Práctica 2:
Análisis Relacional mediante Segmentación

UNIVERSIDAD DE GRANADA

Laura Lázaro Soraluce
lazarosoraluce@correo.ugr.es
Grupo 1
Inteligencia de Negocio

Índice:

Introducción:

Esta práctica trata de utilizar técnicas de clustering para analizar datos, para explorar y segmentar un conjunto de alojamientos turísticos en la ciudad de Granada. Los datos que usaremos han sido obtenidos de la plataforma de Booking.com.

El análisis de clustering sirve para identificar patrones en los datos y agrupar alojamientos que comparten características similares. Esto es de gran utilidad en el ámbito del turismo, pues puede ofrecer sugerencias útiles y acertadas a turistas e incluso a propietarios de alojamientos, permitiéndoles ajustar sus precios o servicios de acuerdo a los patrones que se hayan encontrado en el estudio.

El principal objetivo de esta práctica es analizar y segmentar los alojamientos turísticos en Granada basándonos en algunas de sus características principales, como el precio, la valoración, la distancia a puntos de interés turísticos y la cantidad de camas disponibles. Para ello, haremos varios casos de estudios diferentes y para cada uno utilizaremos 5 algoritmos, con K-means entre ellos. Sabemos que K-means es muy útil en esta práctica porque no sabemos de antemano cuántos clusters existen entre los alojamientos y K-means permite segmentar los datos en clusters de manera no supervisada, es decir, sin tener etiquetas asociadas a los datos.

Durante el proceso de clustering, primero haremos un preprocesado de los datos para eliminar valores faltantes o ruido en caso necesario. Además, normalizaremos las variables cuando haga falta para que las diferentes escalas de los atributos no afecten a los distintos algoritmos utilizados y por ende al resultado final. En cuanto al algoritmo de clustering, elegiremos el número de clusters más adecuado utilizando métricas de evaluación como el Índice de Calinski-Harabasz y el Coeficiente de Silueta, que nos permitirán medir la cohesión y separación de los clusters obtenidos.

Haremos tres casos de estudio principales. El primero se centrará en agrupar alojamientos que tengan valores similares en las categorías de *Precio, Rating, Review* y *Distance*. Se estudiará la relación de los clusters con la *Quality*. El segundo caso agrupará alojamientos en la zona del Albaicín y la zona del Centro de Granada, teniendo en cuenta su *Type, Price, Quality,* y *Guests.* Se explorará la distribución del precio y la calidad en cada zona, y la distribución del tipo de alojamiento por cada cluster. Finalmente, el tercer caso de estudio se enfocará en la clasificación de alojamientos en función de su posición en el ranking de Booking.com, para ver si atributos como el precio o la distancia al centro influyen en el ranking de esta plataforma.

Para los tres caso de estudio, los cuatro algoritmos adicionales que utilizaremos serán: Algoritmo jerárquico aglomerativo, que construye árboles de decisión para agrupar los datos de forma jerárquica; DBSCAN, que identifica densidad de puntos y las agrupa; Gaussian

Mixture, que usa un enfoque probabilístico que permite encontrar clusters que pueden tener formas no esféricas; y HDBSCAN, que …

Los datos que usaremos en este estudio son los proporcionados en los dos archivos CSV. El primer archivo, booking_Granada_2024.csv, contiene información sobre las características generales de los alojamientos, como su ubicación, el precio, las valoraciones y la distancia a los lugares de interés. El segundo archivo, alojamientos_booking_Granada_2024.csv, proporciona detalles adicionales sobre cada alojamiento, como el número de camas disponibles y su posición en el ranking de la plataforma.

Este análisis busca proporcionar una comprensión más profunda de cómo se agrupan los alojamientos turísticos en Granada y qué variables influyen en su segmentación, con el fin de ofrecer una herramienta útil tanto para turistas como para los responsables de la gestión de alojamientos turísticos.

Caso de Estudio 1:

En este caso de estudio, vamos a analizar cómo se agrupan los alojamientos en Granada que tienen al menos alguna cocina, según el precio, la distancia al centro, la cantidad de reseñas que tienen y su calificación. Esto está enfocado en clientes que busquen un poco más de autonomía, y que puedan contar con hacerse su propia comida en alguna ocasión. Este estudio me parece útil, ya que puede haber turistas que busquen alojamiento a un precio bajo aunque estén un poco más lejos del centro y sacrifiquen un poco de nota media, mientras que otro grupo de turistas puede preferir un lugar céntrico y una calificación elevada, aunque esto suponga un precio más alto.

Explicación del código:

```
variables_interes = ['Price', 'Rating', 'Distance', 'Review',
'Quality', 'Kitchens']
data = resultados_busqueda[variables_interes]
data = data[data['Kitchens']>0]

data_subset = data[['Price', 'Rating', 'Distance', 'Review']]
```

Lo primero que hago es seleccionar las variables que me van a ser de utilidad en este caso de estudio. Escojo *Price, Distance, Review* y *Rating* porque son las que voy a usar en los algoritmos de clustering. En otra variable diferente, cojo las mismas y también *Quality* y *Kitchens*, pues las voy a utilizar para hacer un análisis posterior, que me ayude a interpretar y sacar más información de los clusters. La columna *Kitchens* también la uso para filtrar los datos, y quedarme sólo con los alojamientos que dispongan de al menos una cocina. Todos estos datos se encuentran en el archivo *booking_Granada_2024.csv.*

```
data_subset = data_subset.dropna()
```

Este conjunto consta de 128690 datos, tras quitarle los alojamientos sin cocina. Como el conjunto es mayor que 10000, me acabaré quedando sólo con un 10% aleatorio de estos datos, por lo que el conjunto con el que finalmente trabajo tiene 12869 instancias.

Como otra parte del preprocesamiento, elimino todas las filas que contengan valores faltantes.

```
data_scaled = data_subset.apply(norm_to_zero_one)
```

Cuando trabajamos con algoritmos de clustering, siempre es recomendable normalizar los datos, sea cual sea el algoritmo que utilicemos [1]. Esto ayuda al clustering porque los grupos se forman a partir de distancias, y las escalas muy diferentes de algunos atributos, harán que

dominen los atributos de escala mayor en las distancias. Por ello, aplico normalización a los datos antes de todos los algoritmos que uso en este caso de estudio. Utilizo la función de normalización que nos dan como ejemplo.

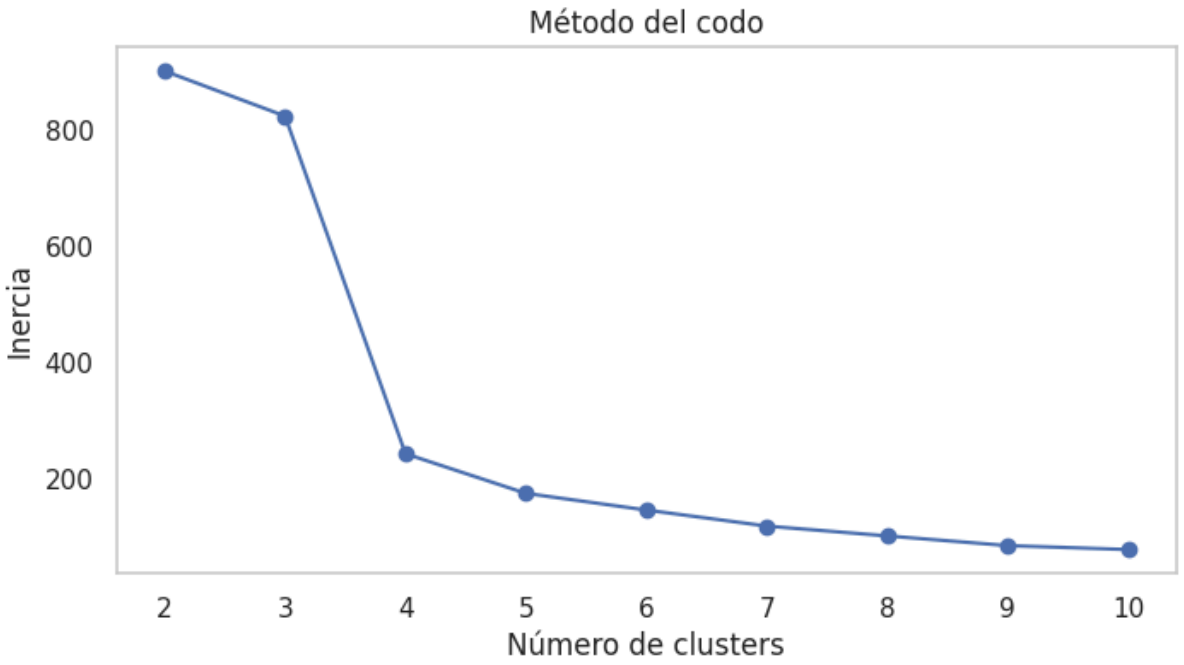
```
if len(data_subset) > 10000:
   sample_data = data_scaled.sample(frac=0.10, random_state=42)
else:
   sample_data = data_scaled
```

Como utilizo un algoritmo de Clustering jerárquico, usa demasiada RAM si trabajo con todos los datos seleccionados. Por ello, como se sugiere en el ejemplo, si tengo más de 10000 instancias, selecciono un 10% de los datos aleatoriamente y sigo trabajando sólo con esos.

Algoritmos utilizados:

K-Means: en este algoritmo, utilizo el método del codo para elegir el número de clústers óptimo [2].

```
inertia = []
k_range = range(2, 11)

for k in k_range:
   kmeans = KMeans(n_clusters=k, random_state=42)
   kmeans.fit(sample_data)
   inertia.append(kmeans.inertia_)

# Graficar el método del codo
plt.figure(figsize=(8, 4))
plt.plot(k_range, inertia, marker='o')
plt.title('Método del codo')
plt.xlabel('Número de clusters')
plt.ylabel('Inercia')
plt.grid()
plt.show()
```



Esto muestra el valor de la inercia para distintos valores de número de clústers. La inercia nos dice cuán dispersos o condensados están los clústers, cuando más baja, mejor será, pues significará que están más condensados. Así, fijándonos en el gráfico en cuándo la inercia deja de bajar drásticamente, podemos escoger el número de clústers. En este caso, he escogido 4.

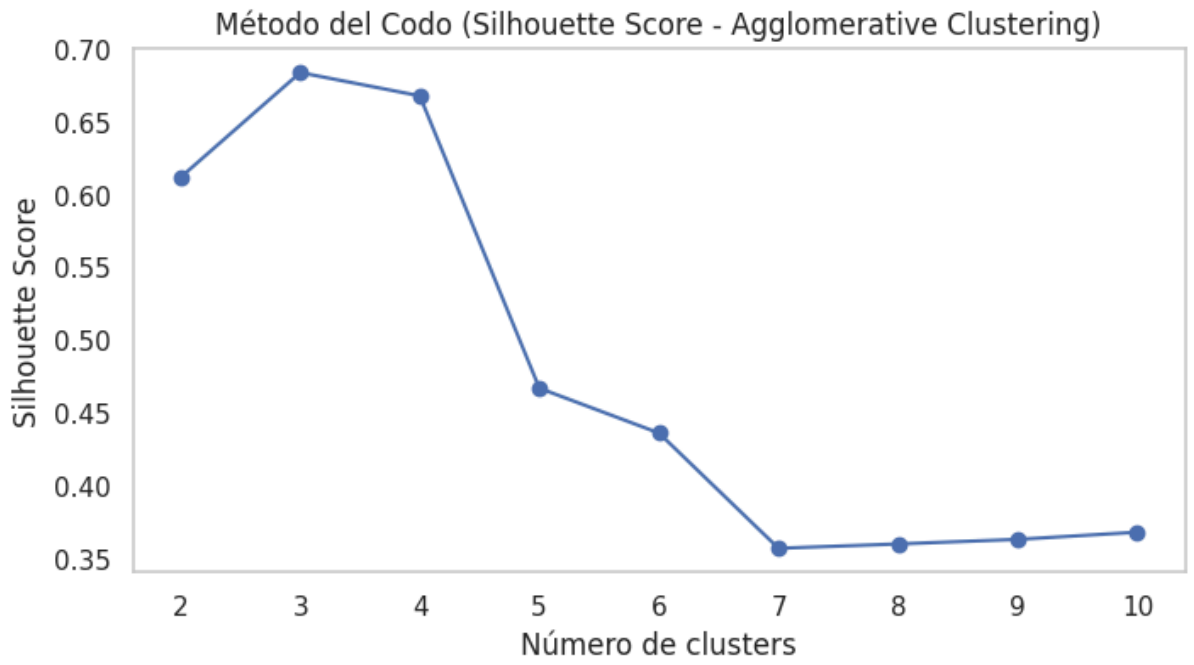
```
kmeans = KMeans(init='k-means++', n_clusters=4, n_init=5,
random_state=123456)
t = time.time()
kmeans_labels = kmeans.fit_predict(sample_data)
tiempo = time.time() - t
print(f"\nTiempo de K-Means: {tiempo} segundos")
```

En todos los algoritmos mido el tiempo que tarda en ejecutarse, después de haber sido configurado.

