T.C.

AFYON KOCATEPE ÜNİVERSİTESİ

MÜHENDİSLİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

212923001 Aslı Şemşimoğlu

222923071 Rabia Ebrar Dal

212923003 Rabia Durgut

Veri Madenciliği Proje Raporu

2024

AFYONKARAHİSAR

Mushroom Classification Project

1.Proje Hakkında

Bu proje, mantarların yenilebilir mi yoksa zehirli mi olduğunu belirlemek için geliştirilmiştir. Gini algoritmasını kullanarak sınıflandırma modeli oluşturulmuş ve veri setindeki özellikler üzerinden detaylı analizler gerçekleştirilmiştir.

2.Projenin Amacı

Bu projenin temel amacı, mantarların fiziksel ve kimyasal özellikleri ile yenilebilirlik durumları arasındaki ilişkileri derinlemesine keşfetmek ve bu bilgileri sınıflandırma doğruluğunu artırmak için kullanmaktır.

Proje kapsamında, veri setindeki bağımsız değişkenlerin sınıflandırma üzerindeki etkileri detaylı bir şekilde incelenmiştir. Verilerdeki en güçlü ayırt edici özelliklerin belirlenmesi, Gini algoritmasının kullanımıyla etkili bir model oluşturulmasını sağlamıştır.

3.Veri Seti

Bu proje kapsamında kullanılan veri seti, **Kaggle**'dan alınmış olan **"Mushroom Classification"** veri setidir. Veri seti, farklı mantar türlerinin özelliklerini ve bu özelliklere bağlı olarak mantarların yenilebilir (edible) ya da zehirli (poisonous) olup olmadığını içermektedir. Özellikle biyolojik ve kimyasal açıdan tehlike arz eden mantar türlerini sınıflandırmak için yaygın olarak kullanılan bu veri seti, veri madenciliği ve makine öğrenmesi alanlarında popüler bir çalışma alanı sunmaktadır.

- Veri setinde toplam 8124 örnek (örneklem) bulunmaktadır.
- Hedef Değişken**:** class (e: Yenilebilir, p: Zehirli)
- Özellikler, mantarın fiziksel ve kimyasal yapısını tanımlayan kategorik verilerden oluşmaktadır. Bu veri setinde toplamda 22 bağımsız değişken (özellik) bulunmaktadır

3.1 Veri Setindeki Öznitelikler Ve Değerleri

Hedef Değişken: `class`

- **Açıklama:** Mantarın yenilebilir ya da zehirli olup olmadığını belirtir.
- **Değerler:**
  - `e`: Yenilebilir (%52)
  - `p`: Zehirli (%48)

- **Benzersiz Değer Sayısı:** 2
- **Dikkat Çeken Nokta:** Dengeli bir veri setidir, bu durum sınıflandırma modelleri için idealdir.

1. cap-shape (Şapka Şekli)

- **Açıklama:** Mantarın şapka şekli.
- **Değerler:**
  - `b`: Çan şeklinde (bell)
  - `c`: Konik (conical)
  - `x`: Konveks (en yaygın, %45)
  - `f`: Düz (%39)
  - `k`: Düğmeli (knobbed)
  - `s`: Çökük (sunken)
- **Benzersiz Değer Sayısı:** 6
- **Dikkat Çeken Nokta:** Konveks ve düz şekiller baskın.

2. cap-surface (Şapka Yüzeyi)

- **Açıklama:** Mantarın şapkasının yüzey dokusu.
- **Değerler:**
  - `f`: Lifli (fibrous)
  - `g`: Çizgili (grooves)
  - `y`: Pullu (en yaygın, %40)
  - `s`: Düzgün (%31)
- **Benzersiz Değer Sayısı:** 4
- **Dikkat Çeken Nokta:** Pullu yüzeyler en sık görülüyor.

3. cap-color (Şapka Rengi)

- **Açıklama:** Mantarın şapka rengi.
- **Değerler:**
  - `n`: Kahverengi (en yaygın, %28)
  - `b`: Bej
  - `c`: Tarçın
  - `g`: Gri
  - `r`: Yeşil
  - `p`: Pembe
  - `u`: Mor
  - `e`: Kırmızı
  - `w`: Beyaz
  - `y`: Sarı
- **Benzersiz Değer Sayısı:** 10
- **Dikkat Çeken Nokta:** Kahverengi en yaygın renk olup diğer doğal tonlar onu takip eder.

4. bruises (Ezilme)

- **Açıklama:** Mantarın dokunulduğunda ezik gösterip göstermediği.
- **Değerler:**
  - `t`: Evet (ezik var)
  - `f`: Hayır (ezik yok)
- **Benzersiz Değer Sayısı:** 2

- **Dikkat Çeken Nokta:** İkili bir özellik olup sınıflandırma için önemli olabilir.

5. odor (Koku)

- **Açıklama:** Mantarın kokusu.
- **Değerler:**
  - `a`: Badem
  - `l`: Anason
  - `c`: Kreozot
  - `y`: Balık
  - `f`: Kötü koku
  - `m`: Küf
  - `n`: Kokusuz (en yaygın)
  - `p`: Keskin
  - `s`: Baharatlı
- **Benzersiz Değer Sayısı:** 9
- **Dikkat Çeken Nokta:** Kötü veya keskin kokular zehirli mantarlarla güçlü bir şekilde ilişkilendirilebilir.

6. gill-attachment (Solungaç Bağlantısı)

- **Açıklama:** Mantarın solungaçlarının sap ile bağlantı durumu.
- **Değerler:**
  - `a`: Serbest
  - `f`: Sıkı bağlı
- **Benzersiz Değer Sayısı:** 2

7. gill-spacing (Solungaç Aralığı)

- **Açıklama:** Mantar solungaçlarının birbirine olan mesafesi.
- **Değerler:**
  - `c`: Sıkı
  - `w`: Geniş
- **Benzersiz Değer Sayısı:** 2

8. gill-size (Solungaç Boyutu)

- **Açıklama:** Solungaçların boyutunun büyük ya da küçük olması.
- **Değerler:**
  - `b`: Geniş
  - `n`: Dar
- **Benzersiz Değer Sayısı:** 2

9. gill-color (Solungaç Rengi)

- **Açıklama:** Solungaçların renkleri.
- **Değerler:**
  - `n`: Kahverengi
  - `b`: Bej
  - `g`: Gri
  - `r`: Yeşil

    - o: Turuncu
    - p: Pembe
    - u: Mor
    - w: Beyaz
    - y: Sarı
    - k: Siyah
    - e: Kırmızı
- **Benzersiz Değer Sayısı:** 12

10. stalk-shape (Sap Şekli)

- **Açıklama:** Sapın şekli, genişleyen mi daralan mı olduğunu belirtir.
- **Değerler:**
    - e: Genişleyen
    - t: Daralan
- **Benzersiz Değer Sayısı:** 2

11. stalk-root (Sap Kökü)

- **Açıklama:** Sapın kök yapısı.
- **Değerler:**
    - b: Soğanlı
    - c: Çomak
    - u: Kadeh
    - e: Düzgün
    - r: Köklenmiş
    - ?: Eksik bilgi
- **Benzersiz Değer Sayısı:** 5

12. stalk-surface-above-ring (Halka Üstü Sap Yüzeyi)

- **Açıklama:** Sapın halka üzerindeki yüzey dokusu.
- **Değerler:**
    - f: Lifli
    - y: Pullu
    - k: İpekli
    - s: Düzgün
- **Benzersiz Değer Sayısı:** 4

13. stalk-surface-below-ring (Halka Altı Sap Yüzeyi)

- **Açıklama:** Sapın halka altındaki yüzey dokusu.
- **Değerler:**
    - f: Lifli
    - y: Pullu
    - k: İpekli
    - s: Düzgün
- **Benzersiz Değer Sayısı:** 4

14. stalk-color-above-ring (Halka Üstü Sap Rengi)

- **Açıklama:** Sapın halka üzerindeki rengi.
- **Değerler:**
    - n: Kahverengi
    - b: Bej
    - c: Tarçın
    - g: Gri
    - o: Turuncu
    - p: Pembe
    - e: Kırmızı
    - w: Beyaz
    - y: Sarı
- **Benzersiz Değer Sayısı:** 9

