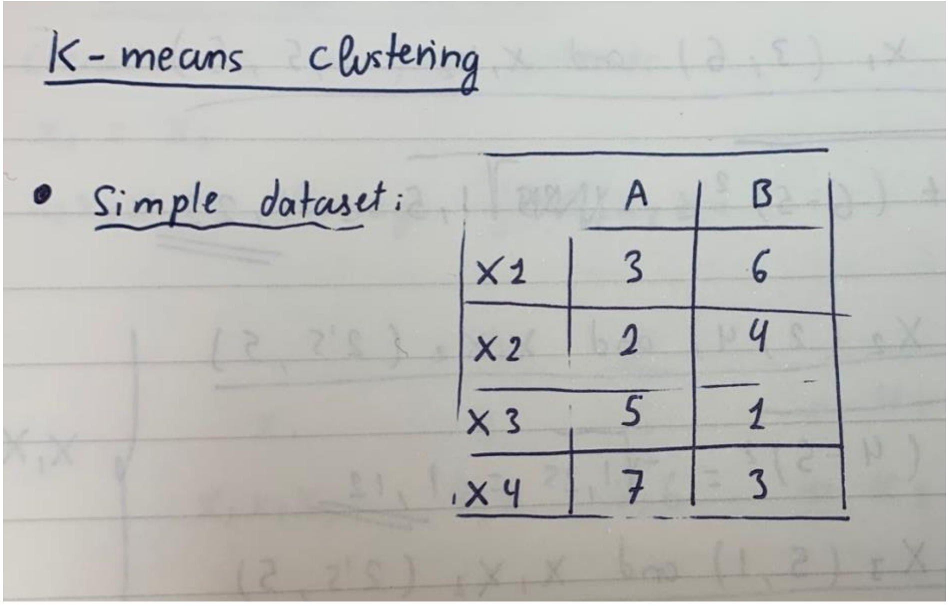
Assignment 2 – K-means clustering

Task 1: simple dataset + load iris dataset



K-means clustering

- Simple dataset:

	A	B
X1	3	6
X2	2	4
X3	5	1
X4	7	3

Iris dataset

```
# Import libraries
import pandas as pd
import numpy as np
%matplotlib inline

# Load iris dataset
iris = pd.read_csv("iris.csv")
iris
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

Task 2: apply k-means clustering on paper

K-means clustering

- Simple dataset:

	A	B
X1	3	6
X2	2	4
X3	5	1
X4	7	3

- Apply K-means clustering:

iteration 1 → choose 1st centroids

First centroids:

X_1X_2 = Average of X_1 and X_2 → K1

X_3X_4 = Average of X_3 and X_4 → K2

	A	B
X_1X_2	2,5	5
X_3X_4	6	2

- Calculate euclidean distances between all datapoints in the simple dataset and the centroids (X_1X_2, X_3X_4)

	X_1	X_2	X_3	X_4
X_1X_2	1,22	1,12	4,72	4,92
X_3X_4	5	4,47	1,41	1,41

K1 → X_1, X_2

K2 → X_3, X_4

(calculations are on the back of the page)

→ e distance for x_1 (3, 6) and x_1x_2 (2,5 , 5)

$$e = \sqrt{(3-2{,}5)^2 + (6-5)^2} = \sqrt{1{,}5} = \underline{\underline{1{,}22}}$$

→ e distance for X_2 (2, 4) and x_1x_2 (2'5 , 5)

$$e = \sqrt{(2-2'5)^2 + (4-5)^2} = \sqrt{1{,}25} = \underline{\underline{1{,}12}}$$

→ e distance for X_3 (5, 1) and x_1x_2 (2'5 , 5)

$$e = \sqrt{(5-2'5)^2 + (1-5)^2} = \sqrt{22{,}25} = \underline{\underline{4{,}72}}$$

→ e distance for x_4 (7, 3) and x_1x_2 (2'5 , 5)

$$e = \sqrt{(7-2'5)^2 + (3-5)^2} = \sqrt{24{,}25} = \underline{\underline{4{,}92}}$$

X_1X_2

→ e distance for x_1 (3, 6) and x_3x_4 (6, 2)

$$e = \sqrt{(3-6)^2 + (6-2)^2} = \sqrt{25} = 5 /\!/$$

→ e distance for x_2 (2, 4) and x_3x_4 (6, 2)

$$e = \sqrt{(2-6)^2 + (4-2)^2} = \sqrt{20} = \underline{\underline{4{,}47}}$$

→ e distance for x_3 (5, 1) and x_3x_4 (6, 2)

$$e = \sqrt{(5-6)^2 + (1-2)^2} = \sqrt{2} = 1{,}41 /\!/$$

Euclidean distance formula:

$$e(x, x') = \| x - x' \| = \sqrt{\sum_{i=1}^{d} (x_i - x'_i)^2}$$

→ e distance for x_4 (7, 3) and x_3x_4 (6, 2)

$$e = \sqrt{(7-6)^2 + (3-2)^2} = \sqrt{2} = 1{,}41 /\!/$$

X_3X_4

- choose new centroids:

iteration 2 → new centroids

$X_1 = X_1$

$X_2X_3X_4$ = Average of $X_2X_3X_4$

	A	B	
X_1	3	6	→ K_1
$X_2X_3X_4$	4,6	2,6	→ K_2

- table of euclidean distances for iteration 2

	X_1	X_2	X_3	X_4	
X_1	0	2,24	5,38	5	K_1 → X_1, X_2
$X_2X_3X_4$	3,76	2,95	1,65	2,43	K_2 → $X_3\ X_4$

- $\boxed{\text{Calculations}}$

→ e distance for X_1 (3,6) and X_1 (3,6)

$e = \sqrt{(3-3)^2 + (6-6)^2} = \sqrt{0} = 0$ //

→ e distance for X_2 (2,4) and X_1 (3,6)

$e = \sqrt{(2-3)^2 + (4-6)^2} = \sqrt{5} = 2,24$ //

→ e distance for X_3 (5,1) and X_1 (3,6)

$e = \sqrt{(5-3)^2 + (1-6)^2} = \sqrt{29} = 5,38$ //

→ e distance for X_4 (7,3) and X_1 (3,6)

$e = \sqrt{(7-3)^2 + (3-6)^2} = \sqrt{25} = 5$ //

X_1

→

2)

→ e distance for x_1 (3, 6) and $x_2x_3x_4$ (4,6, 2,6)

$$e = \sqrt{(3-4{,}6)^2 + (6-2{,}6)^2} = \sqrt{14{,}12} = 3{,}76$$

→ e distance for x_2 (2, 4) and $x_2x_3x_4$ (4,6, 2,6)

$$e = \sqrt{(2-4{,}6)^2 + (4-2{,}6)^2} = \sqrt{8{,}72} = 2{,}95$$

→ e distance for x_3 (5, 1) and $x_2x_3x_4$ (4,6, 2,6)

$$e = \sqrt{(5-4{,}6)^2 + (1-2{,}6)^2} = \sqrt{2{,}72} = 1{,}65$$

→ e distance for x_4 (7, 3) and $x_2x_3x_4$ (4,6, 2,6)

$$e = \sqrt{(7-4{,}6)^2 + (3-2{,}6)^2} = \sqrt{5{,}92} = 2{,}43$$

$x_2x_3x_4$

Clustering is an unsupervised machine learning method, and k-means is a clustering algorithm in which k (the number of clusters) is pre-defined and the centres for k are chosen through iteration. The first step is to choose centroids for the clusters using the column's mean. Then, the k-means algorithm is used to measure the distance between other points and the centroids, and each point is assigned to the cluster with the nearest centroid. There are different distance metrics that can be calculated (Manhattan, Euclidean, etc.). The Euclidean distance is the one normally used since it does not have dimension restrictions. The process is repeated using new centroids until the centres converge.

The simple dataset created consists of 4 datapoints x1 (3, 6), x2 (2, 4), x3 (5, 1) and x4 (7, 3). In this small dataset, only two iterations were necessary to reach conversion since the results obtained in both were the same.

For the first iteration, the centroids chosen were the mean of x1 and x2 (2,5, 5) and the mean of x3 and x4 (6, 2). After calculating the Euclidean distance between points, the results were that the points x1 and x2 were closer to the x1x2 centroid and the points x3 and x4 were closer to the x3x4 centroid, creating two clusters with two points in each.

After that, a second iteration was completed using two new centroids x1 (3, 6) and x2x3x4 (4,6, 2,6). The second iteration obtained the same results, reaching convergence. In this case, having more than two clusters would not make sense since there are only 4 datapoints. Thus, 2 clusters separating the points x1 and x2 from the points x3 and x4 seems to be the optimal result.

Now, a larger dataset (iris dataset) will be used to perform k-means clustering using Python.

Coursework 2: K-means clustering

Task 1: Prepare iris dataset

In [2]:
```python
# Import libraries
import pandas as pd
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
import random as rd
import warnings

# Upload the iris dataset using sklearn
import sys
sys.path.append('/usr/local/lib/python3.8/site-packages')

from sklearn.datasets import load_iris
iris = load_iris()
print(iris.DESCR)

import csv
with open('iris.csv', 'w', newline='') as csvfile:
    writer = csv.writer(csvfile, quoting=csv.QUOTE_NONNUMERIC)
    writer.writerow(iris.feature_names)
    writer.writerows(iris.data.tolist())
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken
from Fisher's paper. Note that it's the same as in R, but not as in the UCI
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field and
is referenced frequently to this day.  (See Duda & Hart, for example.)  The
data set contains 3 classes of 50 instances each, where each class refers to a
type of iris plant.  One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.

