Open in Colab
(https://colab.research.google.com/github/D34dP0oL/4216_Biomedical_DS_and_AI/blob/main/Sheet5/Assignment5_Solutions.ipynb)

In [ ]:

```
import numpy as np
import math
import pandas as pd
import random as rand
from random import randint
from collections import Counter
import bisect
from itertools import combinations

from sklearn import metrics
# from scipy.stats import poisson, sem, poisson, ttest_ind, shapiro, mannwhitneyu
# import scipy.stats as stats # for 'f_oneway'
# from scipy.cluster.hierarchy import cophenet
# from scipy.spatial.distance import pdist
# from scipy.spatial import distance

# from IPython.display import Image

# from sklearn import linear_model
# from sklearn.cluster import AgglomerativeClustering
# from sklearn.decomposition import PCA
# from sklearn.model_selection import train_test_split
# from sklearn.linear_model import LogisticRegression
# from sklearn.metrics import brier_score_loss

# import statsmodels.formula.api as sm # get ANOVA table as R like output
# from statsmodels.formula.api import ols # Ordinary Least Squares (OLS) model

import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture
# from IPython.display import display, HTML
# from scipy.cluster.hierarchy import dendrogram, linkage
# from mpl_toolkits.mplot3d import Axes3D

from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
```

Biomedical Data Science & AI

Assignment 5

Group members: Fabrice Beaumont, Fatemeh Salehi, Genivika Mann, Helia Salimi, Jonah

---

Exercise 1 - *k*-means clustering

1.1. Use the k-means algorithm and Euclidean distance to cluster the 10 data points into k = 3 clusters. The coordinates of the data points are given in table 1. Use the data points a4, a5 and a8 as initialization and perform 2 iteration steps. You can do the cluster assignment also visually without computing the exact distances.

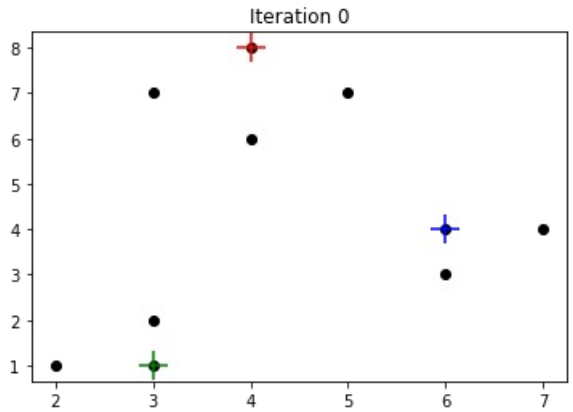
	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
(x, y)	(2, 1)	(5, 7)	(3, 2)	(4, 8)	(3, 1)	(7, 4)	(4, 6)	(6, 4)	(3, 7)	(6, 3)

In [ ]:

```
def plot_clustering(data, clusters, title, membership_vector=None, subplot_string=None):

    if membership_vector is None:
        membership_vector = [-1] * len(data)

    # Plot the data points with the colors of their cluster
    for i, _ in enumerate(data):
        if membership_vector[i] == -1:
            plt.scatter(data[i][0], data[i][1], color="black")
        else:
            plt.scatter(data[i][0], data[i][1], color=center_colors[membership_vector[i]], alpha=0.5)

    # Plot the cluster centers with different colors
    for i, _ in enumerate(clusters):
        plt.scatter(clusters[i][0], clusters[i][1], marker='+', s=300, color=center_colors[i])

    plt.title(title)
    plt.show()
```

In [ ]:

```
# We know, that clustering by hand is allowed. Still, we prefer to give some corresponding code
center_colors = ["red", "green", "blue"]
for _ in range(20-3):
    center_colors.append('#%06X' % randint(0, 0xFFFFFF))

# Initialize cluster centers (usually some sort of randomized)
x_centers = [4, 3, 6]
y_centers = [8, 1, 4]
# A more interesting initialization with the same outcome is:
# x_centers = [3, 5, 6]
# y_centers = [4, 2, 8]

# Example for a vanishing cluster center
# x_centers = [4, 5, 7]
# y_centers = [5, 3, 8]

x_data = [2, 5, 3, 4, 3, 7, 4, 6, 3, 6]
y_data = [1, 7, 2, 8, 1, 4, 6, 4, 7, 3]

plot_clustering(list(zip(x_data, y_data)), list(zip(x_centers, y_centers)), f"Iteration 0")
```

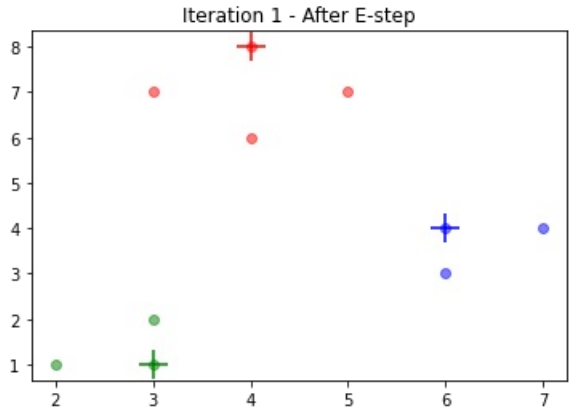


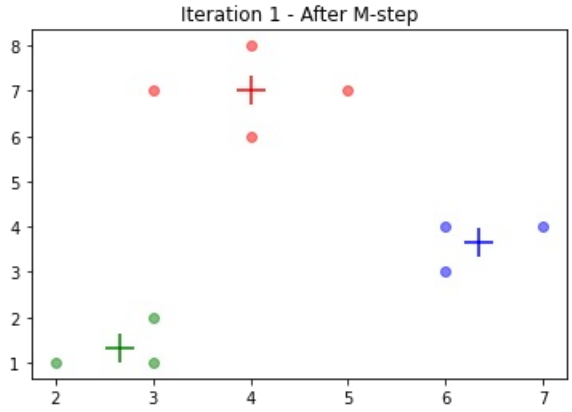
In [ ]:

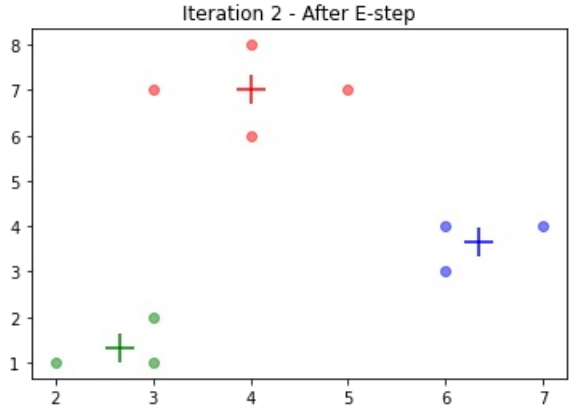
```
def my_k_means(data, centers, nr_iterations, membership_vector=None):

    if membership_vector is None:
        membership_vector = [-1] * len(data)

    for iteration in range(nr_iterations):

        # Iterate through all points and assign them to their closest cluster
        for point_index, point in enumerate(data):
            min_distance = math.inf

            for center_index, center in enumerate(centers):
                # Check if the point could be assignd to the cluster
                tmp_distance = distance.euclidean(point, center)
                # If a closer center has been found, assign the point to it
                if tmp_distance < min_distance:
                        min_distance = tmp_distance
                        membership_vector[point_index] = center_index

        # Print the intermediate clustering after estimating the data to the cluster centers
        plot_clustering(data, centers, f"Iteration {iteration+1} - After E-step", membership_vector)

        # Redefine the cluster centers
        for cluster_index in range(len(centers)):
            # Add all members of this cluster
            cluster_sum = [0, 0]
            cluster_size = 0
            for point_index, point in enumerate(data):
                if membership_vector[point_index] == cluster_index:
                    cluster_sum[0] += point[0]
                    cluster_sum[1] += point[1]
                    cluster_size = cluster_size + 1

            # Get the mean of the cluster
            cluster_mean = [cluster_sum[0] /cluster_size, cluster_sum[1] /cluster_size]
            centers[cluster_index] = (cluster_mean[0], cluster_mean[1])

        # Print the intermediate clustering after reassigning the cluster centers
        plot_clustering(data, centers, f"Iteration {iteration+1} - After M-step", membership_vector)

    plt.show()
```

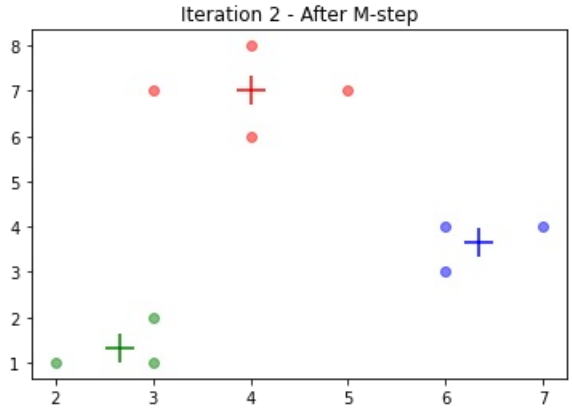
In [ ]:

```
# Execute the k-means using the initialization from above for two iterations
my_k_means(list(zip(x_data, y_data)), list(zip(x_centers, y_centers)), nr_iterations=2, membership_vector=cl
uster_memberships)
```









1.2. Shown are the results of a k-means clustering with two different initializations:

In [ ]:

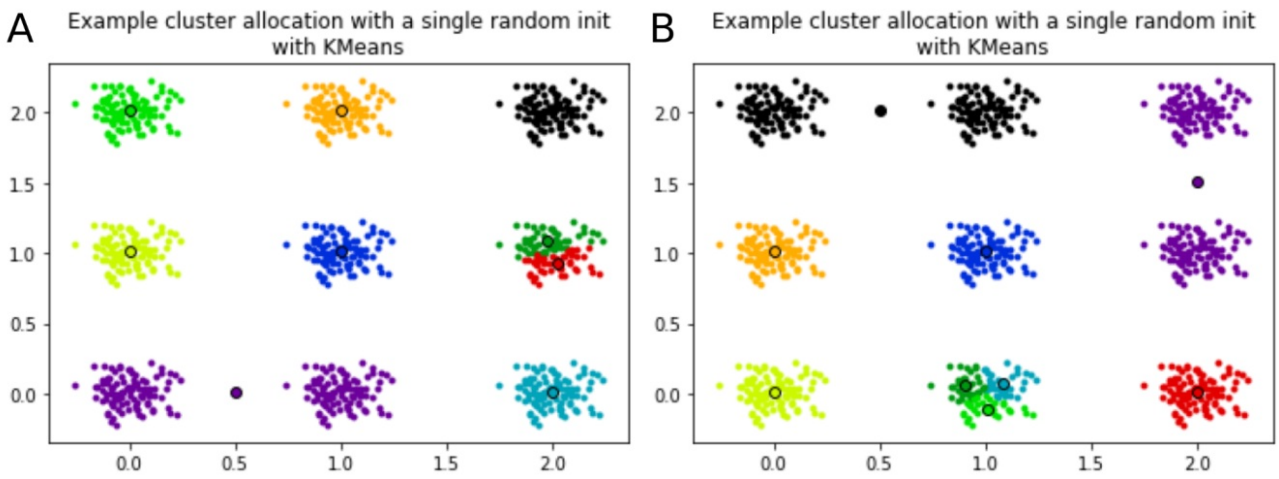
```
!wget https://raw.githubusercontent.com/D34dP0oL/4216_Biomedical_DS_and_AI/main/Sheet5/img_sheet5_ex1-2.png
```

```
--2021-05-26 15:34:06--  https://raw.githubusercontent.com/D34dP0oL/4216_Biomedical_DS_and_AI/m
ain/Sheet5/img_sheet5_ex1-2.png
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109
.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... con
nected.
HTTP request sent, awaiting response... 200 OK
Length: 250464 (245K) [image/png]
Saving to: ‘img_sheet5_ex1-2.png’

img_sheet5_ex1-2.pn 100%[===================>] 244.59K  --.-KB/s    in 0.07s

2021-05-26 15:34:07 (3.20 MB/s) - ‘img_sheet5_ex1-2.png’ saved [250464/250464]
```

In [ ]:

```
Image('img_sheet5_ex1-2.png')
```

Out[ ]:



1.2.a. How does the choice of the initial starting points affect the clustering?

Since in this variant of k-means, the number of cluter centers is fixed, the different initial starting points can not affect the number of cluters. However they can effect the **sizes of the clusters**. Most noticable, depending on the starting points, **empty** clusters may appear.

1.2.b. How can you avoid getting a clustering result that is dependent on the initialization?

This is a general question, and not directed at k-means. In general, no randomization should take place (e.g. the random initialization of the initializ starting points) to make the algorithm deterministic. Secondly, depending on the expected kind of clustering, different algorithms may simply be more suited by construction. (Note that k-means with Euclidean distance for example will always producte convex clusters!)

With respect to k-means, the dependence on the initial starting points can be reduced, by

- **running multiple executions** with different initializations. Then use some measurement to select the most suited outcome. For example by **peanalizing big differences incluster sizes**, or **cluster centers that are close** to each other.
- initializing **more cluster centers than needed** (**distributed uniformly** over the domain)
- allowing clusters to **collapse** (if the centers are to close) or to
- **vanish** (if empty).

Combinations of these methods are also possible.

1.2.c. What are the pros and cons of the k-means clustering?

Pros:

- Simple (easy to implement)
- Few parameters
- Rather fast (fast convergence)
- (Mostly) unsupervised

Cons:

- Only convex clusters
- Parameter k with huge impact on the solution
- Will probabilty stuck in a local optima
- Only hard cluster assignments
- Sensitivity to outliers

1.3. Use the provided breast cancer data (`cancer.csv`) to perform a k-means clustering. Perform the clustering for a range of clusters between 2 and 10. Set the `random_state` to 20 to keep reproducibility.

*We did not understand the constraints w.r.t. "range" and "random_state". We intperet this as running the algorithm with k = 2, ..., 10 initialized clusters and seed for the get random-int method of 20. We will run three iterations for every setting.

In [ ]:

```
nr_trials = 3
```

In [ ]:

```
breast_cancer_db = pd.read_csv('https://raw.githubusercontent.com/D34dP0oL/4216_Biomedical_DS_and_AI/main/Sheet5/cancer.csv', index_col='Unnamed: 0')
breast_cancer_db.head(4)
```

Out[ ]:

	mean radius	mean concavity
0	17.99	0.3001
1	20.57	0.0869
2	19.69	0.1974
3	11.42	0.2414

In [ ]:

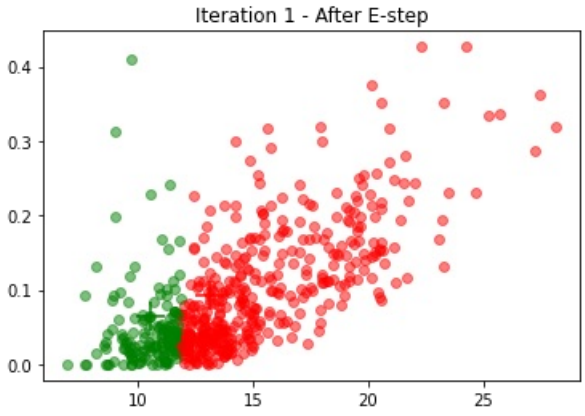
```
# Prepate the data
x_radii = breast_cancer_db['mean radius']
y_concavity = breast_cancer_db['mean concavity']
```

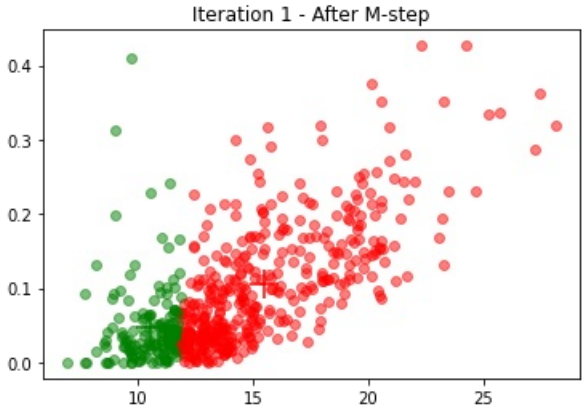
1.3.a. For each clustering plot the cluster assignment within a scatter plot for the features `mean radius` and `mean concavity` .

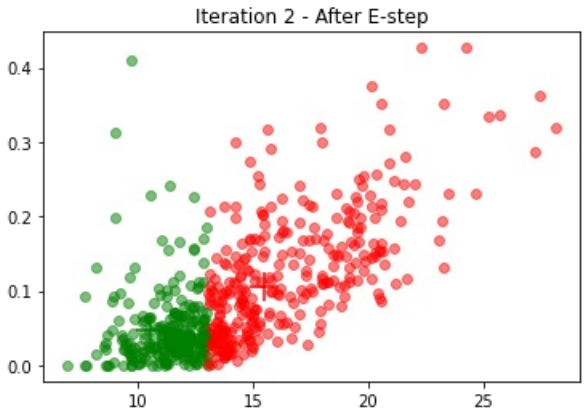
In [ ]:

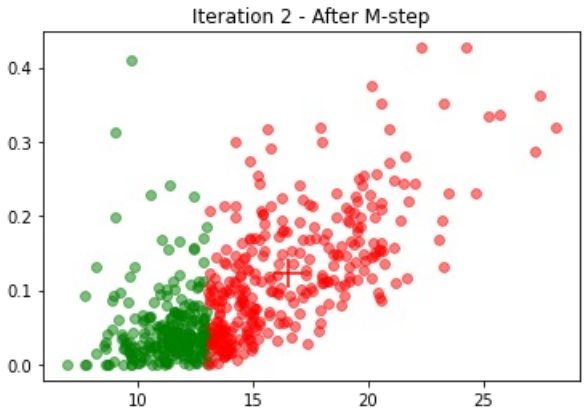
```
for k in range(2, 10):
    rand.seed(20)
    # Initialize k centers at random
    center_initialization_indices = rand.sample(range(0, breast_cancer_db.shape[0]-1), k)
    # Grab the centers
    x_radii_centers = [x_radii[i] for i in center_initialization_indices]
    y_concavity_centers = [y_concavity[i] for i in center_initialization_indices]

    # Run k-means for 'nr_trials' iterations
    print(f"k-means with k={k} clusters for {nr_trials} iterations:")
    my_k_means(list(zip(x_radii, y_concavity)), list(zip(x_radii_centers, y_concavity_centers)), nr_iteratio
ns=nr_trials)
```

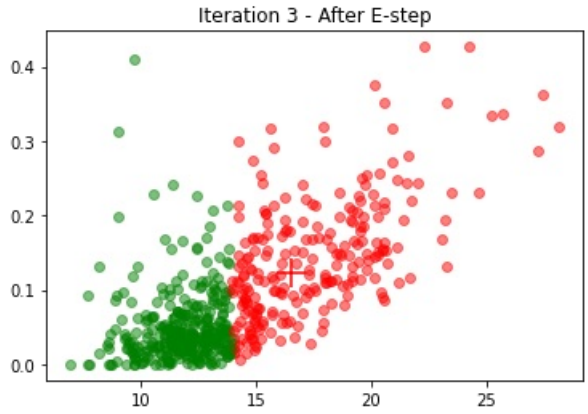
```
k-means with k=2 clusters for 3 iterations:
```

