1. Given: Distance matrix:

	A	B	C	D	E	F
A	0					
B	0.12	0				
C	0.51	0.25	0			
D	0.84	0.16	0.14	0		
E	0.28	0.77	0.70	0.45	0	
F	0.34	0.61	0.93	0.2	0.67	0

(a) Single link Dendogram (min dist)

(i)

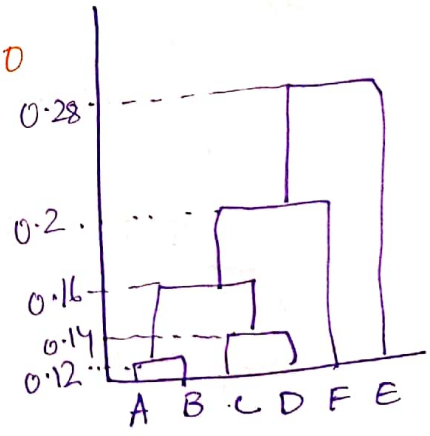
	AB	C	D	E	F
AB	0				
C	0.25	0			
D	0.16	0.14	0		
E	0.28	0.70	0.45	0	
F	0.34	0.93	0.2	0.67	0

(ii)

	AB	CD	E	F
AB	0			
CD	0.16	0		
E	0.28	0.45	0	
F	0.34	0.2	0.67	0

(iii)

	ABCD	E	F
ABCD	0		
E	0.28	0	
F	0.2	0.67	0



(iv)

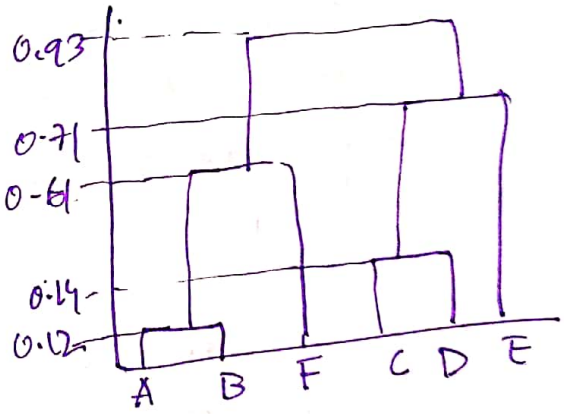
	ABCDF	E
ABCDF	0	
E	0.28	0

b) Complete linkage dendogram (max dist)

	A	B	C	D	E	F
A	0					
B	0.12	0				
C	0.51	0.25	0			
D	0.84	0.16	0.14	0		
E	0.28	0.77	0.70	0.45	0	
F	0.34	0.61	0.93	0.20	0.67	0

	AB	C	D	E	F
AB	0				
C	0.51	0			
D	0.84	0.16	0		
E	0.77	0.70	0.45	0	
F	0.61	0.93	0.20	0.67	0

	AB	CD	E	F
AB	0			
CD	0.84	0		
E	0.77	0.70	0	
F	0.61	0.93	0.67	0



	ABF	CD	E
ABF	0		
CD	0.13	0	
E	0.77	0.7	0

	ABF	CDE
ABF	0	
CDE	0.93	0

(c) Changing 2 values in the matrix so that the answer remains same.

- We see that complete link differs from single link where AB, Fare grouped by dist (AB, F) = dist (B, F) = 0.61. We want dist (AB, CD) = dist (A, D) to be smaller than this value hence

~~dist (A, D) = 0.53~~

dist (A, D) = 0.53

- and dist (ABCD, F) = dist (C, F) = 0.93 to be the smallest so that ABCD, Fare grouped together i.e

dist (C, F) = 0.63

Even after the above changes the dendograms obtained are identical to the previous solutions

(a) Prove that

$$\sigma_{11}+\sigma_{22}+\sigma_{33}+\cdots+\sigma_{pp} = \sum_{i=1}^{p} \text{Var}\, X_i = \lambda_1+\lambda_2+\lambda_3+\cdots+\lambda_p = \sum_{i=1}^{p} \text{Var}\, Y_i$$

Given:

$X' = [X_1, X_2, \ldots X_p]$

Σ = Covariance matrix

(λ_i, e_i) = EValue, EVector pair of Σ

$\forall i \quad Y_i = e_i' X \Rightarrow$ Principal components (P.C)

Soln

Algebraically, we know that PCs are linear combinations of Random variables

i.e as given above:

$$Y_1 = e_1' X = e_{11}X_1 + e_{12}X_2 + \cdots + e_{1p}X_p$$

$$\vdots$$

$$Y_p = e_p' X = e_{p1}X_1 + e_{p2}X_2 + \cdots + e_{pp}X_p$$

Then, we know that

$$\text{var}(Y_i) = e_i' \Sigma e_i$$

Now, by definition:

1st PC = LC of $e_1' X$ that maximizes

$$\text{Var}(Y_i) = \text{Var}(e_1' X)$$

subject to $e_1' e_1 = 1$

normalizing the combination vector

i.e

$$\max_{e} \left(\frac{e'\Sigma e}{e'e} \right) = \text{Var}(Y_1) = e'\Sigma e$$

$$= e'\lambda e = \lambda e'e$$

$$\max_{e} \left(\frac{e'\Sigma e}{e'e} \right) = \lambda_1$$

Thus solving the above maximisation problem.

$$e_1'\Sigma e_1 = \text{var}(Y_1)$$

$$e_1'e_1$$

$$e_1'\Sigma e_1 = \text{var}(Y_1)\, e_1'e_1$$

$$e_1'(\Sigma e_1 - \text{var}(Y_1)e_1) = 0$$

$$\Sigma e_1 = \text{var}(Y_1)\, e_1$$

~~$\Sigma = \text{var}(Y_1)$~~

$$\lambda_1 e_1 = \text{var}(Y_1)\, e_1$$

$$\boxed{\lambda_1 = \text{var}(Y_1)} \quad ; \quad \boxed{Y_1 = e_1'\Sigma}$$

$$\therefore \Sigma \text{var}(Y_i) = \Sigma\lambda_i = \lambda_1 + \lambda_2 + \cdots + \lambda_p$$

Now for $\text{Var}(X_i)$

W.K.T.