Clustering Jerárquico: Aquí también hago un pequeño estudio, comparando los distintos valores de silhouette score que se obtienen para distintos valores de número de clusters. Esto lo hago porque vi que escoger el mismo número de clusters que con K-Means (lo que hubiese sido mi primer instinto) no daba un resultado demasiado bueno.

```
scores = []
k_range = range(2, 11)  # Rango de clusters a probar

for k in k_range:
   clustering = AgglomerativeClustering(n_clusters=k)
   labels = clustering.fit_predict(sample_data)
```

```
            score    =    metrics.silhouette_score(sample_data,    labels,
metric='euclidean', sample_size=len(sample_data), random_state=123456)
    scores.append(score)

# Graficar el método del codo para Silhouette Score
plt.figure(figsize=(8, 4))
plt.plot(k_range, scores, marker='o')
plt.title('Método   del   Codo   (Silhouette   Score   -   Agglomerative
Clustering)')
plt.xlabel('Número de clusters')
plt.ylabel('Silhouette Score')
plt.grid()
plt.show()
```



En este caso, escojo n_clusters = 3, que claramente tiene el mayor silhouette score.

```
hierarchical = AgglomerativeClustering(n_clusters=3)
t = time.time()
hierarchical_labels = hierarchical.fit_predict(sample_data)
tiempo = time.time() - t
print(f"Tiempo de Jerárquico: {tiempo} segundos")
```

DBSCAN: es adecuado cuando los datos pueden contener ruido o puntos atípicos. Podría darse el caso de que esto sucediese, pues podría haber algún alojamiento que esté lejos del centro pero sea más caro que los demás por su tamaño, por ejemplo, aunque su puntuación tampoco sea demasiado buena… Este algoritmo identifica clusters de forma arbitraria e

identifica puntos ruidosos que no pertenecen a ninguno. Una desventaja de este algoritmo es que es difícil ajustar los parámetros de configuración, pues una ligera variación puede producir resultados muy distintos.

```
dbscan = DBSCAN(eps=0.25, min_samples=8)
t = time.time()
dbscan_labels = dbscan.fit_predict(sample_data)
tiempo = time.time() - t
print(f"Tiempo de DBSCAN: {tiempo} segundos")
```

Gaussian Mixture: es adecuado para este problema porque no supone que los clústers tengan una forma esférica, a diferencia de K-Means. Sin embargo, asume que los clusters siguen distribuciones gaussianas, lo cual no tiene por qué ser cierto para todos los datos.

```
gmm = GaussianMixture(n_components=3, random_state=123456)
t = time.time()
gmm_labels = gmm.fit_predict(sample_data)
tiempo = time.time() - t
print(f"Tiempo de Gaussian Mixture: {tiempo} segundos")
```

He probado varios valores de n_components dentro de un rango cercano a 2, hasta encontrar el que ofrecía mejores resultados.

HDBSCAN: es una extensión de DBSCAN que además puede manejar clusters con diferentes densidades lo que es crucial si los datos incluyen alojamientos con distintas características de distancia, precio, o calificación. Tiene además la capacidad de etiquetar puntos que no se ajustan bien a ningún cluster como ruido. Como ajusta automáticamente el parámetro *eps*, evita estar teniendo que decidir cuál escoger y poder equivocarnos. Para los valores de la configuración, he probado varios y he visto cómo afectan a las métricas.

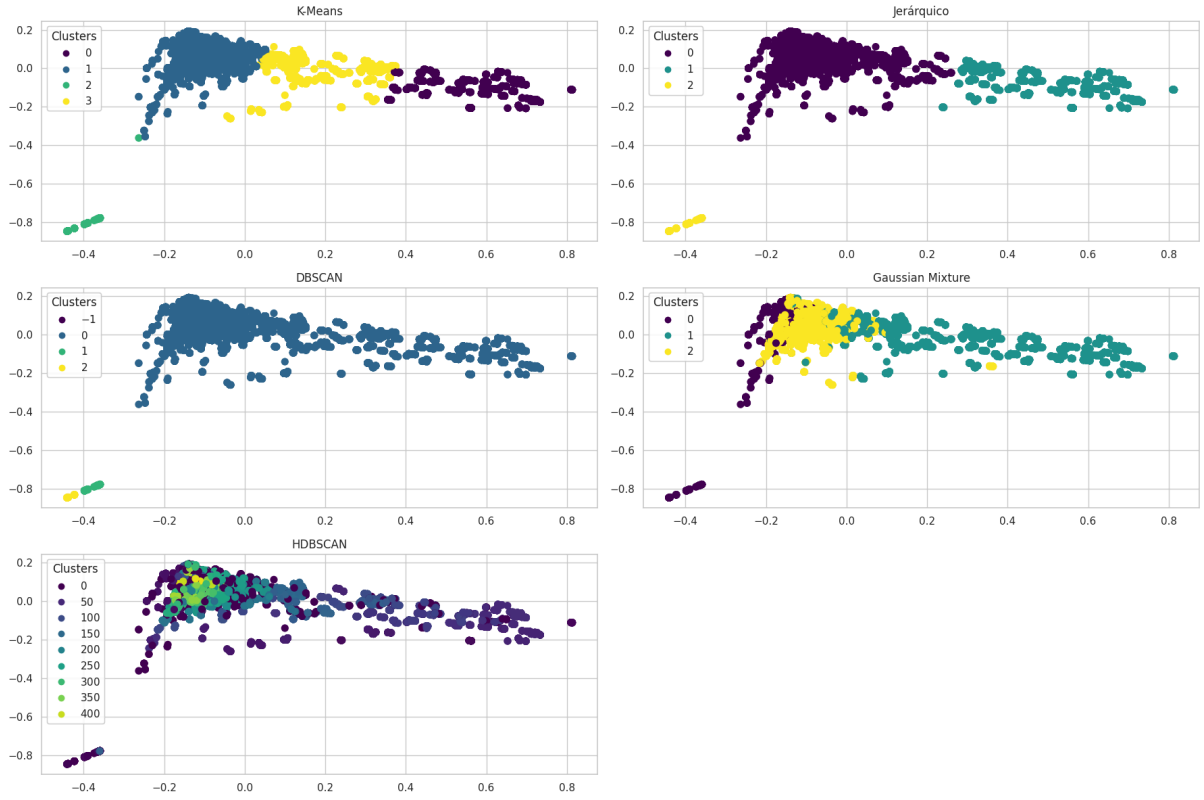
```
hdbscan = HDBSCAN(min_cluster_size=10, min_samples=8)
t = time.time()
hdbscan_labels = hdbscan.fit_predict(sample_data)
tiempo = time.time() - t
print(f"Tiempo de HDBSCAN: {tiempo} segundos")
```

Para evaluar los clusters, utilizo PCA para reducir la dimensionalidad a 2, pues es necesario para representarlos en los gráficos.

```
pca = PCA(n_components=2)
data_reduced = pca.fit_transform(sample_data)
```

Análisis de resultados:

Algoritmo	Tiempo de ejecución	Silhouette Coefficient	Calinski-Harabasz Index	Número de clusters
K-Means	0.046s	0.622	37526.788	4
Clustering Jerárquico	9.668s	0.745	31963.925	3
DBSCAN	1.742s	0.72	2306.080	4
Gaussian Mixture	0.253s	0.391	5593.045	3
HDBSCAN	1.683s	0.333	100.558	401



Basándonos en los resultados de las métricas que podemos ver en la tabla y en los gráficos, vemos que K-Means produce 4 clusters bien definidos y distribuidos de forma uniforme, algo que se refleja en su alto índice de Calinski-Harabasz.

El clustering jerárquico produce 3 clusters, uno de ellos similar a uno de los de K-Means, los otros un poco más diferentes. Tiene un mejor Silhouette Score, por lo que los puntos están más cerca del centro de su cluster que en K-Means. Consume mucho más tiempo debido al

cálculo de distancias jerárquicas. Ofrece más interpretabilidad, ya que permite visualizar las relaciones jerárquicas (dendrogramas).

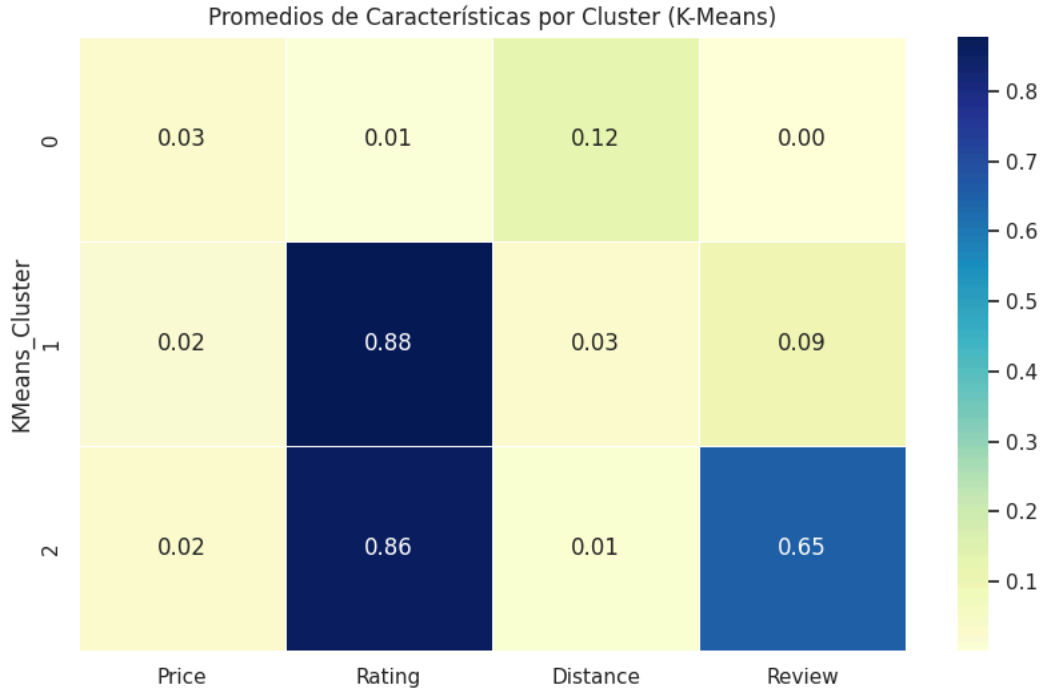
DBSCAN genera también 4 clusters, y la mayor parte de los datos están en el cluster "0". Detecta puntos como ruido, lo cual es útil. El Silhouette Coefficient es similar al del algoritmo anterior. El coeficiente de Calinski-Harabasz es menor, lo que sugiere que el cluster no es tan compacto [3].

