SAKARYA UYGULAMALI BİLİMLER ÜNİVERSİTESİ

VERİ MADENCİLİĞİ VİZE SINAVI RAPORU

DERS KODU: BIL 018

ÖĞRETİM GÖREVLİSİ: Dr. Öğr. Üyesi MUHAMMED ALİ NUR ÖZ

AD – SOYAD: Ayşenur YILDIZ

NO: B200109026 FAKÜLTE: Teknoloji Fakültesi

BÖLÜM: Bilgisayar Mühendisliği

ÖZET

Kümeleme analizi, veri biliminde kullanılan önemli bir tekniktir. Bu yöntem, veri setindeki benzer özelliklere sahip nesneleri gruplandırarak onları aynı kümeye yerleştirme sürecidir. Amaç, veri setindeki öğeler arasındaki benzerlikleri ve farklılıkları keşfetmek ve belirli bir kümeye ait öğelerin birbirlerine ne kadar benzediğini ve farklılık gösterdiğini belirlemektir.

Kümeleme analizi, benzer özelliklere sahip öğeleri aynı kümeye yerleştirerek kümeler arasındaki farklılıkları maksimize etmeyi hedefler. Bu, kümeler içinde benzerliklerin yüksek, farklı kümeler arasında ise belirgin farklılıkların olmasını amaçlar. Yani, bir kümeye ait veri noktaları birbirine yakın olabilirken, farklı kümeler arasındaki noktalar arasında belirgin farklar bulunması istenir. Bu yaklaşım, veri setlerindeki yapıları, desenleri ve ilişkileri anlamak için önemli bir araçtır.

Kümeleme analizinde doğru küme sayısını seçmek kritik bir adımdır ve genellikle sonuçları önemli ölçüde etkiler. Doğru küme sayısını belirlemek genellikle net olmayabilir ve çeşitli teknikler kullanılarak yapılır. Dirsek kuralı (Elbow yöntemi),GAP istatistiği, Silhouette Score, Davies-Bouldin Skoru, BIC Metodu gibi teknikler, doğru küme sayısını belirlemede sıkça kullanılan yöntemlerdir.

GİRİŞ

```
[ ]  import warnings
     warnings.filterwarnings("ignore")
     # Import Libraries
     import numpy as np
     import pandas as pd
     import seaborn as sns
     from matplotlib import pyplot as plt
     import plotly.express as px
     import missingno as msno
     import warnings
     warnings.filterwarnings("ignore", category=DeprecationWarning)
     from sklearn.cluster import KMeans
```

Bu kod bloğu, veri analizi ve kümeleme işlemleri için gerekli olan kütüphaneleri ve modülleri çağırır.

```
[ ]  from sklearn import datasets
     X, y = datasets.make_blobs(n_samples=1000, centers=4,
     cluster_std=[np.random.rand()*2, np.random.rand()*2,
     np.random.rand()*2, np.random.rand()*2])
```

Bu kod kullanılarak make_blobs fonksiyonuyla 1000 örnek oluşturur. Bu örnekler, 4 farklı merkeze ve belirli bir standart sapmaya sahip rasgele verilerden oluşur.

```
# Veri setini DataFrame'e dönüştürme
data = {'Feature_1': X[:, 0], 'Feature_2': X[:, 1], 'Label': y}
df = pd.DataFrame(data)
print(df)
```

```
     Feature_1  Feature_2  Label
0    -5.575705  -4.371711      1
1     8.551318   1.653319      3
2     5.831439  -5.211199      0
3    -2.252943  -3.439814      2
4     4.846332  -5.378027      0
..         ...        ...    ...
995   8.913572  -6.669849      0
996  -5.294589  -3.478012      1
997   1.008354  -1.121809      2
998  -1.928507  -3.508971      2
999  -4.900013  -4.818066      1

[1000 rows x 3 columns]
```

Bu kod, bir NumPy dizisinden (X) ve etiketlerden (y) oluşan bir veri setini Pandas kütüphanesinin DataFrame yapısına dönüştürmek için kullanılır.

```
# Genel Exploration for Dataset
def check_df(dataframe, head=5):
    print("##################### Shape #####################")
    print(dataframe.shape)
    print("##################### Types #####################")
    print(dataframe.dtypes)
    print("##################### Head #####################")
    print(dataframe.head(head))
    print("##################### Tail #####################")
    print(dataframe.tail(head))
```

```
check_df(df)
```

```
##################### Shape #####################
(1000, 3)
##################### Types #####################
Feature_1    float64
Feature_2    float64
Label          int64
dtype: object
##################### Head #####################
   Feature_1  Feature_2  Label
0  -5.575705  -4.371711      1
1   8.551318   1.653319      3
2   5.831439  -5.211199      0
3  -2.252943  -3.439814      2
4   4.846332  -5.378027      0
##################### Tail #####################
     Feature_1  Feature_2  Label
995   8.913572  -6.669849      0
996  -5.294589  -3.478012      1
997   1.008354  -1.121809      2
998  -1.928507  -3.508971      2
999  -4.900013  -4.818066      1
```

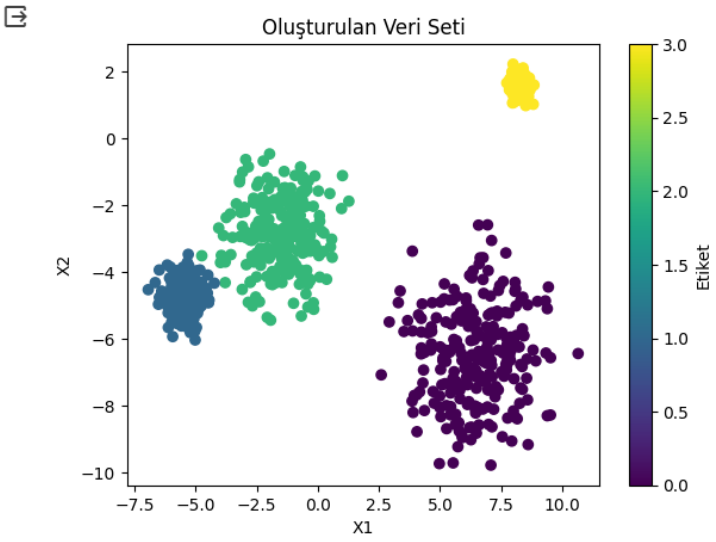
Bu kod bloğu, veri seti keşfi için kullanılabilecek genel bir işlev olan check_df() adlı bir fonksiyon tanımlar.

- def check_df(dataframe, head=5):: Bu satır, check_df() adında bir fonksiyon tanımlar. Bu fonksiyon, dataframe adında bir argüman alır ve varsayılan olarak ilk 5 satırı görüntülemek üzere head parametresine sahiptir.
- print(dataframe.shape): Bu satır, DataFrame'in şeklini yazdırır. Bu, kaç satır ve sütundan oluştuğunu gösterir.
- print(dataframe.dtypes): Bu satır, DataFrame'in sütunlarının veri tiplerini yazdırır.

- print(dataframe.head(head)): Bu satır, DataFrame'in başlangıcını, belirtilen head sayısı kadar satırı gösterir.
- print(dataframe.tail(head)): Bu satır, DataFrame'in sonunu, belirtilen head sayısı kadar satırı gösterir.

Bu fonksiyon, veri setinin başından ve sonundan birkaç satırı ve sütunların genel yapısını incelemek için kullanılabilir. Sütun tipleri, satır ve sütun sayısı gibi genel bilgiler hızlıca kontrol edilebilir.

```
# Veriyi görselleştirme
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis')
plt.title('Oluşturulan Veri Seti')
plt.xlabel('X1')
plt.ylabel('X2')
plt.colorbar(label='Etiket')
plt.show()
```



Bu kod, veri setini scatter plot (nokta grafiği) olarak görselleştirmek için kullanılır. Bu kod, veri setindeki her bir noktayı X1 ve X2 özellikleriyle gösteren bir nokta grafiği çizer. Noktaların renkleri, veri noktalarının ait olduğu etiketlere (y) göre belirlenir. Bu tür görselleştirmeler, veri setinin yapısını anlamak ve farklı etiketlere sahip veri noktalarının dağılımını görsel olarak gözlemlemek için kullanılır.

SORULAR VE ÇÖZÜMLERİ

SORU 1.

Aşağıdaki kod parçası üretilen1000örneği k-means kümeleme yöntemi ile kümeleyiniz.

From sklearn import datasets

import numpy as np

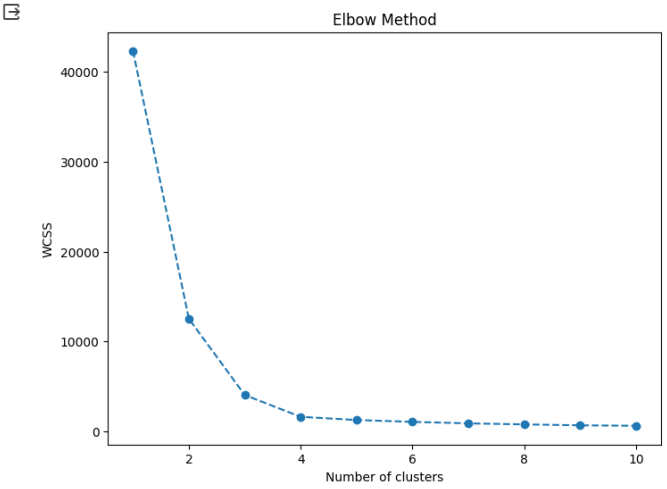
X, y = datasets.make_blobs(n_samples=1000, centers=4, cluster_std=[np.random.rand()*2,np.random.rand()*2, np.random.rand()*2, np.random.rand()*2])

- Geliştirdiğiniz programda en optimal küme sayısını (k)otomatik hesaplansın. Programın arkasındaki mantığı detaylıca açıklayınız.

En optimal küme sayısını (k) hesaplamak için pek çok yöntem kullanılmaktadır. Dirsek kuralı (Elbow yöntemi),GAP istatistiği, Silhouette Score, Davies-Bouldin Skoru, BIC Metodu gibi teknikler, doğru küme sayısını belirlemede sıkça kullanılan yöntemlerdir. Bu çalışmada Elbow yöntemi,GAP istatistiği, Silhouette skoru ve BIC Metodudu kullanılmıştır. Aşağıdaki tabloda kullanılan yöntemler artı ve eksi yönleri ile açıklanmıştır.