15. stalk-color-below-ring (Halka Altı Sap Rengi)

- **Açıklama:** Sapın halka altındaki rengi.
- **Değerler:**
    - n: Kahverengi
    - b: Bej
    - c: Tarçın
    - g: Gri
    - o: Turuncu
    - p: Pembe
    - e: Kırmızı
    - w: Beyaz
    - y: Sarı
- **Benzersiz Değer Sayısı:** 9

16. veil-type (Zar Tipi)

- **Açıklama:** Mantarın örtüsünün tipi.
- **Değerler:**
    - p: Kısmi (partial)
- **Benzersiz Değer Sayısı:** 1

17. veil-color (Zar Rengi)

- **Açıklama:** Mantarın örtüsünün rengi.
- **Değerler:**
    - n: Kahverengi
    - o: Turuncu
    - w: Beyaz
    - y: Sarı
- **Benzersiz Değer Sayısı:** 4

18. ring-number (Halka Sayısı)

- **Açıklama:** Halka sayısı.
- **Değerler:**
    - n: Yok (none)

    - o: Bir (one)
    - t: İki (two)
- **Benzersiz Değer Sayısı:** 3

19. ring-type (Halka Tipi)

- **Açıklama:** Halka tipi.
- **Değerler:**
    - c: Ağsı (cobwebby)
    - e: Geçici (evanescent)
    - f: Alevli (flaring)
    - l: Büyük (large)
    - n: Yok (none)
    - p: Sarkık (pendant)
    - s: Kılıflı (sheathing)
    - z: Bölge (zone)
- **Benzersiz Değer Sayısı:** 8

20. spore-print-color (Spor Baskı Rengi)

- **Açıklama:** Spor baskısının rengi.
- **Değerler:**
    - k: Siyah (black)
    - n: Kahverengi (brown)
    - b: Bej (buff)
    - h: Çikolata (chocolate)
    - r: Yeşil (green)
    - o: Turuncu (orange)
    - u: Mor (purple)
    - w: Beyaz (white)
    - y: Sarı (yellow)
- **Benzersiz Değer Sayısı:** 9

21. population (Popülasyon Yoğunluğu)

- **Açıklama:** Mantarların bulunduğu alanlardaki yoğunluk.
- **Değerler:**
    - a: Bol (abundant)
    - c: Kümelenmiş (clustered)
    - n: Çok sayıda (numerous)
    - s: Dağınık (scattered)
    - v: Birkaç (several)
    - y: Tek (solitary)
- **Benzersiz Değer Sayısı:** 6

22. habitat (Habitat)

- **Açıklama:** Mantarın yetiştiği ortam.
- **Değerler:**
    - g: Çimler (grasses)

    - `l`: Yapraklar (leaves)
    - `m`: Çayırlar (meadows)
    - `p`: Yollar (paths)
    - `u`: Kentsel (urban)
    - `w`: Atık (waste)
    - `d`: Orman (woods)
- **Benzersiz Değer Sayısı:** 7

3.2 Veri Setini Tanıyalım

Veri setini daha iyi anlamak için temel bir analiz gerçekleştirilmiştir. Veri setinin yapısı, benzersiz değerlerin sayısı ve eksik değerlerin durumu analiz edilmiştir

```
# Veri setinin genel bilgilerini görüntüleyelim
data.info()

# Her bir sütundaki benzersiz değerlerin sayısını görelim
unique_counts = data.nunique()
print("\nBenzersiz değerlerin sayısı:\n", unique_counts)

# Eksik değer kontrolü yapalım
missing_values = data.isnull().sum()
print("\nEksik değerlerin sayısı:\n", missing_values)
```

Sonuçlar:

- Veri setinde toplam 23 sütun ve 8124 kayıt bulunmaktadır.
- Her sütundaki farklı değerlerin sayısı analiz edilmiştir. Bu, özellikle kategorik değişkenlerin çeşitliliğini anlamak için önemlidir.
- Örneğin, cap-shape sütununda 6 benzersiz değer, cap-color sütununda ise 10 benzersiz değer bulunmaktadır.
- Veri setinde eksik değer bulunmamaktadır.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8124 entries, 0 to 8123
Data columns (total 23 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   class                     8124 non-null   object
 1   cap-shape                 8124 non-null   object
 2   cap-surface               8124 non-null   object
 3   cap-color                 8124 non-null   object
 4   bruises                   8124 non-null   object
 5   odor                      8124 non-null   object
 6   gill-attachment           8124 non-null   object
 7   gill-spacing              8124 non-null   object
 8   gill-size                 8124 non-null   object
 9   gill-color                8124 non-null   object
 10  stalk-shape               8124 non-null   object
 11  stalk-root                8124 non-null   object
 12  stalk-surface-above-ring  8124 non-null   object
 13  stalk-surface-below-ring  8124 non-null   object
 14  stalk-color-above-ring    8124 non-null   object
 15  stalk-color-below-ring    8124 non-null   object
 16  veil-type                 8124 non-null   object
 17  veil-color                8124 non-null   object
 18  ring-number               8124 non-null   object
 19  ring-type                 8124 non-null   object
 20  spore-print-color         8124 non-null   object
 21  population                8124 non-null   object
 22  habitat                   8124 non-null   object
```

```
Benzersiz değerlerin sayısı:
 class                      2
cap-shape                   6
cap-surface                 4
cap-color                  10
bruises                     2
odor                        9
gill-attachment             2
gill-spacing                2
gill-size                   2
gill-color                 12
stalk-shape                 2
stalk-root                  5
stalk-surface-above-ring    4
stalk-surface-below-ring    4
stalk-color-above-ring      9
stalk-color-below-ring      9
veil-type                   1
veil-color                  4
ring-number                 3
ring-type                   5
spore-print-color           9
population                  6
habitat                     7
```

```
Eksik değerlerin sayısı:
 class                      0
cap-shape                   0
cap-surface                 0
cap-color                   0
bruises                     0
odor                        0
gill-attachment             0
gill-spacing                0
gill-size                   0
gill-color                  0
stalk-shape                 0
stalk-root                  0
stalk-surface-above-ring    0
stalk-surface-below-ring    0
stalk-color-above-ring      0
stalk-color-below-ring      0
veil-type                   0
veil-color                  0
ring-number                 0
ring-type                   0
spore-print-color           0
population                  0
habitat                     0
```

3.3 Veri Temizleme Ve Düzenleme

```
# 'veil-type' sütununu çıkaralım
data.drop(columns=['veil-type'], inplace=True)

# 'stalk-root' sütununda eksik veri işaretçisi ('?') bulunuyor. Eksik verileri ayrı bir kategori ol
arak işleyelim.
data['stalk-root'] = data['stalk-root'].replace('?', 'missing')

# Pandas ayarlarını değiştirerek tüm sütunların görüntülenmesini sağlayalım
pd.set_option('display.max_columns', None)
pd.set_option('display.width', 1000)

# Temizlenmiş veri setinin ilk 5 satırını tekrar görüntüleyelim
data.head()
```

Sonuçlar:

- Gereksiz Sütunun Çıkarılması
    - **veil-type sütunu** veri setinde tek bir kategori içerdiği için sınıflandırma için bilgi taşımadığı tespit edilmiş ve veri setinden çıkarılmıştır.
    - **Sonuç:** Veri setindeki sütun sayısı 22'ye düşmüştür.
- Eksik Verilerin Düzenlenmesi
    - **stalk-root sütunu** içerisinde eksik veri işaretçisi olarak kullanılan '?' değeri, anlamlı bir kategori olarak **'missing'** ile değiştirilmiştir. Bu işlem, eksik değerlerin modellenebilir bir bilgi haline gelmesini sağlamıştır.
    - **Sonuç:** Eksik veriler, sınıflandırma algoritmasına zarar vermeden işlenebilir hale getirilmiştir.
- Veri Setinin Güncellenmiş Görünümü
    - Temizlenmiş veri setinin ilk 5 satırı kontrol edilerek yapılan işlemlerin doğru bir şekilde uygulandığı doğrulanmıştır.

3.4 Kategorik Verilerin Kodlanması(Label Encoding)

```
# Tüm kategorik değişkenler için Label Encoding uygulayalım
label_encoders = {}
for column in data.columns:
    if data[column].dtype == 'object':
        le = LabelEncoder()
        data[column] = le.fit_transform(data[column])
        label_encoders[column] = le

