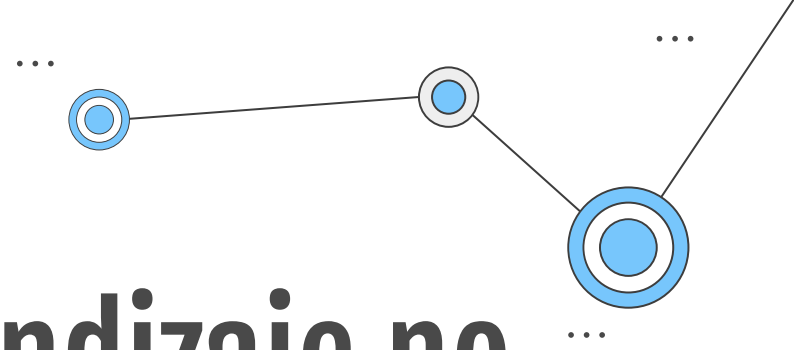


# Técnicas de aprendizaje no supervisado

UCC UNIVERSIDAD CATÓLICA DE CÓRDOBA JESUITAS

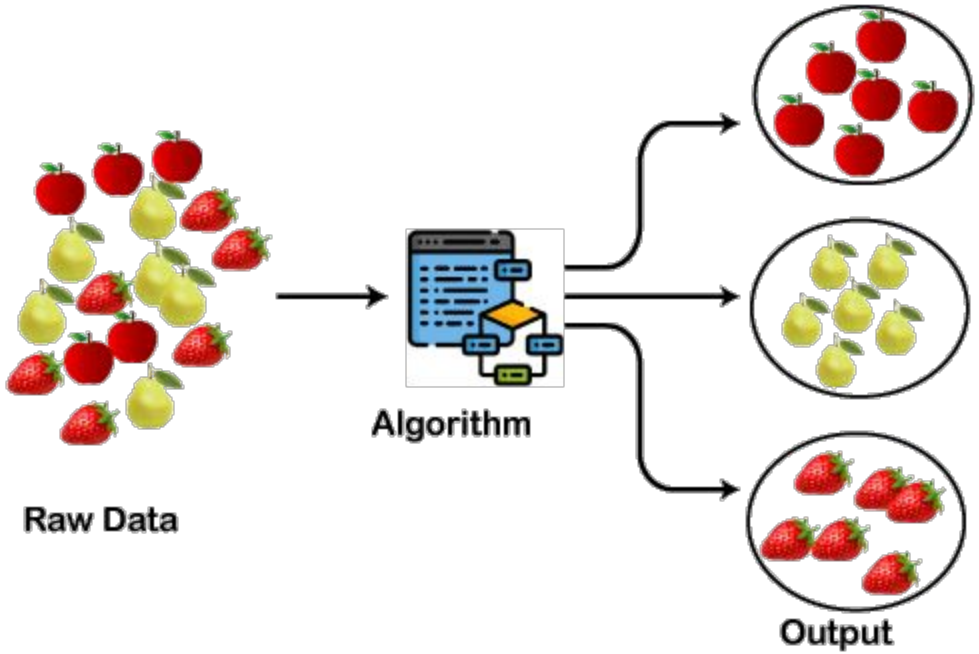
Dr. Francisco Arduh

2023

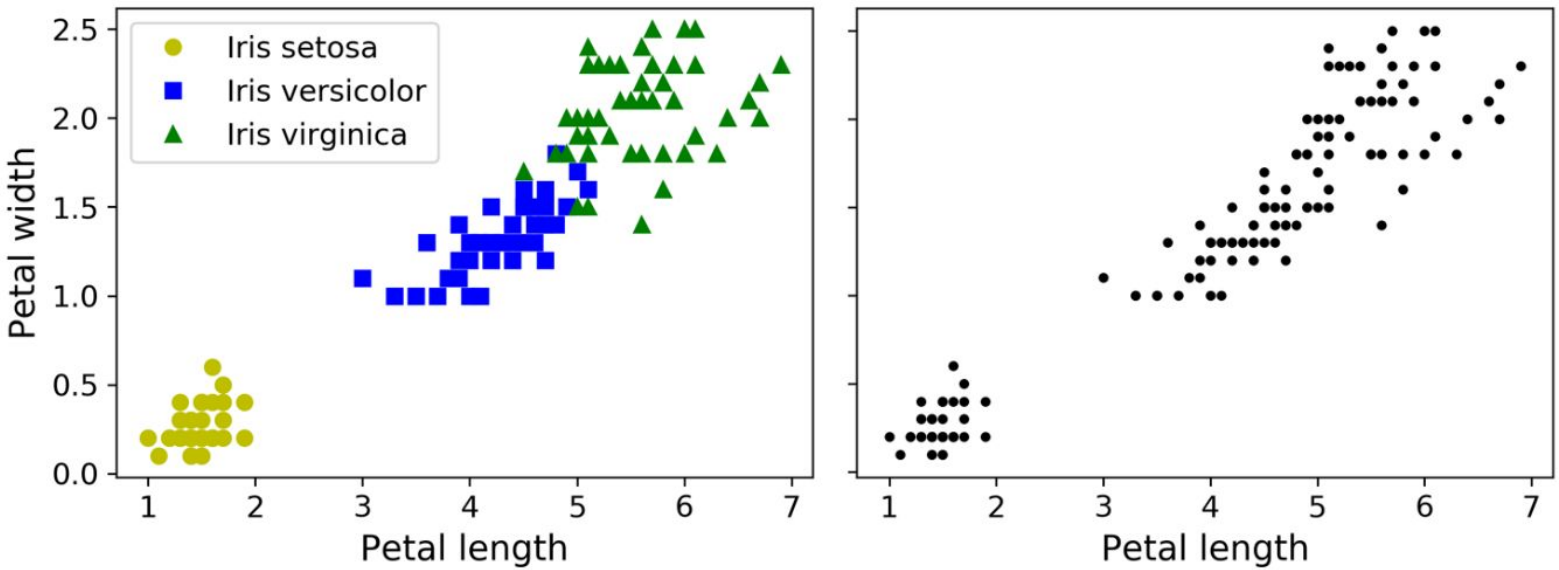
# Tareas del aprendizaje no supervisado

- Clustering: El objetivo es agrupar instancias similares en clusters.
- Detección de anomalías: el objetivo es aprender como se ven los datos “normalmente” .