$$\text{Var}(X_i) = tr(\Sigma) = \Sigma\lambda_i$$

$$\sigma_{11} + \sigma_{22} + \cdots + \sigma_{pp} = \lambda_1 + \lambda_2 + \cdots + \lambda_p$$

Hence proved

$$\sigma_{11} + \sigma_{22} + \cdots + \sigma_{pp} = \sum_{i=1}^{p} \text{Var}(X_i) = \lambda_1 + \lambda_2 + \cdots + \lambda_p = \sum_{i=1}^{p} \text{Var}(Y_i)$$

2b) Given: RV : x_1, x_2, x_3

Cov matrix $\Sigma = \begin{bmatrix} 1 & -2 & 0 \\ -2 & 5 & 0 \\ 0 & 0 & 2 \end{bmatrix}$

Eigen Value	Eigen Vector
$\lambda_1 = 5.83$	$e_1' = (0.383, -0.924, 0)$
$\lambda_2 = 2.00$	$e_2' = (0, 0, 1)$
$\lambda_3 = 0.17$	$e_3' = (-0.924, 0.383, 0)$

(i) Principal Components:

$$Y_1 = e_1' X = 0.383x_1 - 0.924x_2$$

$$Y_2 = e_2' X = x_3$$

$$Y_3 = e_3' X = -0.924x_1 + 0.383x_2$$

(ii) x_3 is one of the principal components because it is uncorrelated with the other two variables

(iii) $Var(Y_1) = Var(0.383x_1 - 0.924x_2)$

$$= (0.383)^2 Var x_1 + (-0.924)^2 Var(x_2) + 2(0.383)(-0.924) Cov(x_1, x_2)$$

From the Σ matrix; $Var(x_1) = 1$, $Var(x_2) = 5$, $Cov(x_1, x_2) = -2$

Substituting the values,

$$\boxed{Var(Y_1) = 5.83 = \lambda_1}$$

$$\boxed{Var(Y_2) = Var(Y_3) = 2 = \lambda_2}$$

$$Var(Y_3) = Var(-0.924x_1 + 0.383x_2)$$

$$= (-0.924)^2 Var(x_1) + (0.383)^2 Var(x_2) + 2(-0.924)(0.383)\, Cov(x_1, x_2)$$

Substituting the values from Σ matrix

$$\boxed{Var(Y_3) = 0.17 = \lambda_3}$$

$$Cov(Y_1, Y_2) = Cov(0.383x_1 - 0.924x_2, x_3)$$

$$= 0.383\, Cov(x_1, x_3) - 0.924\, Cov(x_2, x_3)$$

From Σ matrix $Cov(x_1, x_3) = 0$

$Cov(x_2, x_3) = 0$

Substituting the values:

$$\boxed{Cov(Y_1, Y_2) = 0}$$

$$Cov(Y_2, Y_3) = Cov(x_3, -0.924x_1 + 0.383x_2)$$

$$= -0.924\, Cov(x_3, x_1) + 0.383\, Cov(x_3, x_2)$$

$$\boxed{Cov(Y_2, Y_3) = 0}$$

$$Cov(Y_1, Y_3) = Cov(0.383x_1 - 0.924x_2, 0.924x_1 + 0.383x_2)$$

$$= -0.383(0.924)\, Cov(x_1, x_1) + (0.924)^2 Cov(x_1, x_2) + (0.383)^2 Cov(x_1, x_2) - (0.383)(0.924)\, Cov(x_2, x_2)$$

$$= [illegible]$$

$$\boxed{Cov(Y_1, Y_3) = 0}$$

(iv) Can any PC be ignored?

Proportion of total variance accounted for by

1st PC $\frac{\lambda_1}{\sum\lambda} = \frac{5.83}{8} = 0.73$

2nd PC $\frac{\lambda_2}{\sum\lambda} = \frac{2}{8} = 0.25$

3rd PC $\frac{\lambda_3}{\sum\lambda} = \frac{0.17}{8} = 0.02$

As we see, the 1st two PCs account for

$0.73 + 0.25 = 0.98$ of the population variance

Hence, Y_1 & Y_2 could replace the three variables with little loss of info,

$$B = \lambda_1 e_1 e_1' + \lambda_2 e_2 e_2'$$

$$= \begin{bmatrix} 0.8552 & -2.063 & 0 \\ -2.063 & 4.478 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

$$\approx \Sigma$$

Hence the 3rd PC can be ignored

3. EM Application

(a) we will derive the E-step:

(i) The joint distribution $p(y^{(pr)}, z^{(pr)}, z^{(pr)})$ has the form of a multivariate Gaussian density. Find its associated mean vector and co-variance matrix in terms of the parameters $\mu_p, \sigma_p^2, \nu_r, \tau_r^2$ and σ^2.

(ii) Derive an expression for $Q_{pr}(y^{(pr)}, z^{(pr)}) = p(y^{(pr)}, z^{(pr)} | z^{(pr)})$ (E-Step), using the rules for conditioning on subsets of jointly Gaussian RV.

(b) Derive the M-step updates to the parameters $\{\mu_p, \sigma_p^2, \nu_r, \tau_r^2\}$.

Soln Let Θ denote the whole set of parameters we are estimating, then the EM steps for our problem are (at a higher level):

(a) (E-step) For each p, r set

$$Q_{pr}(y^{(pr)}, z^{(pr)}) = p(y^{(pr)}, z^{(pr)} | z^{(pr)}; \Theta)$$

(b) (M-step) ~~Set~~ $\Theta = \text{argmax}_{\Theta} \sum_{p=1}^{P} \sum_{r=1}^{R} E_{Q_{pr}(y^{(pr)}, z^{(pr)})} \log p(z^{(pr)}, y^{(pr)}, z^{(pr)}; \Theta)$.

Set $\Theta = \text{argmax}_{\Theta} \sum_{p=1}^{P} \sum_{r=1}^{R} E_{Q_{pr}(y^{(pr)}, z^{(pr)})} \log p(z^{(pr)}, y^{(pr)}, z^{(pr)}; \Theta)$.

