[2]:
```
import sys
print(sys.version)
```

```
3.10.12 (main, Nov  6 2024, 20:22:13) [GCC 11.4.0]
```

[3]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[4]:
```
import os
os.chdir('/content/drive/My Drive/proje')
!pwd
```

```
/content/drive/My Drive/proje
```

[5]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization␣
↪,Activation
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from tensorflow.keras.optimizers import Adam
import plotly.express as px
```

[6]:
```
data = pd.read_csv('/content/drive/MyDrive/proje/database.csv',␣
↪encoding='latin-1')
```

[8]:
```
data.head(200)
```

[8]:

	N	P	K	temperature	humidity	ph	rainfall	label
0	90	42	43	20.879744	82.002744	6.502985	202.935536	pirinç

```
1    85  58  41    21.770462  80.319644  7.038096  226.655537  pirinç
2    60  55  44    23.004459  82.320763  7.840207  263.964248  pirinç
3    74  35  40    26.491096  80.158363  6.980401  242.864034  pirinç
4    78  42  42    20.130175  81.604873  7.628473  262.717340  pirinç
..   ..  ..  ..          …          …        …          …       …
195  90  57  24    18.928519  72.800861  6.158860   82.341629   mýsýr
196  67  35  22    23.305468  63.246480  6.385684  108.760300   mýsýr
197  60  54  19    18.748267  62.498785  6.417820   70.234016   mýsýr
198  83  58  23    19.742133  59.662631  6.381202   65.508614   mýsýr
199  83  57  19    25.730444  70.747393  6.877869   98.737713   mýsýr

[200 rows x 8 columns]
```

[ ]:
```
data.columns
```

[ ]:
```
Index(['N', 'P', 'K', 'temperature', 'humidity', 'ph', 'rainfall', 'label'],
dtype='object')
```

[ ]:
```
data.isnull().any()
```

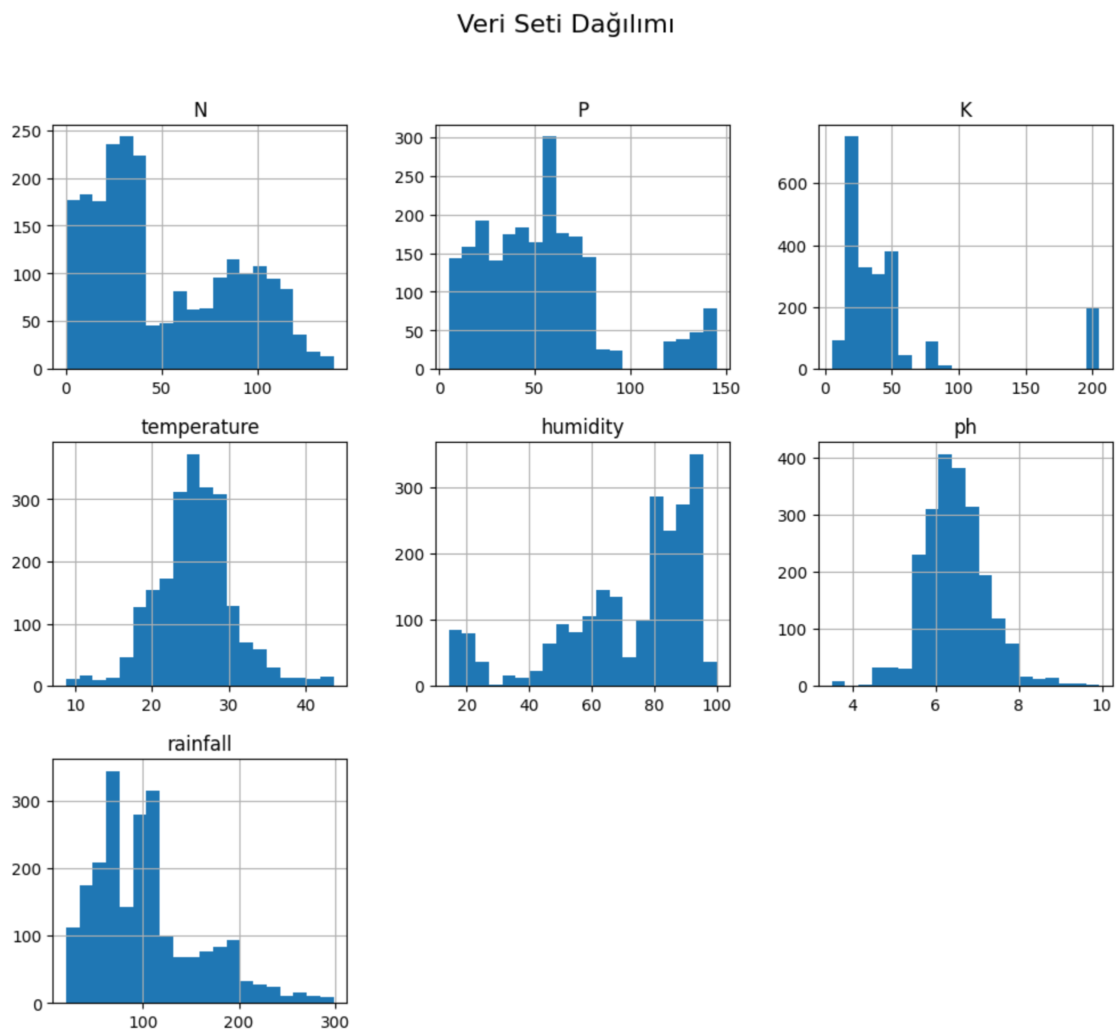
[ ]:
```
N              False
P              False
K              False
temperature    False
humidity       False
ph             False
rainfall       False
label          False
dtype: bool
```

[ ]:
```
data['label'].value_counts()
```

[ ]:
```
label
pirinç                100
mýsýr                 100
jute                  100
pamuk                 100
kokonat               100
papaya                100
portakal              100
elma                  100
kavun                 100
karpuz                100
üzüm                  100
mango                 100
muz                   100
nar                   100
```

