seek to learn, from the properties of the data, an optimal division or discrete labeling of groups of points.

Many clustering algorithms are available in Scikit-Learn and elsewhere, but perhaps the simplest to understand is an algorithm known as *k-means clustering*, which is implemented in `sklearn.cluster.KMeans`. We begin with the standard imports:

```
In[1]: %matplotlib inline
       import matplotlib.pyplot as plt
       import seaborn as sns; sns.set()  # for plot styling
       import numpy as np
```

Introducing k-Means

The *k*-means algorithm searches for a predetermined number of clusters within an unlabeled multidimensional dataset. It accomplishes this using a simple conception of what the optimal clustering looks like:

- The "cluster center" is the arithmetic mean of all the points belonging to the cluster.
- Each point is closer to its own cluster center than to other cluster centers.

Those two assumptions are the basis of the *k*-means model. We will soon dive into exactly *how* the algorithm reaches this solution, but for now let's take a look at a simple dataset and see the *k*-means result.

First, let's generate a two-dimensional dataset containing four distinct blobs. To emphasize that this is an unsupervised algorithm, we will leave the labels out of the visualization (Figure 5-110):

```
In[2]: from sklearn.datasets.samples_generator import make_blobs
       X, y_true = make_blobs(n_samples=300, centers=4,
                              cluster_std=0.60, random_state=0)
       plt.scatter(X[:, 0], X[:, 1], s=50);
```

By eye, it is relatively easy to pick out the four clusters. The *k*-means algorithm does this automatically, and in Scikit-Learn uses the typical estimator API:

```
In[3]: from sklearn.cluster import KMeans
       kmeans = KMeans(n_clusters=4)
       kmeans.fit(X)
       y_kmeans = kmeans.predict(X)
```

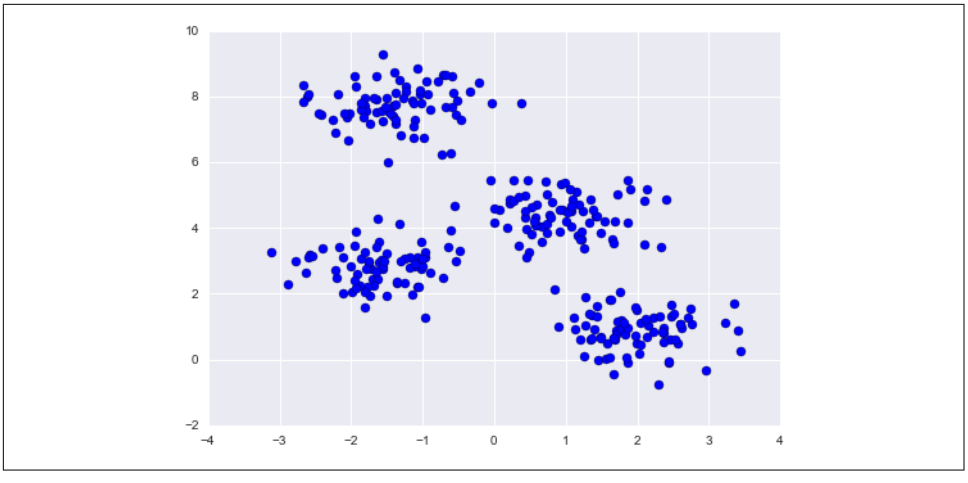

Figure 5-110. Data for demonstration of clustering

Let's visualize the results by plotting the data colored by these labels. We will also plot the cluster centers as determined by the *k*-means estimator (Figure 5-111):

```
In[4]: plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')

       centers = kmeans.cluster_centers_
       plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5);
```

