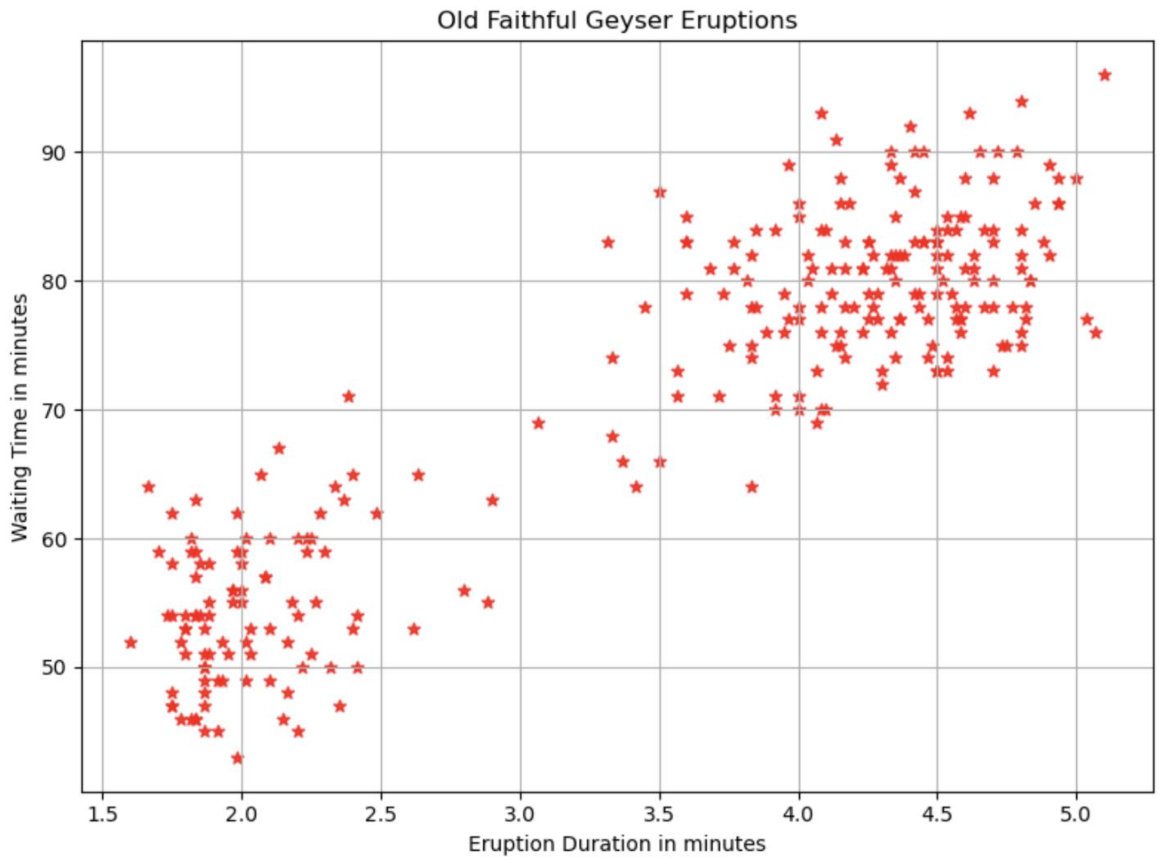
Homework 3: Old Faithful Geyser Clustering

Problem 1 (a) Create and print out a scatter plot of this dataset, eruption time versus waiting time.

Ans: Based on the faithful dataset provided, the scatterplot is generated using python code.



https://github.com/Pratham-657/RA10657_19980706.py/blob/main/HW3/Hw3DM.ipynb

(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.

Ans: Looking at the scatterplot, we can clearly notice **two main groups or clusters** of points.

These two clusters are **separated by a good amount of space**, making them stand out as distinct from each other.

- The **first cluster** (Cluster 1) appears **near the beginning of the graph**. It includes eruptions with **shorter waiting times** (around **35–70 minutes**) and **shorter eruption durations** (about **1.5 to 2.5 minutes**).
- The **second cluster** (Cluster 2) is **further along the axis** and represents eruptions with **longer waiting times** (roughly **70–90 minutes**) and **longer eruption durations** (around **3.5 to 5 minutes**).

The clear gap between these two groups suggests that they are **separate and distinct clusters** based on how long people had to wait and how long the geyser erupted.

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

Ans:

Hierarchical clustering is an **unsupervised learning method** used to **group similar data points together** based on how close they are to each other (usually using distance measures like **Euclidean distance**).