For the E-step $p(y^{(pr)}, z^{(pr)} | z^{(pr)})$, if we use Bayes' Rule to compute, we'll obtain integrals on denominator which might be difficult to compute

Instead, we see that

$$p(y^{(pr)}, z^{(pr)}, x^{(pr)}) = p(y^{(pr)}, z^{(pr)})\, p(x^{(pr)} \mid y^{(pr)}, z^{(pr)})$$

$$\boxed{p(y^{(pr)}, z^{(pr)}, x^{(pr)}) = p(y^{(pr)})\, p(z^{(pr)})\, p(x^{(pr)} \mid y^{(pr)}, z^{(pr)})}$$

$\Downarrow$

is the product of three Gaussian densities

hence itself is a Multivariate Gaussian density

$\Downarrow$

this joint distribution is some type of normal distribution, hence we can use rules for conditioning Gaussians to compute this

W.k.t. a Multivariate Gaussian Density is fully parametrized by its mean and covariance matrix

- To compute the mean vector, from the question $\boxed{x^{(pr)} = y^{(pr)} + z^{(pr)} + e^{(pr)}}$; $e^{(pr)} \sim N(0, \sigma^2)$ noise

Then $E[y^{(pr)}] = \mu_p$, $E[z^{(pr)}] = \nu_{pr}$, and

$$E[x^{(pr)}] = E[y^{(pr)} + z^{(pr)} + e^{(pr)}] = \mu_p + \nu_r + 0$$

$$\boxed{E[x^{(pr)}] = \mu_p + \nu_r}$$

- To compute the covariance matrix

$$\text{Var}[y^{(pr)}] = \sigma_p^2 \qquad \text{Var}[z^{(pr)}] = \tau_r^2$$

$$\text{Cov}[y^{(pr)}, z^{(pr)}] = 0 \quad (\because y, z \text{ are independent})$$

Also $y^{(pr)}, z^{(pr)}, e^{(pr)}$ are also independent

$$\text{Var}[x^{(pr)}] = \text{Var}(y^{(pr)}, z^{(pr)}, e^{(pr)}) = \text{Var}$$

$$\boxed{\text{Var}[x^{(pr)}] = \sigma_p^2 + \tau_r^2 + \sigma^2}$$

Finally, $Cov(y^{(pr)}, x^{(pr)}) = Cov(z^{(pr)}, y^{(pr)})$

$= Cov(z^{(pr)} + y^{(pr)} + e^{(pr)}, xy^{(pr)})$

From independence

$= 0 + \sigma_p^2 + 0 = \sigma_p^2$ //

This can be written as:

$$y^{(pr)}, z^{(pr)}, x^{(pr)} \sim \mathcal{N}\left(\begin{bmatrix} \mu_p \\ \nu_r \\ \mu_p + \nu_r \end{bmatrix}, \begin{bmatrix} \sigma_p^2 & 0 & \sigma_p^2 \\ 0 & \tau_r^2 & \tau_r^2 \\ \sigma_p^2 & \tau_r^2 & \sigma_p^2 + \tau_r^2 + \sigma^2 \end{bmatrix}\right)$$

Now, we can use standard results for conditioning on subsets of variables for Gaussians to obtain

$$Q_{pr}(y^{(pr)}, z^{(pr)}) = \mathcal{N}\left(\begin{bmatrix} \mu_{pr,y} \\ \mu_{pr,z} \end{bmatrix}, \begin{bmatrix} \Sigma_{pr,yy} & \Sigma_{pr,yz} \\ \Sigma_{pr,zy} & \Sigma_{pr,zz} \end{bmatrix}\right)$$

where,

$$\mu_{pr} = \begin{bmatrix} \mu_{pr,y} \\ \mu_{pr,z} \end{bmatrix} = \begin{bmatrix} \mu_p + \frac{\sigma_p^2}{\sigma^2 + \sigma_p^2 + \tau_r^2}(x^{(pr)} - \mu_p - \nu_r) \\ \nu_r + \frac{\tau_r^2}{\sigma^2 + \sigma_p^2 + \tau_r^2}(x^{(pr)} - \mu_p - \nu_r) \end{bmatrix} \quad \text{---(1)}$$

$$\Sigma_{pr} = \begin{bmatrix} \Sigma_{pr,yy} & \Sigma_{pr,yz} \\ \Sigma_{pr,zy} & \Sigma_{pr,zz} \end{bmatrix} = \frac{1}{\sigma_p^2 + \sigma^2 + \tau_r^2} \begin{bmatrix} \sigma_p^2(\tau_r^2 + \sigma^2) & -\sigma_p^2\tau_r^2 \\ -\sigma_p^2\tau_r^2 & \tau_r^2(\sigma_p^2 + \sigma^2) \end{bmatrix}$$

---(2)

For the M Step

An important realization is that the Q_{pr} distribution is defined in terms of θ^t, while we want to choose the parameters for the next step, θ^{t+1}.

This implies that $Q^{(pr)}$ distributions are constant in terms of parameters, we wish to maximize.

Maximizing the expected log-likelihood, we have

(letting $E_Q[\cdot]$ denote expectations with respect to $Q_{pr}(y^{(pr)}, z^{(pr)})$ for each p and r, respectively)

$$\theta = \arg\max_{\theta} \sum_{p=1}^{P} \sum_{r=1}^{R} E_Q \log p(x^{(pr)}, y^{(pr)}, z^{(pr)}; \theta)$$

expanding $\log p(x^{(pr)}, y^{(pr)}, z^{(pr)}; \theta)$

$$= \log \left[\frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(x^{(pr)} - y^{(pr)} - z^{(pr)})^2}{2\sigma^2}} \frac{1}{\sqrt{2\pi\sigma_p^2}} e^{-\frac{(y^{(pr)} - \mu_p)^2}{2\sigma_p^2}} \cdot \frac{1}{\sqrt{2\pi\sigma_r^2}} e^{-\frac{(z^{(pr)} - \nu_r)^2}{2\tau_r^2}} \right]$$

$$= \log \frac{1}{(2\pi)^{3/2}\sigma\sigma_p\tau_r} - \frac{1}{2\sigma^2}(x^{(pr)} - y^{(pr)} - z^{(pr)})^2 - \frac{1}{2\sigma_p^2}(y^{(pr)} - \mu_p)^2 - \frac{1}{2\tau_r^2}(y^{(pr)} - \nu_r)^2$$

$$= \log \frac{1}{\sigma_p\tau_r} - \frac{1}{2\sigma_p^2}(y^{(pr)} - \mu_p)^2 - \frac{1}{2\tau_r^2}(z^{(pr)} - \nu_r)^2$$

$$= \log \frac{1}{\sigma_p\tau_r} - \frac{1}{2\sigma_p^2}\left((y^{(pr)})^2 - 2y^{(pr)}\mu_p + \mu_p^2\right) - \cdots - \frac{1}{2\tau_r^2}\left((z^{(pr)})^2 - 2z^{(pr)}\nu_r + \nu_r^2\right)$$

Replacing the above in the main equation

$$= \underset{\Theta}{\operatorname{argmax}} \sum_{p=1}^{P}\sum_{r=1}^{R}\left[\log\frac{1}{\sigma_p\tau_r} - \frac{1}{2\sigma_p^2}\left(E_q[(y^{(pr)})^2] - 2E_q[y^{(pr)}]\mu_p + \mu_p^2\right) - \frac{1}{2\tau_r^2}\left(E_q[(z^{(pr)})^2] - 2E_q[z^{(pr)}]v_r + v_r^2\right)\right]$$

$$= \underset{\Theta}{\operatorname{argmax}} \sum_{p=1}^{P}\sum_{r=1}^{R}\left[\log\frac{1}{\sigma_p\tau_r} - \frac{1}{2\sigma_p^2}\left(\Sigma_{pr,yy} + \mu_{pr,y}^2 - 2\mu_{pr,y}\mu_p + \mu_p^2\right) - \frac{1}{2\tau_r^2}\left(\Sigma_{pr,zz} + \mu_{pr,z}^2 - 2\mu_{pr,z}v_r + v_r^2\right)\right]$$

as we know that, $E_q[y^{(pr)}] = \mu_{pr,y}$

$$E_q[z^{(pr)}] = \mu_{pr,z}$$

$$\left(E_q[(y^{(pr)})^2] - E_q[y^{(pr)}]^2\right) + E_q[y^{(pr)}]^2 = \Sigma_{pr,yy} + \mu_{pr,y}^2$$

similarly for $E_q[z^{(pr)}]$ also

Differentiating w.r.t the parameters

$\mu_p, v_r, \sigma_p, \tau_r$ & equating it to 0

$$\frac{-1}{2\sigma_p^2}\sum_{r=1}^{R}(2\mu_p - 2\mu_{pr,y}) = 0 \Rightarrow \mu_p = \frac{1}{R}\sum_{r=1}^{R}\mu_{pr,y} \quad \text{---(3)}$$

$$\frac{-1}{2\tau_r^2}\sum_{p=1}^{P}(2v_r - 2\mu_{pr,z}) = 0 \Rightarrow v_r = \frac{1}{P}\sum_{p=1}^{P}\mu_{pr,z} \quad \text{---(4)}$$

$$\sum_{r=1}^{R}\left[\frac{-1}{\sigma_p^2} + \frac{1}{\sigma_p^3}\left(\Sigma_{pr,yy} + \mu_{pr,y}^2 - 2\mu_{pr,y}\mu_p + \mu_p^2\right)\right] = 0$$

$$\Rightarrow \sigma_p^2 = \frac{1}{R}\sum_{r=1}^{R}\left(\Sigma_{pr,yy} + \mu_{pr,y}^2 - 2\mu_{pr,y}\mu_p + \mu_p^2\right) \quad \text{---(5)}$$

$$\sum_{p=1}^{P}\left[-\frac{1}{\tau_r}+\frac{1}{\tau_r^3}\left(\Sigma_{pr,zz}+\mu_{pr,z}^2-2\mu_{pr,z}v_r+v_r^2\right)\right]=0$$

$$\Rightarrow \tau_r^2=\frac{1}{P}\sum_{p=1}^{P}\left(\Sigma_{pr,zz}+\mu_{pr,z}^2-2\mu_{pr,z}v_r+v_p^2\right)$$

— (6)

Using the above results, we can restate our E & M steps in terms of actual computations:

(a) (E-Step) For each p, r, compute μ_{pr}, Σ_{pr} using eqns (1) & (2)

(b) (M-step) Compute $\mu_p, v_r, \sigma_p^2, \tau_r^2$ using equations (3), (4), (5), (6)

1. Clustering

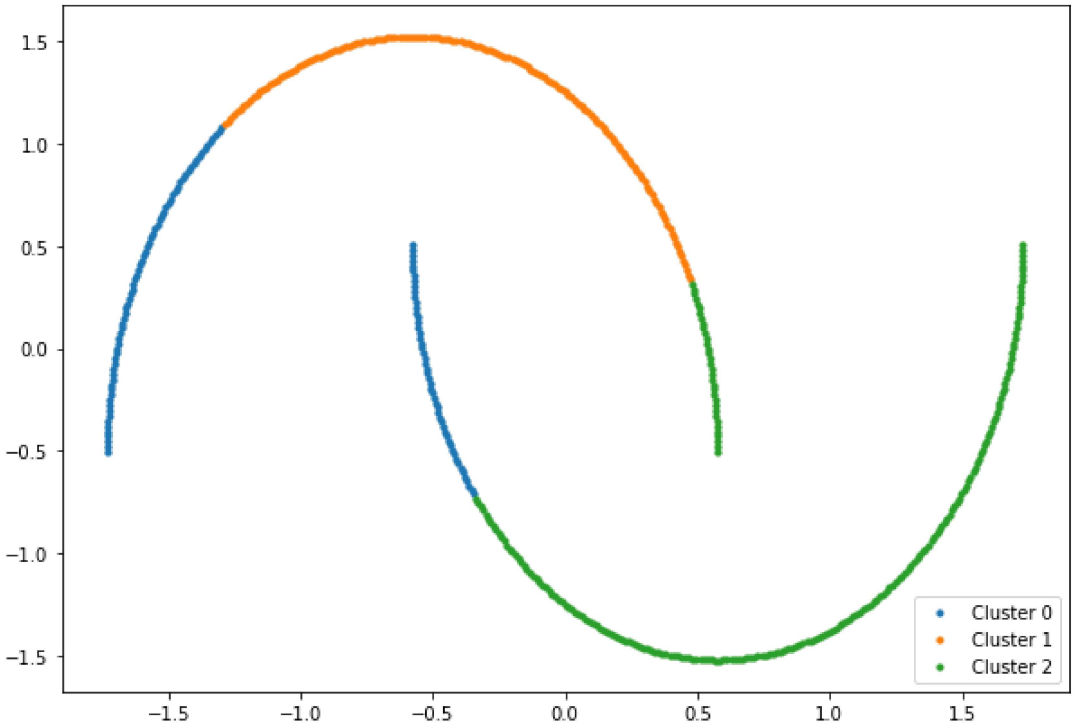
In [ ]:
```
import numpy as np
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from itertools import cycle, islice
import matplotlib.pyplot as plt
import queue
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.neighbors import NearestNeighbors
```