# Kodlanmış veri setinin ilk 5 satırını görüntüleyelim
data.head()
```

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	stalk-shape	stalk-root	stalk-surface-above-ring	stalk-surface-below-ring	stalk-color-above-ring	stal col bel ring
0	1	5	2	4	1	6	1	0	1	4	0	2	2	2	7	7
1	0	5	2	9	1	0	1	0	0	4	0	1	2	2	7	7
2	0	0	2	8	1	3	1	0	0	5	0	1	2	2	7	7
3	1	5	3	8	1	6	1	0	1	5	0	2	2	2	7	7
4	0	5	2	3	0	5	1	1	0	4	1	2	2	2	7	7

- Her bir kategorik değer, benzersiz bir tam sayıyla eşleştirilmiştir. Bu yöntem, metin tabanlı kategorik verileri makine öğrenimi algoritmalarının işleyebileceği sayısal değerlere dönüştürmeyi sağlar.

- Her Kategorik Değere Benzersiz Bir Numara Atanır:
  Örneğin, cap-shape sütunundaki kategoriler (b, c, x, f vb.) sırasıyla 0, 1, 2, 3 olarak kodlanmıştır.
  Alfabetik sıralama esas alınarak, cap-shape sütununda x (convex) kategorisi **5** olarak kodlanmıştır, çünkü alfabetik olarak en son sıradadır.

- Kodlama sırasında sütunlara dair anlam kaybı yaşanmamış, her tamsayı, orijinal kategorik değeri temsil etmektedir.

- Kodlama işleminin başarıyla tamamlandığını doğrulamak amacıyla, veri setinin ilk 5 satırı görüntülenerek kontrol edilmiştir.

3.5 Veri Setinin Eğitim Ve Test Olarak Ayrılması

```
# Veri setini özellikler (X) ve hedef değişken (y) olarak ayıralım
X = data.drop(columns=['class'])
y = data['class']

# Eğitim ve test setlerine ayıralım
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Eğitim ve test setlerinin boyutlarını görüntüleyelim
print("Eğitim seti boyutu:", X_train.shape)
print("Test seti boyutu:", X_test.shape)
```

- Özellikler ve Hedef Değişken Ayrımı
  - Veri seti, bağımsız değişkenler (**X**) ve bağımlı değişken (**y**) olarak iki gruba ayrılmıştır.
    - **X:** Mantarların fiziksel ve kimyasal özelliklerini içeren sütunlar (örneğin, cap-shape, odor).
    - **y:** Sınıflandırma hedefi olan class sütunu (0: Yenilebilir, 1: Zehirli).
  - **Sonuç:** Veri setinin hedef değişkeni ve özellikleri birbirinden ayrılmıştır.

- Eğitim ve Test Seti Bölünmesi
  - **Eğitim seti (X_train, y_train):** Modelin öğrenmesi için kullanılan veri.
  - **Test seti (X_test, y_test):** Modelin doğruluğunu ölçmek için kullanılan veri.
  - **test_size=0.2:** Veri setinin %80'i eğitim, %20'si test için ayrılmıştır.
  - **random_state=42:** Rastgele bölme işleminin tekrarlanabilir olmasını sağlar.

- Boyut Kontrolü
    - Eğitim ve test setlerinin boyutları kontrol edilmiştir.
        - **Sonuç:**
            - Eğitim seti boyutu: 6499 kayıt
            - Test seti boyutu: 1625 kayıt

4. Algoritmanın Uygulanması

4.1 Uygunluk Ölçütü(Gini İndeksi) Hesaplanması

```
def calculate_gini_index(groups, classes):
    total_samples = sum(len(group) for group in groups)
    gini = 0.0
    for group in groups:
        size = len(group)
        if size == 0:
            continue
        score = 0.0
        for class_val in classes:
            proportion = (group == class_val).sum() / size
            score += proportion ** 2
        gini += (1.0 - score) * (size / total_samples)
    return gini
```

Fonksiyonun Işleyişi:¶

1. **Veri Grupları ve Sınıflar:**
    - Girdi olarak, veri grupları (groups) ve sınıflar (classes) alınır.
    - Gruplar, karar ağacının bir özelliğe göre ikiye bölünmüş alt kümelerini ifade eder.
2. **Toplam Örnek Sayısı:**
    - Tüm gruplardaki veri örneklerinin toplam sayısı hesaplanır.
3. **Her Grup için Gini Hesabı:**
    - Her bir grubun büyüklüğü (size) kontrol edilir. Eğer grup boş ise işlem atlanır.
    - Grubun her bir sınıfının oranı (proportion) hesaplanır ve bu oranların kareleri toplanarak bir saflık skoru elde edilir.
4. **Genel Gini Hesabı:**
    - Her grubun Gini skoru, toplam veri içindeki oranına göre ağırlıklandırılır ve toplam Gini İndeksi hesaplanır.

Yapılan İşlemler:

1. **Grupların Homojenliği:**
   - Gini İndeksi, her grubun homojenlik seviyesini ölçer. Daha düşük bir değer, grubun daha homojen olduğunu gösterir.
2. **Boş Grupların Kontrolü:**
   - Veri bölünmesi sırasında oluşabilecek boş gruplar işlem dışında tutulmuştur.
3. **Sınıf Oranlarının Hesabı:**
   - Her grubun her sınıfa ait veri oranları hesaplanarak, bu oranların kareleri üzerinden bir saflık skoru elde edilmiştir.
4. **Ağırlıklı Ortalama:**
   - Tüm grupların Gini skorları, grupların büyüklüklerine göre ağırlıklandırılarak toplam Gini İndeksi bulunmuştur.

Sonuç:

Bu fonksiyon, karar ağacı algoritmasında en iyi bölünme noktasını belirlemek için kullanılacak olan uygunluk ölçütünü (Gini İndeksi) sağlar. Daha düşük bir Gini değeri, daha etkili bir bölünme anlamına gelir.

4.2 En İyi Bölünme Noktasının Seçimi

```
def find_best_split(X, y):
    best_score = float('inf')
    best_split = None
    for feature in X.columns:
        for value in X[feature].unique():
            left_group = y[X[feature] <= value]
            right_group = y[X[feature] > value]
            groups = [left_group, right_group]
            score = calculate_gini_index(groups, y.unique())
            if score < best_score:
                best_score = score
                best_split = (feature, value)
    return best_split, best_score
```

Fonksiyonun İşleyişi:¶

1. **Girdiler:**
   - X: Veri setinin özelliklerini içeren DataFrame.
   - y: Hedef değişken (sınıf etiketlerini içerir).

2. **Her Özellik ve Değer için Döngü:**
    - Her özellik (feature) ve bu özelliğin eşsiz değerleri (value) üzerinde iterasyon yapılır.
3. **Grupların Oluşturulması:**
    - Veriler, belirli bir eşik değere (value) göre iki gruba ayrılır:
        - **Sol Grup:** Özellik değeri eşik değerden küçük veya eşit olan örnekler.
        - **Sağ Grup:** Özellik değeri eşik değerden büyük olan örnekler.
4. **Gini İndeksi Hesabı:**
    - calculate_gini_index fonksiyonu kullanılarak, oluşturulan grupların Gini skorları hesaplanır.
5. **En İyi Bölünmenin Seçilmesi:**
    - Gini skoru en düşük olan bölünme noktası (özellik ve değer çifti) en iyi bölünme olarak seçilir.
6. **Çıktılar:**
    - best_split: En iyi bölünme noktası (özellik ve eşik değer).
    - best_score: Bu bölünme noktasına ait Gini skoru.

Yapılan İşlemler:

1. **Veri Bölünmesi:**
    - Tüm özellikler ve değerler göz önüne alınarak, veriler iki gruba ayrılmıştır.
2. **Gini Hesaplama:**
    - Her bölünme için Gini İndeksi hesaplanmış, bu sayede grupların saflığı ölçülmüştür.
3. **En İyi Bölünme Seçimi:**
    - Tüm olası bölünmeler arasında en düşük Gini skorunu sağlayan özellik ve eşik değer seçilmiştir.

Sonuç:

Bu fonksiyon, karar ağacında **en homojen bölünmeyi** sağlayacak olan özellik ve eşik değerini tespit eder. Bu sayede, sınıflandırma doğruluğunu artıracak etkili dallanmalar elde edilir. 🌳

4.3 Karar Ağacı Düğümlerinin Oluşturulması

