In [ ]:
```
# Import the seaborn library
# seaborn kütüphanesini içe aktar

import seaborn as sns
```

In [ ]:
```
# Load the Iris dataset from seaborn library
# seaborn kütüphanesinden Iris veri kümesini yükle

Iris = sns.load_dataset('iris')
```

In [ ]:
```
# Display the first few rows of the Iris dataset
# Iris veri kümesinin ilk birkaç satırını görüntüle

Iris.head()
```

Out[ ]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [ ]:
```
# Drop the 'species' column from the Iris dataset to create the feature ma
# Iris veri kümesinden 'species' sütununu çıkararak özellik matrisi X_iris

X_iris = Iris.drop('species', axis=1)

# Display the shape of the feature matrix X_iris
# Özellik matrisi X_iris'in boyutunu görüntüle

X_iris.shape
```

Out[ ]:
```
(150, 4)
```

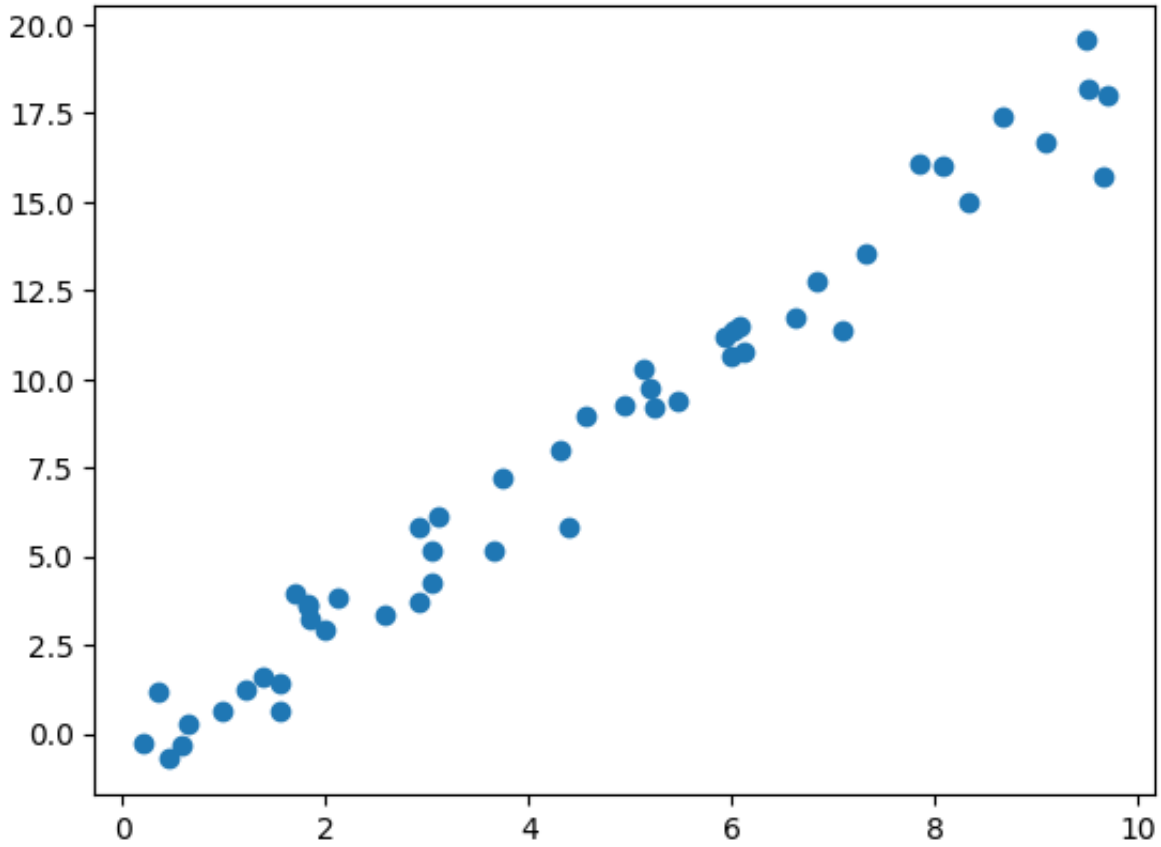
In [ ]:
```
# Extract the 'species' column from the Iris dataset to create the target
# Iris veri kümesinden 'species' sütununu çıkararak hedef vektörü y_iris'i

y_iris = Iris['species']

# Display the shape of the target vector y_iris
# Hedef vektörü y_iris'in boyutunu görüntüle

y_iris.shape
```

Out[ ]: (150,)

In [ ]:
```
# Import the necessary libraries for plotting and numerical operations
# Grafik çizimi ve sayısal işlemler için gerekli kütüphaneleri içe aktar

import matplotlib.pyplot as plt
import numpy as np
```

