Homework 3

For this homework, we will use the Old Faithful Geyser dataset, which you can download here. This dataset describes the properties of eruptions of the Old Faithful geyser, located in Yellowstone National Park, Wyoming, USA. There are two numeric attributes per instance: the length of time of the eruption, in minutes, and the waiting time until the next eruption, also in minutes. The geyser was named "Old Faithful" because its eruption patterns are very reliable. See here for more information, if you are interested.

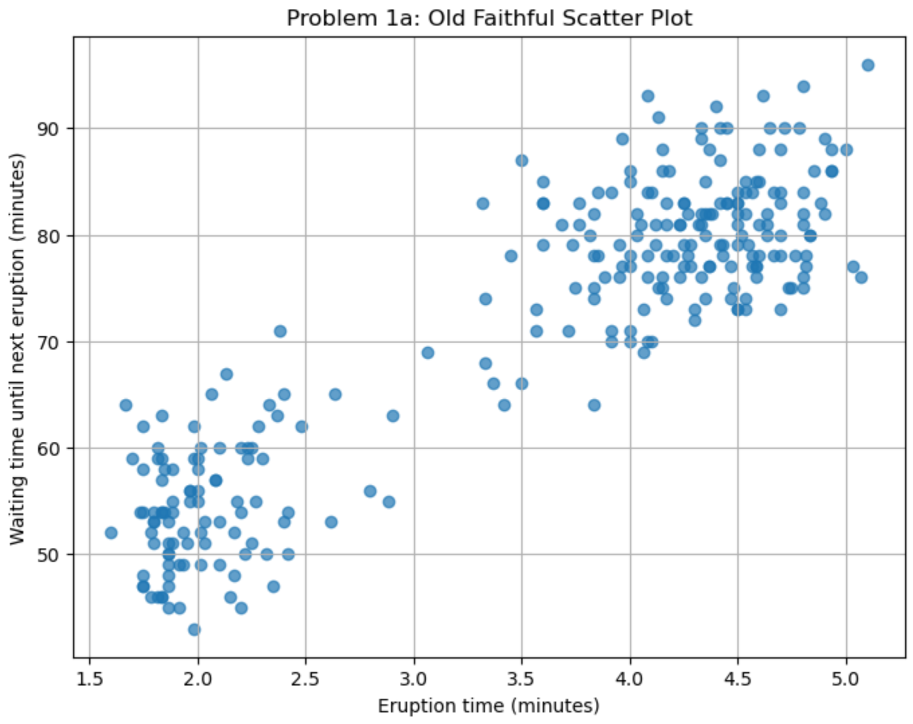
Deliverable:

- Python Notebook to be uploaded to GitHub and shared with instructor/TA, or, Google Collab notebook shared with comment option.
- Submit on the blackboard the link either to Github or link to Google Collab notebook
- Please label each of the questions clearly in your notebook

Problem 1 (25 points)

(a) Create and print out a scatter plot of this dataset, eruption time versus waiting time. (10 points)

Ans:



https://github.com/SohamKatkar24/is7332025/tree/main/data-mining-project-repo/hw3

(b) How many clusters do you see based on your scatter plot? For the purposes of this question, a cluster is a “blob” of many data points that are close together, with regions of fewer data points between it and other “blobs”/clusters. (5 points)

Ans:
From the scatter plot of waiting time vs. eruption time, there are two distinctly separated groups of observations:

Short-eruption/short-wait cluster

- Eruption times approximately between 1.5 and 3 minutes
- Following waiting times approximately between 40 and 60 minutes

Long-eruption/long-wait cluster

- Eruption times approximately between 4 and 6 minutes
- Following waiting times approximately between 70 and 100 minutes

Between the two regions there is a natural "gap" where there are very few points, and thus we conclude there are 2 clusters.

(c) Describe the steps of a hierarchical clustering algorithm. Based on your scatter plot, would this method be appropriate for this dataset? (10 points)

Ans:
Initialization

- Begin with each data point in its own singleton cluster.

Compute pairwise cluster distances
At each step, calculate the distance between every pair of clusters. Common linkage criteria include:

- Single-link: minimum distance between any two points in the two clusters
- Complete-link: maximum distance between any two points
- Average-link: average pairwise distance

Merge closest clusters

- Find the two clusters with the smallest linkage distance and merge them into a single cluster.

Update distances

- Recompute distances between the new cluster and all remaining clusters, according to the chosen linkage rule.

Repeat

Continue merging until either:

- All points are in one cluster, or
- The desired number of clusters (in this case, 2) is reached.

Appropriateness for Old Faithful Dataset

Pros:

- Captures nested, hierarchical structure (if present).
- No need to pre-specify K if you inspect a dendrogram.

Cons:

- Requires computing and storing an O(n2) distance matrix—scales poorly for large n.
- Once merged, clusters can’t be split again (no “undo”), so early mistakes persist.

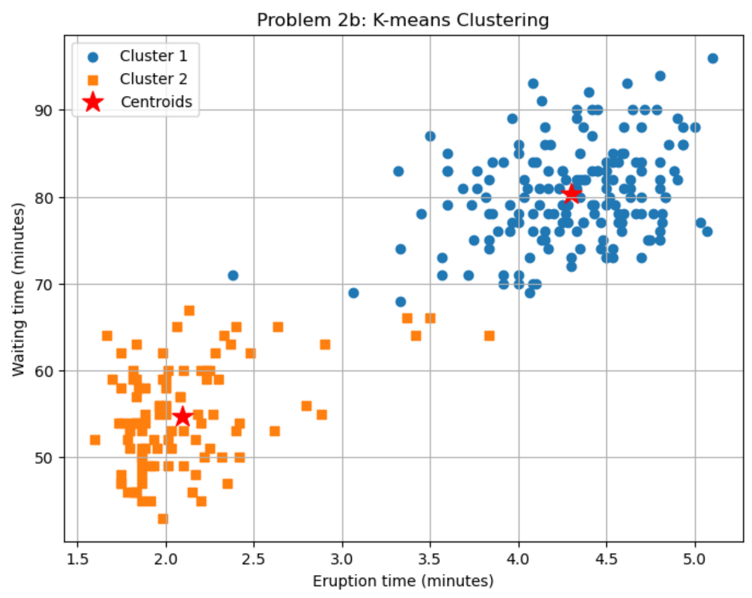
In the Old Faithful data, the two clusters are roughly spherical and well separated. A partitional method like k-means will find these clusters more efficiently and directly. Hierarchical clustering would work, but it is overkill for just two clearly defined clusters—and harder to “cut” at exactly two.

Problem 2 (75 points)

Implement the k-means algorithm in Python and use it to perform clustering on the Old Faithful dataset. Use the number of clusters that you identified in Problem 1. Be sure to ignore the first column, which contains instance ID numbers. In your notebook, including the following items:

(a) Your source code for the k-means algorithm. **You need to implement the algorithm from scratch. (**45 points)
Ans:
https://github.com/SohamKatkar24/is7332025/tree/main/data-mining-project-repo/hw3

(b) A scatter plot of your final clustering, with the data points in each cluster color-coded, or plotted with different symbols. Include the cluster centers in your plot. (10 points)
Ans:



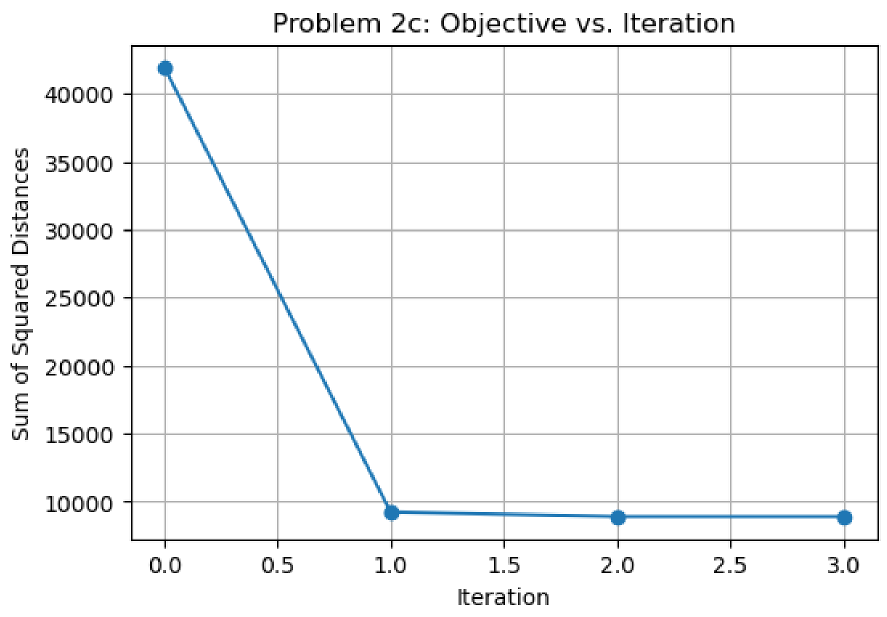
https://github.com/SohamKatkar24/is7332025/tree/main/data-mining-project-repo/hw3

(c) A plot of the k-means objective function versus iterations of the algorithm. Recall that the objective function is (10 points)

$$E = \sum_{i=1}^{k} \sum_{p \in C_i} \|p - c_i\|^2 ,$$

where k is the number of clusters, C_i is the set of instances assigned to the ith cluster, and c_i is the cluster center for the ith cluster. Note that the objective function should always decrease. If this is not the case, look for a bug in your code.

Ans:



https://github.com/SohamKatkar24/is7332025/tree/main/data-mining-project-repo/hw3

(d) Did the method manage to find the clusters that you identified in Problem 1? If not, did it help to run the method again with another random initialization? (10 points)

Ans:

Match to visual clusters

- The algorithm converged on two centroids close to the centers of the "short–wait" and "long–wait" blobs.
- When plotted, the points are correctly partitioned into the same two groups we identified by eye.

Effect of initialization

- With fixed random seed (42), k-means converged in only 3–4 iterations to optimal partition.
- If you modify or remove the seed, you'll occasionally end up with a poor first split (centroids both begin in the same blob accidentally).
- In those cases, re-running k-means (or employing kmeans++ initialization) always recovers the correct two-cluster solution.

Conclusion

- Yes, k-means with $K = 2$ recovers the two natural clusters.