Yöntem	Açıklama	Artılar	Eksiler
Dirsek Kuralı	Küme sayısını belirlemek için WCSS'yi (Within-Cluster Sum of Squares) küme sayısına karşı çizilen grafiğe dayanarak belirler. Grafikte dirseği gösteren nokta, optimal küme sayısının olduğu noktadır	Basit ve kolay anlaşılır.	Dirseği belirlemek bazen net olmayabilir.
GAP İstatistiği	Küme sayısını belirlerken farklı küme sayılarının WCSS değerlerini, simüle edilmiş veriyle karşılaştırır. Gerçek verinin ne kadar simüle edilen veriyle farklı olduğunu ölçer.	İyi bir performans gösterir.	Optimal küme sayısını belirlemekte hassasiyet gerektirir.
Silhouette Skoru	Her örneğin kendi kümesindeki yoğunluğunu ve diğer kümelere olan uzaklığını ölçer. +1 ile -1 arasında bir değer alır, yüksek değerler daha iyi kümelemeyi gösterir.	Anlaşılır bir metrik, geniş kullanım alanı.	Düzgün dağılmamış veri setlerinde yanıltıcı olabilir.
BIC Metodu	Bayes Bilgi Kriteri, olasılığa dayalı bir model seçme kriteridir. Her bir modelin karmaşıklığı ve uyumluluğunu değerlendirerek en uygun modeli seçer.	Model seçiminde kullanışlı ve güvenilir.	Karmaşık modellerde daha zorlu bir analiz gerektirir.

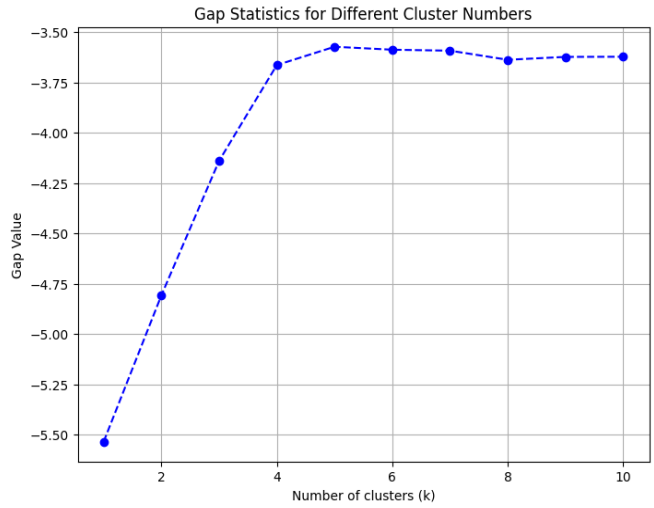
```python
# Calculate WCSS for different values of k
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
    kmeans.fit(df[['Feature_1', 'Feature_2']])
    wcss.append(kmeans.inertia_)

# Plotting the Elbow Method graph
plt.figure(figsize=(8, 6))
plt.plot(range(1, 11), wcss, marker='o', linestyle='--')
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



Bu kod, Dirsek Yöntemini (Elbow Method) kullanarak, farklı küme sayıları için Kümeleme İçi Kareler Toplamı (WCSS - Within-Cluster Sum of Squares) değerlerini hesaplar ve bunları bir grafik üzerinde gösterir.

- **wcss = []**: Boş bir liste oluşturulur. Bu liste, her bir küme sayısı için hesaplanacak olan WCSS değerlerini saklar.
- **for i in range(1, 11):**: Küme sayısını 1'den 10'a kadar (10 dahil değil) değişen değerlerde denemek için döngü oluşturulur.
- **kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)**: Her iterasyonda KMeans kümeleme algoritması çağrılır. **n_clusters=i** ile farklı küme sayıları denenecek şekilde ayarlanır. **init='k-means++'** ile küme merkezlerinin başlangıç noktaları daha etkili bir şekilde seçilir. **random_state=42** ile tekrarlanabilirlik için rastgelelik kontrol altına alınır.
- **kmeans.fit(df[['Feature_1', 'Feature_2']])**: KMeans modeli, DataFrame içindeki belirtilen özellikler (**Feature_1** ve **Feature_2**) üzerinde eğitilir.
- **wcss.append(kmeans.inertia_)**: Her küme sayısı için WCSS değeri (**kmeans.inertia_**) hesaplanır ve **wcss** listesine eklenir.
- Grafik oluşturma adımları: **plt.figure()**, **plt.plot()**, **plt.title()**, **plt.xlabel()**, **plt.ylabel()** ve **plt.show()** fonksiyonları kullanılarak WCSS değerlerinin küme sayısına göre değişimini gösteren bir grafik çizilir.

```python
def calculate_gap(data):
    gaps = []
    for k in range(1, 11):  # Küme sayısını 1 ile 10 arasında değerlendiriyoruz
        kmeans = KMeans(n_clusters=k)
        kmeans.fit(data)
        gap = compute_gap(kmeans, data, k)
        gaps.append(gap)
    return gaps

def compute_gap(kmeans, data, k):
    # Hesaplama yapılacak kümeleme sonuçlarını elde ediyoruz
    cluster_dispersion = kmeans.inertia_

    # Rasgele veri oluşturup k-means ile kümeleme yaparak beklenen dağılımı hesaplıyoruz
    reference_dispersion = []
    for i in range(5):  # Rasgele 5 defa k-means işlemi uygulayarak beklenen dağılımı hesaplıyoruz
        random_data = np.random.random_sample(size=data.shape)  # Rasgele veri oluşturuyoruz
        random_kmeans = KMeans(n_clusters=k)
        random_kmeans.fit(random_data)
        reference_dispersion.append(random_kmeans.inertia_)

    # Beklenen dağılımın ortalamasını alıyoruz
    reference_dispersion_mean = np.mean(reference_dispersion)

    # Gap istatistiğini hesaplıyoruz
    gap = np.log(reference_dispersion_mean) - np.log(cluster_dispersion)
    return gap


gap_values = calculate_gap(df[['Feature_1', 'Feature_2']])
# Gap istatistiğini grafik olarak göster
plt.figure(figsize=(8, 6))
plt.plot(range(1, 11), gap_values, marker='o', linestyle='--', color='b')
plt.xlabel('Number of clusters (k)')
plt.ylabel('Gap Value')
plt.title('Gap Statistics for Different Cluster Numbers')
plt.grid(True)
plt.show()

print("Gap istatistiği değerleri:", gap_values)
```




```
Gap istatistiği değerleri: [-5.535206065718 39, -4.808063964132455, -4.139685722455987, -3.6639585060336417, -3.5730139634871962, -3.5877095768938414, -3.5
```

Bu kod **calculate_gap** ve **compute_gap** fonksiyonlarını kullanarak gap istatistiğini hesaplar ve grafiğini çizer.

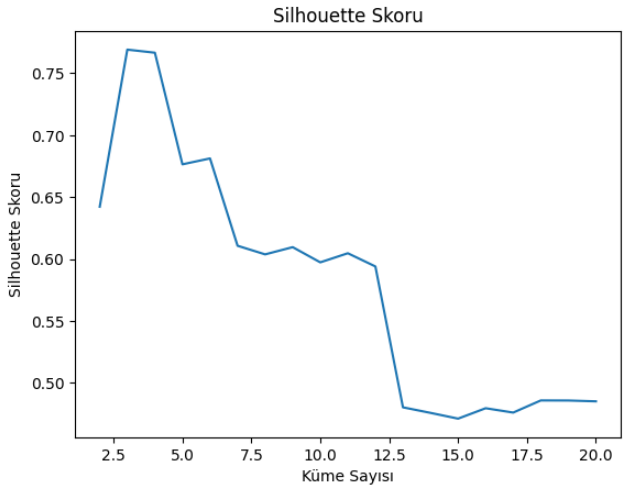
- **calculate_gap** fonksiyonu, farklı küme sayıları için gap değerlerini hesaplamak için bir döngü içinde **compute_gap** fonksiyonunu çağırır ve bu değerleri bir liste olan **gaps** içine kaydeder.
- **compute_gap** fonksiyonu, verilen bir KMeans modeli ve veri üzerinde beklenen dağılımı hesaplar. İlk olarak, **cluster_dispersion** değişkeni, verilen KMeans modelinin

inertia değerini yani kümeleme içi kareler toplamını alır. Ardından, rasgele veriler üzerinde KMeans modeli uygulayarak beklenen dağılım için **reference_dispersion** listesini oluşturur. Son olarak, bu beklenen dağılımların ortalamasını hesaplayarak gap istatistiğini elde eder.

- **gap_values = calculate_gap(df[['Feature_1', 'Feature_2']])**: Bu satır, gap istatistiği değerlerini hesaplar ve **gap_values** adlı bir liste olarak saklar.
- **plt.figure(figsize=(8, 6))**, **plt.plot()**, **plt.xlabel()**, **plt.ylabel()**, **plt.title()**, **plt.grid()**, **plt.show()**: Bu adımlar, hesaplanan gap istatistiği değerlerini **gap_values** listesinden alarak bir grafik oluşturur ve bu grafik üzerinde kümelerin sayısına (**k**) karşılık gelen gap değerlerini gösterir.

