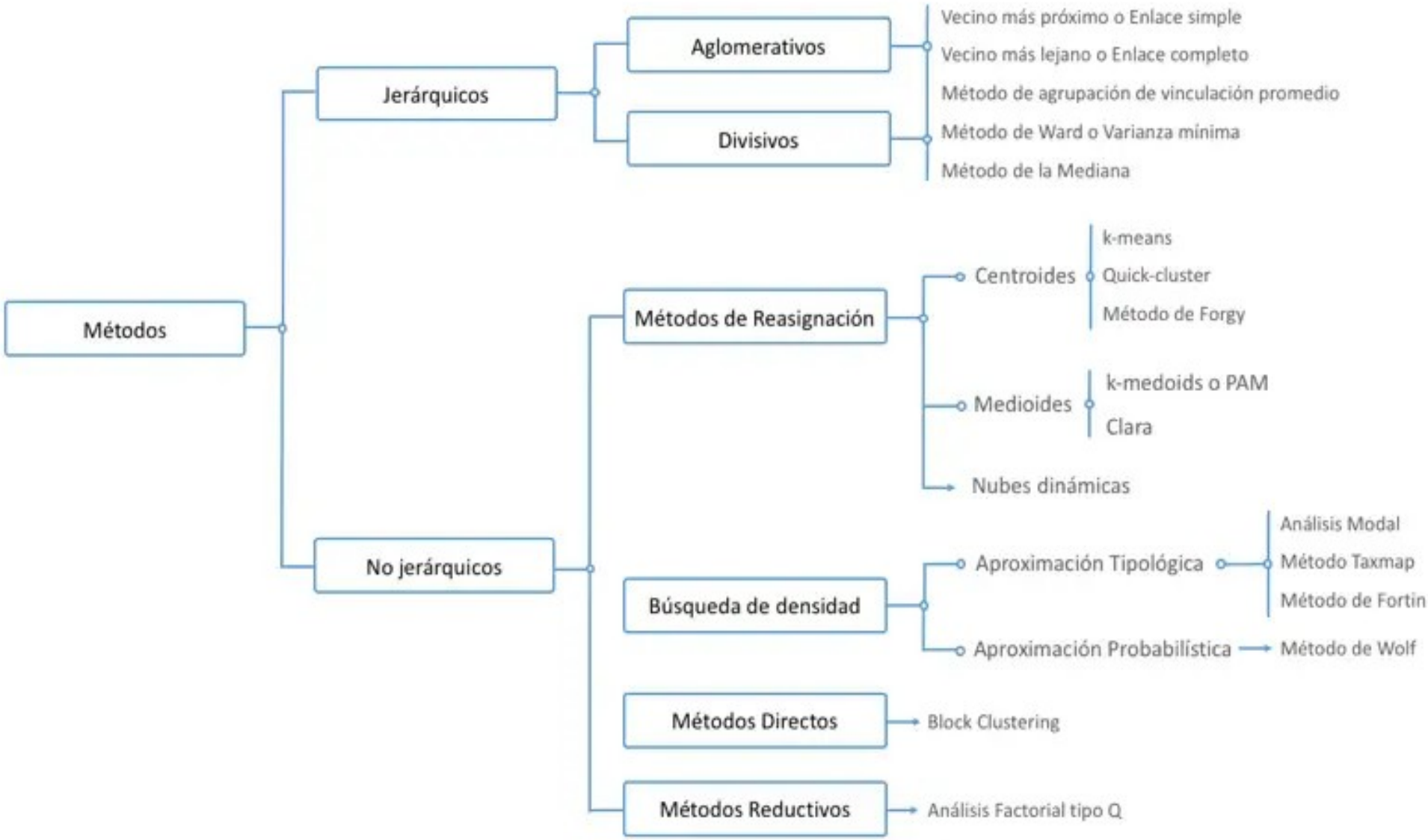
APRENDIZAJE NO SUPERVISADO

Menú:

@ Agrupamiento por semejanza
@ Agrupamiento jerárquico
@ Agrupamiento por densidad
@ Reducción de dimensionalidad