In [ ]:
```
# Create a random number generator with a fixed seed for reproducibility
# Tekrarlanabilirlik için sabit bir tohum değeri ile rastgele sayı üreteci

rng = np.random.RandomState(42)
```

In [ ]:
```
# Generate random X values between 0 and 10
# 0 ile 10 arasında rastgele X değerleri oluştur
x = 10 * rng.random(50)

# Generate Y values with a linear relationship to X and add random noise
# X'e lineer bir ilişkisi olan Y değerlerini oluştur ve rastgele gürültü e

# Linear relationship: Y = 2*X - 1
# Lineer ilişki: Y = 2*X - 1

y = 2 * x - 1 + rng.randn(50)

# Visualize the data using a scatter plot
# Veriyi scatter plot ile görselleştir
plt.scatter(x, y)

# Show the plot
# Grafiği göster
plt.show()
```



In [ ]:
```
# pip install -U scikit-learn
```

In [ ]:
```
# Import the Linear Regression model from scikit-learn library
# scikit-learn kütüphanesinden Lineer Regresyon modelini içe aktar

from sklearn.linear_model import LinearRegression
```

In [ ]:
```
# Create a Linear Regression model with intercept term included
# Araya terim dahil edilmiş Lineer Regresyon modeli oluştur

# LinearRegression sınıfından bir model oluşturuluyor.
# fit_intercept=True parametresi, modelin araya terim eklemesini sağlar.
model = LinearRegression(fit_intercept=True)

# Oluşturulan modeli ekrana yazdır
model
```

Out[ ]:
```
▾ LinearRegression
LinearRegression()
```

In [ ]:
```
# Reshape the feature matrix X by adding a new axis
# Özellik matrisi X'i yeni bir eksen ekleyerek yeniden şekillendir

# x'in boyutunu (50,) olan vektörden (50, 1) boyutlu bir matrise dönüştür
X = x[:, np.newaxis]

# Yeniden şekillendirilmiş özellik matrisi X'in boyutunu görüntüle
X.shape
```

Out[ ]:
```
(50, 1)
```

In [ ]:
```
# Fit the Linear Regression model using the feature matrix X and target ve
# Özellik matrisi X ve hedef vektör y kullanılarak Lineer Regresyon modeli

model.fit(X, y)
```

Out[ ]:
```
▾ LinearRegression
LinearRegression()
```

In [ ]:
```
# Retrieve the coefficients (slopes) of the Linear Regression model
# Lineer Regresyon modelinin katsayılarını (eğimlerini) al

model.coef_
```

Out[ ]:
```
array([1.9776566])
```

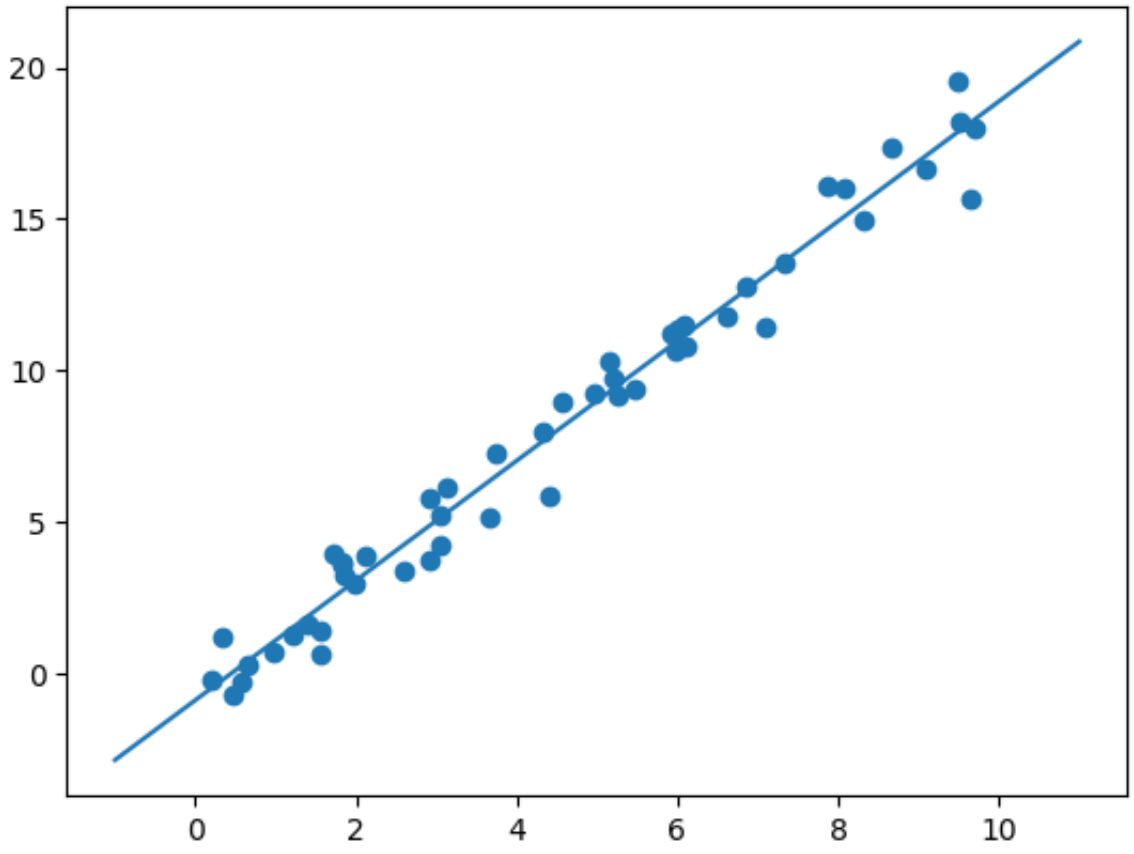
In [ ]:
```
# Retrieve the intercept term of the Linear Regression model
# Lineer Regresyon modelinin araya terimini al

# Oluşturulan Lineer Regresyon modelinin araya terimini elde et
intercept = model.intercept_

# Elde edilen araya terimi ekrana yazdır
print("Intercept Term:", intercept)
```

```
Intercept Term: -0.9033107255311146
```