```python
from sklearn.metrics import silhouette_score
# Silhouette Skoru hesaplama
silhouette_scores = []
for n_clusters in range(2, 21):
    kmeans = KMeans(n_clusters=n_clusters, random_state=0)
    cluster_labels = kmeans.fit_predict(df)
    silhouette_avg = silhouette_score(df, cluster_labels)
    print("For n_clusters =", n_clusters, "The average silhouette_score is :", silhouette_avg)
    silhouette_scores.append(silhouette_avg)

# Silhouette Skoru grafiği
plt.plot(range(2, 21), silhouette_scores)
plt.title('Silhouette Skoru')
plt.xlabel('Küme Sayısı')
plt.ylabel('Silhouette Skoru')

# En iyi performansı gösteren küme sayısını bulma
best_n_clusters_silhouette  = silhouette_scores.index(max(silhouette_scores)) + 2  # En iyi skorun indeksi + 2
print(f"En iyi küme sayısı: {best_n_clusters_silhouette }")
```

```
For n_clusters = 2 The average silhouette_score is : 0.6422283504373048
For n_clusters = 3 The average silhouette_score is : 0.769142673602248
For n_clusters = 4 The average silhouette_score is : 0.7666526932927155
For n_clusters = 5 The average silhouette_score is : 0.6764720776611876
For n_clusters = 6 The average silhouette_score is : 0.6812361624459682
For n_clusters = 7 The average silhouette_score is : 0.6106791613412095
For n_clusters = 8 The average silhouette_score is : 0.6037171561518866
For n_clusters = 9 The average silhouette_score is : 0.609551147717342
For n_clusters = 10 The average silhouette_score is : 0.5972722336117842
For n_clusters = 11 The average silhouette_score is : 0.6046324939569658
For n_clusters = 12 The average silhouette_score is : 0.5939577490125497
For n_clusters = 13 The average silhouette_score is : 0.4801377283830094
For n_clusters = 14 The average silhouette_score is : 0.47565614185834953
For n_clusters = 15 The average silhouette_score is : 0.47099899113485405
For n_clusters = 16 The average silhouette_score is : 0.47941012566023883
For n_clusters = 17 The average silhouette_score is : 0.4758991689168882
For n_clusters = 18 The average silhouette_score is : 0.48570412222891707
For n_clusters = 19 The average silhouette_score is : 0.48564965251443765
For n_clusters = 20 The average silhouette_score is : 0.48499060675049466
En iyi küme sayısı: 3
```

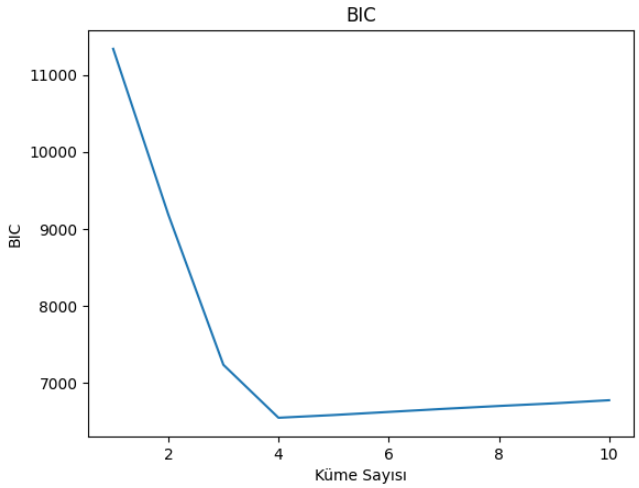


Bu kod, Silhouette skorunu kullanarak farklı küme sayıları için kümeleme performansını değerlendirir ve en iyi performans gösteren küme sayısını belirlemeye çalışır.

- **silhouette_scores = []**: Boş bir liste oluşturulur. Bu liste, her bir küme sayısı için hesaplanacak olan Silhouette skorlarını saklar.
- **for n_clusters in range(2, 21):**: 2 ile 20 arasındaki (20 dahil değil) farklı küme sayıları için döngü oluşturulur.
- **kmeans = KMeans(n_clusters=n_clusters, random_state=0)**: Her bir küme sayısı için bir KMeans modeli oluşturulur.
- **cluster_labels = kmeans.fit_predict(df)**: Belirtilen veri seti (**df**) üzerinde KMeans modeli uygulanır ve her bir örnek için hangi küme etiketine ait olduğunu belirleyen etiketler (**cluster_labels**) elde edilir.
- **silhouette_avg = silhouette_score(df, cluster_labels)**: Her bir küme sayısı için Silhouette skoru hesaplanır. Bu skor, kümeleme algoritmasının her bir veri noktasının kendi kümesine ne kadar uyumlu olduğunu ölçer.
- **silhouette_scores.append(silhouette_avg)**: Her bir küme sayısı için hesaplanan Silhouette skoru, **silhouette_scores** listesine eklenir.
- **plt.plot(range(2, 21), silhouette_scores)**: Oluşturulan Silhouette skorlarını farklı küme sayılarına karşılık gelen x ekseni değerleriyle birlikte grafikte çizer.
- **best_n_clusters_silhouette = silhouette_scores.index(max(silhouette_scores)) + 2**: En yüksek Silhouette skorunu alır ve bunun indeksini bulur. Bu indeks +2 ile kullanılarak, en iyi performansı gösteren küme sayısını elde eder. +2, range fonksiyonunun kullanıldığı için 2'den başlaması gerektiği için eklenmiştir.

```python
# GMM modeli oluşturma ve BIC değerini hesaplama
from sklearn.mixture import GaussianMixture
import numpy as np

# Veri seti için BIC değerlerini saklayacak bir liste
bic_values = []  # BIC: Bayesian Information Criterion -Bayes Bilgi Kriteri

# Farklı küme sayıları için GMM modeli oluşturma ve BIC değerlerini hesaplama
for n_components in range(1, 11):  # 1 ile 10 arasında küme sayısı
    gmm = GaussianMixture(n_components=n_components, random_state=0)  # GMM modeli oluşturma
    gmm.fit(df[['Feature_1', 'Feature_2']])  # BIC değerini hesaplama
    bic_values.append(gmm.bic(df[['Feature_1', 'Feature_2']]))  # BIC değerini saklama

# BIC grafiği
plt.plot(range(1, 11), bic_values)
plt.title('BIC')
plt.xlabel('Küme Sayısı')
plt.ylabel('BIC')

# En düşük BIC değerine sahip olan küme sayısını belirleme
best_n_components = np.argmin(bic_values) + 1  # +1 çünkü range 1'den başlıyor
print(f"En uygun küme sayısı: {best_n_components}")  # En düşük BIC değerine sahip olan küme sayısı
```

```
En uygun küme sayısı: 4
```



Bu kod, farklı küme sayıları için Gaussian Mixture Model (GMM) oluşturarak ve her bir model için Bayesian Information Criterion (BIC) değerini hesaplayarak BIC grafiğini çizer.

- **from sklearn.mixture import GaussianMixture**: GMM (Gaussian Mixture Model) için gerekli kütüphane eklenir.
- **bic_values = []**: Boş bir liste oluşturulur. Bu liste, her bir küme sayısı için hesaplanacak olan BIC değerlerini saklayacak.
- **for n_components in range(1, 11):**: 1 ile 10 arasındaki farklı küme sayıları için döngü oluşturulur.
- **gmm = GaussianMixture(n_components=n_components, random_state=0)**: Her bir küme sayısı için bir GMM modeli oluşturulur.
- **gmm.fit(df[['Feature_1', 'Feature_2']])**: Veri seti üzerinde GMM modeli eğitilir.
- **bic_values.append(gmm.bic(df[['Feature_1', 'Feature_2']]))**: Her bir küme sayısı için hesaplanan BIC değeri (**gmm.bic**) **bic_values** listesine eklenir.
- **plt.plot(range(1, 11), bic_values)**: Oluşturulan BIC değerlerini farklı küme sayılarına karşılık gelen x ekseni değerleriyle birlikte grafikte çizer.

```
# Determine the final number of clusters
if best_n_clusters_silhouette >= best_n_components:
    k = best_n_clusters_silhouette
else:
    k = best_n_components

print(f"Optimal küme sayısı (k): {k}")
```

```
Optimal küme sayısı (k): 4
```

Bu kod, Silhouette skoruyla ve BIC ile önceden hesaplanmış en iyi küme sayılarını karşılaştırarak, en sonunda kullanılacak optimal küme sayısını belirler.

- Yukarıda çözümünüzün devamında girdi olarak bir nokta kabul eden program yazınız. Program bu noktayı sınıflardan birine veya gerektiğinde ise bir anomali olarak (yani hiçbir sınıfa ait olmayan) sınıflandırmalıdır. Arkasındaki mantığı detaylıca açıklayınız.

```python
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
import numpy as np

# KMeans ve GMM modellerini oluşturma
kmeans = KMeans(n_clusters=k, random_state=0)
gmm = GaussianMixture(n_components=k, random_state=0)

# Veri setini belirlenen k değeriyle kümeleme
kmeans_clusters = kmeans.fit_predict(df[['Feature_1', 'Feature_2']])
gmm_clusters = gmm.fit_predict(df[['Feature_1', 'Feature_2']])

# Anormal sınıfın etiketini başlangıçta -1 olarak atıyoruz
anomalous_class_label = -1

# Her bir veri noktası için küme etiketlerini kontrol ederek anormal sınıfı belirleme
anomalous_points = []  # Anormal sınıfın etiketini belirlemek için kullanılacak veri noktaları
for i in range(len(df)):  # Her bir veri noktasını kontrol etme
    if kmeans_clusters[i] == -1 or gmm_clusters[i] == -1:  # Eğer biri ya da her ikisi de -1 (anormal sınıf) ise
        anomalous_points.append(i)  # Anormal sınıfın etiketini belirlemek için kullanılacak veri noktalarını saklama

# Anormal sınıfın etiketini belirliyoruz
for idx in anomalous_points:  # Anormal sınıfın etiketini belirlemek için kullanılacak veri noktalarını kontrol etme
    kmeans_clusters[idx] = anomalous_class_label  # KMeans küme etiketini anormal sınıf etiketi ile değiştiriyoruz
    gmm_clusters[idx] = anomalous_class_label  # GMM küme etiketini anormal sınıf etiketi ile değiştiriyoruz

# Her bir veri noktasının küme etiketlerini ekrana yazdırma
for i in range(len(df)):  # Her bir veri noktasını kontrol etme
    print(f"Data point {i} - KMeans Cluster: {kmeans_clusters[i]}, GMM Cluster: {gmm_clusters[i]}")  # Her bir veri noktası

