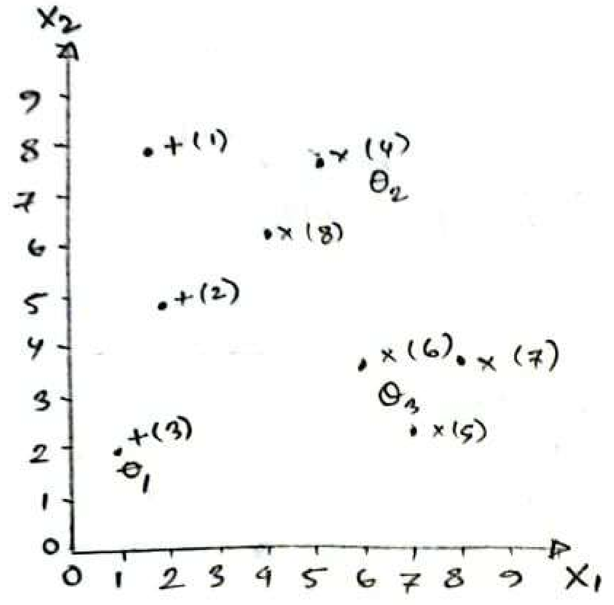
Forgy initialization:



initial cluster centers:

$\theta_1 = x^{(3)}$

$\theta_2 = x^{(4)}$

$\theta_3 = x^{(6)}$

Notation: $x^{(i)} = (x_1^{(i)}, x_2^{(i)})$

$\theta_j = (\theta_{j1}, \theta_{j2})$

1. Iteration

$\mu_1 = \mu_2 = \mu_3 = \{\}$

The data points used for the initialization are easy to assign

$\mu_1 = \{x^{(3)}\},\ \mu_2 = \{x^{(4)}\},\ \mu_3 = \{x^{(6)}\}$

Let's compute the distance of each remaining data points to all cluster centre using the squared euclidean distance:

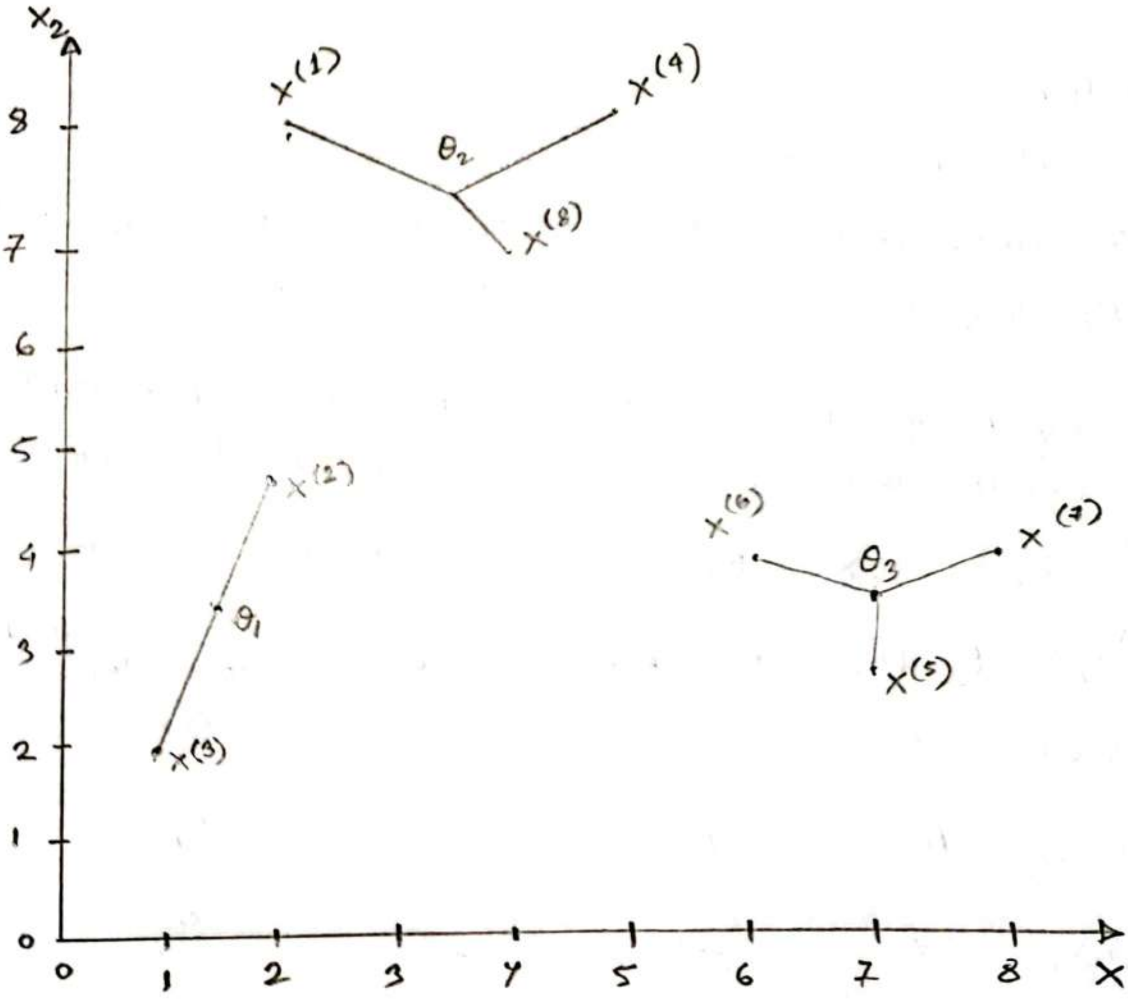
$$d(x^{(i)}, \theta_i) := \| x^{(i)} - \theta_i \|_2^2 = (x_1^{(i)} - \theta_{i1})^2 + (x_2^{(i)} - \theta_{i2})^2$$