Gaussian Mixture utiliza distribuciones gaussianas para modelar los clusters. Los resultados son menos definidos que los del algoritmo jerárquico, y vemos cierta superposición entre los 3 clusters que se generan. Este algoritmo es útil cuando se espera que los clusters se distribuyan de manera probabilística. Aquí, la baja separación y cohesión de clusters reduce sus métricas.

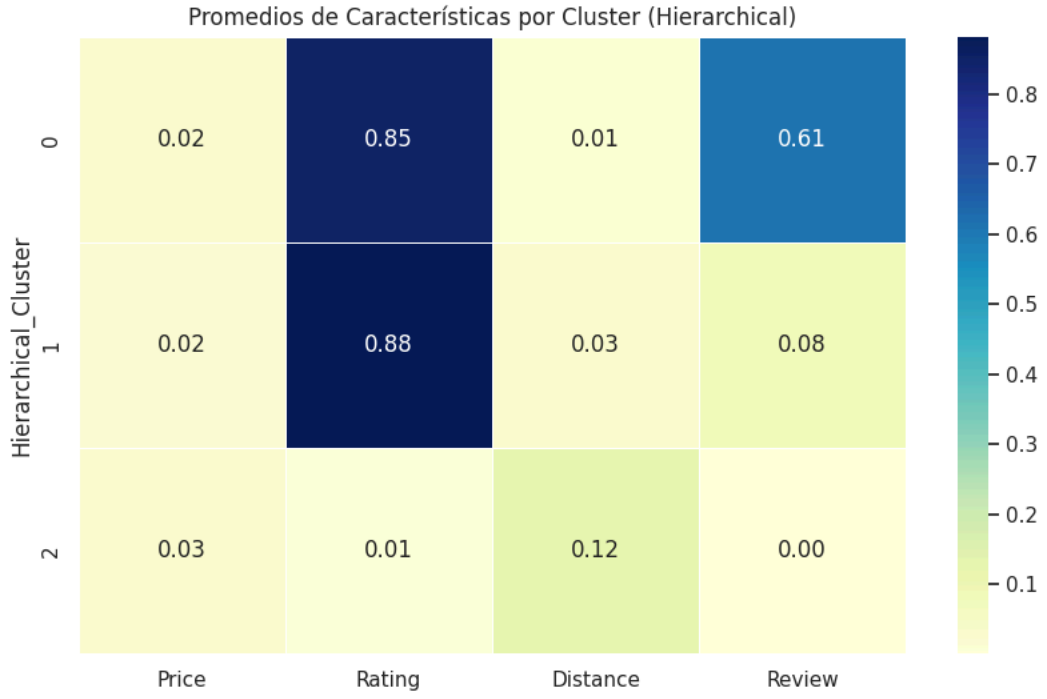
HDBSCAN identifica muchos clusters pequeños, reflejados en el gráfico, donde varios clusters tienen muy pocos puntos. Este comportamiento es adecuado para datos con densidades variadas y ruido, pero las métricas son bajas debido a la fragmentación excesiva.

Concluimos así que el algoritmo de clustering jerárquico es el algoritmo más adecuado para este problema por sus métricas sólidas. Es cierto que sacrificamos un poco de tiempo de cálculo, pero no es excesivo, y merece la pena por la ganancia en métricas. Si quisiesemos un algoritmo más rápido, podríamos quedarnos con K-Means, aunque habría puntos que no están tan cerca del núcleo del cluster al que se han asignado.

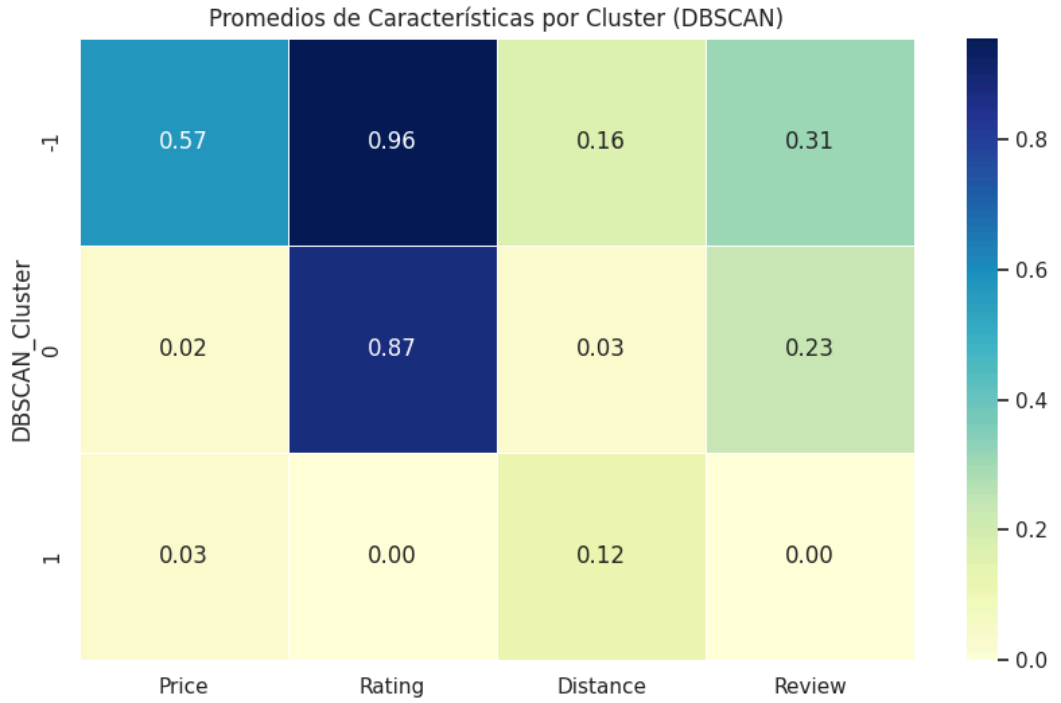
Interpretación de los resultados:
Una vez formados los clusters, he decidido hacer algunos análisis adicionales.



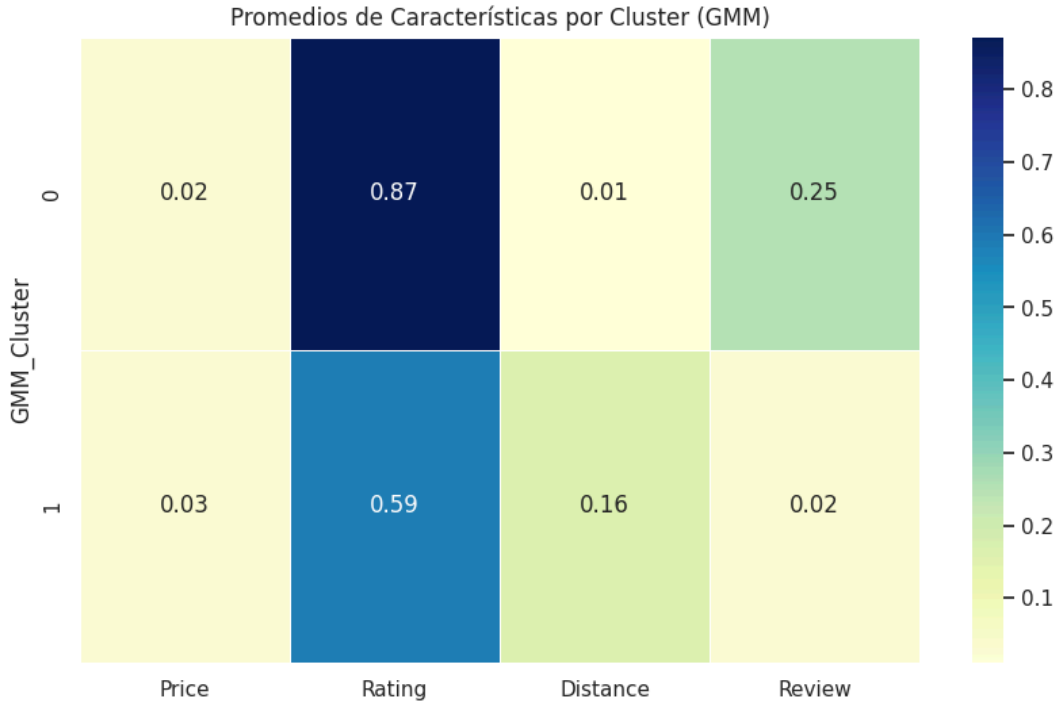
Este heatmap muestra los valores promedios de cada variable para cada cluster. El cluster 0 muestra el valor más alto de precio promedio, los ratings más bajos y la distancia más alta. El hecho de que estos valores sean tan extremos en comparación con los otros dos clusters puede sugerir que se trate de valores outliers o ruido. El cluster 1 y 2 tienen el mismo precio promedio. El cluster 1 contiene alojamientos mejor calificados pero más lejanos al centro. Por otro lado, el cluster 2 contiene alojamientos más populares, ya que tienen una cantidad de reseñas mucho mayor, pudiendo ser este un gran influyente en la agrupación en clusters.



Tiene resultados similares a K-Means. El cluster 2 se asemeja al cluster 0 de K-Means, podría simular ruido o outliers. El cluster 1 es equivalente al 1 de K-Means, y el 0 es equivalente al 3 de K-Means.



El cluster 1 parece tener también valores outliers. El cluster -1 contiene pocos alojamientos, más caros y lejanos, pero muy bien valorados y más populares. El cluster 0 contiene alojamientos más asequibles y céntricos, pero sacrificando un poco de nota y de número de reviews.



Genera sólo 2 clusters que se solapan un poco más que los algoritmos anteriores. El cluster 0 contiene alojamientos más asequibles, muy céntricos y mejor valorados. El segundo cluster contiene "peores" alojamientos, pues son más caros, periféricos, peor valorados y menos conocidos.

```
Estadísticas promedio por cluster (HDBSCAN):
                     Price    Rating  Distance    Review
HDBSCAN_Cluster
-1                0.033364  0.828737  0.023519  0.204982
 0                0.039283  0.000000  0.409582  0.000000
 1                0.018692  0.000000  0.679487  0.000000
 2                0.027813  0.000000  0.679487  0.000000
 3                0.033925  0.000000  0.846491  0.000000
...                    ...       ...       ...       ...
 700              0.009436  0.920000  0.013720  0.013903
 701              0.011653  0.920000  0.003940  0.069562
 702              0.013325  0.920000  0.007609  0.073261
 703              0.017335  0.900000  0.011056  0.003092
 704              0.006372  0.900000  0.004892  0.007973
```