# Anormal sınıfı ekrana yazdırma
print(f"Anomalous Class Label: {anomalous_class_label}")
```

```
Data point 971 - KMeans Cluster: 3, GMM Cluster: 3
Data point 972 - KMeans Cluster: 1, GMM Cluster: 0
Data point 973 - KMeans Cluster: 1, GMM Cluster: 0
Data point 974 - KMeans Cluster: 2, GMM Cluster: 2
Data point 975 - KMeans Cluster: 0, GMM Cluster: 1
Data point 976 - KMeans Cluster: 2, GMM Cluster: 2
Data point 977 - KMeans Cluster: 2, GMM Cluster: 2
Data point 978 - KMeans Cluster: 2, GMM Cluster: 2
Data point 979 - KMeans Cluster: 3, GMM Cluster: 0
Data point 980 - KMeans Cluster: 0, GMM Cluster: 1
Data point 981 - KMeans Cluster: 1, GMM Cluster: 0
Data point 982 - KMeans Cluster: 3, GMM Cluster: 3
Data point 983 - KMeans Cluster: 3, GMM Cluster: 3
Data point 984 - KMeans Cluster: 2, GMM Cluster: 2
Data point 985 - KMeans Cluster: 0, GMM Cluster: 1
Data point 986 - KMeans Cluster: 0, GMM Cluster: 1
Data point 987 - KMeans Cluster: 2, GMM Cluster: 2
Data point 988 - KMeans Cluster: 1, GMM Cluster: 0
Data point 989 - KMeans Cluster: 3, GMM Cluster: 3
Data point 990 - KMeans Cluster: 1, GMM Cluster: 0
Data point 991 - KMeans Cluster: 0, GMM Cluster: 1
Data point 992 - KMeans Cluster: 1, GMM Cluster: 0
Data point 993 - KMeans Cluster: 1, GMM Cluster: 0
Data point 994 - KMeans Cluster: 2, GMM Cluster: 2
Data point 995 - KMeans Cluster: 2, GMM Cluster: 2
Data point 996 - KMeans Cluster: 3, GMM Cluster: 3
Data point 997 - KMeans Cluster: 1, GMM Cluster: 0
Data point 998 - KMeans Cluster: 1, GMM Cluster: 0
Data point 999 - KMeans Cluster: 3, GMM Cluster: 3
Anomalous Class Label: -1
```

Bu kod, KMeans ve Gaussian Mixture Model (GMM) kullanarak veri noktalarını kümelere ayırdıktan sonra anormal veri noktalarını belirler.

1. **KMeans ve GMM Modellerini Oluşturma:**
   - **kmeans = KMeans(n_clusters=k, random_state=0)**: Belirlenen **k** değeri ile KMeans modeli oluşturulur.
   - **gmm = GaussianMixture(n_components=k, random_state=0)**: Aynı **k** değeri ile GMM modeli oluşturulur.
2. **Veri Setini Belirlenen K Değeriyle Kümeleme:**
   - **kmeans_clusters = kmeans.fit_predict(df[['Feature_1', 'Feature_2']])**: Veri seti, KMeans modeli kullanılarak belirlenen **k** değeriyle kümelere ayrılır.

- **gmm_clusters = gmm.fit_predict(df[['Feature_1', 'Feature_2']])**: Veri seti, GMM modeli kullanılarak belirlenen **k** değeriyle kümelere ayrılır.

3. **Anormal Sınıfın Belirlenmesi:**
   - **anomalous_class_label = -1**: Başlangıçta anormal sınıf etiketi **-1** olarak atanır.
   - **anomalous_points = []**: Anormal sınıfın etiketini belirlemek için kullanılacak veri noktalarını saklamak için boş bir liste oluşturulur.
   - **for i in range(len(df)):**: Her bir veri noktasını kontrol ederek, eğer KMeans veya GMM tarafından belirlenen küme etiketi anormal sınıf etiketine denk geliyorsa, bu noktalar **anomalous_points** listesine eklenir.
4. **Anormal Sınıfın Etiketlenmesi:**
   - **for idx in anomalous_points:**: **anomalous_points** listesindeki veri noktalarının küme etiketleri anormal sınıf etiketi ile değiştirilir.
5. **Veri Noktalarının Küme Etiketlerinin Yazdırılması:**
   - Her bir veri noktasının K-Means ve GMM tarafından belirlenen küme etiketleri ekrana yazdırılır.
   - Anormal sınıfın etiketi de ekrana yazdırılır.

   Bu işlem, K-Means ve GMM modelleri ile belirlenen küme etiketlerini kontrol ederek anormal sınıfın etiketini belirlemek için yapılmıştır. Eğer bir veri noktası hem K-Means hem de GMM tarafından anormal sınıf olarak belirlenmişse, bu nokta anormal olarak işaretlenir. Bu işlem, her iki modelin de aynı noktayı anormal olarak işaretlemesi durumunda anormal sınıf etiketini belirlemek için yapılmış bir kontrol mekanizması olarak düşünülebilir.

SORU 2.

Birinci sorudaki veri setini hiyerarşik kümeleme ile çözünüz. K-means ile elde ettiğiniz sonuca en yakın sonucu elde etmek için hangi bağlantıyı kullanabiliriz.

```python
from sklearn.cluster import AgglomerativeClustering

# KMeans ve GMM modellerini oluşturma
kmeans = KMeans(n_clusters=k, random_state=0)
gmm = GaussianMixture(n_components=k, random_state=0)

# Veri setini belirlenen k değeriyle kümeleme
kmeans_clusters = kmeans.fit_predict(df[['Feature_1', 'Feature_2']])
gmm_clusters = gmm.fit_predict(df[['Feature_1', 'Feature_2']])

# KMeans ve GMM'den gelen küme etiketlerini kullanarak hiyerarşik kümeleme yapma
# Bağlantı metodu olarak 'ward' kullanılabilir, ancak farklı bağlantı yöntemlerini de deneyebilirsiniz (örneğin, 'single', 'complete', 'average')
agg_cluster = AgglomerativeClustering(n_clusters=k, linkage='ward')
agg_clusters = agg_cluster.fit_predict(df[['Feature_1', 'Feature_2']])

# Her bir veri noktasının küme etiketlerini ekrana yazdırma
for i in range(len(df)):
    print(f"Data point {i} - KMeans Cluster: {kmeans_clusters[i]}, GMM Cluster: {gmm_clusters[i]}, Agglomerative Cluster: {agg_clusters[i]}")
```

Bu kod, öncelikle KMeans ve GMM (Gaussian Mixture Model) kullanarak veri setini kümelere ayırıyor. Sonrasında, bu iki algoritmadan elde edilen küme etiketlerini kullanarak hiyerarşik kümeleme (Agglomerative Clustering) yapmaktadır.

Agglomerative Clustering, veri noktalarını birbirine yakınlık veya uzaklık ölçüsüne göre birleştirerek hiyerarşik bir yapı oluşturan bir kümeleme yöntemidir. Bu yöntemde, veri noktaları önce tekli nokta olarak başlar ve daha sonra birleştirilerek belirli bir kritere (mesafe, benzerlik vb.) göre belirli bir yapıya ulaşılır.

Bu kodda **linkage** parametresi 'ward' olarak belirlenmiş. 'Ward' metodu, birleştirme kriteri olarak küme içi benzerliklerin varyansını kullanır ve genellikle daha dengeli ve küçük, homojen kümeler elde etmek için tercih edilir.

Sonuç olarak, her bir veri noktası için KMeans, GMM ve Agglomerative Clustering algoritmalarının ürettiği küme etiketleri ekrana yazdırılır. Bu sayede her algoritmanın veri setini farklı şekillerde böldüğü ve hangi algoritmanın hangi veri noktalarını hangi kümelere atadığı gözlemlenebilir.

```
Data point 0 - KMeans Cluster: 3, GMM Cluster: 3, Agglomerative Cluster: 3
Data point 1 - KMeans Cluster: 0, GMM Cluster: 1, Agglomerative Cluster: 2
Data point 2 - KMeans Cluster: 2, GMM Cluster: 2, Agglomerative Cluster: 0
Data point 3 - KMeans Cluster: 1, GMM Cluster: 0, Agglomerative Cluster: 1
Data point 4 - KMeans Cluster: 2, GMM Cluster: 2, Agglomerative Cluster: 0
Data point 5 - KMeans Cluster: 2, GMM Cluster: 2, Agglomerative Cluster: 0
Data point 6 - KMeans Cluster: 0, GMM Cluster: 1, Agglomerative Cluster: 2
Data point 7 - KMeans Cluster: 1, GMM Cluster: 0, Agglomerative Cluster: 1
Data point 8 - KMeans Cluster: 3, GMM Cluster: 3, Agglomerative Cluster: 3
Data point 9 - KMeans Cluster: 3, GMM Cluster: 0, Agglomerative Cluster: 1
Data point 10 - KMeans Cluster: 2, GMM Cluster: 2, Agglomerative Cluster: 0
Data point 11 - KMeans Cluster: 1, GMM Cluster: 0, Agglomerative Cluster: 1
Data point 12 - KMeans Cluster: 2, GMM Cluster: 2, Agglomerative Cluster: 0
Data point 13 - KMeans Cluster: 1, GMM Cluster: 0, Agglomerative Cluster: 1
Data point 14 - KMeans Cluster: 2, GMM Cluster: 2, Agglomerative Cluster: 0
Data point 15 - KMeans Cluster: 3, GMM Cluster: 3, Agglomerative Cluster: 3
Data point 16 - KMeans Cluster: 1, GMM Cluster: 0, Agglomerative Cluster: 1
Data point 17 - KMeans Cluster: 0, GMM Cluster: 1, Agglomerative Cluster: 2
Data point 18 - KMeans Cluster: 2, GMM Cluster: 2, Agglomerative Cluster: 0
Data point 19 - KMeans Cluster: 0, GMM Cluster: 1, Agglomerative Cluster: 2
Data point 20 - KMeans Cluster: 3, GMM Cluster: 0, Agglomerative Cluster: 1
Data point 21 - KMeans Cluster: 3, GMM Cluster: 3, Agglomerative Cluster: 3
```

"Bağlantı yöntemi" veya "linkage method", hiyerarşik kümeleme (agglomerative clustering) sırasında kullanılan bir yöntemdir. Bu yöntem, veri noktalarını birbirine bağlama veya birleştirme şeklini belirler.

- **Ward Bağlantı Yöntemi (Ward's Method):** Küme içi varyansın artışını en aza indirmek için kümeleme işlemi yapar. Yeni bir küme oluşturulurken, oluşturulan yeni kümenin diğer kümelere göre küme içi varyansı en az artacak şekilde birleştirme yapar. Bu yöntem, homojen kümeleme sonuçları elde etmeyi amaçlar.

Ward bağlantı yöntemi, kümeleme işleminde birleştirme adımlarında, yeni oluşturulan kümenin diğer kümelere göre küme içi benzerliğini artırmaya çalışır. Bu şekilde, küme içindeki veri noktalarının birbirine benzerliği maksimize edilerek homojen kümeler elde edilmesi hedeflenir. Bu yöntem genellikle küme içi varyansı azaltmak ve homojen kümeler elde etmek için tercih edilir. K-Means ile elde edilen sonuca benzer kümeleme sonuçları elde etmek için sıklıkla kullanılır.

- Hiyerarşik kümeleme yapıldıktan sonra optimal küme sayısını (k) otomatik hesaplayan bir fonksiyon geliştirin. Arkasındaki mantığı detaylıca açıklayınız.

