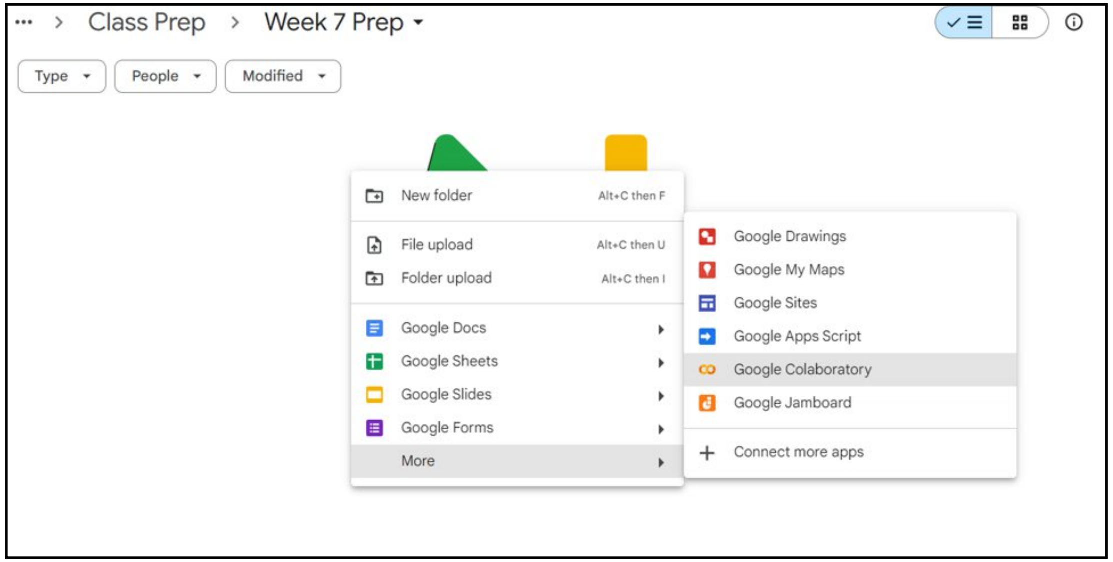
K Means Clustering

Launching Colab

- Log in with your university email account
- Create a folder and enter into the folder
- Select the option – "Google Colaboratory" as shown in the image