.. topic:: References

   - Fisher, R.A. "The use of multiple measurements in taxonomic problems"
     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
     Mathematical Statistics" (John Wiley, NY, 1950).
   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.
     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
     Structure and Classification Rule for Recognition in Partially Exposed
     Environments".  IEEE Transactions on Pattern Analysis and Machine
     Intelligence, Vol. PAMI-2, No. 1, 67-71.
   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions
     on Information Theory, May 1972, 431-433.
   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
     conceptual clustering system finds 3 classes in the data.
   - Many, many more ...
```

Task 3: Create a test harness

A test harness is a way to evaluate a machine learning algorithm in a dataset. In this case, we want to evaluate the implementation of the k-means algorithm in the *iris* dataset. The sklearn KMeans results obtained from the sklearn library will be later compared with the ones obtained in our implementation.

In [ ]:
```python
# test harness
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=3)
kmeans.fit(iris.data)

print(kmeans.labels_)
print(kmeans.cluster_centers_)
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 2 0 0 0 0 2 0 0 0 0
 0 0 2 2 0 0 0 0 2 0 2 0 2 0 0 2 2 0 0 0 0 0 2 0 0 0 0 2 0 0 0 2 0 0 0 2 0
 0 2]
[[6.85       3.07368421 5.74210526 2.07105263]
 [5.006      3.428      1.462      0.246     ]
 [5.9016129  2.7483871  4.39354839 1.43387097]]
```

Task 4: Implement k-means clustering in Python

The k centroids are chosen randomly at first, and then location of the centroids is optimised through iteration. The steps are as folows:

1. The distance between each centroid and all the points is calculated.
2. The points are assigned to the nearest centroid (cluster).
3. The centroids values change by calculating the averages of all the points belonging to them. And this process is repeated until the centroids no longer change their value.

In [4]:
```python
# Upload the csv
iris = pd.read_csv("iris.csv")
iris
```

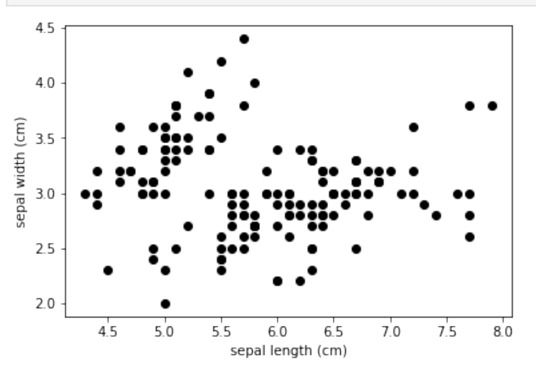
Out[4]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

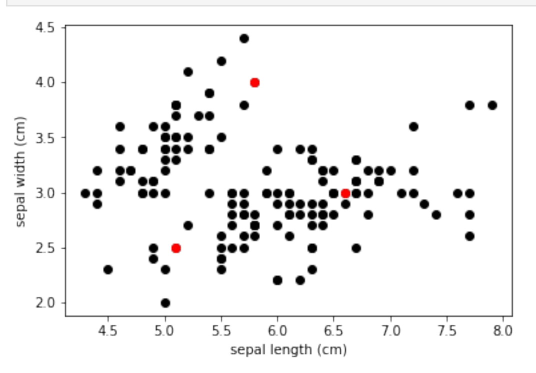
In [ ]:
```python
# First we visualise the datapoints in a scatterplot to get an idea of how many clusters we need
# We use 2 features instead of 4 so all the steps can be visualised (the same can be repeated for the other tw

#Visualise data points
plt.scatter(iris["sepal length (cm)"],iris["sepal width (cm)"],c='black')
plt.xlabel('sepal length (cm)')
plt.ylabel('sepal width (cm)')
plt.show()
```



In [ ]:
```python
# Now we can use K-means to form clusters

# First we select random observation as the centroids
X = iris[["sepal length (cm)","sepal width (cm)"]] # We define X to include the sepal data only for simplifica
K = 3 # From the scatterplot we can infer that it would make sense to have 3 clusters
Centroids = (iris.sample(n=K))
plt.scatter(X["sepal length (cm)"],X["sepal width (cm)"],c='black')
plt.scatter(Centroids["sepal length (cm)"],Centroids["sepal width (cm)"],c='red')
plt.xlabel('sepal length (cm)')
plt.ylabel('sepal width (cm)')
plt.show()
```



The datapoints and the randomised first centroids are visualised to get a sense of how well the clustering algorithm has worked (it can be compared to the visalisation in Task 5).

In [ ]:
```python
# Now we calculate the Euclidean distance between points

