Practica 2- Clustering

Nombre: Paulina Aldape

Clustering o agrupamiento

La idea principal de aplicar el clustering es para poder organizar los puntos que se dibujan en un plan en diferentes categorias o grupos.

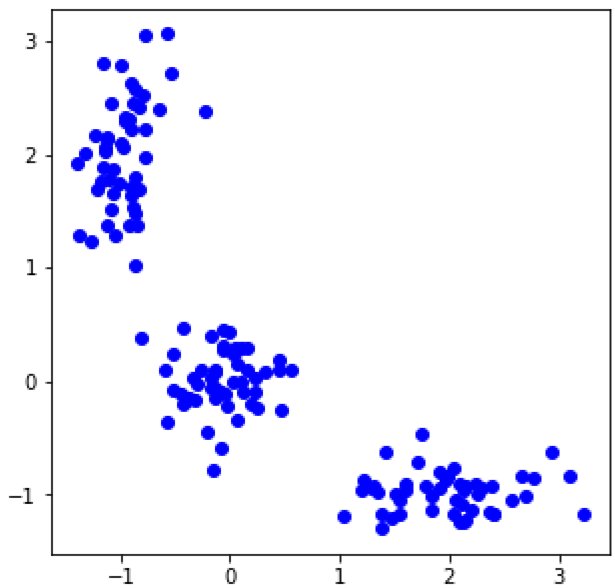
K medias

El algortimo de kmedias es uno de los mas simples y famosos ejemplos que podemos escuchar de los algoritmos de clustering. Estos son los pasos que se realizan para llevarlo a cabo:

- Se selecciona el numero de clusters k que pienses que es numero optimo.
- Se inicializan los k puntos como centroides de manera aleatoria dentro del espacio en donde estan distribuidos nuestros datos.
- Relacionamos cada observacion son el centroide mas cercano.
- Los centroides se actualizan al centro de todos los datos que se le atribuyeron en el paso anterior.
- Estos ultimos dos pasos se repetiran hasta que todos los centroides esten estables.

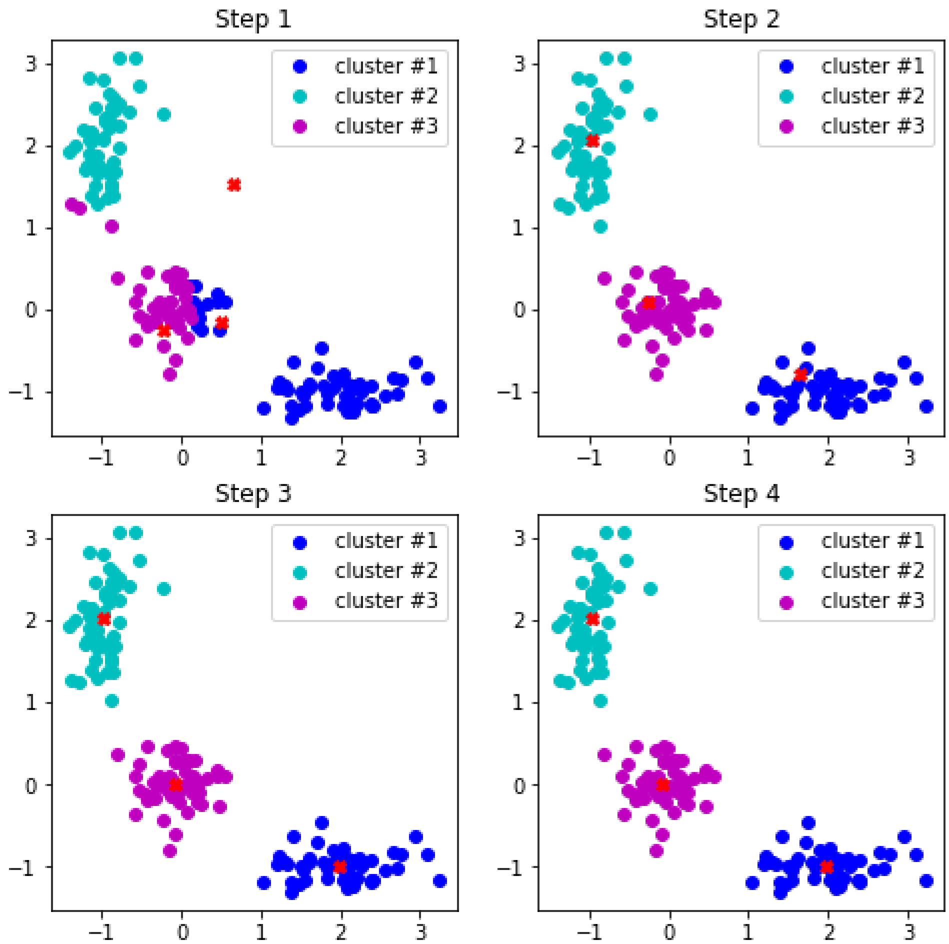
A continuacion utilizaremos las herramientas que vimos en la clase asada para generar datos dummy y ejemplificar el uso del clustering.