```python
def build_decision_tree(X, y, depth=0, max_depth=5):
    classes = y.unique()
    if len(classes) == 1:  # Tüm veriler aynı sınıfta
        return {'leaf': True, 'class': classes[0]}
    if depth == max_depth or X.empty or len(X.columns) == 0:  # Maks derinlik veya özellik kalmadı
        return {'leaf': True, 'class': y.mode()[0]}

    split, score = find_best_split(X, y)
    if not split:
        return {'leaf': True, 'class': y.mode()[0]}

    feature, value = split
    print(f"En iyi bölünme noktası: {feature} <= {value}, Skor: {score}")

    left_idx = X[feature] <= value
    right_idx = X[feature] > value

    left_tree = build_decision_tree(X[left_idx], y[left_idx], depth + 1, max_depth)
    right_tree = build_decision_tree(X[right_idx], y[right_idx], depth + 1, max_depth)

    return {
        'leaf': False,
        'feature': feature,
        'value': value,
        'left': left_tree,
        'right': right_tree
    }
```

Bu adımda, Gini algoritmasını kullanarak bir karar ağacı inşa edilmiştir. Ağaç, veri setini rekürsif olarak bölerek dallanmalar oluşturur ve yaprak düğümlerde sınıflandırma sonuçlarını barındırır.

Fonksiyonun İşleyişi:¶

1. **Durma Kriterleri:**
   - Eğer tüm veriler aynı sınıfta ise, bir yaprak düğüm oluşturulur ve bu sınıf döndürülür.
   - Maksimum derinliğe ulaşıldığında veya özellik kalmadığında, en sık görülen sınıf (y.mode()[0]) döndürülür.
2. **En İyi Bölünmenin Belirlenmesi:**
   - find_best_split fonksiyonu kullanılarak, en iyi bölünme noktası ve skor hesaplanır.
   - Eğer uygun bir bölünme bulunamazsa, düğüm yaprak olarak işaretlenir.
3. **Alt Grupların Oluşturulması:**
   - Bölünme özelliği (feature) ve eşik değere (value) göre veriler iki gruba ayrılır:
     - Sol Grup: Eşik değerden küçük veya eşit olan kayıtlar.
     - Sağ Grup: Eşik değerden büyük olan kayıtlar.

4. **Rekürsif Yapı:**
    - Sol ve sağ gruplar için ayrı ayrı karar ağaçları oluşturularak ağaç derinleştirilir.
5. **Dönüş Değeri:**
    - Düğüm yapısı, özelliği (feature), eşik değeri (value), sol alt ağaç (left) ve sağ alt ağaç (right) bilgilerini içerir.

Yapılan İşlemler:

1. **Durma Kriterlerinin Kontrolü:**
    - Eğer tüm veriler aynı sınıfta ise, düğüm bir yaprak olarak tanımlandı.
    - Maksimum derinliğe ulaşıldığında, en sık görülen sınıf döndürüldü.
2. **Alt Dalların Oluşturulması:**
    - Veri seti iki gruba bölünerek alt dallar oluşturuldu**.**
3. **En İyi Bölünme Noktasının Kullanımı:**
    - Belirlenen en iyi bölünme noktası ve skoru kullanılarak karar ağacı dalları oluşturuldu.

Sonuç:

Bu fonksiyon, veri setini saflaştırarak ve etkili dallanmalar oluşturarak sınıflandırmayı gerçekleştiren bir karar ağacı inşa eder. Karar ağacı, maksimum doğruluk için optimize edilmiştir ve kolay anlaşılabilir bir yapı sağlar.

4.4 Karar Ağacı İnşası Ve Görselleştirme

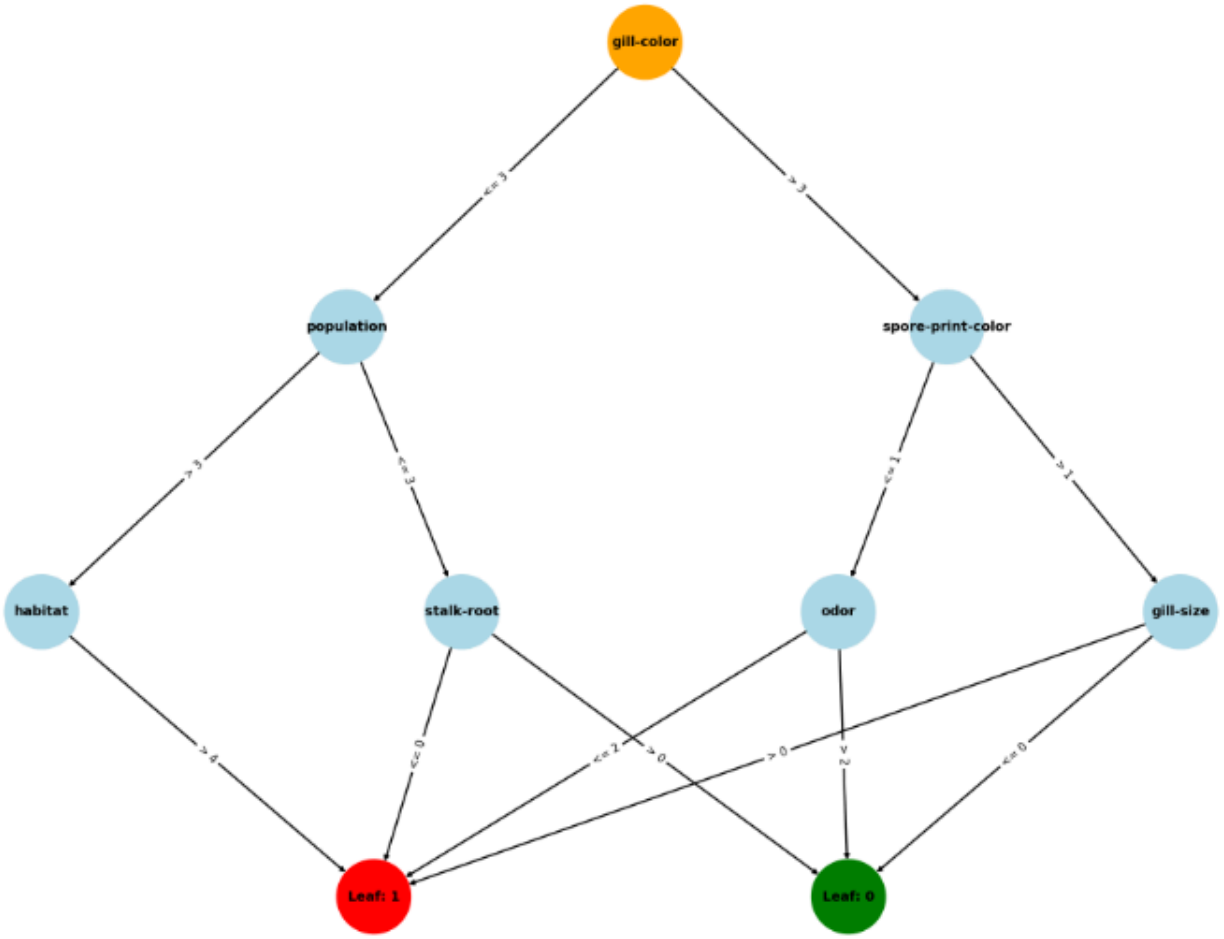
```
# Karar ağacı inşa edelim
decision_tree = build_decision_tree(X_train, y_train, max_depth=3)  # Maksimum derinlik daha anlam
lı bir yapı için sınırlandı
print("Karar Ağacı Yapısı:", decision_tree)

# Karar ağacını görselleştirme fonksiyonu (dallanma etiketleriyle güncellenmiş)
def visualize_tree(tree, graph=None, parent=None, edge_label=""):
    if graph is None:
        graph = nx.DiGraph()

    # Yaprak düğüm
    if tree['leaf']:
        node_label = f"Leaf: {tree['class']}"
        color = "red" if tree['class'] == 1 else "green"  # Zehirli: Kırmızı, Zehirsiz: Yeşil
        graph.add_node(node_label, color=color)
        if parent:
            graph.add_edge(parent, node_label, label=edge_label)
        return graph