```
mercimek              100
blackgram             100
maþ fasulyesi         100
güve fasulyesi        100
güvercin bezelyesi    100
barbunya              100
nohut                 100
kahve                 100
Name: count, dtype: int64
```

[ ]:
```python
data.hist(bins=20, figsize=(12, 10))
plt.suptitle("Veri Seti Dağılımı", fontsize=16)
plt.show()
```

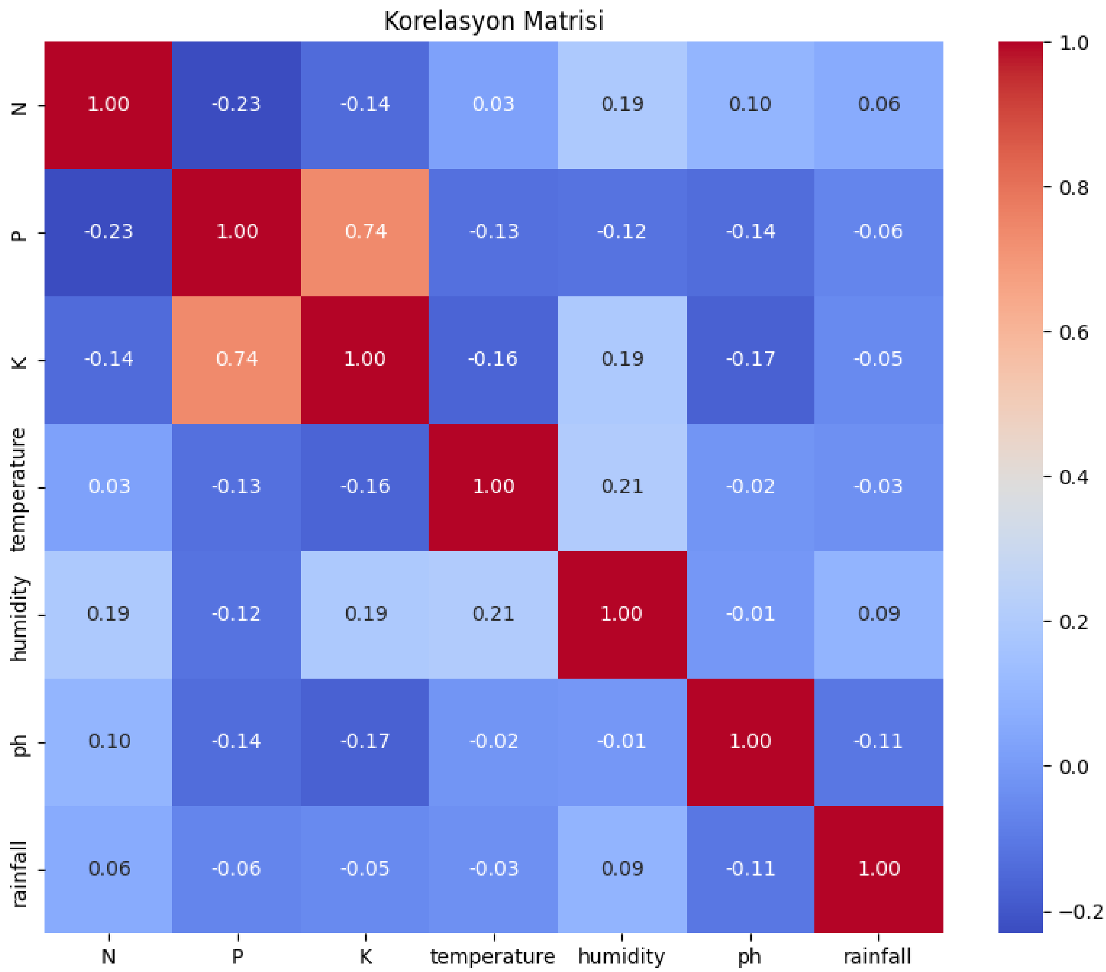


[ ]:
```
# Belirli etiketlere göre veriyi filtrele
crop_scatter = data[(data['label'] == 'pirinç') |
                    (data['label'] == 'blackgram') |
                    (data['label'] == 'pamuk') |
                    (data['label'] == 'kahve') |
                    (data['label'] == 'mercimek')]