In [1]:
```
import numpy as np

import matplotlib.pyplot as plt
%matplotlib inline

X = np.zeros((150, 2))

np.random.seed(seed=42)
X[:50, 0] = np.random.normal(loc=0.0, scale=.3, size=50)
X[:50, 1] = np.random.normal(loc=0.0, scale=.3, size=50)

X[50:100, 0] = np.random.normal(loc=2.0, scale=.5, size=50)
X[50:100, 1] = np.random.normal(loc=-1.0, scale=.2, size=50)

X[100:150, 0] = np.random.normal(loc=-1.0, scale=.2, size=50)
X[100:150, 1] = np.random.normal(loc=2.0, scale=.5, size=50)

plt.figure(figsize=(5, 5))
plt.plot(X[:, 0], X[:, 1], 'bo');
```



In [2]:
```
from scipy.spatial.distance import cdist

# Randomly allocate the 3 centroids
np.random.seed(seed=42)
centroids = np.random.normal(loc=0.0, scale=1., size=6)
centroids = centroids.reshape((3, 2))

cent_history = []
cent_history.append(centroids)

for i in range(3):
    # Calculating the distance from a point to a centroid
    distances = cdist(X, centroids)
    # Checking what's the closest centroid for the point
    labels = distances.argmin(axis=1)

    # Labeling the point according the point's distance
    centroids = centroids.copy()
    centroids[0, :] = np.mean(X[labels == 0, :], axis=0)
    centroids[1, :] = np.mean(X[labels == 1, :], axis=0)
    centroids[2, :] = np.mean(X[labels == 2, :], axis=0)

    cent_history.append(centroids)
```

In [3]:
```
plt.figure(figsize=(8, 8))
for i in range(4):
    distances = cdist(X, cent_history[i])
    labels = distances.argmin(axis=1)

    plt.subplot(2, 2, i + 1)
    plt.plot(X[labels == 0, 0], X[labels == 0, 1], 'bo', label='cluster #1')
    plt.plot(X[labels == 1, 0], X[labels == 1, 1], 'co', label='cluster #2')
    plt.plot(X[labels == 2, 0], X[labels == 2, 1], 'mo', label='cluster #3')
    plt.plot(cent_history[i][:, 0], cent_history[i][:, 1], 'rX')
    plt.legend(loc=0)
    plt.title('Step {:}'.format(i + 1));
```



Como vemos comunmente, en este ejemplo utilizamos la medida de distancia euclidiana (el algoritmo va a converger con cualquier otra metrica). Algunas de las caracteristicas que puedes tomar en cuenta para poder mejorar o cambiar tus resultados es el criterio de convergencia o la medida de distancia que utilizas entre los puntos de los datos y los centroides.

¿Como podemos eleguir el criterio de numero de clusters?