```python
import scipy.cluster.hierarchy as sch

def optimal_cluster_count(df):
    # Dendrogram oluşturma
    dendrogram = sch.dendrogram(sch.linkage(df, method='ward'))

    # Kesme seviyesinin belirlenmesi
    threshold_distance = 0.7 * max(dendrogram['dcoord'][1])  # Örnek bir eşik değeri (0.7 oranında maksimum mesafe)

    # Kesme seviyesini görsel olarak belirleme
    plt.axhline(y=threshold_distance, color='black', linestyle='--')
    plt.title('Dendrogram')
    plt.xlabel('Veri Noktaları')
    plt.ylabel('Uzaklık')
    plt.show()

    # Kesme seviyesindeki küme sayısını belirleme
    optimal_clusters = sum(d > threshold_distance for d in dendrogram['dcoord'][1])

    return optimal_clusters

# Veri setini kullanarak optimal küme sayısını hesaplama
optimal_k = optimal_cluster_count(df[['Feature_1', 'Feature_2']])
print(f"Optimal Küme Sayısı: {optimal_k}")
```

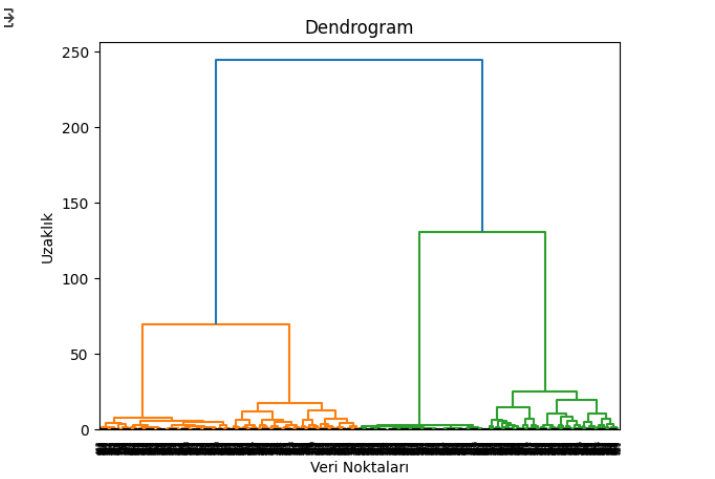
Bu kod, verilen bir veri seti için optimal küme sayısını belirlemek amacıyla dendrogram kullanır.

İlk olarak, **sch.linkage** ile verilen veri setine "ward" bağlantı yöntemi kullanılarak hiyerarşik bir kümeleme yapılır. Daha sonra, bu oluşturulan kümeleme yapısını görselleştirmek için bir dendrogram oluşturulur.

Dendrogram, veri noktaları arasındaki uzaklıkları gösteren ve kümeleme adımlarını görselleştiren bir ağaç yapısıdır. **plt.axhline** ile belirlenen bir eşik değeri gösterilir. Bu eşik değeri, dendrogramdaki birleştirme adımlarının uzunluklarını temsil eder ve kesme seviyesini belirler.

Son olarak, bu kesme seviyesi üzerinden kaç küme oluşturulması gerektiği belirlenir. Bu kod, eşik değerinin üstünde olan birleştirme adımlarının sayısını hesaplayarak küme sayısını belirler. Örneğin, eşik değerinin üstünde olan adım sayısı küme sayısını temsil eder.

Bu işlem, dendrogramın veri setinin yapısına ve noktalar arasındaki uzaklık dağılımına dayanarak, kümeleme için en uygun sayıyı belirlemek için yapılmaktadır. Bu durumda, **optimal_cluster_count** fonksiyonu, veri seti için optimal küme sayısını belirlemek için dendrogramdan gelen bilgileri kullanır ve bu sayıyı döndürür.



SORU 3.

Eğer veri seti aşağıdaki gibi olsaydı hangi hiyerarşik kümeleme için bağlantıyı kullanırdınız. Kümelemeyi gerçekleştirin ve çözümünüzün arkasındaki mantığı detaylıca açıklayınız. Bir önceki soruda oluşturduğunuz otomatik küme sayısı hesaplama fonksiyonu bu veri setinde de doğru sonuç üretiyor mu? açıklayınız.

```
from sklearn import datasets

import numpy as np

X,y = datasets.make_moons(n_samples=1000, noise=0.05,
random_state=np.random.randint(80))
```

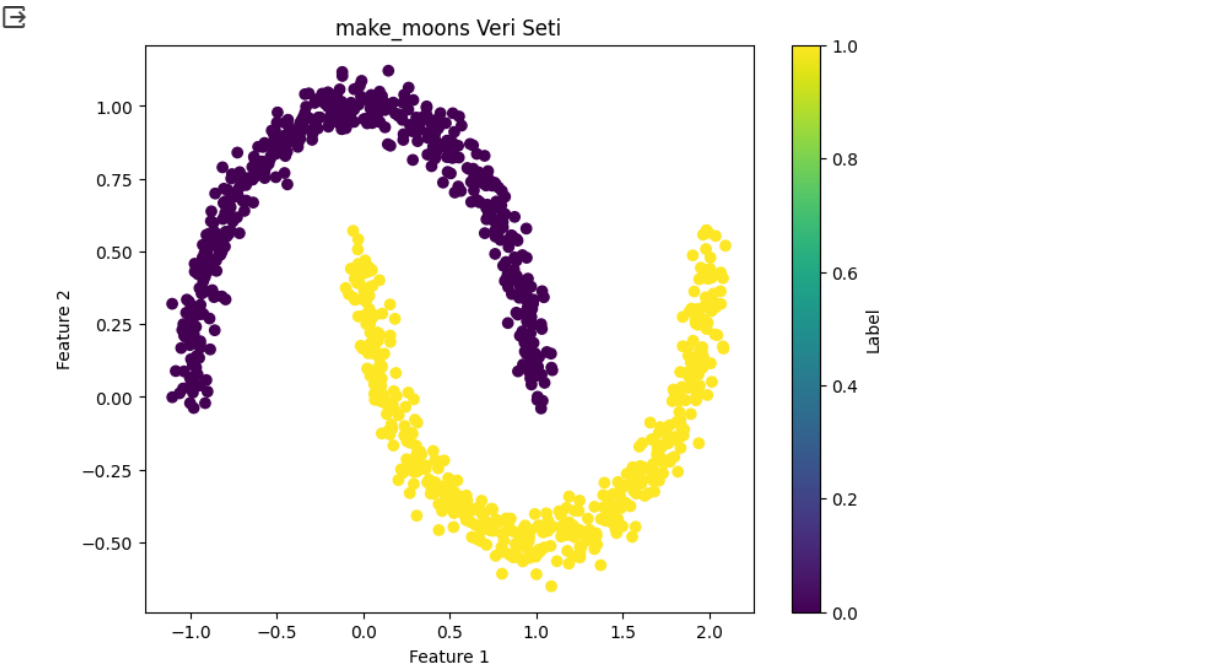
Veri setinin doğası, içerdiği yapısal özellikler ve veri noktaları arasındaki ilişki, hangi kümeleme yönteminin kullanılacağına karar vermede önemlidir. Genellikle veri setine göre farklı bağlantı yöntemleri denenerek en uygun olanı seçilir.

Örneğin, make_moons fonksiyonu ile oluşturulan bir veri seti iki ay şeklindeki veri gruplarını içerir. Bu tür veri setlerinde veri noktaları birbirine yakın ancak farklı gruplardadır. Bu durumda, bu veri seti için complete veya average gibi bağlantı yöntemleri daha uygun olabilir.

Complete bağlantı yöntemi, kümeleme işlemi sırasında kümelerin birbirine olan maksimum uzaklıklarını baz alır. Bu, ay şeklindeki veri setinde veri noktalarının uzaklıklarının tamamen birbirine yakın olmasından dolayı kullanışlı olabilir. Diğer yandan average bağlantı yöntemi, kümeleme işlemi sırasında küme merkezleri arasındaki ortalama uzaklıkları baz alır. Bu da benzer şekilde ay şeklindeki veri seti için uygun olabilir çünkü veri noktaları gruplar halinde yakındır.