# Scatter plot oluştur
fig = px.scatter(
    crop_scatter,
    x="temperature",
    y="humidity",
    color="label",
    symbol="label",
    title="Sıcaklık ve Nem Dağılımı (Ürün Etiketlerine Göre)"
)
fig.update_layout(plot_bgcolor='white')
fig.update_xaxes(showgrid=False, title="Sıcaklık (°C)")
fig.update_yaxes(showgrid=False, title="Nem (%)")

# Grafiği göster
fig.show()
```

[ ]:
```
numeric_columns = data.select_dtypes(include=[np.number])
plt.figure(figsize=(10, 8))
sns.heatmap(numeric_columns.corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Korelasyon Matrisi")
plt.show()
```



[ ]:
```
print("Veri türleri:\n", data.dtypes)
```

```
Veri türleri:
 N                  int64
P                  int64
K                  int64
temperature      float64
humidity         float64
ph               float64
rainfall         float64
label             object
dtype: object
```

[ ]:
```
crop_summary = pd.pivot_table(data,index=['label'],aggfunc='mean')
crop_summary.head()
```

```
[ ]:                        K      N       P   humidity        ph    rainfall  \
     label
     barbunya           20.05  20.75   67.54  21.605357  5.749411  105.919778
     blackgram          19.24  40.02   67.47  65.118426  7.133952   67.884151
     elma              199.89  20.80  134.22  92.333383  5.929663  112.654779
     güve fasulyesi     20.23  21.44   48.01  53.160418  6.831174   51.198487
     güvercin bezelyesi  20.29  20.73   67.73  48.061633  5.794175  149.457564

                        temperature
     label
     barbunya             20.115085
     blackgram            29.973340
     elma                 22.630942
     güve fasulyesi       28.194920
     güvercin bezelyesi   27.741762
```

```
[ ]: # değişkenleriöiz
     X = data[['N', 'P', 'K', 'temperature', 'humidity', 'ph', 'rainfall']]
     y = data['label']
```

```
[ ]: # Etiketleri Encode Etme sayıla çevirme
     label_encoder = LabelEncoder()
     y_encoded = label_encoder.fit_transform(y)
     y_one_hot = to_categorical(y_encoded)  # one-hot matrixine dönüştürerek ysa␣
      ↪için  uygun hale getirme işlemi
```

```
[ ]: # Veriyi Eğitim ve Test Setlerine Ayırma
     X_train, X_test, y_train, y_test = train_test_split(X, y_one_hot, test_size=0.
      ↪2, random_state=42, stratify=y_encoded)
```

```
[ ]: # Veriyi Normalizasyon (Scaling)
     scaler = StandardScaler() #özellikleri normalize ederek veriyi standartlaştırır.
     X_train = scaler.fit_transform(X_train)
     X_test = scaler.transform(X_test)
```

```
[ ]: # Ysa
     model = Sequential()
     model.add(Dense(256, input_dim=X_train.shape[1], activation='relu'))
     model.add(BatchNormalization())
     model.add(Dropout(0.4))
     model.add(Dense(128, activation='relu'))
     model.add(BatchNormalization())
     model.add(Dropout(0.4))
     model.add(Dense(64, activation='relu'))
     model.add(BatchNormalization())
     model.add(Dropout(0.3))
     model.add(Dense(y_one_hot.shape[1], activation='softmax'))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87:
UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using
Sequential models, prefer using an `Input(shape)` object as the first layer in
the model instead.
```

[ ]:
```
# Modeli Derleme

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

[ ]:
```
# Erken Durdurma Tanımlama
early_stopping = EarlyStopping(monitor='val_loss', patience=10,␣
↪restore_best_weights=True)
```