Haciendo contraste con herramientas de aprendizaje supervisado, en donde tenemos problemas de clasificacion o regresion, el agrupamiento requiere mas esfuerzo para elegir el criterio de optimizacion. Usualmente, cuando se trabaja con kmedias, debemos de optimizar la suma de las distancias cuadradas entre las observaciones o puntos y sus centroides. Para esto tenemos la siguiente formula:

$$J(C) = \sum_{k=1}^{K} \sum_{i \in C_k} ||x_i - \mu_k|| \rightarrow \min_C$$

En donde interpretamos a C como el conjunto de clusters a la potencia k. μ_k es el centroide de un cluster C_k.

Esta definicion pareceria estar correcta, sin embargo, sin restriccion, el optimo se alcanza cuando el numero de centroides sea igual al numero de observaciones, por lo que cada uno de los puntos terminaria con su propio cluster. Para evitar sto, tenemos la funcion:

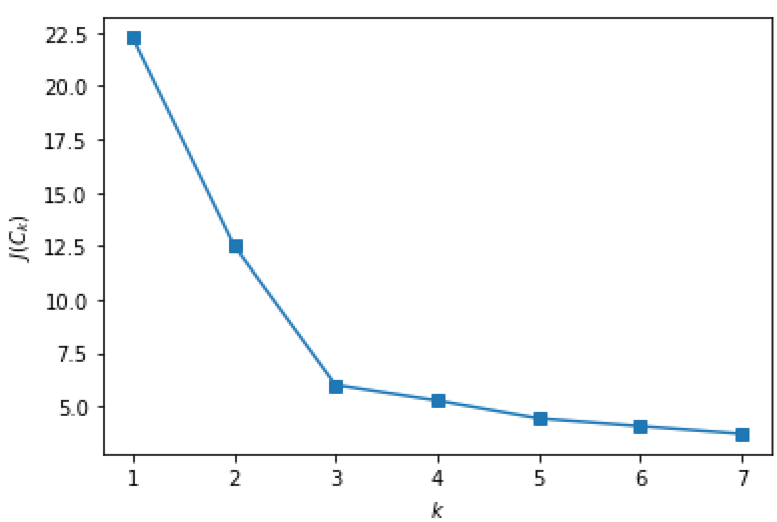
$$D(k) = \frac{|J(C_k) - J(C_{k+1})|}{|J(C_{k-1} - J(C_k))|} \rightarrow \min_k$$

In [4]:
```
from sklearn.cluster import KMeans
```

In [5]:
```
inertia = []
for k in range(1, 8):
    kmeans = KMeans(n_clusters=k, random_state=1).fit(X)
    inertia.append(np.sqrt(kmeans.inertia_))
```

In [6]:
```
plt.plot(range(1, 8), inertia, marker='s');
plt.xlabel('$k$')
plt.ylabel('$J(C_k)$')
```