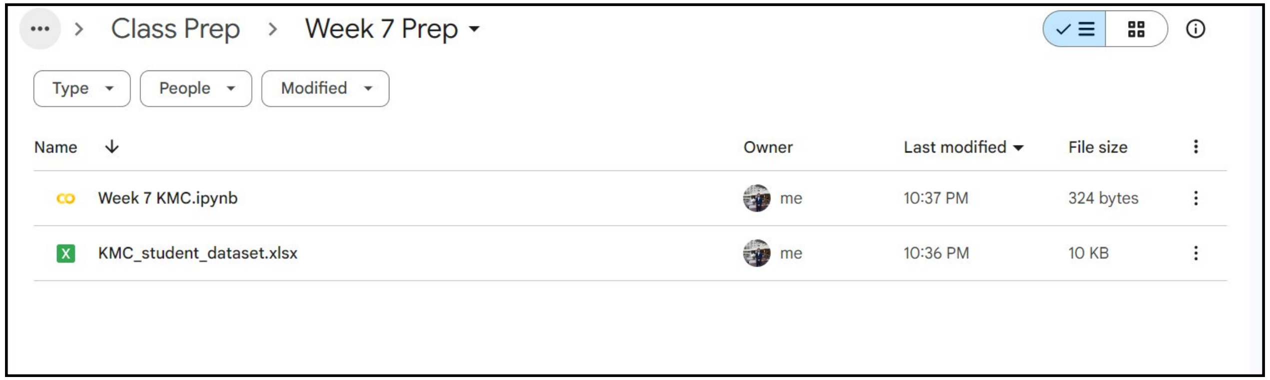


Launching Colab

- Once the colab file opens, give it a suitable filename

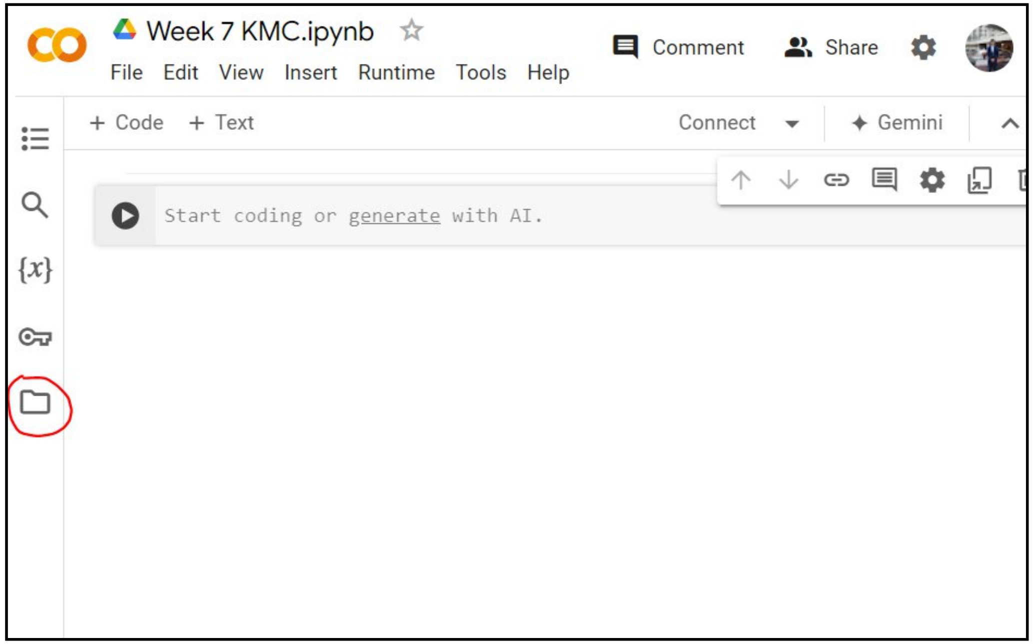
The Dataset

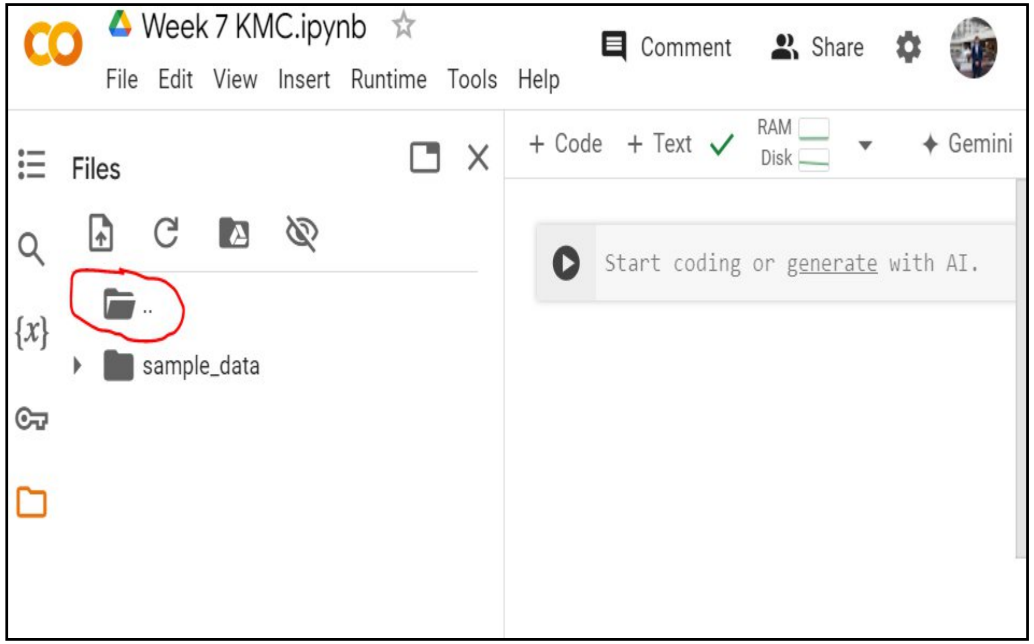
- An excel file has been shared with you – see elms
- Download the excel file first
- Now go to the folder you created
- Upload the excel file there – right click – file upload
- You should have something like this now