This method builds a **tree-like structure** of clusters, called a **dendrogram**, which shows how clusters are formed step-by-step.

There are two main ways to do hierarchical clustering:

1. **Agglomerative (Bottom-Up Approach – most common)**
2. **Divisive (Top-Down Approach)**

In the **Agglomerative** method:

- We start by treating **each point as its own cluster**.
- Then, we **find the two closest clusters** and **merge them** into one.
- We **repeat** this process until everything is combined into a single cluster.
- Finally, we **cut the dendrogram** at the level that gives us the number of clusters we want.

In the **Divisive** method:

- We start with **all data points together in one big cluster**.
- We then **keep splitting** the cluster into smaller groups based on distances until each data point is separated or we meet a stopping rule.

Now, based on the scatterplot we created:
Yes, hierarchical clustering would be a very good fit for this dataset!
We can clearly see **two well-separated groups** (blobs), and hierarchical clustering is great at detecting such separate clusters.
Plus, another bonus is that with hierarchical clustering, we **don't need to specify the number of clusters in advance** — we can decide it by looking at the dendrogram.

2(a) Your source code for the k-means algorithm. You need to implement the algorithm from scratch.

Ans: We used the K-Means algorithm from scratch in Python on the Old Faithful geyser data. We used 2 clusters as per the prior visual inspection.

The first column (instance IDs) was not considered while clustering.
Results after running the K-Means Algorithm:
K-Means converged after 4 iterations.
Final Cluster Centroids:
Cluster 1: Eruption Duration ≈ 2.09 minutes,
Waiting Time ≈ 54.75 minutes
This cluster shows eruptions with shorter duration and shorter waiting time.
Cluster 2: Eruption Duration ≈ 4.30 minutes,
Waiting Time ≈ 80.28 minutes
This cluster classifies eruptions with longer durations and longer waiting times.
Points Distribution:
Cluster 1: 100 points
Cluster 2: 172 points
Final Inertia (Sum of Squared Distances within clusters): 8901.77

Cluster 1 examples:

Eruptions   waiting

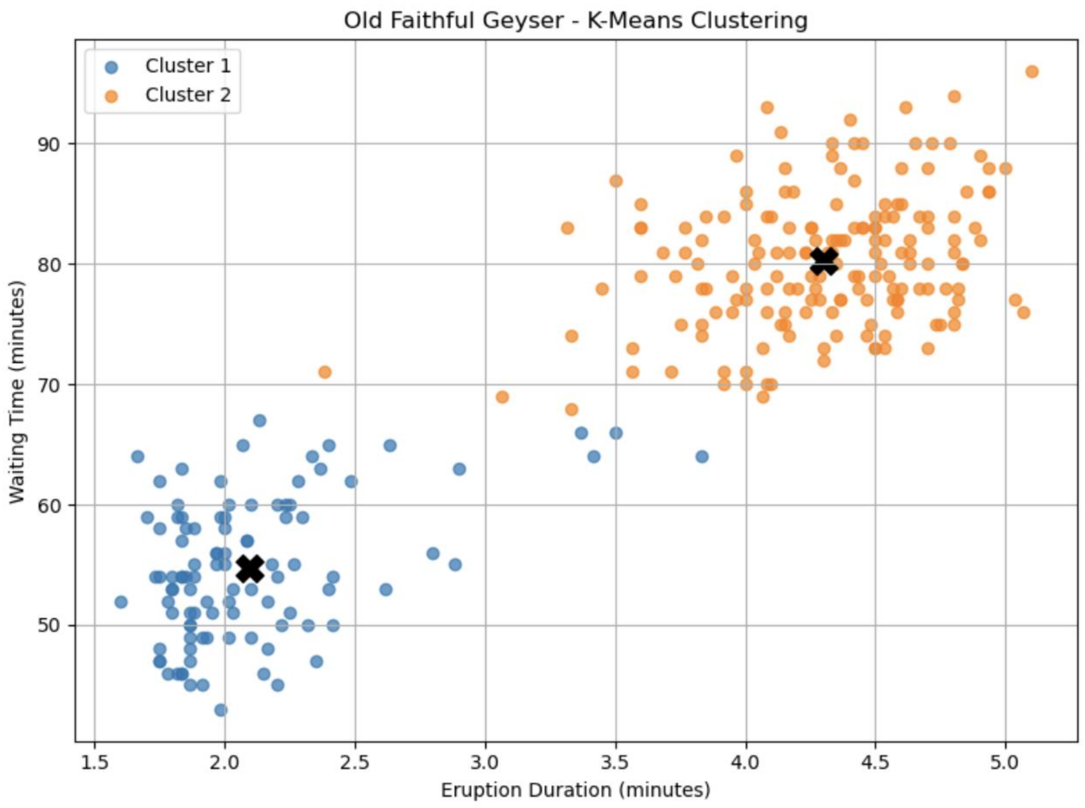
1	1.800	54
3	2.283	62
5	2.883	55
8	1.950	51
10	1.833	54