[ ]:
```
# Modeli Eğitme
history = model.fit(X_train, y_train, epochs=100, batch_size=64,␣
↪validation_split=0.2, verbose=1, callbacks=[early_stopping])
```

```
Epoch 1/100
22/22           3s 24ms/step -
accuracy: 0.1095 - loss: 3.4457 - val_accuracy: 0.5142 - val_loss: 2.7783
Epoch 2/100
22/22           0s 10ms/step -
accuracy: 0.4496 - loss: 1.9224 - val_accuracy: 0.5710 - val_loss: 2.5198
Epoch 3/100
22/22           0s 9ms/step -
accuracy: 0.6242 - loss: 1.3373 - val_accuracy: 0.5682 - val_loss: 2.3130
Epoch 4/100
22/22           0s 10ms/step -
accuracy: 0.7268 - loss: 1.0094 - val_accuracy: 0.5199 - val_loss: 2.1281
Epoch 5/100
22/22           0s 10ms/step -
accuracy: 0.7514 - loss: 0.8457 - val_accuracy: 0.5909 - val_loss: 1.9292
Epoch 6/100
22/22           0s 12ms/step -
accuracy: 0.8077 - loss: 0.7018 - val_accuracy: 0.6562 - val_loss: 1.7134
Epoch 7/100
22/22           0s 10ms/step -
accuracy: 0.8340 - loss: 0.6188 - val_accuracy: 0.7188 - val_loss: 1.5113
Epoch 8/100
22/22           0s 10ms/step -
accuracy: 0.8343 - loss: 0.5669 - val_accuracy: 0.7386 - val_loss: 1.3125
Epoch 9/100
22/22           0s 10ms/step -
accuracy: 0.8725 - loss: 0.4595 - val_accuracy: 0.7841 - val_loss: 1.1280
```

```
Epoch 10/100
22/22            0s 9ms/step -
accuracy: 0.9082 - loss: 0.4058 - val_accuracy: 0.8210 - val_loss: 0.9352
Epoch 11/100
22/22            0s 12ms/step -
accuracy: 0.8984 - loss: 0.3842 - val_accuracy: 0.8466 - val_loss: 0.7875
Epoch 12/100
22/22            0s 10ms/step -
accuracy: 0.8862 - loss: 0.4031 - val_accuracy: 0.8693 - val_loss: 0.6382
Epoch 13/100
22/22            0s 10ms/step -
accuracy: 0.9085 - loss: 0.3489 - val_accuracy: 0.8835 - val_loss: 0.4993
Epoch 14/100
22/22            0s 9ms/step -
accuracy: 0.9328 - loss: 0.2921 - val_accuracy: 0.9091 - val_loss: 0.3948
Epoch 15/100
22/22            0s 7ms/step -
accuracy: 0.9355 - loss: 0.2771 - val_accuracy: 0.9261 - val_loss: 0.3156
Epoch 16/100
22/22            0s 6ms/step -
accuracy: 0.9290 - loss: 0.2704 - val_accuracy: 0.9318 - val_loss: 0.2636
Epoch 17/100
22/22            0s 6ms/step -
accuracy: 0.9367 - loss: 0.2516 - val_accuracy: 0.9403 - val_loss: 0.2222
Epoch 18/100
22/22            0s 6ms/step -
accuracy: 0.9394 - loss: 0.2308 - val_accuracy: 0.9318 - val_loss: 0.2029
Epoch 19/100
22/22            0s 6ms/step -
accuracy: 0.9329 - loss: 0.2320 - val_accuracy: 0.9403 - val_loss: 0.1783
Epoch 20/100
22/22            0s 6ms/step -
accuracy: 0.9346 - loss: 0.2224 - val_accuracy: 0.9688 - val_loss: 0.1280
Epoch 21/100
22/22            0s 6ms/step -
accuracy: 0.9481 - loss: 0.1948 - val_accuracy: 0.9631 - val_loss: 0.1207
Epoch 22/100
22/22            0s 6ms/step -
accuracy: 0.9397 - loss: 0.2014 - val_accuracy: 0.9688 - val_loss: 0.1065
Epoch 23/100
22/22            0s 7ms/step -
accuracy: 0.9556 - loss: 0.1556 - val_accuracy: 0.9688 - val_loss: 0.0976
Epoch 24/100
22/22            0s 6ms/step -
accuracy: 0.9594 - loss: 0.1590 - val_accuracy: 0.9716 - val_loss: 0.0860
Epoch 25/100
22/22            0s 5ms/step -
accuracy: 0.9620 - loss: 0.1466 - val_accuracy: 0.9688 - val_loss: 0.0903
```

```
Epoch 26/100
22/22            0s 6ms/step -
accuracy: 0.9684 - loss: 0.1410 - val_accuracy: 0.9773 - val_loss: 0.0713
Epoch 27/100
22/22            0s 5ms/step -
accuracy: 0.9525 - loss: 0.1585 - val_accuracy: 0.9773 - val_loss: 0.0754
Epoch 28/100
22/22            0s 6ms/step -
accuracy: 0.9524 - loss: 0.1516 - val_accuracy: 0.9688 - val_loss: 0.0678
Epoch 29/100
22/22            0s 6ms/step -
accuracy: 0.9701 - loss: 0.1266 - val_accuracy: 0.9716 - val_loss: 0.0651
Epoch 30/100
22/22            0s 5ms/step -
accuracy: 0.9636 - loss: 0.1347 - val_accuracy: 0.9688 - val_loss: 0.0756
Epoch 31/100
22/22            0s 6ms/step -
accuracy: 0.9700 - loss: 0.1173 - val_accuracy: 0.9716 - val_loss: 0.0630
Epoch 32/100
22/22            0s 5ms/step -
accuracy: 0.9677 - loss: 0.1036 - val_accuracy: 0.9716 - val_loss: 0.0701
Epoch 33/100
22/22            0s 7ms/step -
accuracy: 0.9668 - loss: 0.1254 - val_accuracy: 0.9830 - val_loss: 0.0541
Epoch 34/100
22/22            0s 5ms/step -
accuracy: 0.9719 - loss: 0.1246 - val_accuracy: 0.9773 - val_loss: 0.0573
Epoch 35/100
22/22            0s 6ms/step -
accuracy: 0.9635 - loss: 0.1383 - val_accuracy: 0.9744 - val_loss: 0.0666