Accessing the dataset from colab

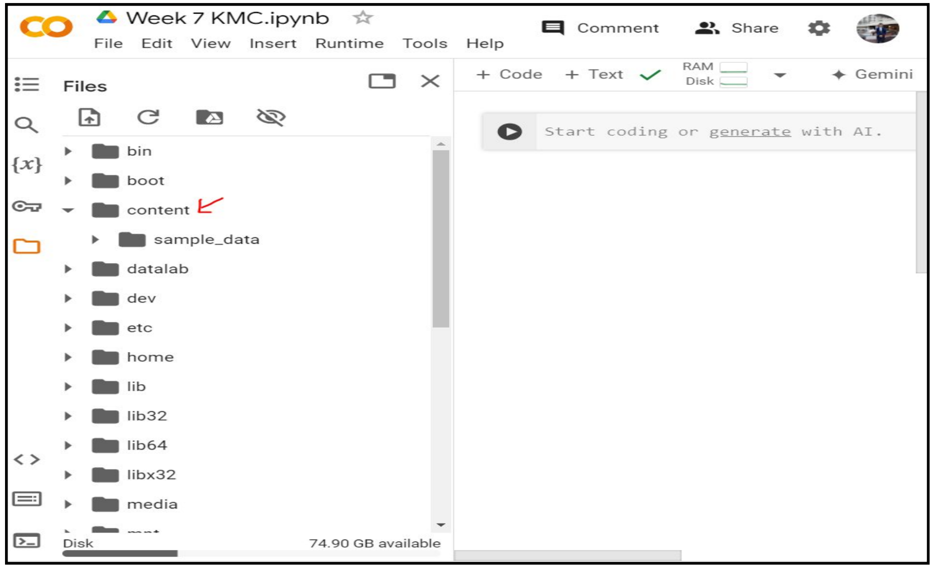
- Click on the folder icon
- Then click on the folder with three dots next to it

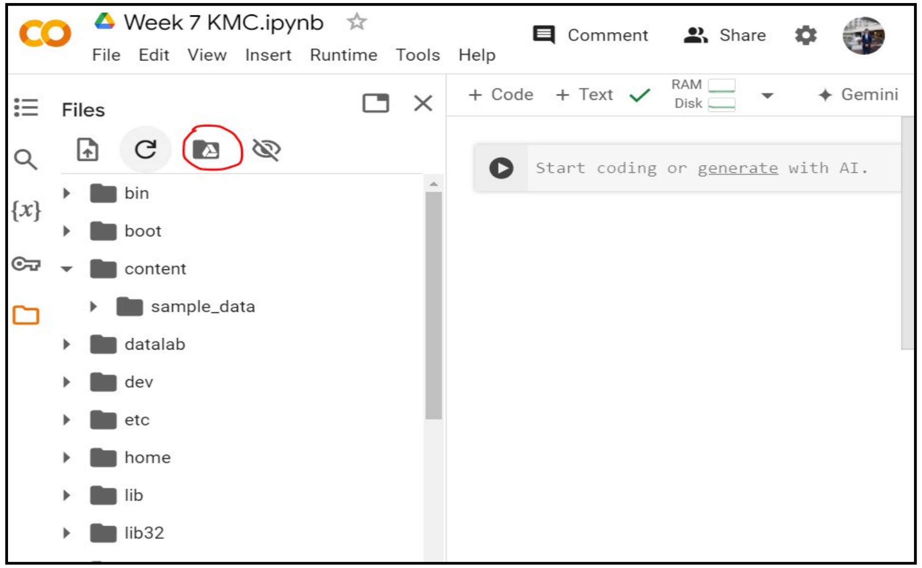




Accessing the dataset from colab

- Click on content
- Then click on the mount google drive icon
- Then give all the permissions (even if they ask for your properties!)