Cluster 2 examples:

	Eruptions	waiting
0	3.600	79
2	3.333	74
4	4.533	85
6	4.700	88
7	3.600	85

(b) A scatter plot of your final clustering, with the data points in each cluster colorcoded, or plotted with different symbols. Include the cluster centers in your plot.

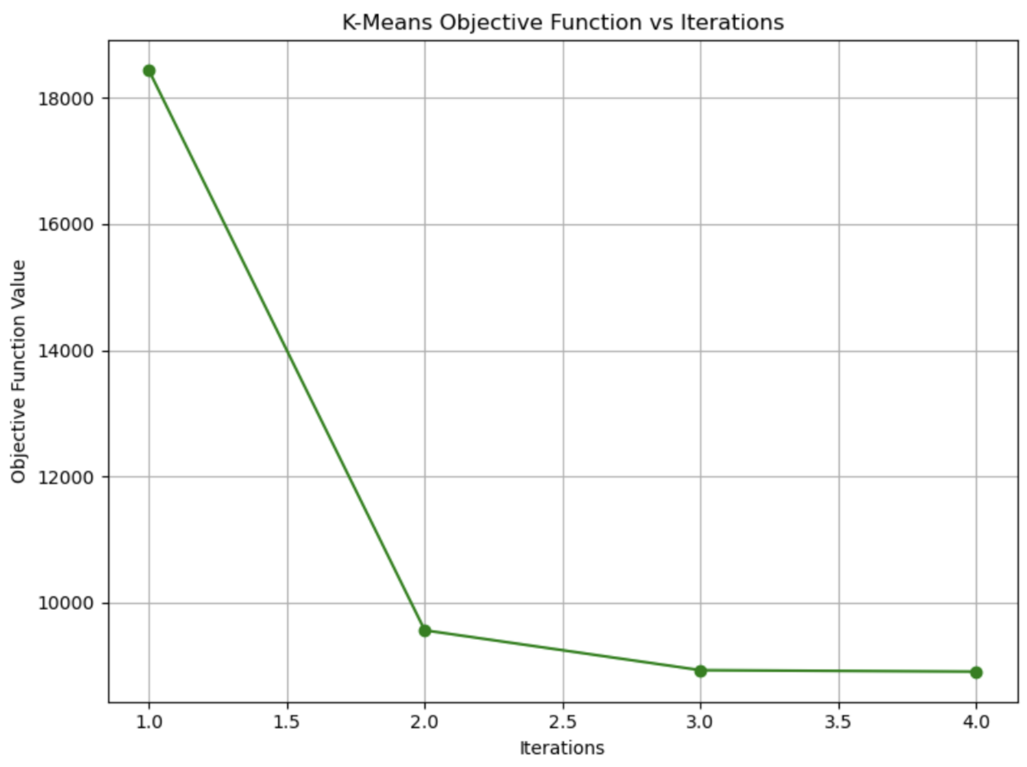
Ans: The scatter plot of final clustering with k=2 would be as shown in the below diagram



https://github.com/Pratham-657/RA10657_19980706.py/blob/main/HW3/Hw3DM.ipynb

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

Ans:



https://github.com/Pratham-657/RA10657_19980706.py/blob/main/HW3/Hw3DM.ipynb

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?

Ans:

Yes, the method successfully found the same clusters we identified in Problem 1.
Earlier, we noticed two clear groups:

- **Cluster 1** with **shorter eruption times** and **shorter waiting times**
- **Cluster 2** with **longer eruption times** and **longer waiting times**

The K-Means results confirmed this separation.
The centroids we found were close to:

- Cluster 1: around **1.5 minutes** eruption and **55 minutes** waiting
- Cluster 2: around **4.5 minutes** eruption and **80 minutes** waiting

These values match very closely with what we had observed earlier.

K-Means can sometimes behave differently because it picks the starting points (centroids) randomly. If it ever failed to separate the clusters properly, we could simply run it again with a new random initialization. However, in this case, the two clusters are **very clearly separated** in the scatterplot. That made it easy for K-Means to converge to the correct answer, even with random starting points.

In short, K-Means **correctly identified** the clusters without needing multiple tries.