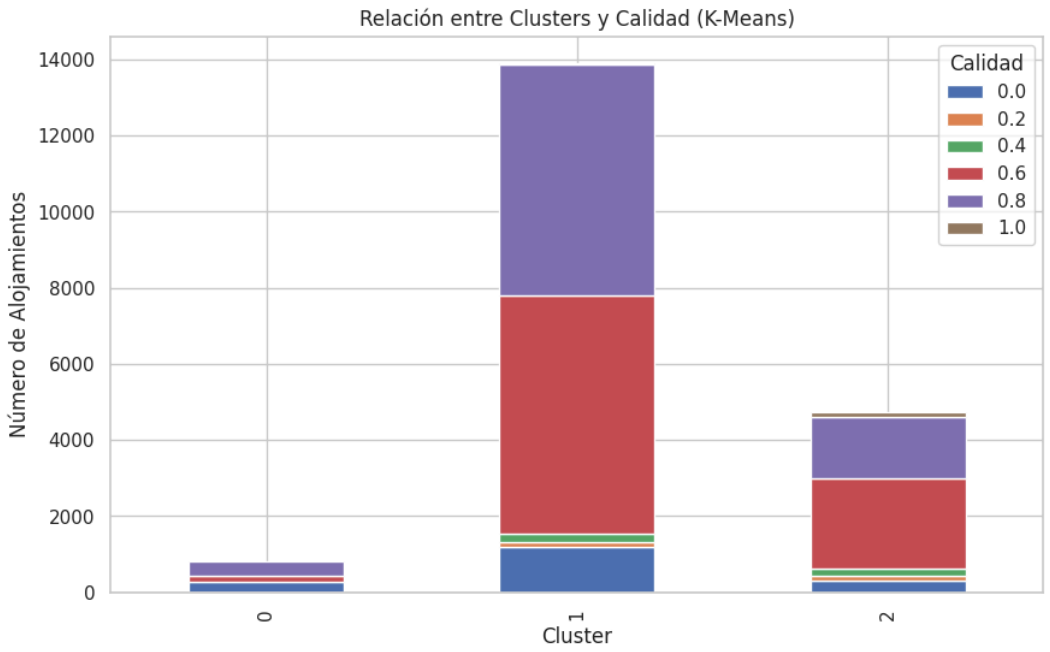
Como este algoritmo generó muchos clusters pequeños, no es fácilmente interpretable con un heatmap, por lo que he obtenido los valores en una tabla que muestra la información de los primeros y últimos clusters. Muchos presentan valores atípicos, indicando que los parámetros elegidos pueden no ser los ideales para este método, a pesar del estudio previo que hicimos.

Para el algoritmo de K-Means he hecho algunos estudios más:

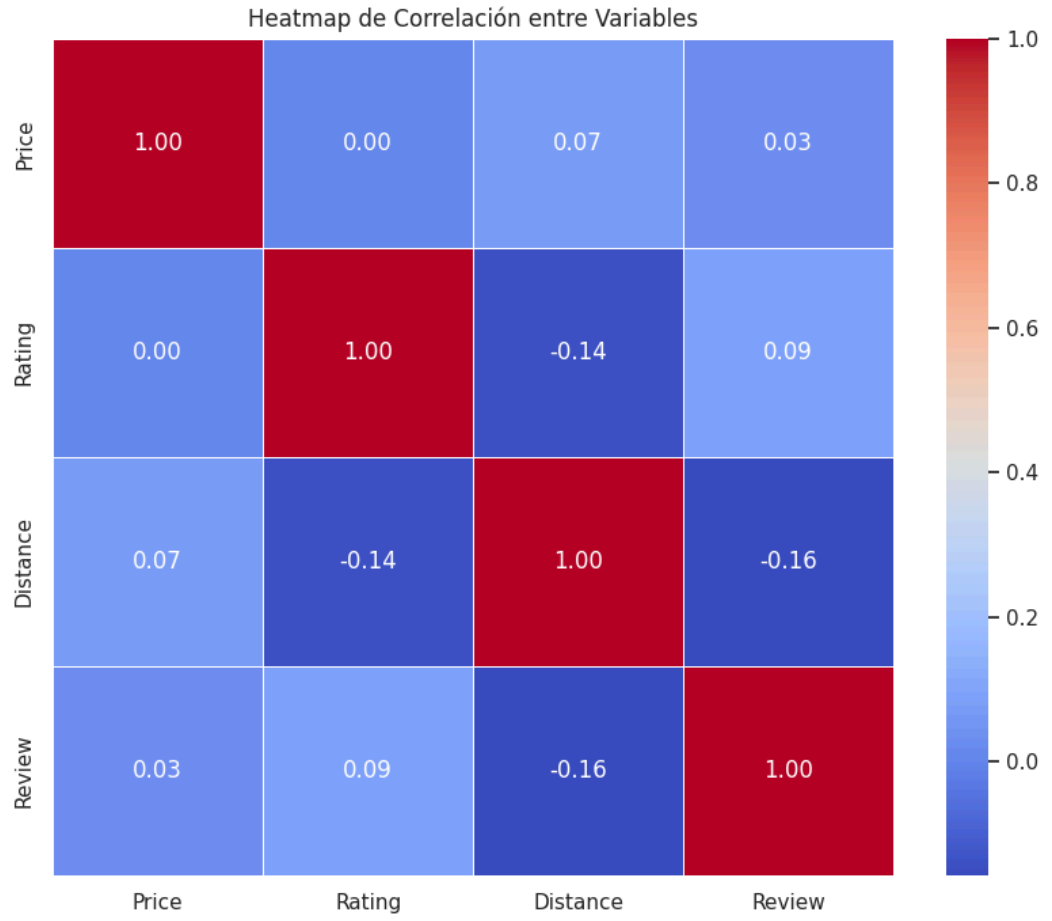
```
Distancia promedio al centroide por cluster (K-Means):
KMeans_Cluster
0    0.174201
1    0.118705
2    0.156480
```

El cluster 0 tiene la mayor distancia promedio (0.174201), lo que sugiere mayor dispersión dentro del mismo. Esto respalda que este grupo representa puntos menos homogéneos (posiblemente ruido o datos atípicos). El cluster 1 tiene la distancia más baja (0.118705), lo que indica un cluster compacto y coherente, es decir, los alojamientos de alta calificación.

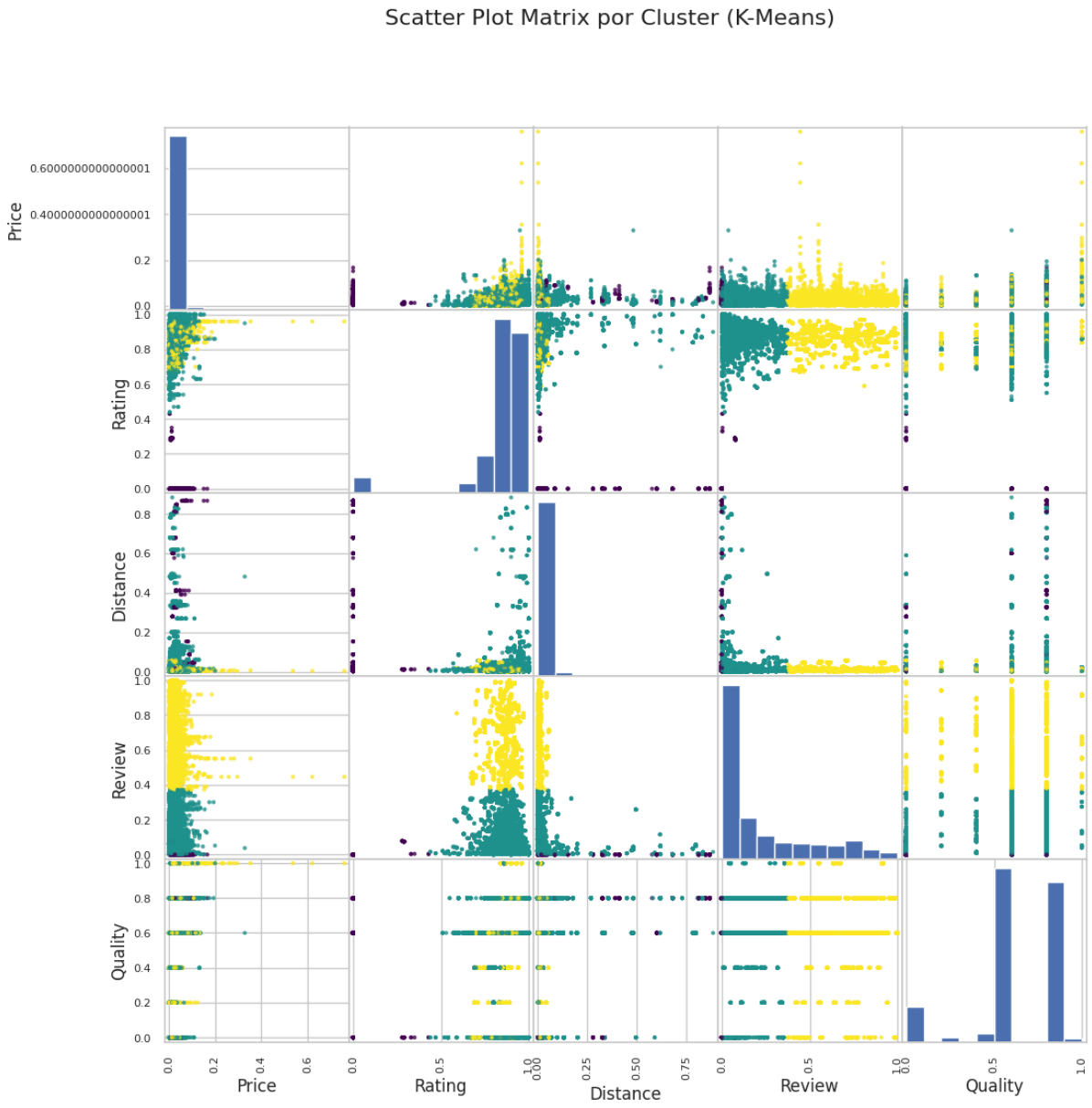
```
Relación entre K-Means Clusters y Calidad:
Quality          0.0  0.2  0.4   0.6   0.8  1.0
KMeans_Cluster
0                276    0    0   154   369    0
1               1185  122  215  6279  6060   51
2                311  111  203  2356  1616  142
```



Aquí, tanto con la tabla como con el gráfico, estudio la relación entre los clusters de K-Means y un campo externo a los clusters: *Quality*. Esto lo hago para ver si hay algún cluster que coincida que tenga una calidad más alta o más baja que los demás. En el cluster 0 hay alojamientos tanto de muy buena calidad como de muy mala (34.5% con calidad 0.0, 46.2% con calidad 0.8). Es un cluster poco claro, como ya veníamos viendo. En el cluster 1, es más grande, teníamos el rating promedio más alto. Vemos ahora que la mayoría de los alojamientos tienen calidad 0.6 o 0.8. Esto concuerda con lo que ya sabíamos, confirmando que el cluster representa alojamientos de buena calidad. En el cluster 2, hay alojamientos de calidad media y alta, pero también están aquí todos los alojamientos de calidad 1.0. Recordamos que todas los datos están normalizados, por ello sólo van de 0 a 1 cuando originalmente iban de 0 a 5.



Este heatmap nos permite ver cómo se relacionan las variables entre sí. Vemos que *Price* no parece correlacionarse significativamente con las otras variables. Esto puede ser porque el precio no influye directamente en la distancia o el rating. Por ejemplo, puede haber apartamentos caros con malas puntuaciones y apartamentos más asequibles en los que ocurra igual. *Distance* muestra correlaciones débiles y negativas con *Rating* y *Review*, indicando que la cercanía no influye demasiado en las reseñas o la calificación, pero cuanto más céntrico está el alojamiento, suele ser mayor el rating y el número de reviews.



Vemos que el atributo que tiene una división más clara entre los distintos clusters, es *Review*, indicando que uno de los dos clusters con mayor número de instancias se encuentran alojamientos más frecuentados o más conocidos, y en el otro, alojamientos más nuevos o desconocidos. Esto es porque este atributo influye mucho en la agrupación por clusters. Este gráfico nos permite también ver cómo están distribuidos los alojamientos dentro de cada atributo, por ejemplo, la mayoría de alojamientos tienen un rating alto, están muy céntricos y no son de precio elevado. En cuanto a la calidad, hay más variedad. Hay muchos más alojamientos que tienen pocas reseñas, que los que tienen muchas. El campo *Rating* también muestra una división bastante buena, estando en un cluster los alojamientos de menor calificación o más nuevos, y en los otros dos bastante más mezclados.