$d(x^{(i)}, \theta_k)$	$\theta_1 = (1,2)$	$\theta_2 = (5,8)$	$\theta_3 = (6,4)$	$x^{(i)}$ is assigned to clusters
$x^{(1)} = (2,8)$	37	9	32	θ_2
$x^{(2)} = (2,5)$	10	18	17	θ_1
$x^{(5)} = (7,3)$	37	29	2	θ_3
$x^{(7)} = (8,4)$	53	25	4	θ_3
$x^{(8)} = (4,7)$	34	2	13	θ_2

$$\Rightarrow \mu_1 = \{x^{(3)}, x^{(2)}\}$$
$$\mu_2 = \{x^{(4)}, x^{(1)}, x^{(8)}\}$$
$$\mu_3 = \{x^{(6)}, x^{(5)}, x^{(7)}\}$$

Compute new cluster centers: $\theta_K = \frac{1}{|\mu_K|} \sum_{x^{(i)} \in \mu_K} x^{(i)}$

$$\theta_1 = \tfrac{1}{2}\left(x^{(3)} + x^{(2)}\right) = (1.5, 3.5)$$
$$\theta_2 = \tfrac{1}{3}\left(x^{(4)} + x^{(1)} + x^{(8)}\right) = \left(\tfrac{11}{3}, \tfrac{23}{3}\right)$$
$$\theta_3 = \tfrac{1}{3}\left(x^{(6)} + x^{(5)} + x^{(7)}\right) = \left(7, \tfrac{11}{3}\right)$$



Loss: $L = \sum_{k=1}^{3} \sum_{x^{(i)} \in \mu_k} d(x^{(i)}, \theta_k) = \sum_{k=1}^{3} \sum_{x^{(i)} \in \mu_k} \left((x_1^{(i)} - \theta_{k1})^2 + (x_2^{(i)} - \theta_{k2})^2 \right)$

$d(x^{(2)}, \theta_1) = 2.5$

$d(x^{(3)}, \theta_1) = 2.5$

$d(x^{(1)}, \theta_2) \approx 2.89$

$d(x^{(4)}, \theta_2) \approx 1.89$

$d(x^{(8)}, \theta_2) \approx 0.56$

$d(x^{(5)}, \theta_3) \approx 0.44$

$d(x^{(6)}, \theta_3) \approx 1.11$

$d(x^{(7)}, \theta_3) \approx 1.11$

$L = 13.0$

2. Iteration

$\mu_1 = \mu_2 = \mu_3 = \{\}$

$d(x^{(i)}, \theta_K)$	$\theta_1 = (1.5, 3.5)$	$\theta_2 = (\frac{11}{3}, \frac{23}{3})$	$\theta_3 = (7, \frac{11}{3})$	$x^{(i)}$ is assigned to cluster
$x^{(1)} = (2,8)$	20.50	2.89	43.78	θ_2
$x^{(2)} = (2,5)$	2.50	9.89	26.78	θ_1
$x^{(3)} = (1,2)$	2.50	39.22	38.78	θ_1
$x^{(4)} = (5,8)$	32.50	1.89	22.78	θ_2
$x^{(5)} = (7,3)$	30.50	32.89	0.44	θ_3
$x^{(6)} = (6,4)$	20.50	18.89	1.11	θ_3
$x^{(7)} = (8,4)$	42.50	32.22	1.11	θ_3
$x^{(8)} = (4,7)$	18.50	0.56	20.11	θ_2