In [ ]:
```
# Reading from the data file
df = pd.read_csv("/content/dataset1.txt", header = None)
df.columns = ["data"]
df = df.join(df['data'].str.split(' ', expand = True).rename(columns = {0:'daata', 1:'x1', 2:'x
del df['data']
df.columns = ['x1', 'x2']
dataset = df.astype(float).values.tolist()
X1 = StandardScaler().fit_transform(dataset)

df = pd.read_csv("/content/dataset2.txt", header = None)
df.columns = ["data"]
df = df.join(df['data'].str.split(' ', expand = True).rename(columns = {0:'daata', 1:'x1', 2:'x
del df['data']
df.columns = ['x1', 'x2']
dataset = df.astype(float).values.tolist()
# normalize dataset
X2 = StandardScaler().fit_transform(dataset)
```

1a Implementing Kmeans clustering on dataset1

In [ ]:
```
clustering = KMeans(3).fit(X1)

plt.figure(figsize=(10,7))
for clust in np.unique(clustering.labels_):
    plt.scatter(X1[clustering.labels_ == clust, 0], X1[clustering.labels_ == clust, 1], s=10, l

plt.legend([f"Cluster {clust}" for clust in np.unique(clustering.labels_)], loc ="lower right")
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x7f002f990f50>
```



1b Implementing DBSCAN from scratch

In [ ]:

```
class DBSCAN_Scratch():
    def __init__(self):
        self.core = -1
        self.border = -2

    # Find all neighbour points at epsilon distance
    def find_neighbours(self, data, pointIdx, eps):
        points = []
        for i in range(len(data)):
            # Euclidian distance
            if np.linalg.norm([a_i - b_i for a_i, b_i in zip(data[i], data[pointIdx])]) <= eps:
                points.append(i)
        return points

    # Fit the data into the DBSCAN model
    def fit(self, data, eps, minNumberPt):
        # initialize all points as outliers
        point_label = [0] * len(data)
        point_count = []

        # initilize list for core/border points
        core = []
        border = []

        # Find the neighbours of each individual point
        for i in range(len(data)):
            point_count.append(self.find_neighbours(data, i, eps))

        # Find all the core points, border points and outliers
        for i in range(len(point_count)):
            if (len(point_count[i]) >= minNumberPt):
                point_label[i] = self.core
                core.append(i)
            else:
                border.append(i)
        for i in border:
            for j in point_count[i]:
                if j in core:
                    point_label[i] = self.border
                    break
```

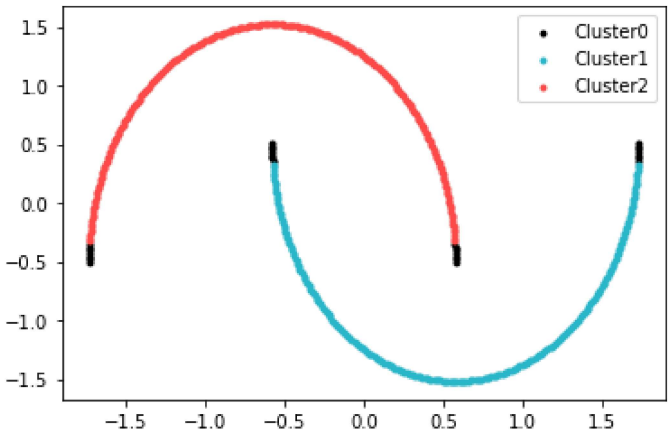
```
        # Assign points to a cluster

        cluster = 1

        # Here we use a queue to find all the neighbourhood points of a core point and find the
        # We are essentially performing Breadth First search of all points which are within Eps
        for i in range(len(point_label)):
            q = queue.Queue()
            if (point_label[i] == self.core):
                point_label[i] = cluster
                for x in point_count[i]:
                    if(point_label[x] == self.core):
                        q.put(x)
                        point_label[x] = cluster
                    elif(point_label[x] == self.border):
                        point_label[x] = cluster
                while not q.empty():
                    neighbors = point_count[q.get()]
                    for y in neighbors:
                        if (point_label[y] == self.core):
                            point_label[y] = cluster
                            q.put(y)
                        if (point_label[y] == self.border):
                            point_label[y] = cluster
                    cluster += 1  # Move on to the next cluster

        return point_label, cluster

    # Visualize the clusters
    def plotData(self, data, cluster, numberOfClusters):
        N = len(data)

        colors = np.array(list(islice(cycle(['#FE4A49', '#2AB7CA']), 3)))

        for i in range(numberOfClusters):
            if (i == 0):
                # Plot all outliers point as black
                color = '#000000'
            else:
                color = colors[i % len(colors)]

            x, y = [], []
            for j in range(N):
                if cluster[j] == i:
                    x.append(data[j, 0])
                    y.append(data[j, 1])
            plt.scatter(x, y, c=color, alpha=1, marker='.', label = 'Cluster'+str(i))
            plt.legend()
        plt.show()
```