Métodos
Jerárquicos
Aglomerativos
Divisivos
Vecino más próximo o Enlace simple
Vecino más lejano o Enlace completo
Método de agrupación de vinculación promedio
Método de Ward o Varianza mínima
Método de la Mediana
No jerárquicos
Métodos de Reasignación
Centroides
k-means
Quick-cluster
Método de Forgy
Medioides
k-medoids o PAM
Clara
Nubes dinámicas
Búsqueda de densidad
Aproximación Tipológica
Análisis Modal
Método Taxmap
Método de Fortin
Aproximación Probabilística
Método de Wolf
Métodos Directos
Block Clustering
Métodos Reductivos
Análisis Factorial tipo Q

AGRUPAMIENTO por SEMEJANZA (CLUSTERING)

- TÉCNICA DE APRENDIZAJE NO SUPERVISADO
- EL OBJETIVO ES ENCONTRAR PATRONES DE SIMILITUD PARA CONFORMAR GRUPOS ENTRE LOS DATOS
- SE UTILIZA CUANDO NO HAY UNA SEPARACIÓN PREVIA ENTRE DISTINTAS CLASES PERO SE SOSPECHA QUE ÉSTAS EXISTEN

EJEMPLOS DE USO:

*SEGMENTACIÓN DE CLIENTES

* CATEGORIZAR INVENTARIO

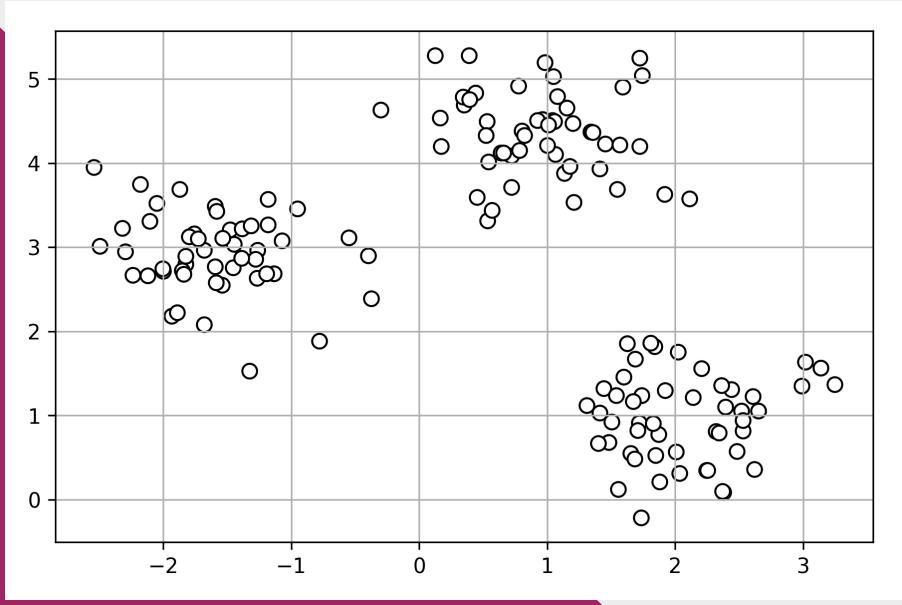
* DETECTAR ANOMALÍAS EN LA WEB

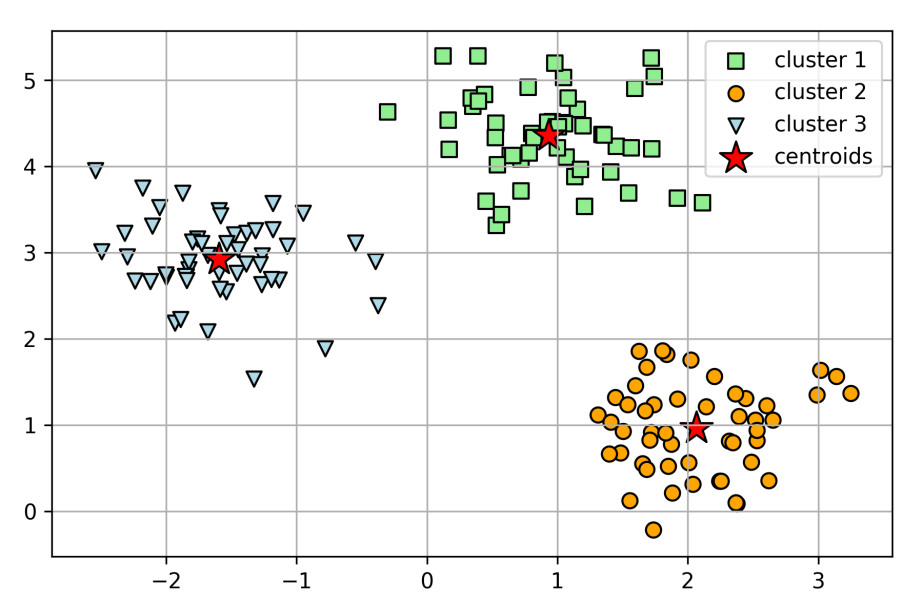
- * CATEGORIZAR ZONAS CON SENSORAMIENTO REMOTO

ALGORITMO K-MEANS

- PERTENECE A LA LIBRERÍA SCIKIT LEARN

  https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
- REQUIERE QUE SE LE INDIQUE LA CANTIDAD DE CLUSTERS A SEPARAR
- class sklearn.cluster.KMeans(n_clusters=8, *, init='k-means++', n_init=10, max_iter=300, tol=0.0001, verbose=0, random_state=None, copy_x=True, algorithm='lloyd')





EJEMPLO: AGRUPAR EL DATASET DE FLORES IRIS

Importing Libraries

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets


#Visualization Libraries
import seaborn as sns
import matplotlib.pyplot as pyplot
sns.set(style="darkgrid")

import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
```

Loading Iris dataset from csv

In [4]:
```
df_iris = pd.read_csv('Iris.csv')
```

In [9]:
```
df_iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