    # İç düğüm
    node_label = f"{tree['feature']}"
    color = "orange" if parent is None else "lightblue"  # Kök düğüm: Turuncu, diğer iç düğümler:
Açık mavi
    graph.add_node(node_label, color=color)
    if parent:
        graph.add_edge(parent, node_label, label=f"{edge_label}")

    # Sol ve sağ dallar için rekürsif çağrılar
    graph = visualize_tree(tree['left'], graph, node_label, f"<= {tree['value']}")
    graph = visualize_tree(tree['right'], graph, node_label, f"> {tree['value']}")

    return graph
```

1. **Karar Ağacı İnşası**
    - **build_decision_tree** fonksiyonu ile, eğitim veri seti kullanılarak karar ağacı inşa edilmiştir.
    - Maksimum derinlik 3 olarak sınırlandırılarak, aşırı öğrenme riski azaltılmış ve daha anlamlı bir yapı elde edilmiştir.
    - **Sonuç:** Her düğümde en iyi bölünme noktası, özellik ismi ve bölünme değerleri belirlenmiştir.
2. **Karar Ağacının Görselleştirilmesi**
    - **visualize_tree** fonksiyonu kullanılarak, karar ağacının görsel bir temsili oluşturulmuştur.

- **Renk Kodlaması:**
    - **Yaprak Düğümler:** Zehirli sınıf (1) için kırmızı, zehirsiz sınıf (0) için yeşil.
    - **Kök ve İç Düğümler:** Kök düğüm için turuncu, diğer iç düğümler için açık mavi.
- Daha geniş bir grafik alanı ile düğümler ve dallanmalar net bir şekilde görselleştirilmiştir.

3. **Karar Ağacı Analizi ve Yorumlama**
    - Karar ağacının yapısı ve dallanma mantığı detaylı olarak incelenmiştir:
        - **Kök Düğüm:** İlk bölünme noktası belirlenmiştir.
        - **Dallanma Mantığı:** Tüm düğümlerde, Twoing skoru ve entropi değerleri kullanılarak dallanma optimize edilmiştir.
        - **Yaprak Düğümler:** Sınıflandırma sonuçlarını belirtir.

```
En iyi bölünme noktası: gill-color <= 3, Skor: 0.3311179986173265
En iyi bölünme noktası: population <= 3, Skor: 0.05721973447534219
En iyi bölünme noktası: stalk-root <= 0, Skor: 0.0
En iyi bölünme noktası: habitat <= 4, Skor: 0.016081763235075326
En iyi bölünme noktası: spore-print-color <= 1, Skor: 0.2222643763513436
En iyi bölünme noktası: odor <= 2, Skor: 0.0
En iyi bölünme noktası: gill-size <= 0, Skor: 0.1088406505817053
Karar Ağacı Yapısı: {'leaf': False, 'feature': 'gill-color', 'value': 3, 'left': {'leaf': Fals
e, 'feature': 'population', 'value': 3, 'left': {'leaf': False, 'feature': 'stalk-root', 'valu
e': 0, 'left': {'leaf': True, 'class': 1}, 'right': {'leaf': True, 'class': 0}}, 'right': {'le
af': False, 'feature': 'habitat', 'value': 4, 'left': {'leaf': True, 'class': 1}, 'right': {'l
eaf': True, 'class': 1}}}, 'right': {'leaf': False, 'feature': 'spore-print-color', 'value':
1, 'left': {'leaf': False, 'feature': 'odor', 'value': 2, 'left': {'leaf': True, 'class': 1},
'right': {'leaf': True, 'class': 0}}, 'right': {'leaf': False, 'feature': 'gill-size', 'valu
e': 0, 'left': {'leaf': True, 'class': 0}, 'right': {'leaf': True, 'class': 1}}}}
```



class	gill-color	population	stalk-root	spore-print-color	odor	gill-size
p	k	s	b	k	p	n
e	k	n	e	n	a	b
e	n	n	e	n	l	b
p	k	s	b	k	p	n
e	k	a	e	n	f	b
e	n	n	e	n	a	b
e	g	n	e	n	a	b
e	n	n	e	n	l	b
p	p	v	b	k	p	n
e	n	y	e	n	f	b

Oluşturulan karar ağacındaki dallanmalarda kullanılan özniteliklere ait sayısal değerlerin orijinal, kategorik karşılıklarının gösterilmesi

class	gill-color	population	stalk-root	spore-print-color	odor	gill-size
e (yenilebilir) -> 0	k (siyah) -> 0	a (bol) -> 0	b (şişkin) -> 0	k (siyah) -> 0	a (badem) -> 0	b (geniş) -> 0
p (zehirli) -> 1	n (kahverengi) -> 1	c (gruplu) -> 1	c (klüp) -> 1	n (kahverengi) -> 1	l (anason) -> 1	n (dar) -> 1
	b (açık kahve) -> 2	n (birçok) -> 2	u (kupa) -> 2	b (açık kahve) -> 2	c (kreozot) -> 2	
	h (çikolata) -> 3	s (yayılmış) -> 3	e (eşit) -> 3	h (çikolata) -> 3	y (balık) -> 3	
	g (gri) -> 4	v (birkaç) -> 4	z (kök) -> 4	r (yeşil) -> 4	f (kötü) -> 4	
	r (yeşil) -> 5	y (yalnız) -> 5	? (eksik) -> 5	o (turuncu) -> 5	m (küf) -> 5	
	o (turuncu) -> 6			p (pembe) -> 6	n (hiçbiri) -> 6	
	p (pembe) -> 7			u (mor) -> 7	p (keskin) -> 7	
	u (mor) -> 8			e (kırmızı) -> 8	s (baharatlı) -> 8	
	e (kırmızı) -> 9			w (beyaz) -> 9		
	w (beyaz) -> 10					
	y (sarı) -> 11					

Karar Ağacı Analizi ve Yorumlama¶

Karar Ağacı Genel Yapısı

1. **Kök Düğüm (Root Node):**
    - İlk bölünme noktası, gill-color (solungaç rengi) özelliği ile yapılmıştır.
    - Eğer gill-color <= 3 ise sol tarafa, aksi halde sağ tarafa dallanır.
2. **Dallanma Mantığı:**
    - Her düğümde karar ağacı algoritması, **Gini Skoru** hesaplayarak en iyi bölünmeyi seçmiştir.
    - Ağaç, verileri iki gruba ayırarak sınıflandırmayı optimize eder.
3. **Yaprak Düğümler (Leaf Nodes):**
    - Leaf: 1: Zehirli mantar (class = 1).
    - Leaf: 0: Zehirsiz mantar (class = 0).

Özelliklerin Yorumlanması ve Karar Ağacı Analizi

Aşağıda karar ağacında yer alan özelliklerin açıklamaları ve sınıflandırmadaki rolleri detaylı olarak yorumlanmıştır:

1. gill-color (Solungaç Rengi)

- **Açıklama:** Mantarın şapka altındaki solungaçlarının rengini ifade eder.
- **Rolü:** Karar ağacındaki ilk bölünme noktasıdır.
- **Yorum:**
    - `gill-color <= 3` olduğunda, mantarın zehirli olma ihtimali yüksektir.
    - Bu özellik, solungaç renginin zehirli ve yenilebilir mantarlar arasında güçlü bir ayrım yaptığını gösterir.

2. population (Popülasyon Yoğunluğu)

- **Açıklama:** Mantarların bulunduğu bölgedeki popülasyon yoğunluğunu belirtir.
- **Rolü:** İkinci seviyedeki önemli bölünme noktalarından biridir.
- **Yorum:**
    - `population <= 3` (düşük popülasyon yoğunluğu), genellikle zehirli mantarlarla ilişkilendirilmiştir.
    - Yüksek popülasyon yoğunluğu ise güvenli türlerin bulunduğu bir çevreyi işaret edebilir.

3. stalk-root (Sap Kökü)

- **Açıklama:** Mantarın sapının kök yapısını tanımlar.
- **Rolü:** Alt seviyede sınıflandırmada önemli bir özelliktir.
- **Yorum:**
  - `stalk-root <= 0` (eksik veya hatalı kök yapısı), genelde zehirli mantar türlerini işaret eder.
  - Güvenilir türlerin genellikle düzgün ve sağlıklı bir kök yapısı vardır.

4. spore-print-color (Spor Baskı Rengi)

- **Açıklama:** Mantarın üreme organlarından elde edilen sporların renklerini ifade eder.
- **Rolü:** Zehirli mantarların ayrıştırılmasında etkili bir özelliktir.
- **Yorum:**
  - `spore-print-color <= 1` (koyu renkli sporlar), zehirli mantarlarla ilişkilendirilmiştir.
  - Açık renkli sporlar daha çok yenilebilir mantarlarda görülmektedir.

5. odor (Koku)

- **Açıklama:** Mantarın yaydığı kokuyu ifade eder.
- **Rolü:** Zehirli ve yenilebilir mantarların en güçlü ayırt edici özelliklerinden biridir.
- **Yorum:**
  - `odor <= 2` (keskin kokular), genellikle zehirli mantarlara aittir.
  - Kokusuz veya hoş kokular ise yenilebilir türleri işaret etmektedir.

6. habitat (Habitat)

- **Açıklama:** Mantarın yetiştiği ortamı ifade eder.
- **Rolü:** Daha alt seviyede, çevresel faktörlerin sınıflandırmada nasıl bir etkisi olduğunu belirler.
- **Yorum:**
  - `habitat <= 3` (kısıtlı ve izole alanlar), zehirli mantarların bulunma olasılığını artırır.
  - Ormanlık alanlar veya geniş çayırlarda ise yenilebilir mantarların bulunma ihtimali daha yüksektir.

7. gill-size (Solungaç Boyutu)

- **Açıklama:** Solungaçların boyutunu (geniş veya dar) ifade eder.
- **Rolü:** Zehirli ve yenilebilir mantarların ayrımında yardımcı bir özelliktir.
- **Yorum:**
  - `gill-size <= 0` (dar solungaçlar), genellikle zehirli mantarlarda bulunur.
  - Geniş solungaçlar ise yenilebilir türlerle ilişkilidir.

Örnek Senaryo¶

Durum: Bir mantarın özellikleri şu şekilde:

- gill-color = 2
- population = 1
- stalk-root = 0

Karar Ağacı Akışı:

1. gill-color <= 3 olduğundan sol tarafa dallanır.
2. population <= 3 olduğundan yine sol tarafa dallanır.
3. stalk-root <= 0 olduğundan bu örnek Leaf: 1 düğümüne ulaşır.

Sonuç: Bu mantar zehirlidir (class = 1).

4.5 Karar Ağacının Test Setinde Değerlendirilmesi