In [ ]:

```
DBSCAN_1 = DBSCAN_Scratch()
point_labels, clusters = DBSCAN_1.fit(X1, 0.25,20)

print(point_labels)
print( clusters)
print(set(point_labels))

DBSCAN_1.plotData(X1, point_labels, clusters)
```

```
[1, 2, 1, 2, 1, 1, 0, 2, 2, 1, 1, 1, 2, 0, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 1, 2, 2, 1, 1, 2, 2,
2, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 2, 1, 2, 1, 2, 2, 2, 1, 2, 0, 1, 1, 2, 2, 2, 0, 2, 0, 1, 2,
2, 1, 2, 2, 1, 1, 2, 2, 1, 1, 1, 2, 2, 2, 1, 1, 2, 1, 0, 1, 2, 1, 1, 2, 1, 0, 2, 0, 1, 2, 2, 1,
1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 1, 2, 1, 2, 2, 2, 0, 2, 2, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1,
2, 2, 2, 2, 2, 2, 0, 1, 1, 2, 2, 1, 2, 1, 2, 2, 2, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 0, 2, 1, 1, 1,
2, 1, 2, 1, 0, 2, 2, 2, 1, 2, 2, 2, 2, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 2, 2, 2, 2,
1, 2, 1, 1, 2, 2, 2, 2, 2, 2, 2, 1, 1, 0, 2, 2, 2, 1, 2, 2, 2, 1, 1, 1, 2, 2, 1, 2, 2, 2, 1, 2,
1, 0, 1, 1, 2, 1, 1, 1, 1, 1, 0, 1, 1, 2, 1, 1, 0, 1, 2, 2, 2, 1, 1, 0, 2, 1, 2, 1, 2, 0, 2, 2,
1, 1, 1, 2, 2, 1, 0, 0, 0, 2, 1, 1, 1, 1, 2, 1, 1, 2, 2, 2, 2, 1, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1,
1, 1, 2, 1, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 1, 2, 2, 1, 2, 2, 1, 2, 1, 2, 2, 2, 1, 2, 2, 1, 1, 0,
2, 2, 1, 1, 2, 1, 2, 1, 2, 2, 0, 1, 2, 2, 1, 2, 2, 1, 2, 1, 1, 1, 2, 2, 2, 2, 2, 1, 1, 1, 2, 1,
```

```
2, 1, 2, 2, 1, 1, 2, 1, 2, 1, 2, 2, 2, 1, 2, 2, 1, 2, 1, 2, 2, 2, 2, 0, 1, 1, 2, 2, 0, 1, 2, 1,
2, 1, 0, 1, 1, 2, 1, 2, 1, 2, 0, 1, 2, 1, 1, 1, 1, 1, 2, 1, 2, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1,
1, 1, 2, 2, 1, 2, 2, 2, 1, 2, 1, 1, 1, 1, 1, 2, 2, 1, 1, 0, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 1,
1, 2, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 1, 1, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 1, 2, 1, 1, 1, 2, 2, 1,
2, 2, 2, 1, 1, 1, 1, 2, 2, 1, 1, 1, 2, 2, 2, 2, 1, 2, 1, 2, 2]
3
{0, 1, 2}
```



1c Difference between KMeans and DBSCAN algorithm outputs

We see that the clusters formed are different in both the algorithms, this is because:

- KMeans forms clusters that are spherical or convex in shape only while DBSCAN forms clusters arbitrarily (anyshape).
- KMeans cannot detect outliers which have been detected perfectly by the DBSCAN algorithm as the **points in the black coloured cluster**

1d KMeans and DBSCAN on Dataset 2

As we see **below** that the clustering of the given datapoints are same in both the algorithms. This is because **the given dataset is trivial thus resulting in the same output as opposed to the first dataset**

Advantages

KMeans: Much faster than DBSCAN

DBSCAN: No prior knowledge of the number of clusters required

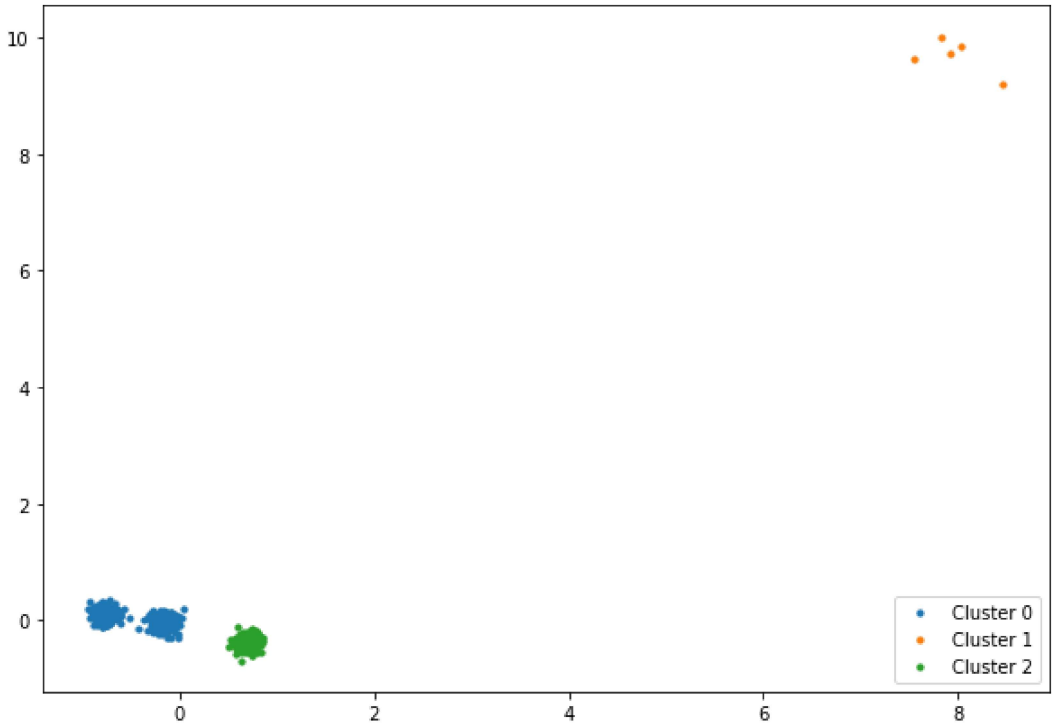
Disadvantages

KMeans: Requires an estimate for the number of clusters

DBSCAN: Does not work well with higher dimensional data and data with different densities

In [ ]:
```
clustering = KMeans(3).fit(X2)

plt.figure(figsize=(10,7))
for clust in np.unique(clustering.labels_):
    plt.scatter(X2[clustering.labels_ == clust, 0], X2[clustering.labels_ == clust, 1], s=10, l

plt.legend([f"Cluster {clust}" for clust in np.unique(clustering.labels_)], loc ="lower right")
```