In [ ]:
```
# Generate a set of x values for prediction
# Tahmin için bir dizi x değeri oluştur

x_fit = np.linspace(-1, 11)

# Reshape the x values to create the feature matrix X_fit
# Özellik matrisi X_fit'i oluşturmak için x değerlerini yeniden şekillendi

X_fit = x_fit[:, np.newaxis]

# Use the trained Linear Regression model to predict y values for the new
# Eğitilmiş Lineer Regresyon modelini kullanarak yeni x değerleri için y d

y_fit = model.predict(X_fit)
```

In [ ]:
```
# Scatter plot of the original data points
# Orijinal veri noktalarının scatter plot'u

plt.scatter(x, y)

# Plot the regression line using the predicted values
# Tahmin edilen değerleri kullanarak regresyon doğrusunu çiz

plt.plot(x_fit, y_fit)
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x16e84c0d0>]
```



In [ ]:
```
# Import the train_test_split function from scikit-learn
# scikit-learn kütüphanesinden train_test_split fonksiyonunu içe aktar

from sklearn.model_selection import train_test_split
```

In [ ]:
```
# Split the Iris dataset into training and testing sets
# Iris veri kümesini eğitim ve test setlerine böl

# X_iris özellik matrisi ile y_iris hedef vektörünü, random_state=1 ile be
X_train, X_test, y_train, y_test = train_test_split(X_iris, y_iris, random_
```

In [ ]:
```
# Import the Gaussian Naive Bayes classifier from scikit-learn
# scikit-learn kütüphanesinden Gaussian Naive Bayes sınıflandırıcısını içe

from sklearn.naive_bayes import GaussianNB
```

In [ ]:
```
# Create a Gaussian Naive Bayes classifier model
# Gaussian Naive Bayes sınıflandırıcı modeli oluştur

model = GaussianNB()
```

In [ ]:
```
# Train the Gaussian Naive Bayes classifier using the training data
# Eğitim verilerini kullanarak Gaussian Naive Bayes sınıflandırıcıyı eğit

model.fit(X_train, y_train)
```

Out[ ]:
▾ GaussianNB

GaussianNB()

In [ ]:
```
# Use the trained Gaussian Naive Bayes classifier to make predictions on t
# Eğitilmiş Gaussian Naive Bayes sınıflandırıcıyı kullanarak test verileri

y_model = model.predict(X_test)
```

In [ ]:
```
# Import the accuracy_score function from scikit-learn metrics module
# scikit-learn kütüphanesinden accuracy_score fonksiyonunu içe aktar

from sklearn.metrics import accuracy_score
```

In [ ]:
```
# Calculate the accuracy score by comparing the true labels (y_test) with
# Gerçek etiketler (y_test) ile tahmin edilen etiketler (y_model) arasında

accuracy_score(y_test, y_model)
```

Out[ ]:
```
0.9736842105263158
```

Unsupervised ML

In [ ]:
```
# Display the first few rows of the Iris dataset
# Iris veri kümesinin ilk birkaç satırını görüntüle

Iris.head()
```

Out[ ]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [ ]:
```
# Import the PCA (Principal Component Analysis) class from scikit-learn
# scikit-learn kütüphanesinden PCA (Temel Bileşen Analizi) sınıfını içe ak

from sklearn.decomposition import PCA
```

