Programming for Artificial Intelligence and Data Science

Coursework 2

K-Means Clustering

Table of Content :

Introduction

A. Libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

#Iris Dataset
```

```
import sys
sys.path.append('/usr/local/lib/python3.8/site-packages')
from sklearn.datasets import load_iris
import csv

#Used as Sanity Check for task 2.
from sklearn.cluster import KMeans

#Used to compute the time calculation for task 4.
import time

#Used in task 4 for the K_Mean class created
from collections import defaultdict
```

B. Datasets and Variables

NBA Dataset

I decided to use NBA data sets because it is the greatest sport for gathering data, everything is recorded and accessible and I'm really passionate about the NBA and especially the analysis of its data. . Moreover, I think that my project is relevant because it fulfils the requested exercise. Furthermore, the selected data is perfectly suited for the realisation of the K-Mean Clustering mechanism.

In this project, I decided to study and classify the 30 highest volume 3-point shooters in the NBA across different periods. The NBA since its inception has been in constant evolution and adjustment. The trend is governed by the players that constitute the league. In the early days of the NBA the 3-point line did not exist, it was only in 1979 that it was introduced. The 3-point shot, which provides more points, progressively became part of the players' habits until it became an indispensable and omnipresent characteristic in the current NBA.

The project, therefore, focuses on the evolution and classification of three-point shooters in the NBA during three distinct periods: 1980 (creation of the 3-point line), 2000 and 2020.

For this purpose, the data that has been extracted comes from the well known website https://www.basketball-reference.com/ which is a reference in terms of NBA statistics.

The 10 players for each period who made the most 3-point shots (3PA) were selected and then sorted alphabetically in order to understand the mechanism of K-Mean Clustering without having the players previously sorted by period or by 3-point volume.

Here is the link to the (10) shooters with the highest volume at 3 points (3PA).

for the year 1980: https://www.basketball-reference.com/leagues/NBA_1980_per_game.html

for the year 2000: https://www.basketball-reference.com/leagues/NBA_2000_per_game.html

for the year 2020: https://www.basketball-reference.com/leagues/NBA_2020_per_game.html

Using the K-Mean Clustering mechanism, the objective is to successfully reclassify players in their respective periods according to their volume of attempted shots. Logically, the volume of 3-point shots has been increasing over time, so there will be a noticeable difference between the 3-point shooters of 1980, 2000 and 2020. It is for this reason that it seems to be interesting to apply the K-Mean Clustering method to this dataset whose players' names have been sorted in alphabetical order to exploit the K-Mean Clustering mechanism at best.

The file **NBA3PA.csv** has information about the per game: 3-Point field goals (3P), 3-Point field goal attempts (3PA) and 3-Point field goal percentage (3P%). There is 30 players listed in alphabetical order over different NBA eras that fits into three clusters. The dataset contains the following fields:

In [2]:
```
three_point= pd.read_csv("NBA3PA.csv", index_col="Player")
three_point
```

Out[2]:

Player	Pos	Team	3P	3PA▼	3P%
Bradley Beal	SG	WAS	3.0	8.4	0.353
Brent Barry	SG	SEA	2.1	5.0	0.411
Brian Taylor	PG	SDC	1.2	3.1	0.377
Buddy Hield	SG	SAC	3.8	9.6	0.394
Chris Ford	SG	BOS	1.0	2.2	0.427
Damian Lillard	PG	POR	4.1	10.2	0.401
Darrell Armstrong	PG	ORL	1.7	4.9	0.340
Dāvis Bertāns	PF	WAS	3.7	8.7	0.424
Devonte' Graham	PG	CHO	3.5	9.3	0.373
Duncan Robinson	SG	MIA	3.7	8.3	0.446
Eddie Jones	SG	CHH	1.8	4.7	0.375
Freeman Williams	SG	SDC	0.5	1.6	0.328
Gary Payton*	PG	SEA	2.2	6.3	0.340
George Gervin*	SG	SAS	0.4	1.3	0.314
James Harden	SG	HOU	4.4	12.4	0.355
Jason Williams	PG	SAC	1.8	6.2	0.287
Joe Hassett	SG	IND	0.9	2.7	0.348
John Roche	PG	DEN	0.6	1.6	0.380
Kemba Walker	PG	BOS	3.2	8.4	0.381
Larry Bird*	PF	BOS	0.7	1.7	0.406
Lindsey Hunter	PG	DET	2.0	4.7	0.432
Luka Dončić	PG	DAL	2.8	8.9	0.316
Micheal Ray Richardson	SG	NYK	0.3	1.3	0.245
Mike Newlin	SG	NJN	0.6	1.9	0.296
Nick Anderson	SG	SAC	1.8	5.5	0.332
Nick Van Exel	PG	DEN	1.7	5.1	0.332
Ray Allen*	SG	MIL	2.1	5.0	0.423
Reggie Miller*	SG	IND	2.0	5.0	0.408
Rick Barry*	SF	HOU	1.0	3.1	0.330
Trae Young	PG	ATL	3.4	9.5	0.361

Simple Dataset

Performing the K-Mean Clustering using NBA statistics was a special concern for me. However, in order to respect the guidelines I also performed the K-Mean Clustering method using a small random set of values without real interpretation allowing a much

more simplified approach of the problem. This dataset emphasize the K-Mean Clustering mechanism, the data set is composed of 12 values that fits into four clusters.

In [3]:
```
values = np.array([[1, 2],
                   [2, 2],
                   [1, 3],
                   [8, 2],
                   [9, 3],
                   [8, 3],
                   [1, 7],
                   [1, 9],
                   [2, 8],
                   [3, 9],
                   [4, 4],
                   [5, 5]])
values
```

Out[3]:
```
array([[1, 2],
       [2, 2],
       [1, 3],
       [8, 2],
       [9, 3],
       [8, 3],
       [1, 7],
       [1, 9],
       [2, 8],
       [3, 9],
       [4, 4],
       [5, 5]])
```

Iris Dataset

The iris data sets consists of 3 different types of irises' (Setosa, Versicolour, and Virginica) petal and sepal length, stored in a 150x4 numpy.ndarray that fits into three clusters. To read more about the iris dataset please remove the # in front of the line print(iris.DESCR)

In [4]:
```
iris = load_iris()
#print(iris.DESCR)

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())
```

C. Glossary

Pos = Position

- PG = Point Guard
- SG = Shooting Guard
- SF = Small Forward
- PF = Power Forward

Team = Team

3P = 3-Point Field Goals Per Game

3PA = 3-Point Field Goal Attempts Per Game

3P% = 3-Point Field Goal Percentage

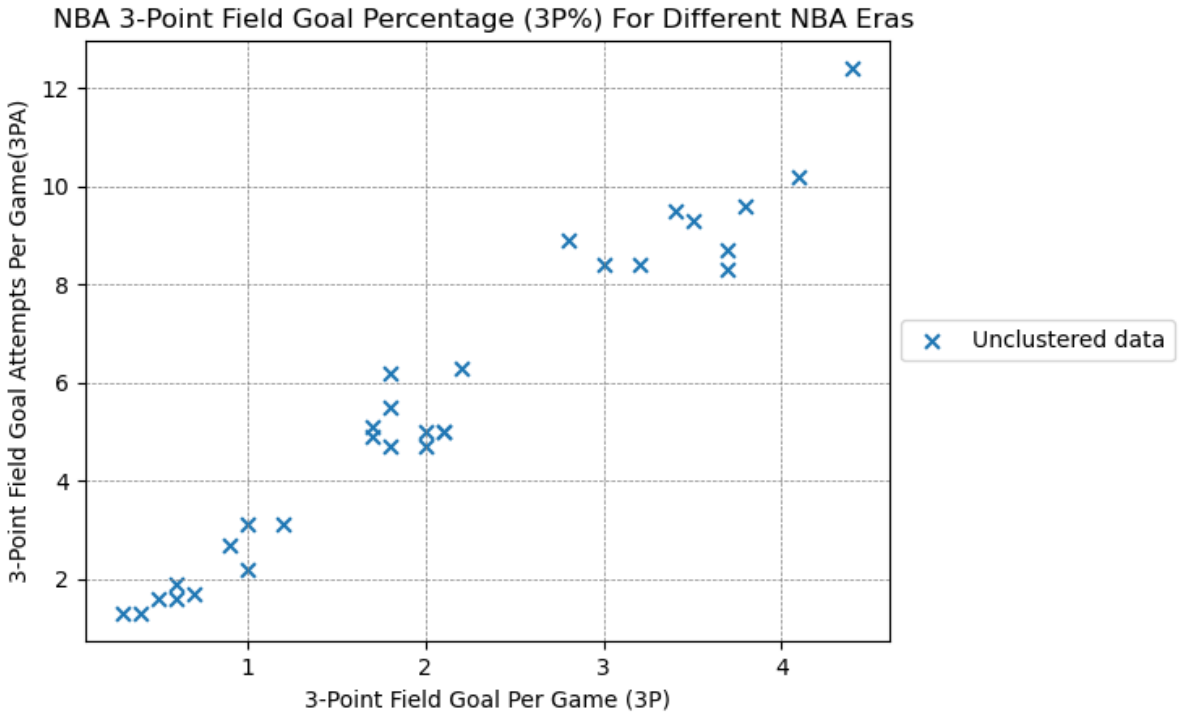
Task 1: Prepare two test datasets

A. NBA Dataset

In [5]:
```
three_point_clustering = three_point[['3P', '3PA▼']].to_numpy()
three_point_clustering
```

Out[5]:
```
array([[ 3. ,  8.4],
       [ 2.1,  5. ],
       [ 1.2,  3.1],
       [ 3.8,  9.6],
       [ 1. ,  2.2],
       [ 4.1, 10.2],
       [ 1.7,  4.9],
       [ 3.7,  8.7],
       [ 3.5,  9.3],
       [ 3.7,  8.3],
       [ 1.8,  4.7],
       [ 0.5,  1.6],
       [ 2.2,  6.3],
       [ 0.4,  1.3],
       [ 4.4, 12.4],
       [ 1.8,  6.2],
       [ 0.9,  2.7],
       [ 0.6,  1.6],
       [ 3.2,  8.4],
       [ 0.7,  1.7],
       [ 2. ,  4.7],
       [ 2.8,  8.9],
       [ 0.3,  1.3],
       [ 0.6,  1.9],
       [ 1.8,  5.5],
       [ 1.7,  5.1],
       [ 2.1,  5. ],
       [ 2. ,  5. ],
       [ 1. ,  3.1],
       [ 3.4,  9.5]])
```

In [6]:
```
plt.scatter(three_point_clustering[:,0], three_point_clustering[:,1],
            s = 40, marker = 'x', label = 'Unclustered data')
plt.grid(color = 'grey', linestyle = '--', linewidth = 0.5)
plt.legend(loc = 'center left', bbox_to_anchor = (1, 0.5))
plt.xlabel('3-Point Field Goal Per Game (3P)')
plt.ylabel('3-Point Field Goal Attempts Per Game(3PA)')
plt.title('NBA 3-Point Field Goal Percentage (3P%) For Different NBA Eras ')
plt.show()
```

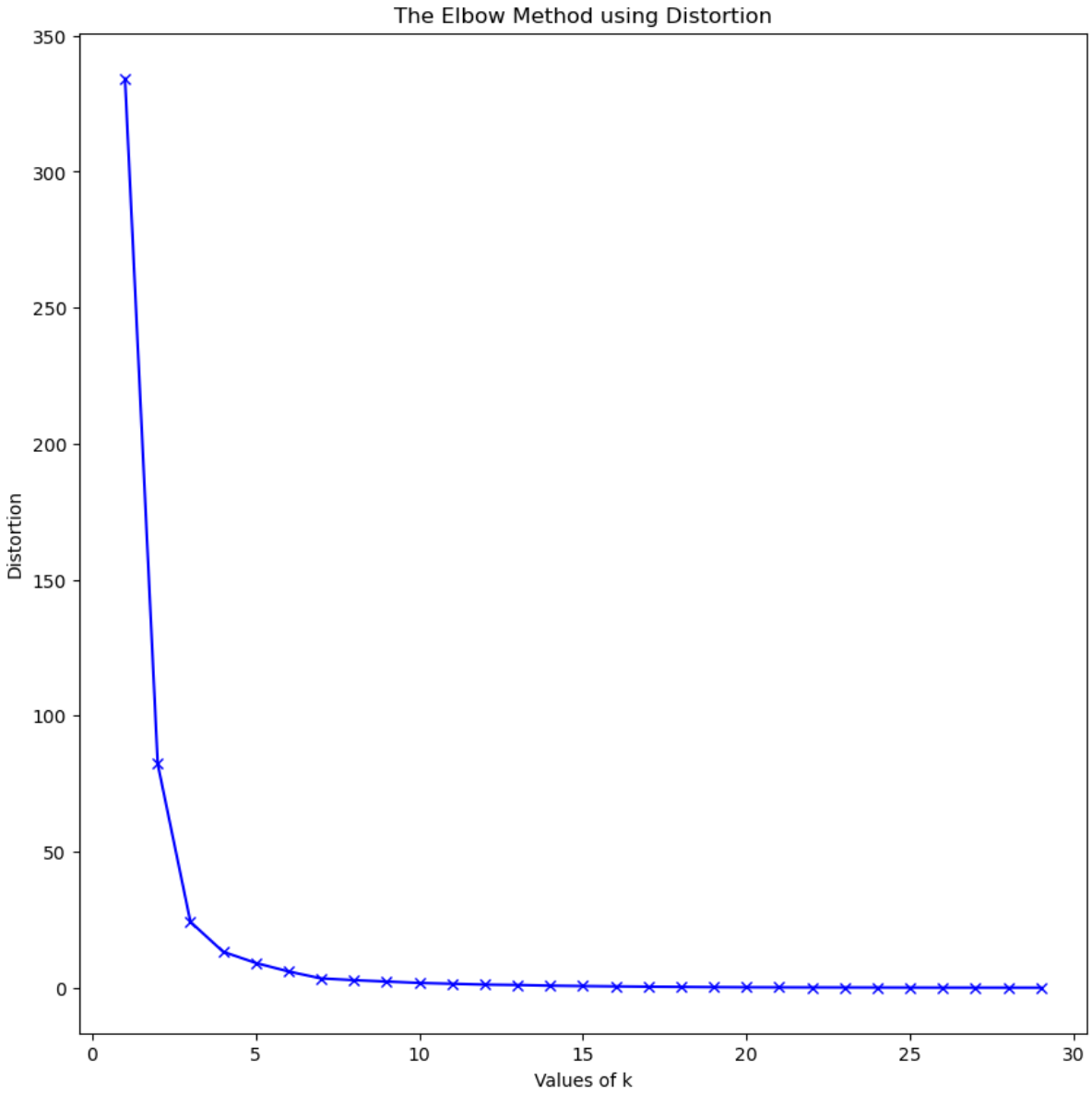


From the visualization above, we can see that the optimal number of clusters should be about **3**. But the visualisation of the data alone may not always give the right answer.

The elbow method is a method for determining the number of clusters, which consists of calculating the variance of the different volumes of clusters envisaged, and then placing the variances obtained on a graph. The result is an elbow-shaped visualisation in which the optimal number of clusters is the point representing the tip of the elbow, i.e. the point corresponding to the number of clusters from which the variance does not decrease significantly.

In [7]:
```
def RangeK(clustering):
    if len(clustering) < 35:
        return len(clustering)
    else:
        return 35

