Kmeans:

In short summary a Kmeans algorithm is a clustering option you can use and it optimizes on its own. This is usefull if you don’t have prior knowledge of the data you want to analyze.

It works by putting a number of clusters k (you can pick those yourself) random in the datafield and then assigning datapoints to each cluster. After this the program optimizes so the clusters are as unique as possible.

To better understand what a Kmeans algorithm does you can follow this tutorial and see for yourself what happens. Open the code that is given to you via canvas.

Note what libraries are imported this might help you if you get stuck.

We start with a simple DataFrame which you can use for this tutorial:

DataFrame: df = pd.DataFrame({

'x': [12, 20, 28, 18, 29, 33, 24, 45, 45, 52, 51, 52, 55, 53, 55, 61, 64, 69, 72, 72, 70, 79, 68, 65],

'y': [39, 36, 30, 52, 54, 46, 55, 59, 63, 70, 66, 63, 58, 23, 14, 8, 19, 7, 24, 42, 45, 62, 55, 59]

})

Now that we have the data we want to add some random clusters in here. To make sure we can run the code several times without the random points changing we use the numpy function random.seed(). In between the brackets you can put a number which makes sure the points are random but when you run your code again the points will be on the same random spot.

Now we choose a number of clusters we want to look at. Start with 3 clusters. Ranging between (0.0) & (80.80). check this by plotting the datapoints and clusters.

# hint, give the clusters different colors so you can spot them in all the datapoints and so that you can assign the datapoints to a cluster with a color to make it visible for yourself. A colormap is very useful for this.

If you were successful you should have a similar plot as figure 1

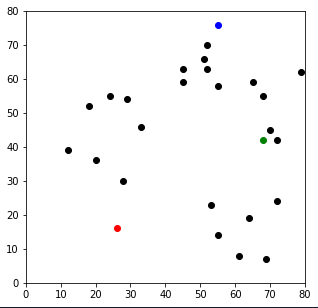


Figure 1, plot of the datapoints with 3 randomly chosen clusters in red green and blue. If for the random seed 200 is chosen

Now you have the clusters it is time to assign each datapoint to the clusters that is closest to it.

Do this by starting with defining a function called ‘assignment’ where the inputs are your DataFrame df and the cluster centroids called centroids. Containing all information of those points.

You want to add columns to your DataFrame called “distance from point “ where you put in the distance the point is from the cluster point you randomly selected. You can calculate it using the Euclidian distance (Pythagoras) .

When you calculated all the distances, you can make extra columns with the closest centroid and the color that the data should get (purely for visualization).

The function should return the DataFrame with all the extra columns. To test this print the head of the new DataFrame. This should look similar to figure 2

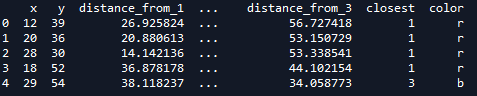


Figure 2, head of the new DataFrame.

When you pulled this off the next step is to plot it and look at which points are part of cluster 1 2 or 3. In figure 3 can be seen what it should look like

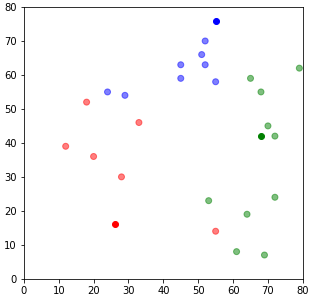


Figure 3, the data with their assigned colors so it can be seen with what cluster they belong.

As can be seen in figure 3 the algorithm does not really work well because we can see that the red point in the bottom right should probably be green. Therefore we are going to optimize.

Until now we just picked the centroids at random. We can see that in this example they are not even close to being right. So first we are going to relocate the centroids to the center of the clusters we just made. To do this we have to find the point with the lowest mean distance from all points in the cluster.

After you update the centroids you can do the same for the assignment of the datapoints. Do this with the help of your previously build function. When you plot the results compare them to the result you got earlier, you can see that points that first were in the red cluster now went to become green and vice versa

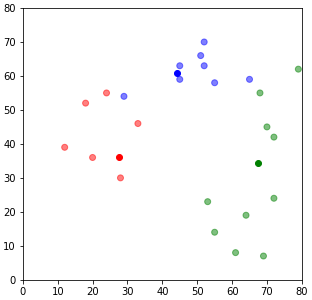
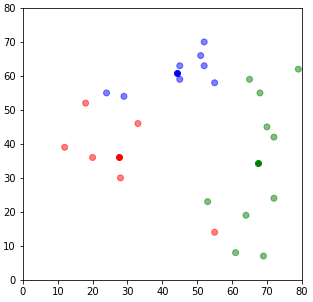
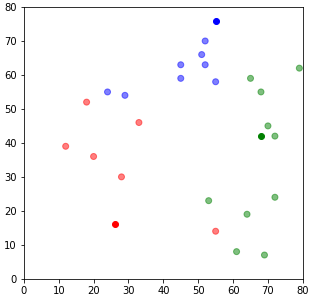


Figure 4, left the old centroids, in the middle the new location of the centroids, right the updated data points

You can see that it is improving but still not quite right. So for the last step in making a self-optimizing loop so you don’t have to check every step of the way because it may take some time when you have a larger DataFrame.