In [ ]:
```
# Create a PCA (Principal Component Analysis) model with 2 components
# 2 bileşeni içeren bir PCA (Temel Bileşen Analizi) modeli oluştur

# Create a model from the PCA class.
# The parameter n_components=2 specifies the number of principal component
# PCA sınıfından bir model oluşturuluyor.
# n_components=2 parametresi, elde edilmek istenen temel bileşen sayısını
model = PCA(n_components=2)
```

In [ ]:
```
# Fit the PCA (Principal Component Analysis) model to the Iris dataset
# PCA (Temel Bileşen Analizi) modelini Iris veri kümesine uydur

model.fit(X_iris)
```

Out[ ]:
```
PCA
PCA(n_components=2)
```

In [ ]:
```
# Transform the original Iris dataset into a 2-dimensional representation
# Eğitilmiş PCA modelini kullanarak orijinal Iris veri kümesini 2 boyutlu

X_2D = model.transform(X_iris)
```

In [ ]:
```
# Add the transformed PCA components to the Iris dataset as new columns
# Dönüştürülmüş PCA bileşenlerini Iris veri kümesine yeni sütunlar olarak

Iris['PCA1'] = X_2D[:, 0]
Iris['PCA2'] = X_2D[:, 1]
```

In [ ]:
```
Iris.head()
```

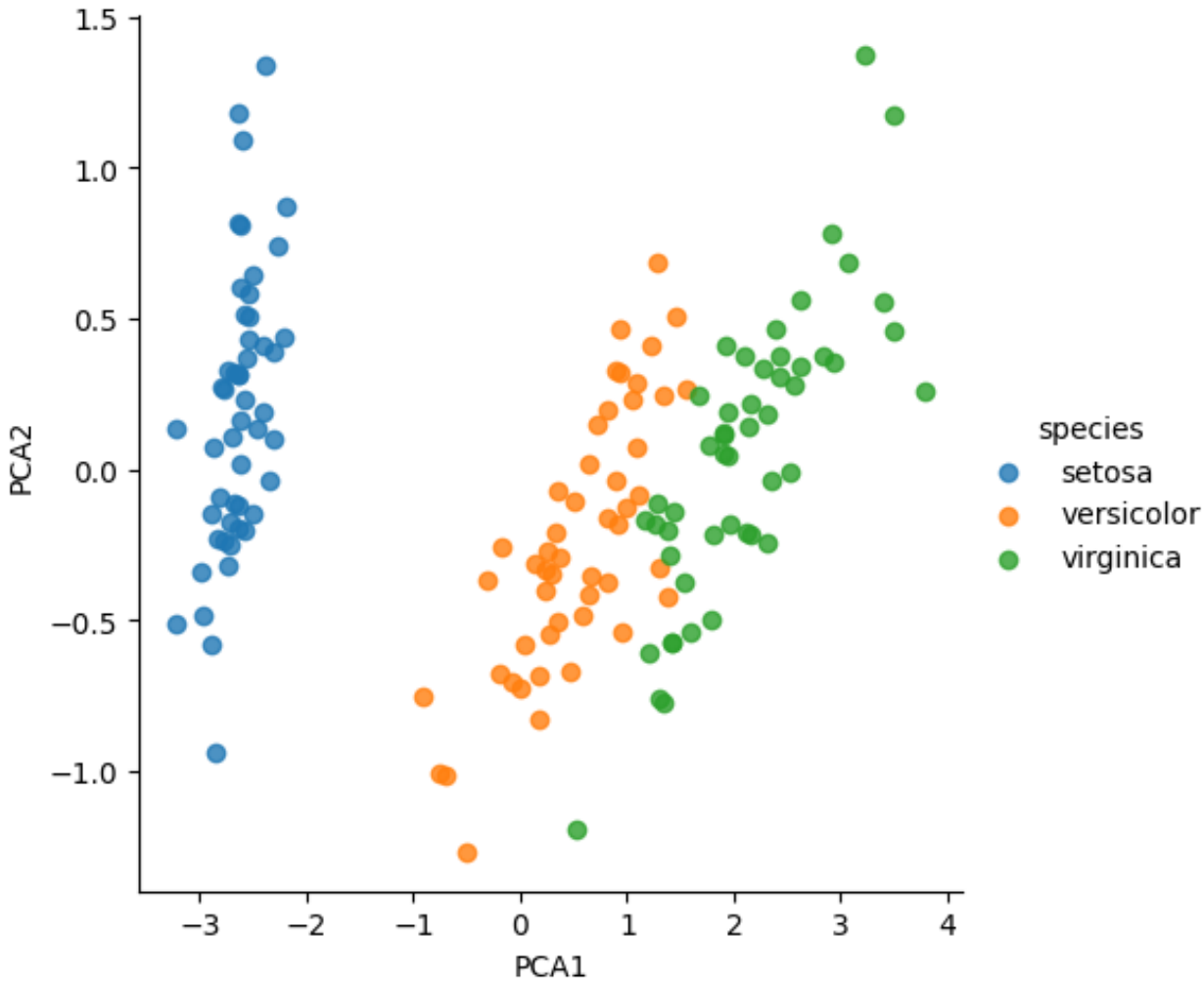
Out[ ]:

	sepal_length	sepal_width	petal_length	petal_width	species	PCA1	PCA2
0	5.1	3.5	1.4	0.2	setosa	-2.684126	0.319397
1	4.9	3.0	1.4	0.2	setosa	-2.714142	-0.177001
2	4.7	3.2	1.3	0.2	setosa	-2.888991	-0.144949
3	4.6	3.1	1.5	0.2	setosa	-2.745343	-0.318299
4	5.0	3.6	1.4	0.2	setosa	-2.728717	0.326755

In [ ]:
```
import seaborn as sns
```

In [ ]:
```
# Scatter plot oluştur
# 'PCA1' sütununu x ekseni, 'PCA2' sütununu y ekseni olarak kullanarak sca
# Noktaları 'species' sütununa göre renklendir
sns.lmplot(x='PCA1', y='PCA2', hue='species', data=Iris, fit_reg=False)
```

Out[ ]: <seaborn.axisgrid.FacetGrid at 0x17d6a5240>



In [ ]:
```
# Import the GaussianMixture class from scikit-learn mixture module
# scikit-learn mixture modülünden GaussianMixture sınıfını içe aktar

from sklearn.mixture import GaussianMixture
# Bu satırda, scikit-learn kütüphanesinden Gaussian Mixture Model'ini içer
# Gaussian Mixture Model, veri setindeki karmaşıklığı ve gizli yapıları ke
```

In [ ]:
```
# Create a Gaussian Mixture Model with 3 components and 'full' covariance
# 3 bileşenli ve 'full' kovaryans tipine sahip bir Gaussian Mixture Modeli

model = GaussianMixture(n_components=3, covariance_type='full')
```

In [ ]:
```
# Fit the Gaussian Mixture Model to the Iris dataset
# Gaussian Mixture Model'i Iris veri kümesine uydur

model.fit(X_iris)
```

Out[ ]:
```
GaussianMixture(n_components=3)
```

In [ ]:
```
# Use the trained Gaussian Mixture Model to predict cluster labels for the
# Eğitilmiş Gaussian Mixture Model'i kullanarak Iris veri kümesi için küme

y_gmm = model.predict(X_iris)
```

In [ ]:
```
# Add the predicted cluster labels from the Gaussian Mixture Model to the
# Gaussian Mixture Model tarafından tahmin edilen küme etiketlerini Iris v

Iris['Cluster'] = y_gmm
```

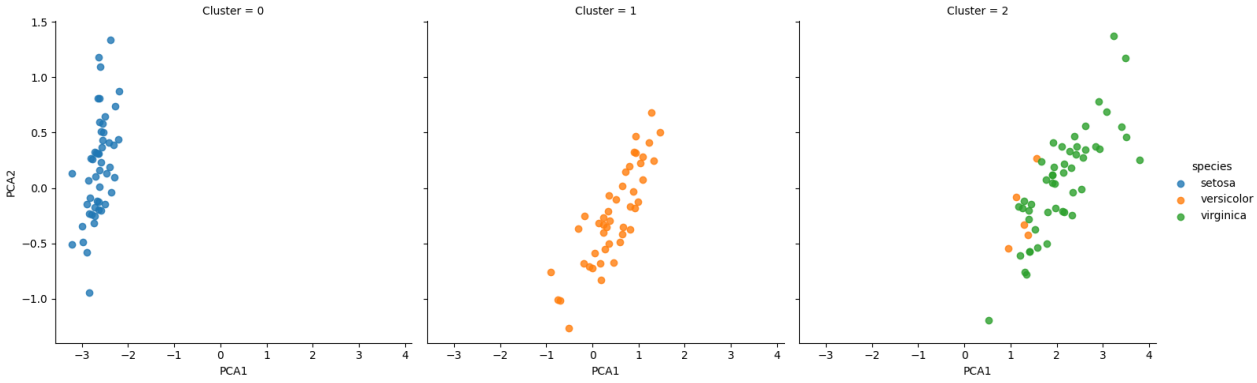
In [ ]:
```
Iris.head()
```

Out[ ]:

	sepal_length	sepal_width	petal_length	petal_width	species	PCA1	PCA2	Clu
0	5.1	3.5	1.4	0.2	setosa	-2.684126	0.319397	
1	4.9	3.0	1.4	0.2	setosa	-2.714142	-0.177001	
2	4.7	3.2	1.3	0.2	setosa	-2.888991	-0.144949	
3	4.6	3.1	1.5	0.2	setosa	-2.745343	-0.318299	
4	5.0	3.6	1.4	0.2	setosa	-2.728717	0.326755	