def ElbowMethod(clustering):
    distortions = []
    K = range(1, RangeK(clustering))
    for k in K:
        kmeans2 = KMeans(n_clusters = k)
        kmeans2.fit(clustering)
        distortions.append(kmeans2.inertia_)

    plt.figure(figsize=(10,10))
    plt.plot(K, distortions, 'bx-')
    plt.xlabel('Values of k')
    plt.ylabel('Distortion')
    plt.title('The Elbow Method using Distortion')
    plt.show()
```

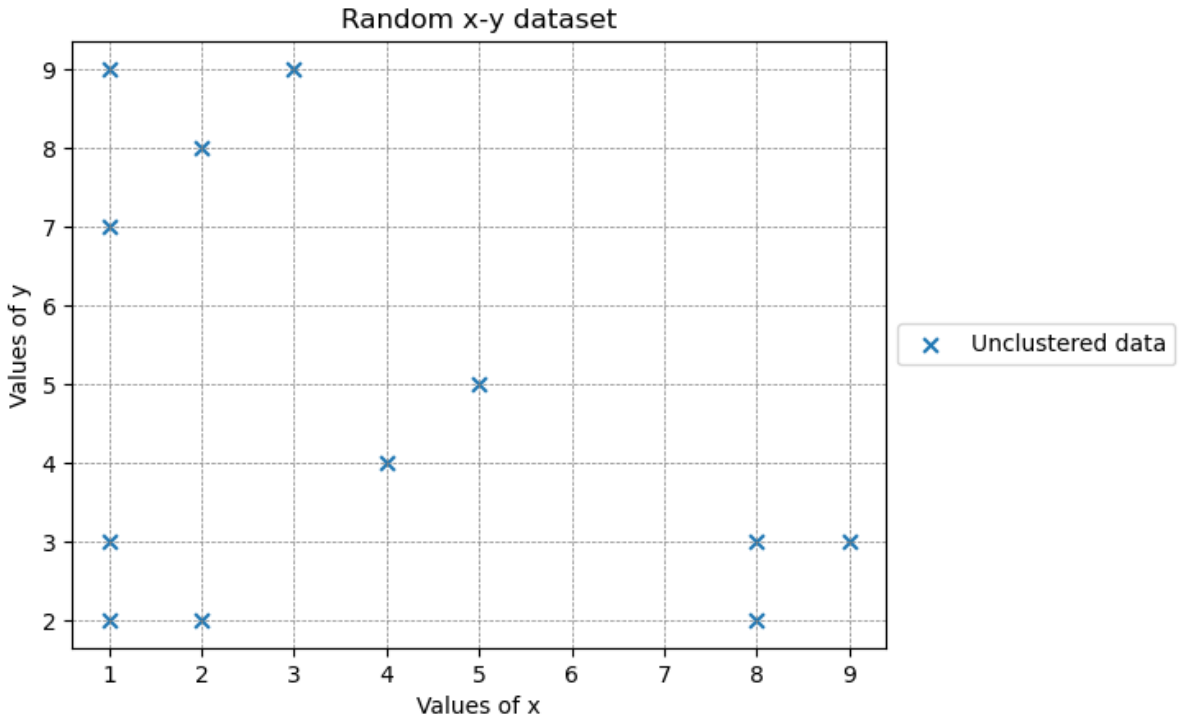
In [8]:
```
ElbowMethod(three_point_clustering)
```



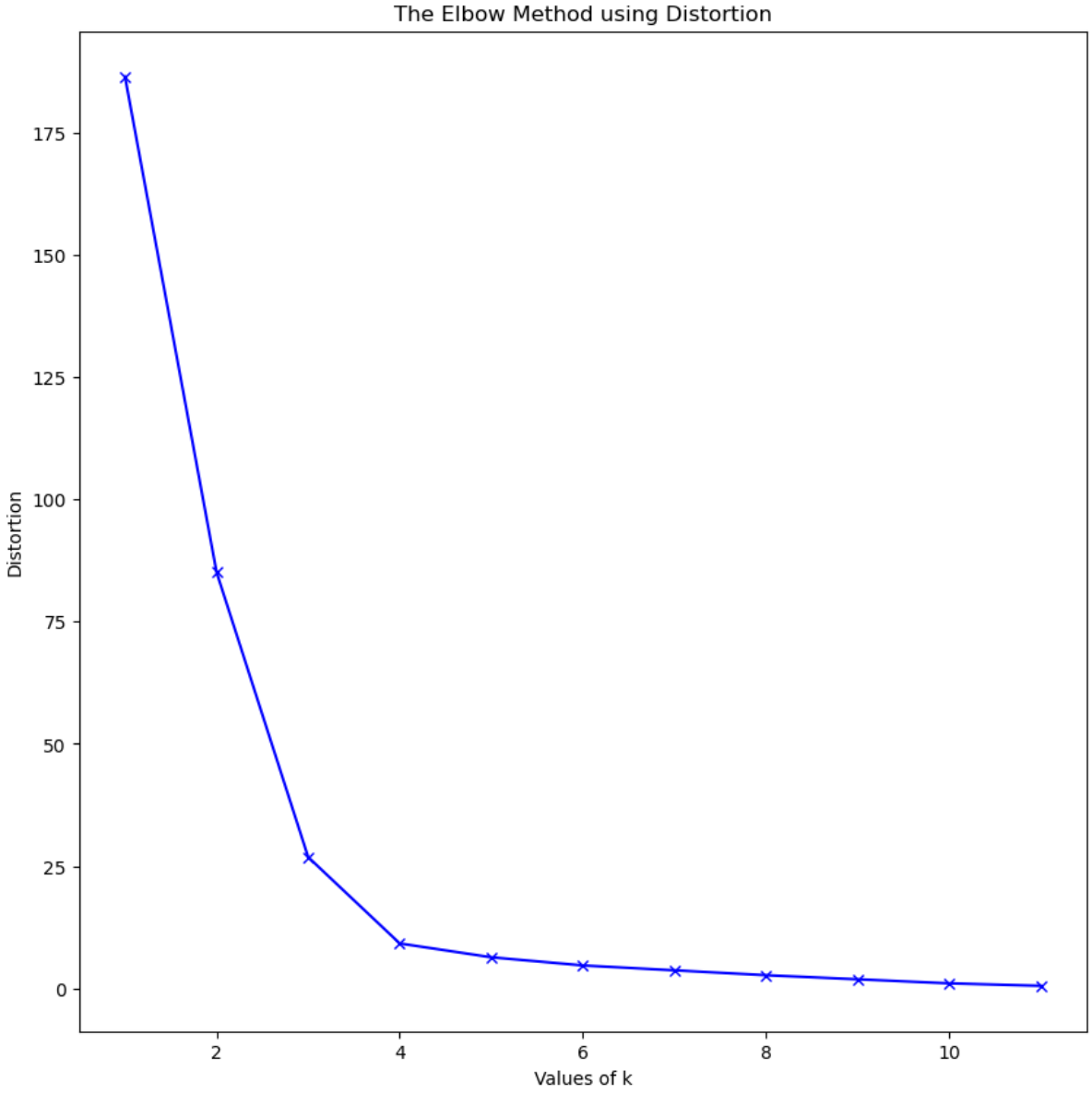
The ideal value of k for the NBA dataset seems to be 3 thus involving 3 clusters.

B. Simple Dataset

In [9]:
```
plt.scatter(values[:,0], values[:,1], s = 40, marker = 'x',
            label = 'Unclustered data')
plt.grid(color = 'grey', linestyle = '--', linewidth = 0.5)
plt.legend(loc = 'center left', bbox_to_anchor = (1, 0.5))
plt.xlabel('Values of x')
plt.ylabel('Values of y')
plt.title('Random x-y dataset')
plt.show()
```



In [10]:
```
ElbowMethod(values)
```



From this plot, the ideal value of k for the simple dataset seems to be 4 thus involving 4 clusters.

C. Iris Dataset

From the Iris dataset we can figure out 3 different classes:

> Iris-Setosa
>
> Iris-Versicolour
>
> Iris-Virginica

Thus, it is interesting to take 3 centroïds (K = 3)

In [46]:
```
X = pd.DataFrame(iris.data,columns=['Sepal Length', 'Sepal Width',
                                    'Petal Length', 'Petal Width'])
y = pd.DataFrame(iris.target,columns=['Classes'])
X.shape
```

Out[46]:
```
(150, 4)
```

In [12]:
```
X['species'] = pd.Series(np.random.randn(150), index=X.index)
X['species'][y['Classes']==0]='Setosa'
X['species'][y['Classes']==1]='versicolor'
X['species'][y['Classes']==2]='virginica'
X
```

Out[12]:

	Sepal Length	Sepal Width	Petal Length	Petal Width	species
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.0	3.6	1.4	0.2	Setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows × 5 columns

In [13]:
```
iris.data
```

Out[13]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
       [4.6, 3.6, 1. , 0.2],
       [5.1, 3.3, 1.7, 0.5],
       [4.8, 3.4, 1.9, 0.2],
       [5. , 3. , 1.6, 0.2],
       [5. , 3.4, 1.6, 0.4],
       [5.2, 3.5, 1.5, 0.2],
       [5.2, 3.4, 1.4, 0.2],
       [4.7, 3.2, 1.6, 0.2],
       [4.8, 3.1, 1.6, 0.2],
       [5.4, 3.4, 1.5, 0.4],
       [5.2, 4.1, 1.5, 0.1],
       [5.5, 4.2, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.2],
       [5. , 3.2, 1.2, 0.2],
       [5.5, 3.5, 1.3, 0.2],
       [4.9, 3.6, 1.4, 0.1],
       [4.4, 3. , 1.3, 0.2],
       [5.1, 3.4, 1.5, 0.2],
       [5. , 3.5, 1.3, 0.3],
       [4.5, 2.3, 1.3, 0.3],
       [4.4, 3.2, 1.3, 0.2],
       [5. , 3.5, 1.6, 0.6],
       [5.1, 3.8, 1.9, 0.4],
       [4.8, 3. , 1.4, 0.3],
       [5.1, 3.8, 1.6, 0.2],
       [4.6, 3.2, 1.4, 0.2],
       [5.3, 3.7, 1.5, 0.2],
       [5. , 3.3, 1.4, 0.2],
       [7. , 3.2, 4.7, 1.4],
       [6.4, 3.2, 4.5, 1.5],
       [6.9, 3.1, 4.9, 1.5],
       [5.5, 2.3, 4. , 1.3],
       [6.5, 2.8, 4.6, 1.5],
       [5.7, 2.8, 4.5, 1.3],
       [6.3, 3.3, 4.7, 1.6],
       [4.9, 2.4, 3.3, 1. ],
       [6.6, 2.9, 4.6, 1.3],
       [5.2, 2.7, 3.9, 1.4],
       [5. , 2. , 3.5, 1. ],
       [5.9, 3. , 4.2, 1.5],
       [6. , 2.2, 4. , 1. ],
       [6.1, 2.9, 4.7, 1.4],
```

```
[5.6, 2.9, 3.6, 1.3],
[6.7, 3.1, 4.4, 1.4],
[5.6, 3. , 4.5, 1.5],
[5.8, 2.7, 4.1, 1. ],
[6.2, 2.2, 4.5, 1.5],
[5.6, 2.5, 3.9, 1.1],
[5.9, 3.2, 4.8, 1.8],
[6.1, 2.8, 4. , 1.3],
[6.3, 2.5, 4.9, 1.5],
[6.1, 2.8, 4.7, 1.2],
[6.4, 2.9, 4.3, 1.3],
[6.6, 3. , 4.4, 1.4],
[6.8, 2.8, 4.8, 1.4],
[6.7, 3. , 5. , 1.7],
[6. , 2.9, 4.5, 1.5],
[5.7, 2.6, 3.5, 1. ],
[5.5, 2.4, 3.8, 1.1],
[5.5, 2.4, 3.7, 1. ],
[5.8, 2.7, 3.9, 1.2],
[6. , 2.7, 5.1, 1.6],
[5.4, 3. , 4.5, 1.5],
[6. , 3.4, 4.5, 1.6],
[6.7, 3.1, 4.7, 1.5],
[6.3, 2.3, 4.4, 1.3],
[5.6, 3. , 4.1, 1.3],
[5.5, 2.5, 4. , 1.3],
[5.5, 2.6, 4.4, 1.2],
[6.1, 3. , 4.6, 1.4],
[5.8, 2.6, 4. , 1.2],
[5. , 2.3, 3.3, 1. ],
[5.6, 2.7, 4.2, 1.3],
[5.7, 3. , 4.2, 1.2],
[5.7, 2.9, 4.2, 1.3],
[6.2, 2.9, 4.3, 1.3],
[5.1, 2.5, 3. , 1.1],
[5.7, 2.8, 4.1, 1.3],
[6.3, 3.3, 6. , 2.5],
[5.8, 2.7, 5.1, 1.9],
[7.1, 3. , 5.9, 2.1],
[6.3, 2.9, 5.6, 1.8],
[6.5, 3. , 5.8, 2.2],
[7.6, 3. , 6.6, 2.1],
[4.9, 2.5, 4.5, 1.7],
[7.3, 2.9, 6.3, 1.8],
[6.7, 2.5, 5.8, 1.8],
[7.2, 3.6, 6.1, 2.5],
[6.5, 3.2, 5.1, 2. ],
[6.4, 2.7, 5.3, 1.9],
[6.8, 3. , 5.5, 2.1],
[5.7, 2.5, 5. , 2. ],
[5.8, 2.8, 5.1, 2.4],
[6.4, 3.2, 5.3, 2.3],
[6.5, 3. , 5.5, 1.8],
[7.7, 3.8, 6.7, 2.2],
[7.7, 2.6, 6.9, 2.3],
[6. , 2.2, 5. , 1.5],
[6.9, 3.2, 5.7, 2.3],
[5.6, 2.8, 4.9, 2. ],
[7.7, 2.8, 6.7, 2. ],
[6.3, 2.7, 4.9, 1.8],
[6.7, 3.3, 5.7, 2.1],
[7.2, 3.2, 6. , 1.8],
[6.2, 2.8, 4.8, 1.8],
[6.1, 3. , 4.9, 1.8],
```

```
[6.4, 2.8, 5.6, 2.1],
[7.2, 3. , 5.8, 1.6],
[7.4, 2.8, 6.1, 1.9],
[7.9, 3.8, 6.4, 2. ],
[6.4, 2.8, 5.6, 2.2],
[6.3, 2.8, 5.1, 1.5],
[6.1, 2.6, 5.6, 1.4],
[7.7, 3. , 6.1, 2.3],
[6.3, 3.4, 5.6, 2.4],
[6.4, 3.1, 5.5, 1.8],
[6. , 3. , 4.8, 1.8],
[6.9, 3.1, 5.4, 2.1],
[6.7, 3.1, 5.6, 2.4],
[6.9, 3.1, 5.1, 2.3],
[5.8, 2.7, 5.1, 1.9],
[6.8, 3.2, 5.9, 2.3],
[6.7, 3.3, 5.7, 2.5],
[6.7, 3. , 5.2, 2.3],
[6.3, 2.5, 5. , 1.9],
[6.5, 3. , 5.2, 2. ],
[6.2, 3.4, 5.4, 2.3],
[5.9, 3. , 5.1, 1.8]])
```

