Here, I will be working with the **digits** dataset from sklearn library.

About the digits dataset:

- The dataset contains **1797 images** of handwritten digits, from **0** to **9**.
- The size of each image is **8x8 pixels**.
- The dataset is **balanced**: each category within it has an approximately equal number of instances.
- The data is stored in the `data` attribute, which is an array of size `n_samples` by `n_features`.
- The `target` attribute of the dataset stores the digit that each image represents.
- The `images` attribute of the dataset stores 8x8 arrays of grayscale values for each image.

```
# Import the libraries
import numpy as np
import pandas as pd
from sklearn import datasets, decomposition
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

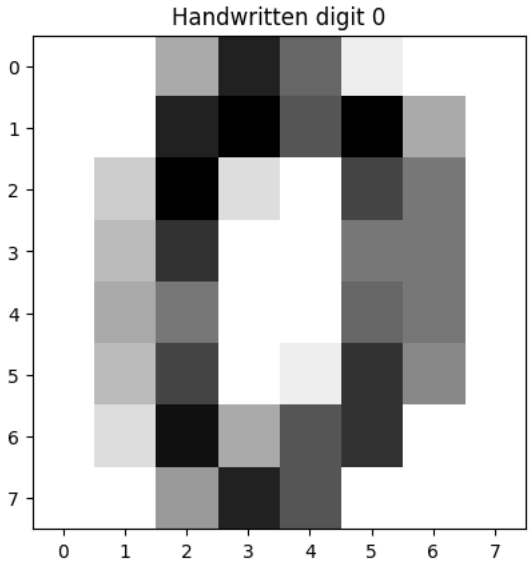
```
# Load the dataset
digits = datasets.load_digits()
```

+ Code + Text

First, I will explore the images attribute first.