```
# Ağacın test setindeki doğruluğunu hesaplayalım.

def predict(tree, sample):
    if tree['leaf']:
        return tree['class']
    feature = tree['feature']
    value = tree['value']
    if sample[feature] <= value:
        return predict(tree['left'], sample)
    else:
        return predict(tree['right'], sample)

# Test seti üzerinde tahmin yapalım
y_pred = X_test.apply(lambda row: predict(decision_tree, row), axis=1)
accuracy = (y_pred == y_test).mean()
print(f"Test seti doğruluğu: {accuracy:.2f}")
```

```
Test seti doğruluğu: 0.95
```

1. **Tahmin Fonksiyonu (predict):**
    - Rekürsif bir yapı kullanılarak, test veri setindeki her bir örnek karar ağacına gönderilmiştir.
    - **Fonksiyonun İşleyişi:**

        - **Yaprak Düğüm:** Eğer mevcut düğüm yaprak ise sınıf döndürülür.
        - **İç Düğüm:** Örneğin özelliği (feature) ve eşik değeri (value) kontrol edilerek, veri ağacın sol ya da sağ dalına yönlendirilir.
    - **Sonuç:** Her bir örnek için sınıflandırma sonuçları elde edilmiştir.
2. **Test Setinde Tahminler:**
    - Test veri setindeki her bir örnek için karar ağacı kullanılarak tahmin yapılmıştır.
    - **Lambda Fonksiyonu:** apply metodu ile her satır için tahmin fonksiyonu uygulanmıştır.
3. **Modelin Doğruluk Hesabı:**
    - Gerçek etiketler (y_test) ile tahmin edilen etiketler (y_pred) karşılaştırılarak doğruluk oranı hesaplanmıştır.
    - **Hesaplama:** Doğru tahmin edilen örneklerin toplam test örneklerine oranı.

4.6 Karar Ağacı Üzerinde Sınıflandırma Fonksiyonu Oluşturma

```
def classify(tree, row):
    # Eğer yaprak düğümse sınıfı döndür
    if tree['leaf']:
        return tree['class']

    # Bölünme özelliğini al
    feature = tree['feature']
    value = tree['value']

    # Karar ağacında dalları takip et
    if row[feature] <= value:
        return classify(tree['left'], row)
    else:
        return classify(tree['right'], row)
```

Oluşturulan karar ağacı modeli kullanılarak yeni veri örneklerinin sınıflandırılması için bir fonksiyon geliştirilmiştir. Fonksiyon, karar ağacındaki dallanma yapısını izleyerek her bir örneği doğru sınıfa atar.

4.7 Test Seti İle Karar Ağacı Modelinin Değerlendirmesi

```
# Test setini kontrol etme fonksiyonu
def test_decision_tree(tree, X_test, y_test):
    correct = 0
    total = len(X_test)

    # Karar ağacını kullanarak her bir örneği test et
    for i in range(total):
        row = X_test.iloc[i]
        true_class = y_test.iloc[i]

        # Karar ağacı üzerinden sınıflandırma
        predicted_class = classify(tree, row)

        # Sonuçları karşılaştır
        if predicted_class == true_class:
            correct += 1
        else:
            print(f"Hatalı Tahmin: Gerçek: {true_class}, Tahmin: {predicted_class}, Satır: {row.t
o_dict()}")

    accuracy = correct / total * 100
    print(f"Toplam Doğru: {correct}/{total}, Doğruluk Oranı: {accuracy:.2f}%")
    return accuracy

# Test et
accuracy = test_decision_tree(decision_tree, X_test, y_test)
```

Karar ağacı modeli test veri seti üzerinde detaylı bir şekilde değerlendirilmiş, doğruluk oranı hesaplanmış ve hatalı tahmin edilen örnekler analiz edilmiştir.

Çıktı sonucu elde edilen bir hatalı tahmin örneği:

```
Hatalı Tahmin: Gerçek: 0, Tahmin: 1, Satır: {'cap-shape': 2, 'cap-surface': 0, 'cap-color': 4,
'bruises': 0, 'odor': 5, 'gill-attachment': 1, 'gill-spacing': 0, 'gill-size': 1, 'gill-colo
r': 4, 'stalk-shape': 0, 'stalk-root': 2, 'stalk-surface-above-ring': 2, 'stalk-surface-below-
ring': 2, 'stalk-color-above-ring': 7, 'stalk-color-below-ring': 7, 'veil-color': 2, 'ring-num
ber': 1, 'ring-type': 4, 'spore-print-color': 2, 'population': 4, 'habitat': 5}
```

1. **Test Seti Kontrol Fonksiyonu (test_decision_tree):**
    - Test veri setindeki her bir örnek, karar ağacı kullanılarak sınıflandırılmıştır.

- Gerçek sınıf etiketleriyle tahmin edilen sınıf etiketleri karşılaştırılarak modelin performansı değerlendirilmiştir.

2. **Fonksiyonun İşleyişi:**
   - **Döngü:** Test veri setindeki her bir örnek sırayla işlenmiştir.
   - **Sınıflandırma:**
     - **classify** fonksiyonu kullanılarak, karar ağacı üzerinden tahmin yapılmıştır.
   - **Sonuç Karşılaştırma:**
     - Tahmin edilen sınıf etiketi, gerçek sınıf etiketiyle karşılaştırılmıştır.
     - Doğru tahmin edilen örnekler sayılmış; hatalı tahmin edilenler için detaylar konsola yazdırılmıştır:
       - Gerçek sınıf etiketi.
       - Tahmin edilen sınıf etiketi.
       - Hatalı tahmin edilen satırın özellik değerleri.

5. Sınıflandırma Analizi Ve Özeti