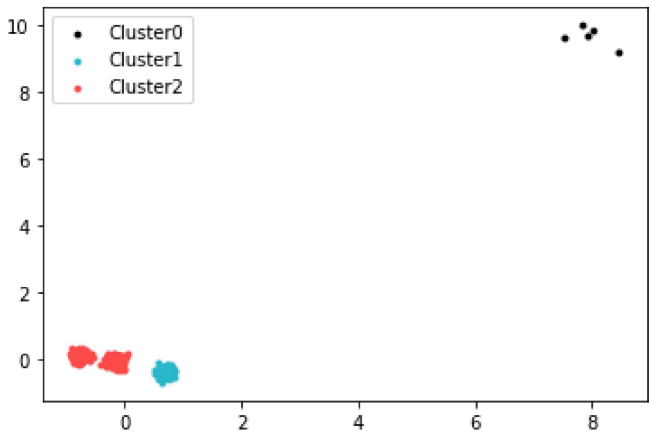
Out[ ]:
```
<matplotlib.legend.Legend at 0x7f002f4a0f10>
```



In [ ]:

```
DBSCAN_2 = DBSCAN_Scratch()
point_labels, clusters = DBSCAN_2.fit(X2, 0.25, 4)

print(point_labels)
print( clusters)
print(set(point_labels))

DBSCAN_2.plotData(X2, point_labels, clusters)
```

```
[1, 2, 2, 2, 2, 2, 2, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2, 2, 2, 1,
2, 2, 2, 2, 2, 2, 2, 1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2,
1, 1, 1, 2, 1, 2, 2, 2, 1, 1, 2, 2, 1, 2, 1, 1, 2, 2, 2, 2, 2, 2, 1, 1, 2, 2, 1, 2, 2, 2, 1, 1,
2, 2, 2, 1, 1, 2, 2, 2, 2, 1, 2, 1, 2, 2, 1, 1, 2, 1, 2, 2, 2, 1, 2, 1, 2, 2, 2, 2, 2, 2, 1, 1,
2, 2, 2, 2, 1, 1, 1, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 1, 1, 2, 2, 2, 1, 2, 2, 1, 1, 2, 2,
2, 1, 2, 2, 2, 1, 1, 2, 2, 1, 2, 1, 2, 2, 2, 2, 1, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2, 1, 2, 2, 1, 2,
2, 2, 1, 2, 2, 2, 2, 1, 1, 1, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 1, 2, 2, 1, 2, 2, 2, 2,
2, 2, 1, 2, 2, 2, 2, 1, 1, 2, 1, 2, 2, 2, 1, 1, 2, 2, 1, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 1, 2, 2,
1, 1, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 2, 2, 1, 1, 2, 2, 1, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1,
2, 2, 1, 2, 2, 1, 1, 2, 2, 2, 1, 2, 1, 1, 1, 1, 1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1,
2, 2, 2, 2, 2, 2, 2, 1, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 1, 1, 2, 2,
2, 2, 2, 2, 1, 2, 1, 1, 1, 2, 2, 2, 2, 1, 1, 2, 2, 1, 2, 1, 2, 2, 2, 1, 2, 1, 2, 2, 2, 2, 1, 1,
1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 1, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2,
2, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 1, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 1, 1, 1, 2, 1, 2,
2, 2, 1, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 1, 1, 1, 2, 2, 2, 2, 1, 1, 2, 1, 2, 1, 1,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 0, 0, 0, 0, 0]
3
{0, 1, 2}
```

