INICIO. Breve descripción. MOTIVACIÓN:

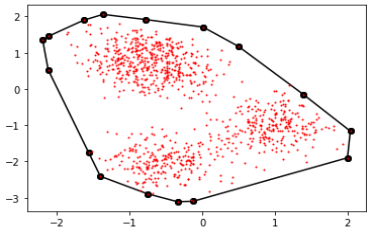
El algoritmo K-medias se utiliza para llevar a cabo una "clasificación por patrones" cuando tenemos un conjunto de puntos/datos X que pueden ser agrupados de alguna manera y no sabemos a priori como clasificarlos. Para ello, lo primero sería definir qué características deben tener estos "patrones" para particionar el conjunto X en clases/conjuntos.

Una de las principales desventajas de este algoritmo es que debemos fijar el número de regiones de antemano, operación que no siempre es fácil o intuitiva, *sobre todo cuando se trabaja con dimensiones mayores que 3*.

Este algoritmo tiene numerosas aplicaciones, por ello, al final de la práctica, se incluirá un ejercicio que tratará de predecir si un café tiene las propiedades (es decir, los "patrones") de calidad excelente, simple o mala. Se utilizará un *predict* para que, dada una serie de puntuaciones, se establezca a qué región (nivel de calidad) pertenece dicho café (el "Santo Grial" de todo estudiante).

Ejercicio i):

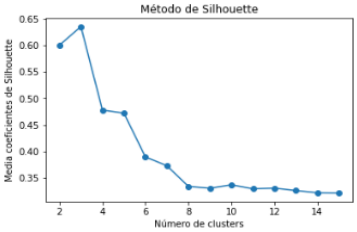
Gráficamente se aprecia que todos los puntos del conjunto X quedan contenidos en $EC(X)$ y que esta envoltura es convexa (podríamos expresarlo como intersección finita de semiplanos):



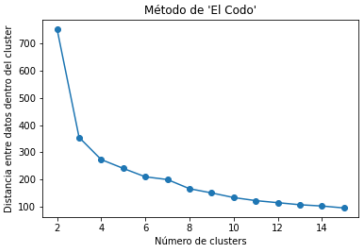
Ejercicio ii): El objetivo es minimizar la distancia intra-cluster (cohesión) y maximizar la distancia inter-cluster (separación), es decir, queremos agrupar puntos/datos que compartan ciertas propiedades con un alto grado de similitud a su vez que los diferenciamos claramente del resto de puntos de los demás clusters.

En este ejercicio se supone un número inicial de clusters igual a 4, pero descubrimos gráficamente (por dos métodos distintos) que ese número de clusters no es el que optimiza (o no optimiza de forma significativa) la partición de X:

Método 1: Este método mide cuán similar (bueno) es un dato (punto de X) con respecto a los demás puntos ubicados en el mismo cluster comparándolo con los datos que se encuentran en el cluster más cercano. Observando la evolución media del coeficiente de Silhouette, el pico máximo se alcanza para $\bar{s} = 0.6355$.

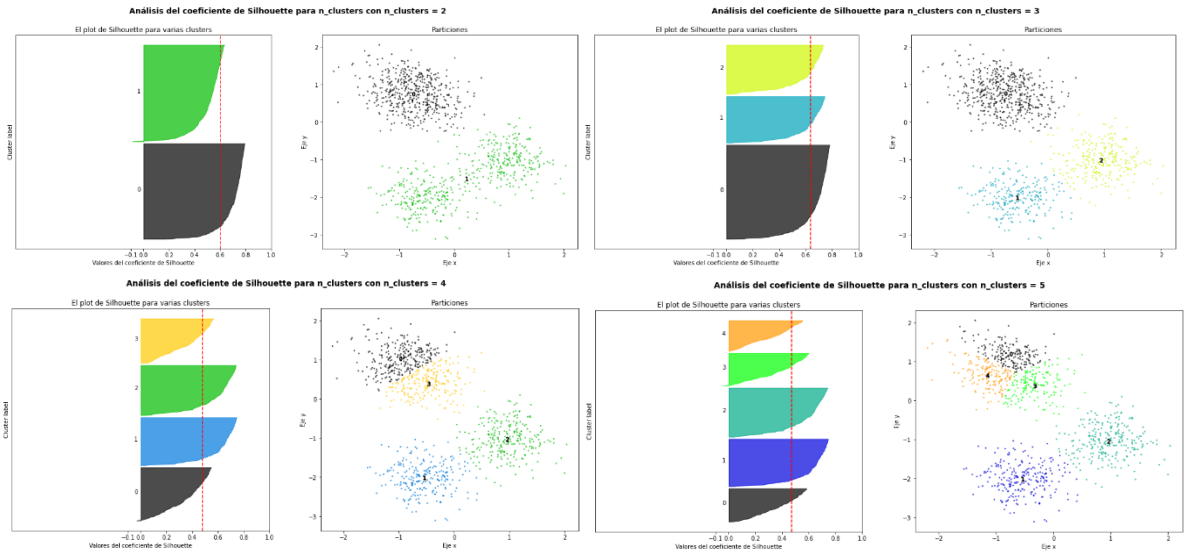


Método 2: Para confirmarlo, utilizaremos un segundo método, el cual calcula la varianza dentro de cada región (intra-cluster) y escogerá como óptimo aquel valor a partir del cual añadir más clusters no consigue una mejora significativa. En la imagen se confirma que lo recomendable son 3 clusters.



NOTA: Existen otros métodos como por ejemplo el ***índice de Duhn***, que identifica clusters con una gran acumulación de puntos y una buena separación. Así, se espera que el diámetro de los clusters sea pequeño y que la distancia entre los mismos sea grande, por tanto, el índice de Duhn tenderá a ser mayor. Por tanto, cuanto mayor sea el valor del índice de Duhn, mejor será la partición.

Para medir la calidad de la partición con el Método I, se ilustran a continuación una serie de diagramas de barras para distintos números de clusters:



Para **n_clusters = 2**: El valor medio del coeficiente de Silhouette $\bar{s} = 0.60035$, lo cual es un buen valor, pero se observa que hay un (o dos) dato(s) en la región 1 con un con un valor s pequeño y negativo, esto significa que ese dato pertenece más a la región 0 que a la región en la que se encuentra y que se ubica cerca de la frontera entre ambos clusters.

Para **n_clusters = 3**: El valor medio del coeficiente de Silhouette $\bar{s} = 0.6355$, lo cual es un valor mejor que el anterior (de hecho, será el óptimo) y no hay un solo dato mal clasificado, pues todos los valores s son positivos.

Para **n_clusters = 4**: El valor medio del coeficiente de Silhouette $\bar{s} = 0.47803$, lo que significa que el valor ha empeorado drásticamente. Además, en la región 0 hay un punto que parece pertenecer más a la región 3 que a la región 0. Situación análoga para **n_clusters = 5**.

Ejercicio iii): Con el apartado anterior, estamos preparados para trazar los respectivos diagramas de Voronoi. Lo haremos para **n_clusters = 3** y **n_clusters = 4**. De esta manera, todo punto contenido en dichas regiones compartirá las mismas propiedades, en este caso, son puntos que se encuentran a menor distancia del centroide de su respectivo cluster en comparación con la distancia de dicho punto a los centroides de los cluster vecinos:

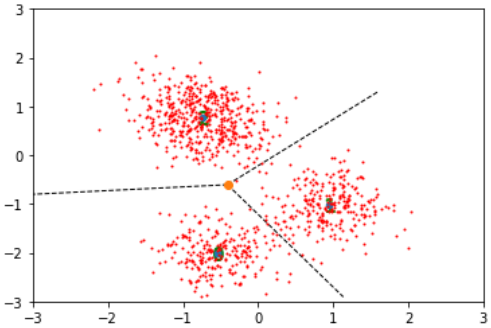


Diagrama de Voronoi para 3 clusters

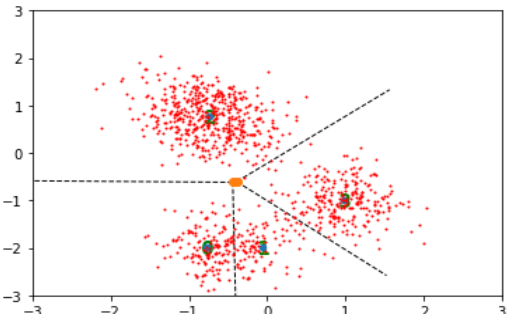
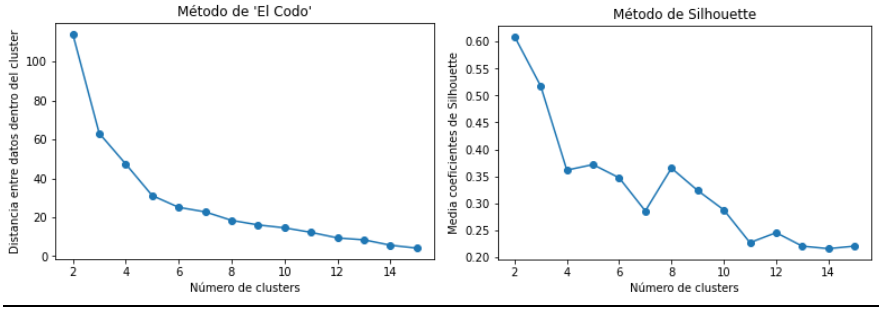


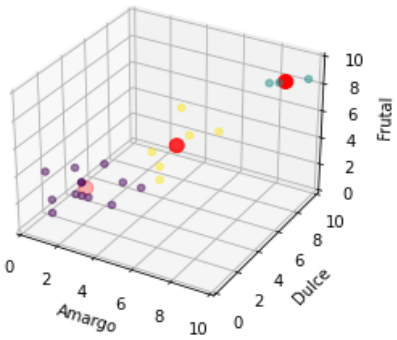
Diagrama de Voronoi para 4 clusters

Ejercicio iv): Se realiza una predicción que nos devolverá una tupla de las clases a la que pertenecen los puntos a y b. Los resultados encajan empíricamente con el diagrama de Voronoi anterior:

```
El par de elementos [0, 0] y [-0.5, -1] pertenecen a las
vencidades: [2 0] , respectivamente
```

Extra. Aplicación práctica: De forma análoga al **ejercicio ii)**, nosotros queremos clasificar esta vez en `n_clusters = 3` pero, en este caso, no maximiza el *coeficiente de Silhouette* (donde el pico máximo se alcanza para `n_clusters = 2`) y en cambio sí es una buena estimación por el *índice de 'El Codo'*. Esto se debe a que, dado que este último se basa también en la distancia intra-cluster, los puntos quedan claramente diferenciados (hay buena cohesión) en el interior de los clusters. Además, tomando `n_clusters = 3` todos los coeficientes de Silhouette son positivos (ejecutar código), lo que indica que están bien (aunque no lo mejor) clasificados, lo que será importante para predecir la calidad del producto.





Observación: En cuanto al índice del codo, sería una buena conclusión tomar `n_clusters = 5` (aunque no es buena conclusión para Silhouette) pero, dados nuestros propósitos, sirve más bien para verificar si tomar `n_clusters = 3` es una opción correcta, aunque no sea la mejor.

Bibliografía

Envoltura convexa. (s.f.). https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.ConvexHull.html

***Módulos Clustering*.** (s.f.). https://scikit-learn.org/stable/modules/clustering.html

***Gráficas Clustering*.** (s.f.). https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html

Método de "El Codo". (s.f.). *https://www.scikit-yb.org/en/latest/api/cluster/elbow.html*

Visualización gráfica del impacto de inicializar de una forma u otra el algoritmo k-medias. (s.f.). https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_stability_low_dim_dense.html#sphx-glr-auto-examples-cluster-plot-kmeans-stability-low-dim-dense-py

Clustering. Tablas Python. (s.f.). https://www.cienciadedatos.net/documentos/py20-clustering-con-python.html

Plot de Voronoi en 3D. Instalación del paquete plato-draw. (s.f.). https://freud.readthedocs.io/en/v1.2.0/examples/examples/Visualizing%203D%20Voronoi%20and%20Voxelization.html

Plot de Voronoi en 3D. Instalación del paquete freud. (s.f.). https://freud.readthedocs.io/en/v1.2.0/voronoi.html#freud.voronoi.Voronoi

```
# -*- coding: utf-8 -*-
"""
Plantilla 1 de la práctica 3

Referencia:
   https://scikit-learn.org/stable/modules/clustering.html
   https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
   https://docs.scipy.org/doc/scipy/reference/spatial.distance.html
"""

import numpy as np

from sklearn.cluster import KMeans
from sklearn import metrics
from sklearn.datasets import make_blobs
from scipy.spatial import ConvexHull
from scipy.spatial import Voronoi
import matplotlib.pyplot as plt
from scipy.spatial import voronoi_plot_2d #Para plotear el diagrama de Voronoi de iii).


#Para las tablas comparativas del ejercicio 2.
from sklearn.metrics import silhouette_score #Para la función k_medias_iteracion
#from sklearn.decomposition import PCA #Para mostrar el plot de Voronoi.
import matplotlib.cm as cm #Para el plot
from sklearn.metrics import silhouette_samples #Para calcular coeficiente de Silhouette en cada caso.
```

```
#############################################################################
#
# Aquí tenemos definido el sistema X de 1000 elementos de dos estados
# construido a partir de una muestra aleatoria entorno a unos centros:
initial_centers = [[-0.5, -2],  [1, -1], [-1, 1], [-0.5, 0.5]]
X, labels_true = make_blobs(n_samples=1000, centers=initial_centers, cluster_std=0.4,
                            random_state=0)

#Otra posibilidad sería generar valores totalmente aleatorios de esta forma
#X = np.random.rand(100, 2)   # 30 random points in 2-D

#Si quisieramos estandarizar los valores del sistema, haríamos:        #Acción previa para mejorar K-means.
#from sklearn.preprocessing import StandardScaler
#X = StandardScaler().fit_transform(X)

plt.plot(X[:,0],X[:,1],'ro', markersize=1)
plt.show()

# ##################### PARTE 1 ################################################

hull = ConvexHull(X)
print(hull.simplices)

#Código: SOLUCIÓN: Mostrar gráficamente la envoltura convexa

for simplex in hull.simplices: #Simplexes, generalización de la noción de triángulo
                               #o tetaedro a más dimensiones: 2-simplex triángulo, 3-simplex es tetraedro...
```

```
    plt.plot(X[simplex, 0], X[simplex, 1], 'ko-') #ko para que me señale los vértices.




plt.plot(X[:,0],X[:,1],'ro', markersize=1)
plt.show()

"""
#Caso para vecindades (neighbours). En una nueva gráfica:

for simplex in hull.neighbors:
    plt.plot(X[simplex, 0], X[simplex, 1], 'bo-') #bo para que me señale los vértices.

plt.plot(X[:,0],X[:,1],'ro', markersize=1)
plt.show()
"""
##############################################################################
##
##############################################################################
##
     ####################CÁLCULO DE CENTROIDES A MANO:####################
##############################################################################
##

def centroide(x,y):
    x_coord = 0
    y_coord = 0
    for i in range(len(x)):
        y_coord += y[i]
        x_coord += x[i]
    return x_coord/len(x), y_coord/len(x)
```

```
def distancia_a_centroides(centroides, X, Y):
    distancias = []

    c_x, c_y = centroides

    for x, y in list(zip(X, Y)):
        dist_x = (x - c_x) ** 2
        dist_y = (y - c_y) ** 2
        distancias = np.sqrt(dist_x + dist_y)
        distancias.append(distancias)

    return distancias
```

####DISTANCIAS sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.euclidean_distances.html

####DISTANCIAS PARA PARES DE PUNTOS: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.paired_distances.html#sklearn.metrics.pairwise.paired_distances

####GRÁFICAS clustering: https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html#sphx-glr-auto-examples-cluster-plot-kmeans-silhouette-analysis-py

#pdist(X[, metric]) Pairwise distances between observations in n-dimensional space.

#cdist(XA, XB[, metric]) Compute distance between each pair of the two collections of inputs.

```
# ###################### PARTE 2 ################################################

# Obtención del número de vecindades de Voronói óptimo según Silhouette

# Suponemos que es:

n_clusters = 4
```

```
kmeans = KMeans(n_clusters=n_clusters, init='random', n_init=1, random_state=0, max_iter=1000)

kmeans.fit(X)  #Entrenamos/Ajustamos el sistema para el conjunto de datos X.

#Consideramos init='random', pero otra posibilidad es considerar init='k-means++' (otro método de Python),

#donde podríamos obtener mejores resultados que con un inicio random, pues con

#una buena elección de los centroides podríamos acelerar la convergencia del

#algoritmo K-means en un tiempo y coste computacional razonable.



##############################Cálculo del coeficiente de Shilouette dadas las regiones o clusters y el conjunto X que se quiere estudiar:##############################


#######Cálculo del coeficiente de Silhouette para un rango de valores [2,3,4,5] de n_clusters:#######


"""

El método que se sigue aquí es el de la hoja:

  1. Asignar ALEATORIAMENTE k individuos para que sean los centroides de los clusters a formar.

  2. Calcular las distancias entre cada uno de los individuos restantes y los k centroides asignados aleatoriamente para asignarlos al cluster más cercano.

  3. Calcular los centroides de los clusters, es decir, el valor promedio que tienen los individuos que los conforman, en cada cluster.

  4. Reasignar a cada individuo el número de cluster cuyo centroide esté más cercano.


Se repiten los pasos 3 y 4 hasta que se cumpla el criterio de convergencia del algoritmo.

"""


lista_n_clusters = list(range(2,16))


def k_mediasv1(lista_n_clusters,X):

   if (lista_n_clusters == 1): #Necesitamos k > 1 clusters.
```

```
        return None
    for n_clusters in lista_n_clusters:
        kmeans = KMeans(n_clusters=n_clusters, init='random', n_init=1, random_state=0, max_iter=1000).fit(X)
        labels = kmeans.labels_
        silhouette = metrics.silhouette_score(X, labels, metric='euclidean') #Calcular coeficiente de Silhouette utilizando la métrica euclidiana.
        print("k_mediasv1: Para las {} regiones, el coeficiente de Silhouette es {})".format(n_clusters, silhouette))


k_mediasv1(lista_n_clusters,X)


blanco = ' ' * 3 #Espacios en blanco
print (blanco)


"""
k_mediasv1: Para las 2 regiones, el coeficiente de Silhouette es 0.6003510229447147)
k_mediasv1: Para las 3 regiones, el coeficiente de Silhouette es 0.6355332439272563)
k_mediasv1: Para las 4 regiones, el coeficiente de Silhouette es 0.547025605646528)
k_mediasv1: Para las 5 regiones, el coeficiente de Silhouette es 0.39495086718759276)
k_mediasv1: Para las 6 regiones, el coeficiente de Silhouette es 0.33581764210179893)
.
.
.
"""


"""
La diferencia con respecto al anterior es que aquí dejamos que el sistema haga predicciones
sobre el conjunto X pues el sistema no lo hemos "entrenado" (hacer correcciones, testeos sobre X)
previamente. Se utiliza el comando fit_predict().
```

"""

```
def k_mediasv2(lista_n_clusters,X):
   if (lista_n_clusters == 1): #Necesitamos k > 1 clusters.
      return None
   for n_clusters in lista_n_clusters:
      kmeans_preds = KMeans(n_clusters=n_clusters).fit_predict(X) #Entrenar/Ajustar y realizar predicciones sobre la información en X.
      total = metrics.silhouette_score(X, kmeans_preds, metric='euclidean')
      print("k_mediasv2: Para las {} regiones, el coeficiente de Silhouette es {})".format(n_clusters, total))

k_mediasv2(lista_n_clusters,X)

blanco = ' ' * 3 #Espacios en blanco
print (blanco)
```

"""

k_mediasv2: Para las 2 regiones, el coeficiente de Silhouette es 0.6003510229447147)

k_mediasv2: Para las 3 regiones, el coeficiente de Silhouette es 0.6355332439272563)

k_mediasv2: Para las 4 regiones, el coeficiente de Silhouette es 0.4781060805839868)

k_mediasv2: Para las 5 regiones, el coeficiente de Silhouette es 0.4702770612760678) #Aquí ya hay desviación con respecto a los valores de k_mediasv1

k_mediasv2: Para las 6 regiones, el coeficiente de Silhouette es 0.3890734029969042) #Aquí ya hay desviación con respecto a los valores de k_mediasv1

.

.

.

"""

#Ahora vamos a definir k_mediasv3 (k_medias versión 3), en el que consideraremos init='k-means++':

"""

La ventaja de la función k_mediasv3 inicializando con k-means++ es que es más probable obtener resultados

óptimos globales y no solo locales, debido al funcionamiento de esta forma de inicializar:

1. Se toma el primer centroide.
2. Se calculan distancias entre los puntos y el primer centroide.
3. Se establece un segundo centroide.
4. Se calculan distancias entre cada punto y su centroide MÁS CERCANO.
5. Se establece un tercer centroide...

"""

"""

El método que se sigue aquí es el siguiente:

1. Asignar k individuos para que sean los centroides de los clusters a formar. ¡Aquí NO hay aleatoriedad!
2. Calcular las distancias entre cada uno de los individuos restantes y los k centroides asignados aleatoriamente para asignarlos al cluster más cercano.
3. Calcular los centroides de los clusters, es decir, el valor promedio que tienen los individuos que los conforman, en cada cluster.
4. Reasignar a cada individuo el número de cluster cuyo centroide esté más cercano.

Se repiten los pasos 3 y 4 hasta que se cumpla el criterio de convergencia del algoritmo.

"""

```
def k_mediasv3(lista_n_clusters,X):
    if (lista_n_clusters == 1): #Necesitamos k > 1 clusters.
        return None
    for n_clusters in lista_n_clusters:
        kmeans = KMeans(n_clusters=n_clusters, init='k-means++', n_init=1, random_state=0, max_iter=1000).fit(X)
        labels = kmeans.labels_
        silhouette = metrics.silhouette_score(X, labels, metric='euclidean') #Calcular coeficiente de Silhouette utilizando la métrica euclidiana.
        print("k_mediasv3: Para las {} regiones, el coeficiente de Silhouette es {})".format(n_clusters, silhouette))
```

```
k_mediasv3(lista_n_clusters,X)

"""
k_mediasv3: Para las 2 regiones, el coeficiente de Silhouette es 0.6003510229447147)
k_mediasv3: Para las 3 regiones, el coeficiente de Silhouette es 0.6355332439272563)
k_mediasv3: Para las 4 regiones, el coeficiente de Silhouette es 0.47803684223993603)
k_mediasv3: Para las 5 regiones, el coeficiente de Silhouette es 0.47222448876696593)
"""

blanco = ' ' * 3 #Espacios en blanco
print (blanco)

#Hacemos unos plot, analizando qué coeficiente de Silhouette escoger para tomar el valor óptimo de n_clusters (que supusimos como n_clusters = 4).
#Comprobaremos que ese número de clusters no es el óptimo para el coeficiente de Silhouette s:


  #MÉTODO 1: El codo de Jambú:
lista_n_clusters = list(range(2,16))
distancias = []

for n_clusters in lista_n_clusters:
   modelo_kmeans = KMeans(n_clusters =  n_clusters, n_init = 1, random_state = 0)
   modelo_kmeans.fit_predict(X) #Entrenar y predecir sobre el sistema de datos X.
   distancias.append(modelo_kmeans.inertia_) #inertia_ es la suma de las distancias euclídeas de cada muestra de X al centroide más cercano de su cluster correspondiente.


fig, ax = plt.subplots(1, 1, figsize=(6, 3.84))
ax.plot(lista_n_clusters, distancias, marker='o')
```

```
ax.set_title("Método de 'El Codo'")

ax.set_xlabel('Número de clusters')

ax.set_ylabel('Distancia entre datos dentro del cluster')


  #MÉTODO 2: El coeficiente de Silhouette: Nos mostrará una gráfica que representa la evolución

  #media de los coeficientes de Silhouette.

lista_n_clusters = range(2, 16)

valores_medios_silhouette = []


for n_clusters in lista_n_clusters:

   modelo_kmeans = KMeans(n_clusters = n_clusters, n_init = 1, random_state = 0, max_iter=1000)


   cluster_labels = modelo_kmeans.fit_predict(X) #Calcular coeficiente de Silhouette utilizando la métrica euclidiana.

   silhouette_avg = metrics.silhouette_score(X, cluster_labels, metric='euclidean')

   valores_medios_silhouette.append(silhouette_avg)




fig, ax = plt.subplots(1, 1, figsize=(6, 3.84))

ax.plot(lista_n_clusters, valores_medios_silhouette, marker='o')

ax.set_title("Método de Silhouette")

ax.set_xlabel('Número de clusters')

ax.set_ylabel('Media coeficientes de Silhouette');


lista_n_clusters = list(range(2,6)) #No quiero tantos plots, por eso solo pongo [2, 3, 4, 5]:

for n_clusters in lista_n_clusters:

   # Hacer 4 plots (lista_n_clusters = [2,3,4,5]) con una fila y dos columnas:

   fig, (ax1, ax2) = plt.subplots(1, 2) #Preparar las dos gráficas
```

```
    fig.set_size_inches(18, 7)

    ax1.set_xlim([-1,1]) #El valor del coeficiente de Silhouette está entre -1 y 1.
    clusterer = KMeans(n_clusters=n_clusters, n_init=1, random_state=0)
    cluster_labels = clusterer.fit_predict(X)
    #silhouette_total calcula el valor medio de todo el sistema X:
    silhouette_total = metrics.silhouette_score(X, cluster_labels,  metric='euclidean')
    print("Para ", n_clusters, "regiones la media de silhouette_total es:", silhouette_total)

    sample_silhouette_values = silhouette_samples(X, cluster_labels) #silhouette_samples calculamos (b_i - a_i) / max{a_i, b_i} para cada i
    #Después en cada iteración se marcará con una línea horizontal roja la media en cada iteración (fórmula de la hoja).
    #(ver ax1.axvline más abajo...)
    y_lower = 10
    for i in range(n_clusters):
        # Agregar los coeficientes de Silhouette para los datos de X pertenecientes a
        #la k-ésima vecindad y ordenarlos de mayor a menor:
        ith_cluster_silhouette_values = \
            sample_silhouette_values[cluster_labels == i]

        ith_cluster_silhouette_values.sort() #Ordenar

        size_cluster_i = ith_cluster_silhouette_values.shape[0] #Acceder al elemento en la primera posición del array
        y_upper = y_lower + size_cluster_i

        color = cm.nipy_spectral(float(i) / n_clusters)
        ax1.fill_betweenx(np.arange(y_lower, y_upper), 0, ith_cluster_silhouette_values, facecolor=color, edgecolor=color, alpha=0.7)

        # Etiquetar con el correspondiente k en el medio de cada gráfica de barras:
```

```
    ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

    # Calcular el nuevo y_lower para el siguiente plot, así quedará centrado y no se "saldrá" la gráfica del plano:
    y_lower = y_upper + 10


  ax1.set_title("El plot de Silhouette para varias clusters")
  ax1.set_xlabel("Valores del coeficiente de Silhouette")
  ax1.set_ylabel("Cluster label")
  ax1.axvline(x=silhouette_total, color="red", linestyle="--") #La media del coeficiente de Silhouette en cada iteración

  ax1.set_yticks([])  # No quiero etiquetas en el eje y.
  ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1]) #"Escala" de etiquetas para el eje x.
  # El plot de la derecha donde se ilustran las particiones coloreadas:
  colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
  ax2.scatter(X[:, 0], X[:, 1], marker='.', s=30, lw=0, alpha=0.7, c=colors, edgecolor='k')

  # Etiquetar cada cluster:
  centers = clusterer.cluster_centers_

  for i, c in enumerate(centers): #Enumerar los centros de los clusters.
    ax2.scatter(c[0], c[1], marker='$%d$' % i, alpha=1, s=50, edgecolor='k')

  ax2.set_title("Particiones")
  ax2.set_xlabel("Eje x")
  ax2.set_ylabel("Eje y")

  plt.suptitle(("Análisis del coeficiente de Silhouette para n_clusters"
          " con n_clusters = %d" % n_clusters),
         fontsize=14, fontweight='bold')
```

```
plt.show()



# Índice de los centros de vencindades o regiones de Voronoi para cada elemento (punto)
centers = kmeans.cluster_centers_
y_kmeans = kmeans.predict(X)
labels = kmeans.labels_
silhouette = metrics.silhouette_score(X, labels)

print(labels)
print(centers)



#iii)

# Resultado de la clasificación de regiones (clusters) para n_clusters = 4:
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=5, cmap='summer')
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=100, alpha=0.5)
plt.show()

#Diagrama de Voronói (¡para más de dos centros!):

"""
Recordemos que el número óptimo de clusters es 3, no 4. Pero más abajo se realiza
el diagrama de Voronoi para n_clusters = 4.
"""
def Voronoiv2(n):
   kmeans = KMeans(n_clusters=n, init='random', n_init=1,random_state=0,
max_iter=1000).fit(X)
   centers = kmeans.cluster_centers_ #Calculamos centroides óptimos.
```

```
   figura = voronoi_plot_2d(Voronoi(centers),point_size=9) #Se muestran los centroides en
azul
   plt.plot(X[:,0],X[:,1],'ro', markersize=1)
   for i, c in enumerate(centers): #Enumerar los centros de los clusters.
      plt.scatter(c[0], c[1], marker='$%g$' % i, alpha=1, s=100, edgecolor='g')
   plt.xlim(-3, 3); plt.ylim(-3, 3)
   plt.show()

Voronoiv2(3)
Voronoiv2(4)



#iv)
kmeans = KMeans(n_clusters=3, init='random', n_init=1, random_state=0,
max_iter=1000).fit(X)
predecir = np.array([[0, 0], [-0.5, -1]])
prediccion = kmeans.predict(predecir)
print("El par de elementos [0, 0] y [-0.5, -1] pertenecen a las vencidades:",prediccion,",
respectivamente")



"""
El par de elementos [0, 0] y [-0.5, -1] pertenecen a las vencidades: [2 0], respectivamente.
"""



#############################################################################
#
#############################################################################
#
####################################EXTRA#####################################
#
```

```
################################################################################
#
################################################################################
#


"""
Queremos clasificar si un café es malo, normal o excelente a través de tres
de sus muchas propiedades: Amargo, dulce y frutal:
"""
from pandas import DataFrame #Para introducir tablas de datos


from mpl_toolkits.mplot3d import Axes3D




fig = plt.figure()
ax1 = fig.add_subplot(111, projection='3d')
ax1.set_xlabel('Amargo')
ax1.set_ylabel('Dulce')
ax1.set_zlabel('Frutal')

ax1.set_xlim(0,10)
ax1.set_ylim(0,10)
ax1.set_zlim(0,10)

Data = {'x': [9,8,10,5,6,7,2,4,5,1,2,1,0,3,4,5,2,1,3,1,3,2],
       'y': [8,9,9,6,5,6,4,5,4,3,2,1,2,3,4,4,2,1,1,3,4,2],
       'z': [9,8,9,7,6,6,3,4,3,1,2,1,3,1,2,4,3,2,3,3,2,3]
      }

#Data muestra por columnas las notas en las 3 propiedades del café.
```

```
df = DataFrame(Data,columns=['x','y','z'])


kmeans = KMeans(n_clusters=3).fit(df)
centroids = kmeans.cluster_centers_
print(centroids)



"""
[[5.33333333 5.         5.        ] Centroide 1.
 [1.92307692 2.46153846 2.23076923] Centroide 2.
 [9.         8.66666667 8.66666667]] Centroide 3.

Estos serán nuestros initial_centers
"""


ax1.scatter(df['x'], df['y'], df['z'], c= kmeans.labels_.astype(float), s=20, alpha=0.5)
ax1.scatter(centroids[:, 0], centroids[:, 1], centroids[:,2], c='red', s=80)




#################CLUSTERING 3D#################


lista_n_clusters = [2,3,4,5]
def k_mediasv3(lista_n_clusters,df):
   if (lista_n_clusters == 1): #Necesitamos k > 1 clusters.
       return None
   for n_clusters in lista_n_clusters:
      kmeans = KMeans(n_clusters=n_clusters, init='k-means++', n_init=1, random_state=0,
max_iter=1000).fit(df)
      labels = kmeans.labels_
      silhouette = metrics.silhouette_score(df, labels, metric='euclidean') #Calcular coeficiente
de Silhouette utilizando la métrica euclidiana.
```

```
    print("k_mediasv3: Para las {} regiones, el coeficiente de Silhouette es
{})".format(n_clusters, silhouette))


k_mediasv3(lista_n_clusters,df)


                        ####Método de 'El codo':


#¡En este caso el n_clusters óptimo es 3!

lista_n_clusters = list(range(2,16))

distancias = []


for n_clusters in lista_n_clusters:

  modelo_kmeans = KMeans(n_clusters =  n_clusters, n_init = 1, random_state = 0)

  modelo_kmeans.fit_predict(df) #Entrenar y predecir sobre el sistema de datos df.

  distancias.append(modelo_kmeans.inertia_) #inertia_ es la suma de las distancias euclídeas
de cada muestra de df al centroide más cercano de su cluster correspondiente.



fig, ax = plt.subplots(1, 1, figsize=(6, 3.84))

ax.plot(lista_n_clusters, distancias, marker='o')

ax.set_title("Método de 'El Codo'")

ax.set_xlabel('Número de clusters')

ax.set_ylabel('Distancia entre datos dentro del cluster')



#################################################################



lista_n_clusters = range(2, 16)

valores_medios_silhouette = []
```

```
for n_clusters in lista_n_clusters:
    modelo_kmeans = KMeans(n_clusters = n_clusters, n_init = 1, random_state = 0, max_iter=1000)

    cluster_labels = modelo_kmeans.fit_predict(df) #Calcular coeficiente de Silhouette utilizando la métrica euclidiana.
    silhouette_avg = metrics.silhouette_score(df, cluster_labels, metric='euclidean')
    valores_medios_silhouette.append(silhouette_avg)



fig, ax = plt.subplots(1, 1, figsize=(6, 3.84))
ax.plot(lista_n_clusters, valores_medios_silhouette, marker='o')
ax.set_title("Método de Silhouette")
ax.set_xlabel('Número de clusters')
ax.set_ylabel('Media coeficientes de Silhouette');



#######PLOT DE LA EVOLUCIÓN DEL CLUSTERING:#######

lista_n_clusters = list(range(2,6)) #No quiero tantos plots, por eso solo pongo [2, 3, 4, 5]:
for n_clusters in lista_n_clusters:
    # Hacer 4 plots (lista_n_clusters = [2,3,4,5]) con una fila y dos columnas:
    fig = plt.figure()
    fig.set_size_inches(22, 7)
    ax1 = fig.add_subplot(1,2,1) #Preparar las dos gráficas
    #fig, ax1 = plt.subplots(1,2)
    ax2 = fig.add_subplot(111, projection='3d')

    ax1.set_xlim([-0.5,1]) #El valor del coeficiente de Silhouette está entre -1 y 1.
```

```
    clusterer = KMeans(n_clusters=n_clusters, n_init=1, random_state=0)
    cluster_labels = clusterer.fit_predict(df)
    #silhouette_total calcula el valor medio de todo el sistema df:
    silhouette_total = metrics.silhouette_score(df, cluster_labels,  metric='euclidean')
    print("Para ", n_clusters, "regiones la media de silhouette_total es:", silhouette_total)

    sample_silhouette_values = silhouette_samples(df, cluster_labels) #silhouette_samples calculamos (b_i - a_i) / max{a_i, b_i} para cada i
    #Después en cada iteración se marcará con una línea horizontal roja la media en cada iteración (fórmula de la hoja).
    #(ver ax1.axvline más abajo...)
    y_lower = 10
    for i in range(n_clusters):
        # Agregar los coeficientes de Silhouette para los datos de df pertenecientes a
        #la k-ésima vecindad y ordenarlos de mayor a menor:
        ith_cluster_silhouette_values = \
            sample_silhouette_values[cluster_labels == i]

        ith_cluster_silhouette_values.sort() #Ordenar

        size_cluster_i = ith_cluster_silhouette_values.shape[0] #Acceder al elemento en la primera posición del array
        y_upper = y_lower + size_cluster_i

        color = cm.nipy_spectral(float(i) / n_clusters)
        ax1.fill_betweenx(np.arange(y_lower, y_upper), 0, ith_cluster_silhouette_values, facecolor=color, edgecolor=color, alpha=0.7)

        # Etiquetar con el correspondiente k en el medio de cada gráfica de barras:
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

        # Calcular el nuevo y_lower para el siguiente plot, así quedará centrado y no se "saldrá" la gráfica del plano:
```

```
        y_lower = y_upper + 10



    ax1.set_title("El plot de Silhouette para varias clusters")
    ax1.set_xlabel("Valores del coeficiente de Silhouette")
    ax1.set_ylabel("Cluster label")
    ax1.axvline(x=silhouette_total, color="red", linestyle="--") #La media del coeficiente de
Silhouette en cada iteración


    ax1.set_yticks([])  # No quiero etiquetas en el eje y.
    ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1]) #"Escala" de etiquetas para el eje x.
    # El plot de la derecha donde se ilustran las particiones coloreadas:
    colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
    ax2.scatter(df['x'], df['y'], df['z'], marker='.', s=120, lw=0, alpha=0.7, c=colors, edgecolor='k')


    # Etiquetar cada cluster:
    centroids = clusterer.cluster_centers_
    for i, c in enumerate(centroids):
      ax2.scatter(c[0],c[1],c[2], marker='$%d$' % i, c='red', alpha=1, s=80, edgecolor='k')


    ax2.set_title("Grupos de cafés")
    ax1.set_xlabel('Valores del coeficiente de Silhouette')
    ax2.set_xlabel('Amargo')
    ax2.set_ylabel('Dulce')
    ax2.set_zlabel('Frutal')
    plt.suptitle(("Análisis del coeficiente de Silhouette para n_clusters"
                  " con n_clusters = %d" % n_clusters),
                 fontsize=14, fontweight='bold')

plt.show()
```

```
##############################

"""
def Voronoiv2(n):
   kmeans = KMeans(n_clusters=n, init='random', n_init=1,random_state=0,
max_iter=1000).fit(df)
   centroids = kmeans.cluster_centers_ #Calculamos centroides óptimos.
   figura = Voronoi(centroids)
   plt.plot(df['x'], df['y'], df['z'], markersize=1)
   for i, c in enumerate(centroids): #Enumerar los centros de los clusters.
      plt.scatter(c[0], c[1],c[2], marker='$%g$' % i, alpha=1, s=100, edgecolor='g')
   #plt.xlim(0,10); plt.ylim(-3, 3)
   plt.show()

Voronoiv2(5)
Voronoiv2(6)
"""

"""
He tenido problemas para hacer un plot de un diagrama de Voronoi en 3D para
verificar los futuros predict que hagamos. Las referencias que he intentado
utilizar son las siguientes:

Instalación del paquete plato-draw:
https://freud.readthedocs.io/en/v1.2.0/examples/examples/Visualizing%203D%20Voronoi%20and%20Voxelization.html

Instalación del paquete freud:
https://freud.readthedocs.io/en/v1.2.0/voronoi.html#freud.voronoi.Voronoi
```

Esta aplicación es muy básica pues los sabores y aromas del café son mucho
más numerosos (véase "rueda de cata de café"), lo cual implicaría aumentar
el número de dimensiones considerablemente, pero quería hacerlo más "intuitivo".
Dado el problema descrito, he decidido hacer varios predict en
zonas "críticas" para comprobar cómo de bien se clasifica con n_clusters = 3
como hemos querido establecer.
"""

############PREDICCIONES:############

```
kmeans = KMeans(n_clusters=3, init='random', n_init=1, random_state=0,
max_iter=1000).fit(df)
predecir = np.array([[4,4,4]])
prediccion = kmeans.predict(predecir)
print(prediccion)

#Está en la clase [0]



kmeans = KMeans(n_clusters=3, init='random', n_init=1, random_state=0,
max_iter=1000).fit(df)
predecir = np.array([[6,6,6]])
prediccion = kmeans.predict(predecir)
print(prediccion)

#Está en la clase [0]

kmeans = KMeans(n_clusters=3, init='random', n_init=1, random_state=0,
max_iter=1000).fit(df)
predecir = np.array([[3,4,3]])
```

```
prediccion = kmeans.predict(predecir)

print(prediccion)


#Está en la clase [1].



kmeans = KMeans(n_clusters=3, init='random', n_init=1, random_state=0,
max_iter=1000).fit(df)

predecir = np.array([[0, 0, 0]])

prediccion = kmeans.predict(predecir)

print(prediccion)


#Está en la clase [1].



kmeans = KMeans(n_clusters=3, init='random', n_init=1, random_state=0,
max_iter=1000).fit(df)

predecir = np.array([[1,2,2]])

prediccion = kmeans.predict(predecir)

print(prediccion)


#Está en la clase [1]


kmeans = KMeans(n_clusters=3, init='random', n_init=1, random_state=0,
max_iter=1000).fit(df)

predecir = np.array([[1,1,1]])

prediccion = kmeans.predict(predecir)

print(prediccion)


#Está en la clase [1]
```

```
kmeans = KMeans(n_clusters=3, init='random', n_init=1, random_state=0,
max_iter=1000).fit(df)

predecir = np.array([[7,7,7]])

prediccion = kmeans.predict(predecir)

print(prediccion)


#Está en la clase [2]



kmeans = KMeans(n_clusters=3, init='random', n_init=1, random_state=0,
max_iter=1000).fit(df)

predecir = np.array([[8,8,8]])

prediccion = kmeans.predict(predecir)

print(prediccion)


#Está en la clase [2]


kmeans = KMeans(n_clusters=3, init='random', n_init=1, random_state=0,
max_iter=1000).fit(df)

predecir = np.array([[9,5,9]])

prediccion = kmeans.predict(predecir)

print(prediccion)



#Está en la clase [2]


kmeans = KMeans(n_clusters=3, init='random', n_init=1, random_state=0,
max_iter=1000).fit(df)

predecir = np.array([[10,10,10]])

prediccion = kmeans.predict(predecir)

print(prediccion)
```

#Está en la clase [2]

"""

En vista de lo anterior, se espera lo siguiente:

Notas muy altas entre 7-10: En la clase [2]

Notas muy bajas entre 0-4: En la clase [1]

Notas medias entre 4-6.9: En la clase [0]

"""