```
print('The size of the images attribute: \n', digits.images.shape)
print('The matrix representing the first image: \n', digits.images[0])
plt.imshow(digits.images[0], cmap=plt.cm.gray_r)
plt.title(f"Handwritten digit {digits.target[0]}")
plt.show()
```

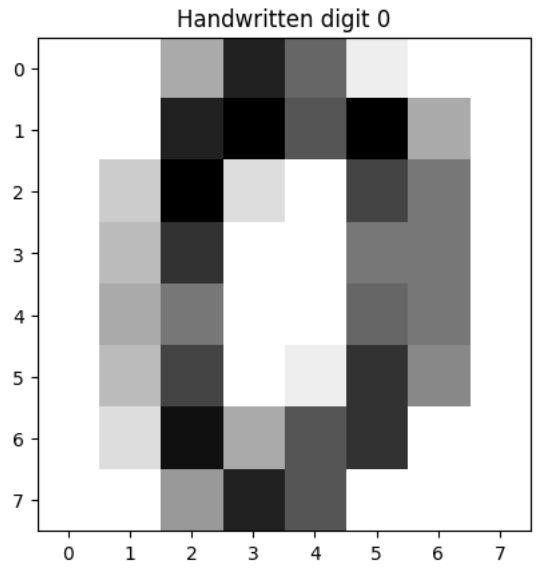
```
The size of the images attribute:
 (1797, 8, 8)
The matrix representing the first image:
 [[ 0.  0.  5. 13.  9.  1.  0.  0.]
 [ 0.  0. 13. 15. 10. 15.  5.  0.]
 [ 0.  3. 15.  2.  0. 11.  8.  0.]
 [ 0.  4. 12.  0.  0.  8.  8.  0.]
 [ 0.  5.  8.  0.  0.  9.  8.  0.]
 [ 0.  4. 11.  0.  1. 12.  7.  0.]
 [ 0.  2. 14.  5. 10. 12.  0.  0.]
 [ 0.  0.  6. 13. 10.  0.  0.  0.]]
```

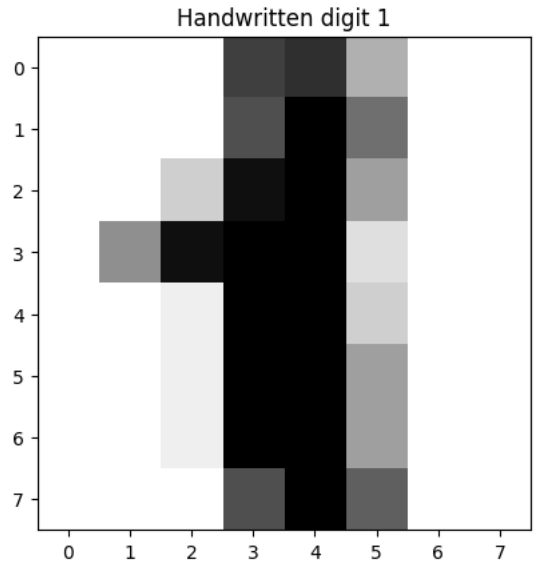


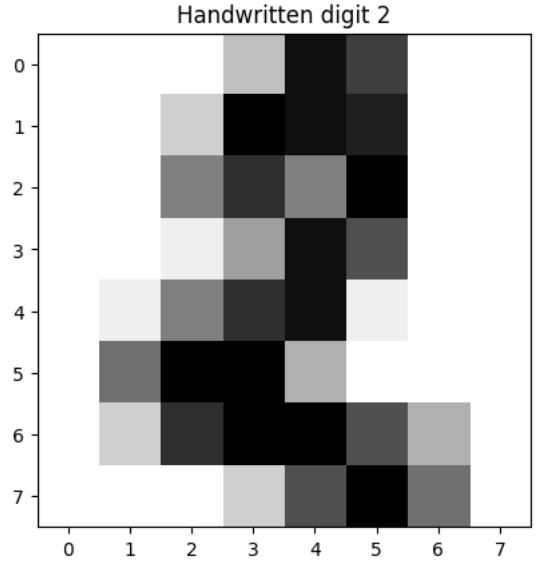
