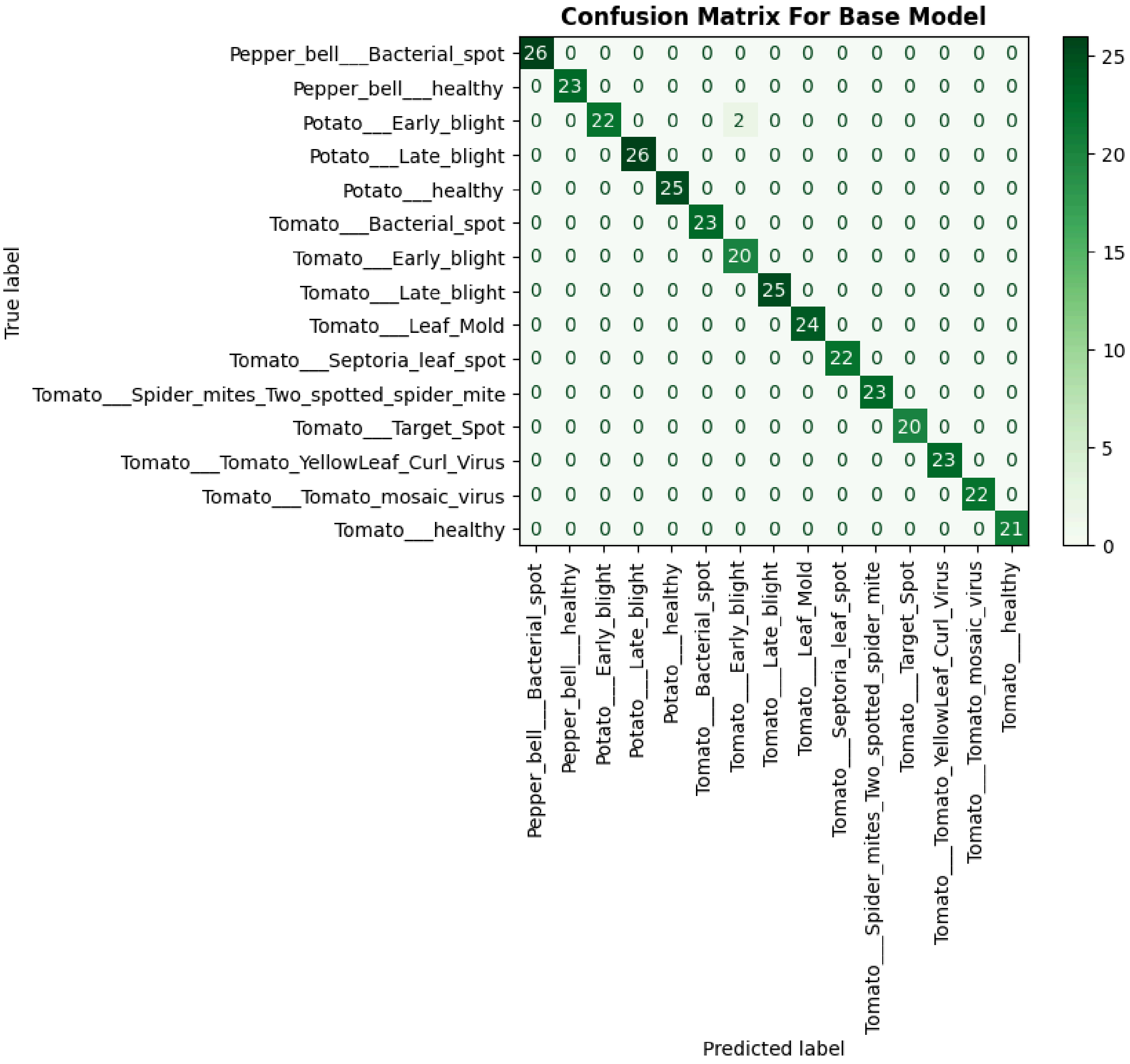
Logistic Base Model

In [90]:

```
# Build and fit thet model
model_0 = LogisticRegression(max_iter=1000)
model_0.fit(X_train, y_train)

# Make predictions
y_preds = model_0.predict(X_test)
y_probs = model_0.predict_proba(X_test)
```