Task 2: Apply k-means clustering on paper

In this part, the purpose is to apply the K-Mean Clustering method on the two datasets (NBA and simple) created in part 1. All the steps to perform the K-Mean Clustering will be detailed using the markdown option. At the end, using the KMean library, we will be able to see if we have found consistent results for both datasets.

Step 0: Initialization:

We randomly associate 3 centroids.

Step 1:

Using the "classical" method based on the Euclidean distance, we calculate the distance between the values and each centroid

Step 2:

Each value is assigned to the nearest centroid.

Step 3:

The centres of gravity of the groups are then computed and become the new centroids

Iterative loop:

Steps 1, 2 and 3 are repeated as long as values are reassigned to new groups after one iteration.

A. NBA Dataset

Step 0: Initialization:

First of all, it is important to choose a number "k" of centroids. In this example, k = 3 (according to the Elbow Method). This represents the three-point shooting trend over time. The initial centroids thus have their values randomly associated within a range of values included in our dataset. Thus, the Centroid 1 will take the values (1,2), Centroid 2 (3,6) and Centroid 3 (1,10).

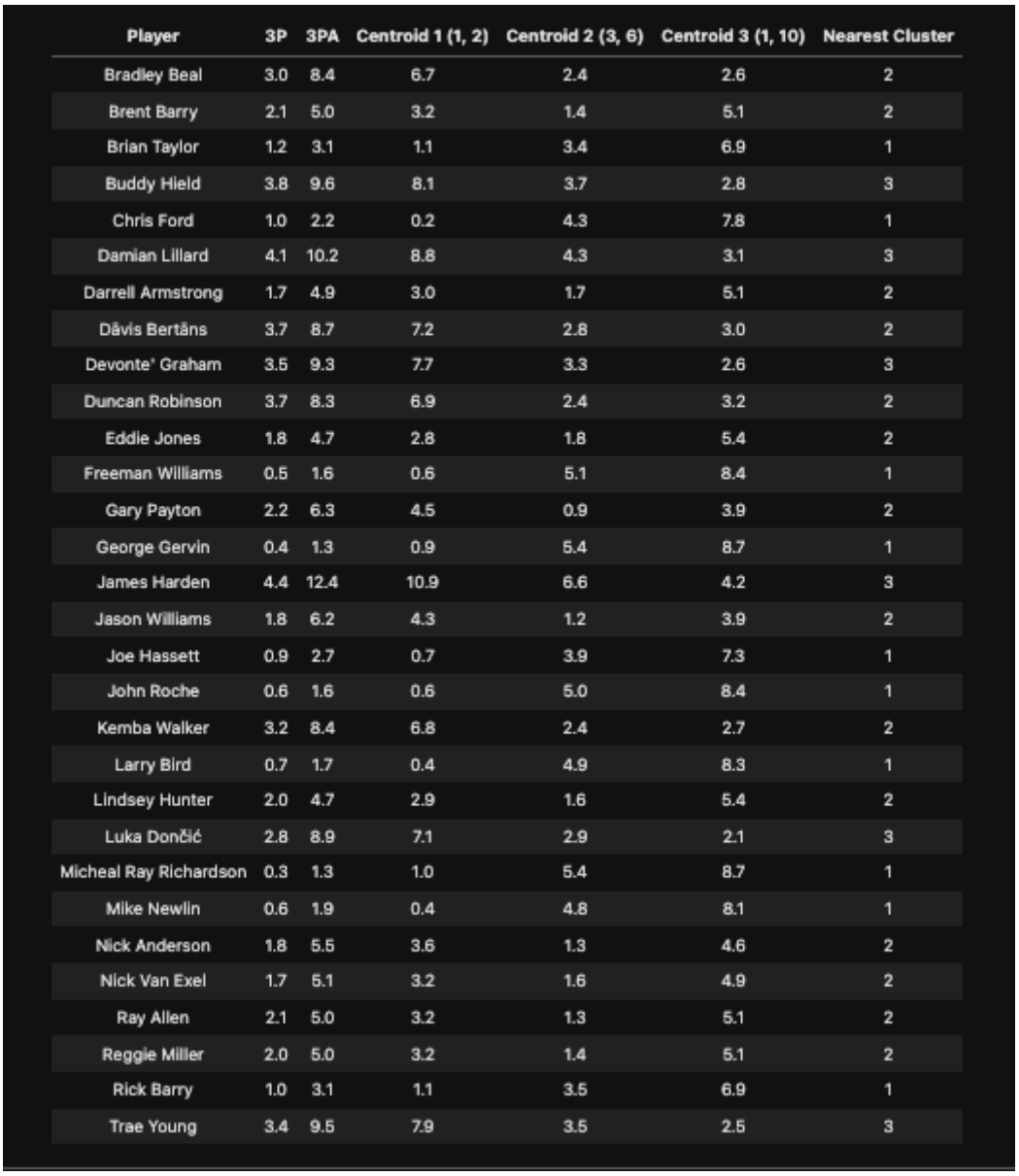
Step 1 and 2:

The Euclidean distance e is given by $$ e\left(\mathbf{x}, \mathbf{x}^{\prime}\right)=\left\|\mathbf{x}-\mathbf{x}^{\prime}\right\|=\sqrt{\sum_{j=1}^d\left(x_j-x_j^{\prime}\right)^2} $$

After calculating the distance between the value and each centroid using the euclidean distance (Step 1). It is necessary to find the nearest values to these centroids (Step 2)

Player	3P	3PA	Centroid 1 (1, 2)	Centroid 2 (3, 6)	Centroid 3 (1, 10)	Nearest Cluster
Bradley Beal	3.0	8.4	6.7	2.4	2.6	2
Brent Barry	2.1	5.0	3.2	1.4	5.1	2
Brian Taylor	1.2	3.1	1.1	3.4	6.9	1
Buddy Hield	3.8	9.6	8.1	3.7	2.8	3
Chris Ford	1.0	2.2	0.2	4.3	7.8	1
Damian Lillard	4.1	10.2	8.8	4.3	3.1	3
Darrell Armstrong	1.7	4.9	3.0	1.7	5.1	2
Dāvis Bertāns	3.7	8.7	7.2	2.8	3.0	2
Devonte' Graham	3.5	9.3	7.7	3.3	2.6	3
Duncan Robinson	3.7	8.3	6.9	2.4	3.2	2
Eddie Jones	1.8	4.7	2.8	1.8	5.4	2
Freeman Williams	0.5	1.6	0.6	5.1	8.4	1
Gary Payton	2.2	6.3	4.5	0.9	3.9	2
George Gervin	0.4	1.3	0.9	5.4	8.7	1
James Harden	4.4	12.4	10.9	6.6	4.2	3
Jason Williams	1.8	6.2	4.3	1.2	3.9	2
Joe Hassett	0.9	2.7	0.7	3.9	7.3	1
John Roche	0.6	1.6	0.6	5.0	8.4	1
Kemba Walker	3.2	8.4	6.8	2.4	2.7	2
Larry Bird	0.7	1.7	0.4	4.9	8.3	1
Lindsey Hunter	2.0	4.7	2.9	1.6	5.4	2
Luka Dončić	2.8	8.9	7.1	2.9	2.1	3
Micheal Ray Richardson	0.3	1.3	1.0	5.4	8.7	1

Player	3P	3PA	Centroid 1 (1, 2)	Centroid 2 (3, 6)	Centroid 3 (1, 10)	Nearest Cluster
Mike Newlin	0.6	1.9	0.4	4.8	8.1	1
Nick Anderson	1.8	5.5	3.6	1.3	4.6	2
Nick Van Exel	1.7	5.1	3.2	1.6	4.9	2
Ray Allen	2.1	5.0	3.2	1.3	5.1	2
Reggie Miller	2.0	5.0	3.2	1.4	5.1	2
Rick Barry	1.0	3.1	1.1	3.5	6.9	1
Trae Young	3.4	9.5	7.9	3.5	2.5	3



Player	3P	3PA	Centroid 1 (1, 2)	Centroid 2 (3, 6)	Centroid 3 (1, 10)	Nearest Cluster
Bradley Beal	3.0	8.4	6.7	2.4	2.6	2
Brent Barry	2.1	5.0	3.2	1.4	5.1	2
Brian Taylor	1.2	3.1	1.1	3.4	6.9	1
Buddy Hield	3.8	9.6	8.1	3.7	2.8	3
Chris Ford	1.0	2.2	0.2	4.3	7.8	1
Damian Lillard	4.1	10.2	8.8	4.3	3.1	3
Darrell Armstrong	1.7	4.9	3.0	1.7	5.1	2
Dāvis Bertāns	3.7	8.7	7.2	2.8	3.0	2
Devonte' Graham	3.5	9.3	7.7	3.3	2.6	3
Duncan Robinson	3.7	8.3	6.9	2.4	3.2	2
Eddie Jones	1.8	4.7	2.8	1.8	5.4	2
Freeman Williams	0.5	1.6	0.6	5.1	8.4	1
Gary Payton	2.2	6.3	4.5	0.9	3.9	2
George Gervin	0.4	1.3	0.9	5.4	8.7	1
James Harden	4.4	12.4	10.9	6.6	4.2	3
Jason Williams	1.8	6.2	4.3	1.2	3.9	2
Joe Hassett	0.9	2.7	0.7	3.9	7.3	1
John Roche	0.6	1.6	0.6	5.0	8.4	1
Kemba Walker	3.2	8.4	6.8	2.4	2.7	2
Larry Bird	0.7	1.7	0.4	4.9	8.3	1
Lindsey Hunter	2.0	4.7	2.9	1.6	5.4	2
Luka Dončić	2.8	8.9	7.1	2.9	2.1	3
Micheal Ray Richardson	0.3	1.3	1.0	5.4	8.7	1
Mike Newlin	0.6	1.9	0.4	4.8	8.1	1
Nick Anderson	1.8	5.5	3.6	1.3	4.6	2
Nick Van Exel	1.7	5.1	3.2	1.6	4.9	2
Ray Allen	2.1	5.0	3.2	1.3	5.1	2
Reggie Miller	2.0	5.0	3.2	1.4	5.1	2
Rick Barry	1.0	3.1	1.1	3.5	6.9	1
Trae Young	3.4	9.5	7.9	3.5	2.5	3

Step 3:

Cluster Centroid K_1 = ($\frac{1.2 + 1.0 + 0.5 + 0.4 + 0.9 + 0.6 + 0.7 + 0.3 + 0.6 + 1.0}{10}$, $\frac{3.1 + 2.2 + 1.6 + 1.3 + 2.7 + 1.6 + 1.7 + 1.3 + 1.9 + 3.1}{10}$) = (0.72, 2.05)

Cluster Centroid K_2 = ($\frac{3.0 + 2.1 + 1.7 + 3.7 + 3.7 + 1.8 + 2.2 + 1.8 + 3.2 + 2.0 + 1.8 + 1.7 + 2.1 + 2.0}{14}$, $\frac{8.4 + 5.0 + 4.9 + 8.7 + 8.3 + 4.7 + 6.3 + 6.2 + 8.4 + 4.7 + 5.5 + 5.1 + 5.0 + 5.0}{14}$) = (2.34, 6.16)

Cluster Centroid K_3 = ($\frac{3.8 + 4.1 + 3.5 + 4.4 + 2.8 + 3.4}{6}$, $\frac{9.6 + 10.2 + 9.3 + 12.4 + 8.9 + 9.5}{6}$) = (3.67, 9.98)

The new centroids of the clusters take the following values: (0.72, 2.05), (2.34, 6.16), (3.67, 9.98).

Iterative loop 1:

Step 1 and 2:

Player	3P	3PA	Centroid 1 (0.72, 2.05)	Centroid 2 (2.34, 6.16)	Centroid 3 (3.67, 9.98)	Best Cluster
Bradley Beal	3.0	8.4	6.7	2.3	1.7	3
Brent Barry	2.1	5.0	3.3	1.2	5.2	2
Brian Taylor	1.2	3.1	1.2	3.3	7.3	1
Buddy Hield	3.8	9.6	8.2	3.7	0.4	3
Chris Ford	1.0	2.2	0.3	4.2	8.2	1
Damian Lillard	4.1	10.2	8.8	4.4	0.5	3
Darrell Armstrong	1.7	4.9	3.0	1.4	5.5	2
Dāvis Bertāns	3.7	8.7	7.3	2.9	1.3	3
Devonte' Graham	3.5	9.3	7.8	3.3	0.7	3
Duncan Robinson	3.7	8.3	6.9	2.5	1.7	3
Eddie Jones	1.8	4.7	2.9	1.6	5.6	2
Freeman Williams	0.5	1.6	0.5	4.9	9.0	1
Gary Payton	2.2	6.3	4.5	0.2	4.0	2
George Gervin	0.4	1.3	0.8	5.2	9.3	1
James Harden	4.4	12.4	11.0	6.6	2.5	3
Jason Williams	1.8	6.2	4.3	0.5	4.2	2
Joe Hassett	0.9	2.7	0.7	3.7	7.8	1
John Roche	0.6	1.6	0.5	4.9	8.9	1
Kemba Walker	3.2	8.4	6.8	2.4	1.7	3
Larry Bird	0.7	1.7	0.4	4.8	8.8	1
Lindsey Hunter	2.0	4.7	2.9	1.5	5.5	2
Luka Dončić	2.8	8.9	7.2	2.8	1.4	3
Micheal Ray Richardson	0.3	1.3	0.9	5.3	9.3	1
Mike Newlin	0.6	1.9	0.2	4.6	8.6	1
Nick Anderson	1.8	5.5	3.6	0.9	4.9	2
Nick Van Exel	1.7	5.1	3.2	1.2	5.3	2
Ray Allen	2.1	5.0	3.3	1.2	5.2	2
Reggie Miller	2.0	5.0	3.2	1.2	5.3	2

Player	3P	3PA	Centroid 1 (0.72, 2.05)	Centroid 2 (2.34, 6.16)	Centroid 3 (3.67, 9.98)	Best Cluster
Rick Barry	1.0	3.1	1.1	3.3	7.4	1
Trae Young	3.4	9.5	7.9	3.5	0.6	3

Player	3P	3PA	Centroid 1 (0.72, 2.05)	Centroid 2 (2.34, 6.16)	Centroid 3 (3.67, 9.98)	Best Cluster
Bradley Beal	3.0	8.4	6.7	2.3	1.7	3
Brent Barry	2.1	5.0	3.3	1.2	5.2	2
Brian Taylor	1.2	3.1	1.2	3.3	7.3	1
Buddy Hield	3.8	9.6	8.2	3.7	0.4	3
Chris Ford	1.0	2.2	0.3	4.2	8.2	1
Damian Lillard	4.1	10.2	8.8	4.4	0.5	3
Darrell Armstrong	1.7	4.9	3.0	1.4	5.5	2
Dāvis Bertāns	3.7	8.7	7.3	2.9	1.3	3
Devonte' Graham	3.5	9.3	7.8	3.3	0.7	3
Duncan Robinson	3.7	8.3	6.9	2.5	1.7	3
Eddie Jones	1.8	4.7	2.9	1.6	5.6	2
Freeman Williams	0.5	1.6	0.5	4.9	9.0	1
Gary Payton	2.2	6.3	4.5	0.2	4.0	2
George Gervin	0.4	1.3	0.8	5.2	9.3	1
James Harden	4.4	12.4	11.0	6.6	2.5	3
Jason Williams	1.8	6.2	4.3	0.5	4.2	2
Joe Hassett	0.9	2.7	0.7	3.7	7.8	1
John Roche	0.6	1.6	0.5	4.9	8.9	1
Kemba Walker	3.2	8.4	6.8	2.4	1.7	3
Larry Bird	0.7	1.7	0.4	4.8	8.8	1
Lindsey Hunter	2.0	4.7	2.9	1.5	5.5	2
Luka Dončić	2.8	8.9	7.2	2.8	1.4	3
Micheal Ray Richardson	0.3	1.3	0.9	5.3	9.3	1
Mike Newlin	0.6	1.9	0.2	4.6	8.6	1
Nick Anderson	1.8	5.5	3.6	0.9	4.9	2
Nick Van Exel	1.7	5.1	3.2	1.2	5.3	2
Ray Allen	2.1	5.0	3.3	1.2	5.2	2
Reggie Miller	2.0	5.0	3.2	1.2	5.3	2
Rick Barry	1.0	3.1	1.1	3.3	7.4	1
Trae Young	3.4	9.5	7.9	3.5	0.6	3

