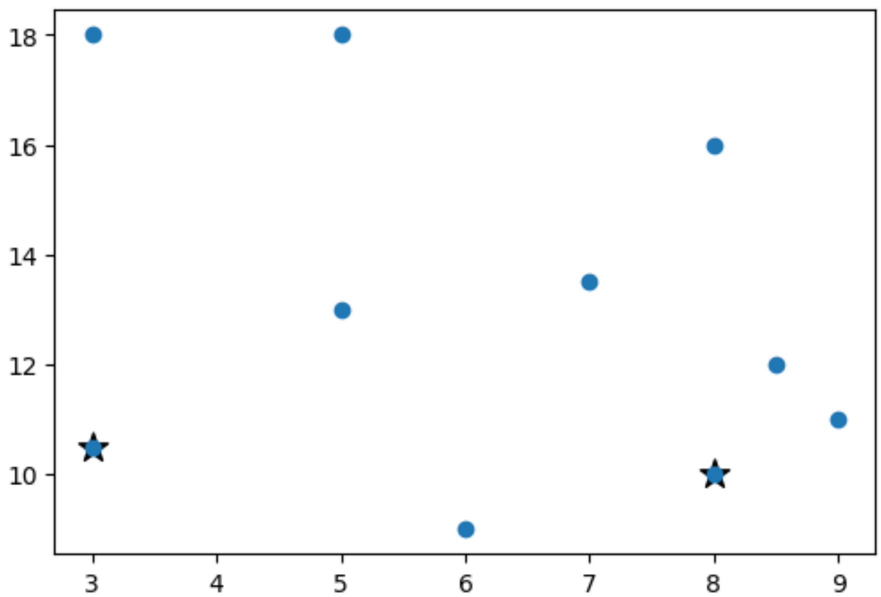
```python
import numpy as np
from matplotlib import pyplot as plt
plt.rcParams['figure.figsize'] = (6, 4)
```

```python
np.linalg.norm(np.array([8,10]) - np.array([3,10.5]))
```

```
np.float64(5.024937810560445)
```

```python
data = np.array([[8,10],[3,10.5],[7,13.5],[5,18],[5,13],[6,9],[9,11],[3,18],[8.5,12],[8,16]]
#C = np.array([ [8,10],[3,10.5],[7,13.5] ])
C = np.array([ [8,10],[3,10.5] ])
# Gráfica
plt.scatter(C[:,0], C[:,1],marker='*',s=150,c='k') # Centroides
plt.scatter(data[:,0],data[:,1]) # Datos
```

```
<matplotlib.collections.PathCollection at 0x78fd08c49b90>
```




```python
def dist(a, b):
  return np.linalg.norm(a[:,np.newaxis] - b, axis=2)
```

```python
x = np.array([[8,10]])
y = np.array([3,10.5])
dist(x, y)
```

```
array([[5.02493781]])
```

```python
x.shape, x[:,np.newaxis].shape
```

```
((1, 2), (1, 1, 2))
```

```python
# Asignar puntos a cada grupo
distancias = dist(data, C)
grupos = np.argmin(distancias, axis=1)
print(grupos, distancias)
```

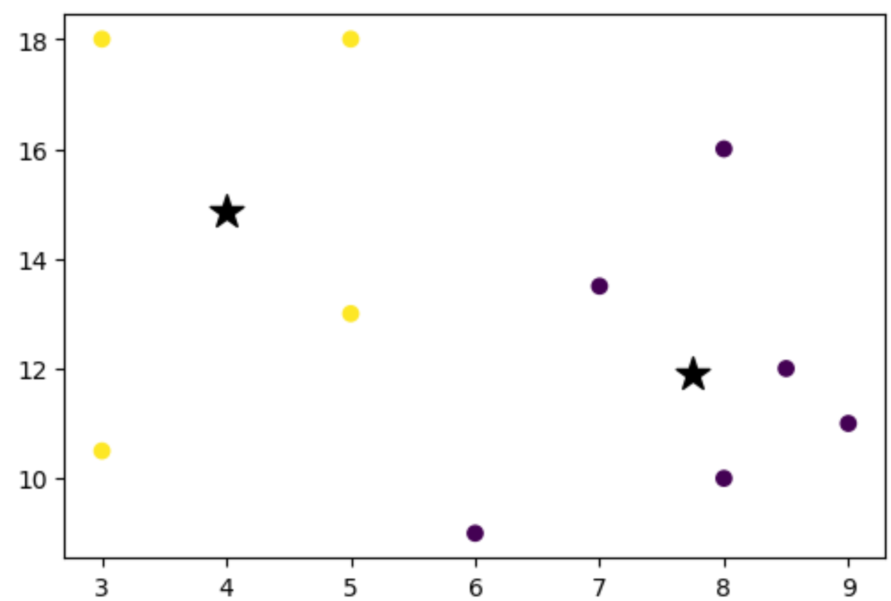
```
[0 1 0 1 1 0 0 1 0 0] [[0.         5.02493781]
 [5.02493781 0.        ]
 [3.64005494 5.        ]
 [8.54400375 7.76208735]
 [4.24264069 3.20156212]
 [2.23606798 3.35410197]
 [1.41421356 6.02079729]
 [9.43398113 7.5       ]
 [2.06155281 5.70087713]
 [6.         7.43303437]]
```

```python
# Actualizar centroides
# list comprehension [ oper(e) for e in list /if cond/ ]
for i in range(len(C)):
    puntos=[ data[j] for j in range(len(data)) if grupos[j]==i ]
    #print(i,np.array(puntos))
    C[i] = np.mean(puntos, axis=0)
print(C)
```

```
[[ 7.75       11.91666667]
 [ 4.         14.875     ]]
```

```python
# Gráfica
plt.scatter(C[:,0], C[:,1], marker='*', s=200, c='k')
plt.scatter(data[:,0], data[:,1], c=grupos)
```

```
<matplotlib.collections.PathCollection at 0x78fd0831c950>
```




```python
''' scikit-learn '''
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
plt.rcParams['figure.figsize'] = (6,4)
plt.style.use('ggplot')
```

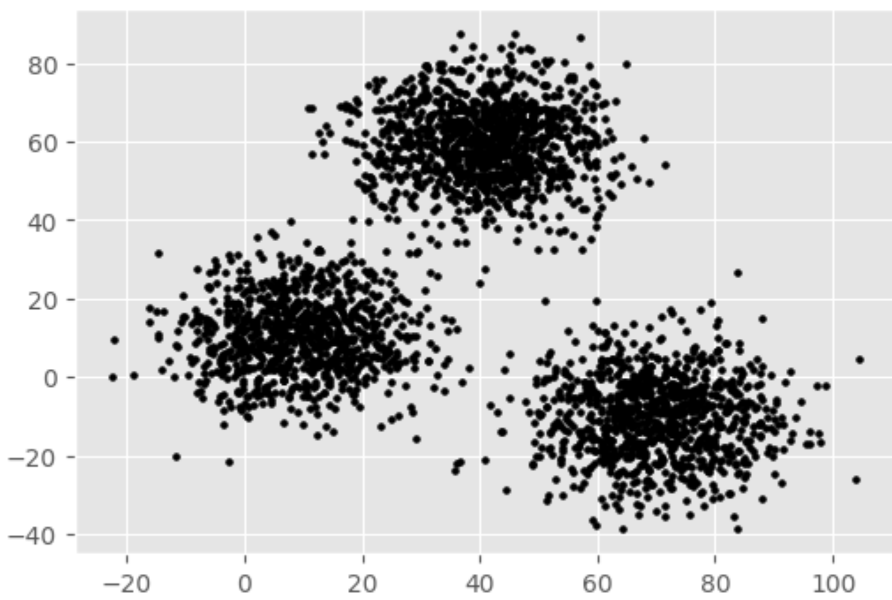
```
# Conjunto de datos xclara
data = pd.read_csv('https://bit.ly/3BxsIDR')
print(data.shape)
data.tail()
```

```
(3000, 3)
```

	rownames	V1	V2
2995	2996	85.65280	-6.461061
2996	2997	82.77088	-2.373299
2997	2998	64.46532	-10.501360
2998	2999	90.72282	-12.255840
2999	3000	64.87976	-24.877310

```
# Gráfica
x1 = data['V1'].values
x2 = data.V2.values
X = np.array(list(zip(x1,x2)))
plt.scatter(x1, x2, c='black', s=7)
```

```
<matplotlib.collections.PathCollection at 0x78fcd9a8eb50>
```