In [91]:

```
# Evaluate model
from sklearn.metrics import classification_report, confusion_matrix, accurac
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc

cm = confusion_matrix(y_test, y_preds)
disp = ConfusionMatrixDisplay(confusion_matrix = cm, display_labels=le.class
disp.plot(cmap = 'Greens')
plt.title('Confusion Matrix For Base Model', weight = 'bold')
plt.xticks(rotation = 90)
plt.show()

print(classification_report(y_test, y_preds, target_names=le.classes_))
acc = accuracy_score(y_test, y_preds)
print(f"\nAccuracy: {acc:.4f}")
```



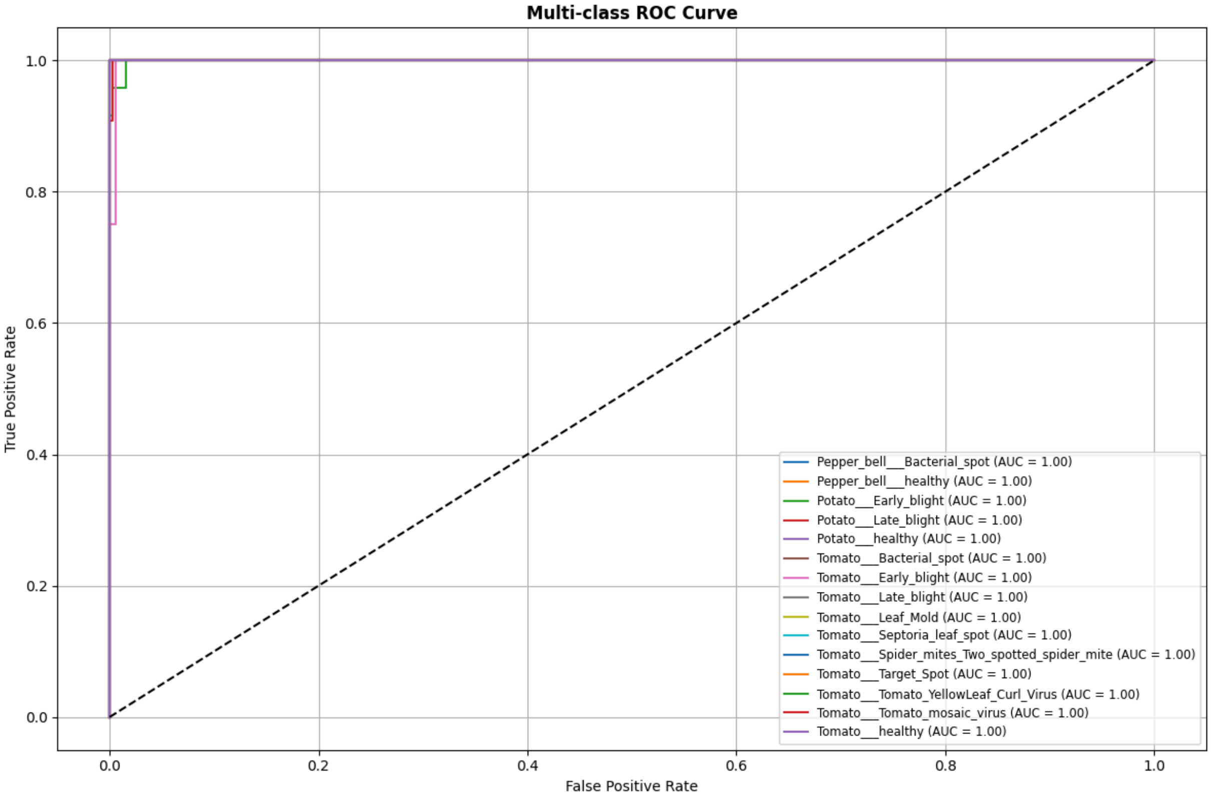
Confusion Matrix For Base Model

```
                                               precision    recall  f1-score
support

                 Pepper_bell___Bacterial_spot       1.00      1.00      1.00
26
                        Pepper_bell___healthy       1.00      1.00      1.00
23
                        Potato___Early_blight       1.00      0.92      0.96
24
                         Potato___Late_blight       1.00      1.00      1.00
26
                             Potato___healthy       1.00      1.00      1.00
25
                      Tomato___Bacterial_spot       1.00      1.00      1.00
23
                        Tomato___Early_blight       0.91      1.00      0.95
20
                         Tomato___Late_blight       1.00      1.00      1.00
25
                           Tomato___Leaf_Mold       1.00      1.00      1.00
24
                  Tomato___Septoria_leaf_spot       1.00      1.00      1.00
22
Tomato___Spider_mites_Two_spotted_spider_mite       1.00      1.00      1.00
23
                         Tomato___Target_Spot       1.00      1.00      1.00
20
        Tomato___Tomato_YellowLeaf_Curl_Virus       1.00      1.00      1.00
23
                 Tomato___Tomato_mosaic_virus       1.00      1.00      1.00
22
                             Tomato___healthy       1.00      1.00      1.00
21

                                     accuracy                           0.99
347
                                    macro avg       0.99      0.99      0.99
347
                                 weighted avg       0.99      0.99      0.99
347


Accuracy: 0.9942
```