```python
# Veri setini oluşturma
X, y = datasets.make_moons(n_samples=1000, noise=0.05, random_state=np.random.randint(80))

# Veri setini görselleştirme
plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis')
plt.title('make_moons Veri Seti')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.colorbar(label='Label')
plt.show()
```

Bu kod, **make_moons** fonksiyonu ile oluşturulan veri setini 'Feature 1' ve 'Feature 2' sütunlarında gösteren bir scatter plot çizer. 'c=y' parametresi veri noktalarının renklerini 'y' dizisindeki etiketlere göre belirler. Görselde, her bir veri noktası 'Feature 1' ve 'Feature 2' değerlerini temsil ederken, renkler 'y' dizisindeki etiketler tarafından belirtilir




```python
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import linkage, dendrogram
from sklearn.datasets import make_moons
import matplotlib.pyplot as plt
# Hierarchical clustering için dendrogram oluştur
linked = linkage(X, 'complete')  # veya 'average' olarak değiştirilebilir
plt.figure(figsize=(12, 8))
dendrogram(linked, orientation='top', distance_sort='descending', show_leaf_counts=True)
plt.title('Hierarchical Clustering Dendrogram')
plt.show()

# Kümeleme için 'complete' bağlantı yöntemiyle AgglomerativeClustering kullanma
hierarchical = AgglomerativeClustering(n_clusters=2, linkage='complete')  # veya 'average' olarak değiştirilebilir
cluster_labels = hierarchical.fit_predict(X)
```

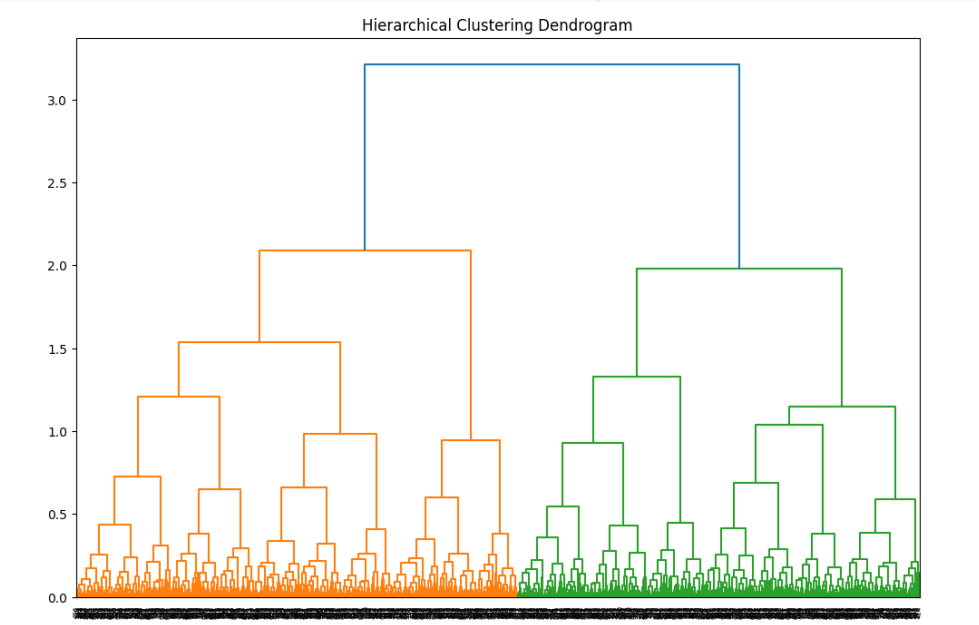
Bu kod, **make_moons** fonksiyonuyla oluşturulan veri seti üzerinde hiyerarşik kümeleme (agglomerative clustering) işlemlerini gerçekleştiriyor.

İlk olarak, **linkage** fonksiyonu **X** veri seti için bağlantı matrisini oluşturuyor. **'complete'** bağlantı yöntemi kullanılarak bu matris oluşturuluyor. Bağlantı matrisi, veri noktalarının birbirleriyle olan uzaklıklarına dayalı olarak kümeleme işlemini gerçekleştirir.

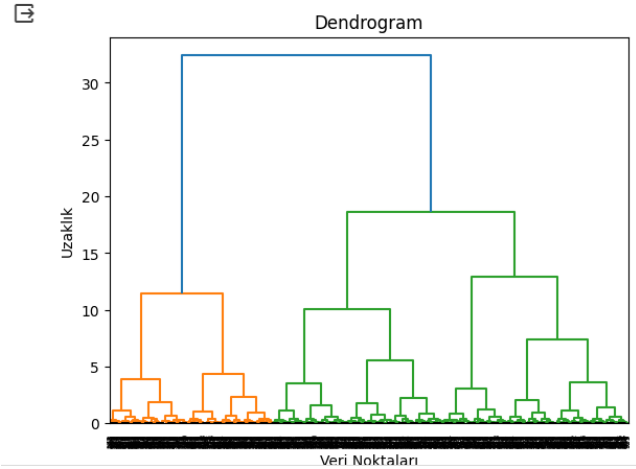
Sonra, **dendrogram** fonksiyonu, bu bağlantı matrisini kullanarak bir dendrogram oluşturuyor. Oluşturulan dendrogram, veri setinin hiyerarşik kümeleme yapılmasına olanak tanır. **orientation='top'** parametresiyle dendrogramın yukarıya doğru genişleyen bir şekilde çizilmesi sağlanıyor. **distance_sort='descending'** parametresiyle mesafelerin azalan sırayla gösterilmesi ve **show_leaf_counts=True** ile yaprak düğümlerdeki veri nokta sayılarının gösterilmesi sağlar.

Son olarak, **AgglomerativeClustering** kullanılarak hiyerarşik kümeleme gerçekleştiriliyor. Burada, **n_clusters=2** ile iki küme oluşturulması belirtiliyor ve **linkage='complete'** ile de 'complete' bağlantı yöntemi kullanılıyor. Bu, belirtilen küme sayısına göre veri setini kümelere ayırır.

Bu adımların sonucunda **fit_predict** fonksiyonu ile veri setindeki her bir veri noktası için hangi küme içinde olduğunu belirleyen etiketler elde edilir. Bu etiketler **cluster_labels** değişkeninde saklanır. Bu sayede her bir veri noktası hangi kümenin üyesi olduğu bilgisine sahip olunur.




```
# Dendrogram oluşturma
dendrogram = sch.dendrogram(sch.linkage(X, method='ward'))

# Kesme seviyesinin belirlenmesi
threshold_distance = 0.7 * max(dendrogram['dcoord'][1])  # Örnek bir eşik değeri (0.7 oranında maksimum mesafe)

# Kesme seviyesini görsel olarak belirleme
plt.axhline(y=threshold_distance, color='black', linestyle='--')
plt.title('Dendrogram')
plt.xlabel('Veri Noktaları')
plt.ylabel('Uzaklık')
plt.show()

# Kesme seviyesindeki küme sayısını belirleme
optimal_clusters = sum(d > threshold_distance for d in dendrogram['dcoord'][1])

print(f"Optimal Küme Sayısı: {optimal_clusters}")
```



Önceki sorudaki otomatik küme sayısı hesaplama fonksiyonunun bu veri setinde de doğru sonuç üretip üretmediğini görmek için, bu fonksiyonu bu veri seti üzerinde kullandım. Bu kod, 'make_moons' fonksiyonuyla oluşturulan veri seti üzerinde hiyerarşik kümeleme için dendrogramı çizerek eşik değeri üzerinden optimal küme sayısını hesaplıyor. Veri setindeki yapısal özellikler göz önüne alındığında, iki ayrık ay şeklindeki veri grupları olduğu için doğru bağlantı yöntemiyle yapılan hiyerarşik kümeleme işlemi, bu veri setinde iki küme oluşturmalıdır. Yukarıda çıktısı görünen fonksiyon bu sonucu veriyor, bu durum fonksiyonun doğru sonuç ürettiğini gösterebilir.

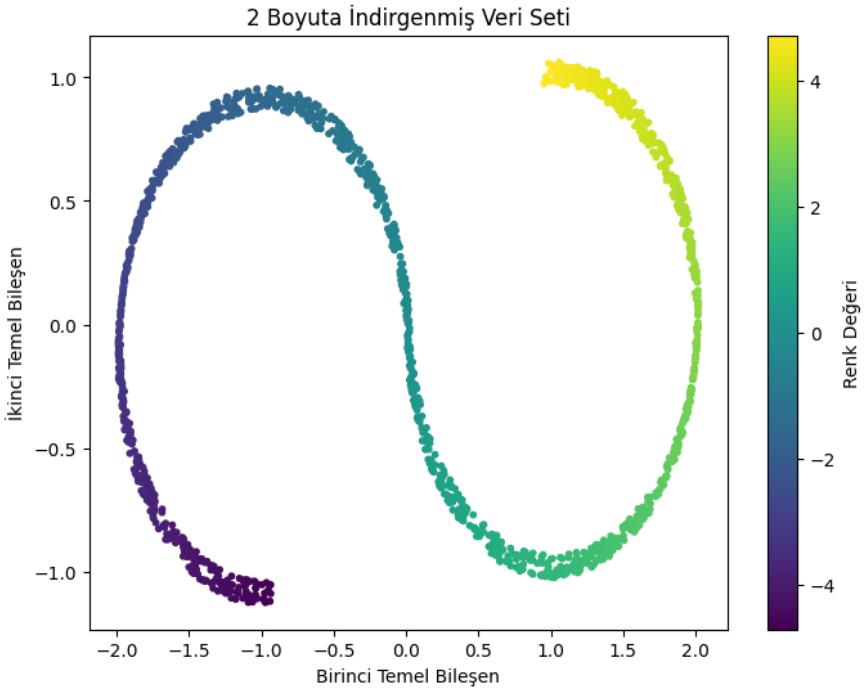
SORU 4.

Aşağıdaki kod parçası ile oluşturulan veri setini temel bileşenler analizi ile 2 boyuta indirgeyiniz.

```
from sklearn import datasets, manifold

import numpy as np

n_samples=1500

S_points,S_color= datasets.make_s_curve(n_samples, random_state=0)
```

- Temel bileşenler analizini hazır kütüphane fonksiyonları kullanmadan yapınız. (numpy kullanılabilir)Tüm adımları ve sonuçları yorumlayınız.