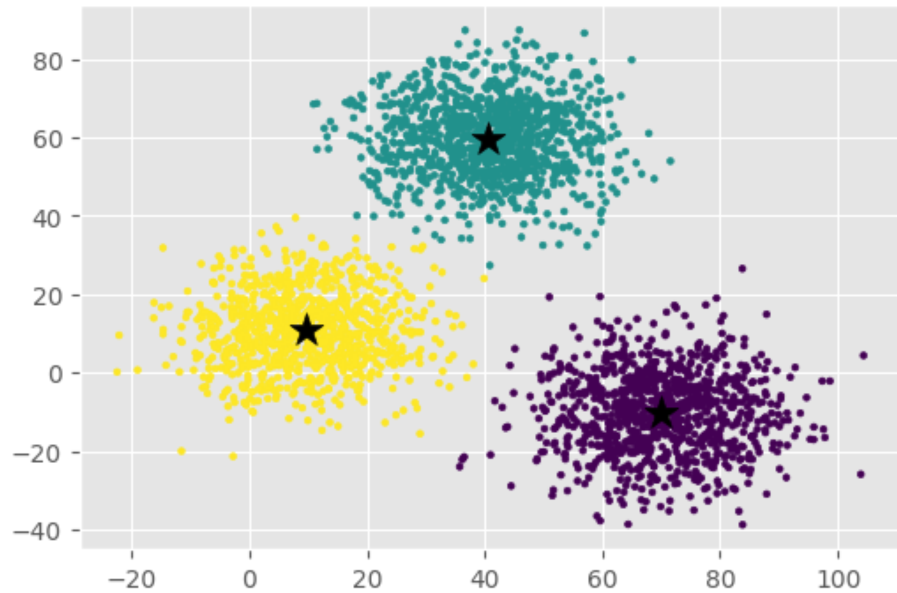



```
from sklearn.cluster import KMeans
# Inicializar con número de grupos
km = KMeans(n_clusters=3)
# Ajuste: ejecutar el algoritmo de aprendizaje -> los centroides de cada clase
km = km.fit(X)
# Etiquetas de cada clase: obtener la clase a la que pertenece cada punto
y = km.predict(X)
# Centroides
C_ = km.cluster_centers_
print(C_)
```

```
[[ 69.92418447 -10.11964119]
 [ 40.68362784  59.71589274]
 [  9.4780459   10.686052  ]]
```

```python
plt.scatter(X[:, 0], X[:, 1], c=y, s=7)
plt.scatter(C_[:, 0], C_[:, 1], marker='*', s=200, c='k')
```

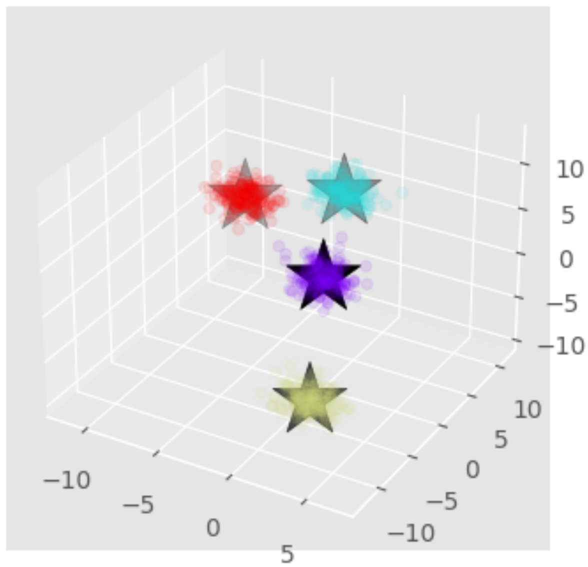
```
<matplotlib.collections.PathCollection at 0x78fcc1710590>
```




```python
# Ejemplo 3D
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
plt.rcParams['figure.figsize'] = (9, 4)
# 4 nubes de puntos de 3D
X, y = make_blobs(n_samples=800, n_features=3, centers=4, random_state=42)
```

```python
# Initializing KMeans
kmeans = KMeans(n_clusters=4, random_state=42)
# Fitting with inputs
kmeans = kmeans.fit(X)
# Predicting the clusters
y = kmeans.predict(X)
# Getting the cluster centers
C = kmeans.cluster_centers_
```

```python
fig, ax = plt.subplots(subplot_kw={"projection":"3d"})
ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=y, alpha=0.1, cmap='rainbow')
ax.scatter(C[:, 0], C[:, 1], C[:, 2], marker='*', c='k', s=1000)
plt.show()
```

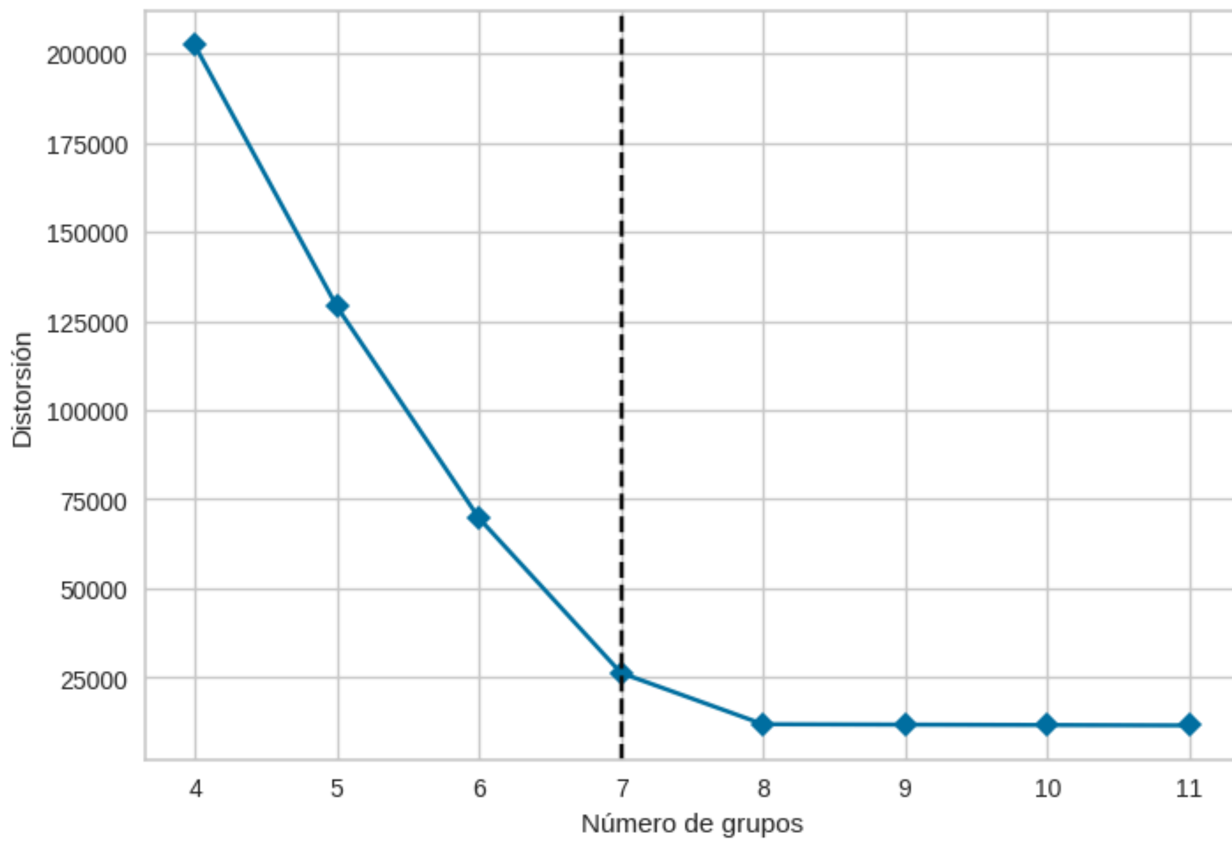



```
# ¿Cómo saber el valor inicial para K? => Gráfica del codo
# Importamos bibliotecas
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import datasets
import seaborn as sns; sns.set()  # Para el estilo de los gráficos
```

```
#!pip install yellowbrick
```

```
# Elección de k con la gráfica de codo KElbowVisualizer
# https://www.scikit-yb.org/en/latest/api/cluster/elbow.html
# Por omisión la métrica es "distortion", que es la suma de las
# distancias cuadráticas de cada punto con el centro de su grupo
from matplotlib import pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
from yellowbrick.cluster import KElbowVisualizer
# Generamos 8 nubes de puntos de dimensión 12
X, y = make_blobs(n_samples=1000, n_features=12, centers=8, random_state=42)

vis = KElbowVisualizer(KMeans(random_state=42), k=(4,12), timings=False)
vis.fit(X)
plt.xlabel('Número de grupos')
plt.ylabel('Distorsión')
plt.show()
```




```
# KMeans in depth
# https://github.com/jakevdp/PythonDataScienceHandbook/blob/master/notebooks/05.11-K-Means.i
```