- Estimación de densidad: estiman la función de densidad de probabilidad (PDF) de un proceso random que genera el dataset.
- Reglas de asociación: Se utilizan para descubrir hechos que ocurren en común dentro de un determinado conjunto de datos

# Clustering

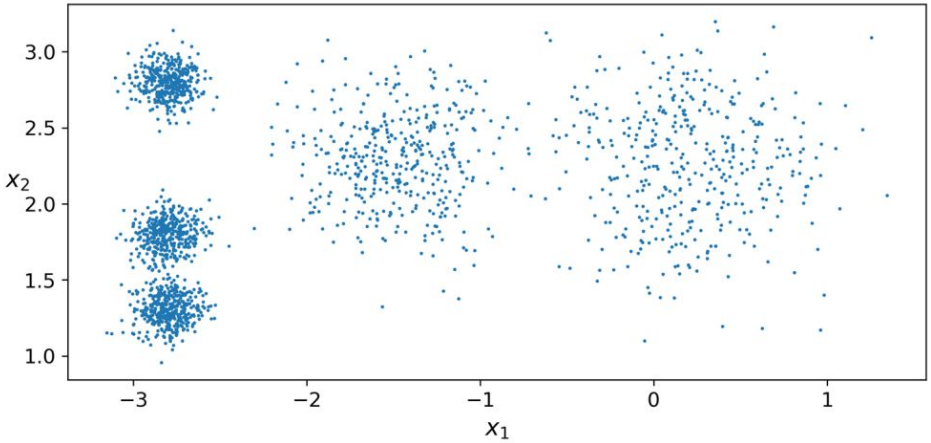


# Clustering: Iris dataset



# K-means

Supongamos que tenemos este dataset:




K-means va a buscar el centro de cada mancha.
Importamos y entrenamos el modelo:

```
from sklearn.cluster import KMeans
k = 5
kmeans = KMeans(n_clusters=k)
y_pred = kmeans.fit_predict(X)
```

# K-means

En este caso los labels (etiquetas) van a ser la predicción del algoritmo

```
>>> y_pred
array([4, 0, 1, ..., 2, 1, 0], dtype=int32)
>>> y_pred is kmeans.labels_
True
```

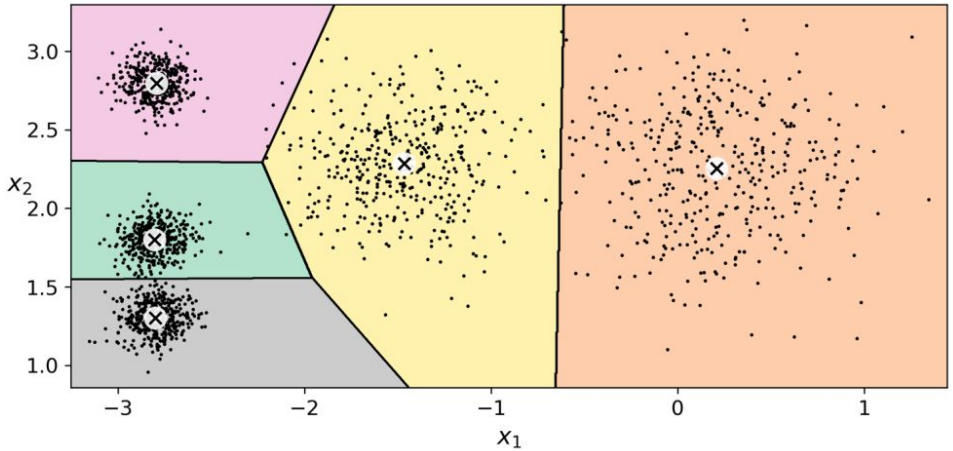
Se puede acceder a los centroides encontrado por el algoritmo:

```
>>> kmeans.cluster_centers_
array([[-2.80389616,  1.80117999],
       [ 0.20876306,  2.25551336],
       [-2.79290307,  2.79641063],
       [-1.46679593,  2.28585348],
       [-2.80037642,  1.30082566]])
```

En caso de tener nuevas instancias simplemente utilizo el método predict para predecir el cluster:

```
>>> X_new = np.array([[0, 2], [3, 2], [-3, 3], [-3, 2.5]])
>>> kmeans.predict(X_new)
array([1, 1, 2, 2], dtype=int32)
```

# K-means: límites de decisión

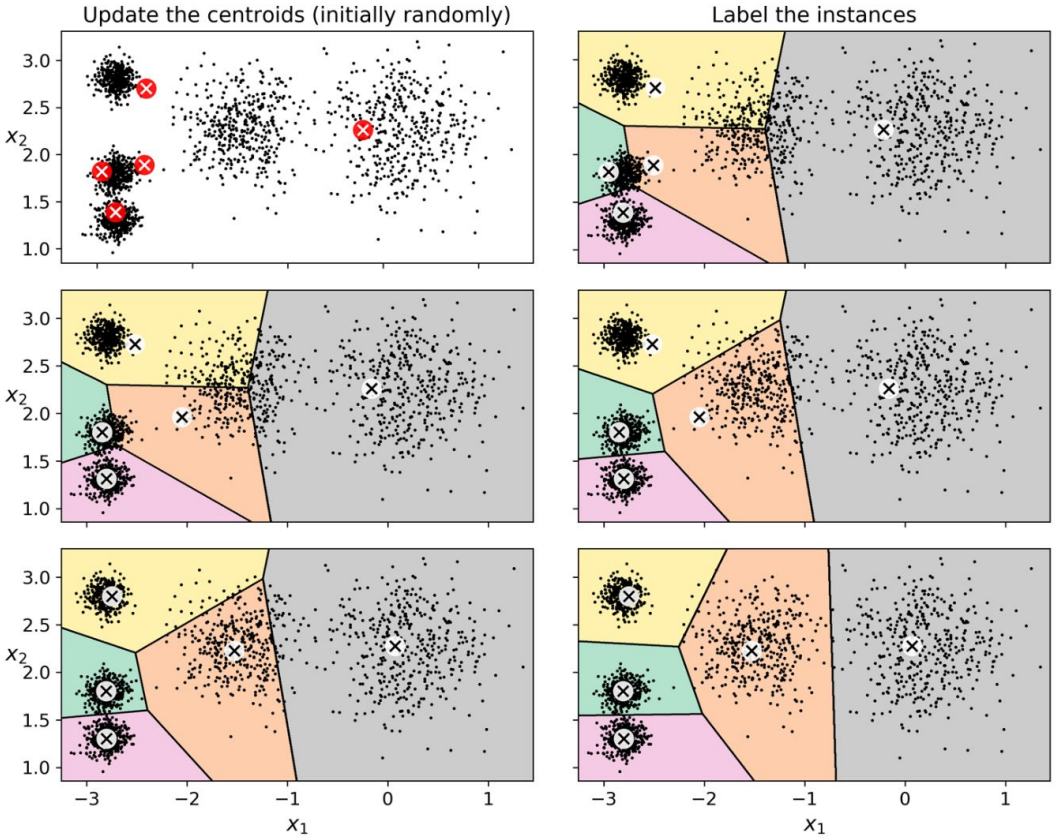



**Hard clustering:** asignar a cada instancia una sola clase.

**Soft clustering:** asignar un puntaje por cluster (Similar a las medidas de similaridad). Se utiliza el método `transform`.

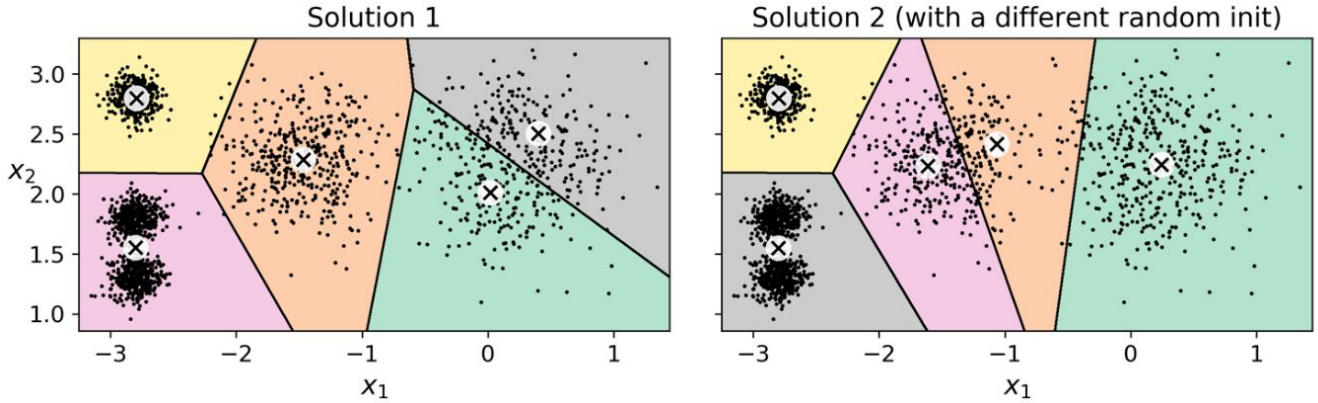
```
>>> kmeans.transform(X_new)
array([[2.81093633, 0.32995317, 2.9042344 , 1.49439034, 2.88633901],
       [5.80730058, 2.80290755, 5.84739223, 4.4759332 , 5.84236351],
       [1.21475352, 3.29399768, 0.29040966, 1.69136631, 1.71086031],
       [0.72581411, 3.21806371, 0.36159148, 1.54808703, 1.21567622]])
```

# K-means: ¿Cómo funciona?



# K-means: Algunos complicaciones

La convergencia a la solución óptima no está asegurada, va a depender de la inicialización de los centroides:



# K-means: Posible soluciones

1. Si se cuenta con información de antemano se puede inicializar los centroides:

```
good_init = np.array([[-3, 3], [-3, 2], [-3, 1], [-1, 2], [0, 2]])
kmeans = KMeans(n_clusters=5, init=good_init, n_init=1)
```

2. Otra posible solución es inicializar el algoritmo varias veces y quedarse con la mejor solución (usando `n_init`).
   Para comparar entre distintas soluciones se utiliza la métrica inercia, que define como la suma de las distancias  todas las instancias y su centroides.
   A menor inercia, mejor la solución:

```
>>> kmeans.inertia_
211.59853725816856
```

```
>>> kmeans.score(X)
-211.59853725816856
```

# K-means++

Propuesto en un paper David Arthur and Sergei Vassilvitskii, “k-Means++: The Advantages of Carefúl Seeding,” Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (2007): 1027–1035.
Se inicializa el algoritmo con centroides alejados entre sí.
Los pasos del algoritmo son:

1. Se elige un centroide $c^{(1)}$, elegido aleatoriamente en el dataset.
2. Se tomar un nuevo centroide $c^{(i)}$, eligiendo una instancia $x^{(i)}$ con una probabilidad definida como:

$$\frac{D(x^{(i)})^2}{\sum_{m}^{j=1} D(x^{(j)})^2}$$

   donde $D(x^{(i)})$ es la distancia entre la instancia $x^{(i)}$ y el centroide más cercano que ya ha sido elegido. Esta distribución de probabilidad garantiza que las instancias más alejadas de los centroides ya elegidos tengan una mayor probabilidad de ser seleccionadas como centroides.
3. Repetir el paso anterior hasta que se hayan elegido todos los k centroides.

La clase K-means utiliza esta inicialización por defecto, en caso de no querer utilizarla se define el hiperparámetros `init` con el valor `"random"`

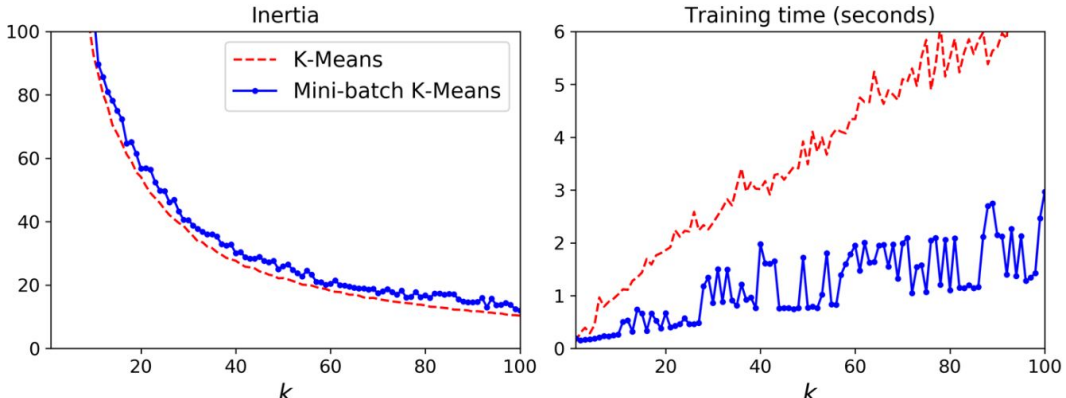
# K-means en minilotes

- Propuesto en el paper David Sculley, “Web-Scale K-Means Clustering,” Proceedings of the 19th International Conference on World Wide Web (2010): 1177–1178.
- En vez de utilizar todo el dataset en cada iteración se utiliza un mini lote.
- Se utiliza para dataset muy grandes

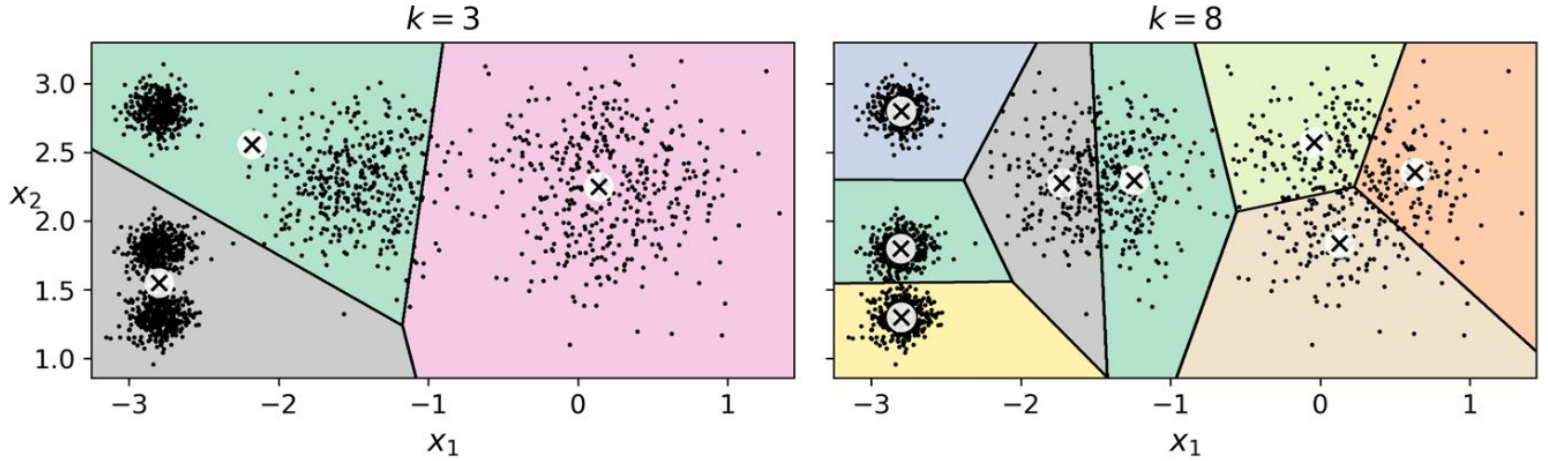
```
from sklearn.cluster import MiniBatchKMeans

minibatch_kmeans = MiniBatchKMeans(n_clusters=5)
minibatch_kmeans.fit(X)
```

- Si el dataset es muy grande se puede utilizar el método `partial_fit`.

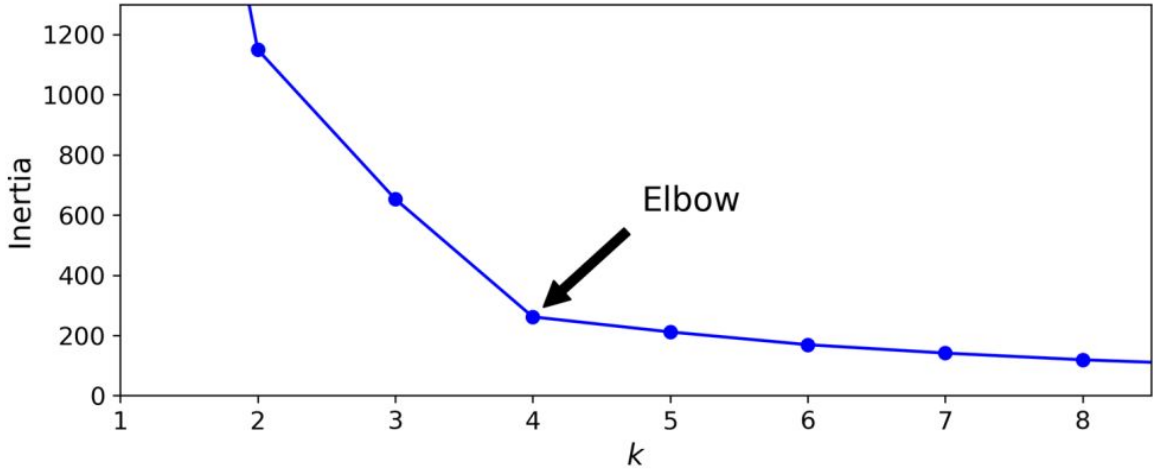


# ¿Cómo obtengo el k óptimo?



# ¿Cómo obtengo el k óptimo? Método gráfico

Graficando la inercia para distintos K obtenemos:



# ¿Cómo obtengo el k óptimo? coeficiente Silhouette

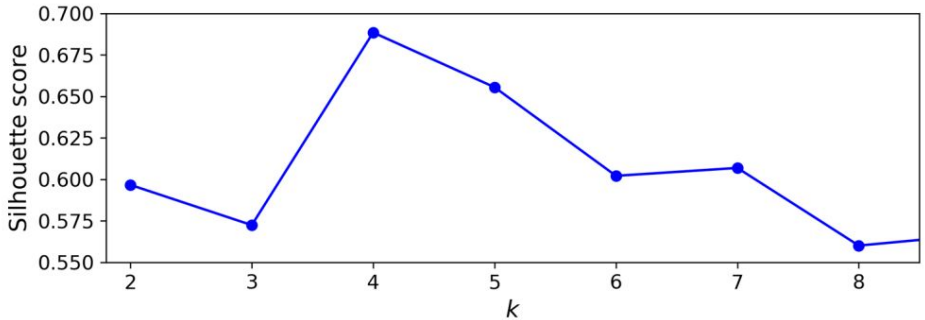
Se define como:

$$s = \frac{b - a}{max(a, b)}$$

Donde *a* es la distancia media a las otras instancias del mismo clúster y *b* es la distancia media al clúster más cercano.En Scikit-learn está implementado como:

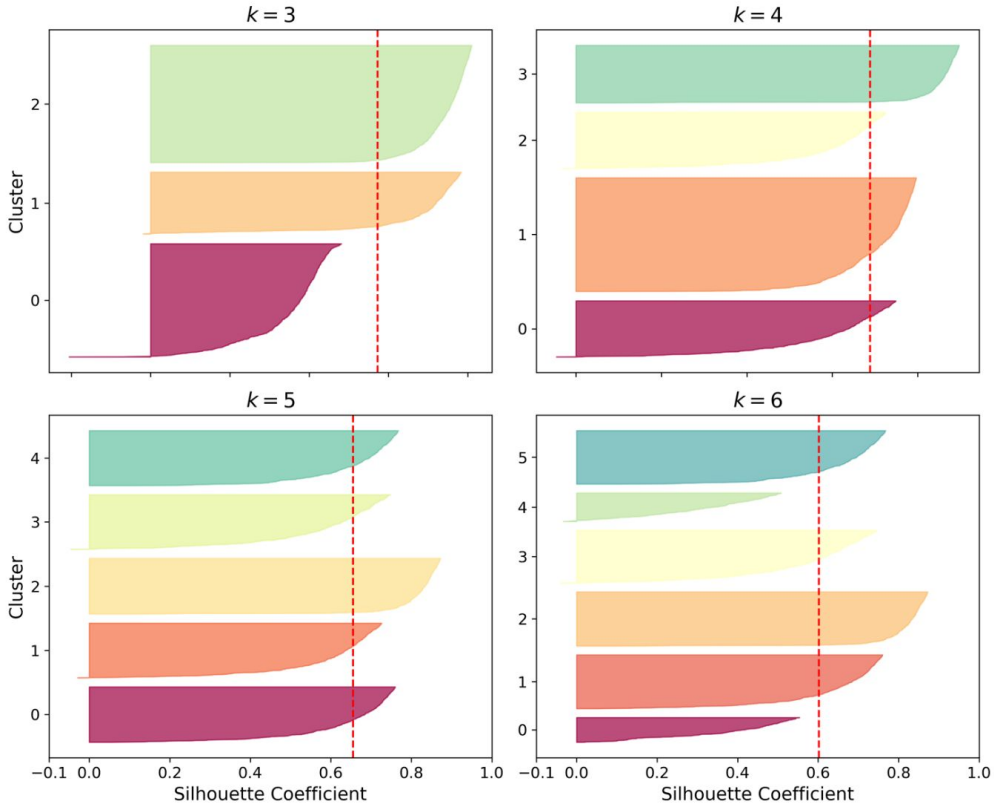
```
>>> from sklearn.metrics import silhouette_score
>>> silhouette_score(X, kmeans.labels_)
0.655517642572828
```

Graficamos para distintos K:



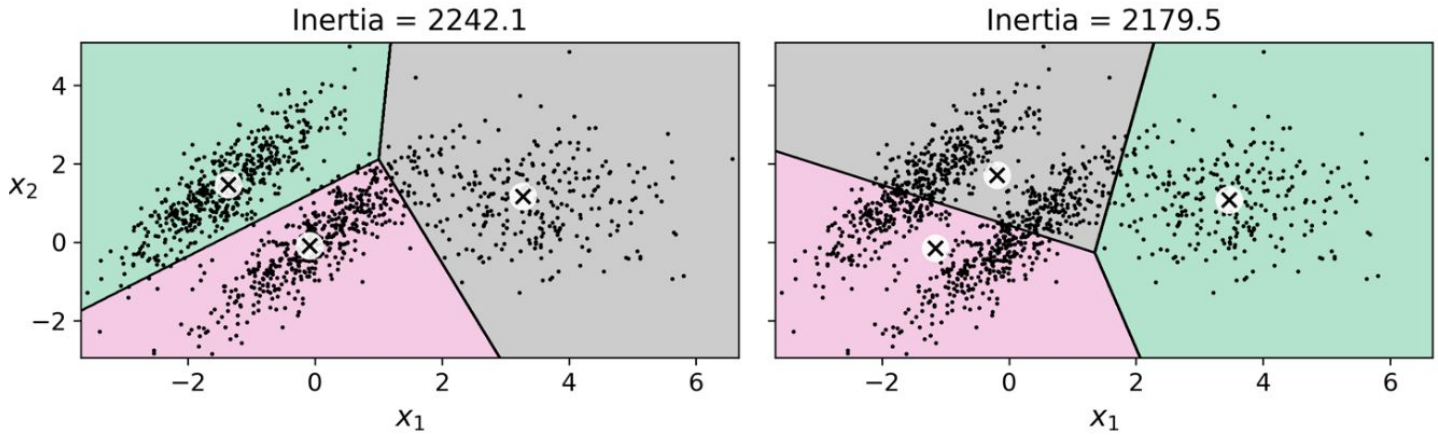
# Diagrama Silhouette

- La altura indica el número de instancias.
- El ancho, los coeficientes de las instancias del clúster.
- La línea de puntos, el coeficiente medio.

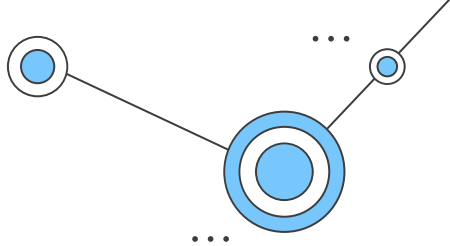


# Límites de K-means

- Se necesita ejecutar varias veces el algoritmo
- Se necesita especificar el número de clústers.
- No se comporta bien cuando los clústers tienen diferentes tamaños, densidades o son no esféricos.




Nota: es necesitario antes de ejecutar este algoritmo escalar las variables, aunque no asegura clúster esféricos.



# ¿Dudas?