Out[6]:
```
Text(0, 0.5, '$J(C_k)$')
```

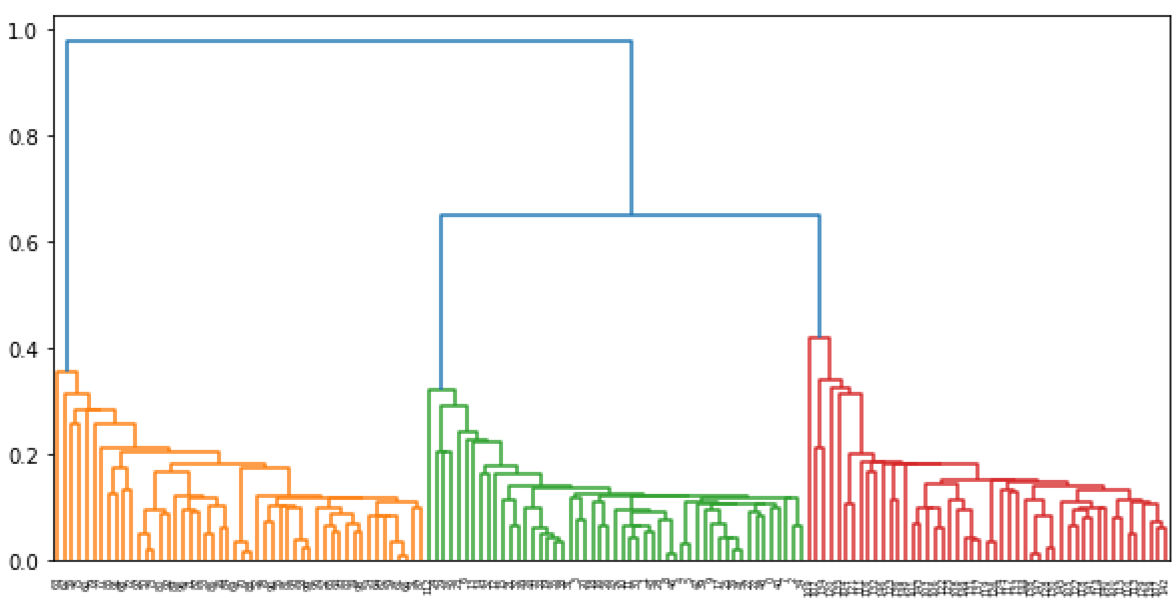


Con esta funcion vemos que $J(C_k)$ llega hasta el punto 3, en donde a partir de ahi no tiene cambios grandes, por lo que podemos inferir que ese es el valor optimo de clusters.

Este tipo de problema es np-dificil, ya que su complejidad va en aumento dependiento de las dimensiones, los clusters y el numero de observaciones. La implementacion de este tipo de problema con sklearn tiene la ventaja de su funcion de inicializacion default nos ayuda a identificar centroides robustos. (Existen elementos de programacion en paralelo que tambien pueden implementarse en esta libreria que mejora el tiempo y capacidad computacional del problema).

Existe otra version del algoritmo llamado agrupacion aglomerativa. Esta se encuentra en una libreria llamada scipy. No entraremos a detalle, pero es una opcion que pueden explorar por su cuenta.

In [7]:
```
from scipy.cluster import hierarchy
from scipy.spatial.distance import pdist

X = np.zeros((150, 2))

np.random.seed(seed=42)
X[:50, 0] = np.random.normal(loc=0.0, scale=.3, size=50)
X[:50, 1] = np.random.normal(loc=0.0, scale=.3, size=50)

X[50:100, 0] = np.random.normal(loc=2.0, scale=.5, size=50)
X[50:100, 1] = np.random.normal(loc=-1.0, scale=.2, size=50)

X[100:150, 0] = np.random.normal(loc=-1.0, scale=.2, size=50)
X[100:150, 1] = np.random.normal(loc=2.0, scale=.5, size=50)

# pdist will calculate the upper triangle of the pairwise distance matrix
distance_mat = pdist(X)
# linkage — is an implementation if agglomerative algorithm
Z = hierarchy.linkage(distance_mat, 'single')
plt.figure(figsize=(10, 5))
dn = hierarchy.dendrogram(Z, color_threshold=0.5)
```



Practica:

Utilizando la libreria de sklearn y la base de datos de iris, realiza el proceso de kmeans.

In [9]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

In [10]:
```
#leer la base de datos
base_iris=pd.read_csv('/content/Iris.csv')
```

In [11]:
```
#ver como esta la base de datos
base_iris.head(10)
```

Out[11]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
5	6	5.4	3.9	1.7	0.4	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa
7	8	5.0	3.4	1.5	0.2	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
9	10	4.9	3.1	1.5	0.1	Iris-setosa

In [14]:
```
#encontrar las diferentes clases de especies
np.unique(base_iris['Species'])
```

Out[14]:
```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

Se tienen 3 tipos de clases distintos.

In [15]:
```
#obtener el tamaño de la base de datos con la que trabajaremos
base_iris.shape
```

Out[15]:
```
(150, 6)
```

In [16]:
```
#informacion de las columnas y validar que no haya valores nulos
base_iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             150 non-null    int64  
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object 
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

No se tienen valores nulos en la base de datos y el tipo de valor asignado a cada una de las varibles es el correcto.