In [ ]:
```
# Create a scatter plot for the Iris dataset, using 'PCA1' as the x-axis,
# coloring points based on the 'species' column, and creating separate colu
# Iris veri kümesi için scatter plot oluştur, 'PCA1' sütununu x ekseni, 'P
# Noktaları 'species' sütununa göre renklendir, her 'Cluster' için ayrı sü

sns.lmplot(x='PCA1', y='PCA2', hue='species', data=Iris, col='Cluster', fi
```

Out[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x17fd02290>
```



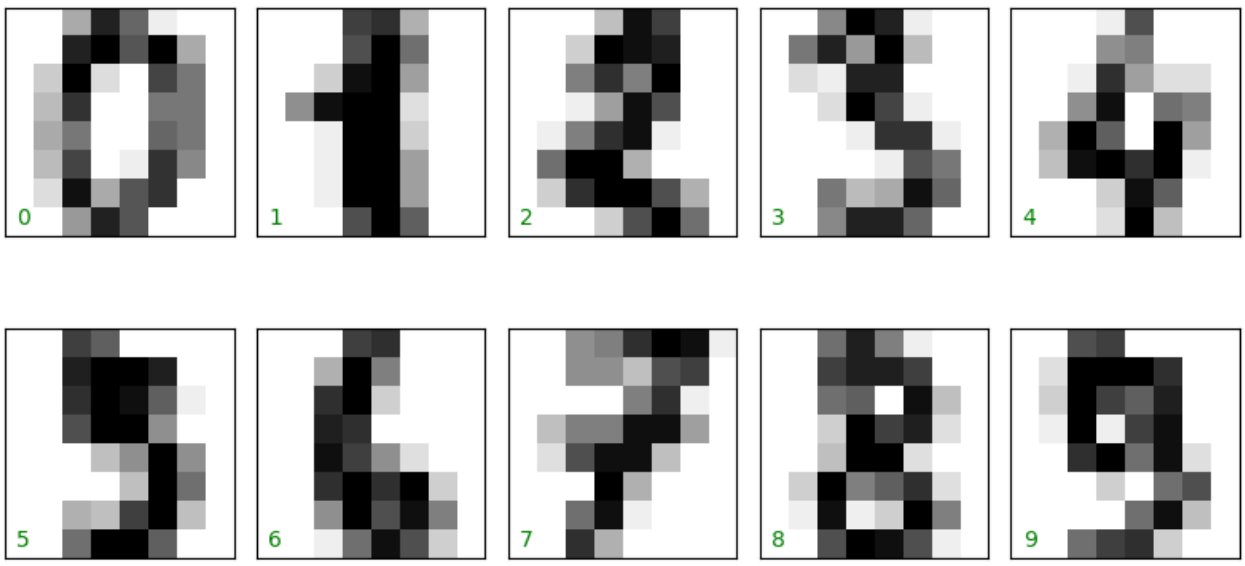
In [ ]:
```
# Import the load_digits function from scikit-learn datasets module
# scikit-learn datasets modülünden load_digits fonksiyonunu içe aktar

from sklearn.datasets import load_digits
```

In [ ]:
```
# Load the digits dataset using the load_digits function
# load_digits fonksiyonunu kullanarak el yazısı rakam veri kümesini yükle

digits = load_digits()
```

In [ ]:
```
# Display the shape of the images in the digits dataset
# digits veri kümesindeki görüntülerin şeklini göster

digits.images.shape
```

Out[ ]:
```
(1797, 8, 8)
```

In [ ]:
```
# 2x5 boyutunda bir figür oluştur
fig, axes = plt.subplots(2, 5, figsize=(10, 5), subplot_kw={'xticks':[], '

# Her bir eksen üzerindeki imajları ve hedef değerleri göster
for i, ax in enumerate(axes.flat):
    # İmajları göster
    ax.imshow(digits.images[i], cmap='binary', interpolation='nearest')

    # Hedef değerleri eksenin altına metin olarak ekle
    # Bu metot, eksenin sol alt köşesinden itibaren belirtilen koordinatla
    ax.text(0.05, 0.05, str(digits.target[i]), transform=ax.transAxes, colo

# Figürü göster
plt.show()
```



In [ ]:
```
# Assign the features (X) and target labels (y) from the digits dataset
# digits veri kümesinden özellikleri (X) ve hedef etiketleri (y) atayın
X = digits.data
y = digits.target

# Display the shape of the features and target labels
# Özelliklerin ve hedef etiketlerin şeklini göster
print(X.shape)
print(y.shape)
```

```
(1797, 64)
(1797,)
```

In [ ]:
```
# Import the Isomap class from scikit-learn manifold module
# scikit-learn manifold modülünden Isomap sınıfını içe aktar