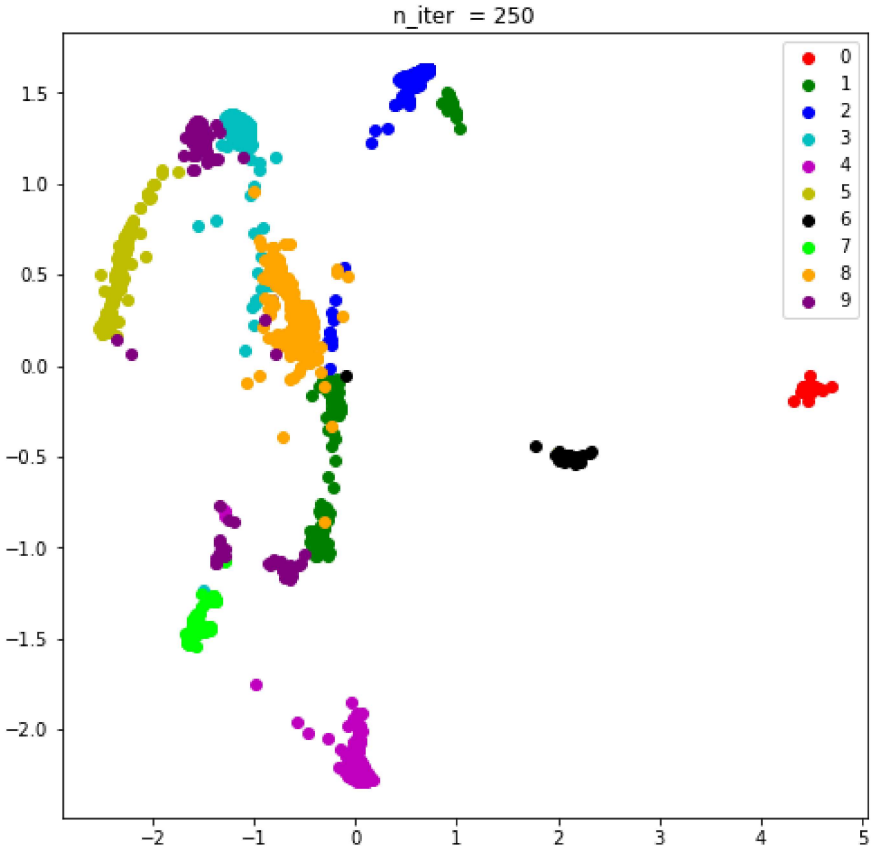


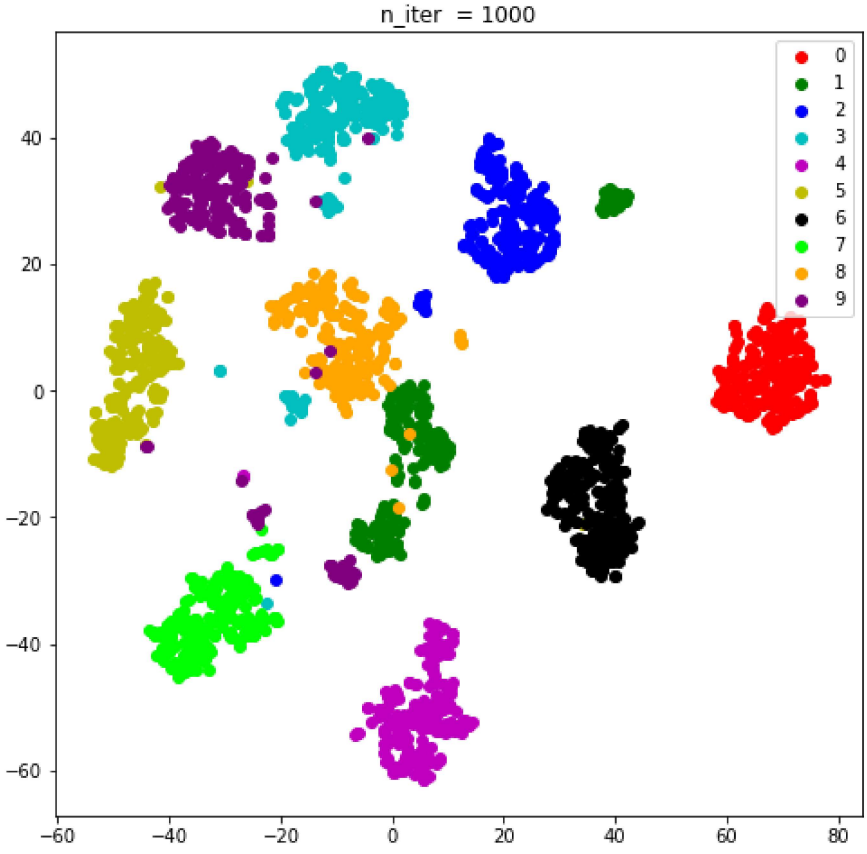
2. t-SNE

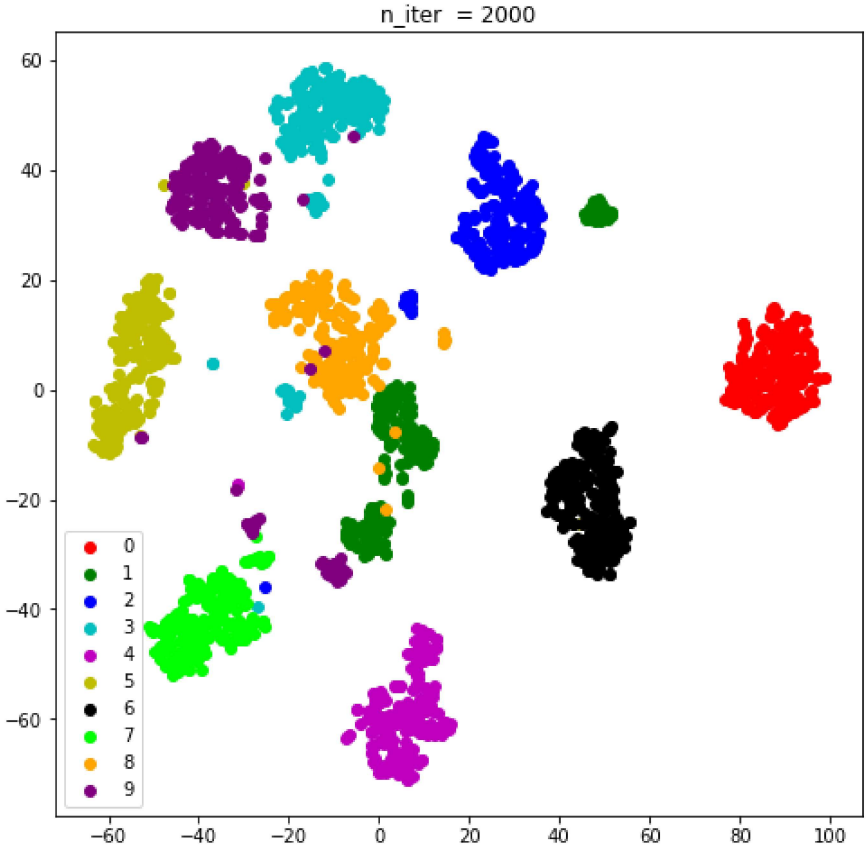
In [1]:
```
# Load the data
from sklearn import datasets
from sklearn.manifold import TSNE
from matplotlib import pyplot as plt
```

In [15]:
```
digits = datasets.load_digits()
data = digits.data
target = digits.target
print('Input data shape: ', data.shape)
iteration = [250,1000,2000]
# Fit and transform with a TSNE
for n_iter in iteration:
    tsne = TSNE(n_components=2, random_state=0, perplexity = 30, n_iter = n_iter)

    # Project the data in 2D
    reduced_data = tsne.fit_transform(data)
    print("Reduced Data shape: ",reduced_data.shape)
    # Visualize the data
    target_ids = range(len(digits.target_names))

    plt.figure(figsize=(8, 8))
    colors = 'r', 'g', 'b', 'c', 'm', 'y', 'k', 'lime', 'orange', 'purple'
    for i, c, label in zip(target_ids, colors, digits.target_names):
        plt.scatter(reduced_data[target == i, 0], reduced_data[target == i, 1], c=c, label=labe
    plt. title(label = 'n_iter  = '+str(n_iter))
    plt.legend()
plt.show()
```

```
Input data shape:  (1797, 64)
Reduced Data shape:  (1797, 2)
Reduced Data shape:  (1797, 2)
Reduced Data shape:  (1797, 2)
```







2(a) As the number of iterations increase the density of the clusters increase, this is because as the number of iterations increase the model gets more training on the dataset and hence better estimate of the similarity among the points belonging to the same class. We also see that as the number of iterations becomes high (1000,2000) the output remains the same as the method has reaches its convergent point i.e.,

minimizing the Kullback–Leibler divergence (KL divergence) has achieved it's lowest possible point.

2(b) In t-SNE the objective function (Kullback - Leibler divergence) is minimized using a gradient descent approach for optimization which is initiated randomly during every run which might be one of the reasons for the different outputs observed during different runs. We conduct several runs of t-SNE with the same parameters so as to achieve the lowest value for the objective function for our final visualisation.