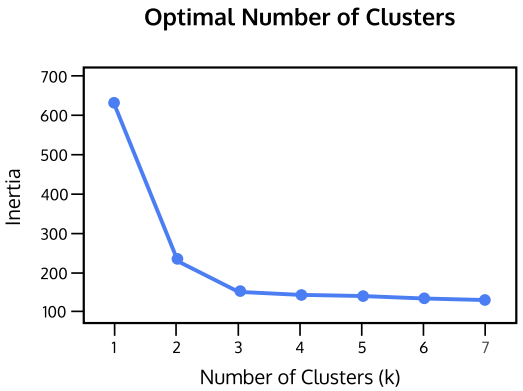
Clustering: K-Means

K-Means: Inertia

Inertia measures how well a dataset was clustered by K-Means. It is calculated by measuring the distance between each data point and its centroid, squaring this distance, and summing these squares across one cluster.

A good model is one with low inertia AND a low number of clusters (K). However, this is a tradeoff because as K increases, inertia decreases.

To find the optimal K for a dataset, use the *Elbow method*; find the point where the decrease in inertia begins to slow. $K=3$ is the "elbow" of this graph.



Unsupervised Learning Basics

Patterns and structure can be found in unlabeled data using *unsupervised learning*, an important branch of machine learning. *Clustering* is the most popular unsupervised learning algorithm; it groups data points into clusters based on their similarity. Because most datasets in the world are unlabeled, unsupervised learning algorithms are very applicable.

Possible applications of clustering include:

- Search engines: grouping news topics and search results
- Market segmentation: grouping customers based on geography, demographics, and behaviors

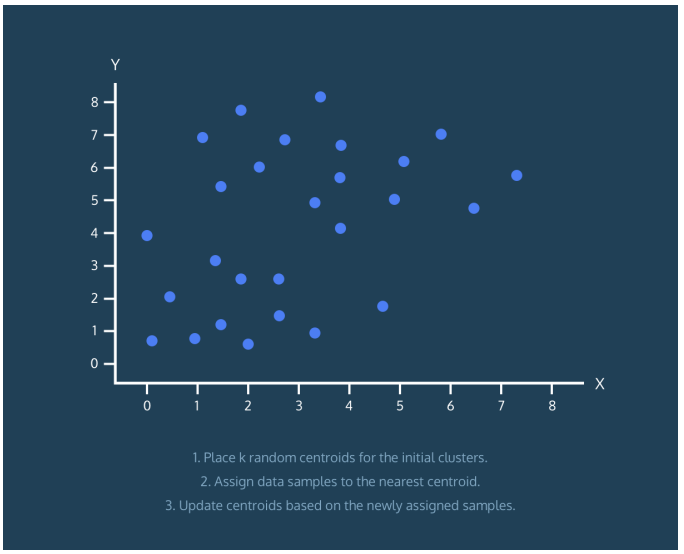
codecademy

K-Means Algorithm: Intro

K-Means is the most popular clustering algorithm. It uses an iterative technique to group unlabeled data into K clusters based on cluster centers (*centroids*). The data in each cluster are chosen such that their average distance to their respective centroid is *minimized*.

1. Randomly place K centroids for the initial clusters.
2. Assign each data point to their nearest centroid.
3. Update centroid locations based on the locations of the data points.

Repeat Steps 2 and 3 until points don't move between clusters and centroids stabilize.



K-Means Algorithm: 2nd Step

After randomly choosing centroid locations for *K-Means*, each data sample is allocated to its closest centroid to start creating more precise clusters. The distance between each data sample and every centroid is calculated, the minimum distance is selected, and each data sample is assigned a label that indicates its closest cluster.
The distance formula is implemented as `.distance()` and used for each data point. `np.argmin()` is used to find the minimum distance and find the cluster at that distance.

```
# distance formula
def distance(a, b):
  one = (a[0] - b[0]) **2
  two = (a[1] - b[1]) **2
  distance = (one+two) ** 0.5
  return distance
```

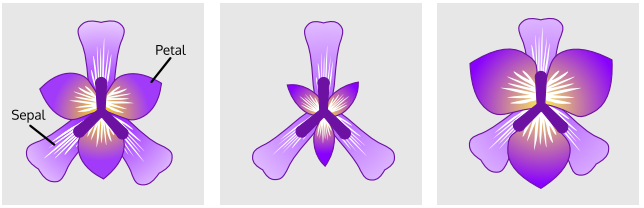
codecademy

Scikit-Learn Datasets

The `scikit-learn` library contains built-in datasets in its `datasets` module that are often used in machine learning problems like classification or regression.
Examples:

- Iris dataset (classification)
- Boston house-prices dataset (regression)

The format of these datasets are important to their use with algorithms. For example, each piece of data in the Iris dataset is a *sample* (flower type), and each element within a sample is a *feature* (i.e. petal width).