Iterative loop 1:

Step 3:

Cluster Centroid K_1 = (0.72, 2.05)

Cluster Centroid K_2 = (1.92, 5.24)

Cluster Centroid K_3 = (3.56, 9.37)

Iterative loop 2:

Step 1 and 2:

Player	3P	3PA	Centroid 1 (0.72, 2.05)	Centroid 2 (1.92, 5.24)	Centroid 3 (3.56, 9.37)	Best Cluster
Bradley Beal	3.0	8.4	6.7	3.3	1.1	3
Brent Barry	2.1	5.0	3.3	0.3	4.6	2
Brian Taylor	1.2	3.1	1.2	2.3	6.7	1
Buddy Hield	3.8	9.6	8.2	4.7	0.3	3
Chris Ford	1.0	2.2	0.3	3.2	7.6	1
Damian Lillard	4.1	10.2	8.8	5.4	1.0	3
Darrell Armstrong	1.7	4.9	3.0	0.4	4.8	2
Dāvis Bertāns	3.7	8.7	7.3	3.9	0.7	3
Devonte' Graham	3.5	9.3	7.8	4.4	0.1	3
Duncan Robinson	3.7	8.3	6.9	3.5	1.1	3
Eddie Jones	1.8	4.7	2.9	0.6	5.0	2
Freeman Williams	0.5	1.6	0.5	3.9	8.4	1
Gary Payton	2.2	6.3	4.5	1.1	3.4	2
George Gervin	0.4	1.3	0.8	4.2	8.7	1
James Harden	4.4	12.4	11.0	7.6	3.1	3
Jason Williams	1.8	6.2	4.3	1.0	3.6	2
Joe Hassett	0.9	2.7	0.7	2.7	7.2	1
John Roche	0.6	1.6	0.5	3.9	8.3	1
Kemba Walker	3.2	8.4	6.8	3.4	1.0	3
Larry Bird	0.7	1.7	0.4	3.7	8.2	1
Lindsey Hunter	2.0	4.7	2.9	0.5	4.9	2
Luka Dončić	2.8	8.9	7.2	3.8	0.9	3
Micheal Ray Richardson	0.3	1.3	0.9	4.3	8.7	1
Mike Newlin	0.6	1.9	0.2	3.6	8.0	1
Nick Anderson	1.8	5.5	3.6	0.3	4.3	2
Nick Van Exel	1.7	5.1	3.2	0.3	4.7	2
Ray Allen*	2.1	5.0	3.3	0.3	4.6	2
Reggie Miller	2.0	5.0	3.2	0.3	4.6	2
Rick Barry	1.0	3.1	1.1	2.3	6.8	1
Trae Young	3.4	9.5	7.9	4.5	0.2	3

Player	3P	3PA	Centroid 1 (0.72, 2.05)	Centroid 2 (1.92, 5.24)	Centroid 3 (3.56, 9.37)	Best Cluster
Bradley Beal	3.0	8.4	6.7	3.3	1.1	3
Brent Barry	2.1	5.0	3.3	0.3	4.6	2
Brian Taylor	1.2	3.1	1.2	2.3	6.7	1
Buddy Hield	3.8	9.6	8.2	4.7	0.3	3
Chris Ford	1.0	2.2	0.3	3.2	7.6	1
Damian Lillard	4.1	10.2	8.8	5.4	1.0	3
Darrell Armstrong	1.7	4.9	3.0	0.4	4.8	2
Dāvis Bertāns	3.7	8.7	7.3	3.9	0.7	3
Devonte' Graham	3.5	9.3	7.8	4.4	0.1	3
Duncan Robinson	3.7	8.3	6.9	3.5	1.1	3
Eddie Jones	1.8	4.7	2.9	0.6	5.0	2
Freeman Williams	0.5	1.6	0.5	3.9	8.4	1
Gary Payton	2.2	6.3	4.5	1.1	3.4	2
George Gervin	0.4	1.3	0.8	4.2	8.7	1
James Harden	4.4	12.4	11.0	7.6	3.1	3
Jason Williams	1.8	6.2	4.3	1.0	3.6	2
Joe Hassett	0.9	2.7	0.7	2.7	7.2	1
John Roche	0.6	1.6	0.5	3.9	8.3	1
Kemba Walker	3.2	8.4	6.8	3.4	1.0	3
Larry Bird	0.7	1.7	0.4	3.7	8.2	1
Lindsey Hunter	2.0	4.7	2.9	0.5	4.9	2
Luka Dončić	2.8	8.9	7.2	3.8	0.9	3
Micheal Ray Richardson	0.3	1.3	0.9	4.3	8.7	1
Mike Newlin	0.6	1.9	0.2	3.6	8.0	1
Nick Anderson	1.8	5.5	3.6	0.3	4.3	2
Nick Van Exel	1.7	5.1	3.2	0.3	4.7	2
Ray Allen*	2.1	5.0	3.3	0.3	4.6	2
Reggie Miller	2.0	5.0	3.2	0.3	4.6	2
Rick Barry	1.0	3.1	1.1	2.3	6.8	1
Trae Young	3.4	9.5	7.9	4.5	0.2	3

Iterative loop 2:

Step 3:

Cluster Centroid K_1 = (0.72, 2.05)

Cluster Centroid K_2 = (1.92, 5.24)

Cluster Centroid K_3 = (3.56, 9.37)

We obtain the same clusters as before after this further iteration. There is thus convergence and our hand-made algorithm is complete. Our centroids clusters take the following center values: (0.72, 2.05), (1.92, 5.24), (3.56, 9.37).

B. Sanity Check and Visualisation

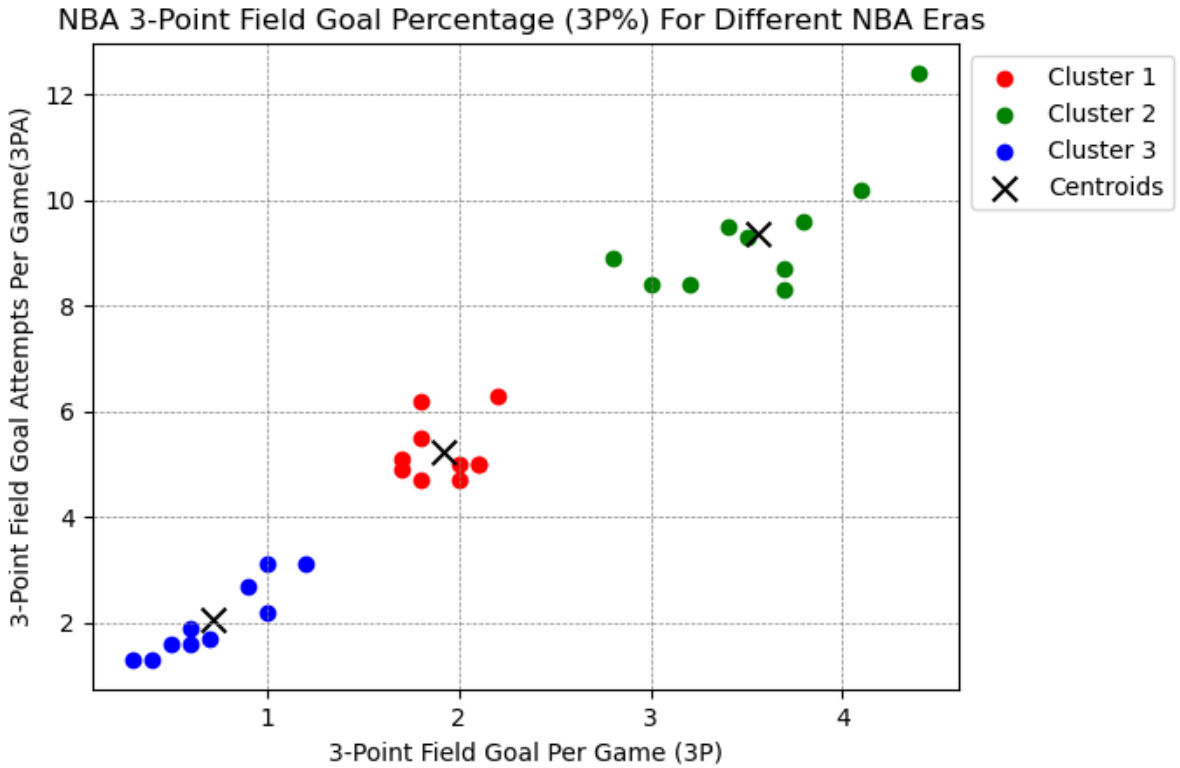
As a sanity check has been required, the library from sklearn.cluster import KMeans is used to perform a very quick clustering of my values

In [14]:
```
start_time_NBA_1 = time.time()
```

In [15]:
```
kmeans = KMeans(n_clusters= 3)
label = kmeans.fit_predict(three_point_clustering)
print(kmeans.labels_)

print("Centroid clusters take the following center values",
      kmeans.cluster_centers_)
print("- %.4f seconds -" % (time.time() - start_time_NBA_1))
```

```
[1 0 2 1 2 1 0 1 1 1 0 2 0 2 1 0 2 2 1 2 0 1 2 2 0 0 0 0 2 1]
Centroid clusters take the following center values [[1.92 5.24]
 [3.56 9.37]
 [0.72 2.05]]
- 2.6974 seconds -
```

In [16]:
```
label0 = three_point_clustering[label == 0]
label1 = three_point_clustering[label == 1]
label2 = three_point_clustering[label == 2]

plt.scatter(label0[:,0], label0[:,1], color = 'red', label = 'Cluster 1')
plt.scatter(label1[:,0], label1[:,1], color = 'green', label = 'Cluster 2')
plt.scatter(label2[:,0], label2[:,1], color = 'blue', label = 'Cluster 3')

plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            color = 'black', marker = 'x', s = 100, label = 'Centroids')

plt.xlabel('3-Point Field Goal Per Game (3P)')
plt.ylabel('3-Point Field Goal Attempts Per Game(3PA)')
plt.title('NBA 3-Point Field Goal Percentage (3P%) For Different NBA Eras ')
plt.grid(color = 'grey', linestyle = '--', linewidth = 0.5)
plt.legend(bbox_to_anchor = (1, 1))
plt.show()
```



C. Simple Dataset

Step 0: Initialization:

The Centroid 1 will take the values (1, 2), Centroid 2 (3, 4), Centroid 3 (5, 6) and Centroid 4 (7, 8)

Step 1 and 2:

Data	Centroid 1 (1, 2)	Centroid 2 (3, 4)	Centroid 3 (5, 6)	Centroid 4 (9, 4)	Best Cluster
[1, 2]	0.0	2.8	5.7	8.2	1
[2, 2]	1.0	2.2	5.0	7.3	1
[1, 3]	1.0	2.2	5.0	8.1	1
[8, 2]	7.0	5.4	5.0	2.2	3
[9, 3]	8.1	6.1	5.0	1.0	3
[8, 3]	7.1	5.1	4.2	1.4	3
[1, 7]	5.0	3.6	4.1	8.5	2
[1, 9]	7.0	5.4	5.0	9.4	3
[2, 8]	6.1	4.1	3.6	8.1	3
[3, 9]	7.3	5.0	3.6	7.8	3
[4, 4]	3.6	1.0	2.2	5.0	2
[5, 5]	5.0	2.2	1.0	4.1	3

Step 3:

Cluster Centroid K_1 = ($\frac{1 + 2 + 1}{3}$, $\frac{2 + 2 + 3}{3}$) = ($\frac{4}{3}$, $\frac{7}{3}$)