Interpretación de la segmentación:
Para K-Means por ejemplo, podemos decir que el cluster 0 contiene alojamientos periféricos, nuevos, y puede tener outliers. El cluster 1 tiene alojamientos menos conocidos de calidad medio-alta, un poco más lejos del centro y muy bien valorados. Perfecto para huśpedes que prefieran alojamientos más económicos aún si sacrifican centricidad, pero con muy buena calificación. El cluster 2 tiene alojamientos muy céntricos y populares. Este es ideal para turistas cuya prioridad sea estar en el centro de la ciudad en un alojamiento fiable.

Caso de Estudio 2:

En este caso de estudio, vamos a comparar los alojamientos en el centro de Granada y en el barrio del Albayzín. Analizaremos cómo se agrupan según tipo, calidad, precio, medio y clientes. Esto me parece útil, ya que puede haber turistas que puede que los turistas busquen alojamientos en uno de estos dos barrios específicamente, y es interesante poder proponerles alojamientos específicos basándonos en su perfil. Queremos además determinar si hay diferencias significativas en las características de los alojamientos entre los dos barrios.

De nuevo, como en el caso de estudio anterior, los datos que voy a utilizar están en el archivo *booking_Granada_2024.csv*.

```
variables_interes = ['Type', 'Quality', 'Price', 'Location', 'Guests']
data_subset = resultados_busqueda[variables_interes]

zonas_filtradas = ['Albaicín, Granada', 'Centro de Granada, Granada']
data_filtrada                                                        =
data_subset[data_subset['Location'].isin(zonas_filtradas)]

data_subset = data_filtrada[['Type', 'Quality', 'Price', 'Guests']]
```

Escojo las variables *Type, Quality, Price* y *Guests*, pues son las que voy a estudiar como ya había mencionado. Me aseguro de filtrar las zonas para sólo estudiar alojamientos del Albaicín y del centro de Granada. Guardaré en una variables los datos de *Location* también, pues los utilizaré después de aplicar los algoritmos, para hacer un estudio más completo. También me deshago de las columnas que contengan missing values.

En principio, este subset de datos con alojamientos sólo en las zonas del Albaicín y del Centro de Granada, tiene 113620 instancias. Como el conjunto es mayor que 10000, me acabaré quedando sólo con un 10% aleatorio de estos datos, por lo que el conjunto con el que finalmente trabajo tiene 11362 instancias.

```
numerical_cols = ['Price', 'Guests', 'Quality']
data_scaled = data_subset[numerical_cols].apply(norm_to_zero_one)
data_scaled = pd.concat([data_subset[['Type']], data_scaled], axis=1)
```

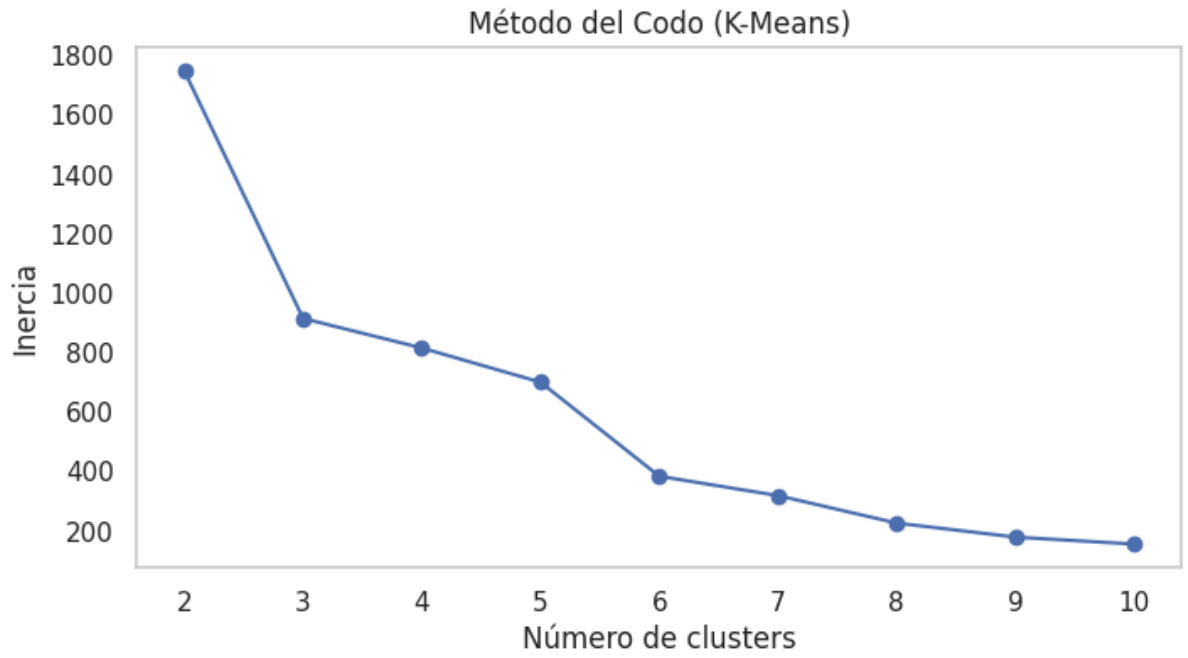
Aplico la normalización de sólo a las columnas numéricas y luego concateno con las categóricas.

```
encoder = ce.OneHotEncoder(cols=['Type'], use_cat_names=True)
sample_data_encoded = encoder.fit_transform(sample_data)
```

Como los algoritmos de clustering sólo trabajan con variables numéricas y *Type* es categórica, utilizo OneHotEncoder para convertir cada tipo de alojamiento en una columna nueva.

Algoritmos utilizados:

K-Means:



Como en el caso anterior, utilizo el método del codo para saber cuántos clusters utilizar con K-Means. En este caso escojo 3, pues vemos que a partir de ahí, la mejora en inercia no es tan significativa.

```
kmeans = KMeans(init='k-means++', n_clusters=3, n_init=5,
random_state=123456)
t = time.time()
kmeans_labels = kmeans.fit_predict(sample_data)
tiempo = time.time() - t
print(f"\nTiempo de K-Means: {tiempo} segundos")
```

Jerárquico:

```
hierarchical = AgglomerativeClustering(n_clusters=3)
t = time.time()
hierarchical_labels = hierarchical.fit_predict(sample_data)
tiempo = time.time() - t
print(f"Tiempo de Jerárquico: {tiempo} segundos")
```

Escojo el mismo número de clusters que he escogido en K-Means.

DBSCAN:

```
dbscan = DBSCAN(eps=0.4, min_samples=5)
t = time.time()
dbscan_labels = dbscan.fit_predict(sample_data)
tiempo = time.time() - t
print(f"Tiempo de DBSCAN: {tiempo} segundos")
```

Para la elección de estos parámetros, hice varias pruebas similares a las que hago en HDBSCAN, para ver cómo afectaba su cambio a los resultados.

Gaussian Mixture:

```
gmm = GaussianMixture(n_components=2, random_state=123456)
t = time.time()
gmm_labels = gmm.fit_predict(sample_data)
tiempo = time.time() - t
print(f"Tiempo de Gaussian Mixture: {tiempo} segundos")
```

HDBSCAN:

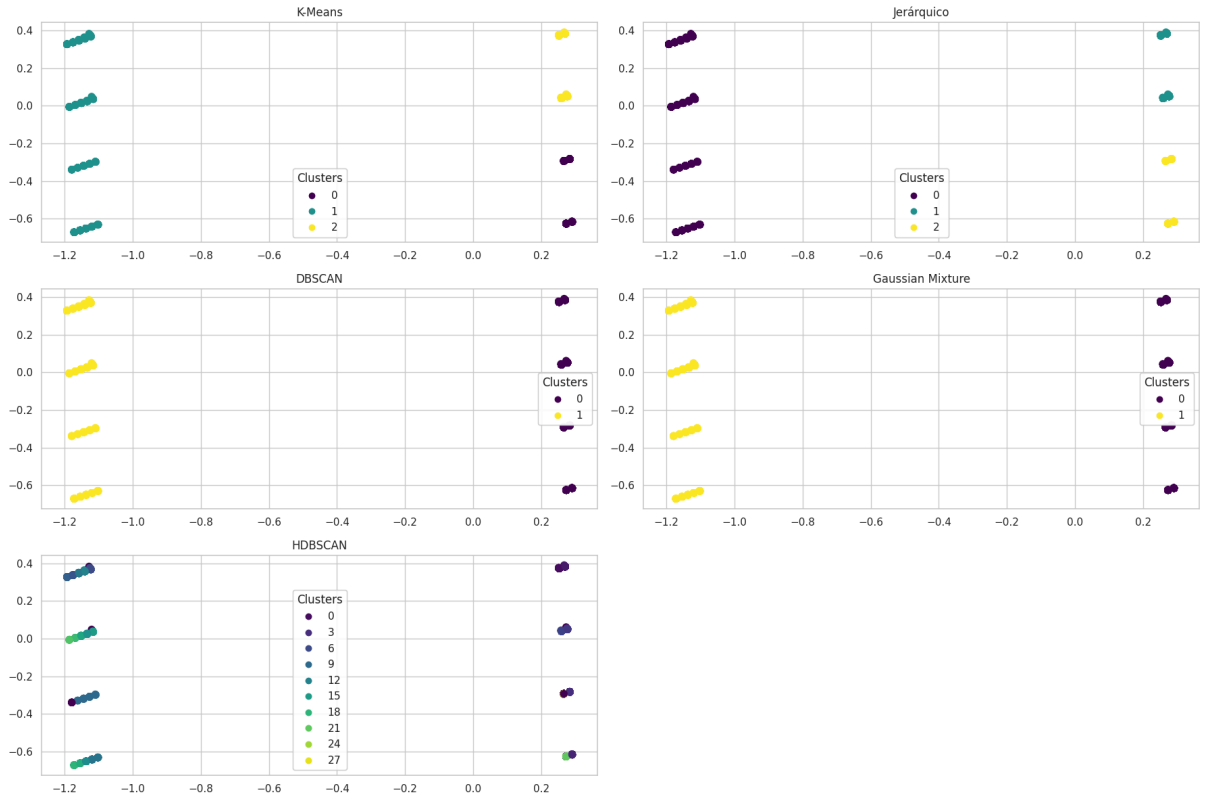
min_samples	Silhouette Score	Clusters
15	0.413	60
20	0.539	30
25	0.694	30
30	0.662	32

De la misma manera que en el caso de estudio 1, calculo qué valor para min_samples puede funcionar bien en HDBSCAN con nuestros datos. Escojo min_samples = 25, pues es el que mejor Silhouette Score tiene dentro de los valores que he probado, y además no tiene un número de clusters excesivo.

Para evaluar los clusters, utilizo PCA para reducir la dimensionalidad a 2, pues es necesario para representarlos en los gráficos.

Análisis de resultados:

Algoritmo	Tiempo de ejecución	Silhouette Coefficient	Calisnki-Harabasz Index	Número de clusters
K-Means	0.026s	0.71581	39704.406	3
Jerárquico	13.684	0.71581	39704.406	3
DBSCAN	1.546	0.74555	27401.255	2
Gaussian Mixture	0.044	0.74555	27401.255	2
HDBSCAN	3.601	0.83864	55217.489	28



En K-Means tiene el tiempo de ejecución más rápido. El Calinski-Harabasz Index nos indica que tenemos buena separación de clusters (también vemos en el gráfico que tenemos los clusters bien definidos).