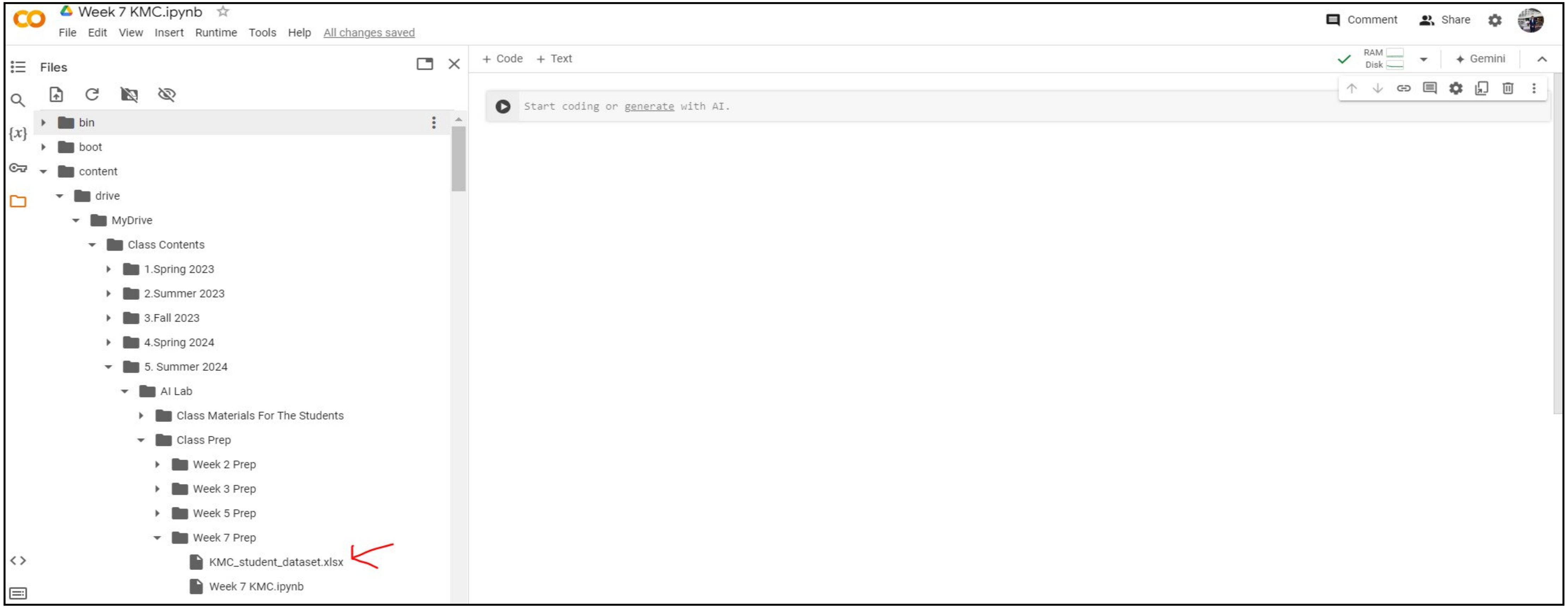


Accessing the dataset from colab

- You will that a folder named “drive” will appear.
- If it does not appear, click on the refresh icon.

Accessing the dataset from colab

- Now find and click on the dataset that you had uploaded moments ago.
- See the following image for reference



Accessing the dataset via python

- You have already learned about the library – pandas
- You need to use the library here as well
- Basically you will read the excel file and print its contents.
- Write down the following code snippet (use your own file path – do not copy the one in image!)

```python
import pandas as pd

# Define the path to the Excel file
file_path = '/content/drive/MyDrive/Class Contents/5. Summer 2024/AI Lab/Class Prep/Week 7 Prep/KMC_student_dataset.xlsx'

# Read the Excel file into a Pandas DataFrame
df = pd.read_excel(file_path)

# Print the DataFrame
print(df)
```

```
   Study Hours  Sleep Hours  Attendance Percentage  Exercise Hours   SSC GPA  \
0            7            7                     63               4  4.134026
1            4            7                     66               4  4.370527
2            8            4                     85               1  3.817880
3            5            6                     99               6  4.778903
4            7            8                     89               4  3.953231
5            3            6                     53               1  4.744879
6            7            8                     51               0  4.550116
7            8            4                     55               3  3.348352
8            5            5                     91               3  2.286230
9            4            7                     53               3  3.112455
```

