<https://jakevdp.github.io/PythonDataScienceHandbook/05.11-k-means.html>

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.

%**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 pre-determined 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

**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);

**from** **sklearn.cluster** **import** KMeans

kmeans = KMeans(n\_clusters=4)

kmeans.fit(X)

y\_kmeans = kmeans.predict(X)

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);

**from** **sklearn.metrics** **import** pairwise\_distances\_argmin

**def** find\_clusters(X, n\_clusters, rseed=2):

*# 1. Randomly choose clusters*

rng = np.random.RandomState(rseed)

i = rng.permutation(X.shape[0])[:n\_clusters]

centers = X[i]

**while** **True**:

*# 2a. Assign labels based on closest center*

labels = pairwise\_distances\_argmin(X, centers)

*# 2b. Find new centers from means of points*

new\_centers = np.array([X[labels == i].mean(0)

**for** i **in** range(n\_clusters)])

*# 2c. Check for convergence*

**if** np.all(centers == new\_centers):

**break**

centers = new\_centers

**return** centers, labels

centers, labels = find\_clusters(X, 4)

plt.scatter(X[:, 0], X[:, 1], c=labels,

s=50, cmap='viridis');