$\Rightarrow$ each datapoint is already assigned to its closed cluster center

$\Rightarrow \mu_1 = \{x^{(3)}, x^{(2)}\}$

$\mu_2 = \{x^{(4)}, x^{(1)}, x^{(8)}\}$

$\mu_3 = \{x^{(6)}, x^{(5)}, x^{(7)}\}$

remains unchanged

$\Rightarrow L = 13.0$ unchanged

Statistical data analysis

Problem Sheet 9

Exercise 2

```
# necessary libraries
from PIL import Image
import numpy as np
import os
import matplotlib.pyplot as plt
```

Loading the Image and converting into ndarray

```
file = os.path.abspath("/content/Rubix_cube_ps9.jpg")
path = os.path.dirname(file)
```

```
arr = np.array(Image.open(file))
print("The image and pixel has shape: " + str(arr.shape))

```

```
The image and pixel has shape: (410, 400, 3)
```

So, we can see the resolution of the image is 410x400 where each pixel has RGB values. Which means each pixel is a combination of three values in range of 0-255 .

Implementing the k-means algorithm.

```
def getCenter(X, R):
    # initializing shapes
    le, wi, n = X.shape
    k = R.shape[-1]
    # initializing Means
    Miu = np.zeros((k, n))

    # converting R shape into 2d
    R = R.reshape((le*wi, k))
    # converting X as 2d
    X = X.reshape((le*wi, n))

    # iterate over each cluster
    for i in range(k):
        # get the pixels that belong to cluster i and calculate their mean value
        Miu[i] = X[np.where(R[:, i] == 1)].mean(axis=0)

    return Miu
```

```
def getCluster(X, Miu):
    # initializing shapes
    le, wi, n = X.shape
    k = Miu.shape[0]
    # initializing R in 2D shape
    R = np.zeros((le*wi, k))

    # convert the image ndarray into 2d shape
    X = arr.reshape((le*wi,n))

    # calculating the interval
```

```
    interv = np.linalg.norm(np.repeat(X, k, axis=0) - np.tile(Miu, (le * wi, 1)), axis=1, keepdims=True)

    # converting interval array shape
    interv = interv.reshape(le*wi, k)

    # finding ic as the index of the closest mean
    ic = np.argmin(interv, axis=1)

    # plugging 1 to the closest mean
    R[np.arange(le*wi),ic] = 1
    # converting R into 3d shape
    R = R.reshape((le,wi,k))

    return R

```

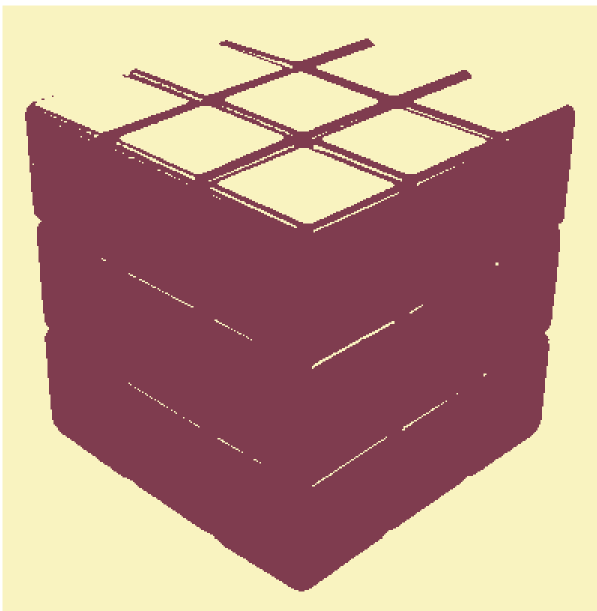
```
def kmeans(X, k):
    # initializing shapes
    le, wi, n = X.shape
    y = np.zeros([le,wi,])
    # initializing R as 3d shape
    R = np.zeros((le,wi,k))
    i = 0
    verbose = 0
    maxiter = 200
    tolerance = 1e-2

    # Initializing means randomly
    np.random.seed(9)
    Miu = np.random.uniform(low=np.min(X), high=np.max(X), size=(k, n))

    # iterate until convergence or maxiter
    while True:
        # stopping criterion maxiter
        if i>maxiter:
            break
        i += 1

        if verbose == 1:
            print(Miu)

        try:
            # calculating the cluster pixel for
            newR = getCluster(X, Miu)
            # if newR ~ R, then break
            assert np.abs(newR - R).sum() < tolerance*newR.size
            break
        except AssertionError:
            # if R is different than newR, then plugin the newR
            R = newR
            # calculating the new cluster means
            Miu = getCenter(X, R)

    # calculate y as the cluster pixel
    y = np.argmax(R.reshape((le*wi, k)), axis=1).reshape((le, wi))

    return (y, Miu)

```

Using k-means cluster all the pixels of an image into k clusters and assign each pixel the color represented by its nearest cluster center.

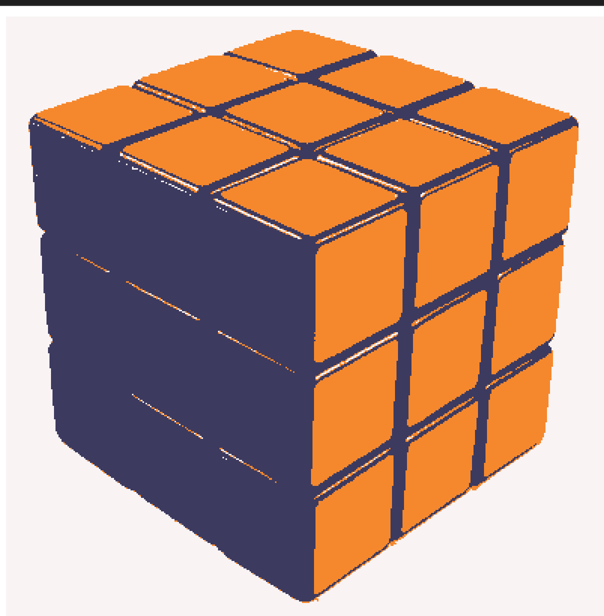
```
def getImage(X, k):
```

```
    # initializing segmented image
    seImage = np.zeros(X.shape)
    # run k-means
    y, Miu = kmeans(X, k)

    # assigning each pixel the color of the nearest cluster center
    for i in range(k):
        seImage[np.where(y == i)] = Miu[i]

    # generating the segmented Image
    data = Image.fromarray(np.uint8(seImage))

    return data
```

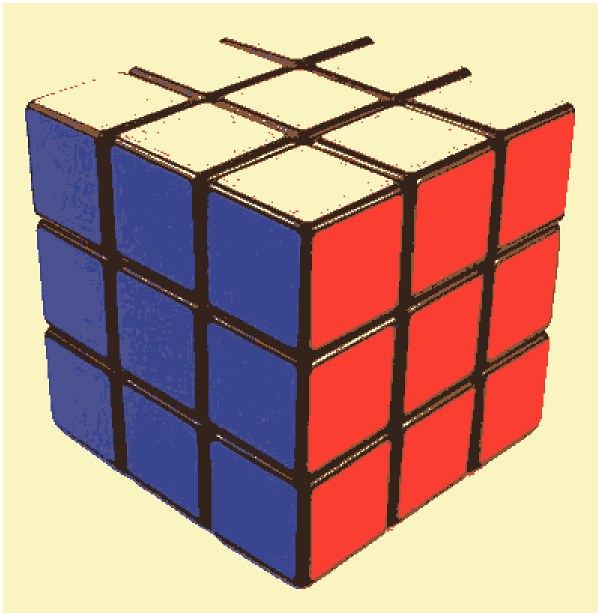
```
# 2 clusters
image = getImage(arr, 2)
image
```



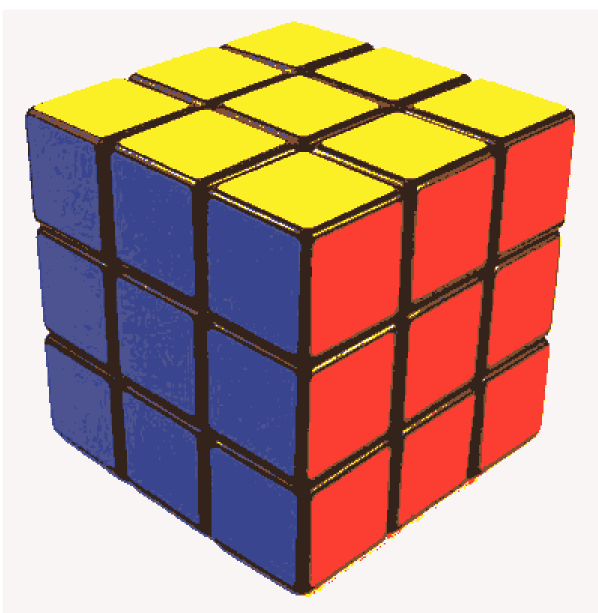
```
# 3 clusters
image = getImage(arr, 3)
image
```



```
# 6 clusters
image = getImage(arr, 6)
image
```



```
# 7 clusters
image = getImage(arr, 7)
image
```



These are different clusters where each pixel color is taken from its nearest cluster center.