Los resultados del algoritmo jerárquico son equivalentes a los de K-Means, pero el tiempo de cálculo es mucho mayor, lo que lo hace menos eficiente. En el gráfico vemos que K-Means y Jerárquico identifican los mismos clusters, pero tienen una disposición más lineal, lo que puede indicar limitaciones en la detección de formas más complejas.

DBSCAN tiene un tiempo de ejecución moderado y un Silhouette Coefficient mejor que los dos algoritmos anteriores. Sin embargo, el Calinski-Harabasz Index es más bajo. Genera sólo dos clusters, lo cual puede ser poco representativo si hay más grupos significativos en los datos.

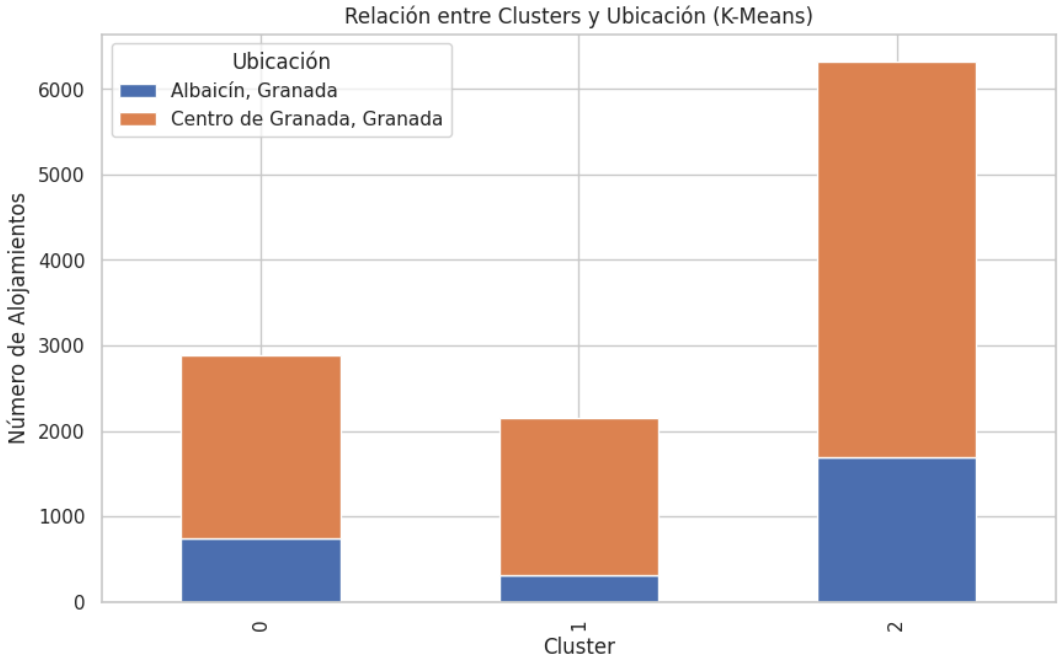
Gaussian Mixture da los mismos resultados que DBSCAN pero es bastante más rápido. En los gráficos vemos que Gaussian Mixture y DBSCAN también encuentran los mismos clusters pero tienen menos detalles o diversidad en los grupos.

HDBSCAN vemos que forma más clusters y que estos tienen formas irregulares lo cual puede capturar mejor relaciones más complejas.

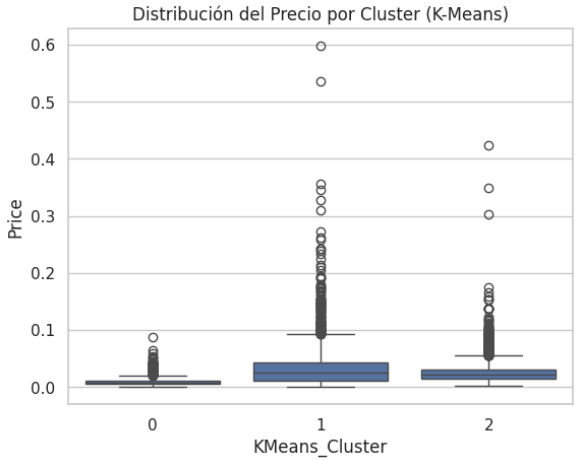
Interpretación de los resultados:
Una vez formados los clusters, hago algunos análisis adicionales.

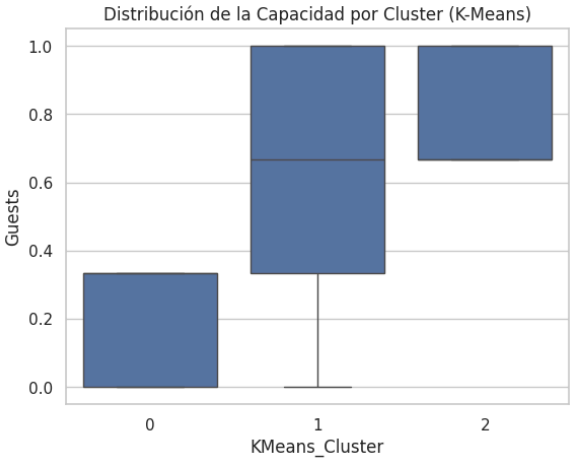
```
Distancia promedio al centroide por cluster (K-Means):
KMeans_Cluster
0                0.204923
1                0.450426
2                0.208793
```

Vemos que, en K-Means, el cluster 0 muestra una buena cohesión interna, los puntos están relativamente cerca del centroide. El cluster 1 muestra una distancia mayor al centroide, por lo que agrupa elementos menos homogéneos, hay mayor dispersión. El cluster 2 se parece bastante al cluster 0.

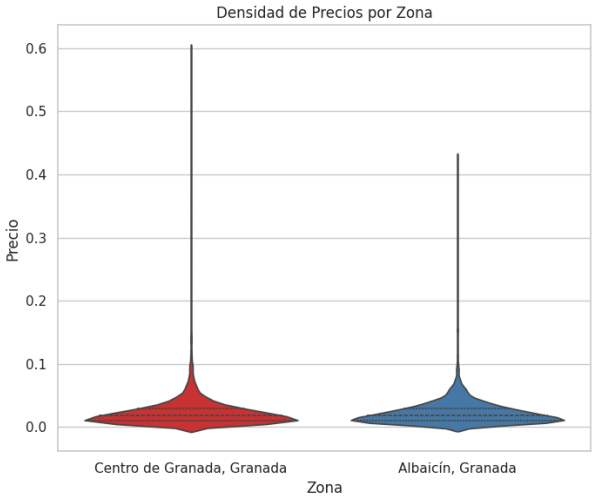


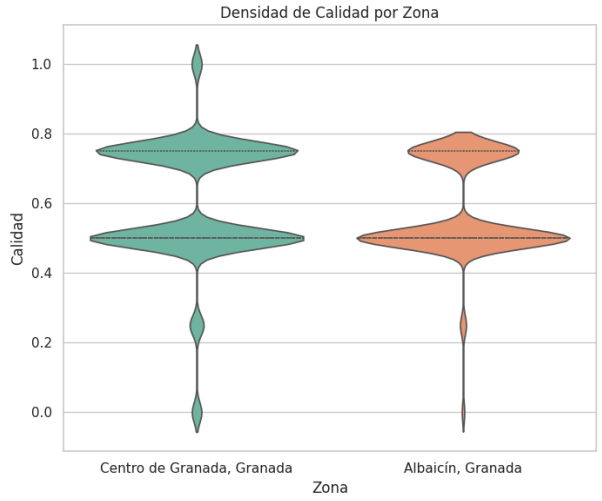
Los clusters 0 y 1 tienen aproximadamente un tercio de sus alojamientos en el Albaicín y dos tercios en el centro de Granada, lo que nos dice que la ubicación no es un gran determinante a la hora de agrupar los alojamientos en un cluster u otro. El cluster que mayor diferencia muestra es el 1, pues ahí menos de un 20% de los alojamientos están en el Albaicín. Aún así, no parece un determinante fuerte. El cluster 2 es el más grande, pues tienen unos 6000 alojamientos.



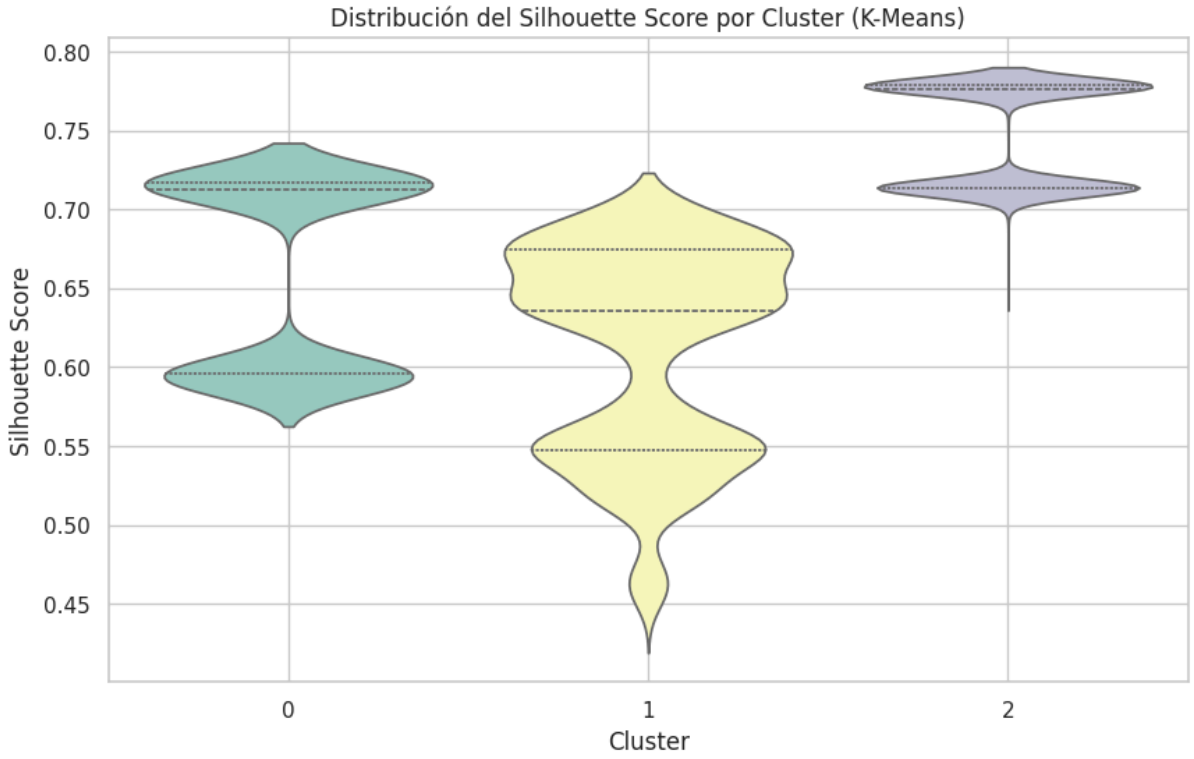


Estos boxplot muestran cómo varían los precios y la capacidad de los alojamientos dentro de cada cluster. En los clusters 0 y 2, los precios son mayoritariamente bajos, con poca variabilidad. El Cluster 1 tiene una mayor dispersión de precios, con valores atípicos más elevados. Esto sugiere que este cluster incluye alojamientos con una mayor variedad en precios. En cuanto a las capacidades de huéspedes, hay mayor diferencia entre clusters. El Cluster 1 tiene mayores capacidades, pero también mayor variabilidad. El Cluster 2 también tiene capacidades altas (mayoritariamente pensado para 8 huéspedes), pero con menor variabilidad que el Cluster 1. El Cluster 0 tiene capacidades significativamente más bajas, lo que sugiere que podría estar agrupando alojamientos pensados para parejas o familias pequeñas.