Converting the data frame into NumPy

- Remember the numpy library?
  - It makes the mathematical operations easier for us
- Now, convert the dataframe into a numpy array
- This is actually a matrix

```
# Convert the DataFrame to a NumPy array
data_array = df.to_numpy()
data_array=data_array.astype(float)
# Print the NumPy array
print(data_array)
```

Normalization

- Observe the columns.
- You will find that some of the values are larger than the others
- This will impact the decision making of the ML Algorithm (KMC)
- So we need to bring them all down to the same scale (0-1)
- Solution - Normalization

$$X_{\text{normalized}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

Normalization

- Here is the code for normalization
- Take out your notebooks, I will explain how I wrote this code

```python
import numpy as np
# Create an empty array to store the normalized data
normalized_data = np.zeros_like(data_array)

# determine the number of columns
noOfColumns = data_array.shape[1]

# Normalize one column at a time using aloop
for i in range(noOfColumns):
    min_val = np.min(data_array[:, i])
    max_val = np.max(data_array[:, i])
    normalized_data[:, i] = (data_array[:, i] - min_val) / (max_val - min_val)


# print normalized data
print(normalized_data)
```

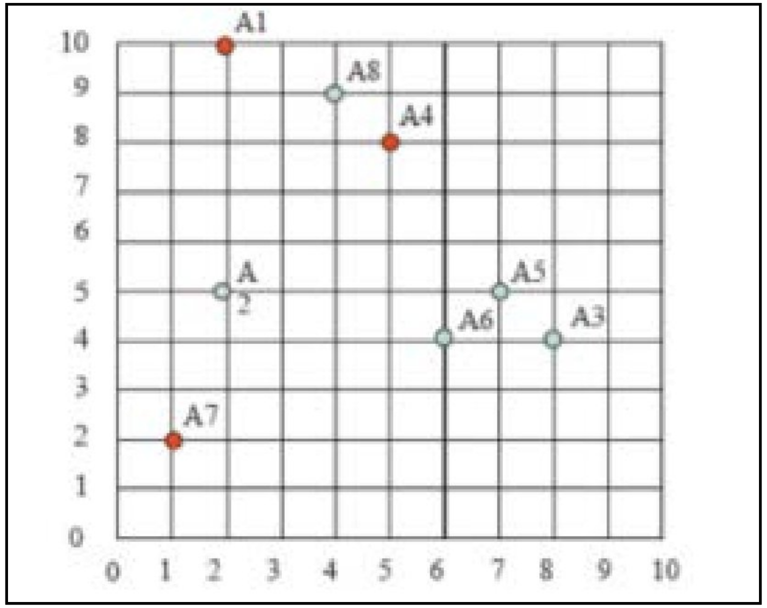
```
[[0.75       0.6        0.25       0.66666667 0.72041646 0.11491476]
 [0.375      0.6        0.3125     0.66666667 0.80506738 0.94376372]
 [0.875      0.         0.70833333 0.16666667 0.60725807 0.44171714]
 [0.5        0.4        1.         1.         0.95123777 0.54669639]
 [0.75       0.8        0.79166667 0.66666667 0.65570442 0.70462204]
```

Now, Some Theory

- Let us understand using an example:
- Cluster the following eight points (with (x_1,x_2) representing locations): into three clusters A1(2,10), A2(2,5) , A3(8,4), A4(5,8), A5(7.5), A6(6,4), A7(1,2) and A8(4,9)
- Assume that k=3
- So there will be three clusters
- Let us at first plot these points in a graph