In [18]:
```
#eliminar la columna de ID que no aporta informacion relevante
#DROPPING ID
base_iris_nva= base_iris.drop(['Id'], axis = 1)
```

In [19]:
```
base_iris_nva.shape
```

Out[19]: (150, 5)

In [20]:
```
base_iris_nva.head(10)
```

Out[20]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa
6	4.6	3.4	1.4	0.3	Iris-setosa
7	5.0	3.4	1.5	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
9	4.9	3.1	1.5	0.1	Iris-setosa

In [22]:
```
#encontrar la estadistica decriptiva de la base
base_iris_nva.describe()
```

Out[22]:

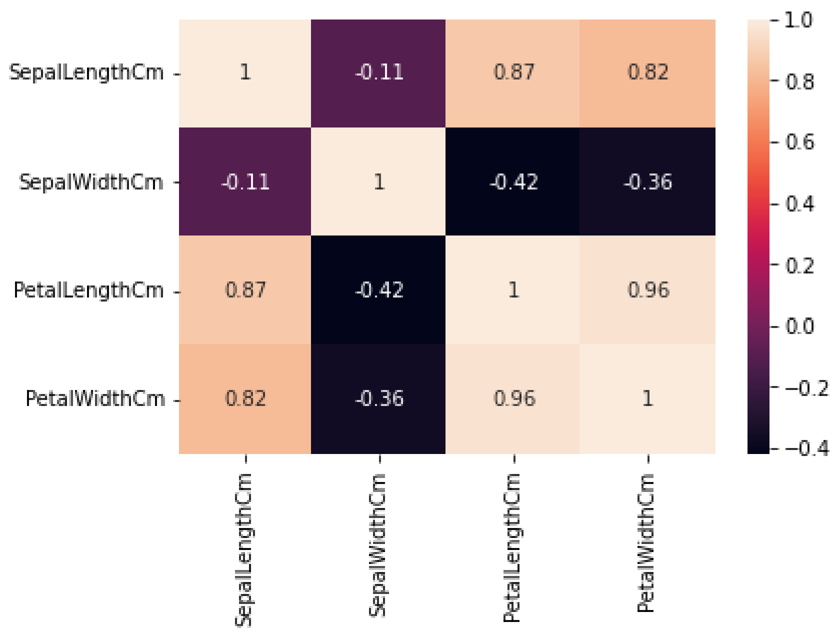
	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

In [23]:
```
#distibucion de la variable categorica
base_iris_nva.Species.value_counts().sort_index()
```

Out[23]:
```
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: Species, dtype: int64
```

In [21]:
```
#encontrar las correlaciones entre las variables
import seaborn as sns
correl=base_iris_nva.corr()
sns.heatmap(correl,annot=True)
```

Out[21]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2eb14e91f0>
```



Se puede observar del análisis de correlación que las variables más correlacionadas se encuentran el ancho del sepalo y el largo del petalo con un coeficiente de correlacion negativo de -0.42.

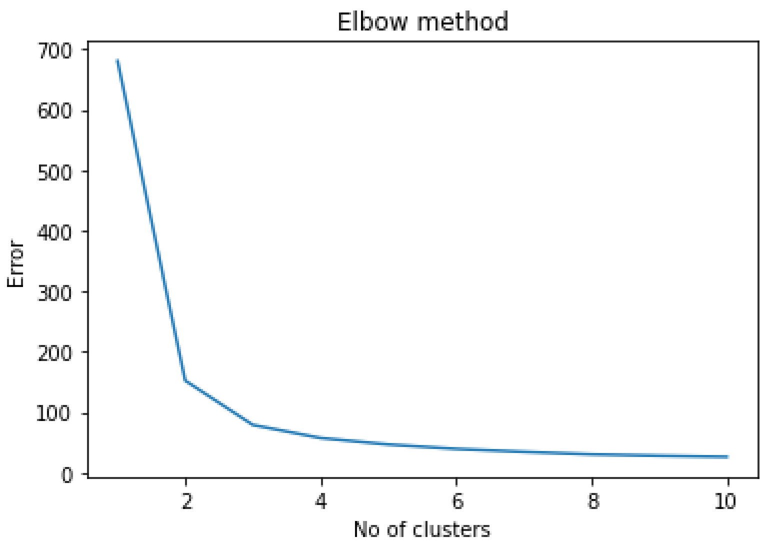
In [25]:
```
#poner en un arreglo las variables numericas
x=base_iris_nva.iloc[:,[0,1,2,3]].values
x
```

