**Assignment 7**

**Clustering**

**Question 1 of 10:** Conduct k-means clustering with K=2, using sklearn’s implementation of KMeans. Look at the Cluster Centers. Which attributes have the biggest difference between cluster 0 and cluster 1?

|  |
| --- |
| **Hint 1:** To recall what k-Means clustering is, you can go back to the Clustering video and find an explanation beginning at 1:41. |
| **Hint 2:** First, you need to load your data into a dataframe. You can do this using the pandas module. In order to do this, you first need to import it, e.g.,  import pandas as pd |
| **Hint 3:** Once you've imported the pandas module, you can now load your data into a dataframe, e.g.,  data = pd.read\_csv(path\_to\_csv) |
| **Hint 4:** Now that you have loaded your data, it is time to fit a k-Means clustering model, where K=2. In order to do this, you first need to import KMeans from the sklearn module as follows:  from sklearn.cluster import KMeans |
| **Hint 5:** Once KMeans is imported, you can now fit your model, e.g.,  kmeans = KMeans(n\_clusters=K).fit(data) |
| **Hint 6:** Remember that K=2. |
| **Hint 7:** Finally, now that your model has been fit on the data, you will want to check the cluster centers for each of the attributes in order to see differences between clusters 0 and 1. |
| **Hint 8:** You can do this by printing the cluster centers, i.e.,  print(kmeans.cluster\_centers\_) |
| **Hint 9:** This will print out a 2D list. Each entry is a list of cluster centers, from attribute A to F. You can tell which attributes have the biggest difference between cluster 0 and 1 by looking at how far apart the centers are between both clusters. Bigger differences between centers indicate a bigger difference between clusters. |
| **Hint 10:** For instance, the cluster center for Attribute A in cluster 0 is 892.503, and in cluster 1 is 381.360. That is a difference of 511.143. |
| **Hint 11:** A and F have bigger differences between cluster 0 and cluster 1 than the other attributes. Choose "A and F". |

**Question 2 of 10:** It's time to plot the data. Set the X axis to be the first answer from Question 1 (A), and set the Y axis to be the second answer from Question 1 (F). Then, set the color column to reflect cluster numbers. There are seven major groupings ("lumps") in this data. How many of them are blue?

|  |
| --- |
| **Hint 1:** In order to plot the data, you will first need to transform your dataframe into a numpy array. |
| **Hint 2:** Do this by first importing numpy and then casting your dataframe as an array, i.e.,  import numpy as np  X = np.array(data) |
| **Hint 3:** Next, you will want to set the colors of the clusters such that each cluster will be plotted using a different color. Let's use blue and red. You do this by first setting up a dictionary of colors and their cluster assignments, e.g.,  color\_map = {0: 'blue', 1: 'red'} |
| **Hint 4:** You will now want to get the labels of each point, i.e., the cluster each point was assigned to by your KMeans model. You do this by pulling the following attribute:  labels = kmeans.labels\_ |
| **Hint 5:** You will then use those labels to create an array of colors that hold the color assignments for each point. You do this by:  colors = [color\_map[l] for l in labels] |
| **Hint 6:** Finally, you can plot the data. You do this by first importing pyplot from the matplotlib module as follows:  import matplotlib.pyplot as plt |
| **Hint 7:** Make sure to select attribute A for the X-axis, and attribute F for the Y-axis. You can do this as follows, where 0 is the column index for attribute A in your array, and 5 is the column index for attribute F:  plt.scatter(X[:, 0], X[:, 5], c=colors, s=10) |
| **Hint 8:** Then, show your plot, i.e.,  plt.show() |
| **Hint 9:** If you followed the directions, the dots should all plot in blue and red. How many of the lumps are blue? |
| **Hint 10:** The definition of a "lump" is a bit fuzzy, but how many dense groups of points are blue? |
| **Hint 11:** Only one of the dense groups of points is blue. Write 1. |

**Question 3 of 10:** What did k-Means do here?

|  |
| --- |
| Sorry, no hint is available at this step. |

**Question 4 of 10:** Now re-run k-Means with k=7. Did k-Means find the 7 lumps in the data that you saw earlier?

|  |
| --- |
| **Hint 1:** To set k=7, simply change n\_clusters to 7, e.g.,  kmeans = KMeans(n\_clusters=7).fit(data) |
| **Hint 2:** Don't forget to extend your color\_map dictionary to include colors for all the other clusters. For the purpose of this test, let's use the following assignments:  color\_map = {0: 'blue', 1: 'red', 2: 'orange', 3: 'yellow', 4: 'green', 5: 'violet', 6: 'black'} |
| **Hint 3:** Re-run and look at the plot. |
| **Hint 4:** Look at the points. Are the colors distributed into the 7 lumps you saw? (i.e., one color per lump). |
| **Hint 5:** Not at all. In fact, even the least central lump off at the top-right is split into two colors. |
| **Hint 6:** So did this process find the 7 lumps in the data that you saw earlier? |
| **Hint 7:** No, it didn't. |

**Question 5 of 10:** Plot each of the other variables against each other (not including the variables in question 1). Does there appear to be meaningful structure in any of these variables?

|  |
| --- |
| **Hint 1:** To plot the variables against each other, change the attributes plotted each time. For example, try "B" and "C", "B" and "D", and so on. |
| **Hint 2: Y**ou can do this by changing the values of first and second in the following stub of code below every time you run your script:  first = 0  second = 1  plt.scatter(X[:, first], X[:, second], c=colors, s=10)  plt.show() |
| **Hint 3:** You don't need to run these models with a different k. You can use the current num\_clusters=7. |
| **Hint 4:** Do lumps correspond to colors for any of the pairs of variables? |
| **Hint 5:** The answer is no. |

**Question 6 of 10:** Filter out all of the variables except the ones in question 1, and re-run KMeans using just these two variables, with k=7. Are all seven of the seven data lumps now more or less incorporated into seven reasonable clusters?

|  |
| --- |
| **Hint 1:** First, remember that you are keeping values of ONLY attributes A and F. In your array, these attributes are referred to by columns 0 and 5, respectively. |
| **Hint 2:** Filter out all other columns. You can do this by creating a new array that has only the contents of columns 0 and 5, by taking a slice out of the original array.  X\_1 = np.delete(X, np.s\_[1:5], axis=1) |
| **Hint 3:** Re-run KMeans with your new array, and plot. Notice that you only have two columns now, column 0 (attribute A) and column 1 (attribute F).  kmeans = KMeans(n\_clusters=7).fit(X\_1)  plt.scatter(X\_1[:, 0], X\_1[:, 1], c=colors, s=10)  plt.show() |
| **Hint 4:** Now look at the plot. Are the seven data lumps now more or less incorporated into seven reasonable clusters? |
| **Hint 5:** Notice that one color is assigned to many outlier points. Meanwhile, one data lump is split between two other colors. |
| **Hint 6:** Six data lumps are connected to six reasonable clusters. But one data lump and one cluster are not properly assigned. |
| **Hint 7:** The answer is no. |

**Question 7 of 12:** What happened?

|  |
| --- |
| Sorry, no hint is available at this step. |

**Question 8 of 10:** For fun, you might want to try playing with different values of k, and the other parameters within k-Means. When you're ready to move on, type the number 0.

|  |
| --- |
| Please enter 0 (zero) to proceed. |

**Question 9 of 10:** When you're done looking, try running Agglomerative Clustering. Look at the Dendrogram. Nifty, huh?

|  |
| --- |
| **Hint 1:** To recall what Agglomerative Clustering is, you can go back to the Advanced Clustering Algorithms video for an explanation beginning at 5:29. |
| **Hint 2:** To use Agglomerative Clustering, you will first need to import it from the sklearn module:  from sklearn.cluster import AgglomerativeClustering |
| **Hint 3:** Next, fit your data into an Agglomerative Clustering model:  aggc = AgglomerativeClustering(n\_clusters=2).fit(X\_1) |
| **Hint 4:** Now that you've fit your model, you will need to plot the dendrogram. Use the following stub of code to produce your dendrogram:<br/>  def plot\_dendrogram(model, \*\*kwargs):  # Children of hierarchical clustering  children = model.children\_    # Distances between each pair of children  # Since we don't have this information, we can use a uniform one for plotting  distance = np.arange(children.shape[0])    # The number of observations contained in each cluster level  no\_of\_observations = np.arange(2, children.shape[0]+2)    # Create linkage matrix and then plot the dendrogram  linkage\_matrix = np.column\_stack([children, distance, no\_of\_observations]).astype(float)    # Plot the corresponding dendrogram  dendrogram(linkage\_matrix, \*\*kwargs) |
| **Hint 5:** Finally, plot your dendrogram using the method defined in the previous hint:  plt.show(plot\_dendrogram(aggc, labels=aggc.labels\_)) |
| **Hint 6:** Isn't that nifty? |
| **Hint 7:** Yes, that \*is\* nifty. |

**Question 10 of 10:** OK, fine. Squint really hard and look at the top-right of the dendrogram. You'll see at the very top fork, that a branch goes down the right side. How many nodes are in this branch? (e.g. how many data points end up in this branch). Note that an immediate branch to a small subset of the data indicates strong outliers.

|  |
| --- |
| **Hint 1:** On your dendrogram screen, use the zoom tool to zoom-in on the upper-right-most part of the graph. |
| **Hint 2:** Use the pan tool to then pan through the dendrogram, dragging the dendrogram up and down to follow the path of the right-most branch down to its nodes. |
| **Hint 3:** The answer is 2. |