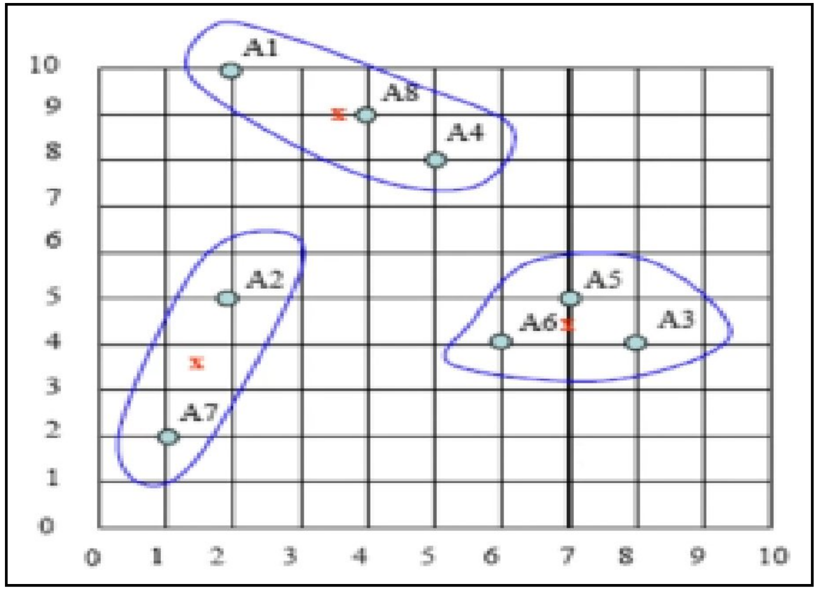
After Plotting

- After plotting it should look like this
- Can you group (i.e) cluster the points into three?



After Plotting

- After plotting it should look like this
- Can you group (i.e) cluster the points into three?
- Now let’s see what the KMC algorithm does for us!



KMC

Step 1

- At first KMC selects k centroids.
- Let k = 3.
- So there will be three centroids.
- We can select any three random points as centroids.
- Let us select A1(2,10), A4(5,8) and A7(1,2) as centroids (RANDOMLY)

KMC

- **Step 2**
- Next let us calculate how far all the other points are from these three centroids.
- In your notebook, you should sketch a table like this.

		(2,10)	(5,8)	(1,2)	
	Point	Dist Mean 1	Dist Mean 2	Dist Mean 3	Cluster
A1	(2,10)				
A2	(2,5)				
A3	(8,4)				
A4	(5,8)				
A5	(7,5)				
A6	(6,4)				
A7	(1,2)				
A8	(4,9)				

KMC

- **Step 3**
- Calculate the Manhattan distance of all the points from the centroids.
- And based on the smallest distance, assign each point to a cluster.

	(2,10)	(5,8)	(1,2)	
Point	Dist Mean 1	Dist Mean 2	Dist Mean 3	Cluster
(2,10)	0	5	9	1
(2,5)	5	6	4	3
(8,4)	12	7	9	2
(5,8)	5	0	10	2
(7,5)	10	5	9	2
(6,4)	10	5	7	2
(1,2)	9	10	0	3
(4,9)	3	2	10	2

KMC

- **Step 4**
- Now calculate new centroids

Cluster 1	Cluster 2	Cluster 3
(2, 10)	(8, 4)	(2, 5)
	(5, 8)	(1, 2)
	(7, 5)	
	(6, 4)	
	(4, 9)	

	(2,10)	(5,8)	(1,2)	
Point	Dist Mean 1	Dist Mean 2	Dist Mean 3	Cluster
(2,10)	0	5	9	1
(2,5)	5	6	4	3
(8,4)	12	7	9	2
(5,8)	5	0	10	2
(7,5)	10	5	9	2
(6,4)	10	5	7	2
(1,2)	9	10	0	3
(4,9)	3	2	10	2

- For Cluster 1, we only have one point A1(2, 10), which was the old mean, so the cluster center remains the same.
- For Cluster 2, we have ((8+5+7+6+4)/5, (4+8+5+4+9)/5) = (6, 6)
- For Cluster 3, we have ((2+1)/2, (5+2)/2) = (1.5, 3.5)

KMC

- Now keep on repeating the steps 2 – 4.
- Stop when you find that the clusters remain the same after two consecutive iterations.
- Iteration 2