# Bu satırda, scikit-learn kütüphanesinden Isomap sınıfını içe aktarıyorsu
# Isomap, veri setindeki yapının düşük boyutlu bir temsilini öğrenmek için
from sklearn.manifold import Isomap
```

In [ ]:
```
# Create an Isomap model with 2 components
# 2 bileşenli bir Isomap modeli oluştur

iso = Isomap(n_components=2)
```

In [ ]:
```
# Fit the Isomap model to the features (X) from the digits dataset
# Isomap modelini digits veri kümesindeki özelliklere (X) uydur

iso.fit(X)
```

```
/Users/onurgumus/Desktop/Python ile Projeler/Scikit-Learn/.venv/lib/python
3.10/site-packages/sklearn/manifold/_isomap.py:359: UserWarning: The number
of connected components of the neighbors graph is 2 > 1. Completing the gra
ph to fit Isomap might be slow. Increase the number of neighbors to avoid t
his issue.
  self._fit_transform(X)
/Users/onurgumus/Desktop/Python ile Projeler/Scikit-Learn/.venv/lib/python
3.10/site-packages/scipy/sparse/_index.py:100: SparseEfficiencyWarning: Cha
nging the sparsity structure of a csr_matrix is expensive. lil_matrix is mo
re efficient.
  self._set_intXint(row, col, x.flat[0])
```

Out[ ]:
▾ Isomap

Isomap()

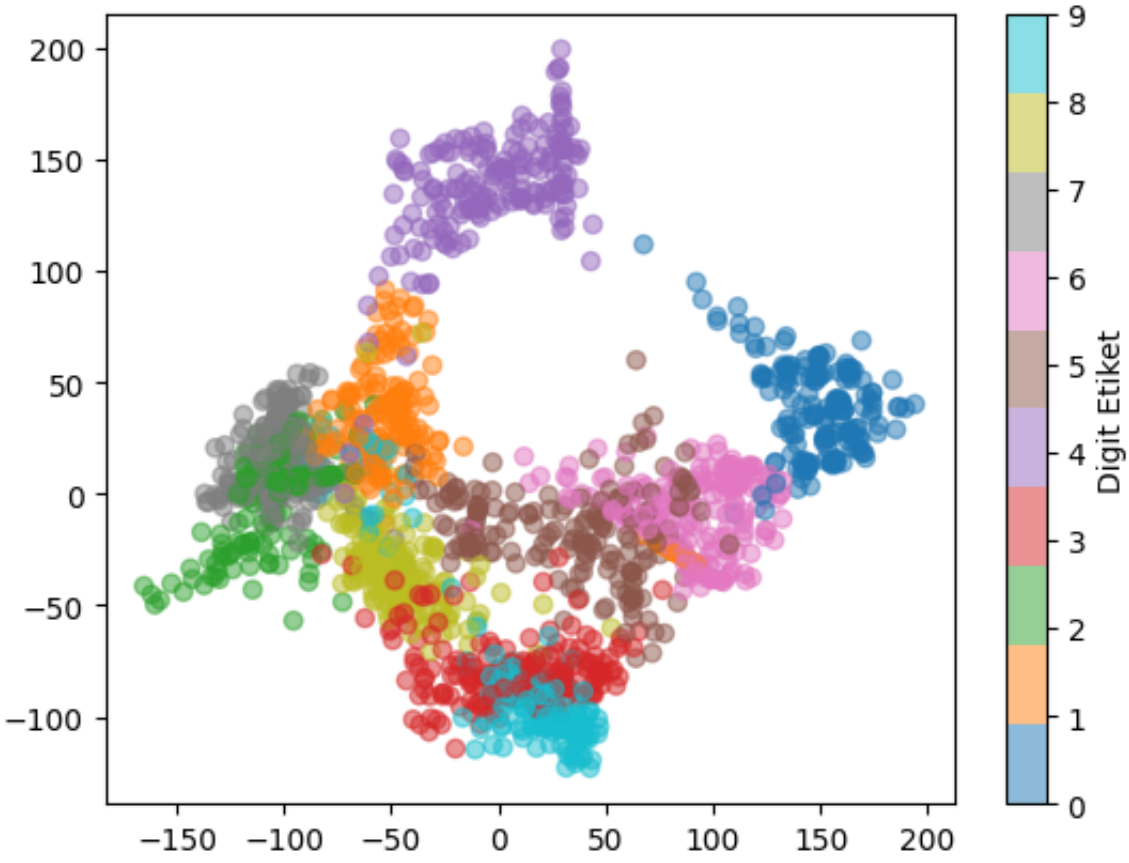
In [ ]:
```
# Transform the features from the digits dataset using the trained Isomap
# Eğitilmiş Isomap modelini kullanarak digits veri kümesindeki özellikleri

data2 = iso.transform(X)
data2.shape
```

Out[ ]: (1797, 2)

In [ ]:
```
# Scatter plot oluştur
# 'data2[:, 0]' x eksenindeki değerleri, 'data2[:, 1]' y eksenindeki değer
# 'c=digits.target' her bir noktanın rengini 'digits.target' dizisinin değ
# 'alpha=0.5' noktaların saydamlığını belirler
# 'cmap=plt.cm.get_cmap('tab10', 10)' renk haritasını belirler; 'tab10' ha
plt.scatter(data2[:, 0], data2[:, 1], c=digits.target, alpha=0.5, cmap=plt