Epoch 36/100
22/22            0s 5ms/step -
accuracy: 0.9778 - loss: 0.0946 - val_accuracy: 0.9716 - val_loss: 0.0589
Epoch 37/100
22/22            0s 6ms/step -
accuracy: 0.9784 - loss: 0.0936 - val_accuracy: 0.9716 - val_loss: 0.0617
Epoch 38/100
22/22            0s 6ms/step -
accuracy: 0.9695 - loss: 0.0939 - val_accuracy: 0.9801 - val_loss: 0.0491
Epoch 39/100
22/22            0s 5ms/step -
accuracy: 0.9698 - loss: 0.1095 - val_accuracy: 0.9744 - val_loss: 0.0570
Epoch 40/100
22/22            0s 6ms/step -
accuracy: 0.9631 - loss: 0.1114 - val_accuracy: 0.9716 - val_loss: 0.0668
Epoch 41/100
22/22            0s 6ms/step -
accuracy: 0.9653 - loss: 0.1104 - val_accuracy: 0.9716 - val_loss: 0.0594
```

```
Epoch 42/100
22/22            0s 6ms/step -
accuracy: 0.9726 - loss: 0.0844 - val_accuracy: 0.9801 - val_loss: 0.0416
Epoch 43/100
22/22            0s 5ms/step -
accuracy: 0.9797 - loss: 0.0674 - val_accuracy: 0.9688 - val_loss: 0.0627
Epoch 44/100
22/22            0s 7ms/step -
accuracy: 0.9732 - loss: 0.0938 - val_accuracy: 0.9773 - val_loss: 0.0489
Epoch 45/100
22/22            0s 6ms/step -
accuracy: 0.9788 - loss: 0.0953 - val_accuracy: 0.9716 - val_loss: 0.0542
Epoch 46/100
22/22            0s 6ms/step -
accuracy: 0.9809 - loss: 0.0852 - val_accuracy: 0.9830 - val_loss: 0.0430
Epoch 47/100
22/22            0s 6ms/step -
accuracy: 0.9756 - loss: 0.1078 - val_accuracy: 0.9716 - val_loss: 0.0561
Epoch 48/100
22/22            0s 6ms/step -
accuracy: 0.9714 - loss: 0.0973 - val_accuracy: 0.9773 - val_loss: 0.0456
Epoch 49/100
22/22            0s 6ms/step -
accuracy: 0.9771 - loss: 0.0823 - val_accuracy: 0.9801 - val_loss: 0.0432
Epoch 50/100
22/22            0s 5ms/step -
accuracy: 0.9740 - loss: 0.0825 - val_accuracy: 0.9858 - val_loss: 0.0381
Epoch 51/100
22/22            0s 6ms/step -
accuracy: 0.9737 - loss: 0.0814 - val_accuracy: 0.9773 - val_loss: 0.0482
Epoch 52/100
22/22            0s 5ms/step -
accuracy: 0.9781 - loss: 0.0688 - val_accuracy: 0.9773 - val_loss: 0.0450
Epoch 53/100
22/22            0s 6ms/step -
accuracy: 0.9777 - loss: 0.0724 - val_accuracy: 0.9830 - val_loss: 0.0404
Epoch 54/100
22/22            0s 7ms/step -
accuracy: 0.9756 - loss: 0.0735 - val_accuracy: 0.9801 - val_loss: 0.0499
Epoch 55/100
22/22            0s 6ms/step -
accuracy: 0.9761 - loss: 0.0701 - val_accuracy: 0.9688 - val_loss: 0.0598
Epoch 56/100
22/22            0s 6ms/step -
accuracy: 0.9768 - loss: 0.0746 - val_accuracy: 0.9830 - val_loss: 0.0444
Epoch 57/100
22/22            0s 6ms/step -
accuracy: 0.9720 - loss: 0.0735 - val_accuracy: 0.9830 - val_loss: 0.0412
```

```
Epoch 58/100
22/22           0s 6ms/step -
accuracy: 0.9770 - loss: 0.0668 - val_accuracy: 0.9801 - val_loss: 0.0433
Epoch 59/100
22/22           0s 6ms/step -
accuracy: 0.9833 - loss: 0.0639 - val_accuracy: 0.9801 - val_loss: 0.0461
Epoch 60/100
22/22           0s 6ms/step -
accuracy: 0.9778 - loss: 0.0683 - val_accuracy: 0.9886 - val_loss: 0.0373
Epoch 61/100
22/22           0s 5ms/step -
accuracy: 0.9761 - loss: 0.0667 - val_accuracy: 0.9858 - val_loss: 0.0337
Epoch 62/100
22/22           0s 5ms/step -
accuracy: 0.9858 - loss: 0.0585 - val_accuracy: 0.9716 - val_loss: 0.0532
Epoch 63/100
22/22           0s 6ms/step -
accuracy: 0.9804 - loss: 0.0653 - val_accuracy: 0.9773 - val_loss: 0.0458
Epoch 64/100
22/22           0s 7ms/step -
accuracy: 0.9781 - loss: 0.0640 - val_accuracy: 0.9801 - val_loss: 0.0457
Epoch 65/100
22/22           0s 10ms/step -
accuracy: 0.9792 - loss: 0.0665 - val_accuracy: 0.9716 - val_loss: 0.0569
Epoch 66/100
22/22           0s 9ms/step -
accuracy: 0.9816 - loss: 0.0487 - val_accuracy: 0.9801 - val_loss: 0.0512
Epoch 67/100
22/22           0s 12ms/step -
accuracy: 0.9734 - loss: 0.0709 - val_accuracy: 0.9773 - val_loss: 0.0469
Epoch 68/100
22/22           0s 9ms/step -
accuracy: 0.9876 - loss: 0.0508 - val_accuracy: 0.9744 - val_loss: 0.0551
Epoch 69/100
22/22           0s 9ms/step -
accuracy: 0.9860 - loss: 0.0504 - val_accuracy: 0.9716 - val_loss: 0.0582
Epoch 70/100
22/22           0s 12ms/step -
accuracy: 0.9814 - loss: 0.0580 - val_accuracy: 0.9801 - val_loss: 0.0430
Epoch 71/100
22/22           0s 9ms/step -
accuracy: 0.9858 - loss: 0.0514 - val_accuracy: 0.9801 - val_loss: 0.0468
```