		(2, 10)	(6, 6)	(1.5, 3.5)	
	Point	Dist Mean 1	Dist Mean 2	Dist Mean 3	**Cluster**
A1	(2, 10)	0	8	7	1
A2	(2, 5)	5	5	2	3
A3	(8, 4)	12	4	7	2
A4	(5, 8)	5	3	8	2
A5	(7, 5)	10	2	7	2
A6	(6, 4)	10	2	5	2
A7	(1, 2)	9	9	2	3
A8	(4, 9)	3	5	8	1

KMC

- Now keep on repeating the steps 2 – 4.
- Stop when you find that the clusters remain the same after two consecutive iterations.
- Iteration 3

		(3, 9.5)	(6.5 ,5.25)	(1.5, 3.5)	
	Point	Dist Mean 1	Dist Mean 2	Dist Mean 3	**Cluster**
A1	(2, 10)	1.5	9.25	7	1
A2	(2, 5)	5.5	4.75	2	3
A3	(8, 4)	10.5	2.75	7	2
A4	(5, 8)	3.5	4.25	8	1
A5	(7, 5)	8.5	0.75	7	2
A6	(6, 4)	8.5	1.75	5	2
A7	(1, 2)	9.5	8.75	2	3
A8	(4, 9)	1.5	6.25	8	1

SHOULD WE STOP NOW?

KMC

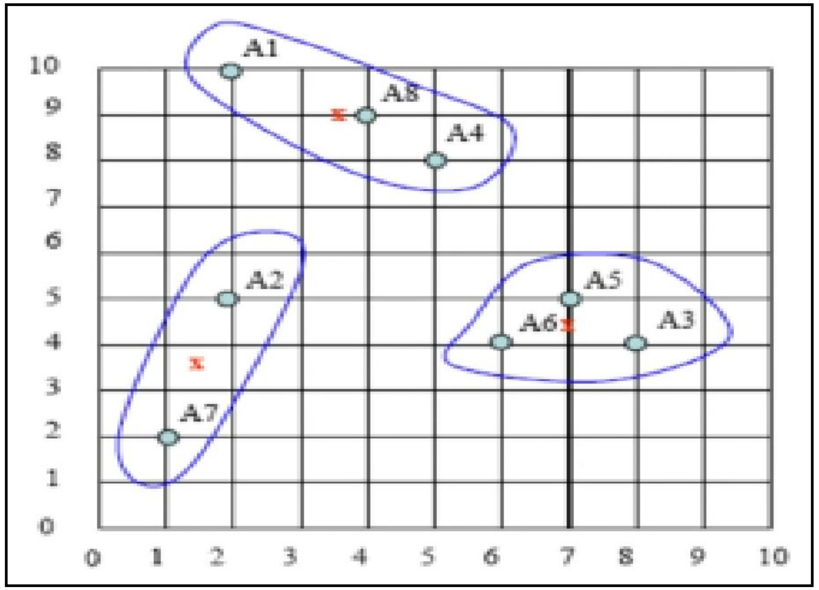
- Now keep on repeating the steps 2 – 4.
- Stop when you find that the clusters remain the same after two consecutive iterations.
- Iteration 4

		(3.67, 9)	(7 ,4.3)	(1.5, 3.5)	
	Point	Dist Mean 1	Dist Mean 2	Dist Mean 3	**Cluster**
A1	(2, 10)	2.67	10.7	7	1
A2	(2, 5)	5.67	5.7	2	3
A3	(8, 4)	9.33	1.3	7	2
A4	(5, 8)	2.33	5.7	8	1
A5	(7, 5)	7.33	0.7	7	2
A6	(6, 4)	7.33	1.3	5	2
A7	(1, 2)	9.67	8.3	2	3
A8	(4, 9)	0.33	7.7	8	1

SHOULD WE STOP NOW?