# Renk skalasını ekleyerek renklerle digit etiketleri göster
# 'label='Digit Etiket'' renk skalasının etiketini belirler
# 'ticks=range(10)' renk skalasındaki işaretlerin konumlarını belirler; bu
plt.colorbar(label='Digit Etiket', ticks=range(10))
```

```
/var/folders/26/pvvz5dxx7lb978b1_d113_r40000gn/T/ipykernel_1466/1387186522.
py:6: MatplotlibDeprecationWarning: The get_cmap function was deprecated in
Matplotlib 3.7 and will be removed two minor releases later. Use ``matplotl
ib.colormaps[name]`` or ``matplotlib.colormaps.get_cmap(obj)`` instead.
  plt.scatter(data2[:, 0], data2[:, 1], c=digits.target, alpha=0.5, cmap=pl
t.cm.get_cmap('tab10', 10))
```

Out[ ]: <matplotlib.colorbar.Colorbar at 0x28dea3190>



In [ ]:
```
# Split the digits dataset into training and testing sets using train_test_
# train_test_split kullanarak digits veri kümesini eğitim ve test setlerin

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

In [ ]:
```
# Import the Gaussian Naive Bayes class from scikit-learn naive_bayes modu
# scikit-learn naive_bayes modülünden Gaussian Naive Bayes sınıfını içe ak

# Bu satırda, scikit-learn kütüphanesinden Gaussian Naive Bayes sınıfını i
# Gaussian Naive Bayes, özellikler arasındaki bağımsızlık varsayımını koru
from sklearn.naive_bayes import GaussianNB
```

In [ ]:
```
# Create a Gaussian Naive Bayes model
# Gaussian Naive Bayes modeli oluştur

model = GaussianNB()
```

In [ ]:
```
# Fit the Gaussian Naive Bayes model to the training data
# Gaussian Naive Bayes modelini eğitim verisine uydur

model.fit(X_train, y_train)
```

Out[ ]:
```
▾ GaussianNB
GaussianNB()
```

In [ ]:
```
# Use the trained Gaussian Naive Bayes model to predict labels for the tes
# Eğitilmiş Gaussian Naive Bayes modelini kullanarak test verisi için etik

y_model = model.predict(X_test)
```

In [ ]:
```
# Calculate the accuracy score by comparing predicted labels with true lab
# Tahmin edilen etiketlerle gerçek etiketleri karşılaştırarak doğruluk sko

accuracy_score(y_test, y_model)
```

Out[ ]:
```
0.8333333333333334
```

In [ ]:
```
# Import the confusion_matrix function from scikit-learn metrics module
# scikit-learn metrics modülünden confusion_matrix fonksiyonunu içe aktar

from sklearn.metrics import confusion_matrix

# Calculate the confusion matrix by comparing predicted labels with true l
# Tahmin edilen etiketlerle gerçek etiketleri karşılaştırarak karışıklık m

# Bu kod bloğu, confusion_matrix fonksiyonunu kullanarak test verisi için
# Karışıklık matrisi, sınıflandırma modelinin performansını değerlendirmek
mat = confusion_matrix(y_test, y_model)
```

In [ ]:
```
# Import the heatmap function from the seaborn library
# seaborn kütüphanesinden heatmap fonksiyonunu içe aktar

import seaborn as sns

# Visualize the confusion matrix using a heatmap
# Karışıklık matrisini bir heatmap ile görselleştir

# Bu kod bloğu, seaborn kütüphanesinin heatmap fonksiyonunu kullanarak kar
# square=True parametresi, heatmap'in kare biçiminde olmasını sağlar.
# annot=True parametresi, her hücredeki değerlerin heatmap üzerinde görünt
# cbar=False parametresi ise renk çubuğunu (colorbar) kapatır.
# plt.xlabel ve plt.ylabel fonksiyonları ile x ve y ekseni etiketleri ekle
sns.heatmap(mat, square=True, annot=True, cbar=False)
plt.xlabel('Tahmin Değer')
plt.ylabel('Gerçek Değer')
```

Out[ ]: Text(113.9222222222222, 0.5, 'Gerçek Değer')