```
Out[25]: 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, 0.1],
                [5. , 3.2, 1.2, 0.2],
                [5.5, 3.5, 1.3, 0.2],
                [4.9, 3.1, 1.5, 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]])
```

In [26]:
```
#implementar el grafico del codo para elegir el numero optimo de k clusters
Error =[]
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i).fit(x)
    kmeans.fit(x)
    Error.append(kmeans.inertia_)
import matplotlib.pyplot as plt
plt.plot(range(1, 11), Error)
plt.title('Elbow method')
plt.xlabel('No of clusters')
plt.ylabel('Error')
plt.show()
```



Se puede observar que el grafico se quiebra en el numero 3 donde ya el error tiene una tasa de cambio muy pequeña a medida que se tienen mas clusters. Por lo tanto, elegiremos k=3 para generar el modelo de k medias.

In [27]:
```
#aplicar el modelo de kmedias con el numero optimo de clusters
kmeans=KMeans(n_clusters=3)
y_kmeans=kmeans.fit_predict(x)
print(y_kmeans)
```

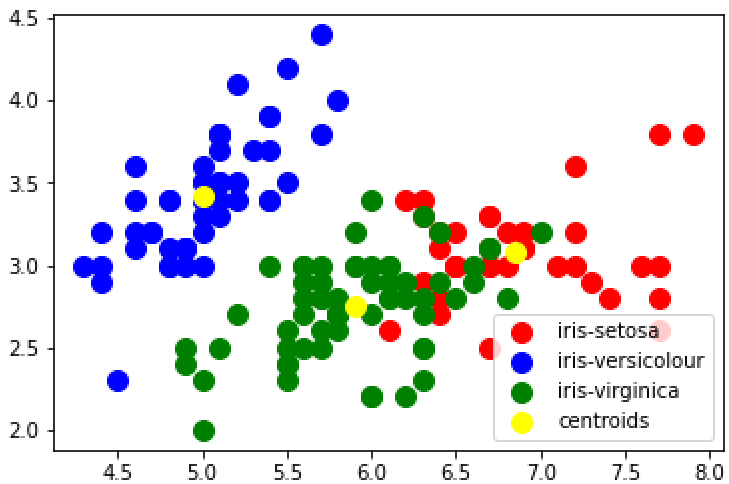
```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 2 0 0 0 0 2 0 0 0 0
 0 0 2 2 0 0 0 0 2 0 2 0 2 0 0 2 2 0 0 0 0 0 2 0 0 0 0 2 0 0 0 2 0 0 0 2 0
 0 2]
```

In [28]:
```
#visualizar los centroides de los clusters
kmeans.cluster_centers_
```

Out[28]:
```
array([[6.85      , 3.07368421, 5.74210526, 2.07105263],
       [5.006     , 3.418     , 1.464     , 0.244     ],
       [5.9016129 , 2.7483871 , 4.39354839, 1.43387097]])
```

In [31]:
```
#ver de forma grafica como se formaron los clusters
plt.scatter(x[y_kmeans== 0,0], x[y_kmeans==0,1], s=100, c='red',label='iris-setosa')
plt.scatter(x[y_kmeans== 1,0], x[y_kmeans==1,1], s=100, c='blue',label='iris-versicolour')
plt.scatter(x[y_kmeans== 2,0], x[y_kmeans==2,1], s=100, c='green',label='iris-virginica')

#agregar los centroides de los clusters
plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1], s=100, c='yellow',label='centroids')
plt.legend()
```

Out[31]: <matplotlib.legend.Legend at 0x7f2eb0e3d9a0>



En el grafico se pueden observar los 3 clusters que corresponden a las 3 categorias de especies que se tenian en la variable de species. En el gráfico se alcanza a distinguir que uno de los clusters (azul) se encuentra más definido al estar separado de los otros dos clusters. Los clusters verde y rojo se encuentran no tan definidos al no poder marcar una linea divisoria más clara entre ambos.