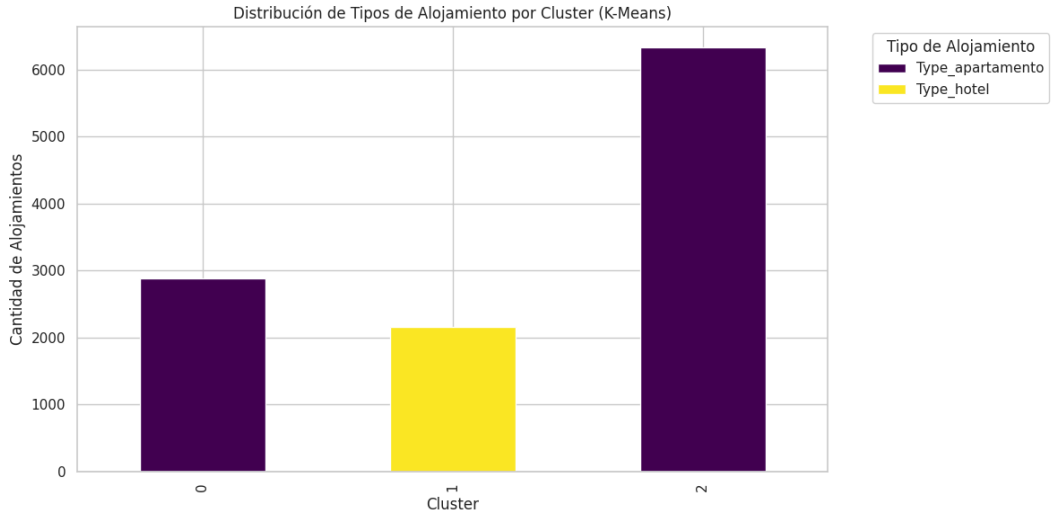


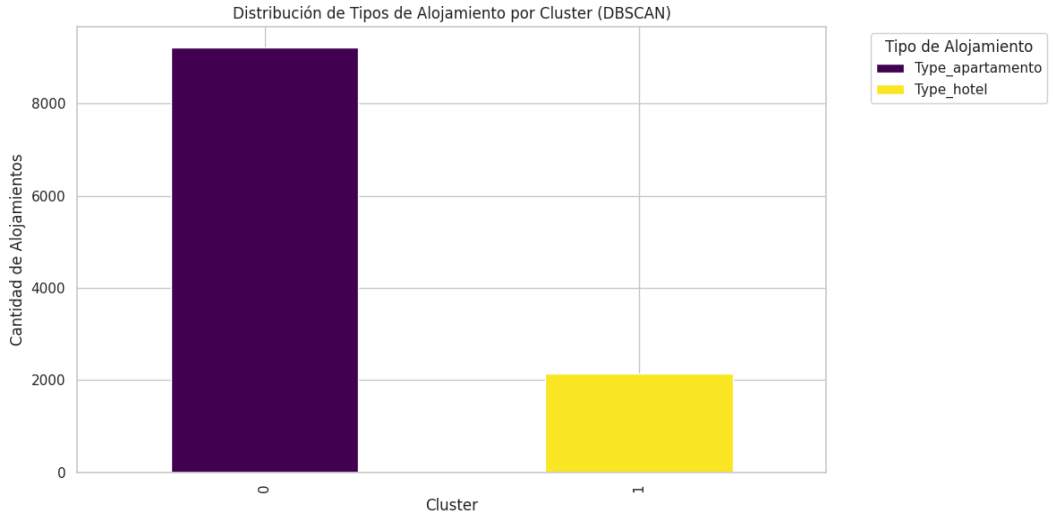


Con estos Violin Plots, vemos la distribución del precio medio de los alojamientos por zona, y la calidad por zona. En el Centro de Granada, los precios varían más, de manera que los alojamientos más costosos de esta zona son más caros en general que los más costosos del Albaicín. En cuanto a la calidad, vemos que en ambas zonas, la mayoría de los alojamientos se encuentran en el rango de [0.4, 0.6]. Sin embargo, en el Centro de Granada, hay mayor número de alojamientos con mejor calidad, y la calidad máxima es mayor que en la zona del Albaicín.



Este gráfico muestra cómo se distribuyen los valores del *silhouette score* en cada cluster generado con K-Means. Los clusters 0 y 2 tienen Scores relativamente altos, lo que sugiere que los puntos en estos clusters están bien agrupados entre sí, y alejados de otros clusters. El cluster 1 tiene una distribución más amplia y puntuaciones más bajas en general.





He decidido analizar también si el *Type* del alojamiento influye a la hora de crear los clusters. Lo he comprobado tanto para los clusters de K-Means, como para los de DBSCAN, pues sabemos que el algoritmo jerárquico y el Gaussian Mixture son muy similares en cuanto a clusters a estos dos respectivamente. Vemos que en ambos algoritmos, los clusters sólo contienen uno de los dos tipos de alojamiento que hay, o bien apartamentos o bien hoteles, por lo que los clusters vienen totalmente determinados por esta variable.

Interpretación de la segmentación:

En K-Means, podemos decir que el cluster 0 contiene apartamentos de precios muy asequibles, con capacidad para 2 o para 4 personas. Puede ser perfecto para parejas o familias pequeñas que buscan apartamentos donde tener mayor autonomía. Del cluster 1 podemos decir que contiene hoteles con un gran rango de precios y de capacidades para huéspedes. Esto es perfecto para clientes que tengan claro que buscan un hotel. Por último, en el cluster 2 tenemos apartamentos perfectos para un mayor número de huéspedes, aunque de precios un poco más elevados. Es perfecto para gente que viaja en grupo y tiene un presupuesto un poco

más elevado. En los tres clusters tenemos alojamientos repartidos por las dos zonas estudiadas de Granada, pero para clientes que buscan quedarse en el Albaicín, es más recomendable el cluster 2, y para el Centro de Granada el cluster 1.

Caso de Estudio 3:

En este caso de estudio, nos enfocamos en agrupar los alojamientos en función de su posición en el ranking de Booking.com, teniendo en cuenta atributos como el precio o la distancia al centro para ver si estos influyen en el ranking de esta plataforma. Sólo trabajaremos con los alojamientos cuyo *Rating* sea mayor que 5, es decir, alojamientos con relativa buena nota, dejando fuera los peores alojamientos. Además, imponemos la condición de que sólo se muestren los primeros 170 alojamientos del ranking. Esto lo hago para simular una búsqueda de clientes que sólo quieran quedarse en alojamientos relativamente bien valorados y de la mitad superior del ranking. También haremos estudios posteriores para ver si el *Rating* está directamente relacionado con el *Ranking*.

Para el preprocesado de datos, he procedido de la misma manera que en los casos anteriores. En una variable he guardado los datos seleccionando *Price, Distance, Ranking position, Type* y *Rating*, y en la otra he guardado las mismas sin el *Rating*, pues esta se utilizará para análisis posteriores, no para los algoritmos de clustering.

A las variables numéricas, le aplico la función de normalización que nos dan en el código de ejemplo, para que no influyan en el cálculo de los algoritmos, pues como sabemos, siempre es recomendable normalizar en clustering. Si los datos son demasiado grandes, me quedo con sólo un 10% de ellos. Utilizo OneHotEncoder, al igual que en el caso de estudio 2, para transformar los valores de la columna *Type* a columnas independientes numéricas.

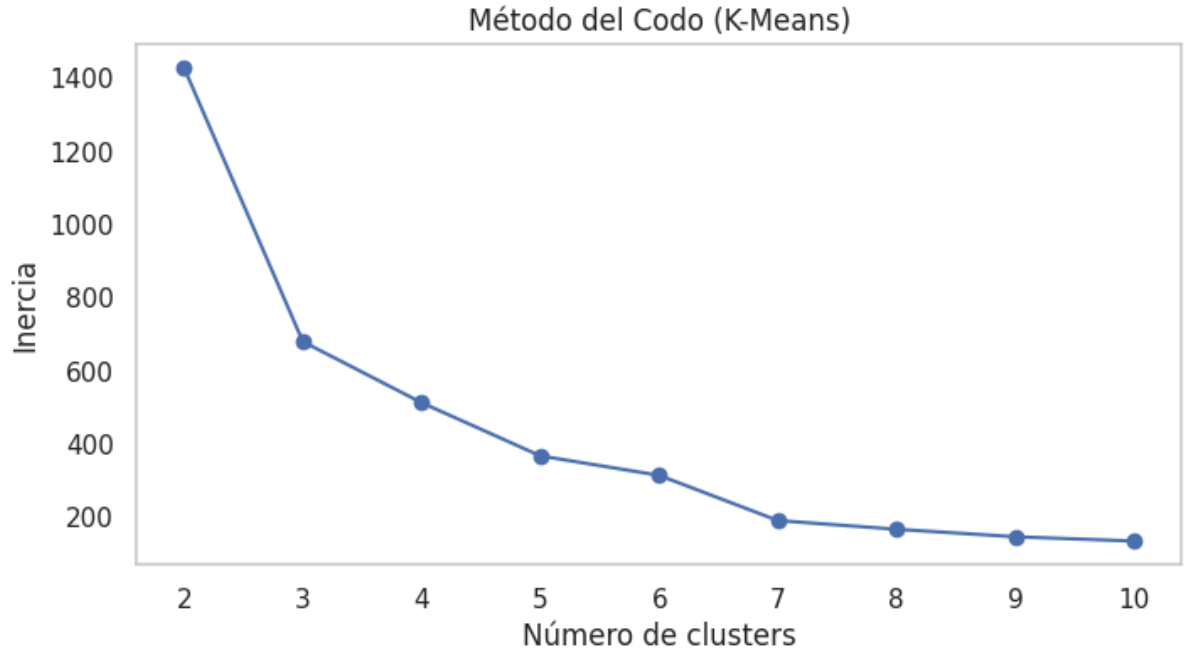
```
variables_interes = ['Price', 'Distance', 'Ranking position', 'Type',
'Rating']
data_clustering = resultados_busqueda[variables_interes]

data_clustering = data_clustering[data_clustering['Ranking
position'].between(1,170)]
data_clustering = data_clustering[data_clustering['Rating']>5]

data_subset = data_clustering[['Price', 'Distance', 'Ranking position',
'Type']]
```

En principio, este conjunto tiene 90583, pero tras eliminar los valores no nulos se me queda en 84054 instancias. Como es mayor que 10000, me acabo quedando con un 10% aleatorio, por lo que el conjunto de datos con el que finalmente trabajo tiene 8405 instancias.

Algoritmos utilizados:
Esta vez utilizo K-Means, Clustering jerárquico, OPTICS, Gaussian Mixture y Birch. He decidido hacer esto porque, a diferencia de DBSCAN, OPTICS puede detectar clusters de densidad variable y es menos sensible al parámetro eps, por lo que podría producir resultados

más estables. Como hemos escogido tantos atributos importantes en este caso de estudio, DBSCAN puede clasificar muchos puntos como ruido, y OPTICS evita clasificar puntos válidos como ruido [4]. Birch lo uso porque funciona bien también incluso cuando hay algo de ruido, y además suele ser más rápido que HDBSCAN [5].