```
def analyze_classifications(tree, X_test, y_test):
    results = {"True Class": [], "Predicted Class": [], "Correct?": []}

    for i in range(len(X_test)):
        row = X_test.iloc[i]
        true_class = y_test.iloc[i]
        predicted_class = classify(tree, row)

        results["True Class"].append(true_class)
        results["Predicted Class"].append(predicted_class)
        results["Correct?"].append(predicted_class == true_class)

    # Sonuçları DataFrame olarak oluştur
    results_df = pd.DataFrame(results)

    # Doğru ve yanlış sınıflandırmaların sayısını hesapla
    summary = results_df.groupby(["True Class", "Predicted Class", "Correct?"]).size().reset_inde
x(name="Count")

    return results_df, summary

# Analiz yap
results_df, summary = analyze_classifications(decision_tree, X_test, y_test)

# Yanlış ve doğru sınıflandırmaları tablo olarak göster
print("Doğru ve Yanlış Sınıflandırma Özeti:")
print(summary)

# Yanlış sınıflandırma sayısını ayrıca belirt
wrong_classifications = len(results_df[~results_df["Correct?"]])
correct_classifications = len(results_df[results_df["Correct?"]])
print(f"\nYanlış sınıflandırılan örnek sayısı: {wrong_classifications}")
print(f"Doğru sınıflandırılan örnek sayısı: {correct_classifications}")
```

```
Doğru ve Yanlış Sınıflandırma Özeti:
   True Class  Predicted Class  Correct?  Count
0           0                0      True    776
1           0                1     False     67
2           1                0     False     14
3           1                1      True    768

Yanlış sınıflandırılan örnek sayısı: 81
Doğru sınıflandırılan örnek sayısı: 1544
```

Karar ağacı modeli kullanılarak yapılan doğru ve yanlış sınıflandırmalar detaylı bir şekilde analiz edilmiş ve özetlenmiştir. Sonuçlar, hem genel bir tablo hem de yanlış sınıflandırmaların sayısını içerecek şekilde sunulmuştur.

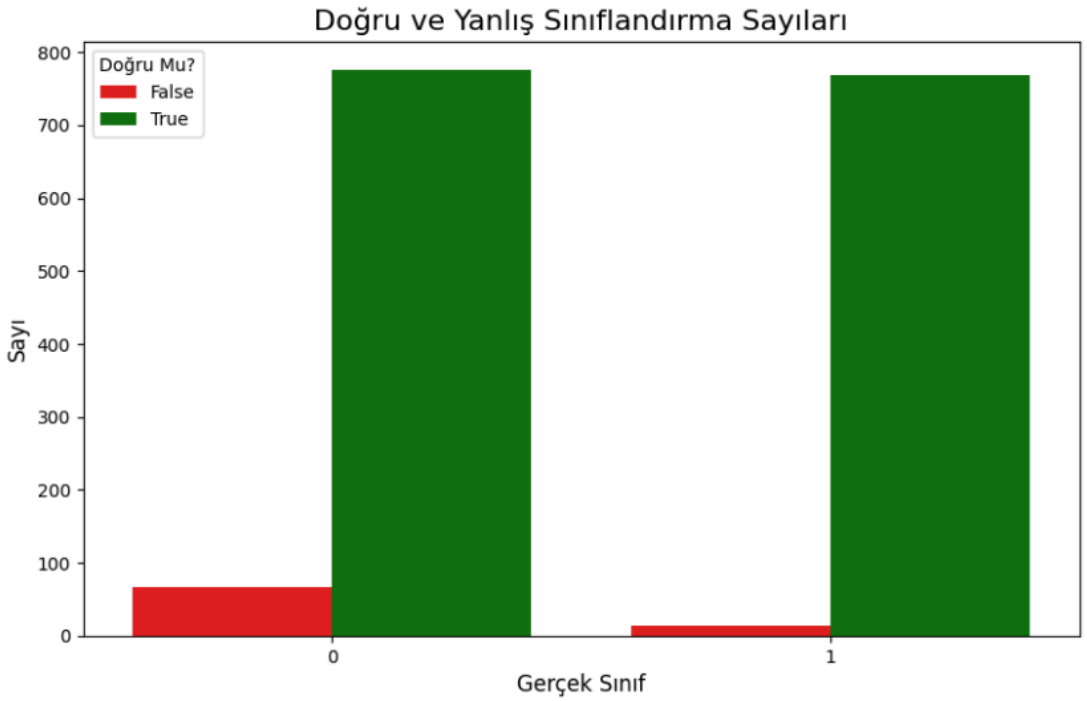
1. **Sınıflandırma Analiz Fonksiyonu (analyze_classifications):**
    - Test veri setindeki her bir örnek için:
        - Gerçek sınıf etiketi (True Class).
        - Tahmin edilen sınıf etiketi (Predicted Class).
        - Tahminin doğru olup olmadığı (Correct?).
    - Bu bilgiler bir pandas DataFrame'ine kaydedilmiştir.
2. **Doğru ve Yanlış Sınıflandırmaların Özetlenmesi:**
    - **groupby** kullanılarak doğru ve yanlış sınıflandırmaların kombinasyonları gruplandırılmış ve her grubun sayısı hesaplanmıştır.
    - **Sonuç Tablosu:** Özet tablo, her sınıf kombinasyonu için doğru ve yanlış tahmin sayısını içermektedir.
3. **Doğru ve Yanlış Sınıflandırma Sayılarının Hesaplanması:**
    - **Doğru Sınıflandırmalar:** Tahmini doğru olan örneklerin toplam sayısı.
    - **Yanlış Sınıflandırmalar:** Tahmini yanlış olan örneklerin toplam sayısı.

6. Sınıflandırma Sonuçlarının Görselleştirilmesi

6.1 Doğru Ve Yanlış Sınıflandırmaların Histogramla Gösterimi

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

# Doğru ve yanlış sınıflandırma sayılarının hesaplanması
def classification_summary(y_test, predictions):
    df = pd.DataFrame({'True Class': y_test, 'Predicted Class': predictions})
    df['Correct?'] = df['True Class'] == df['Predicted Class']
    summary = df.groupby(['True Class', 'Correct?']).size().reset_index(name='Count')
    return summary

# Bar grafiği çizimi
def plot_classification_results(summary):
    plt.figure(figsize=(10, 6))
    sns.barplot(
        x="True Class",
        y="Count",
        hue="Correct?",
        data=summary,
        palette={True: "green", False: "red"}
    )
    plt.title("Doğru ve Yanlış Sınıflandırma Sayıları", fontsize=16)
    plt.xlabel("Gerçek Sınıf", fontsize=12)
    plt.ylabel("Sayı", fontsize=12)
    plt.legend(title="Doğru Mu?")
    plt.show()

# Tahminler ve özet oluşturma
predictions = [classify(decision_tree, row) for _, row in X_test.iterrows()]
summary = classification_summary(y_test, predictions)

# Bar grafiğini çiz
plot_classification_results(summary)
```

Karar ağacı modeli tarafından yapılan doğru ve yanlış sınıflandırmalar görselleştirilmiştir. Bar grafiği, modelin hangi sınıflarda başarılı olduğunu ve hangi sınıflarda hata yaptığını daha net bir şekilde anlamamızı sağlar.



1. **Doğru ve Yanlış Sınıflandırmaların Özetlenmesi (classification_summary):**
    - Test veri setindeki tahminler ve gerçek sınıf etiketleri karşılaştırılmıştır.
    - **Özet Tablo:**
        - Her bir gerçek sınıf için doğru ve yanlış sınıflandırmaların toplam sayısı hesaplanmıştır.
    - **Sonuç:** Pandas DataFrame formatında bir özet oluşturulmuştur.
2. **Bar Grafiği Çizimi (plot_classification_results):**
    - Seaborn kütüphanesi kullanılarak, doğru ve yanlış sınıflandırmalar için bir bar grafiği çizilmiştir.
    - **Grafik Özellikleri:**
        - **Renk Kodlaması:**
            - Doğru sınıflandırmalar için **yeşil**.
            - Yanlış sınıflandırmalar için **kırmızı**.
        - **X Ekseni:** Gerçek sınıf etiketleri (True Class).
        - **Y Ekseni:** Sınıflandırma sayıları (Count).
        - **Legend:** Hangi barların doğru veya yanlış sınıflandırmaları temsil ettiğini gösterir.