Next, I will plot the first 16 images from the dataset in one figure.

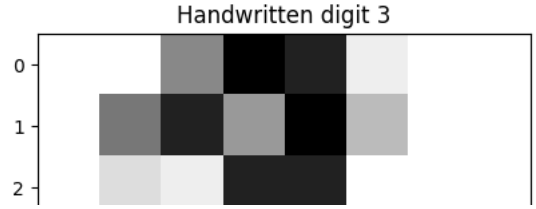
```
for i in range(17):
    plt.imshow(digits.images[i], cmap=plt.cm.gray_r)
```

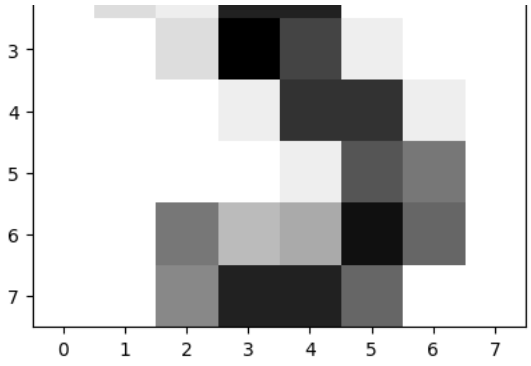
```
        plt.title(f"Handwritten digit {digits.target[i]}")
        plt.show()
```

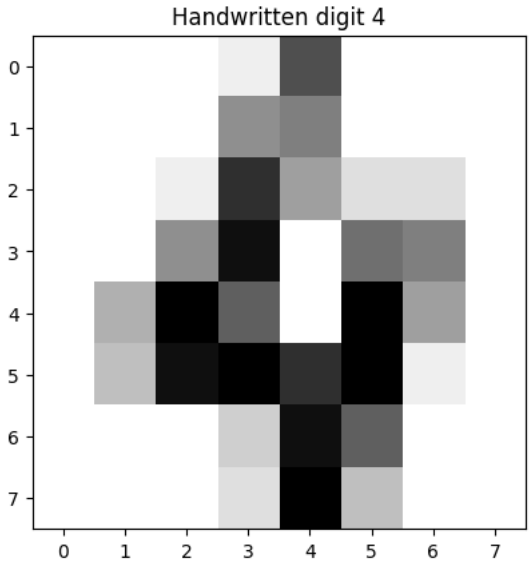

Handwritten digit 0
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7

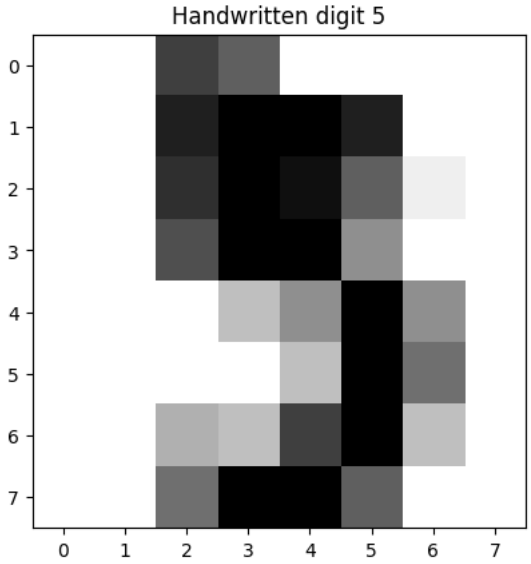

Handwritten digit 1
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7

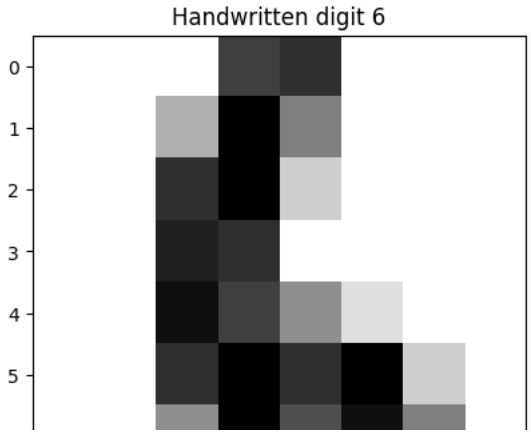

Handwritten digit 2
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7

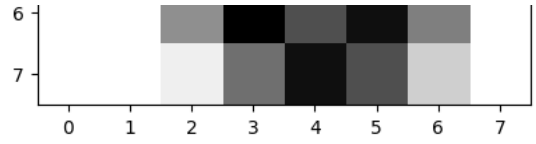

Handwritten digit 3
0 1 2

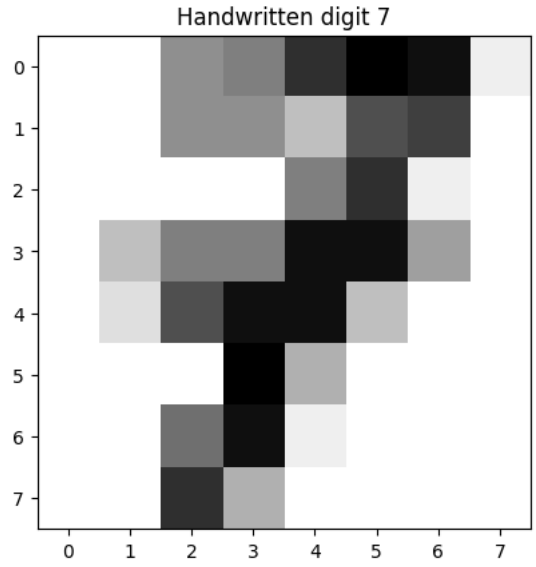