KMC

- Match the table with the diagram



		(3.67, 9)	(7 ,4.3)	(1.5, 3.5)	
	Point	Dist Mean 1	Dist Mean 2	Dist Mean 3	**Cluster**
A1	(2, 10)	2.67	10.7	7	1
A2	(2, 5)	5.67	5.7	2	3
A3	(8, 4)	9.33	1.3	7	2
A4	(5, 8)	2.33	5.7	8	1
A5	(7, 5)	7.33	0.7	7	2
A6	(6, 4)	7.33	1.3	5	2
A7	(1, 2)	9.67	8.3	2	3
A8	(4, 9)	0.33	7.7	8	1

Now let's get back to coding

- Step 1 :
  - Select three points as centroids (k=3)
  - np.random.choice –
    - First parameter – we give it the number of data points
    - Second parameter – the number of random samples we want
    - Third Parameter – setting it false means that the same point will not be selected again

```python
# Randomly determine 3 indices
random_indices = np.random.choice(normalized_data.shape[0], 3, replace=False)
print(random_indices)
# Get those the points at those 3 indices
centroids = data_array[random_indices]
# Print the selected points
print("Selected random points (initial centroids):")
print(centroids)
```

```
[10  0 16]
Selected random points (initial centroids):
[[ 8.          4.          78.          4.          4.00652376  3.7965964 ]
 [ 7.          7.          63.          4.          4.13402586  2.60515761]
 [ 8.          5.          85.          0.          4.91513629  3.36960371]]
```

Now let’s get back to coding

- Step 2 :
  - Now calculate the distance of all the points with these centroids
  - **Take out your notebooks**
  - Also write this code

```
[14] # This is the table that we saw earlier
     distances = np.zeros((data_array.shape[0], centroids.shape[0]))

     # This is the
     for i in range(centroids.shape[0]):
         distances[:, i] = np.sqrt(np.sum((data_array - centroids[i]) ** 2, axis=1))

     # Print the distances
     print("Centroids:")
     print(centroids)
     print("\nDistances from each centroid to all points:")
     print(distances)
```

Now let’s get back to coding

- Step 3 :
  - Now calculate the distance of all the points with these centroids
  - **Take out your notebooks**
  - Also write this code

```
[14] # This is the table that we saw earlier
     distances = np.zeros((data_array.shape[0], centroids.shape[0]))

     # This is the
     for i in range(centroids.shape[0]):
         distances[:, i] = np.sqrt(np.sum((data_array - centroids[i]) ** 2, axis=1))

     # Print the distances
     print("Centroids:")
     print(centroids)
     print("\nDistances from each centroid to all points:")
     print(distances)
```

```
Centroids:
[[ 2  5]
 [ 5  8]
 [ 2 10]]

Distances from each centroid to all points:
[[5.         3.60555128 0.        ]
 [0.         4.24264069 5.        ]
 [6.08276253 5.         8.48528137]
 [4.24264069 0.         3.60555128]
 [5.         3.60555128 7.07106781]
 [4.12310563 4.12310563 7.21110255]
 [3.16227766 7.21110255 8.06225775]
 [4.47213595 1.41421356 2.23606798]]
```

Now let’s get back to coding

- Step 3 (continued) :
  - Now assign the data points to clusters
  - What does axis = 1 mean here (recall from previous classes)

```
# Determine the closest centroid for each point
closest_centroids = np.argmin(distances, axis=1)
print("\nClosest centroid for each point (0-indexed):")
print(closest_centroids)
```

Now let’s get back to coding

- Step 4 :
  - Now we are going to calculate new centroids
  - Take out your notebooks again!

```
# Initialize the new centroids array
new_centroids = np.zeros_like(centroids)

for i in range(len(centroids)):
    # Get points assigned to the current centroid
    # get those rows of the data array where the closest centroid is equal to the current centroid i
    points = data_array[i==closest_centroids]

    # Update the centroid to the mean of these points, if there are any
    if points.size > 0:
        new_centroids[i] = points.mean(axis=0)
```

Now let’s get back to coding

- Step5
  - putting it all in a loop.
  - This is a part of your assignment -2.
  - You are going to execute the previous code in a loop
  - How can you determine the loop ending condition?
  - Hint: (you can compare two variables, what are those?)

Thank You