Cluster Centroid K_2 = ($\frac{25}{3}$, $\frac{8}{3}$)

Cluster Centroid K_3 = (2.5, 5.5)

Cluster Centroid K_4 = (2.75, 7.75)

Iterative loop 1:

Step 1 and 2:

Data	Centroid 1	Centroid 2	Centroid 3	Centroid 4	Best Cluster
[1, 2]	0.5	7.4	3.8	6.0	1
[2, 2]	0.7	6.4	3.5	5.8	1
[1, 3]	0.7	7.3	2.9	5.1	1
[8, 2]	6.7	0.7	6.5	7.8	2
[9, 3]	7.7	0.7	7.0	7.9	2
[8, 3]	6.7	0.5	6.0	7.1	2
[1, 7]	4.7	8.5	2.1	1.9	4

Data	Centroid 1	Centroid 2	Centroid 3	Centroid 4	Best Cluster
[1, 9]	6.7	9.7	3.8	2.2	4
[2, 8]	5.7	8.3	2.5	0.8	4
[3, 9]	6.9	8.3	3.5	1.3	4
[4, 4]	3.1	4.5	2.1	4.0	3
[5, 5]	4.5	4.1	2.5	3.6	3

Iterative loop 1:

Step 3:

Cluster Centroid K_1 = ($\frac{4}{3}$, $\frac{7}{3}$)

Cluster Centroid K_2 = ($\frac{25}{3}$, $\frac{8}{3}$)

Cluster Centroid K_3 = (4.5, 4.5)

Cluster Centroid K_4 = (1.75, 8.25)

Iterative loop 2:

Step 1 and 2:

Data	Centroid 1	Centroid 2	Centroid 3	Centroid 4	Best Cluster
[1, 2]	0.5	7.4	4.3	6.3	1
[2, 2]	0.7	6.4	3.5	6.3	1
[1, 3]	0.7	7.3	3.8	5.3	1
[8, 2]	6.7	0.7	4.3	8.8	2
[9, 3]	7.7	0.7	4.7	9.0	2
[8, 3]	6.7	0.5	3.8	8.2	2
[1, 7]	4.7	8.5	4.3	1.5	4
[1, 9]	6.7	9.7	5.7	1.1	4
[2, 8]	5.7	8.3	4.3	0.4	4
[3, 9]	6.9	8.3	4.7	1.5	4
[4, 4]	3.1	4.5	0.7	4.8	3
[5, 5]	4.5	4.1	0.7	4.6	3

Iterative loop 2:

Step 3:

Cluster Centroid K_1 = ($\frac{4}{3}$, $\frac{7}{3}$)

Cluster Centroid K_2 = ($\frac{25}{3}$, $\frac{8}{3}$)

Cluster Centroid K_3 = (4.5, 4.5)

Cluster Centroid K_4 = (1.75, 8.25)

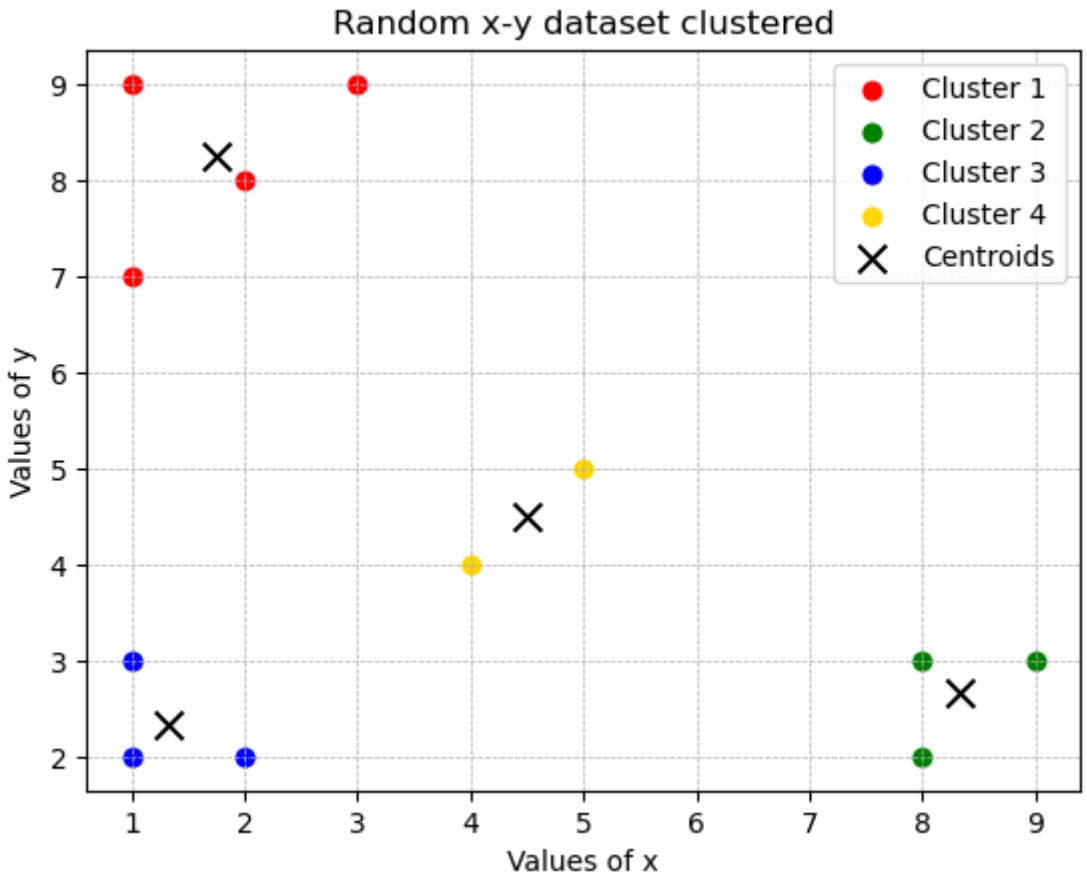
We obtain the same clusters as before after this further iteration. There is thus convergence and our hand-made algorithm is complete. Our centroid clusters take the following center values: ($\frac{4}{3}$, $\frac{7}{3}$), ($\frac{25}{3}$, $\frac{8}{3}$), (4.5, 4.5), (1.75, 8.25).

D. Sanity Check and Visualisation

In [17]:
```
kmeans = KMeans(n_clusters= 4)
label = kmeans.fit_predict(values)
print("Centroid clusters take the following center values",
      kmeans.cluster_centers_)
```

```
Centroid clusters take the following center values [[1.75       8.25      ]
 [8.33333333 2.66666667]
 [1.33333333 2.33333333]
 [4.5        4.5       ]]
```

In [18]:
```
label0 = values[label == 0]
label1 = values[label == 1]
label2 = values[label == 2]
label3 = values[label == 3]


plt.scatter(label0[:,0], label0[:,1], color = 'red', s = 40,
            label = 'Cluster 1')
plt.scatter(label1[:,0], label1[:,1], color = 'green', s = 40,
            label = 'Cluster 2')
plt.scatter(label2[:,0], label2[:,1], color = 'blue', s = 40,
            label = 'Cluster 3')
plt.scatter(label3[:,0], label3[:,1], color = 'gold', s = 40,
            label = 'Cluster 4')

plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            color = 'black', marker = 'x', s = 100, label = 'Centroids')


plt.xlabel('Values of x')
plt.ylabel('Values of y')
plt.title('Random x-y dataset clustered')
plt.grid(linestyle = '--', linewidth = 0.5)
plt.legend(bbox_to_anchor = (1, 1))
plt.show()
```



Task 3: Create a test harness

A test harness is a set of test programs and data used by developers to perform unit tests on software models under development. The one that is presented below, allows to check if the final centroids obtained by my algorithm (task 4) correspond to those obtained by the library from sklearn.cluster.

In [44]:
```
def harness_test():
    rows = 2
    columns = 4


    myalgorithm = []
    for j in range(columns):
        for i in range(rows):
            myalgorithm.append(center_centroids[i][j],)

    sklearnlibrary = []
    for l in range(rows):
        for k in range(columns):
            sklearnlibrary.append(kmeans.cluster_centers_.tolist()[k][l],)

    myalgorithm = [round(num, 5) for num in myalgorithm]
    sklearnlibrary = [round(num, 5) for num in sklearnlibrary]

    print('Our Algorithm Centroids coordinates values: '
          + str(sorted(myalgorithm)))
    print("Sklearn's Centoids coordinates values: "
          + str(sorted(sklearnlibrary)))

    if sorted(sklearnlibrary) == sorted(myalgorithm):
        print('Harness test is Successful')
```

```
    else:
        print('Harness test is Unsuccessful ')
```

Task 4: Implement k-means clustering in Python

A. NBA Dataset

In [20]:
```
start_time_NBA_2 = time.time()
```

In [21]:
```
clusters_NBA=len(np.unique(y))
```

In [22]:
```
def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1 - x2)**2))
```

In [23]:
```
class K_Means_Clustering:

    def __init__(self, data, k, maximum_iterations):
        self.data = data
        self.k = k
        self.maximum_iterations = maximum_iterations

    def predict(self):

        center_centroids = defaultdict(int)

        Cluster = self.k
        maximum_iterations = self.maximum_iterations

        for i in range(Cluster):
            center_centroids[i] = self.data[i]

        r = 0

        for i in range(maximum_iterations):
            r = r + 1
            group = defaultdict(list)

            for key in range(Cluster):

                group[key]=[]

            for datapoint in self.data:

                distance=[]

                for j in range(Cluster):

                    distance.append(euclidean_distance
                                    (datapoint,center_centroids[j]))


                index = distance.index(min(distance))

                group[index].append(datapoint)

                previous_centroid = dict(center_centroids)
```

```
            for t in range(Cluster):

                group_ = group[t]

                center_centroids[t] = np.mean(group_,axis=0)

            f = 1

            for t in range(Cluster):
                if np.sum(((center_centroids[t]) -
                           (previous_centroid[t]))/
                          (previous_centroid[t]) * 100) > 0.001:

                    f = 0

            if f == 1:

                break

        return group,center_centroids
```

In [24]:
```
kmeans=K_Means_Clustering(three_point_clustering[:,:4],clusters_NBA,10000)

group,center_centroids=kmeans.predict()


for i in range(0,3):
    group[i]=np.array(group[i]).tolist()

for i in range(0,3):
    print(len(group[i]))
print(center_centroids)
```

```
10
10
10
defaultdict(<class 'int'>, {0: array([3.56, 9.37]), 1: array([1.92, 5.24]),
2: array([0.72, 2.05])})
```

In [25]:
```
print("--- %.4f seconds -" % (time.time() - start_time_NBA_2))
```

```
--- 0.1323 seconds -
```

Using the NBA database representing the 3-point trend, we can observe that we recover with our algorithm the same centroids as with the pen & paper method. So, our centroids clusters take the following center values: (0.72, 2.05), (1.92, 5.24), (3.56, 9.37) with 10 players for each cluster. Furthermore, in the previous section (cf. Sanity Check and Visualisation) I calculated the time taken to perform K_Mean_Clustering via the library from sklearn.cluster. the time was - 2.9302 seconds - compared to the time taken by my algorithm being - 0.1323 seconds - seconds -

B. Iris Dataset

In [26]:
```
start_time_iris_1 = time.time()
```

In [27]:
```
clusters_iris=len(np.unique(y))
```

In [28]:
```
kmeans=K_Means_Clustering(iris.data[:,:4],clusters_iris,10000)

group,center_centroids=kmeans.predict()


for i in range(0,3):
    group[i]=np.array(group[i]).tolist()

for i in range(0,3):
    print(len(group[i]))
print(center_centroids)
print("- %.4f seconds -" % (time.time() - start_time_iris_1))
```

```
39
61
50
defaultdict(<class 'int'>, {0: array([6.85384615, 3.07692308, 5.71538462, 2.
05384615]), 1: array([5.88360656, 2.74098361, 4.38852459, 1.43442623]), 2: a
rray([5.006, 3.428, 1.462, 0.246])})
- 0.1533 seconds -
```