Iris Versicolor | Iris Setosa | Iris Virginica

K-Means Using Scikit-Learn

Scikit-Learn, or `sklearn`, is a machine learning library for Python that has a K-Means algorithm implementation that can be used instead of creating one from scratch.
To use it:

- Import the `KMeans()` method from the `sklearn.cluster` library to build a model with `n_clusters`
- Fit the model to the data samples using `.fit()`
- Predict the cluster that each data sample belongs to using `.predict()` and store these as `labels`

```
from sklearn.cluster import KMeans

model = KMeans(n_clusters=3)

model.fit(data_samples)

labels = model.predict(data_samples)
```

codecademy

Cross Tabulation Overview

Cross-tabulations involve grouping pieces of data together in order to examine their relationship in a different way. Sometimes correlations within data can be seen better when not just looking at total responses.

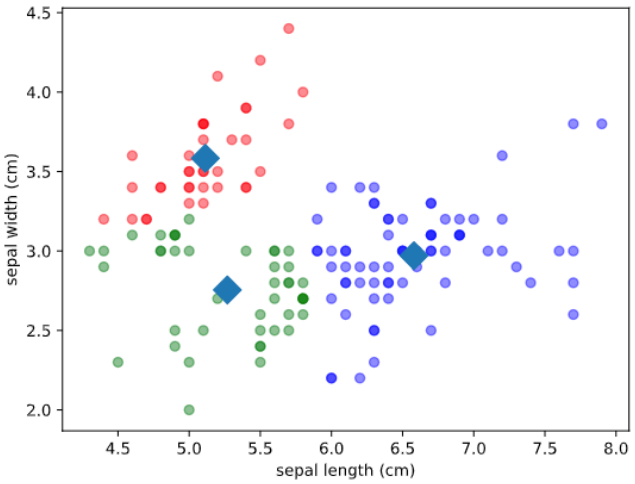
This technique is often performed in Python after running K-Means; the Pandas method `.crosstab()` allows for comparison between resulting cluster labels and user-defined labels for each data sample. In order to validate the results of a K-Means model with this technique, there must be user-defined labels for all data samples.

```
import pandas as pd

cross_tab =
pd.crosstab(df['pred_labels'],
df['user_labels'])
```

K-Means: Reaching Convergence

In K-Means, after placing K random centroids, the data samples are repeatedly assigned to the nearest centroid and then centroid locations are updated. This continues until each of the centroids' coordinates converge, or stop changing. This sequence of events can be implemented in Python using a `while` loop. The loop continues until the difference between each element of the updated `centroids` and each element of the past `centroids_old` is 0. This will mean the centroids have converged and the clusters are complete!

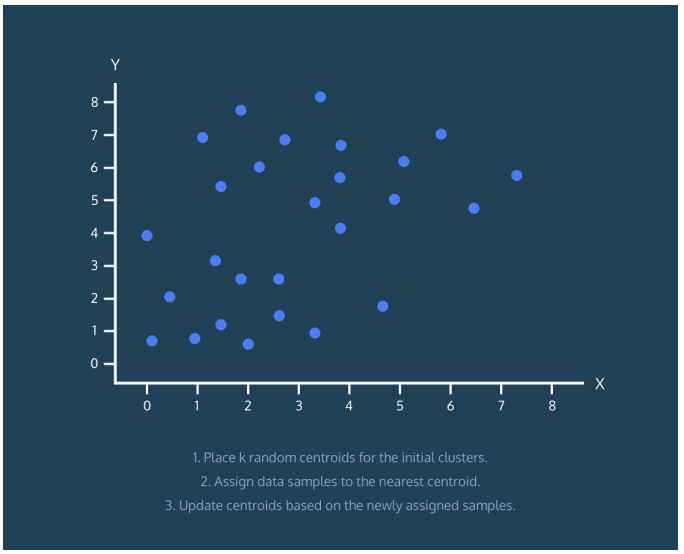


codecademy

K-Means Algorithm: 3rd Step

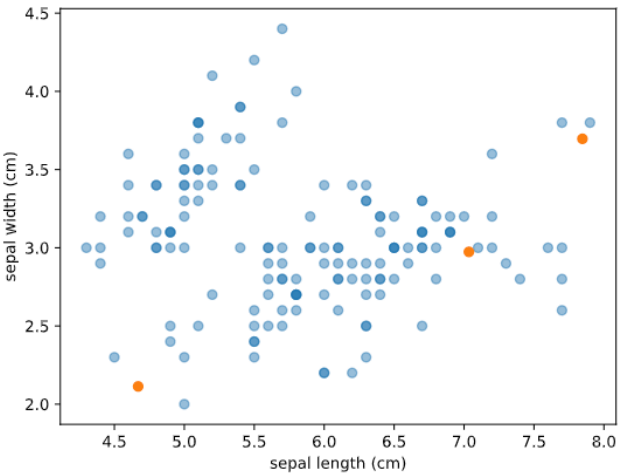
The third step of K-Means updates centroid locations. After the data are assigned to their respectively closest centroid in step 2, each cluster center location is adjusted to be the average of its assigned data points.

The NumPy `.mean()` function is used to find the average x and y-coordinates of all data points for each cluster and store these as the new centroid locations.



K-Means Algorithm: 1st Step

The first step of the K-Means clustering algorithm requires placing K random centroids which will become the centers of the K initial clusters. This step can be implemented in Python using the Numpy `random.uniform()` function; the x and y-coordinates are randomly chosen within the x and y ranges of the data points.



Print Share

0

codecademy