+ Código + Texto

```
# BIBLIOTECA
#!pip install kmodes
```

```
# bibliotecas
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from kmodes.kmodes import KModes
```

```
data = np.array([['x', 'y', 'z'],
                 ['y', 'z', 'x'],
                 ['z', 'x', 'x'],
                 ['y', 'z', 'z'],
                 ['x', 'z', 'y'],
                 ['z', 'y', 'x'],
                 ['x', 'x', 'y'],
                 ['z', 'y', 'x']])
# modelo con 2 grupos
# n_init Number of time the k-modes algorithm will be run with different
#        centroid seeds. The final results will be the best output of
#        n_init consecutive runs in terms of cost
km = KModes(n_clusters=2,init='random',n_init=5,verbose=False)
grupos = km.fit_predict(data)
grupos
```

```
array([0, 0, 1, 0, 0, 1, 0, 1], dtype=uint16)
```

```
km.cluster_centroids_
```

```
array([['x', 'z', 'y'],
       ['z', 'y', 'x']], dtype='<U1')
```

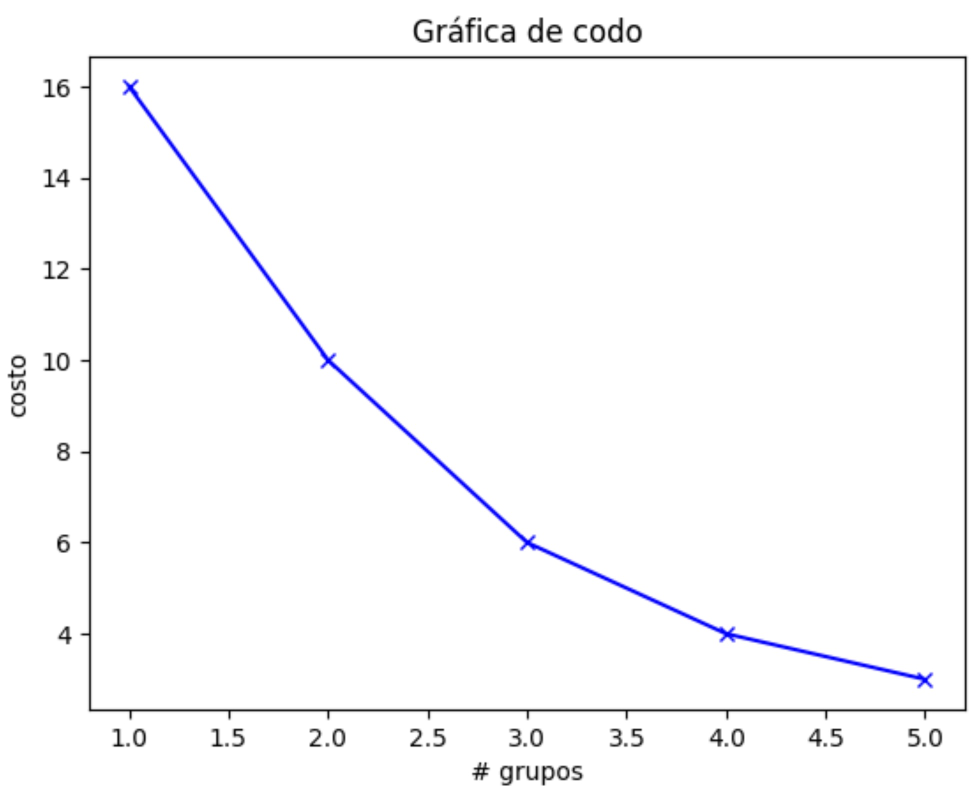
```
# bibliotecas
import pandas as pd
import numpy as np
from kmodes.kmodes import KModes
import matplotlib.pyplot as plt
```

```
# datos
col_cabello = np.array(['rubio', 'castaño', 'pelirrojo', 'negro', 'castaño', 'negro', 'peli
col_ojos = np.array(['azul', 'gris', 'verde', 'café', 'azul', 'gris', 'azul', 'café'])
tipo_cabello = np.array(['lacio', 'chino', 'ondulado', 'ondulado', 'chino', 'chino', 'ondula
personas = ['P1','P2','P3','P4','P5','P6','P7','P8']
data = pd.DataFrame({'person':personas, 'col_cabello':col_cabello, 'col_ojos':col_ojos, 'ti
data = data.set_index('person')
data
```

person	col_cabello	col_ojos	tipo_cabello
P1	rubio	azul	lacio
P2	castaño	gris	chino
P3	pelirrojo	verde	ondulado
P4	negro	café	ondulado
P5	castaño	azul	chino
P6	negro	gris	chino
P7	pelirrojo	azul	ondulado
P8	rubio	café	lacio

```
# Gráfica del codo
cost = []
K = range(1,6)
for num_clusters in list(K):
  kmode = KModes(n_clusters=num_clusters,init='random',n_init=5,verbose=False)
  kmode.fit_predict(data)
  cost.append(kmode.cost_)

plt.plot(K, cost, 'bx-')
plt.xlabel('# grupos')
plt.ylabel('costo')
plt.title('Gráfica de codo')
plt.show()
```



```
# modelo con 3 grupos
kmode = KModes(n_clusters=3, init="random", n_init=5, verbose=False)
grupos = kmode.fit_predict(data)
grupos
```

```
array([0, 2, 1, 0, 2, 2, 1, 0], dtype=uint16)
```

```
kmode.cluster_centroids_
```

```
array([['rubio', 'café', 'lacio'],
       ['pelirrojo', 'azul', 'ondulado'],
       ['castaño', 'gris', 'chino']], dtype='<U9')
```

```
data.insert(0, "grupo", grupos)
data
```

	grupo	col_cabello	col_ojos	tipo_cabello
person				
P1	0	rubio	azul	lacio
P2	2	castaño	gris	chino
P3	1	pelirrojo	verde	ondulado
P4	0	negro	café	ondulado
P5	2	castaño	azul	chino
P6	2	negro	gris	chino
P7	1	pelirrojo	azul	ondulado
P8	0	rubio	café	lacio

```
# Jerárquico
# Ejemplo 1
# bibliotecas
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
# datos
#https://archive.ics.uci.edu/ml/machine-learning-databases/00292/Wholesale%20customers%20dat
url = 'https://bit.ly/2COHM14'
data = pd.read_csv(url)
data.head(2)
```

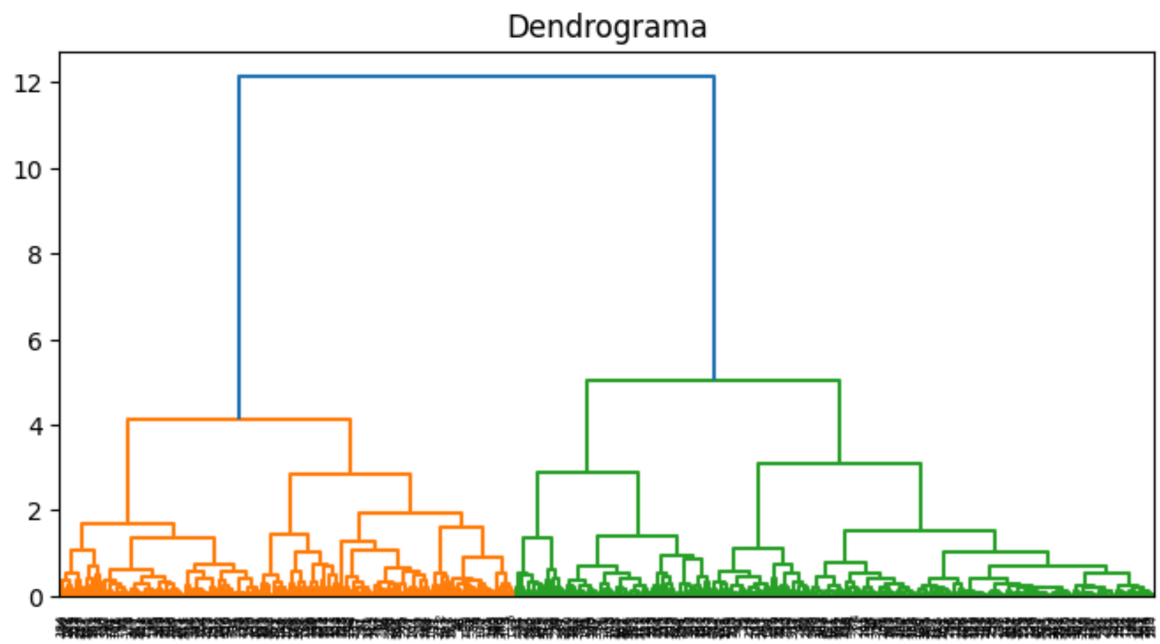
	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	2	3	12669	9656	7561	214	2674	1338
1	2	3	7057	9810	9568	1762	3293	1776

```
# Preprocesamiento : Escalar los valores de las columnas
from sklearn.preprocessing import normalize
data_scaled = normalize(data)
data_scaled = pd.DataFrame(data_scaled, columns=data.columns)
data_scaled.head(2)
```

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	0.000112	0.000168	0.708333	0.539874	0.422741	0.011965	0.149505	0.074809
1	0.000125	0.000188	0.442198	0.614704	0.599540	0.110409	0.206342	0.111286

```
#data_scaled.min(), data_scaled.max()
```

```
import scipy.cluster.hierarchy as sch
# Dendrograma
plt.figure(figsize=(8,4))
plt.title("Dendrograma")
dend = sch.dendrogram(sch.linkage(data_scaled, method='ward'))
```

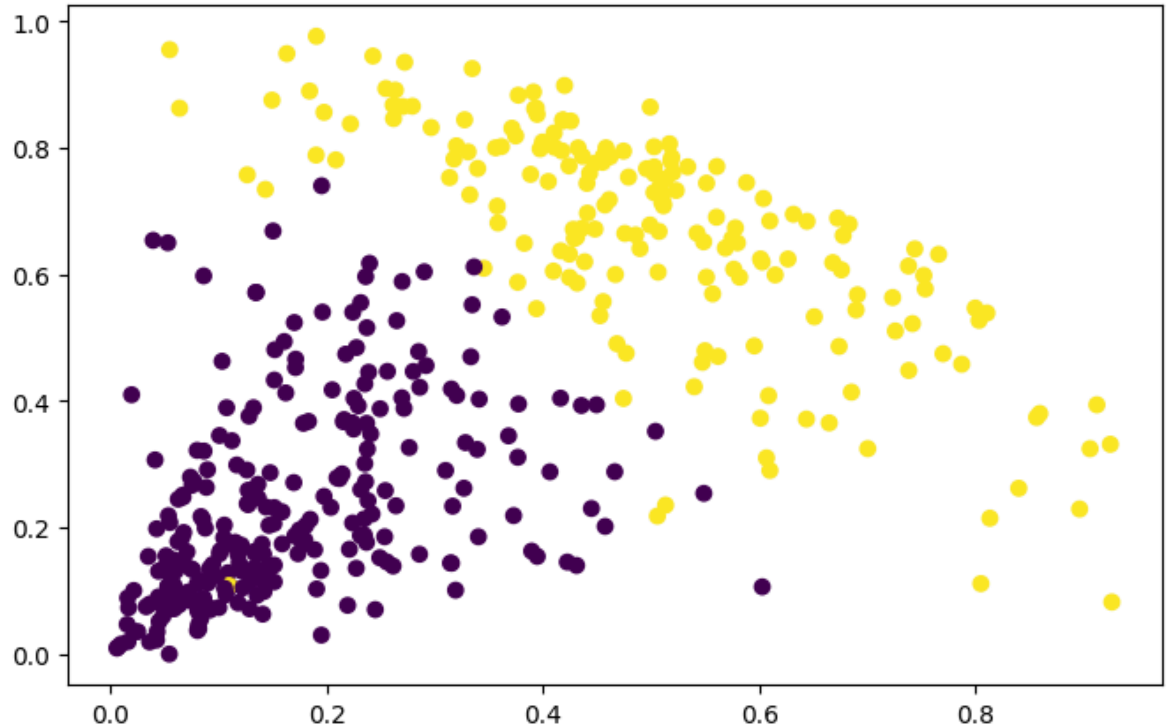



```python
from sklearn.cluster import AgglomerativeClustering
cluster = AgglomerativeClustering(n_clusters=2, metric='euclidean', linkage='ward')
cluster.fit_predict(data_scaled)
```

```
array([1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1,
       1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1,
       0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1,
       0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0,
       0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1])
```

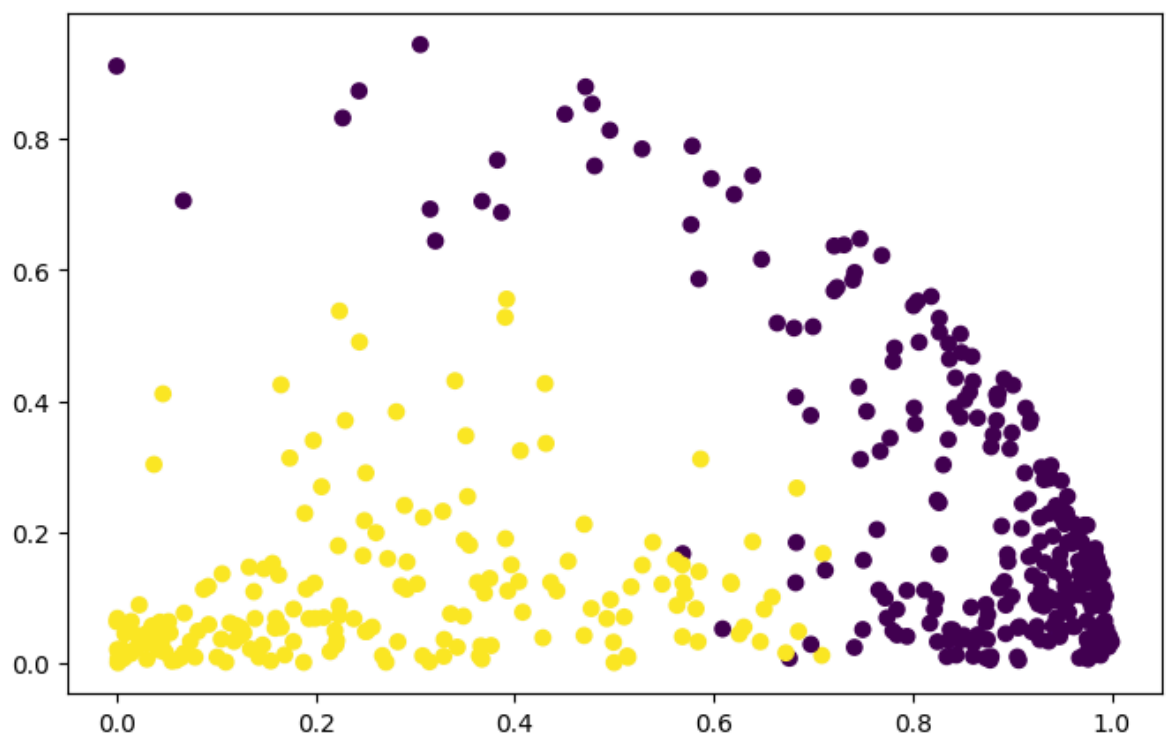
```python
#Visualización de grupos
plt.figure(figsize=(8, 5))
plt.scatter(data_scaled.Milk,data_scaled.Grocery,c=cluster.labels_)
```

<matplotlib.collections.PathCollection at 0x7fd355f82490>



```
plt.figure(figsize=(8, 5))
plt.scatter(data_scaled.Fresh, data_scaled.Frozen, c=cluster.labels_)
```

<matplotlib.collections.PathCollection at 0x7fd355aadc90>



```
# Ejemplo 2
# Bibliotecas
```

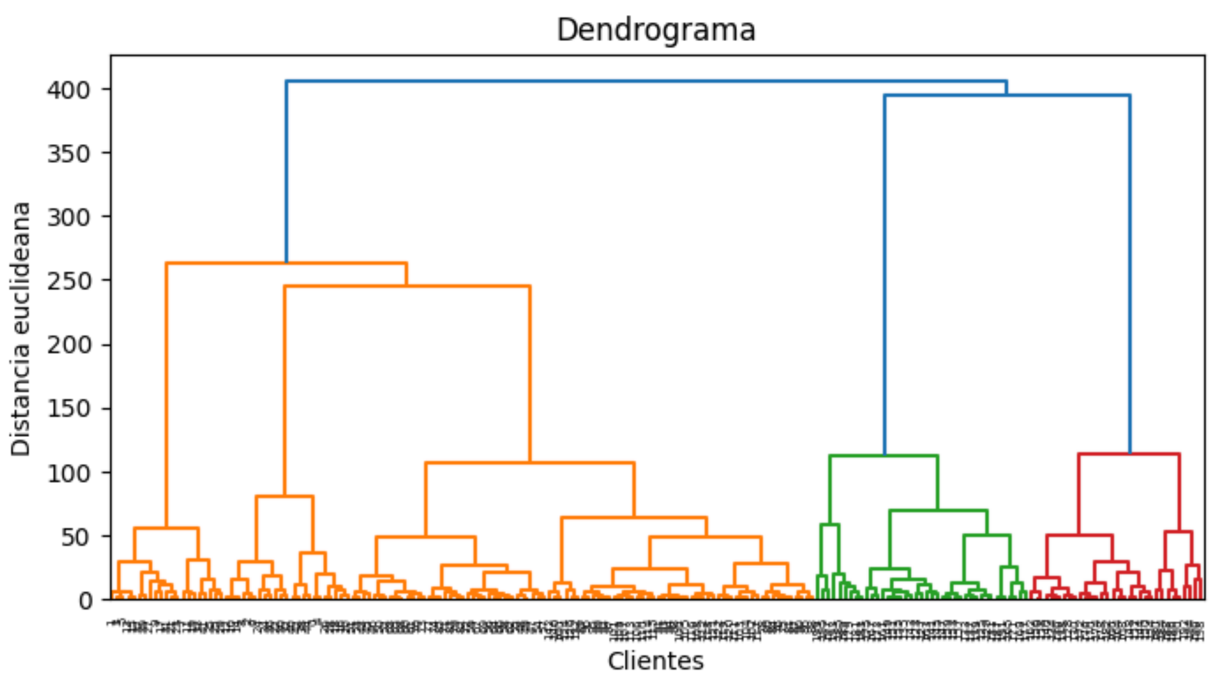
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
# Dataset
dataset = pd.read_csv('Mall_Customers.csv')
dataset.head(3)
```

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6

```
# Trabajamos solamente con Annual Income (k$)   Spending Score (1-100)
X = dataset.iloc[:, [3, 4]].values
print(X.shape)
# Dendrograma para determinar el número óptimo de grupos
import scipy.cluster.hierarchy as sch
plt.figure(figsize=(8,4))
dend = sch.dendrogram(sch.linkage(X, method='ward'))
plt.title('Dendrograma')
plt.xlabel('Clientes')
plt.ylabel('Distancia euclideana')
plt.show()
```

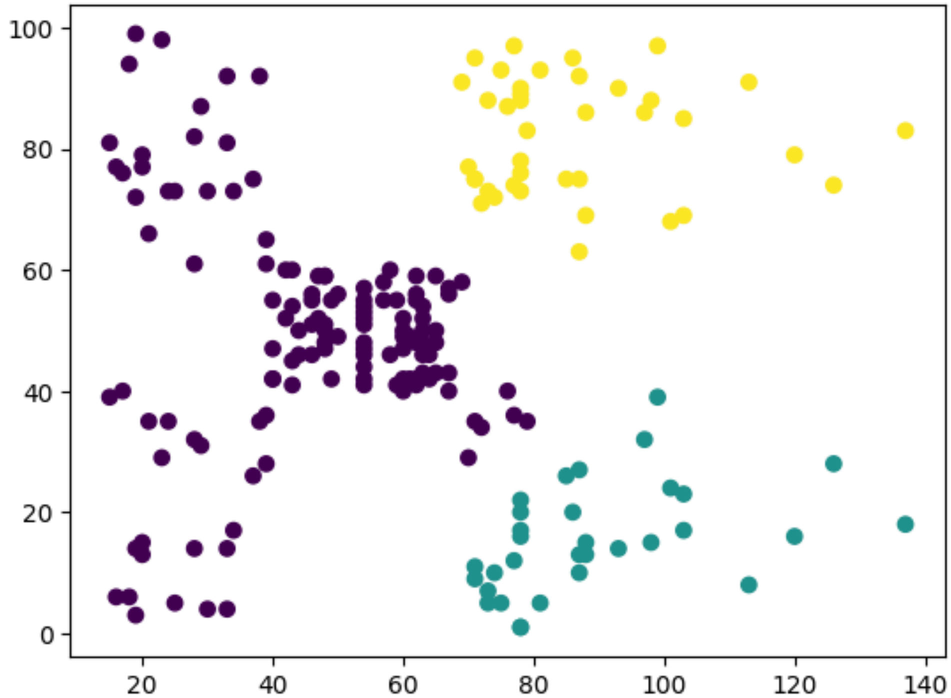
```
(200, 2)
```




```
# Con 3 grupos, siguiendo la recomendación del dendrograma
from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters=3,metric='euclidean',linkage='ward')
y_hc = hc.fit_predict(X)
```

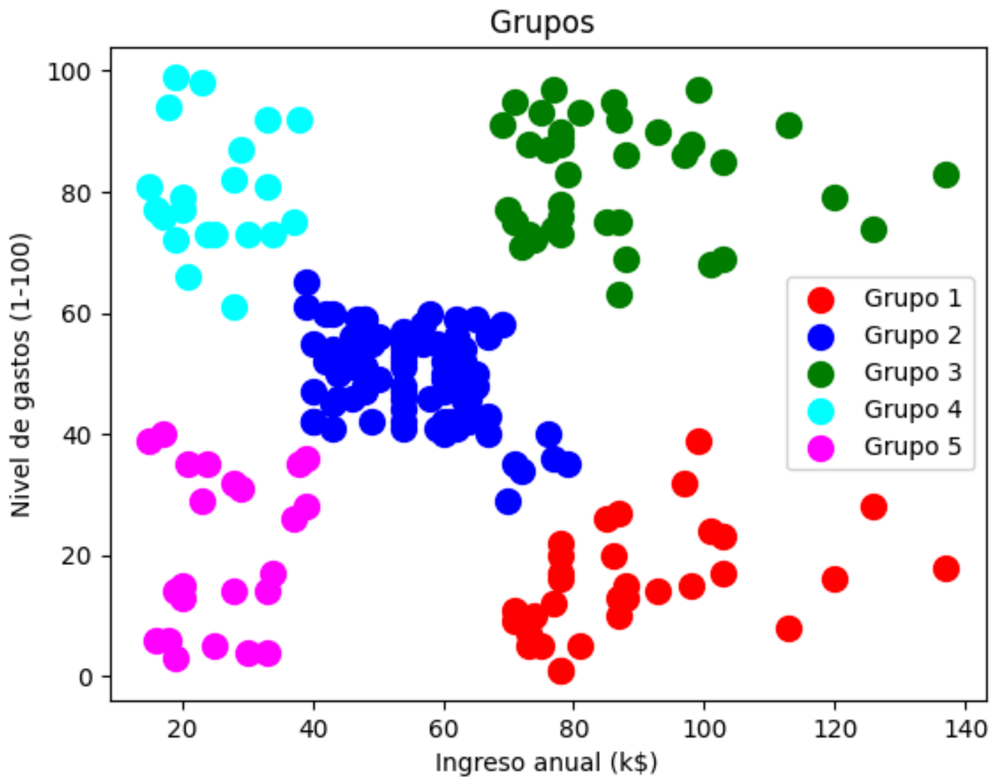
```python
plt.scatter(X[:,0], X[:,1], c=y_hc)
```

```
<matplotlib.collections.PathCollection at 0x7fd355ae2310>
```




```python
# Con 5 grupos, versión del analista humano
from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters=5,metric='euclidean',linkage='ward')
y_hc = hc.fit_predict(X)
```

```python
# Graficando los resultados
plt.scatter(X[y_hc==0, 0], X[y_hc==0, 1], s=100, c='red', label='Grupo 1')
plt.scatter(X[y_hc==1, 0], X[y_hc==1, 1], s=100, c='blue', label='Grupo 2')
plt.scatter(X[y_hc==2, 0], X[y_hc==2, 1], s=100, c='green', label='Grupo 3')
plt.scatter(X[y_hc==3, 0], X[y_hc==3, 1], s=100, c='cyan', label='Grupo 4')
plt.scatter(X[y_hc==4, 0], X[y_hc==4, 1], s=100, c='magenta', label='Grupo 5')
plt.title('Grupos ')
plt.xlabel('Ingreso anual (k$)')
plt.ylabel('Nivel de gastos (1-100)')
plt.legend()
plt.show()
```



1

1

1

1

1

1

1

1

1

1

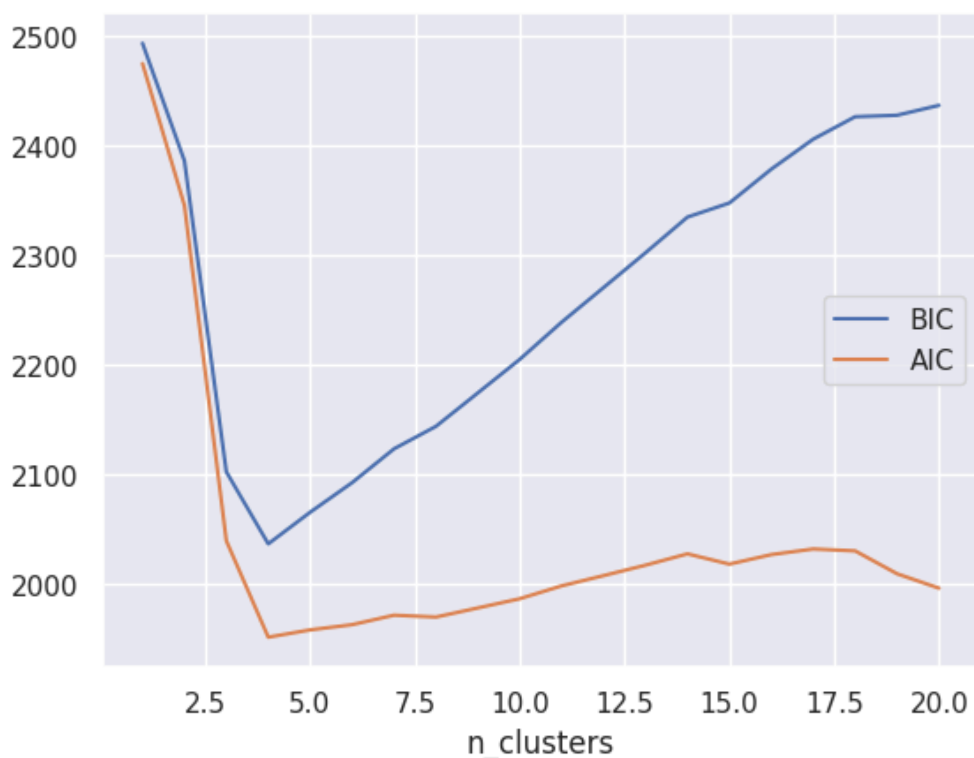
1

1

```python
# Ejemplo 1
# Bibliotecas
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
from matplotlib import pyplot as plt
from sklearn.mixture import GaussianMixture
from sklearn.datasets import make_blobs
```

```python
X, y = make_blobs(n_samples=300,centers=4,cluster_std=0.6,random_state=0)
#plt.scatter(X[:,0],X[:,1])
```

```python
# El número óptimo de grupos es el valor que minimiza el
# criterio de información de Akike (AIC) o el criterio Bayesiano (BIC)
# https://en.wikipedia.org/wiki/Akaike_information_criterion
# https://en.wikipedia.org/wiki/Bayesian_information_criterion
n_clusters = np.arange(1, 21)
models=[GaussianMixture(n,covariance_type='full',random_state=0).fit(X) for n in n_clusters
plt.plot(n_clusters, [m.bic(X) for m in models], label='BIC')
plt.plot(n_clusters, [m.aic(X) for m in models], label='AIC')
plt.legend(loc='best')
plt.xlabel('n_clusters')
plt.show()
```



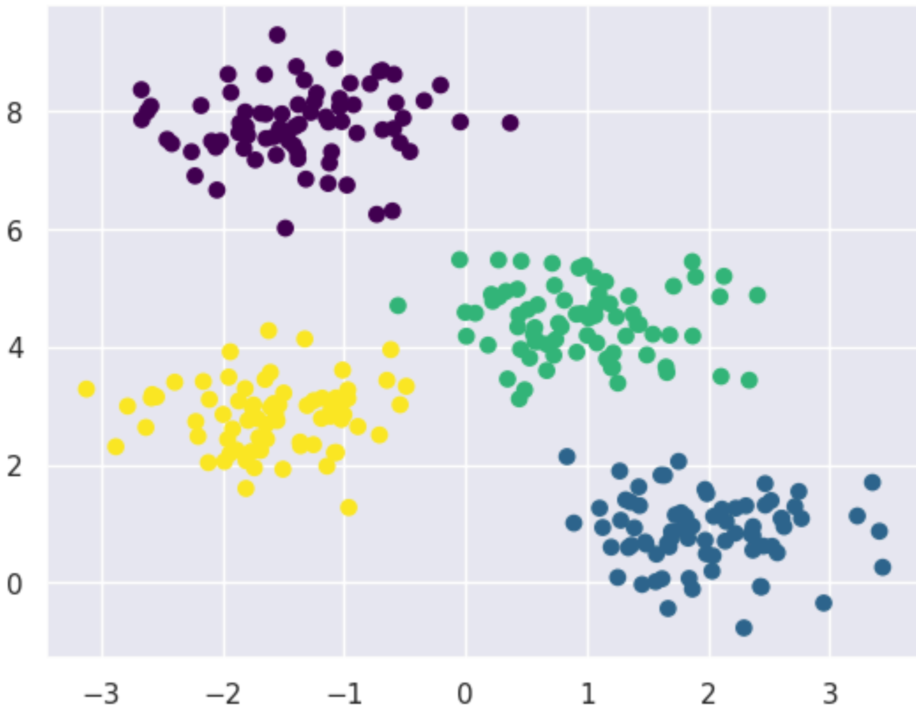

```python
gmm = GaussianMixture(n_components=4)
gmm.fit(X)
```

▾ GaussianMixture

```
GaussianMixture(n_components=4)
```

```python
y_pred = gmm.predict(X)
plt.scatter(X[:,0], X[:,1], c=y_pred, cmap='viridis')
```

```
<matplotlib.collections.PathCollection at 0x7fc1436e4d90>
```



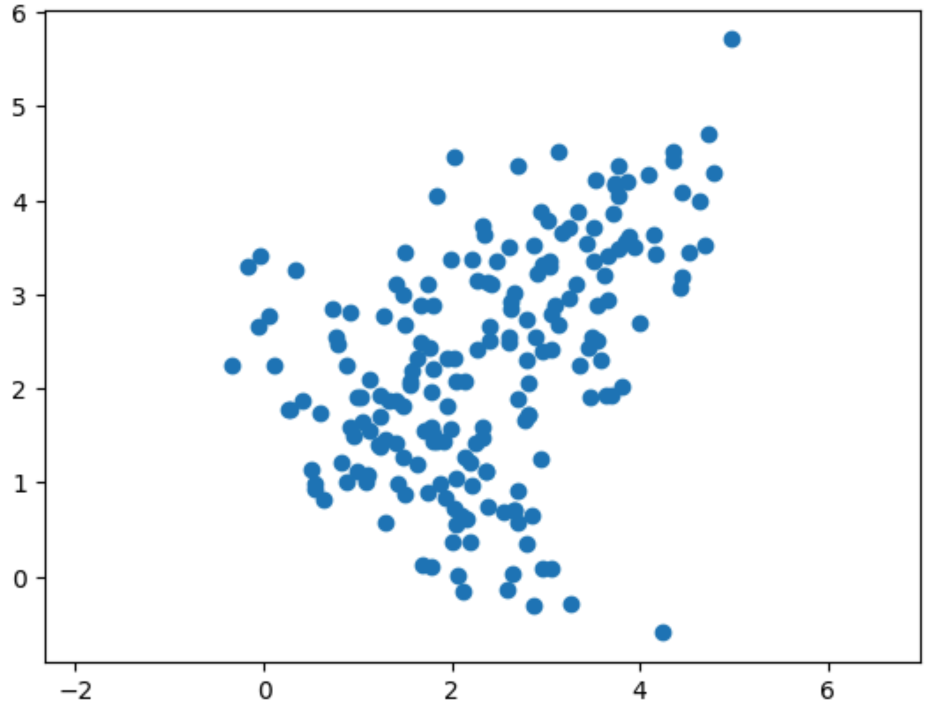

```python
# Ejemplo 2
# Bibliotecas
import numpy as np
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture
```

```python
# Datos
X_train = np.load('data.npy')
```

```python
X_train.shape
```

```
(200, 2)
```

```python
plt.scatter(X_train[:,0],X_train[:,1])
plt.axis('equal')
plt.show()
```

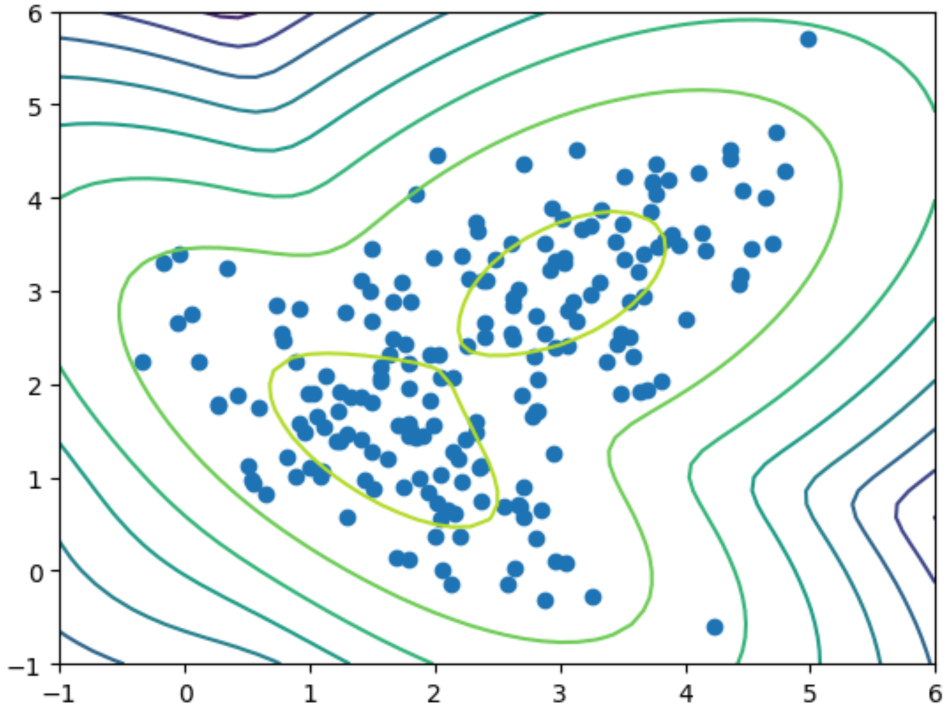



```
gmm = GaussianMixture(n_components=2)
gmm.fit(X_train)
print("Medias: \n", gmm.means_)
print("Covarianzas: \n",gmm.covariances_)
```

```
Medias:
 [[3.0363831  3.09828041]
 [1.60629419 1.3470999 ]]
Covarianzas:
 [[[ 0.8465178   0.38644336]
  [ 0.38644336  0.73395863]]

 [[ 0.75275611 -0.5054196 ]
  [-0.5054196   0.74286061]]]
```

```
X, Y = np.meshgrid(np.linspace(-1, 6), np.linspace(-1,6))
XX = np.array([X.ravel(), Y.ravel()]).T
Z = gmm.score_samples(XX)
Z = Z.reshape((50,50))
plt.contour(X, Y, Z)
plt.scatter(X_train[:,0], X_train[:,1])
plt.show()
```



1

1

1

1

1

1

1

1

1

1

1

1

1

DBSCAN

Density Based Spatial Clustering of Applications with Noise

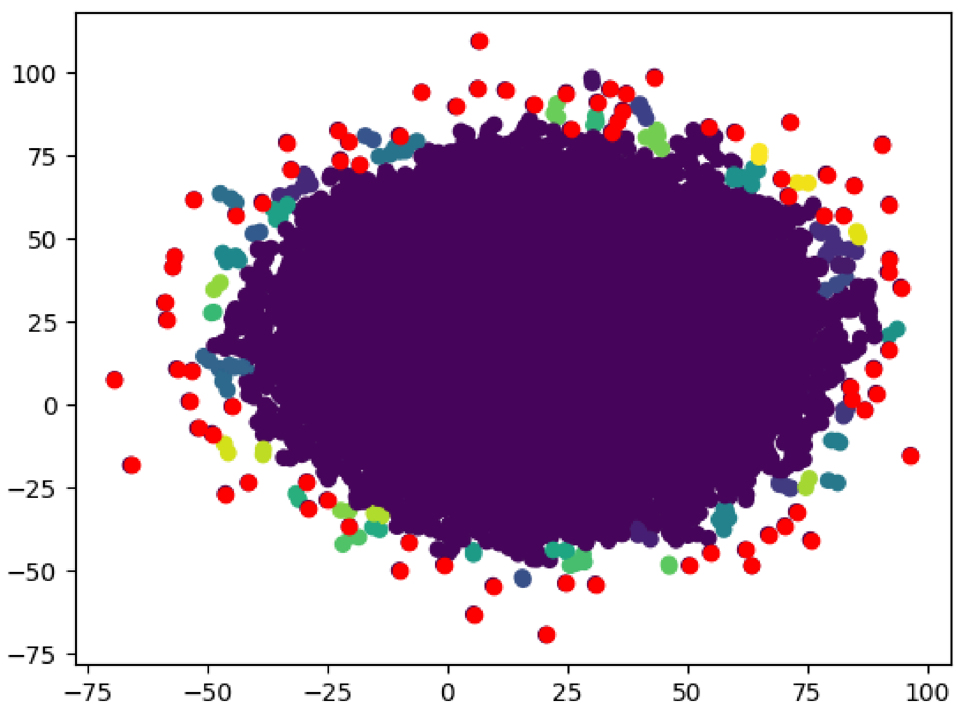
```
# Agrupamientos
import numpy as np
from sklearn.cluster import DBSCAN
np.random.seed(42)
data = np.random.randn(50000, 2) * 20 + 20 # 50,000 puntos
# Detección de anomalías
dbscan = DBSCAN(min_samples=2, eps=3)
clusters = dbscan.fit_predict(data)
list(clusters).count(0)
```

```
49769
```

Las anomalías se marcan con la etiqueta *-1*; para dimensiones altas reduce su eficacia. Estimar el valor de *eps* puede resultar complicado

```
# Visualización
import matplotlib.pyplot as plt
plt.scatter(data[:,0],data[:,1],c=dbscan.labels_)
plt.scatter(data[dbscan.labels_==-1, 0],data[dbscan.labels_==-1, 1],c='red') # anomalías
```

```
<matplotlib.collections.PathCollection at 0x7efdc1eb1650>
```




```
import pandas as pd
import matplotlib.pyplot as plt
```

```
import numpy as np
from sklearn.cluster import DBSCAN
from collections import Counter
```

```
df = pd.read_csv("winequality.csv") # en la carpeta ppcd/datasets !!!!
df.shape, df.head(2)
```

```
((6463, 13),
    type1  fixed acidity  volatile acidity  citric acid  residual sugar  \
 0  white            7.0              0.27         0.36            20.7
 1  white            6.3              0.30         0.34             1.6

    chlorides  free sulfur dioxide  total sulfur dioxide  density   pH  \
 0      0.045                 45.0                 170.0    1.001  3.0
 1      0.049                 14.0                 132.0    0.994  3.3

    sulphates  alcohol  quality
 0       0.45      8.8        6
 1       0.49      9.5        6  )
```

Usamos solamente *fixed acidity* y *volatile acidity*

```
data = df.iloc[:,1:3]
data.head(2)
```

	fixed acidity	volatile acidity
0	7.0	0.27
1	6.3	0.30

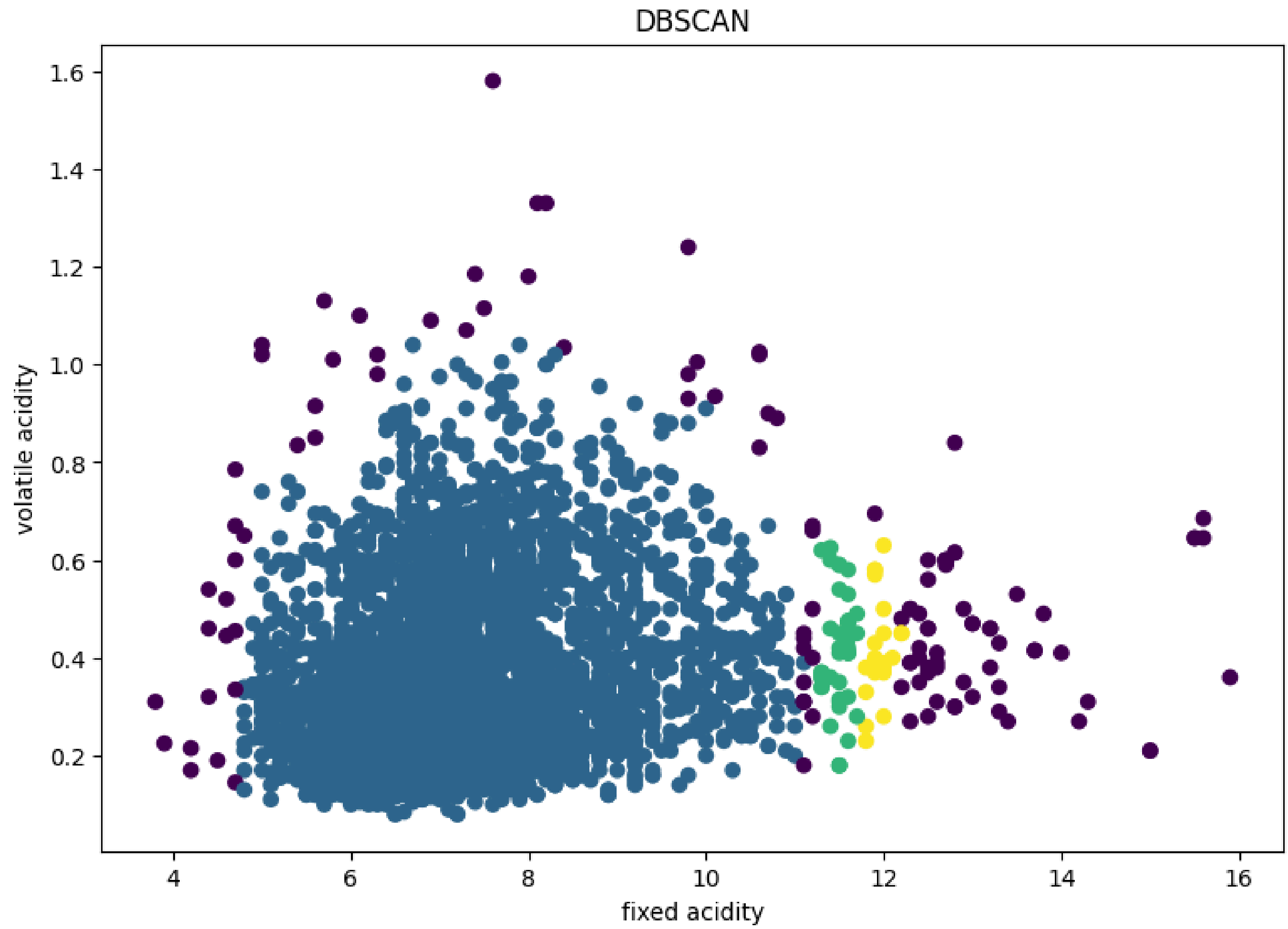
```
# Hiperparámetros
dbs = DBSCAN(eps=0.2, min_samples=20)
dbs.fit(data)
```

```
DBSCAN
DBSCAN(eps=0.2, min_samples=20)
```

```
# Mostrar el total de puntos de cada grupo
print(Counter(dbs.labels_))
```

```
Counter({np.int64(0): 6281, np.int64(-1): 117, np.int64(1): 40, np.int64(2): 25})
```

```
# Visualización
fig = plt.figure()
ax = fig.add_axes([0,0,1.1,1])
ax.scatter(data.iloc[:,0],data.iloc[:,1],c=dbs.labels_)
ax.set_xlabel('fixed acidity')
ax.set_ylabel('volatile acidity')
plt.title('DBSCAN')
plt.show()
```


DBSCAN
volatile acidity
fixed acidity
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
4
6
8
10
12
14
16