Description of Dataset

In [12]:
```
df_iris.describe()
```

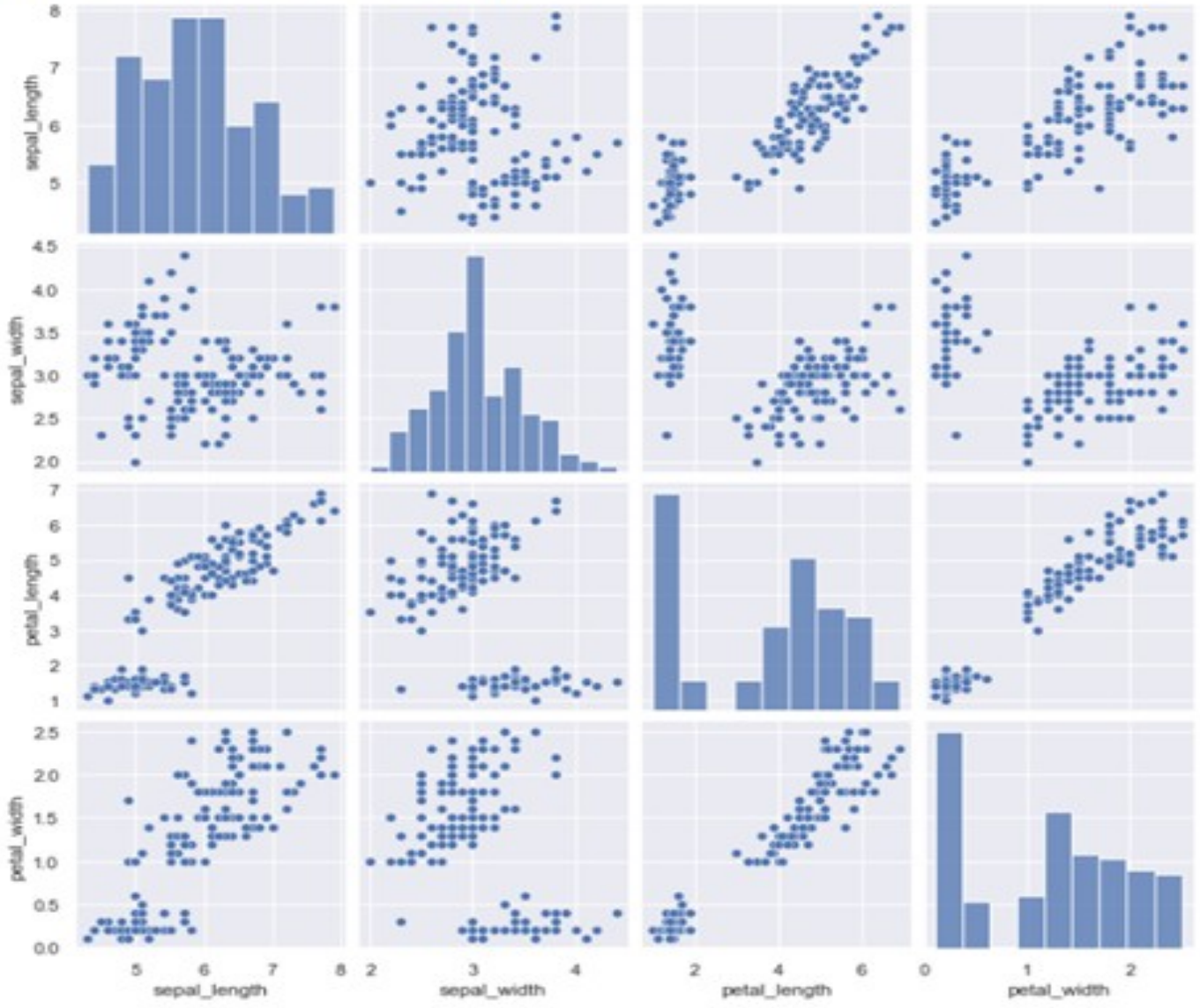
Out[12]:

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
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

Exploratory Data Analysis

```
In [15]: sns.pairplot(df_iris)
```

```
Out[15]: <seaborn.axisgrid.PairGrid at 0x1c1038fb790>
```

