Handwritten Digits Clustering

In this code snippet, you are going to practice *K-means* clustering using `scikit-learn` (https://scikit-learn.org), which is the well-known machine learning package in Python. We cluster samples of a dataset, containing 8x8 pixel images of handwritten digits (totally 10 clusters for `0` to `9`). Then, we will see how to assign a new sample to the corresponding cluster by comparing the sample distance to the centroids.

In [1]:
```
import numpy as np
from sklearn.datasets import load_digits
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from utils import plot_images, plot_clusters, plot_centroids
```

Step 1. Load Data

The handwritten image dataset in the `scikit-learn` package contains 1797 samples of 10 digits (around 180 samples per class). We use `load_digits` (https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_digits.html) function to load the dataset.

In [2]:
```
X, y = load_digits(return_X_y=True)
print(np.shape(X))
plot_images(X)
```

```
(1797, 64)
```

Split Test and Train Sets

Using the `train_test_split` (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html) in `scikit-learn.model_selection`, you can shuffle the dataset randomly; then, split the dataset into train and test sets according to your desired train or test size.

In [3]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=20)

print("Train set: ")
print(np.shape(X_train))
plot_images(X_train)

print("Test set: ")
print(np.shape(X_test))
plot_images(X_test)
```

```
Train set:
(1777, 64)
```

```
Test set:
(20, 64)
```

Step 2. K-means Clustering

The [`KMeans` (https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html#sklearn.cluster.KMeans)](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html#sklearn.cluster.KMeans) in the `scikit-learn` package is convenient to use. The init function to initialize an instance of the class is defined as follows:

`KMeans(n_clusters, n_init, max_iter)`

- `n_clusters` : The number of clusters to form as well as the number of centroids to generate.
- `n_init` : Number of time the k-means algorithm will be run with different centroid seeds. The final results will be the best output of n_init consecutive runs in terms of inertia.
- `max_iter` : Maximum number of iterations of the k-means algorithm for a single run.

Then, the `fit(X=input)` function clusters the input into groups.

In [4]:
```
kmeans_obj = KMeans(n_clusters=10, n_init=50, max_iter=100)
clusters_train = kmeans_obj.fit(X_train)
plot_clusters(X_train, clusters_train.labels_)
```

```
cluster 0: (163, 64)
```

```
cluster 1: (244, 64)
```

```
cluster 2: (219, 64)
```

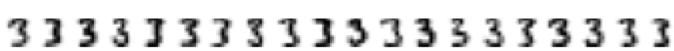
```
cluster 3: (146, 64)
```

```
cluster 4: (177, 64)
```

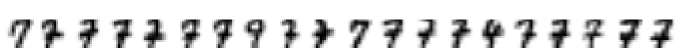
```
cluster 5: (170, 64)
```

```
cluster 6: (181, 64)
```

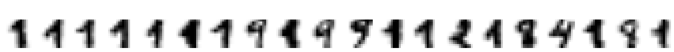
```
cluster 7: (180, 64)
```



```
cluster 8: (205, 64)
```



```
cluster 9: (92, 64)
```



Step 3. Test New Samples

The `predict` function evaluates unseen samples to predict the closest cluster each sample in X belongs to.

In [5]:
```
clusters_test = kmeans_obj.predict(X_test)
plot_images(X_test)
print(clusters_test)
```

```
[0 8 4 3 8 5 8 1 0 7 6 2 5 2 3 4 7 5 0 5]
```

In [6]:
```
plot_centroids(clusters_train, clusters_test)
```