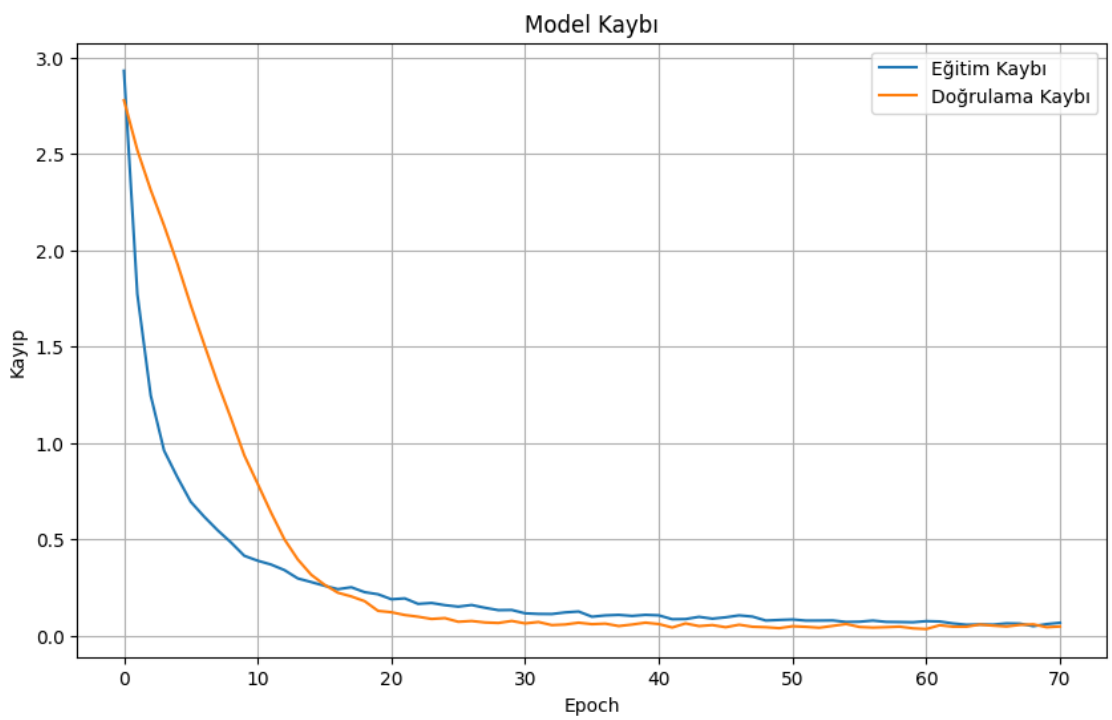
[ ]:

```
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f"Test Loss: {loss}")
print(f"Test Accuracy: {accuracy}")
```