In [29]:
```
start_time_iris_2 = time.time()
```

In [47]:
```
kmeans = KMeans(n_clusters= 3)
label = kmeans.fit_predict(iris.data)
print(kmeans.labels_)
print("Centroid clusters take the following center values",
      kmeans.cluster_centers_)
print("- %.4f seconds -" % (time.time() - start_time_iris_2))
```

```
[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 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0 2 2 2 2
 2 2 0 0 2 2 2 2 0 2 0 2 0 2 2 0 0 2 2 2 2 2 0 2 2 2 2 0 2 2 2 0 2 2 2 0 2
 2 0]
Centroid clusters take the following center values [[5.9016129  2.7483871
4.39354839 1.43387097]
 [5.006      3.428      1.462      0.246     ]
 [6.85       3.07368421 5.74210526 2.07105263]]
- 1705.8097 seconds -
```

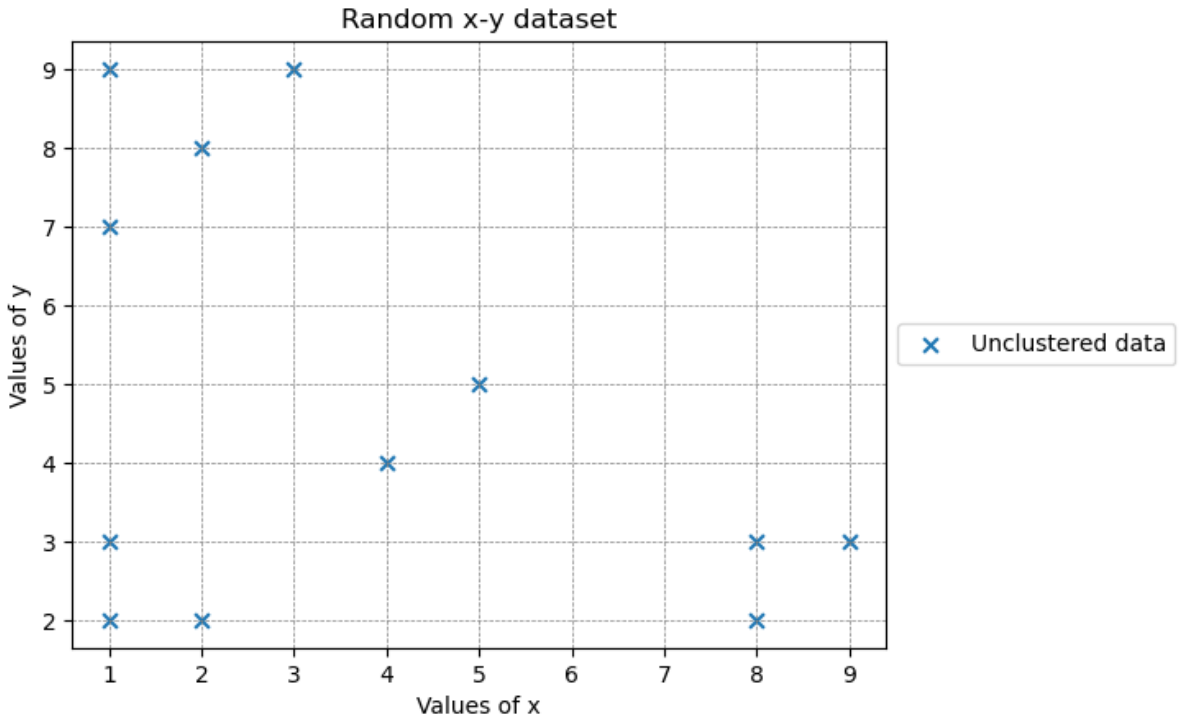
Using the same algorithm as for the NBA data, I computed the centroids for the iris database. The time taken to perform K_Mean_Clustering via my algorithm is - 0.1241 seconds - compared to the time taken by the library from sklearn.cluster which is - 2.9302s seconds -. Moreover, thanks to the algorithm we know through the code line "len(group[i]" that Cluster 1 is composed of 39 values Cluster 2 is composed of 61 values and Cluster 3 is composed of 50 values

So it is possible to find the same number of flower categories as presented by the library above ([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 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 2 0 2 0 0 0 2 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])

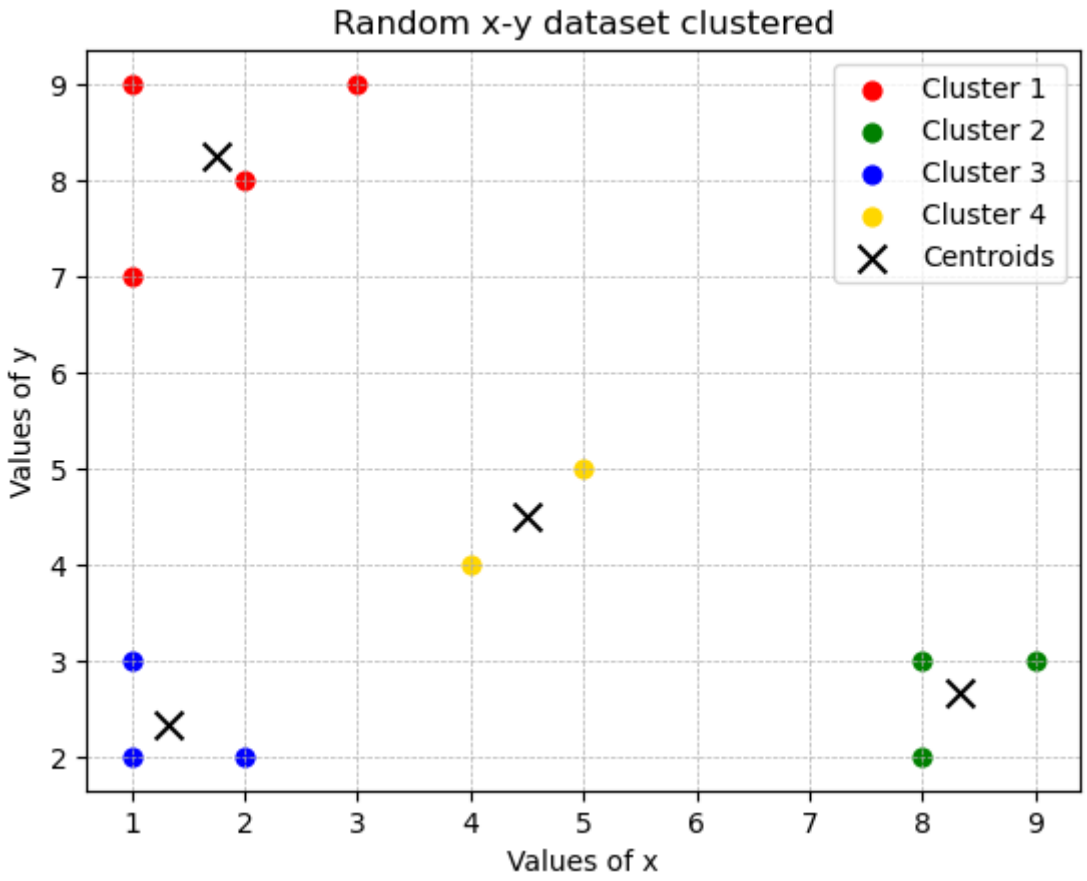
Task 5 (Optional): Add a visualisation

This part has been carried out throughout the project, providing graphs to better understand the findings presented. Please find below a summary of the different graphs.

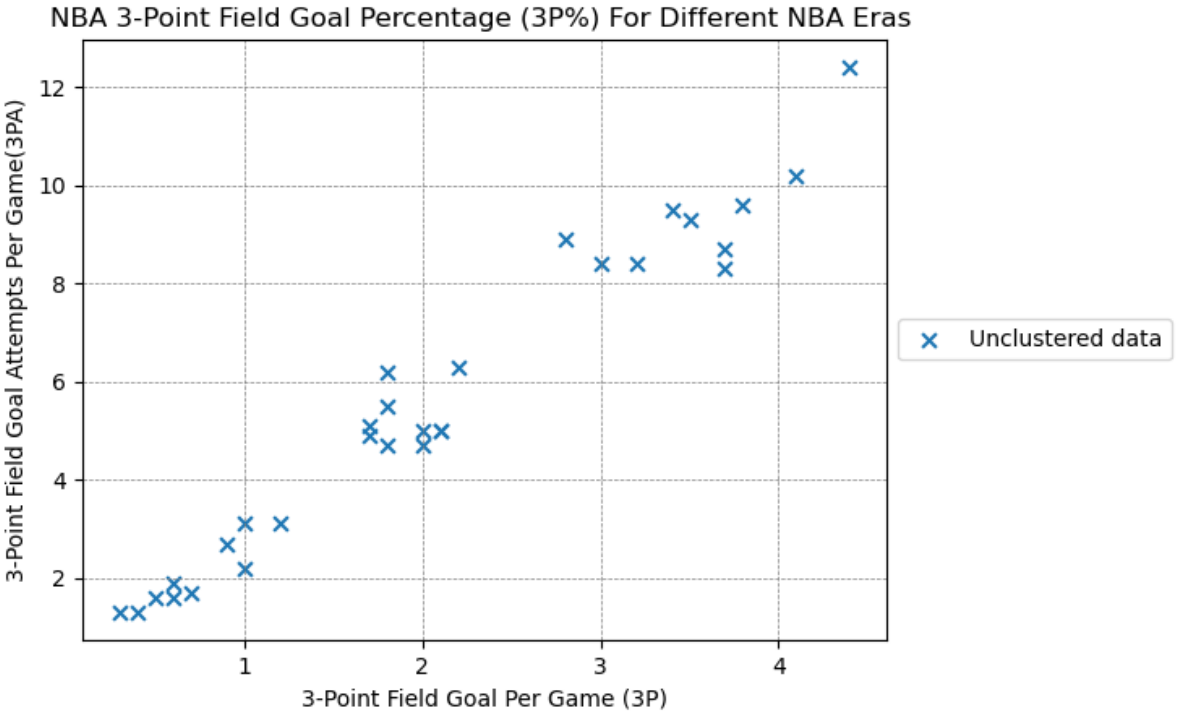
In [31]:
```
plt.scatter(values[:,0], values[:,1], s = 40, marker = 'x', label = 'Unclust
plt.grid(color = 'grey', linestyle = '--', linewidth = 0.5)
plt.legend(loc = 'center left', bbox_to_anchor = (1, 0.5))
plt.xlabel('Values of x')
plt.ylabel('Values of y')
plt.title('Random x-y dataset')
plt.show()
```



In [18]:
```
label0 = values[label == 0]
label1 = values[label == 1]
label2 = values[label == 2]
label3 = values[label == 3]


plt.scatter(label0[:,0], label0[:,1], color = 'red', s = 40, label = 'Cluste
plt.scatter(label1[:,0], label1[:,1], color = 'green', s = 40, label = 'Clus
plt.scatter(label2[:,0], label2[:,1], color = 'blue', s = 40, label = 'Clust
plt.scatter(label3[:,0], label3[:,1], color = 'gold', s = 40, label = 'Clust

plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            color = 'black', marker = 'x', s = 100, label = 'Centroids')


plt.xlabel('Values of x')
plt.ylabel('Values of y')
plt.title('Random x-y dataset clustered')
plt.grid(linestyle = '--', linewidth = 0.5)
plt.legend(bbox_to_anchor = (1, 1))
plt.show()
```