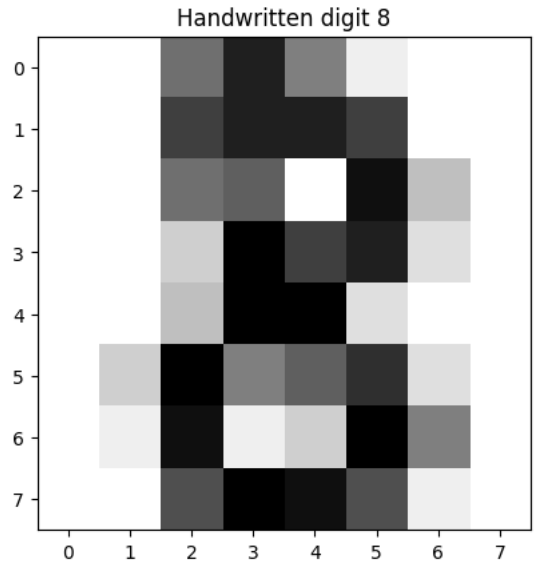

Handwritten digit 4

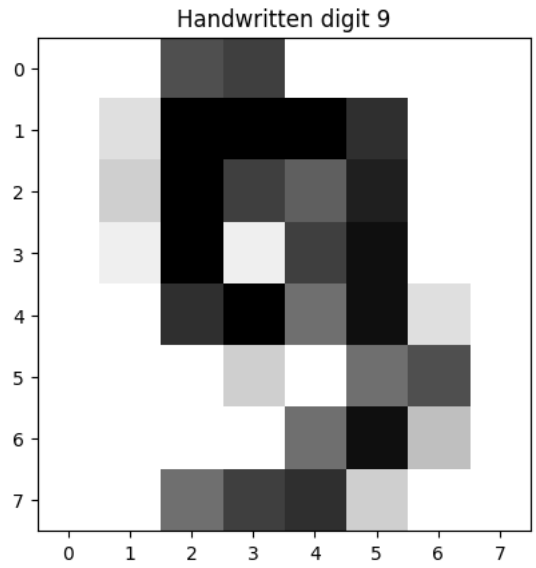

Handwritten digit 5

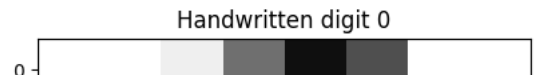

Handwritten digit 6

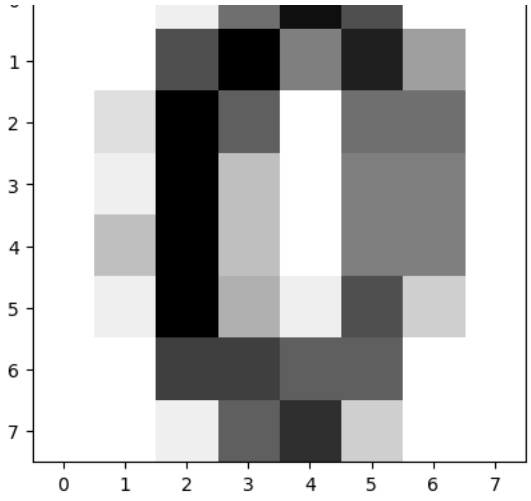


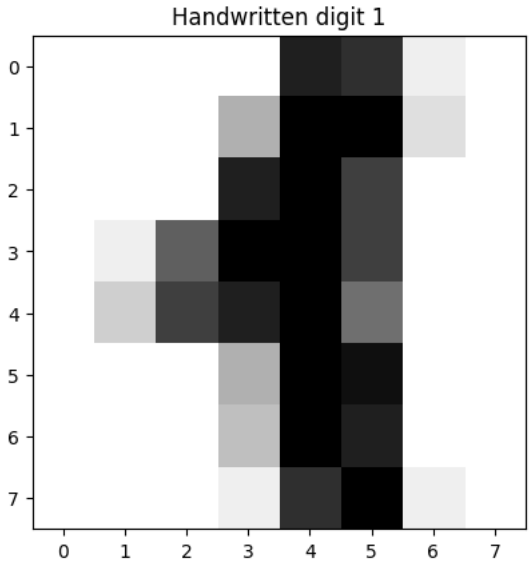

Handwritten digit 7

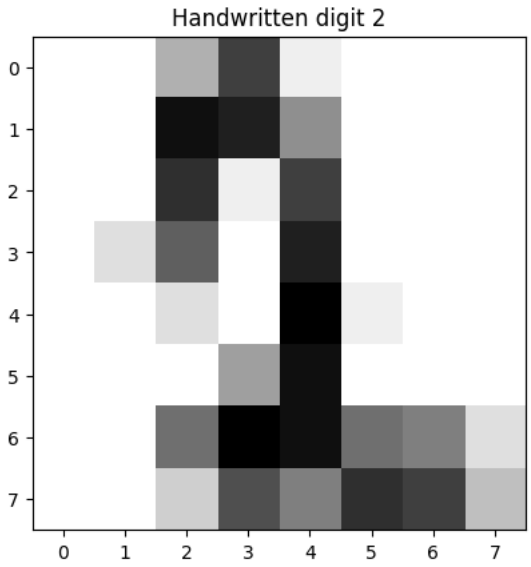

Handwritten digit 8

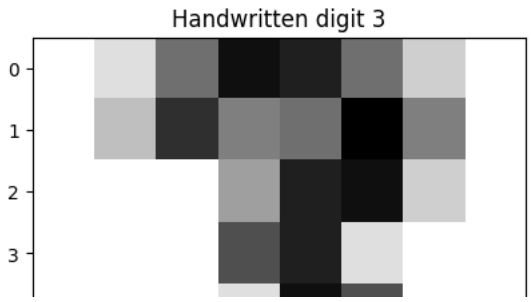

Handwritten digit 9

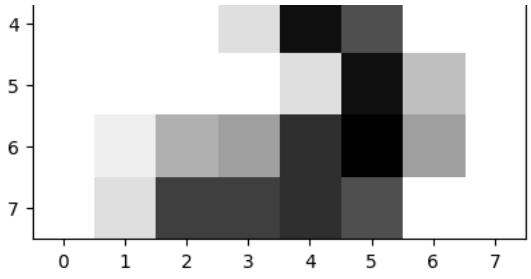

Handwritten digit 0

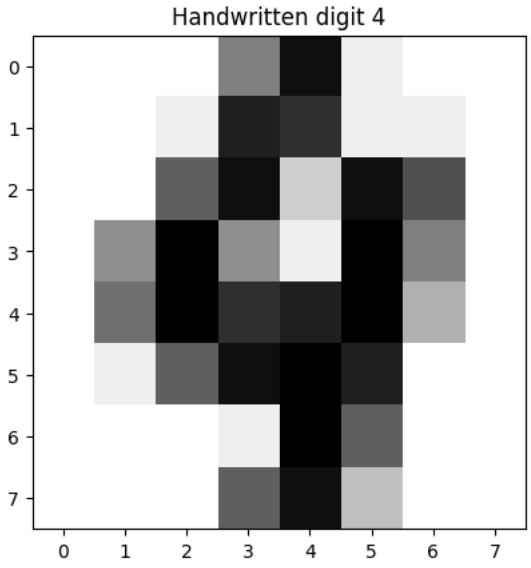


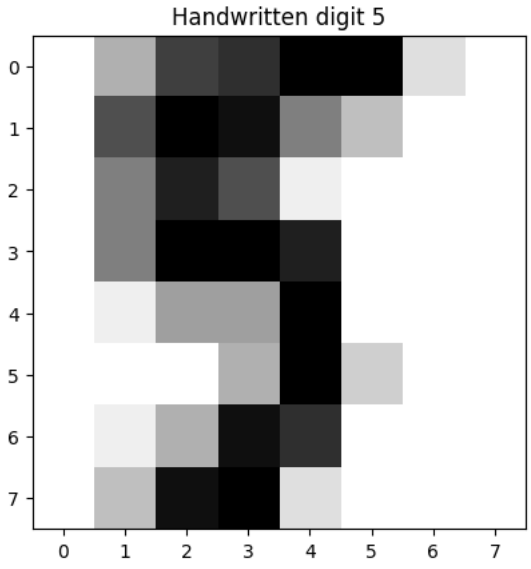

Handwritten digit 1

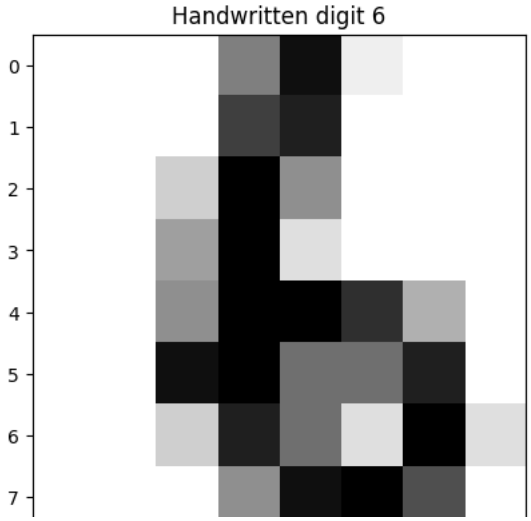

Handwritten digit 2


Handwritten digit 3




Handwritten digit 4


Handwritten digit 5


Handwritten digit 6


0
1
2
3
4
5
6
7

```
X_0 = digits.data
print('The size of the data attribute: \n', X_0.shape)
```

```
The size of the data attribute:
 (1797, 64)
```

The data and images attributes contain the same information in different shapes.

```
print('The matrix representing the first image: \n', digits.images[0])
print('The vector representing the first image: \n', X_0[0])
```

```
The matrix representing the first image:
 [[ 0.  0.  5. 13.  9.  1.  0.  0.]
 [ 0.  0. 13. 15. 10. 15.  5.  0.]
 [ 0.  3. 15.  2.  0. 11.  8.  0.]
 [ 0.  4. 12.  0.  0.  8.  8.  0.]
 [ 0.  5.  8.  0.  0.  9.  8.  0.]
 [ 0.  4. 11.  0.  1. 12.  7.  0.]
 [ 0.  2. 14.  5. 10. 12.  0.  0.]
 [ 0.  0.  6. 13. 10.  0.  0.  0.]]
The vector representing the first image:
 [ 0.  0.  5. 13.  9.  1.  0.  0.  0.  0. 13. 15. 10. 15.  5.  0.  0.  3.
 15.  2.  0. 11.  8.  0.  0.  4. 12.  0.  0.  8.  8.  0.  0.  5.  8.  0.
  0.  9.  8.  0.  0.  4. 11.  0.  1. 12.  7.  0.  0.  2. 14.  5. 10. 12.
  0.  0.  0.  0.  6. 13. 10.  0.  0.  0.]
```

Next, I wanted to see if X_0 centered