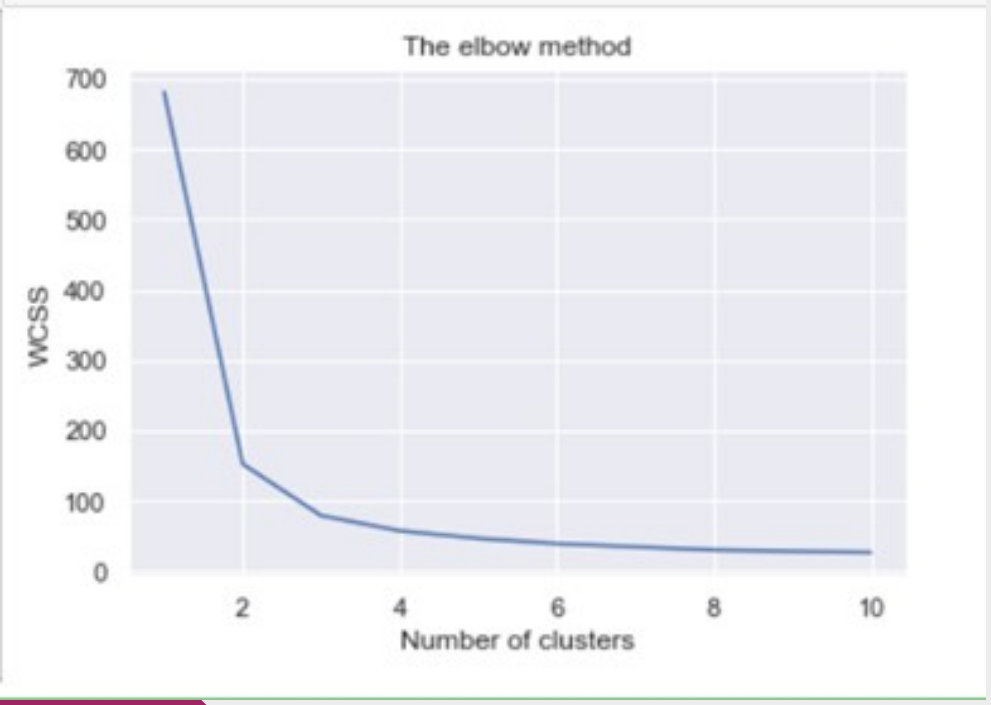


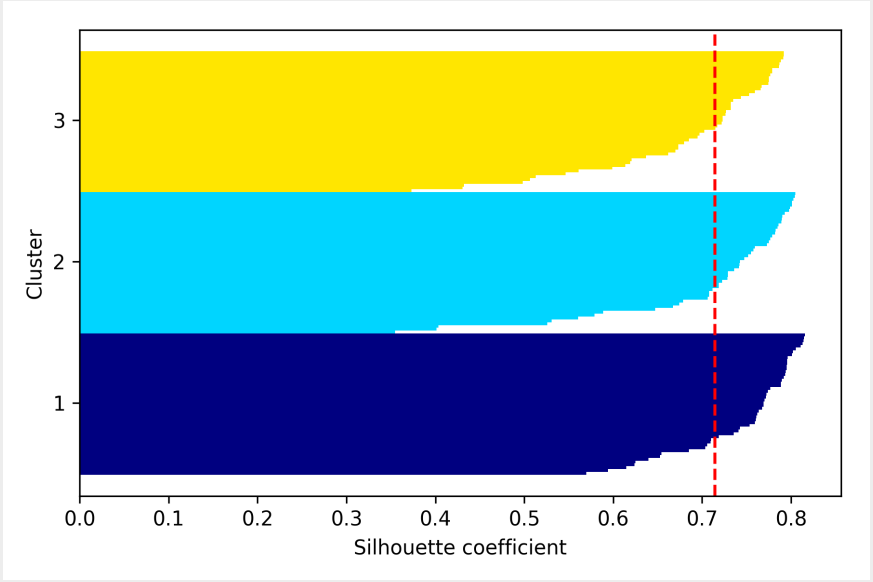
In [13]:
```
x = df_iris.iloc[:, [0,1,2,3]].values
```

In [6]:
```
#Finding the optimum number of clusters for k-means classification
from sklearn.cluster import KMeans
wcss = []

for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
    kmeans.fit(x)
    wcss.append(kmeans.inertia_)

#Plotting the results onto a line graph, allowing us to observe 'The elbow'
plt.plot(range(1, 11), wcss)
plt.title('The elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS') #within cluster sum of squares
plt.show()
```





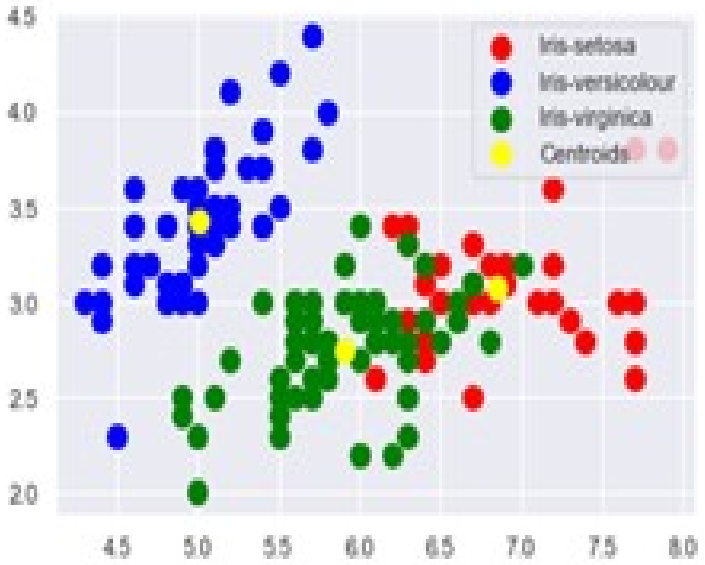
In [7]:
```
#Applying kmeans to the dataset / Creating the kmeans classifier
kmeans = KMeans(n_clusters = 3, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
y_kmeans = kmeans.fit_predict(x)
```

In [8]:
```
#Visualising the 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')

#Plotting the centroids of the clusters
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,1], s = 100, c = 'yellow', label = 'Centroids')

plt.legend()
```

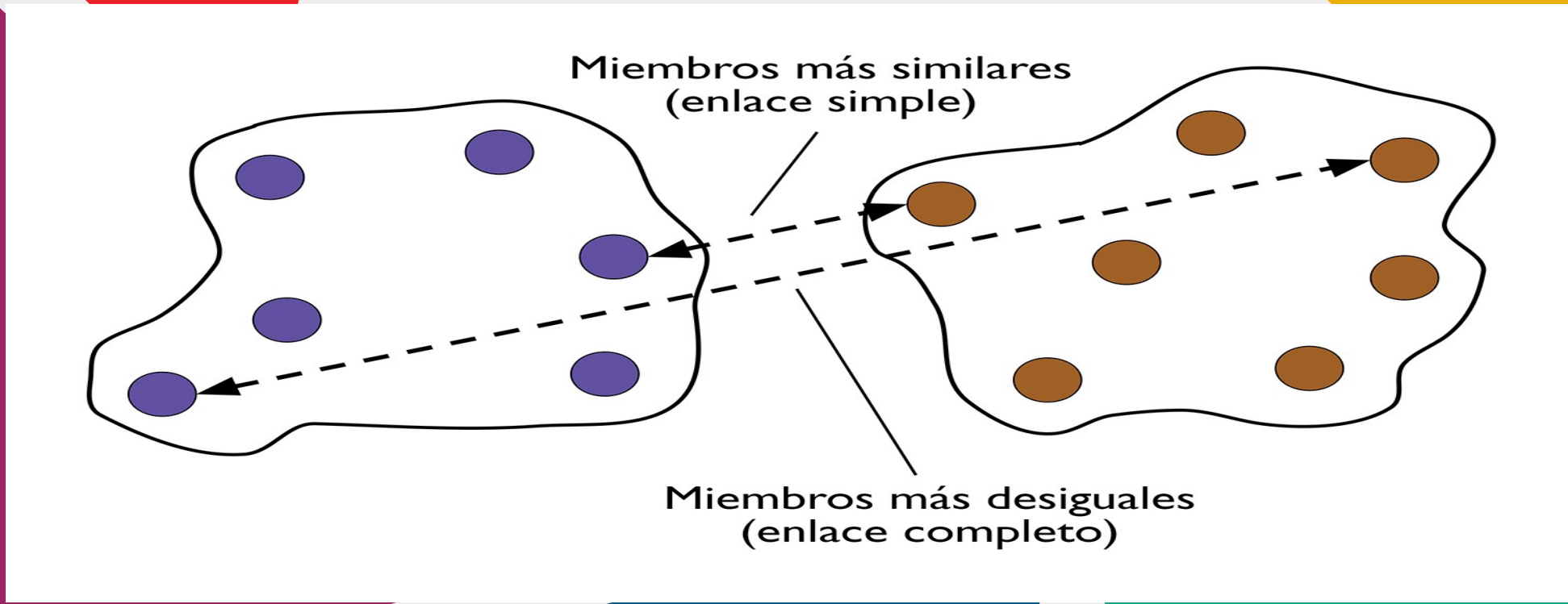
Out[8]:
```
<matplotlib.legend.Legend at 0x1c1038ab280>
```



AGRUPAMIENTO JERÁRQUICO

- 2 enfoques: DIVISIVO y AGLOMERATIVO
- DIVISIVO: comienza tomando todas las muestras como un único cluster y procede iterativamente a dividirlo hasta que cada grupo contiene una única muestra
- AGLOMERATIVO: comienza de a una muestra y va agrupando por semejanza

- class sklearn.cluster.AgglomerativeClustering(n_clusters=2, *, affinity='deprecated', metric=None, memory=None, connectivity=None, compute_full_tree='auto', linkage='ward', distance_threshold=None, compute_distances=False)



AGRUPAMIENTO POR DENSIDAD (DBSCAN)

No analiza por jerarquía ni por proximidad, sino por densidad

Distingue en 3 tipos a cada muestra:

* punto núcleo: si hay un nro mínimo de puntos en un radio E

* punto borde: tiene menos que el nro mínimo de ptos en un radio E pero está dentro del radio de un pto núcleo

* punto ruidoso: ni núcleo ni borde

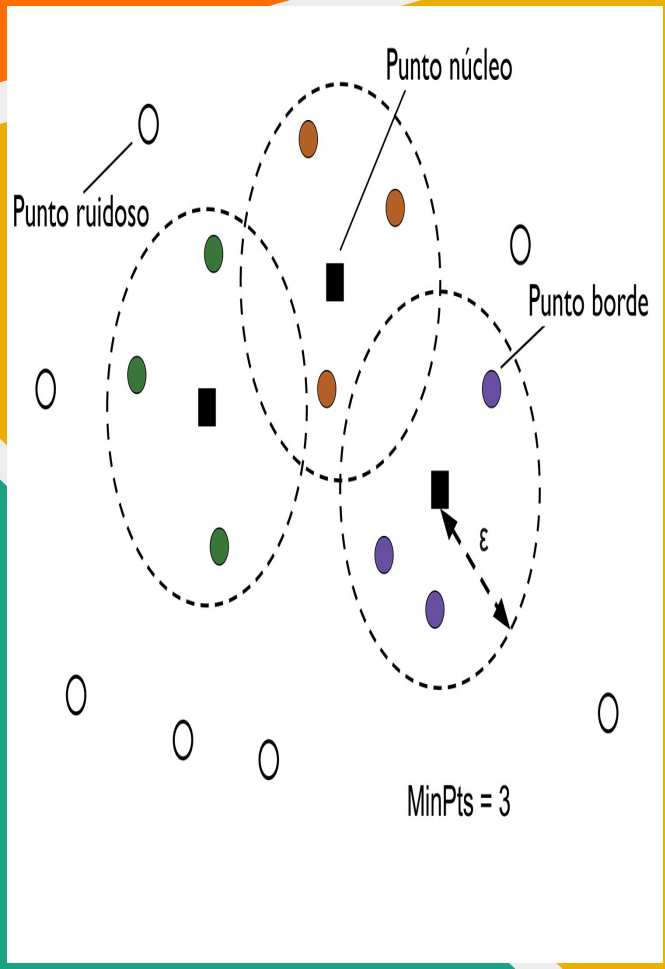
Una vez clasificadas las muestras, forma un grupo separado por cada pto núcleo grupo conectado de ptos núcleo y finalmente, asigna los puntos borde a cada grupo

class sklearn.cluster.DBSCAN(eps=0.5, *, min_samples=5, metric='euclidean', metric_params=None, algorithm='auto', leaf_size=30, p=None, n_jobs=None)

Inconvenientes:

* 2 hiperparámetros para asignar

* con muchas dimensiones lo afecta la ‘maldición de la dimensionalidad’



REDUCCIÓN DE DIMENSIONALIDAD (PCA)

- Permite reducir la dimensión del espacio de características ubicando las direcciones de mayor varianza y descartando las que menos aportan
- Pasos:

  * Estandarizar características

  * Construir matriz de covarianza

  * Obtener autovalores y autovectores de la matriz de covarianza

  * Ordenar autovalores en descendente para identificar sus autovectores

  * Separar los k autovectores

  * Reproyectar las características en el nuevo espacio ‘reducido’

- class sklearn.decomposition.PCA(n_components=None, *, copy=True, whiten=False, svd_solver='auto', tol=0.0, iterated_power='auto', n_oversamples=10, power_iteration_normalizer='auto', random_state=None)