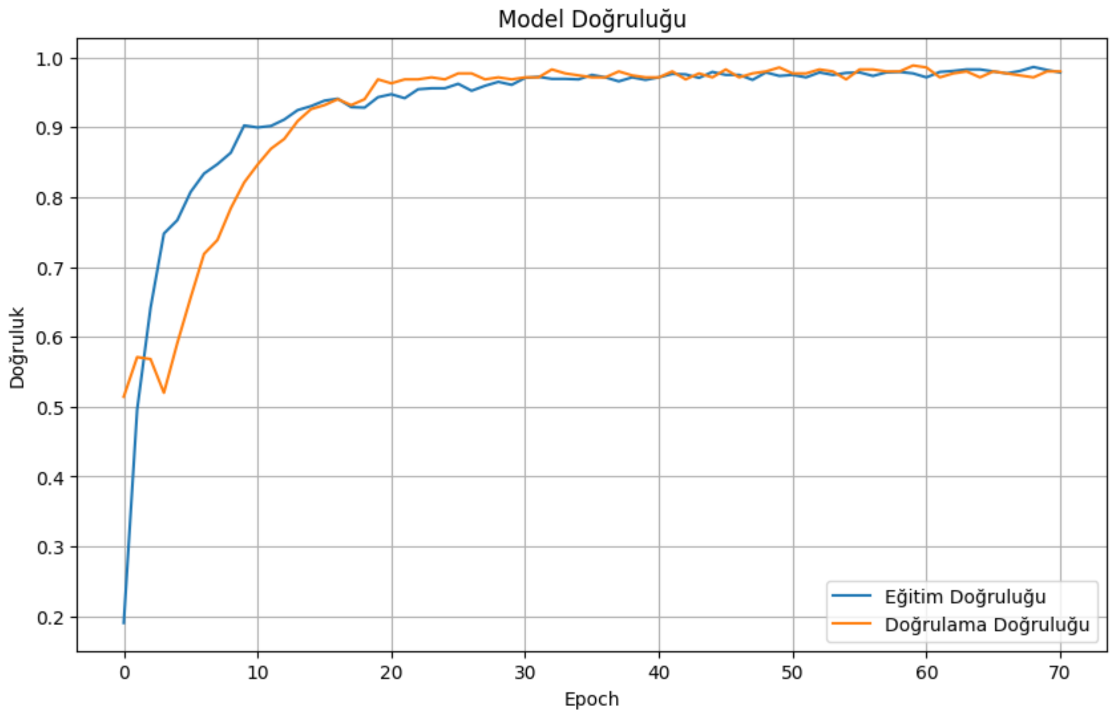
```
Test Loss: 0.025304868817329407
```

```
Test Accuracy: 0.9931818246841431
```

[ ]:
```
plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'], label='Eğitim Kaybı')
plt.plot(history.history['val_loss'], label='Doğrulama Kaybı')
plt.title('Model Kaybı')
plt.xlabel('Epoch')
plt.ylabel('Kayıp')
plt.legend()
plt.grid(True)
plt.show()
```



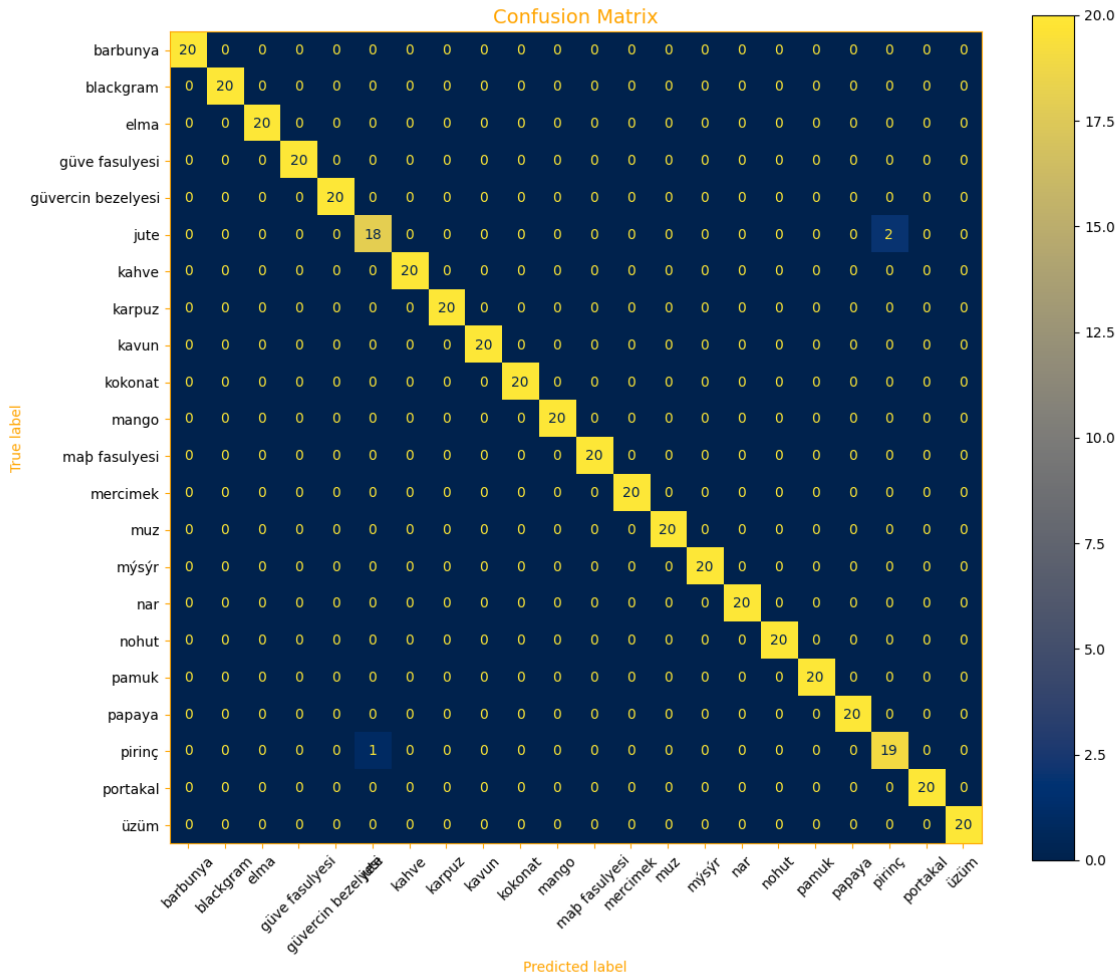
[ ]:
```
# Eğitim ve Doğrulama Doğruluğu
plt.figure(figsize=(10, 6))
plt.plot(history.history['accuracy'], label='Eğitim Doğruluğu')
plt.plot(history.history['val_accuracy'], label='Doğrulama Doğruluğu')
plt.title('Model Doğruluğu')
plt.xlabel('Epoch')
plt.ylabel('Doğruluk')
plt.legend()
plt.grid(True)
plt.show()
```



[ ]:
```python
# Tahminler ve Karışıklık Matrisi
predictions = model.predict(X_test)
y_pred = np.argmax(predictions, axis=1)
y_true = np.argmax(y_test, axis=1)

cm = confusion_matrix(y_true, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=label_encoder.
 ↪classes_)
fig, ax = plt.subplots(figsize=(12, 10))
disp.plot(cmap='cividis', ax=ax, xticks_rotation=45)
plt.gca().patch.set_facecolor('black')  # Arka planı siyah yap
plt.gca().spines['bottom'].set_color('orange')
plt.gca().spines['top'].set_color('orange')
plt.gca().spines['left'].set_color('orange')
plt.gca().spines['right'].set_color('orange')
plt.gca().xaxis.label.set_color('orange')
plt.gca().yaxis.label.set_color('orange')
plt.gca().title.set_color('orange')
plt.tick_params(colors='orange', which='both')
plt.xticks(fontsize=10, rotation=45, color='black')  # Etiketler siyah
plt.yticks(fontsize=10, color='black')  # Etiketler siyah
plt.title("Confusion Matrix", color='orange', fontsize=14)
plt.tight_layout()
plt.show()
```

```
14/14                  0s 22ms/step
```




```
[ ]: def predict_new_data(new_data):
         scaled_data = scaler.transform([new_data])
         prediction = model.predict(scaled_data)
         predicted_class = np.argmax(prediction, axis=1)
         return label_encoder.inverse_transform(predicted_class)[0]
```

```
[ ]: new_input = [78         ,33,         45,         18.928519         ,72.
     ↪800861         ,6.158860         ,82.341629]
```

```
[ ]: predicted_label = predict_new_data(new_input)
```

```
1/1                0s 49ms/step

/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:2739:
UserWarning:
```

```
X does not have valid feature names, but StandardScaler was fitted with feature
names
```

[ ]:
```python
print(f"Ekmenizi önerdiğimiz hasat: {predicted_label}")
```

```
Ekmenizi önerdiğimiz hasat: mýsýr
```