```
feature_means = np.mean(X_0, axis=0)

# check if dataset is centered
print("Feature means:\n", feature_means)
print("\nIs the dataset centered? ", np.allclose(feature_means, 0))
```

```
Feature means:
 [0.00000000e+00 3.03839733e-01 5.20478575e+00 1.18358375e+01
 1.18480801e+01 5.78185865e+00 1.36227045e+00 1.29660545e-01
 5.56483027e-03 1.99387869e+00 1.03823038e+01 1.19794101e+01
 1.02793545e+01 8.17584864e+00 1.84641068e+00 1.07957707e-01
 2.78241514e-03 2.60155815e+00 9.90317195e+00 6.99276572e+00
 7.09794101e+00 7.80634391e+00 1.78853645e+00 5.00834725e-02
 1.11296605e-03 2.46967168e+00 9.09126322e+00 8.82136895e+00
 9.92710072e+00 7.55147468e+00 2.31775181e+00 2.22593211e-03
 0.00000000e+00 2.33945465e+00 7.66722315e+00 9.07178631e+00
 1.03016138e+01 8.74401781e+00 2.90929327e+00 0.00000000e+00
 8.90372844e-03 1.58375070e+00 6.88146912e+00 7.22815804e+00
 7.67223150e+00 8.23650529e+00 3.45631608e+00 2.72676683e-02
 7.23427935e-03 7.04507513e-01 7.50695604e+00 9.53923205e+00
 9.41624930e+00 8.75848637e+00 3.72509738e+00 2.06455203e-01
 5.56483027e-04 2.79354480e-01 5.55759599e+00 1.20890373e+01
 1.18091263e+01 6.76405120e+00 2.06789093e+00 3.64496383e-01]

Is the dataset centered?  False
```

The feature means indicate that the dataset X_0 is not centered, as the means are not all zero. In fact, some values are far from zero (e.g., ~9 or ~6). This confirms that the dataset needs centering if required for PCA or other analyses.

Next, I will write a function that takes a standardized dataset and an integer k and returns the transformed dataset with k principal components. One can implement PCA using the following steps:

1. Calculate the covariance matrix of the standardized dataset.
2. Compute the eigenvalues and eigenvectors of the covariance matrix.
3. Sort the eigenvalues in descending order and select the top k.
4. Select the eigenvectors corresponding to the k largest eigenvalues.
5. Project the standardized dataset onto the k principal components.

```
#X is the standarized dataset
```

```
# Load the dataset
digits = datasets.load_digits()

#  Function to perform PCA and return k principal components
def pca_transform(X, k):
     # Ensure data is standardized (centered and scaled)
    X_centered = X - np.mean(X, axis=0)

    # Calculate the covariance matrix
    cov_matrix = np.cov(X_centered.T)

    # Compute eigenvalues and eigenvectors of the covariance matrix
    eigenvalues, eigenvectors = np.linalg.eigh(cov_matrix)

    # Sort the eigenvalues in descending order and select the top k
    sorted_indices = np.argsort(eigenvalues)[::-1]
    top_k_indices = sorted_indices[:k]

    # Select the eigenvectors corresponding to the k largest eigenvalues
    top_k_eigenvectors = eigenvectors[:, top_k_indices]

    # Step 5: Project the dataset onto the k principal components
    X_pca = np.dot(X_centered, top_k_eigenvectors)

    return X_pca
```

Using the previous function, i will make a scatter plot to visualize the projected dataset with $k = 2$.

```
k = 2
X_0 = digits.data
X_pca = pca_transform(X_0, k)

# Scatter plot to visualize the projected dataset with k=2
plt.figure(figsize=(10, 8))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=digits.target, cmap='viridis', edgecolor='k', s=50)
plt.colorbar(label='Digit Label')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA of Digits Dataset with k=2')
plt.show()
```