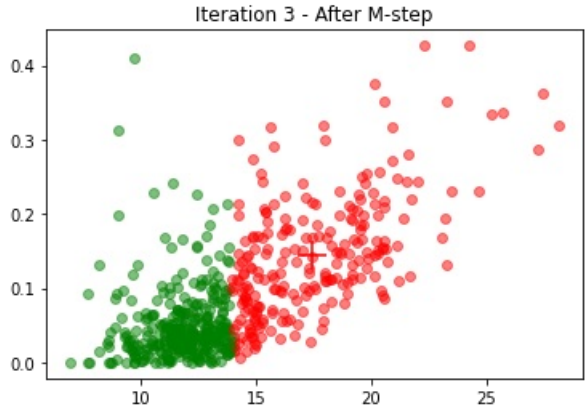

Iteration 1 - After E-step
0.4
0.3
0.2
0.1
0.0
10
15
20
25


Iteration 1 - After M-step
0.4
0.3
0.2
0.1
0.0
10
15
20
25

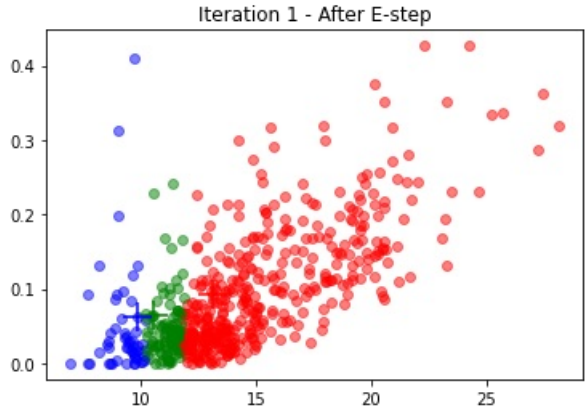

Iteration 2 - After E-step
0.4
0.3
0.2
0.1
0.0
10
15
20
25

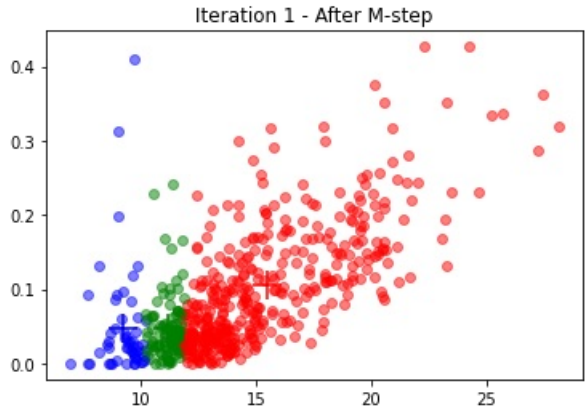

Iteration 2 - After M-step
0.4
0.3
0.2
0.1
0.0
10
15
20
25

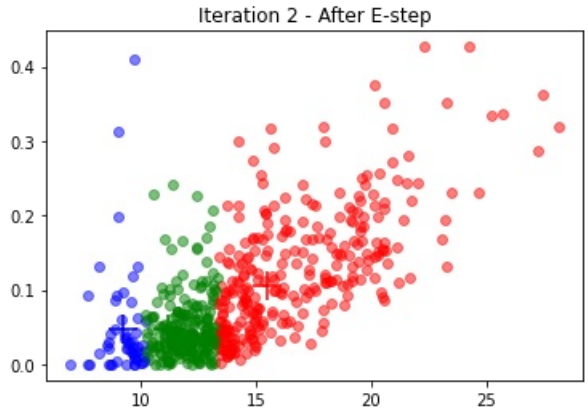


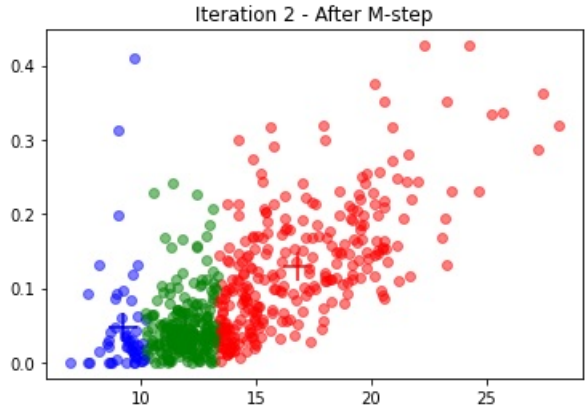


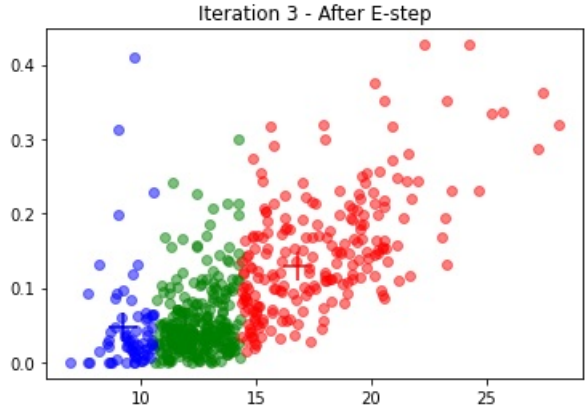
k-means with k=3 clusters for 3 iterations:

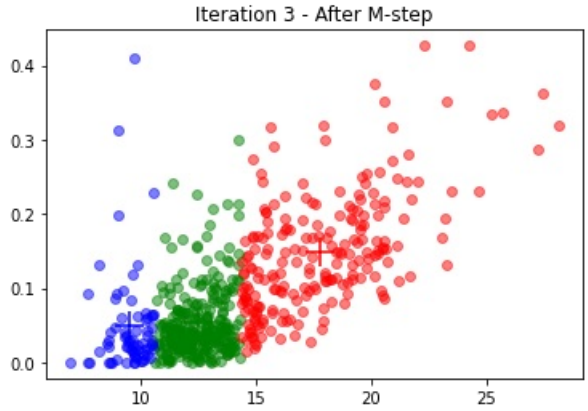




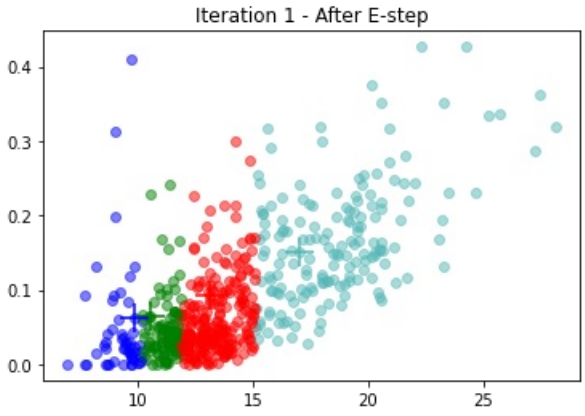


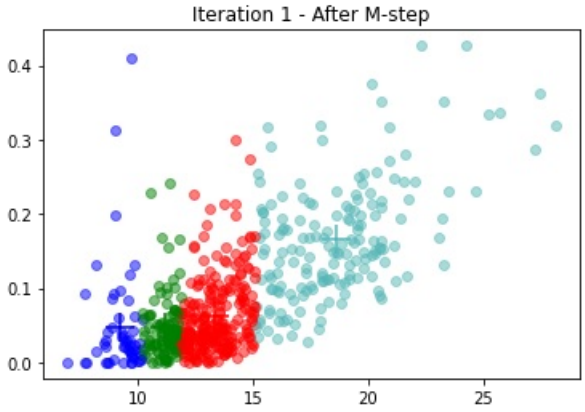


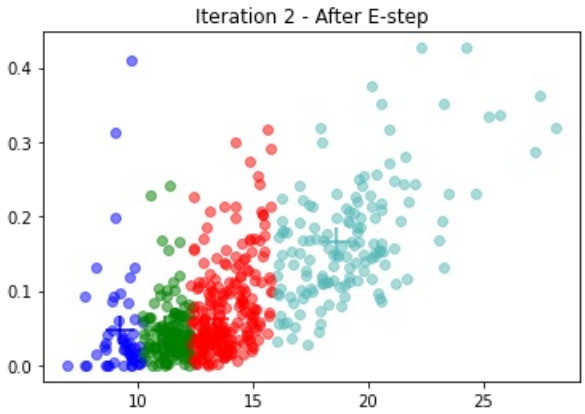


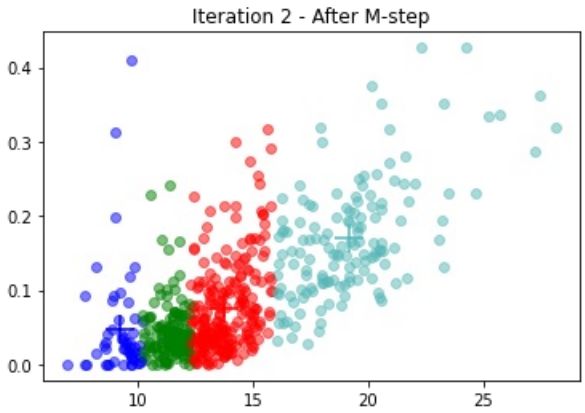


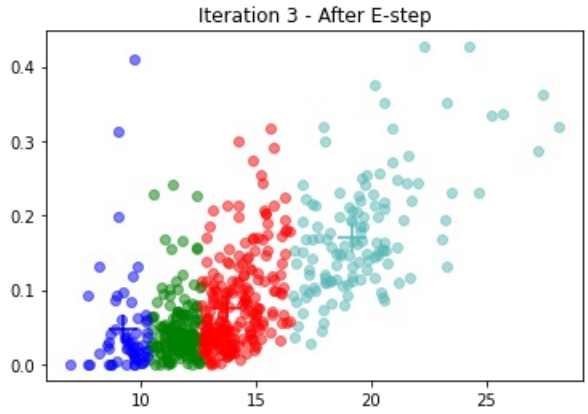
k-means with k=4 clusters for 3 iterations:

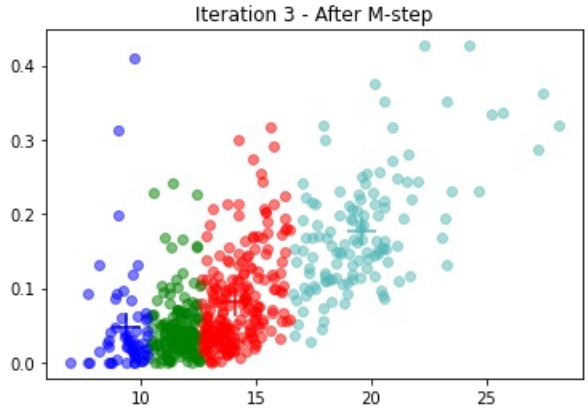

Iteration 1 - After E-step
0.4
0.3
0.2
0.1
0.0
10
15
20
25


Iteration 1 - After M-step
0.4
0.3
0.2
0.1
0.0
10
15
20
25

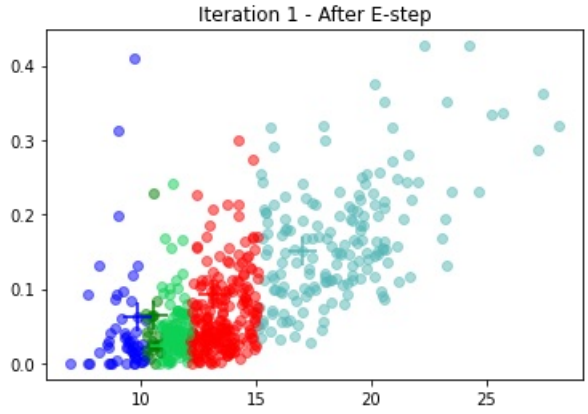

Iteration 2 - After E-step
0.4
0.3
0.2
0.1
0.0
10
15
20
25

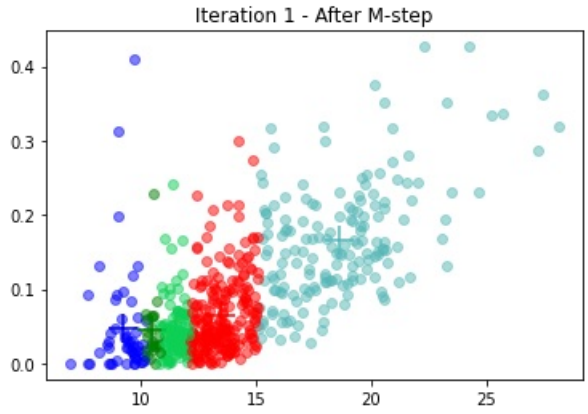

Iteration 2 - After M-step
0.4
0.3
0.2
0.1
0.0
10
15
20
25

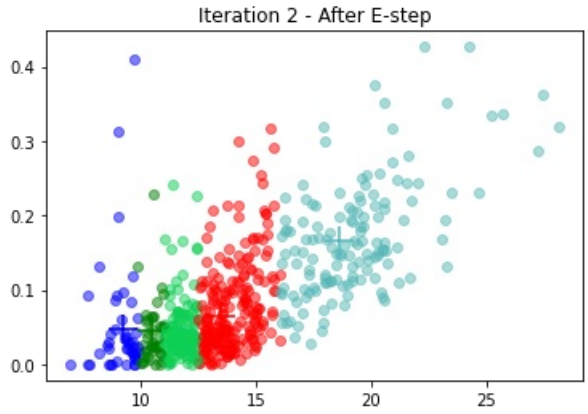


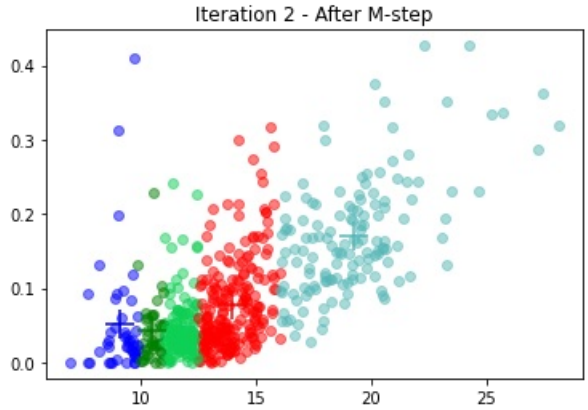


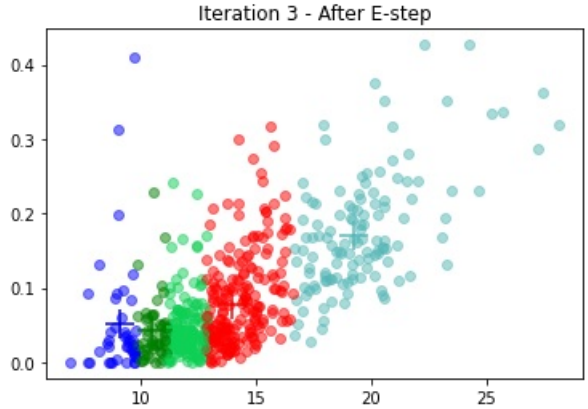
k-means with k=5 clusters for 3 iterations:

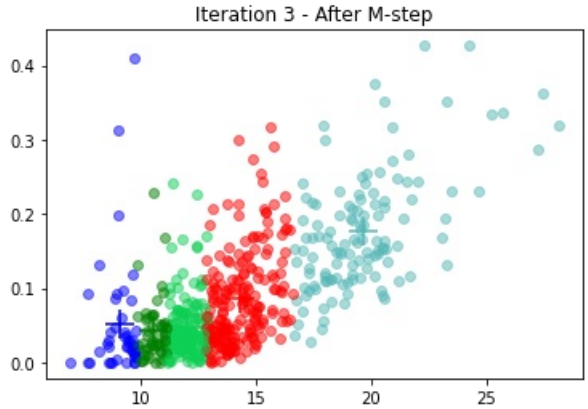




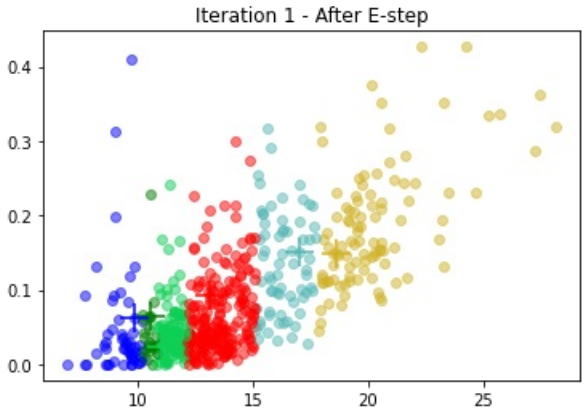


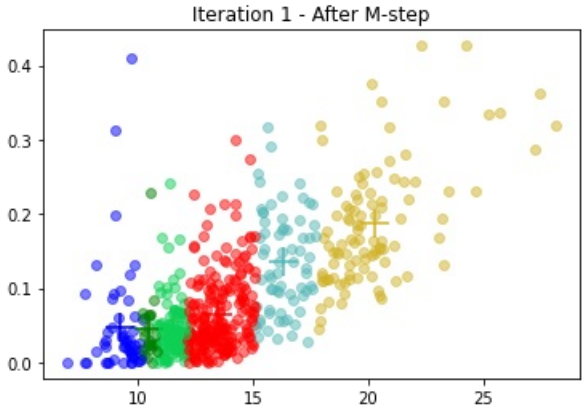


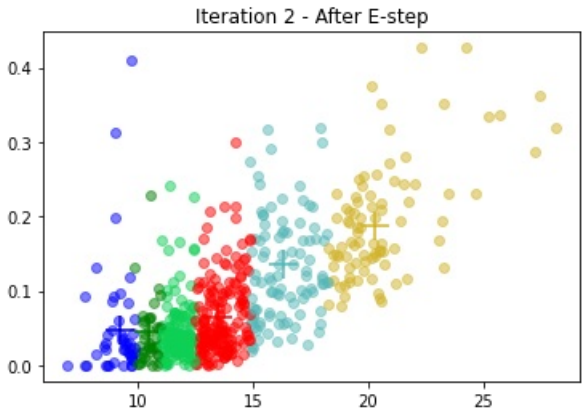


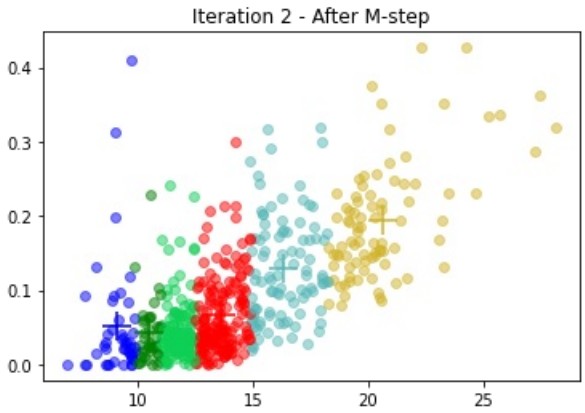


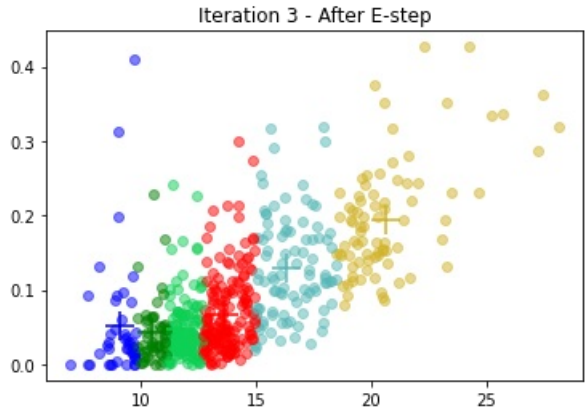
k-means with k=6 clusters for 3 iterations:

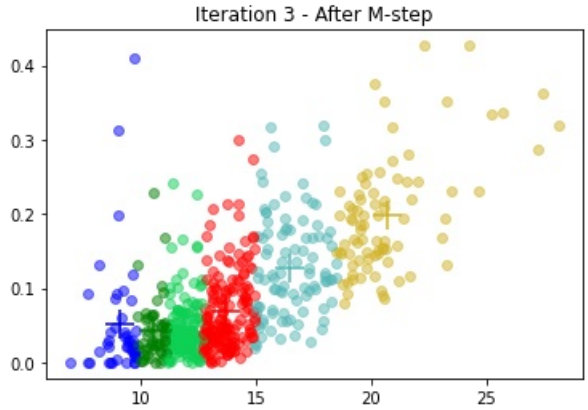

Iteration 1 - After E-step
0.4
0.3
0.2
0.1
0.0
10
15
20
25


Iteration 1 - After M-step
0.4
0.3
0.2
0.1
0.0
10
15
20
25

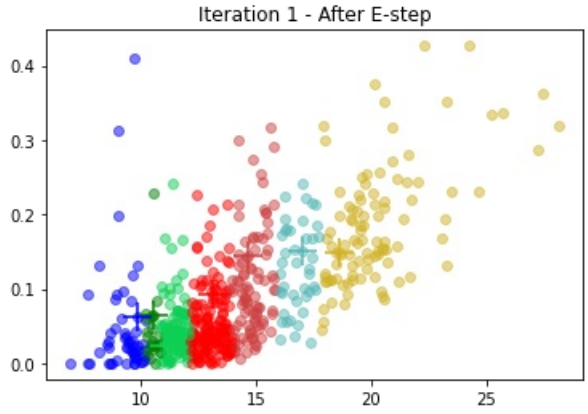

Iteration 2 - After E-step
0.4
0.3
0.2
0.1
0.0
10
15
20
25

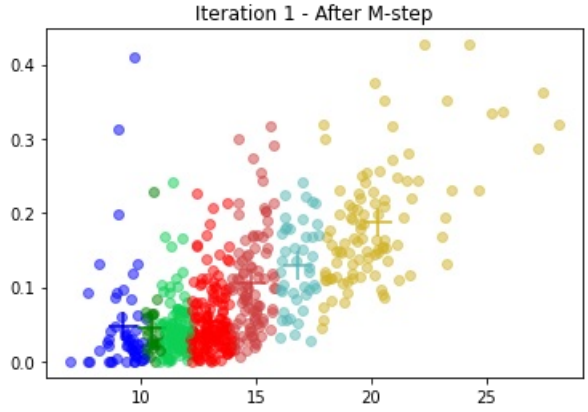

Iteration 2 - After M-step
0.4
0.3
0.2
0.1
0.0
10
15
20
25

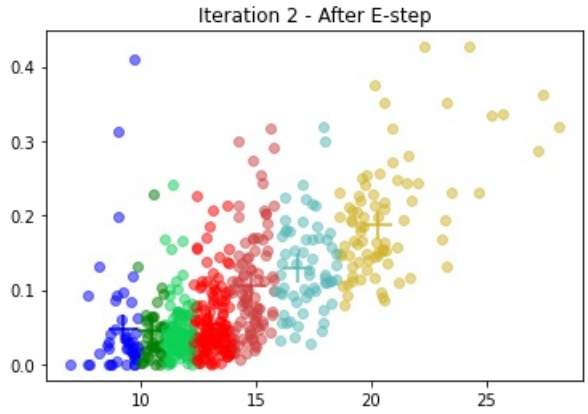


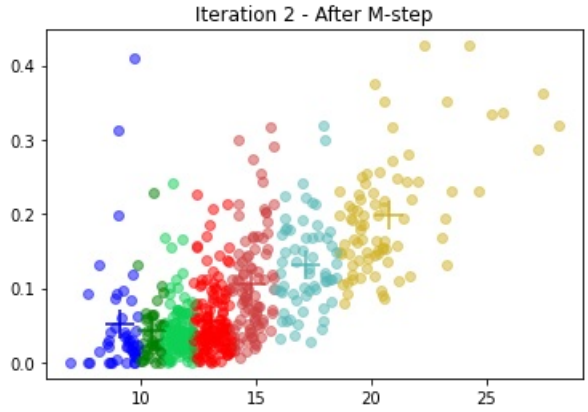


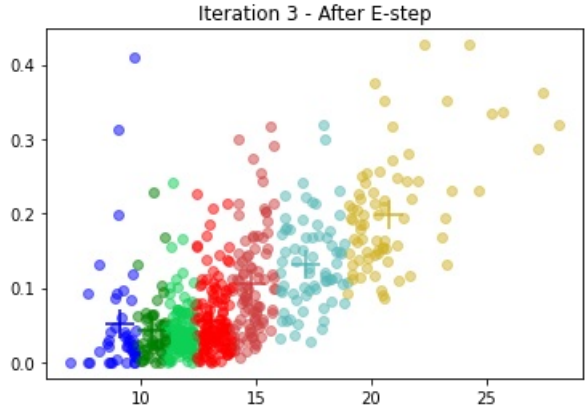
k-means with k=7 clusters for 3 iterations:

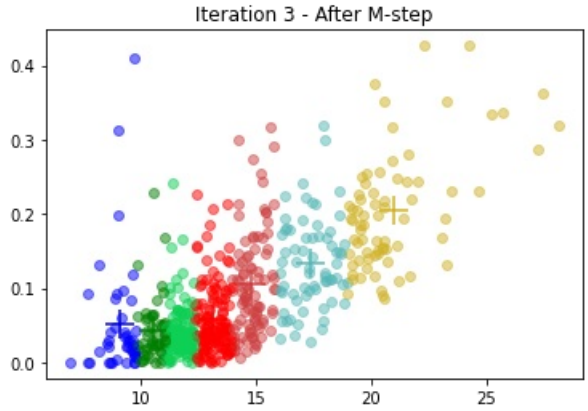




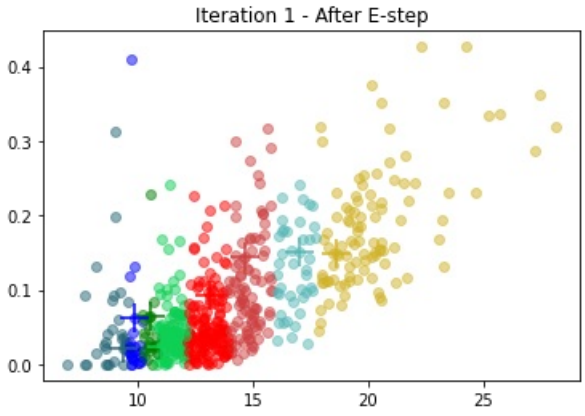


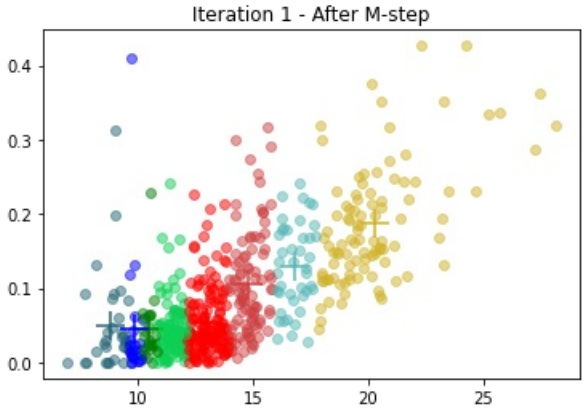


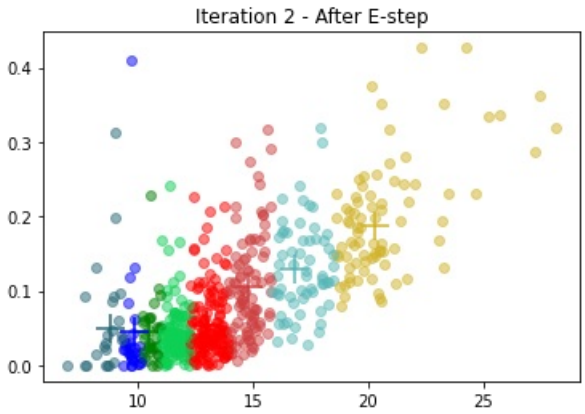


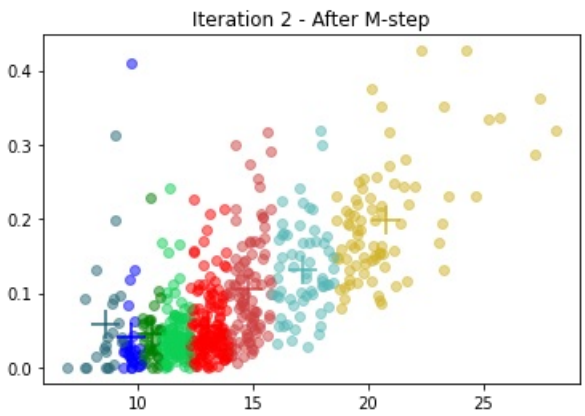


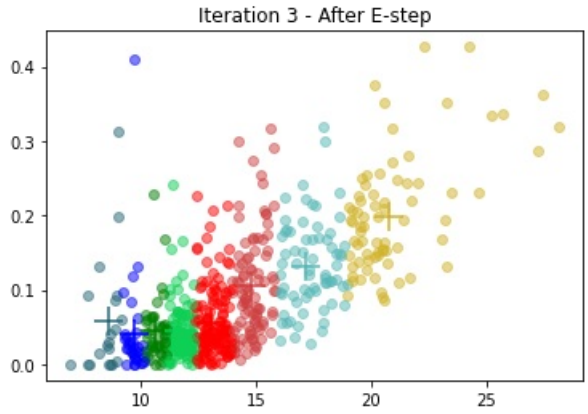
k-means with k=8 clusters for 3 iterations:

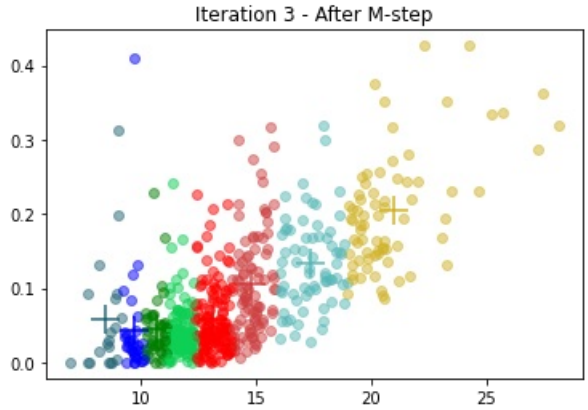

Iteration 1 - After E-step
0.4
0.3
0.2
0.1
0.0
10
15
20
25


Iteration 1 - After M-step
0.4
0.3
0.2
0.1
0.0
10
15
20
25

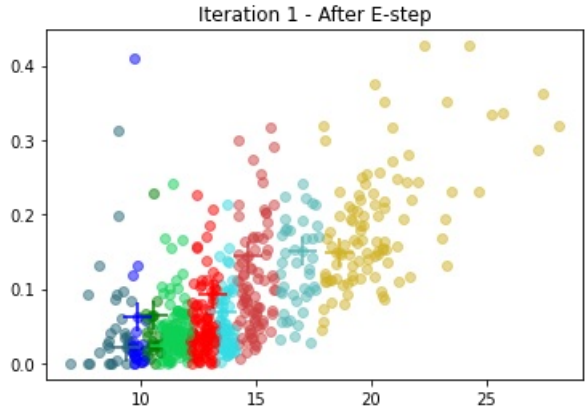

Iteration 2 - After E-step
0.4
0.3
0.2
0.1
0.0
10
15
20
25

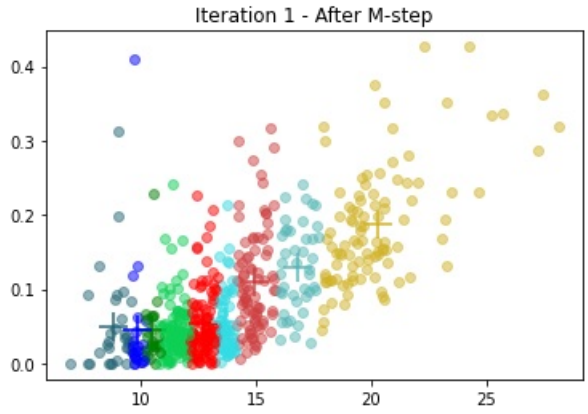

Iteration 2 - After M-step
0.4
0.3
0.2
0.1
0.0
10
15
20
25

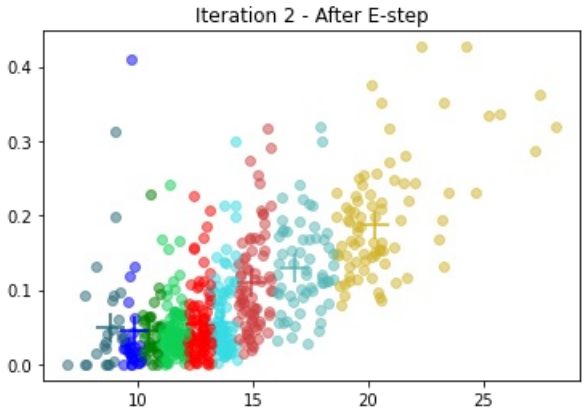


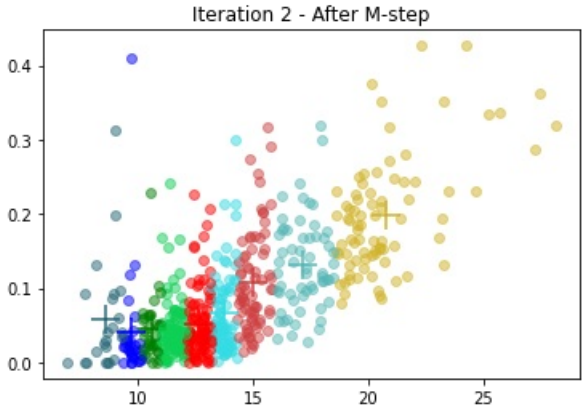


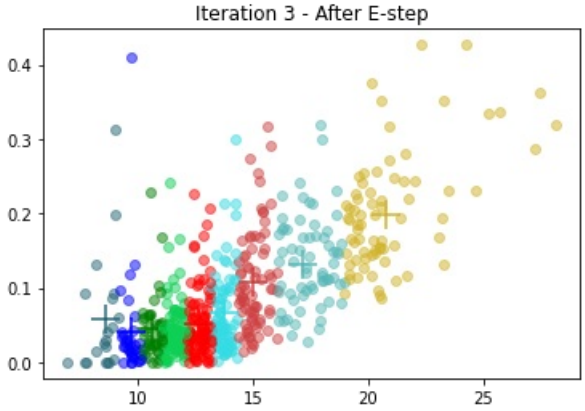
k-means with k=9 clusters for 3 iterations:

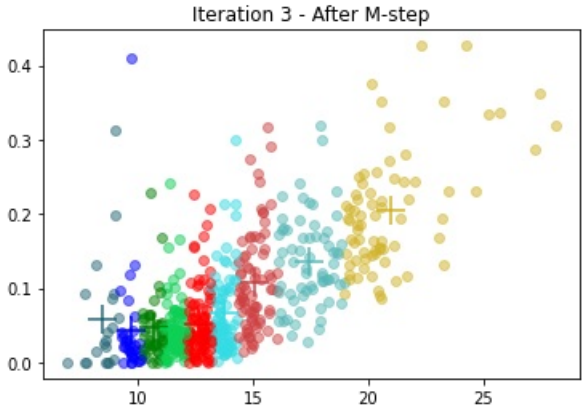












1.3.b. For each clustering create silhouette plots and print out the score.

In [ ]:

```
# I am not sure which score is referenced at. Also silhouett plot is not clear.
# While I guess some fort of execution of the following function is desired, I am unwilling to continue righ
t now..
# metrics.silhouette_score(list(zip(x_radii, y_concavity)), list(zip(x_radii_centers)), metric="euclidean",
sample_size=len(x_radii))
```

1.3.c. Which is the best choice for the number of clusters? Why?

First of all, note that the plots are heavily skewed, since `mean raduis` has much bigger values than `mean concavity` and the two axis have different scale!

1.4. Explain the difference between k-means and k-medoids.

While a **mean** corresponds to the geometric gravity, a **medoid** is a *data point* itsefl - which comes closest to the mean. Thus using medoids means only using real data and clustering the data arount existing data points.

Depending on the data, this may lead to better or worse results.

Exercise 2 - Gaussian misture models

2.1. Explain the EM-Algorithm in your own words, without using any formula.

An **Expectation Maximization (EM)** algorithm is an algorithm that makes an assumption/initialization/first guess about a solution, and then refines it. The two main steps, which are executed repetedly alternatingly, are:

- *E-step*: Estimate some solution, as expectation based on the currently gathered knowledge
- *M-step*: Based on the last estimation, refine/improve the solution with help of some measurement.

2.2. The complexity of the Gaussian mixture model can be controlled by restricting how the covariance matrices are allowed to vary. Assume your data has three features and you want to cluster it into 2 clusters.

In [ ]:

```
# Useful resource see: http://ethen8181.github.io/machine-learning/clustering/GMM/GMM.html
```

2.2.a. How many parameters (depending on the number of clusters) need to be estimated in the most general model (no restrictions on the covariances)?

Let D = 3 be the dimensions of the gaussians, k = 2 be the no. of gaussian mixtures or clusters.

The no. of parameters to be estimated are $3 * 3 * 2 + 3 * 2 + 2 = 26$ parameters

In the most general GMM with no restriction on co-variances, the no. of parameters which need to be estimated by the model are, $(D * D * k) + (D * k) + k$ Here $D * D * k$ term is due to k covariance matrices of size $D * D$, the term $D * k$ is due to k mean vectors of length D and the last term k is due to weight vector of length k.

An alternate formula to calculate no. of parameters to be estimated: $Df = (D * D - D)/2 + 2 * D + 1$ for each gaussian. Given you have k components, you have (k*Df)-1 parameters. Because the mixing weights must sum to 1, you only need to find k-1 of them. The kth weight can be calculated by subtracting the sum of the (k-1) weights from 1 so in our case $Df = (3$-3)/2 + 6 + 1, Df= 10andgiven2componentsthen : (2*10)-1 = 19$parameters

2.2.b. Assuming that there is no correlation between the variables for each Gaussian, how many parameters does this model need to estimate?

Since there is no correlation hence the non-diagonal elements of the covariance matrix will be zero. The no. of parameters to be estimated are $D * k + D * k + k = 6 + 6 + 2 = 14$ parameters

2.2.c. Assuming that there is neither correlation nor does the variation for each feature change. How many parameters does the model have to estimate now?

Since there is no correlation hence the non-diagonal elements of the covariance matrix will be zero. Also, the diagonal elements of the covariance matrix will also be constant as the variation between the feature does not change. The no. of parameters to be estimated are $k + k + k = 2 + 2 + 2 = 6$ parameters

2.3. Cluster the breast cancer dataset (on the entire dataset: `cancer_all.csv`) with the help of a Gaussian mixture model. Perform the clustering for a range of clusters between 2 and 10 and for all possible assumptions for the covariance matrices. Plot the BIC of each clustering.

In [ ]:

```
cancer_data = pd.read_csv('https://raw.githubusercontent.com/D34dP0oL/4216_Biomedical_DS_and_AI/main/Sheet5/cancer_all.csv', index_col = 'Unnamed: 0')
cancer_data
```

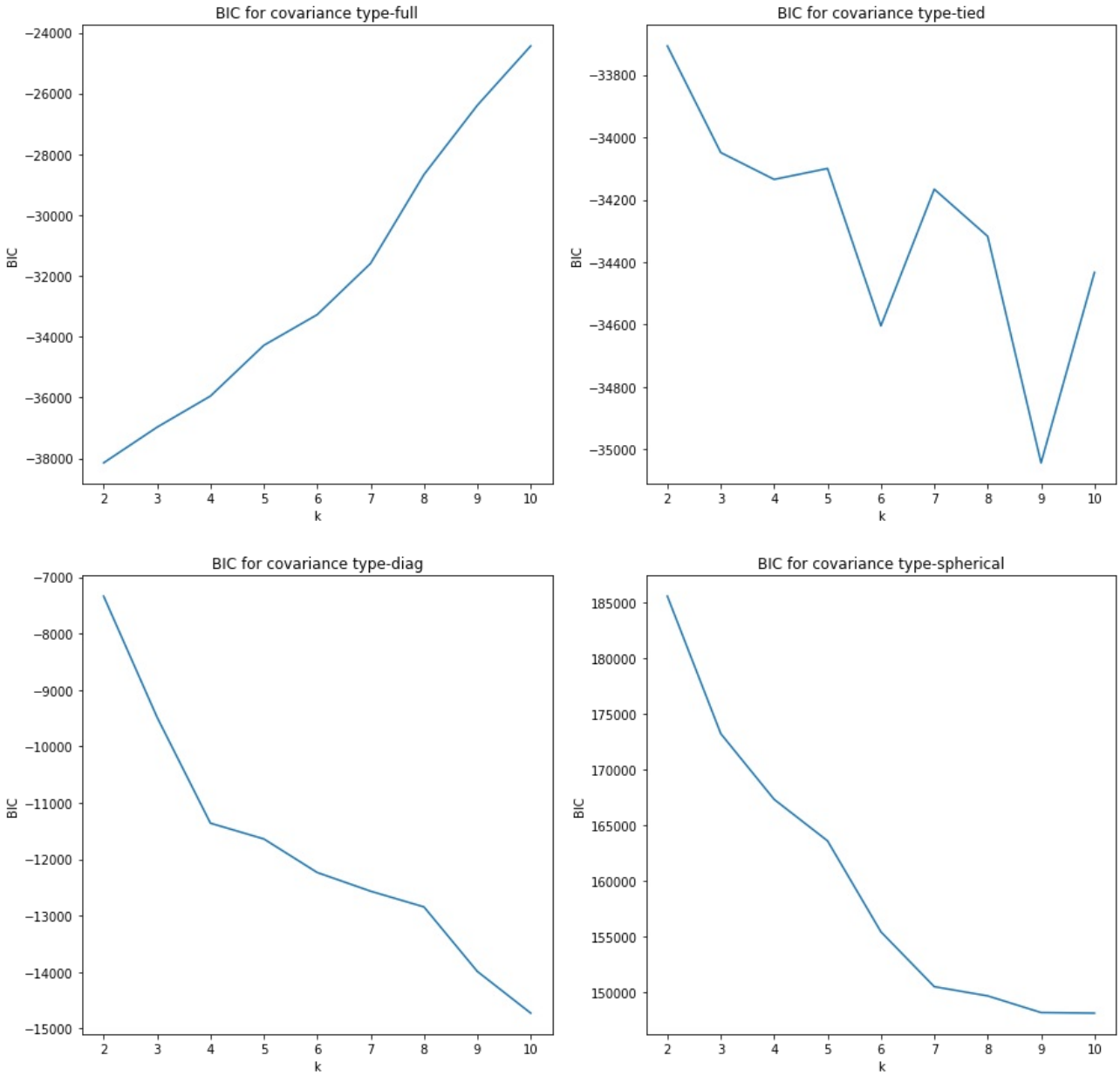
Out[ ]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	rad er
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	1.0
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	0.5
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	0.7
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	0.4
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	0.7
...	...	...	...	...	...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	1.1
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	0.7
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	0.4
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	0.7
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	0.3

569 rows × 30 columns

In [ ]:

```
bic_results_full = []
bic_results_tied = []
bic_results_diag = []
bic_results_spherical = []

for k in range(2, 11, 1):
    # create models for each covariance type
    gmm_model_full = GaussianMixture(n_components = k).fit(cancer_data)
    gmm_model_tied = GaussianMixture(n_components = k, covariance_type = 'tied').fit(cancer_data)
    gmm_model_diag = GaussianMixture(n_components = k, covariance_type = 'diag').fit(cancer_data)
    gmm_model_spherical = GaussianMixture(n_components = k, covariance_type = 'spherical').fit(cancer_data)

    # bic score for each model
    bic_results_full.append(gmm_model_full.bic(cancer_data))
    bic_results_tied.append(gmm_model_tied.bic(cancer_data))
    bic_results_diag.append(gmm_model_diag.bic(cancer_data))
    bic_results_spherical.append(gmm_model_spherical.bic(cancer_data))

fig, axs = plt.subplots(2,2, figsize=(15,15))
axs[0, 0].plot(range(2, 11, 1), bic_results_full)
axs[0, 0].set_title('BIC for covariance type-full')
axs[0, 0].set_xlabel('k')
axs[0, 0].set_ylabel('BIC')


axs[0, 1].plot(range(2, 11, 1), bic_results_tied)
axs[0, 1].set_title('BIC for covariance type-tied')
axs[0, 1].set_xlabel('k')
axs[0, 1].set_ylabel('BIC')


axs[1, 0].plot(range(2, 11, 1), bic_results_diag)
axs[1, 0].set_title('BIC for covariance type-diag')
axs[1, 0].set_xlabel('k')
axs[1, 0].set_ylabel('BIC')


axs[1, 1].plot(range(2, 11, 1), bic_results_spherical)
axs[1, 1].set_title('BIC for covariance type-spherical')
axs[1, 1].set_xlabel('k')
axs[1, 1].set_ylabel('BIC')
plt.show()
```



2.3.a. Which is the best choice for the clustering? Why?

BIC gives us an estimation on how much is good the GMM in terms of predicting the data we actually have. The lower is the BIC, the better is the model to actually predict the data we have. In order to avoid overfitting, this technique penalizes models with big number of clusters. So now in our case k=2 with covariance_type='full' is the best choice of number of cluster because it has the lowest BIC.

2.3.b. Plot the data (features `mean radius` and `mean compactness`), the cluster assignment and ellipses (to show the Gaussian component) for your selected model.

In [ ]:

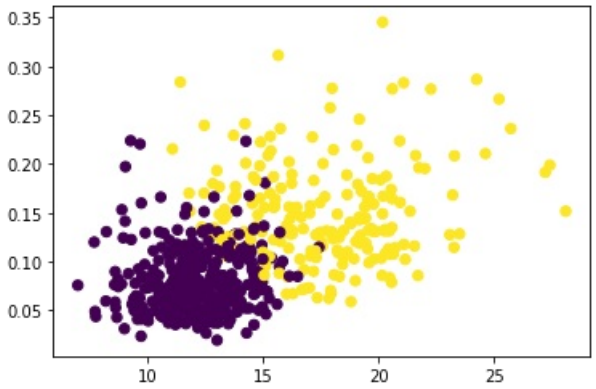
```
def draw_ellipse(position, covariance, ax=None, **kwargs):
    """Draw an ellipse with a given position and covariance"""
    ax = ax or plt.gca()

    # Convert covariance to principal axes
    if covariance.shape == (2, 2):
        U, s, Vt = np.linalg.svd(covariance)
        angle = np.degrees(np.arctan2(U[1, 0], U[0, 0]))
        width, height = 2 * np.sqrt(s)
    else:
        angle = 0
        width, height = 2 * np.sqrt(covariance)

    # Draw the Ellipse
    for nsig in range(1, 4):
        ax.add_patch(Ellipse(position, nsig * width, nsig * height, angle, **kwargs))
```

In [ ]:

```
gmm = GaussianMixture(n_components=2)
gmm.fit(cancer_data)
labels = gmm.predict(cancer_data)
ax = plt.gca()
ax.scatter(cancer_data.loc[:, 'mean radius'], cancer_data.loc[:, 'mean compactness'], c=labels, s=40, cmap='
viridis')
```

Out[ ]:

```
<matplotlib.collections.PathCollection at 0x7f8e5af2bd90>
```



2.4. How does the k-means model differ from the GMM model? Which model would you prefer for the given data and why?

K-Means Clustering:

It is an algorithm, which classifies samples based on attributes/features into K number of clusters. Clustering or grouping of samples is done by minimizing the distance between sample and the centroid. i.e. Assign the centroid and optimize the centroid based on the distances from the points to it. This is called as Hard Assignment i.e. We are certain that particular points belong to particular centroid and then based on the least squares distance method, we will optimize the place of the centroid.

Advantages of K-Means:

1. Running Time
2. Better for high dimensional data.
3. Easy to interpret and Implement.

Disadvantages of K-Means:

1. Assumes the clusters as spherical, so does not work efficiently with complex geometrical shaped data(Mostly Non-Linear)
2. Hard Assignment might lead to mis grouping.

Guassian Mixture:

Instead of Hard assgning data points to a cluster, if we are uncertain about the data points where they belong or to which group, we use this method. It uses probability of a sample to determine the feasibility of it belonging to a cluster.

Advantages:

1. Does not assume clusters to be of any geometry. Works well with non-linear geometric distributions as well.
2. Does not bias the cluster sizes to have specific structures as does by K-Means (Circular).

Disadvantages:

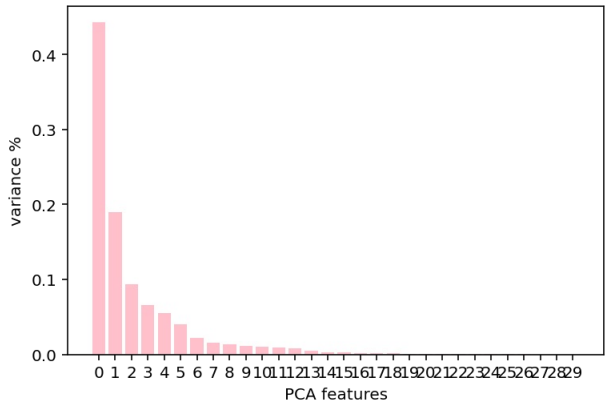
1. Uses all the components it has access to, so initialization of clusters will be difficult when dimensionality of data is high.
2. Difficult to interpret

Based on the above points, I think k-mean should perform better on this dataset as it has 30 dimensions so clustering will be difficult for GMM.