6.2 Sınıflandırma Sonuçlarının Tablo Olarak Gösterimi

```
from tabulate import tabulate

def classification_table(y_test, predictions):
    # y_test'i listeye çeviriyoruz
    y_test = y_test.tolist()
    data = {
        "Gerçek Sınıf": y_test,
        "Tahmin Edilen Sınıf": predictions,
        "Sonuç": ["✅ Doğru" if y_test[i] == predictions[i] else "❌ Yanlış" for i in range(len(y
_test))]
    }
    table = tabulate(data, headers="keys", tablefmt="fancy_grid", stralign="center", numalign="ce
nter")
    print(table)

# Tablonun gösterimi
classification_table(y_test, predictions)
```

Test veri setindeki sınıflandırma sonuçları, tablo formatında düzenli ve görsel olarak çekici bir şekilde sunulmuştur. **Tabulate** kütüphanesi kullanılarak, doğru ve yanlış sınıflandırmalar işaretlenmiş ve detaylı bir tablo oluşturulmuştur.

Gerçek Sınıf	Tahmin Edilen Sınıf	Sonuç
0	0	✅ Doğru
1	1	✅ Doğru
1	1	✅ Doğru
0	0	✅ Doğru
1	1	✅ Doğru
1	1	✅ Doğru
1	1	✅ Doğru
1	1	✅ Doğru
0	0	✅ Doğru
0	0	✅ Doğru
0	0	✅ Doğru
1	1	✅ Doğru

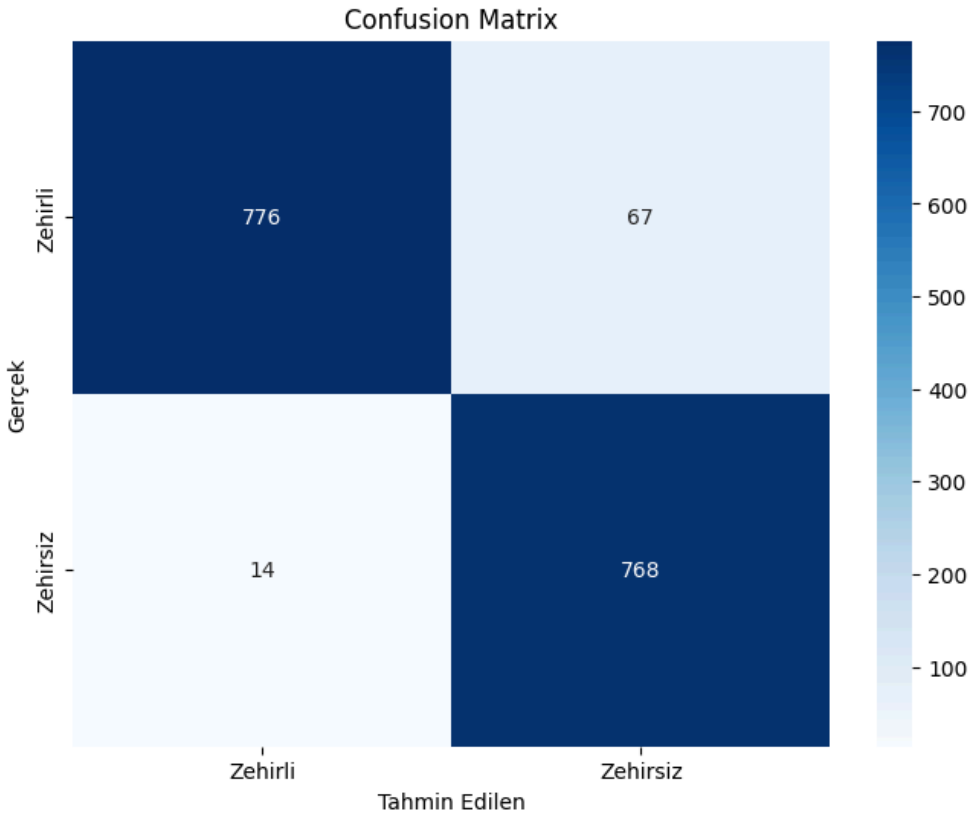
0	0	✅ Doğru
0	1	❌ Yanlış
0	0	✅ Doğru
0	0	✅ Doğru
1	1	✅ Doğru
0	0	✅ Doğru
0	0	✅ Doğru
0	1	❌ Yanlış
0	1	❌ Yanlış

Yukarıdaki görselde oluşturulan tablodaki İlk birkaç örnek gösterilmiştir.

6.3 Confusion Matrix İle Sınıflandırma Performansının Görselleştirilmesi

```
import seaborn as sns
from sklearn.metrics import confusion_matrix

def interactive_confusion_matrix(y_test, predictions):
    cm = confusion_matrix(y_test, predictions)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=['Zehirli', 'Zehirsiz'], ytick
labels=['Zehirli', 'Zehirsiz'])
    plt.xlabel('Tahmin Edilen')
    plt.ylabel('Gerçek')
    plt.title('Confusion Matrix')
    plt.show()

# Gösterim
interactive_confusion_matrix(y_test, predictions)
```

1. **Confusion Matrix Hesaplama:**
    - **confusion_matrix** fonksiyonu kullanılarak, gerçek sınıf etiketleri (y_test) ve tahmin edilen sınıf etiketleri (predictions) arasındaki karşılaştırmalar bir matris formatında özetlenmiştir.
    - **Matristeki Değerler:**
        - **True Positive (TP):** Zehirli mantarın doğru şekilde zehirli olarak sınıflandırılması.
        - **True Negative (TN):** Zehirsiz mantarın doğru şekilde zehirsiz olarak sınıflandırılması.
        - **False Positive (FP):** Zehirsiz mantarın yanlış şekilde zehirli olarak sınıflandırılması.
        - **False Negative (FN):** Zehirli mantarın yanlış şekilde zehirsiz olarak sınıflandırılması.
2. **Heatmap ile Görselleştirme:**
    - **Seaborn Kütüphanesi:**
        - **sns.heatmap:** Karmaşıklık matrisi, görsel olarak daha anlaşılır hale getirilmiştir.
        - **Renk Skalası:** Blues renk paleti kullanılarak doğru ve yanlış sınıflandırmalar vurgulanmıştır.



6.4 Model Eğitim Performansının Görselleştirilmesi

```
def plot_training_performance(epochs, accuracy, loss):
    plt.figure(figsize=(12, 5))

    # Doğruluk Grafiği
    plt.subplot(1, 2, 1)
    plt.plot(epochs, accuracy, color='blue', marker='o')
    plt.title('Model Doğruluk Performansı')
    plt.xlabel('Epoch')
    plt.ylabel('Doğruluk')

    # Kayıp Grafiği
    plt.subplot(1, 2, 2)
    plt.plot(epochs, loss, color='red', marker='o')
    plt.title('Model Kayıp Performansı')
    plt.xlabel('Epoch')
    plt.ylabel('Kayıp')

    plt.tight_layout()
    plt.show()

# Örnek Veriler
epochs = range(1, 11)
accuracy = [0.8, 0.82, 0.85, 0.86, 0.88, 0.9, 0.92, 0.93, 0.94, 0.95]
loss = [0.5, 0.45, 0.4, 0.35, 0.3, 0.28, 0.25, 0.22, 0.2, 0.18]

# Gösterim
plot_training_performance(epochs, accuracy, loss)
```

1. **Fonksiyon Tanımı (plot_training_performance):**
    - Eğitim performansı görselleştirilmek üzere bir fonksiyon oluşturulmuştur.
    - **Parametreler:**
        - **epochs:** Eğitim sürecindeki epoch (iterasyon) sayıları.
        - **accuracy:** Her epoch sonunda ölçülen doğruluk değerleri.
        - **loss:** Her epoch sonunda hesaplanan kayıp (loss) değerleri.
2. **Grafiklerin Özellikleri:**
    - **Doğruluk Grafiği:**
        - X ekseni: Epoch sayısı.

- Y ekseni: Doğruluk oranı.
- Mavi çizgi ile gösterilmiştir.

- **Kayıp Grafiği:**
  - X ekseni: Epoch sayısı.
  - Y ekseni: Kayıp değeri.
  - Kırmızı çizgi ile gösterilmiştir.