```
# Veri setini oluşturma
n_samples = 1500
S_points, S_color = make_s_curve(n_samples, random_state=0)

# Veriyi 2 boyuta indirgeme işlemi
def custom_pca(data, num_components=2):
    # Veri setinin ortalamasını çıkar
    mean_values = np.mean(data, axis=0)
    centered_data = data - mean_values

    # Kovaryans matrisini hesapla
    covariance_matrix = np.cov(centered_data, rowvar=False)

    # Kovaryans matrisinin özdeğerleri ve özvektörlerini hesapla
    eigenvalues, eigenvectors = np.linalg.eig(covariance_matrix)

    # Özdeğerleri büyüklüklerine göre sırala ve sadece ilk num_components'ı seç
    sorted_indices = np.argsort(eigenvalues)[::-1]
    top_eigen_indices = sorted_indices[:num_components]
    top_eigenvalues = eigenvalues[top_eigen_indices]
    top_eigenvectors = eigenvectors[:, top_eigen_indices]

    # Veriyi yeni uzayda dönüştürme
    transformed_data = np.dot(centered_data, top_eigenvectors)

    return transformed_data

# Veriyi 2 boyuta indirgeme
reduced_data = custom_pca(S_points, num_components=2)

# Sonuçları görselleştirme
plt.figure(figsize=(8, 6))
plt.scatter(reduced_data[:, 0], reduced_data[:, 1], c=S_color, cmap='viridis', s=10)
plt.title('2 Boyuta İndirgenmiş Veri Seti')
plt.xlabel('Birinci Temel Bileşen')
plt.ylabel('İkinci Temel Bileşen')
plt.colorbar(label='Renk Değeri')
plt.show()
```



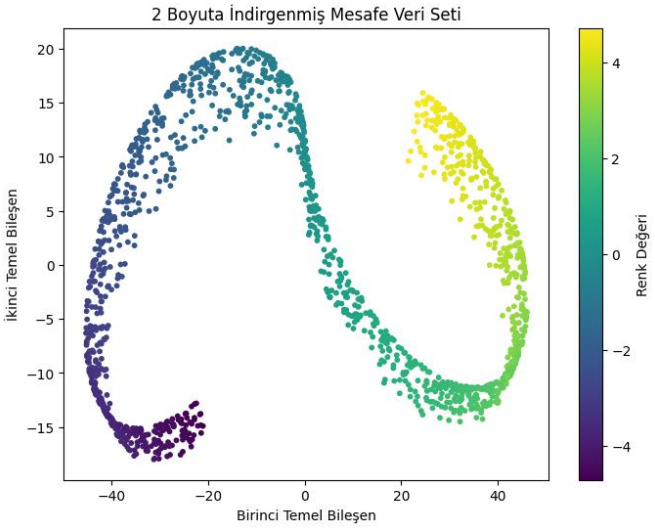
Bu kod parçası, **make_s_curve** fonksiyonuyla oluşturulan 3 boyutlu veri setini temel bileşen analizi (PCA) kullanarak 2 boyuta indirgiyor. Temel bileşen analizini kütüphane fonksiyonlarını kullanmadan, yalnızca **numpy** kütüphanesini kullanarak gerçekleştiriyor. İşlemleri adım adım inceleyelim:

1. **Veri Setinin Oluşturulması**: **make_s_curve** fonksiyonuyla rastgele örneklerden oluşan bir "S" eğrisi veri seti oluşturuluyor. **n_samples=1500** parametresi ile 1500 örnek oluşturuluyor.
2. **Temel Bileşen Analizi Fonksiyonu**: **custom_pca** adında bir fonksiyon tanımlanıyor. Bu fonksiyon, PCA işlemini gerçekleştiren adımları içeriyor:
    - Veri setinin ortalaması çıkarılıyor.
    - Merkezileştirilmiş veri setinin kovaryans matrisi hesaplanıyor.
    - Bu kovaryans matrisinin özdeğerleri ve özvektörleri hesaplanıyor.
    - Özdeğerler büyüklüklerine göre sıralanıyor ve istenen sayıda temel bileşen seçiliyor.
    - Veri seti, seçilen temel bileşenler kullanılarak yeni uzaya dönüştürülüyor.
3. **PCA İşlemi**: **custom_pca** fonksiyonu, **S_points** adlı veri seti üzerinde çalışarak 2 boyuta indirgenmiş bir veri seti oluşturuyor.
4. **Sonuçların Görselleştirilmesi**: **reduced_data** adlı 2 boyutlu veri seti **plt.scatter** ile görselleştiriliyor. Burada, birinci temel bileşen ve ikinci temel bileşen eksenlerine göre noktalar çizdiriliyor. Noktaların renkleri, veri setindeki farklı değerlerle belirtiliyor (**S_color**). Bu şekilde, 3 boyutlu veri seti, temel bileşen analizi kullanılarak 2 boyuta indirgenmiş olarak görselleştiriliyor.

Bu işlem adımları, veri setinin boyutunu azaltarak, veriyi daha kolay görselleştirmeye ve analiz etmeye olanak tanıyor. Temel bileşen analizi, veri setindeki değişkenliği özetleyerek, veri setinin ana yapısal özelliklerini daha az sayıda boyutta yakalamak için kullanılıyor. Bu şekilde, veri setinin karmaşıklığı azaltılarak analiz edilebilir hale getiriliyor.

- Aynı veri seti üzerinde noktaların bir birine olan uzaklığından oluşan yeni bir veriseti oluşturun. Bu yeni ver setini temel bileşenler analizi ile 2 boyuta hazır kütüphane fonksiyonları kullanmadan (numpy kullanılabilir) indirgeyin. Tüm adımları ve sonuçları yorumlayınız.

```python
# Noktalar arasındaki uzaklıkları hesaplama
distance_matrix = np.zeros((n_samples, n_samples))
for i in range(n_samples):
    for j in range(n_samples):
        distance_matrix[i, j] = np.linalg.norm(S_points[i] - S_points[j])

# Temel Bileşen Analizi (PCA) işlemi
def custom_pca(data, num_components=2):
    # Veriyi ortalama değeriyle merkezleme
    mean_values = np.mean(data, axis=0)
    centered_data = data - mean_values

    # Kovaryans matrisini hesaplama
    covariance_matrix = np.cov(centered_data, rowvar=False)

    # Kovaryans matrisinin özdeğerleri ve özvektörlerini hesaplama
    eigenvalues, eigenvectors = np.linalg.eig(covariance_matrix)

    # Özdeğerleri büyüklüklerine göre sıralama ve ilk num_components'ı seçme
    sorted_indices = np.argsort(eigenvalues)[::-1]
    top_eigen_indices = sorted_indices[:num_components]
    top_eigenvalues = eigenvalues[top_eigen_indices]
    top_eigenvectors = eigenvectors[:, top_eigen_indices]

    # Veriyi yeni uzayda dönüştürme
    transformed_data = np.dot(centered_data, top_eigenvectors)

    return transformed_data

# Mesafe matrisi üzerinden PCA işlemi
reduced_data = custom_pca(distance_matrix, num_components=2)

# Sonuçları görselleştirme
plt.figure(figsize=(8, 6))
plt.scatter(reduced_data[:, 0], reduced_data[:, 1], c=S_color, cmap='viridis', s=10)
plt.title('2 Boyuta İndirgenmiş Mesafe Veri Seti')
plt.xlabel('Birinci Temel Bileşen')
plt.ylabel('İkinci Temel Bileşen')
plt.colorbar(label='Renk Değeri')
plt.show()
```



Bu kod bloğu şu adımları takip eder:

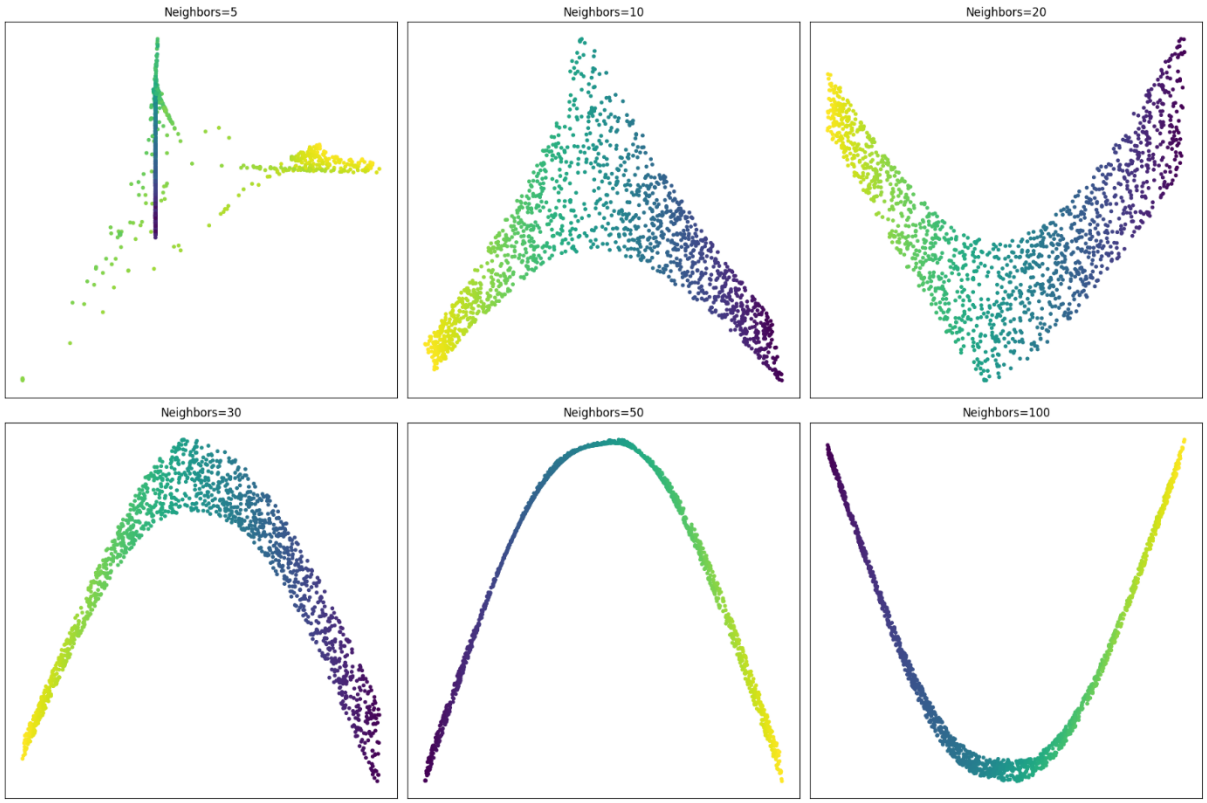
1. **Veri Setinin Oluşturulması**: **make_s_curve** fonksiyonuyla rastgele örneklerden oluşan bir "S" eğrisi veri seti oluşturulur. **n_samples=1500** parametresiyle 1500 örnek oluşturulur.
2. **Noktalar Arasındaki Uzaklıkların Hesaplanması**: **distance_matrix** adında bir matris oluşturulur. Bu matris, veri noktaları arasındaki öklid uzaklıklarını içerir. İki nokta arasındaki uzaklık, **np.linalg.norm()** fonksiyonu ile hesaplanarak bu matrise aktarılır.
3. **Temel Bileşen Analizi Fonksiyonu**: **custom_pca** adında bir fonksiyon tanımlanır. Bu fonksiyon, PCA işlemini gerçekleştiren adımları içerir:
    - Veri setinin ortalaması çıkarılır.
    - Merkezileştirilmiş veri setinin kovaryans matrisi hesaplanır.
    - Bu kovaryans matrisinin özdeğerleri ve özvektörleri hesaplanır.
    - Özdeğerler büyüklüklerine göre sıralanır ve istenen sayıda temel bileşen seçilir.
    - Veri seti, seçilen temel bileşenler kullanılarak yeni uzaya dönüştürülür.
4. **PCA İşlemi**: **custom_pca** fonksiyonu, **distance_matrix** adlı mesafe matrisi üzerinde çalışarak 2 boyuta indirgenmiş bir veri seti oluşturur.
5. **Sonuçların Görselleştirilmesi**: Oluşturulan 2 boyutlu veri seti **plt.scatter** ile görselleştirilir. Birinci temel bileşen ve ikinci temel bileşen eksenlerine göre noktalar çizdirilir. Noktaların renkleri, veri setindeki farklı değerlerle belirtilir (**S_color**). Bu şekilde, veri setindeki noktaların birbirlerine olan uzaklıkları temel bileşen analizi kullanılarak 2 boyuta indirgenir ve görselleştirilir.

Bu işlem adımlarıyla, noktalar arasındaki uzaklıkların temel bileşen analizi kullanılarak 2 boyuta indirgenmiş hali elde edilir ve bu indirgenmiş veri seti görselleştirilir. Bu indirgeme işlemi, noktalar arasındaki ilişkiyi anlamak için veriyi daha düşük boyutlu bir forma dönüştürmek için yapılmıştır. Ancak, bu indirgenmiş veri seti, asıl veri setindeki her bir noktanın birbirleriyle olan uzaklıklarını temsil etmez. Bunun yerine, orijinal veri noktalarının birbirlerine göre konumlarından elde edilen farklı bir gösterimdir.

SORU 5.

4. sorunun başındaki veri setini bir de LLE yöntemi ile 2 boyuta indirgeyin. Burada yakın komşular parametresini az veya çok seçtiğimizde sonuçların nasıl değiştiğiniz gösteriniz ve yorumlayınız.