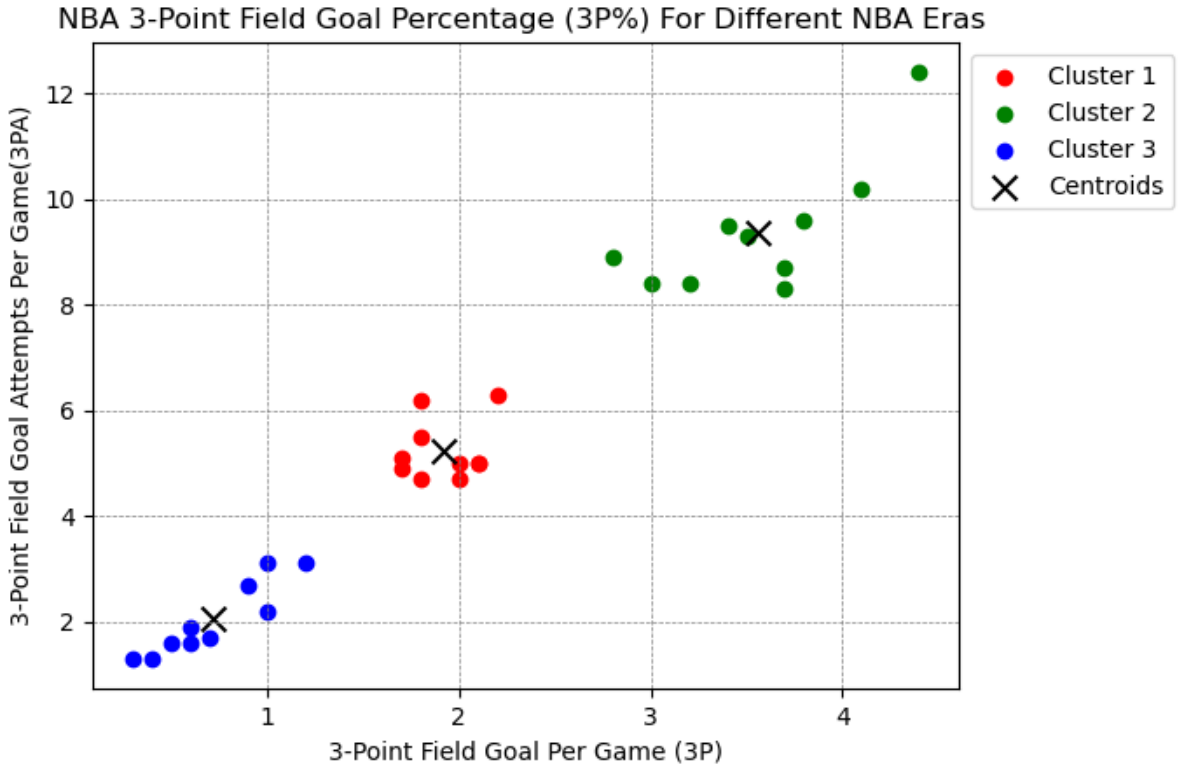


In [34]:
```
plt.scatter(three_point_clustering[:,0], three_point_clustering[:,1], s = 40
plt.grid(color = 'grey', linestyle = '--', linewidth = 0.5)
plt.legend(loc = 'center left', bbox_to_anchor = (1, 0.5))
plt.xlabel('3-Point Field Goal Per Game (3P)')
plt.ylabel('3-Point Field Goal Attempts Per Game(3PA)')
plt.title('NBA 3-Point Field Goal Percentage (3P%) For Different NBA Eras ')
plt.show()
```

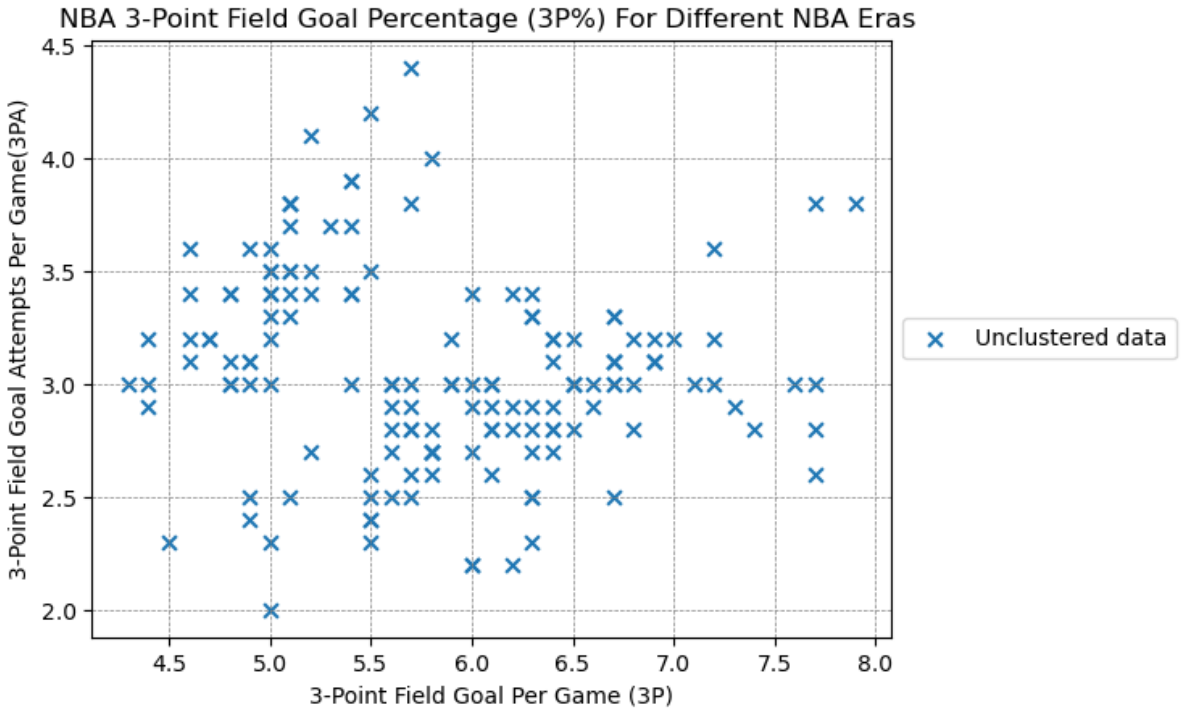


In [16]:
```
label0 = three_point_clustering[label == 0]
label1 = three_point_clustering[label == 1]
label2 = three_point_clustering[label == 2]
```

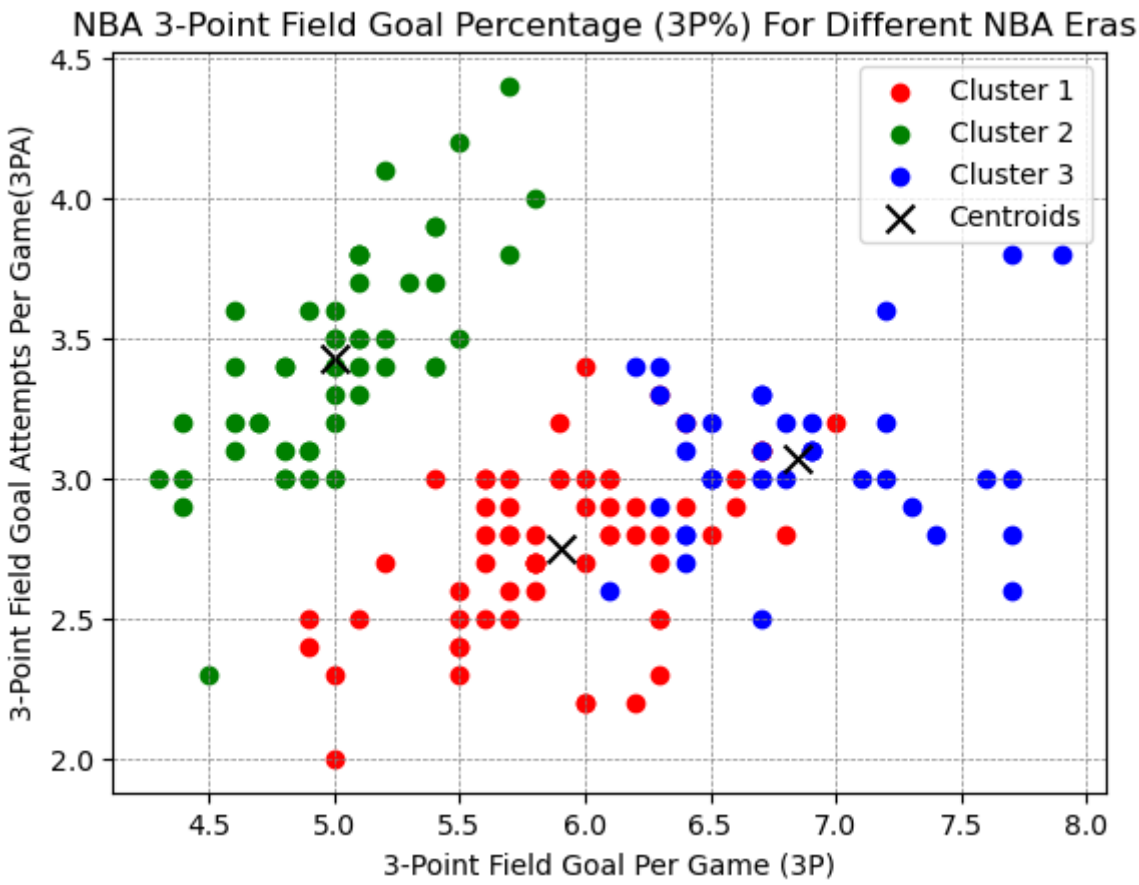
```
plt.scatter(label0[:,0], label0[:,1], color = 'red', label = 'Cluster 1')
plt.scatter(label1[:,0], label1[:,1], color = 'green', label = 'Cluster 2')
plt.scatter(label2[:,0], label2[:,1], color = 'blue', label = 'Cluster 3')

plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            color = 'black', marker = 'x', s = 100, label = 'Centroids')

plt.xlabel('3-Point Field Goal Per Game (3P)')
plt.ylabel('3-Point Field Goal Attempts Per Game(3PA)')
plt.title('NBA 3-Point Field Goal Percentage (3P%) For Different NBA Eras ')
plt.grid(color = 'grey', linestyle = '--', linewidth = 0.5)
plt.legend(bbox_to_anchor = (1, 1))
plt.show()
```



In [39]:
```
plt.scatter(iris.data[:,0], iris.data[:,1], s = 40, marker = 'x', label = 'U
plt.grid(color = 'grey', linestyle = '--', linewidth = 0.5)
plt.legend(loc = 'center left', bbox_to_anchor = (1, 0.5))
plt.xlabel('Length')
plt.ylabel('Width')
plt.show()
```



In [38]:
```
label0 = iris.data[label == 0]
label1 = iris.data[label == 1]
label2 = iris.data[label == 2]

plt.scatter(label0[:,0], label0[:,1], color = 'red', label = 'Cluster 1')
plt.scatter(label1[:,0], label1[:,1], color = 'green', label = 'Cluster 2')
plt.scatter(label2[:,0], label2[:,1], color = 'blue', label = 'Cluster 3')

plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            color = 'black', marker = 'x', s = 100, label = 'Centroids')

plt.xlabel('Length')
plt.ylabel('Width')
plt.grid(color = 'grey', linestyle = '--', linewidth = 0.5)
plt.legend(bbox_to_anchor = (1, 1))
plt.show()
```


NBA 3-Point Field Goal Percentage (3P%) For Different NBA Eras
Cluster 1
Cluster 2
Cluster 3
Centroids
3-Point Field Goal Attempts Per Game(3PA)
3-Point Field Goal Per Game (3P)
4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0

In [ ]: