2B. Práctica: PCA

Nombre: **Luis Fernando Izquierdo Berdugo**

Materia: **Procesamiento de Información**

Fecha: **24 de Septiembre de 2024**

Instrucciones:

Seguir las instrucciones del notebook: `pca-separar-datos.ipynb`

PCA para preprocesar datos

El algoritmo PCA tiene la particularidad de que sus proyecciones tienden a separar los datos cuando se aplican a bajas dimensiones. En este notebook se verá ese efecto.

In [21]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.cluster import KMeans
from sklearn import decomposition
from sklearn import datasets
```

Leer datos

Se lee el conjunto de datos, `load_digits` son imágenes de números escritos a mano. El parámetro `n_class` indica el número de clases que queremos, en este caso sólo la clase de 0 y 1.

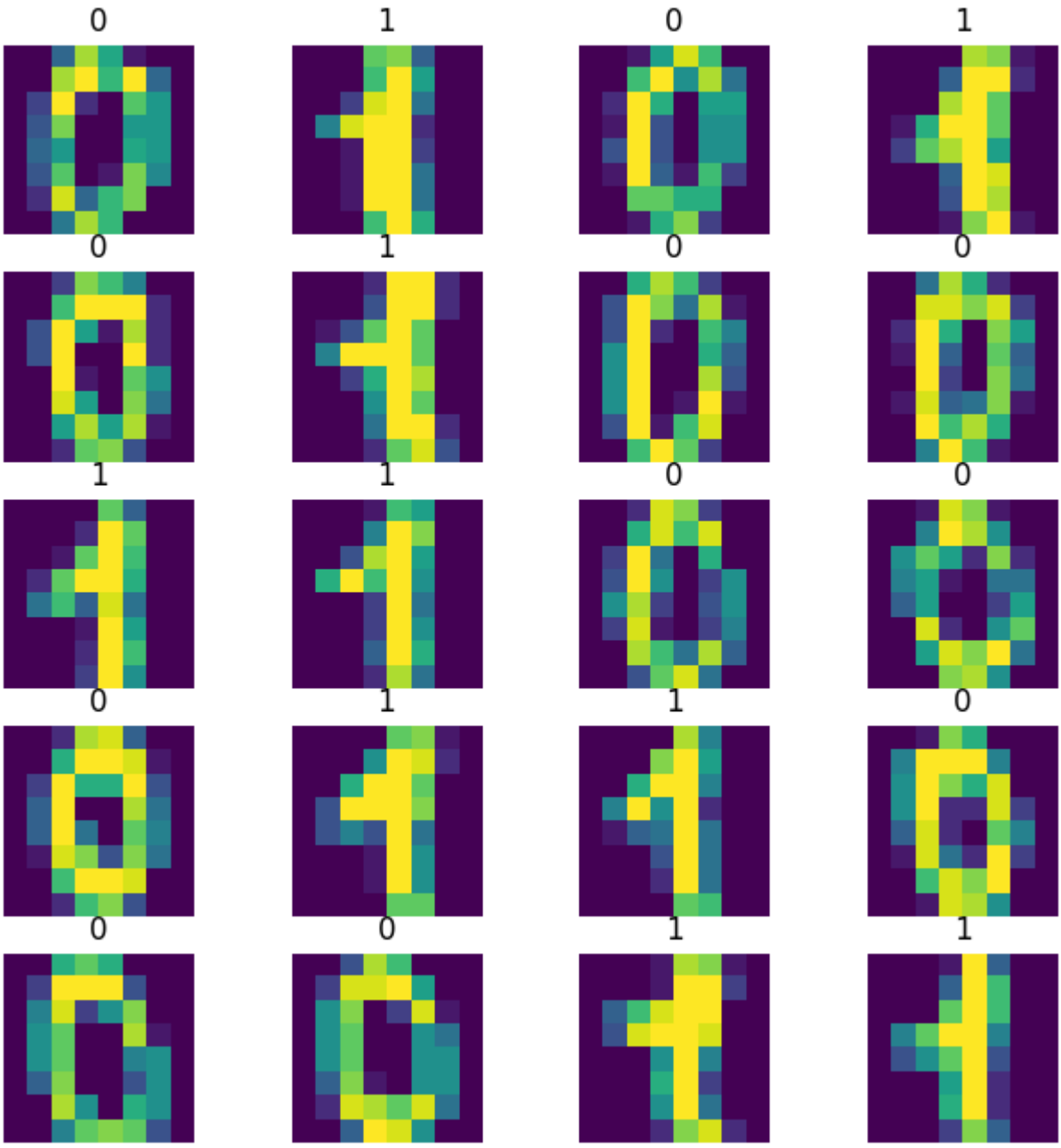
In [22]:
```
digits = datasets.load_digits(n_class=2)
X = digits.images.reshape((len(digits.images),-1))
y = digits.target
X.shape, y.shape
```

Out[22]:
```
((360, 64), (360,))
```

Mostramos las primeras 20 imágenes, todas son en escala de grises y tienen una resolución de 8x8. La variable y contiene el número que hay en la imagen correspondiente.

In [23]:
```
fig=plt.figure(figsize=(8, 8))
columns = 4
rows = 5
for i in range(20):
    img = digits.images[i]
    fig.add_subplot(rows, columns, i + 1)
    plt.imshow(img)
    plt.axis('off')
```

```
    plt.title(y[i])
plt.show()
```



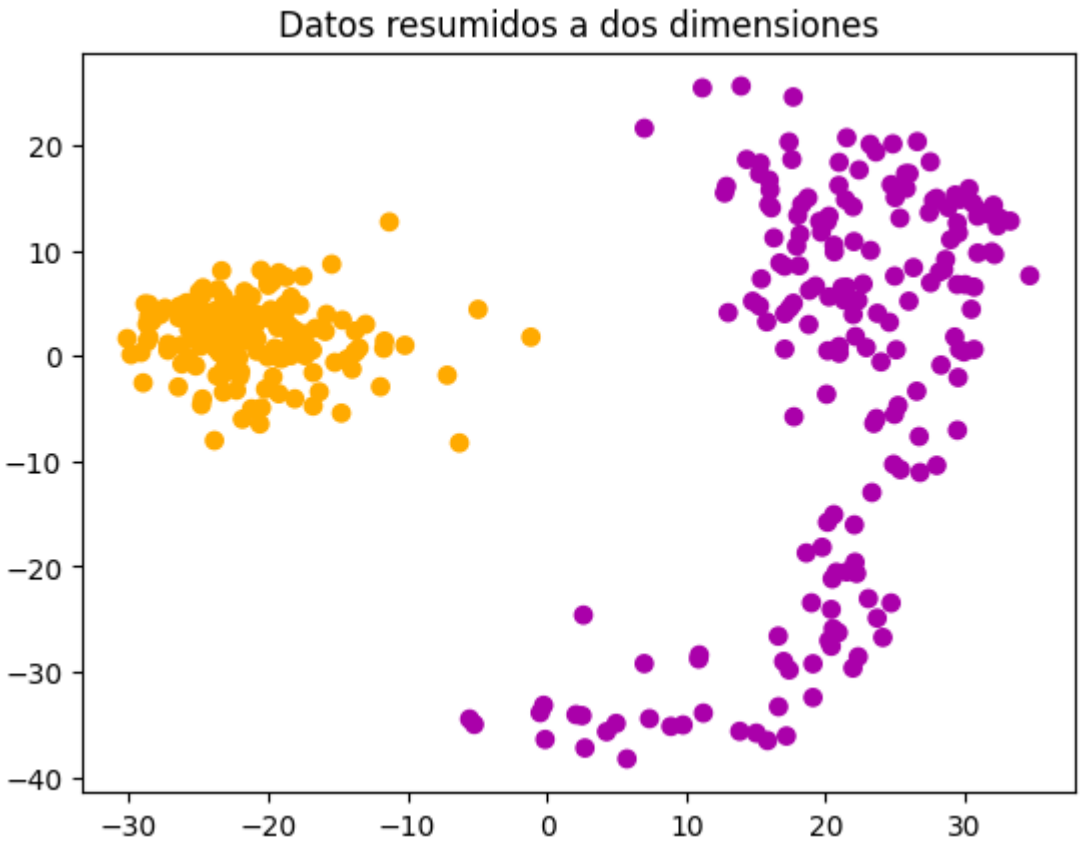
Aplicar PCA y graficar en dos dimensiones.

Aplicar PCA para reducir la dimensión a dos y graficar los puntos, el color de los puntos debe indicar la clase a la que pertenece.

In [24]:
```
import matplotlib.colors as mcolors

pca = decomposition.PCA(n_components=2)
colors = ['#FFAA00', '#AA00AA']
cm = mcolors.ListedColormap(colors)
labels = y
#Se usa la función fit_transform para evitar usar fit y después transform
x_trans = pca.fit_transform(X)
plt.scatter(x_trans[:, 0], x_trans[:, 1], c=labels, cmap = cm)
```

```
plt.title('Datos resumidos a dos dimensiones')
plt.show()
```



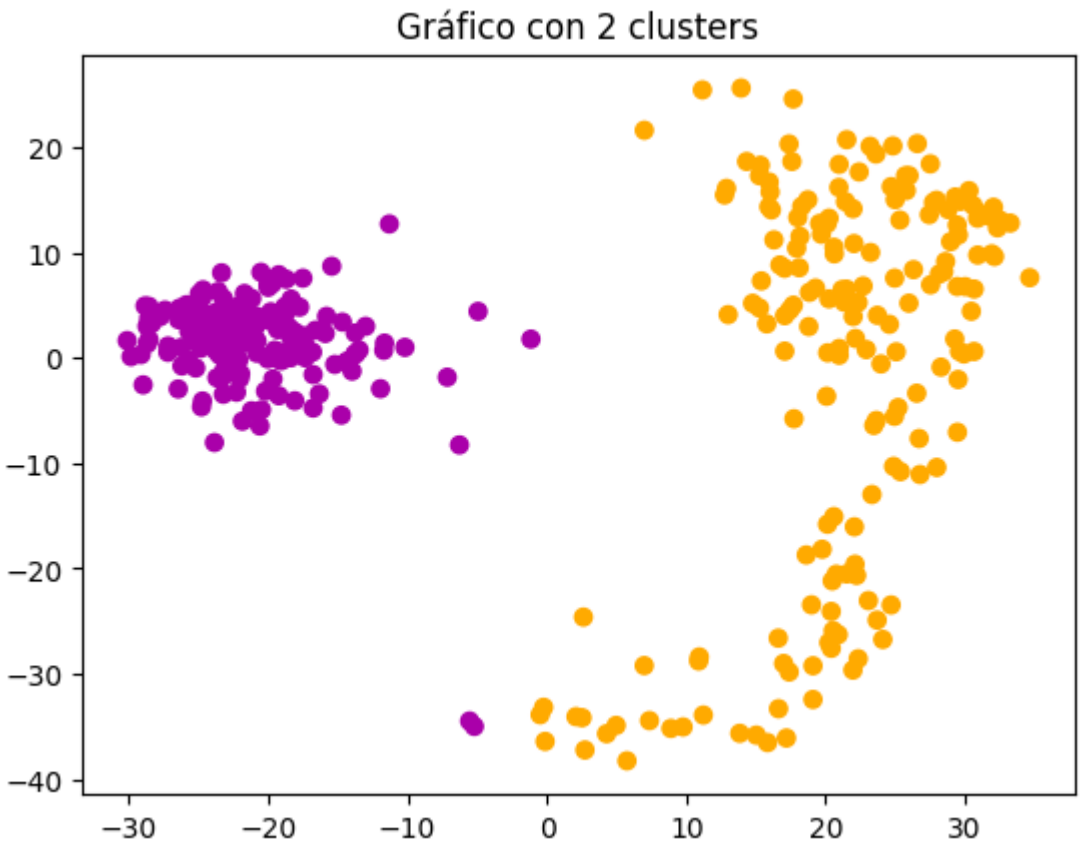
Ahora que los puntos se encuentran separados, aplicar kmeans para separar los dos clusters.

In [25]:
```
km = KMeans(2, n_init='auto', random_state=1004)
y_km = km.fit_predict(x_trans)
```

Graficar otra vez los puntos en dos dimensiones pero donde el color de los puntos indique la pertenencia a uno de los clusters. Comparar con la imagen anterior.

In [26]:
```
labels = y_km

plt.scatter(x_trans[:, 0], x_trans[:, 1], c=labels, cmap = cm)
plt.title('Gráfico con 2 clusters')
plt.show()
```



Hacer lo mismo pero dentro de un ciclo `for` donde el número de clases vaya de 2 a 10. Comparar las imágenes conforme se van agregando más clases.

In [29]:
```
colors2 = ['#FFAA00','#AA00AA','#000000','#0000AA','#00AA00','#00AAAA','#AA0000
cm2 = mcolors.ListedColormap(colors2)

for num_clusters in range(2, 10):
    digits = datasets.load_digits(n_class=num_clusters)
    X = digits.images.reshape((len(digits.images),-1))
    y = digits.target
    #Se usa la función fit_transform para evitar usar fit y después transform
    x_trans = pca.fit_transform(X)
    km = KMeans(num_clusters, n_init='auto', random_state=1004)
    y_km = km.fit_predict(x_trans)
    labels = y_km
    plt.scatter(x_trans[:, 0], x_trans[:, 1], c=labels, cmap = cm2)
    title = f"Gráfico con {num_clusters} clusters"
    plt.title(title)
    plt.show()
```

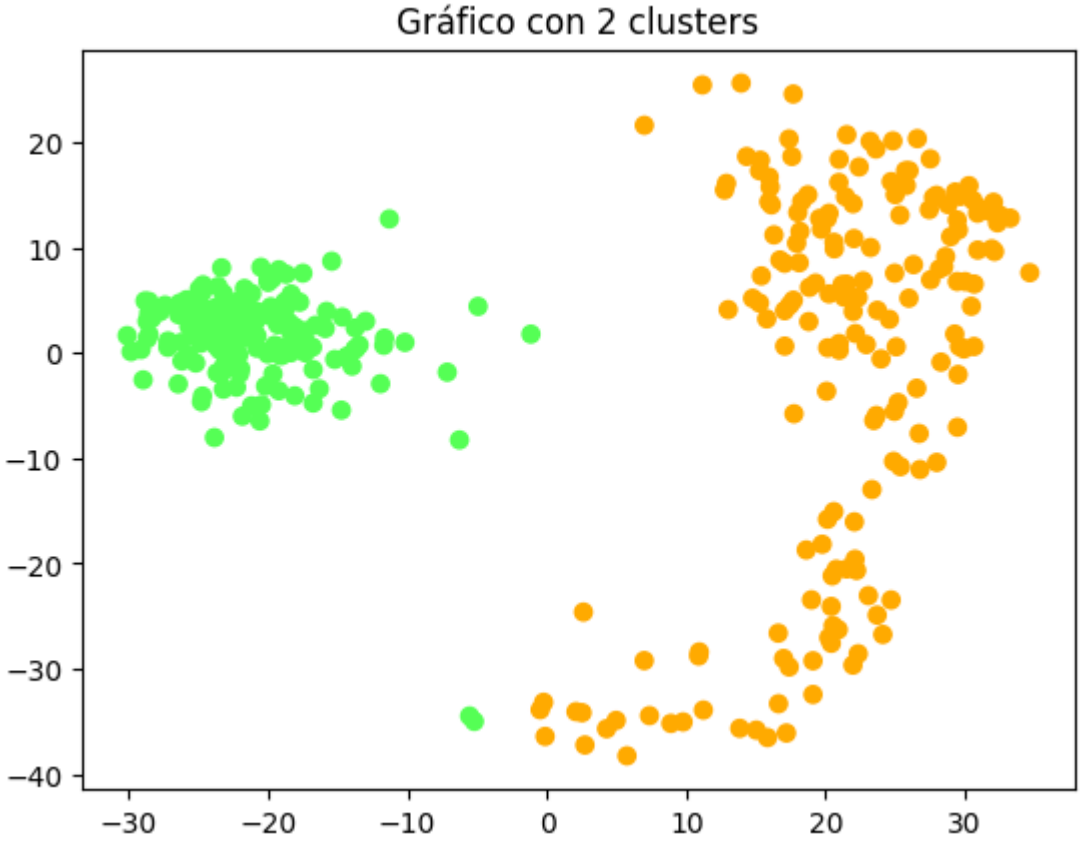


Gráfico con 2 clusters
20
10
0
−10
−20
−30
−40
−30
−20
−10
0
10
20
30

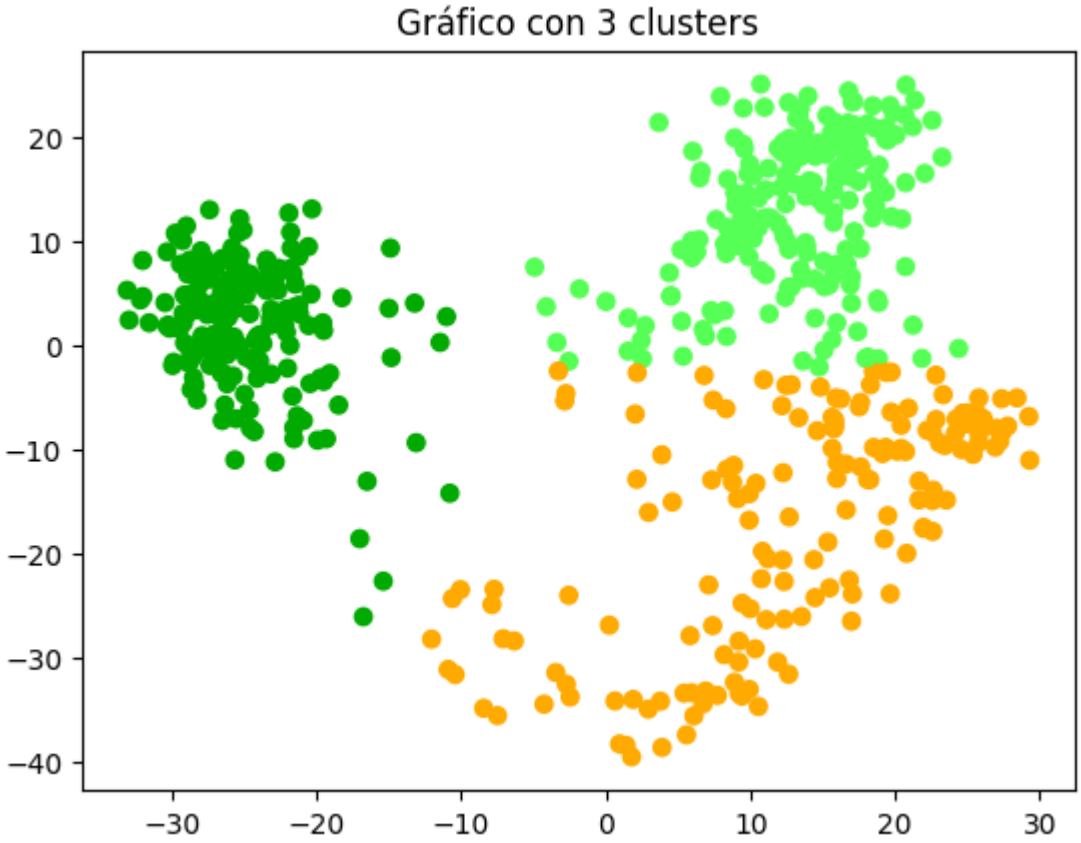


Gráfico con 3 clusters
20
10
0
−10
−20
−30
−40
−30
−20
−10
0
10
20
30

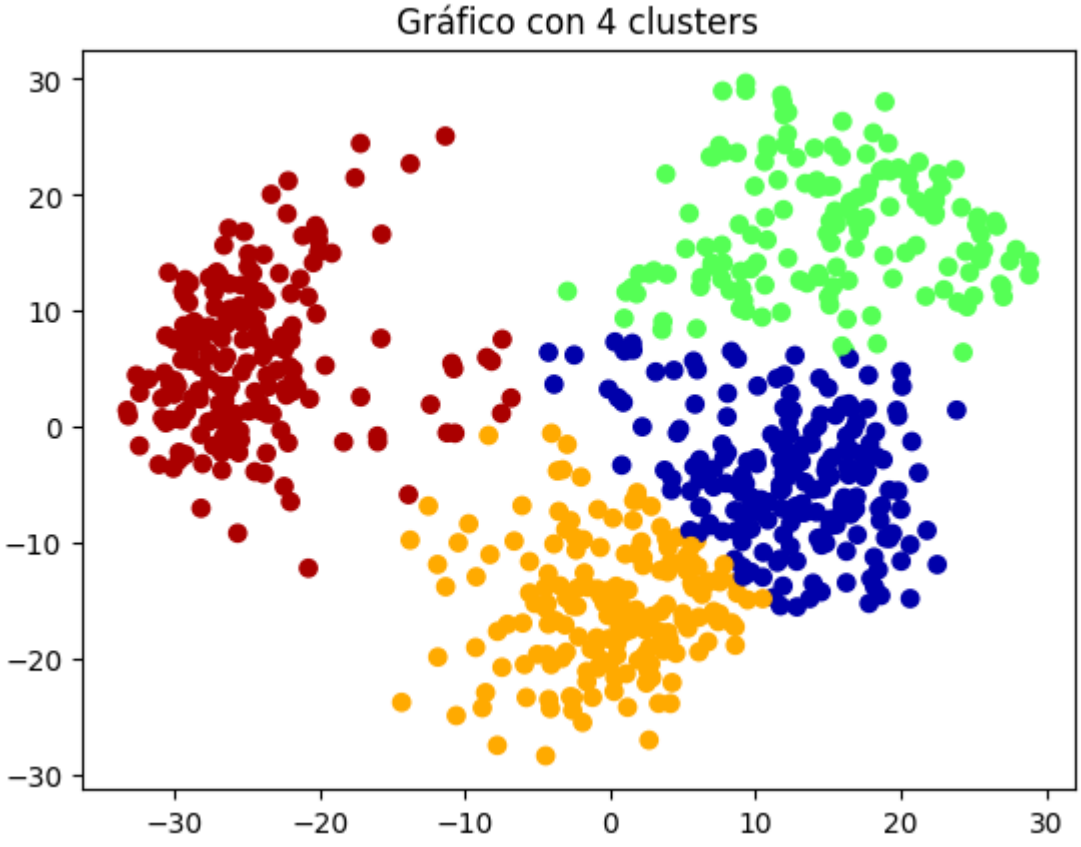


Gráfico con 4 clusters
30
20
10
0
−10
−20
−30
−30
−20
−10
0
10
20
30

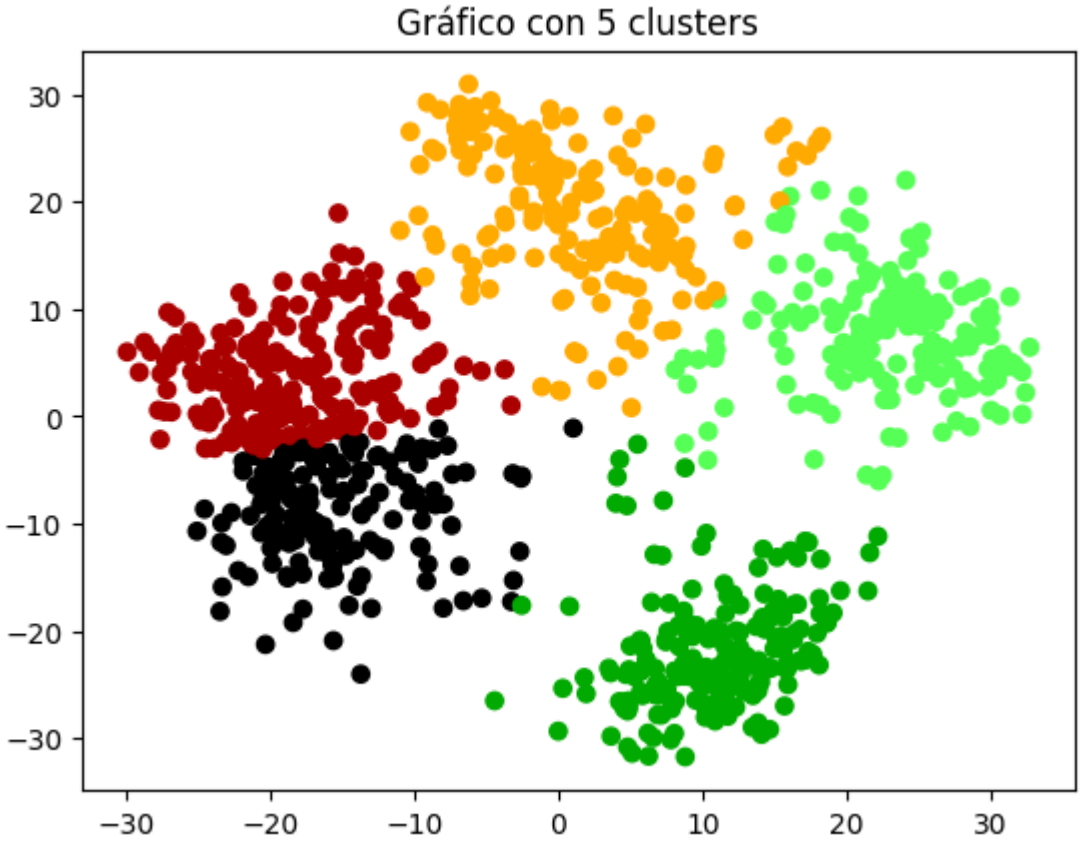


Gráfico con 5 clusters
30
20
10
0
−10
−20
−30
−30
−20
−10
0
10
20
30

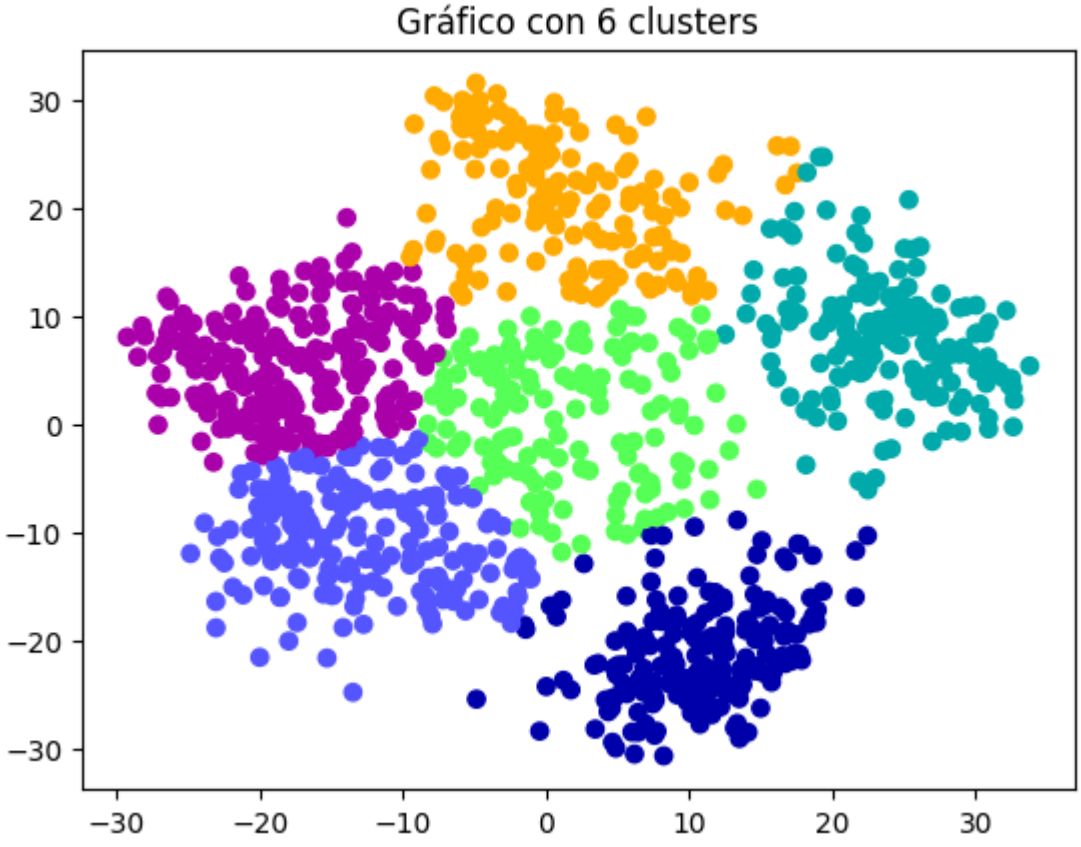


Gráfico con 6 clusters
30
20
10
0
−10
−20
−30
−30
−20
−10
0
10
20
30

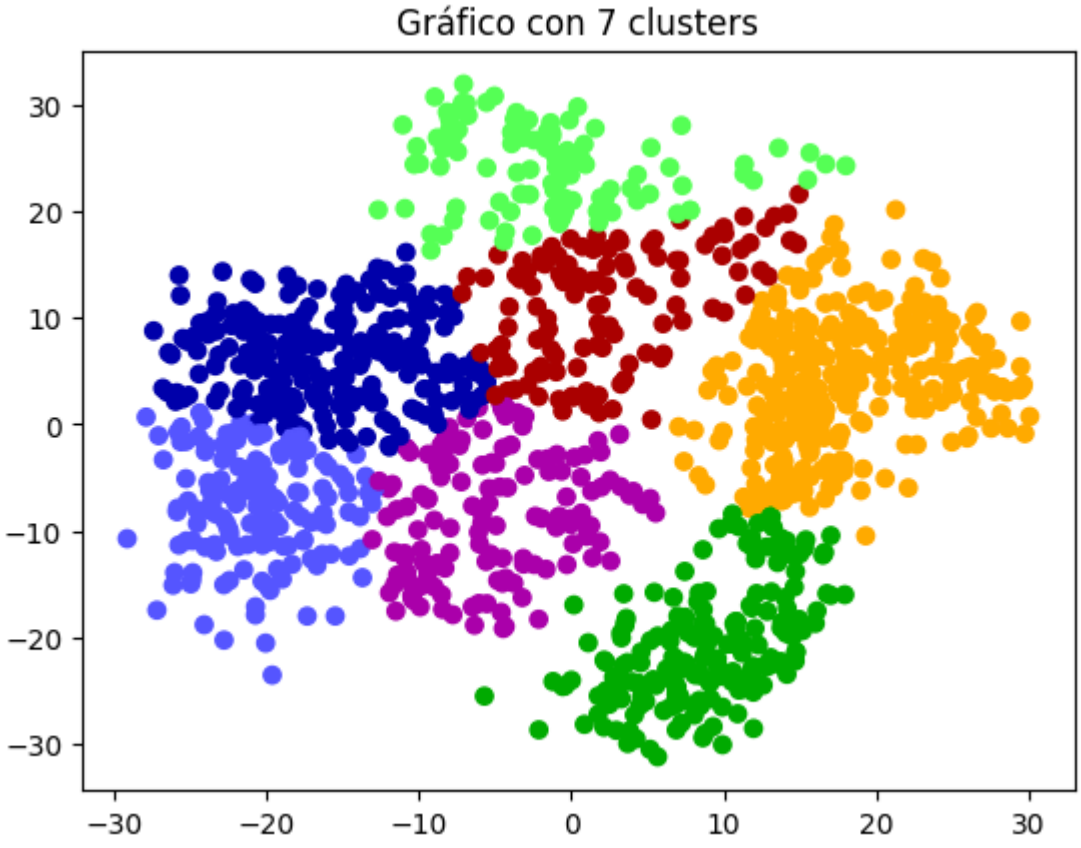


Gráfico con 7 clusters
30
20
10
0
−10
−20
−30
−30
−20
−10
0
10
20
30

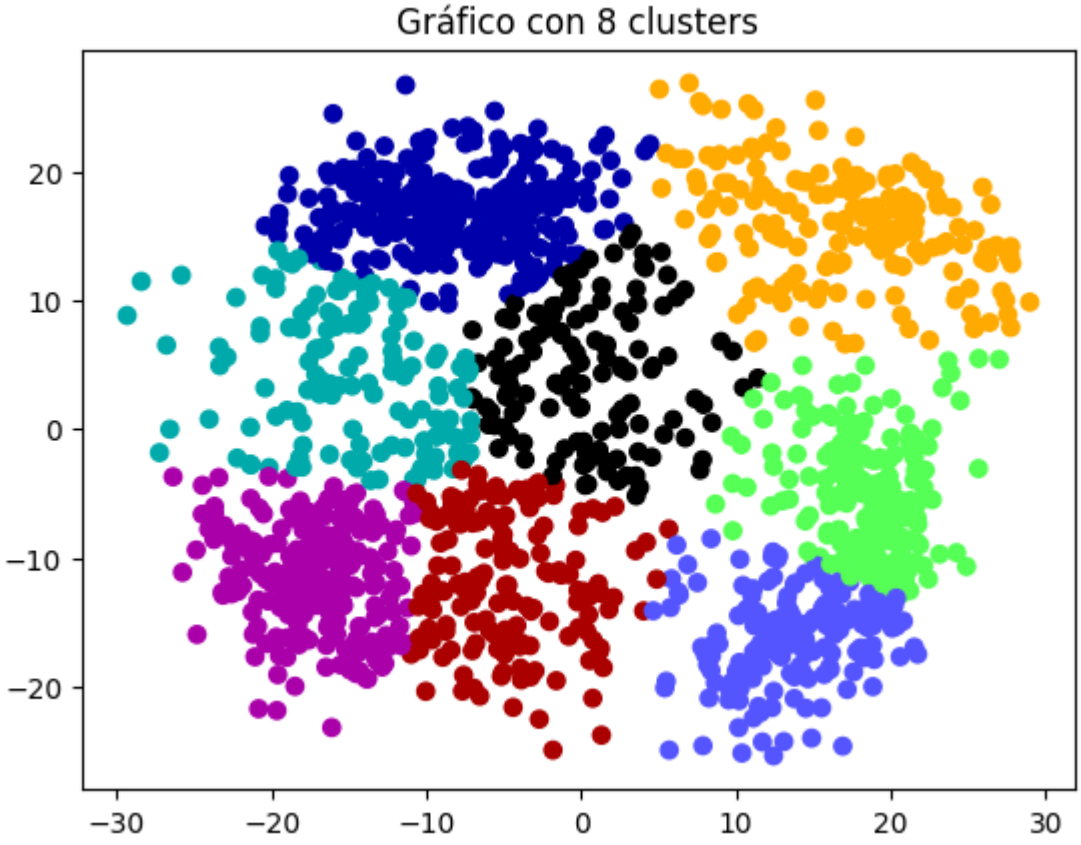


Gráfico con 8 clusters
20
10
0
−10
−20
−30
−20
−10
0
10
20
30

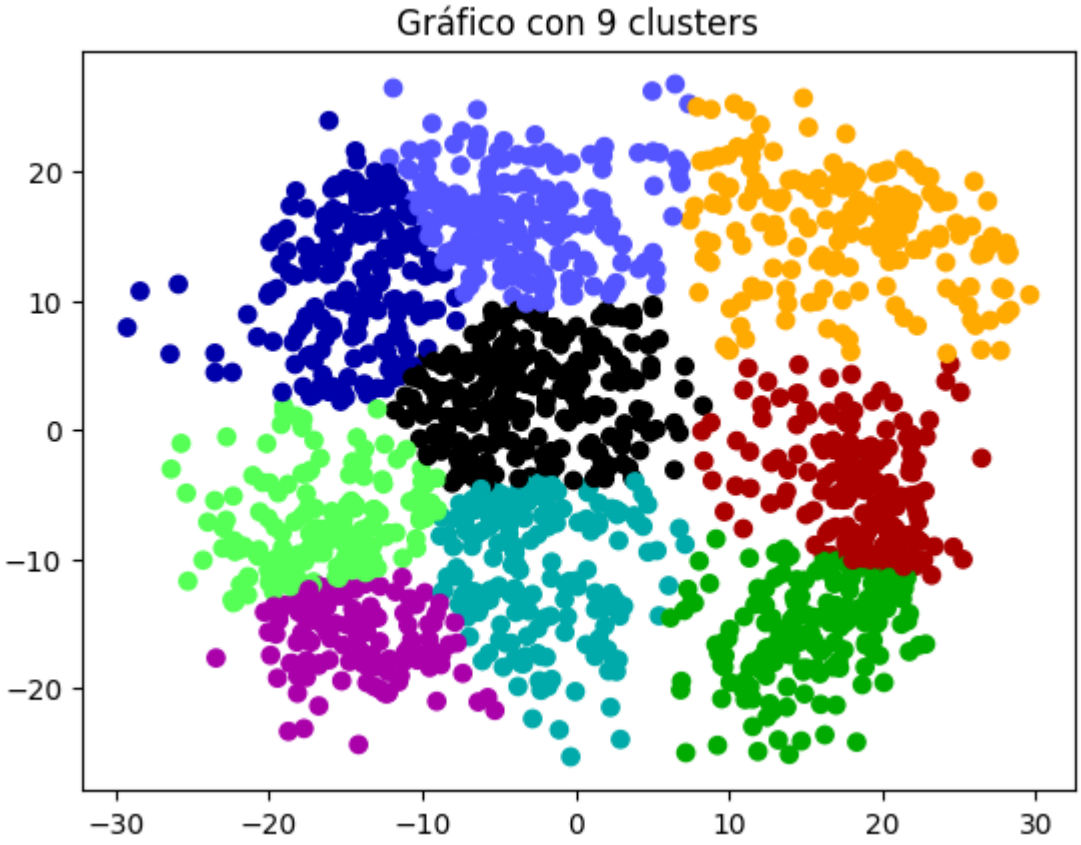


Gráfico con 9 clusters
20
10
0
−10
−20
−30
−20
−10
0
10
20
30