# While loop

difference = 1 # values difference
j=0 # stop point

while(difference!=0):
    XD=X
    i=1
    for index1,row_c in Centroids.iterrows():
        ED=[]
        for index2,row_d in XD.iterrows():
            d1=(row_c["sepal length (cm)"]-row_d["sepal length (cm)"])**2
            d2=(row_c["sepal width (cm)"]-row_d["sepal width (cm)"])**2
            d=np.sqrt(d1+d2)
            ED.append(d)
        X[i]=ED
        i=i+1

    C=[]
    for index,row in X.iterrows():
        min_dist=row[1]
        pos=1
        for i in range(K):
            if row[i+1] < min_dist:
                min_dist = row[i+1]
                pos=i+1
        C.append(pos)
    X["Cluster"]=C
    Centroids_new = X.groupby(["Cluster"]).mean()[["sepal width (cm)","sepal length (cm)"]]
    if j == 0:
        diff=1
        j=j+1
    else:
        difference = (Centroids_new['sepal width (cm)'] - Centroids['sepal width (cm)']).sum()
        + (Centroids_new['sepal length (cm)'] - Centroids['sepal length (cm)']).sum()
        print(difference.sum())
    Centroids = X.groupby(["Cluster"]).mean()[["sepal width (cm)","sepal length (cm)"]]

print(Centroids) # Location of the optimised centroids
```

```
-0.0900258738995916
0.04160002022872922
0.01429020260022762
0.03131572840769303
0.0014943866005641127
-0.015627902589109066
-0.007054822470525579
-0.01792310472576153
-0.007610998422357795
-0.013567683576757084
-0.0033263228000071088
0.0005594586082384723
0.003713675664895
0.006828391734052808
-0.004010912218459062
0.013758928825521544
-0.008405797101449952
0.0
         sepal width (cm)  sepal length (cm)
Cluster
1                3.409804           5.003922
2                2.700000           5.800000
3                3.078261           6.823913
```

The results obtained for the location of the centroids are not too different from the ones obtained using the test harness, which is a good indication.

The clustering algorithm starts by setting the difference to 1, and the stop point (*j*) to 0. The iterations will be repeated until the difference between the previous location of the centroids and the current location is 0.

After that, the code uses a while loop, several for loops and the Euclidean distance formula to calculate the distance between the points in sepal lenght and sepal width and assign a centroid. This process is repeated until *j* is reached.

Finally, and if else statement is used to assign the points to the optimised values of the centroids found after the iterations and form the final clusters.

Execution time

The execution time for K-means clustering can be calculated using the Lloyd's algorithm, which goes as follows:

O(n *K* I * d)

n : number of points

K : number of clusters

I : number of iterations

d : number of attributes

In this case, we could estimate that the execution time will be O = 300 *3* 10 * 2 = 18000

Assumptions and limitations

The main limitation could be that we have manually selected 3 clusters, without testing if that is the optimal number of clusters.

Another limitation is that the k-means implementation has not been generalised, and for that reason it would strugle handeling data that has varying sizes and density.

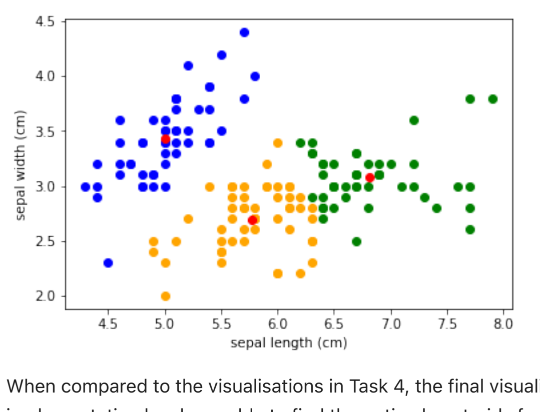
Moreover, the data has not been cleaned and outliers have the potential of dragging and misplacing clusters.

An assumption can be that, since the algorithm is randomised, it is possible to get different results for the same data using the same code.

Finally, in order to simplify and better visualise the progress of the k-means implementation we have only used two dimensional data, excluding two of the features. And the same process can be used for the other two features. PCA could be used to reduce dimensionality and solve this problem, although k-means can handle data with more than 2 dimensions per se.

Optional Task 5: Add a visualisation

In [ ]:
```python
# Visualisation of the clusters
color=['blue','orange','green']
for k in range(K):
      data=X[X["Cluster"]==k+1]
      plt.scatter(data["sepal length (cm)"],data["sepal width (cm)"],c=color[k])
plt.scatter(Centroids["sepal length (cm)"],Centroids["sepal width (cm)"],c='red')
plt.xlabel("sepal length (cm)")
plt.ylabel("sepal width (cm)")
plt.show()
```



When compared to the visualisations in Task 4, the final visualisation of the clusters helps understand how the k-means implementation has been able to find the optimal centroids for the 3 clusters and separate the data.