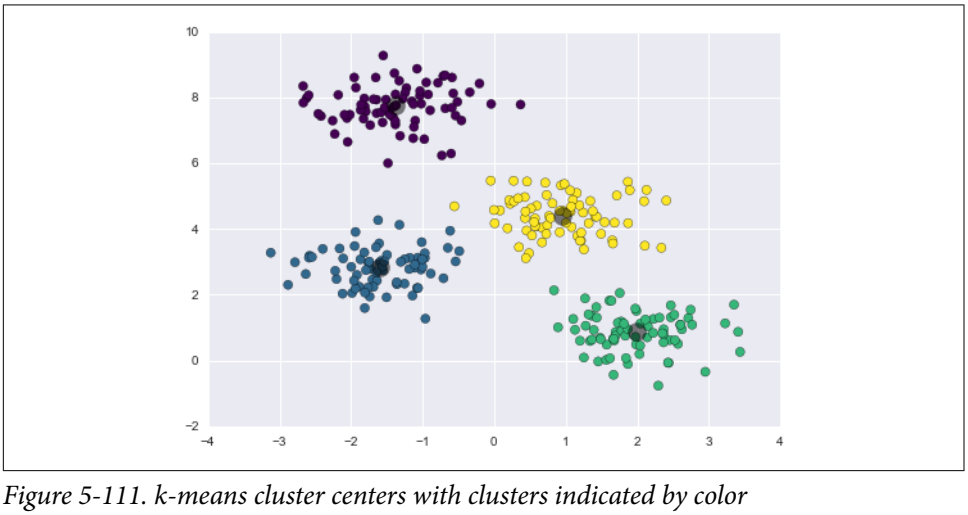

Figure 5-111. k-means cluster centers with clusters indicated by color

The good news is that the *k*-means algorithm (at least in this simple case) assigns the points to clusters very similarly to how we might assign them by eye. But you might wonder how this algorithm finds these clusters so quickly! After all, the number of possible combinations of cluster assignments is exponential in the number of data

points—an exhaustive search would be very, very costly. Fortunately for us, such an exhaustive search is not necessary; instead, the typical approach to *k*-means involves an intuitive iterative approach known as *expectation–maximization*.

k-Means Algorithm: Expectation–Maximization

Expectation–maximization (E–M) is a powerful algorithm that comes up in a variety of contexts within data science. *k*-means is a particularly simple and easy-to-understand application of the algorithm, and we will walk through it briefly here. In short, the expectation–maximization approach consists of the following procedure:

1. Guess some cluster centers
2. Repeat until converged
   a. *E-Step*: assign points to the nearest cluster center
   b. *M-Step*: set the cluster centers to the mean

Here the "E-step" or "Expectation step" is so named because it involves updating our expectation of which cluster each point belongs to. The "M-step" or "Maximization step" is so named because it involves maximizing some fitness function that defines the location of the cluster centers—in this case, that maximization is accomplished by taking a simple mean of the data in each cluster.

The literature about this algorithm is vast, but can be summarized as follows: under typical circumstances, each repetition of the E-step and M-step will always result in a better estimate of the cluster characteristics.

We can visualize the algorithm as shown in Figure 5-112.

For the particular initialization shown here, the clusters converge in just three iterations. For an interactive version of this figure, refer to the code in the online appendix.

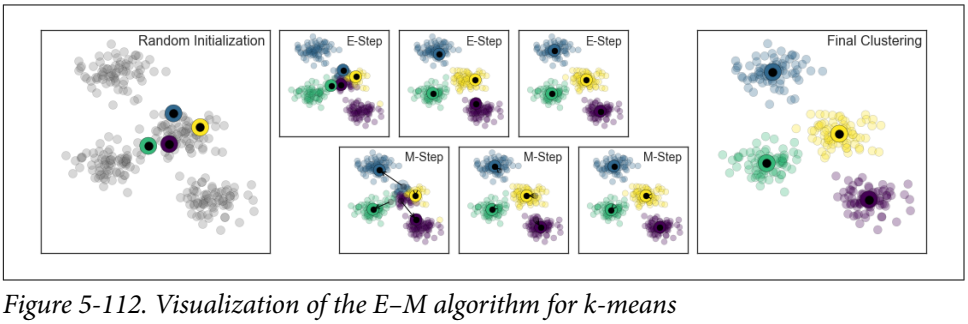

Figure 5-112. Visualization of the E–M algorithm for k-means

The *k*-means algorithm is simple enough that we can write it in a few lines of code. The following is a very basic implementation (Figure 5-113):