2.5. Generate the k-Means model for the entire dataset and visualise both k-Mean and GMM models using PCA.

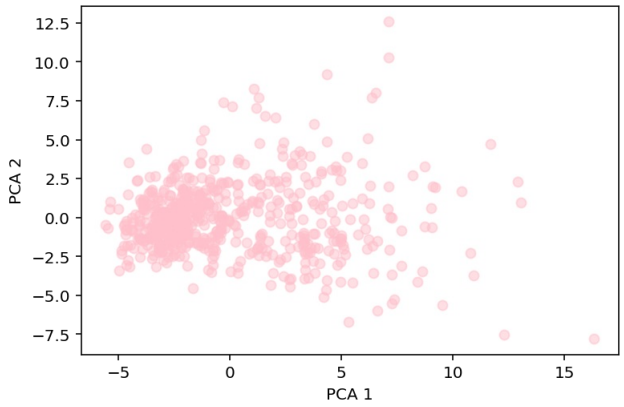
In [ ]:

```
%config InlineBackend.figure_format='retina'
# Standardize the data to have a mean of ~0 and a variance of 1
X_std = StandardScaler().fit_transform(cancer_data)
# Create a PCA instance: pca
pca = PCA()
principalComponents = pca.fit_transform(X_std)
# Plot the explained variances
features = range(pca.n_components_)
plt.bar(features, pca.explained_variance_ratio_, color='pink')
plt.xlabel('PCA features')
plt.ylabel('variance %')
plt.xticks(features)
# Save components to a DataFrame
PCA_components = pd.DataFrame(principalComponents)
```



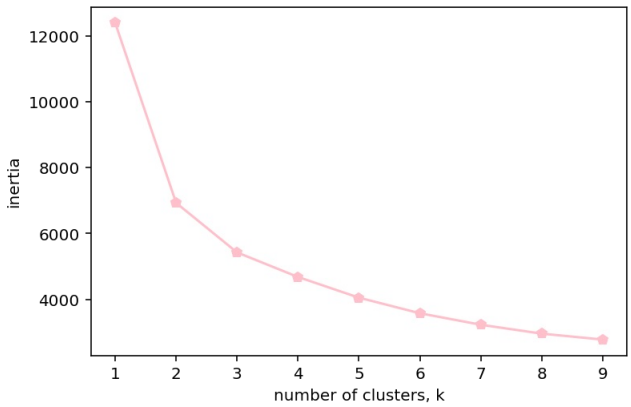
In [ ]:

```
# scatter plot of first two PCs
plt.scatter(PCA_components[0], PCA_components[1], alpha=.5, color='pink')
plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
plt.show()
```



In [ ]:

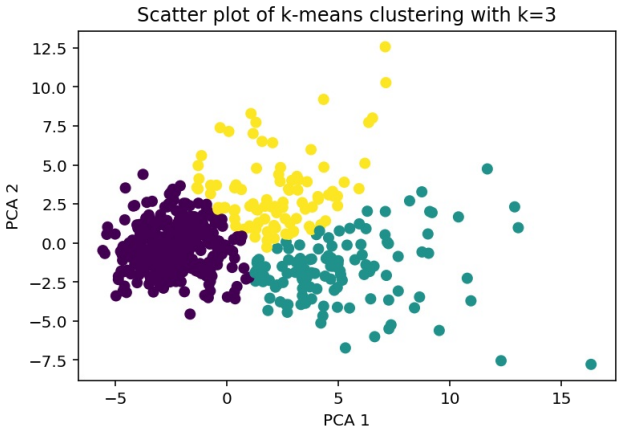
```
inertias = []

# Creating 10 K-Mean models while varying the number of clusters (k)
for k in range(1,10):
    model = KMeans(n_clusters=k)

    # Fit model to samples
    model.fit(PCA_components.iloc[:,:3])

    # Append the inertia to the list of inertias
    inertias.append(model.inertia_)

plt.plot(range(1,10), inertias, '-p', color='pink')
plt.xlabel('number of clusters, k')
plt.ylabel('inertia')
plt.show()
```



The figure shows that after 3 clusters the change in the value of inertia is no longer significant and most likely, neither is the variance of the rest of the data after the elbow point. Therefore we can discard everything after k=3 and proceed to the last step in the proces

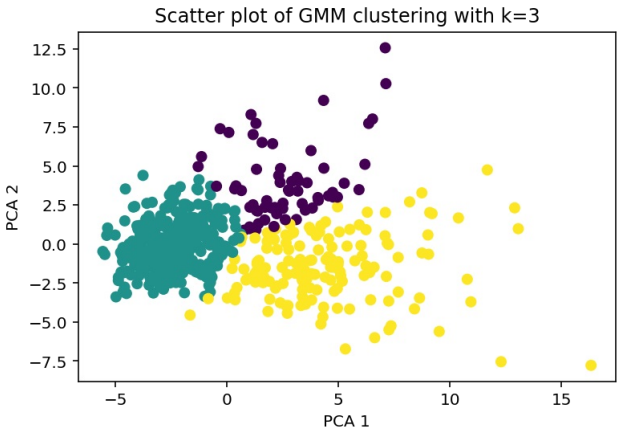
In [ ]:

```
model = KMeans(n_clusters=3)
model.fit(PCA_components.iloc[:,:2])

labels = model.predict(PCA_components.iloc[:,:2])
plt.scatter(PCA_components[0], PCA_components[1], c=labels)
plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
plt.title('Scatter plot of k-means clustering with k=3')
plt.show()
```



In [ ]:

```
model = GaussianMixture(n_components=3)
model.fit(PCA_components.iloc[:,:2])

labels = model.predict(PCA_components.iloc[:,:2])
plt.scatter(PCA_components[0], PCA_components[1], c=labels)
plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
plt.title('Scatter plot of GMM clustering with k=3')
plt.show()
```



2.6. What are the advantages of GMMs over k-means?

k-means only considers the mean to update the centroid while GMM takes into account the mean as well as the variance of the data. GMM is a lot more flexible in terms of cluster covariance k-means is actually a special case of GMM in which each cluster's covariance along all dimensions approaches 0. This implies that a point will get assigned only to the cluster closest to it. With GMM, each cluster can have unconstrained covariance structure. Think of rotated and/or elongated distribution of points in a cluster, instead of spherical as in kmeans. As a result, cluster assignment is much more flexible in GMM than in k-means. GMMs have soft boundaries because they perform probabilistic cluster assignments, while k-means have hard boundaries as they directly assign the point to a single cluster.GMMs represent the membership of each point in a cluster probabilistically.

Exercise 3 - Consensus clustering

Perform (k-means) consensus clustering of samples for the given gene expression data `allData.csv` . Take minimum clusters as 2, maximum clusters as 6, resampling proportion as 80% and number of iterations as 10. Find the following:

In [ ]:

```
gene_data = pd.read_csv('https://raw.githubusercontent.com/D34dP0oL/4216_Biomedical_DS_and_AI/main/Sheet5/allData.csv', index_col="Unnamed: 0")
gene_data = gene_data.fillna(0)
gene_data.head()
```

Out[ ]:

	36638_at	39318_at	38514_at	266_s_at	38585_at	41266_at	36108_at	39389_at	31525_s_at	32612_at
01005	0.583368	0.535258	0.642984	0.891901	0.269871	0.605566	0.069070	0.938101	0.314737	0.590410
01010	0.505321	0.704177	0.913612	0.657634	0.402911	0.429698	0.803187	0.360471	0.718665	0.773198
03002	0.375805	0.073716	0.707562	0.847162	0.792428	0.819212	0.644334	0.735292	0.828776	0.821131
04006	1.000000	0.226960	0.119596	0.394317	0.115411	0.050117	0.440698	0.649291	0.498758	0.692290
04007	0.890125	0.631314	0.518785	0.880312	1.000000	0.858408	0.638519	0.933899	0.963169	0.818224

5 rows × 5000 columns

3.1. Best number of clusters.

In [ ]:

```
sample_proportion = 0.8
MnC = 2
MxC = 6
H = 10
```

In [ ]:

```
def sample():
  sampled_indices = np.random.choice( range(gene_data.shape[0]), size=int(gene_data.shape[0]*sample_proporti
on), replace=False)
  return sampled_indices, gene_data.iloc[sampled_indices, :]
```

In [ ]:

```
Mk = np.zeros((MxC-MnC+1, gene_data.shape[0], gene_data.shape[0]))
Is = np.zeros((gene_data.shape[0],)*2)
for k in range(MnC, MxC):
  i = k-MnC
  for h in range(H):
    sampled_indices, sampled_data = sample()
    Mh = KMeans(n_clusters=k).fit_predict(sampled_data)
    id_clusts = np.argsort(Mh)
    sorted = Mh[id_clusts]
    for i in range(k):  # for each cluster
      ia = bisect.bisect_left(sorted, i)
      ib = bisect.bisect_right(sorted, i)
      is_ = id_clusts[ia:ib]
      ids = np.array(list(combinations(is_, 2))).T
    ids_2 = np.array(list(combinations(sampled_indices, 2))).T
    Is[ids_2[0], ids_2[1]] += 1
  Mk[i] /= Is+1e-8
  Mk[i] += Mk[i].T
  Mk[i, range(gene_data.shape[0]), range(gene_data.shape[0])] = 1
  Is.fill(0)  # reset counter
Ak = np.zeros(MxC-MnC+1)
for i, m in enumerate(Mk):
  hist, bins = np.histogram(m.ravel(), density=True)
  Ak[i] = np.sum(h*(b-a) for b, a, h in zip(bins[1:], bins[:-1], np.cumsum(hist)))

deltaK = np.array([(Ab-Aa)/Aa if i > 2 else Aa
                              for Ab, Aa, i in zip(Ak[1:], Ak[:-1], range(MnC, MxC))])
bestK = np.argmax(deltaK) + \
            MnC if deltaK.size > 0 else MnC
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:24: DeprecationWarning: Calling np
.sum(generator) is deprecated, and in the future will give a different result. Use np.sum(np.fr
omiter(generator)) or the python sum builtin instead.
```

In [ ]:

```
print(bestK)
```

```
2
```

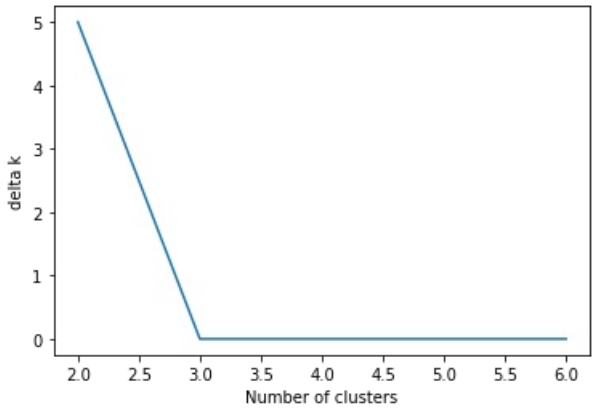
3.2. Change in area under CDF.

In [75]:

```
x = np.arange(2, 7, 1)
plt.plot(x, deltaK)
plt.xlabel("Number of clusters")
plt.ylabel("delta k")
```

Out[75]:

```
Text(0, 0.5, 'delta k')
```



3.3. Best cluster from the consensus matrix for each sample.