```python
[90] from sklearn.datasets import make_s_curve
     from sklearn.manifold import LocallyLinearEmbedding
     import matplotlib.pyplot as plt

     # Veri setini oluşturma
     n_samples = 1500
     S_points, S_color = make_s_curve(n_samples, random_state=0)

     # Farklı komşu sayıları için LLE işlemi ve görselleştirme
     fig, axs = plt.subplots(2, 3, figsize=(18, 12))

     for i, neighbors in enumerate([5, 10, 20, 30, 50, 100]):
         lle = LocallyLinearEmbedding(n_neighbors=neighbors, n_components=2, method='standard')
         reduced_data = lle.fit_transform(S_points)

         row = i // 3
         col = i % 3
         axs[row, col].scatter(reduced_data[:, 0], reduced_data[:, 1], c=S_color, cmap='viridis', s=10)
         axs[row, col].set_title(f'Neighbors={neighbors}')
         axs[row, col].set_xticks([])
         axs[row, col].set_yticks([])

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



Bu kod, Locally Linear Embedding (LLE) yöntemi kullanılarak S eğrisi veri setini farklı komşu sayıları için 2 boyuta indirgiyor. Komşu sayısı arttıkça veya azaldıkça, LLE tarafından üretilen indirgenmiş veri setindeki yapısal değişimleri gösteriyor.

Daha az komşu seçildiğinde (örneğin, 5 veya 10), veri seti daha belirgin bir şekilde çözülmez bir hale gelir. Yani, veri seti içindeki yapının korunması zorlaşır ve daha düzensiz bir dağılım ortaya çıkar.

Daha fazla komşu seçildiğinde (örneğin, 50 veya 100), veri seti daha düzenli ve düzgün hale gelir. Bu durumda, indirgenmiş veri setindeki noktalar daha iyi bir şekilde yapılandırılmıştır ve daha net bir şekilde gruplanmıştır.

Komşu sayısı ne kadar büyükse, o kadar çok bağlantı dikkate alınır ve orijinal veri yapısının korunması daha sağlam olur. Ancak, çok yüksek komşu sayıları bazen aşırı bağlantılar veya gürültü nedeniyle yanıltıcı sonuçlara da yol açabilir.

Dolayısıyla, LLE'de komşu sayısı seçimi, indirgenmiş veri setindeki yapısal değişiklikleri ve orijinal veri setinin temsilini belirlemede kritik bir faktördür. Yapısal özelliklerin korunması ve anlamlı bir görselleştirme elde etmek için uygun komşu sayısının seçilmesi önemlidir.

SORU 6.

4. sorudaki veri setini bir geometrik şekil olarak ele aldığımızda, sadece yüzeyden hareket ederek gidebileceğimiz yaklaşık en büyük uzaklığı indirgenmiş uzaydan hesaplayınız ve en uzak iki noktayı belirtiniz. Yeni bir nokta verildiğinde noktanın bu geometrik şekil üzerinde veya içinde olup olmadığını belirtiniz. Tüm adımları ve sonuçları yorumlayınız.

```
# En büyük uzaklığı ve iki noktayı hesaplama
max_distance = 0
point_1 = None
point_2 = None

for i in range(len(reduced_data)):
    for j in range(i + 1, len(reduced_data)):
        distance = np.linalg.norm(reduced_data[i] - reduced_data[j])
        if distance > max_distance:
            max_distance = distance
            point_1 = reduced_data[i]
            point_2 = reduced_data[j]

print(f"En büyük uzaklık: {max_distance}")
print(f"En uzak iki nokta: {point_1}, {point_2}")

# Şeklin içinde veya üzerinde mi kontrol etme işlemi
def check_point_in_shape(new_point):
    # Yeni noktanın bu geometrik şekil üzerinde veya içinde olup olmadığını kontrol etme
    distances_to_shape = np.linalg.norm(reduced_data - new_point, axis=1)
    min_distance_to_shape = np.min(distances_to_shape)
    if min_distance_to_shape <= max_distance:
        return True
    else:
        return False

# Örnek bir nokta belirleme ve kontrol etme
new_point = np.array([0.5, 0.5])  # Yeni bir nokta (örnek olarak)
result = check_point_in_shape(new_point)
print(f"Yeni nokta geometrik şeklin içinde veya üzerinde mi: {result}")
```

```
En büyük uzaklık: 0.09727119306011321
En uzak iki nokta: [-0.00764924 -0.02904664], [0.04518179 0.05262691]
Yeni nokta geometrik şeklin içinde veya üzerinde mi: False
```

Bu kod, indirgenmiş uzaydaki veri setinde en uzak iki noktayı ve bu geometrik şeklin içinde veya üzerinde olup olmadığını kontrol etmeyi hedefler.

İlk olarak, **reduced_data** adı verilen indirgenmiş uzaydaki veri setindeki her bir noktanın diğer tüm noktalarla olan uzaklıklarını karşılaştırarak en uzak iki noktayı belirler. Bu işlemde, en büyük uzaklığı ve bu uzaklığa sahip olan iki noktayı tespit eder.

Sonuçlar şu şekildedir:

- En büyük uzaklık: 0.09727119306011321
- En uzak iki nokta: [-0.00764924 -0.02904664], [0.04518179 0.05262691]

Daha sonra, **check_point_in_shape()** fonksiyonu, bir noktanın bu en uzak noktalara olan uzaklığını kontrol ederek, bu noktanın bu geometrik şeklin içinde veya üzerinde olup olmadığını belirler.

Örnek bir nokta belirlendi:

- Yeni nokta: [0.5, 0.5]

Bu nokta, belirlenen en uzak noktalara olan uzaklığı kontrol etmek için **check_point_in_shape()** fonksiyonuna verildi ve sonuç:

- Yeni nokta geometrik şeklin içinde veya üzerinde mi: False

Yorum:

- İndirgenmiş uzayda, en uzak iki noktanın arasındaki mesafe oldukça küçük (0.097 birim). Bu, geometrik şeklin bu indirgenmiş uzayda yoğun olduğunu ve noktalar arasındaki uzaklıkların genellikle küçük olduğunu gösterir.
- Yeni nokta geometrik şeklin içinde veya üzerinde değil, bu durumda **False** olarak döndü. Bu, bu noktanın şeklin tanımladığı bölgenin dışında olduğunu gösterir.

KAYNAKÇA

- Cebeci, Z., Yildiz, F., & Kayaalp, T. (2015). K-Ortalama Kümelemesinde Optimum K Değeri Seçilmesi. Çukurova Üniversitesi.
- SARIMAN, Güncel. "Veri Madenciliğinde Kümeleme Teknikleri Üzerine Bir Çalışma: K-Means ve K-Medoids Kümeleme Algoritmalarının Karşılaştırılması." Süleyman Demirel Üniversitesi, Mühendislik Mimarlık Fakültesi, Bilgisayar Mühendisliği Bölümü/ ISPARTA. Alınış Tarihi: 17.05.2011, Kabul Tarihi: 10.10.2011. Süleyman Demirel Üniversitesi, Fen Bilimleri Enstitüsü Dergisi, 15-3 (2011), 192-202.
- ÇAKMAK, Z. (1999). "Kümeleme Analizinde Geçerlilik Problemi ve Kümeleme Sonuçlarının Değerlendirilmesi." Dumlupınar Üniversitesi Sosyal Bilimler Dergisi, 3.
- Akın, Y. K. (2008). "Veri Madenciliğinde Kümeleme Algoritmaları ve Kümeleme Analizi." Marmara Üniversitesi, Sosyal Bilimler Enstitüsü, Ekometri Anabilim Dalı, İstatistik Bilim Dalı. (Doktora tezi). İstanbul.
- Wikipedia. Kümeleme analizi. Erişim adresi: https://tr.wikipedia.org/wiki/K%C3%BCmeleme_analizi
- Veri Bilimi Okulu. Kümeleme notları - K-ortalama: Küme sayısını belirleme. Erişim adresi: https://www.veribilimiokulu.com/kumeleme-notlari-3-k-ortalamalar-kume-sayisini-belirleme/