In [92]:

```
y_test_bin = label_binarize(y_test, classes=list(range(len(le.classes_)))) #

fpr = {}
tpr = {}
roc_auc = {}

# Loop through each class
for i in range(len(le.classes_)):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_probs[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot
```

```
plt.figure(figsize=(12, 8))

for i in range(len(le.classes_)):
    plt.plot(fpr[i], tpr[i], label=f'{le.classes_[i]} (AUC = {roc_auc[i]:.2f

plt.plot([0, 1], [0, 1], 'k--')  # diagonal line
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Multi-class ROC Curve', weight='bold')
plt.legend(loc='lower right', fontsize='small')
plt.grid(True)
plt.tight_layout()
plt.show()
```



Multinomial Naive Bayes Model

In [94]:

```
#Library importation
from sklearn.pipeline import Pipeline
from sklearn.naive_bayes import MultinomialNB
from sklearn.base import BaseEstimator, TransformerMixin
import spacy
import joblib

nlp = spacy.load('en_core_web_sm')
```

In [95]:

```
# Train_Test_split our data
X_train, X_test, y_train, y_test = train_test_split(
    des_df["clean_description"],
    des_df["target_class"],
    test_size=0.2,
    random_state=42,
    stratify=des_df["target_class"]
)
```

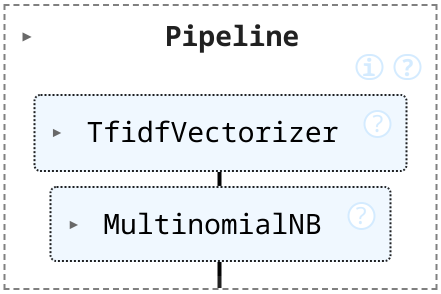
In [96]:

```
# Initialize the pipelie
nb_pipeline = Pipeline([
    ('tfidf', TfidfVectorizer(
        max_features=2000,
        ngram_range=(1, 2)
    )),
    ('clf', MultinomialNB())
])
```

In [97]:

```
# Fit the pipeline on training data
nb_pipeline.fit(X_train, y_train)
```

Out[97]:



Predictions and Evaluations

In [99]:

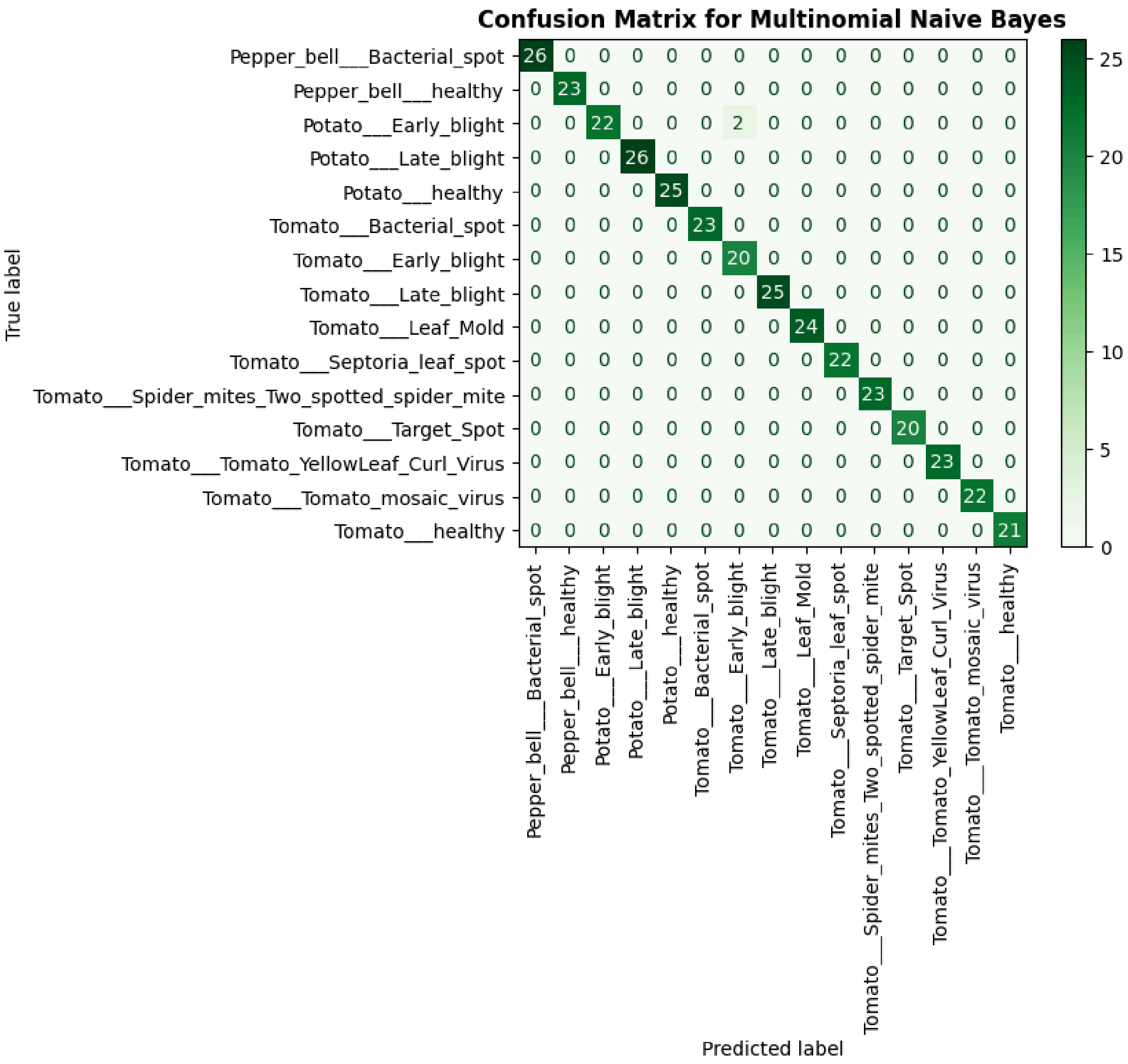
```
# Predict on the test set
y_pred = nb_pipeline.predict(X_test)

# Evaluate
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.9942363112391931

Classification Report:

                                              precision    recall  f1-score
support

                 Pepper_bell___Bacterial_spot      1.00      1.00      1.00
26
                        Pepper_bell___healthy      1.00      1.00      1.00
23
                        Potato___Early_blight      1.00      0.92      0.96
24
                         Potato___Late_blight      1.00      1.00      1.00
26
                             Potato___healthy      1.00      1.00      1.00
25
                      Tomato___Bacterial_spot      1.00      1.00      1.00
23
                        Tomato___Early_blight      0.91      1.00      0.95
20
                         Tomato___Late_blight      1.00      1.00      1.00
25
                           Tomato___Leaf_Mold      1.00      1.00      1.00
24
                  Tomato___Septoria_leaf_spot      1.00      1.00      1.00
22
Tomato___Spider_mites_Two_spotted_spider_mite      1.00      1.00      1.00
23
                         Tomato___Target_Spot      1.00      1.00      1.00
20
        Tomato___Tomato_YellowLeaf_Curl_Virus      1.00      1.00      1.00
23
                 Tomato___Tomato_mosaic_virus      1.00      1.00      1.00
22
                             Tomato___healthy      1.00      1.00      1.00
21

                                     accuracy                          0.99
347
                                    macro avg      0.99      0.99      0.99
347
                                 weighted avg      0.99      0.99      0.99
347
```

Plot Confusion Matrix

In [101…

```
cm = confusion_matrix(y_test, y_pred, labels=nb_pipeline.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=nb_pipelir
disp.plot(xticks_rotation=90, cmap='Greens')
plt.title('Confusion Matrix for Multinomial Naive Bayes', weight='bold')
plt.show()
```



Saving Models and Usable Variables

In [103…

```
# Saving CNN Model
model.save("cnn_model.keras")

#Saving NLP Model
joblib.dump(nb_pipeline, 'mnb_nlp_pipeline.pkl')
```

Out[103…

```
['mnb_nlp_pipeline.pkl']
```

In [ ]:

This notebook was converted with convert.ploomber.io