K-Means:



Utilizo de nuevo el método del codo para elegir el número de clusters de K-Means, ya que en otros apartados me ha sido muy útil para tener una buena aproximación al problema. En este caso he elegido 3 clusters, pues es cuando se empieza a aplanar la curva y no hay tanta mejora al subir de clusters.

```
kmeans = KMeans(init='k-means++', n_clusters=3, n_init=5,
random_state=123456)
t = time.time()
kmeans_labels = kmeans.fit_predict(sample_data)
tiempo = time.time() - t
print(f"\nTiempo de K-Means: {tiempo} segundos")
```

Jerárquico:

He escogido el mismo número de clusters que para K-Means, pues vemos que puede hacer una representación buena de los grupos de datos.

```
hierarchical = AgglomerativeClustering(n_clusters=3)
t = time.time()
hierarchical_labels = hierarchical.fit_predict(sample_data)
tiempo = time.time() - t
```

```
print(f"Tiempo de Jerárquico: {tiempo} segundos")
```

OPTICS:

He probado con distintos valores de xi para ver qué resultados daban, cuántos clusters se obtenían… A pesar de que se obtienen mejores métricas con xi=0.2 que creaba 2 clusters, he decidido coger xi=0.18 que crea 3, para poder estudiar qué pasa cuando quitamos los datos que se interpretan como ruido, como veremos más adelante.

```
optics_model = OPTICS(min_samples=15, xi=0.18, min_cluster_size=15)
t = time.time()
optics_labels = optics_model.fit_predict(sample_data)
tiempo = time.time() - t
print(f"Tiempo de OPTICS: {tiempo} segundos")
```

Gaussian Mixture:

```
gmm = GaussianMixture(n_components=2, random_state=123456)
t = time.time()
gmm_labels = gmm.fit_predict(sample_data)
tiempo = time.time() - t
print(f"Tiempo de Gaussian Mixture: {tiempo} segundos")
```

Birch:

```
birch_model = Birch(n_clusters=2)
t = time.time()
birch_labels = birch_model.fit_predict(sample_data)
tiempo = time.time() - t
print(f"Tiempo de Birch: {tiempo} segundos")
```

Después reduzco dimensionalidad para poder evaluar y graficar los clusters.

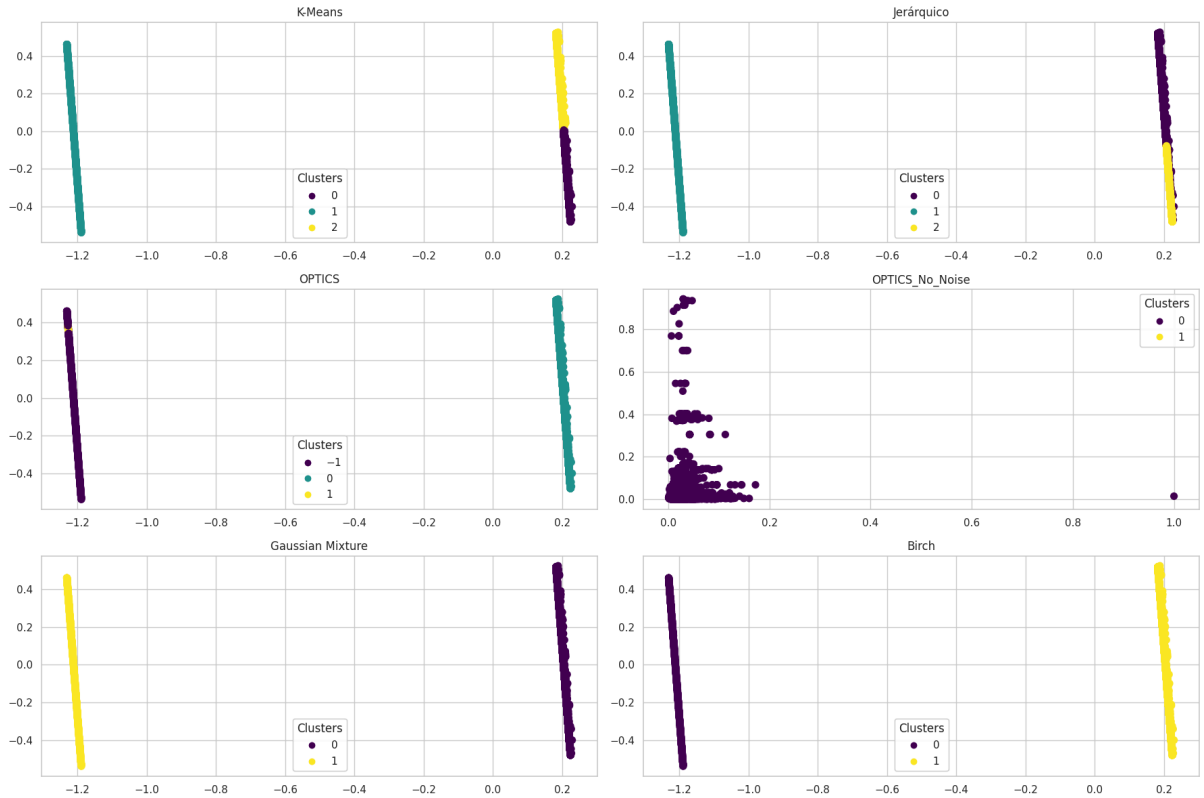
```
pca = PCA(n_components=2)
data_reduced = pca.fit_transform(sample_data)
```

Para OPTICS, he decidido evaluar el cluster en dos conjuntos distintos: en el conjunto normal y en el conjunto sin ruido. Esto es porque puede bajar el nivel de las métricas, pues el ruido significa que ese dato no pertenece a ningún cluster. Evaluarlo con ruido es un escenario más realista, como venimos evaluando hasta ahora, pues en este caso el ruido son simplemente hoteles o apartamentos muy diferentes a los demás, pero no son datos erróneos. Evaluarlo sin ruido es un estudio más focalizado en ver la calidad de los clusters que se han formado, sin que sean afectados por la negatividad que introduce el ruido. Analizando las diferencias entre

estas dos evaluaciones, podemos sacar conclusiones sobre si el conjunto de datos contiene mucho ruido o no, y sobre si la definición de clusters es robusta o no.

```
no_noise_mask = optics_labels != -1
X_no_noise = sample_data[no_noise_mask]
labels_no_noise = optics_labels[no_noise_mask]
evaluar_clusters(X_no_noise, labels_no_noise, "OPTICS_No_Noise")
evaluar_clusters(data_reduced, optics_labels, "OPTICS")
```

Algoritmo	Tiempo de ejecución	Silhouette Coefficient	Calinski-Harabasz Index	Número de clusters
K-Means	0.125s	0.645	41551.395	3
Clustering Jerárquico	5.805s	0.63	38395.195	3
OPTICS	13.474s	0.654	12412.826	3
OPTICS_No_ Noise		0.767	439.130	2
Gaussian Mixture	0.022s	0.773	24699.119	2
Birch	0.19s	0.773	24699.119	2



K-Means agrupa los datos en tres clusters que están bien definidos y separados en el espacio, indicando que son grupos compactos, lo cual coincide con sus métricas altas.

El clustering jerárquico forma también tres clusters similares a los de K-Means.

OPTICS detecta tres clusters en total: dos principales y un grupo de ruido.

OPTICS_No_Noise detecta sólo 2 clusters pues se han eliminado los puntos que se habían detectado como ruido. Visualmente parece que los clusters están menos claros. Esto concuerda con los valores obtenidos en las métricas, pues el Silhouette Coefficient es superior, indicando que los puntos cuadran mejor en el cluster que se les ha asignado (tiene sentido porque los puntos que menos cuadraban eran el ruido que se ha eliminado), pero el Calinski-Harabasz Index ha disminuido lo cual indica que el cluster es menos compacto, los puntos están más dispersos en él [6].

Gaussian Mixture identifica 2 clusters, que aparentemente, están muy bien separados y definidos. Por sus métricas, podemos intuir que también son muy compactos.

Birch identifica 2 clusters principales, equivalentes a los de Gaussian Mixture.

Interpretación de los resultados:

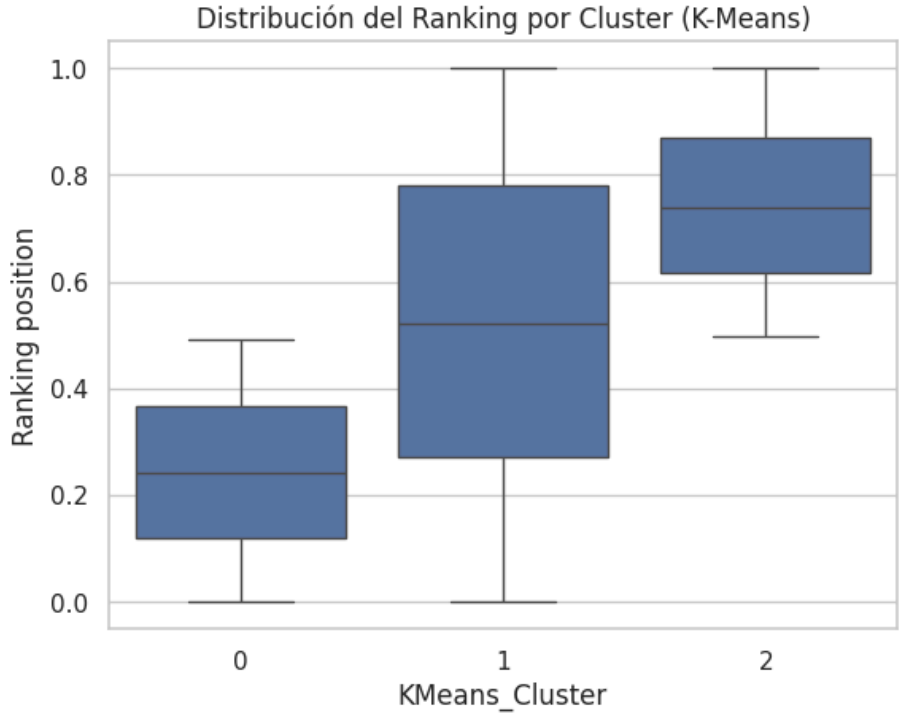
```
Distancia promedio al centroide por cluster (K-Means):
KMeans_Cluster
0    0.133295
1    0.259167
2    0.142540
```

Vemos que el cluster 1 es el que menos compacto está, mientras que los otros dos son bastante similares en cuanto a dispersión de los datos dentro del cluster.

```
Promedio de variables por cluster:
                   Price    Rating  Distance  Ranking position
KMeans_Cluster
0               0.022494  0.754454  0.023015          0.242855
1               0.026300  0.684078  0.013788          0.521608
2               0.023997  0.751944  0.028981          0.743995
```

En los 3 clusters el precio promedio es similar, indicando que esta no es una variable muy determinante en la separación de los datos, aunque el cluster 1 destaca por tener alojamientos un poco más caros que los otros dos. En cuanto a *Rating*, podemos decir que el que destaca es de nuevo el 1, por tener alojamientos peor evaluados que los otros dos. En distancia al centro, el cluster 1 tiene los alojamientos más céntricos, seguido del 0 y después el 2. El cluster 2

tiene los alojamientos mejor posicionados en el ranking, en general, seguido del 1 y terminando con el 0, que tiene los peores situados en el ranking dentro de nuestros datos. Me parecía interesante seguir estudiando la posición del ranking según cada cluster.



De este boxplot podemos sacar como conclusión que el cluster 1 contiene alojamientos que están a mitad del ranking, pero contiene alojamientos más dispersos, con algunos que sí que están situados arriba del ranking y otros bastante más abajo. Mientras tanto los otros dos clusters son más compactos y sólo tienen alojamientos en una zona determinada y concreta del ranking.

En este gráfico de Violin del Silhouette Score de K-Means, vemos que el cluster 1 tiene una distribución concentrada en los valores altos, indicando un grupo muy compacto y bien definido. Los clusters 0 y 2 muestran distribuciones casi idénticas, con scores que van desde muy bajos hasta más de 0.8, lo que muestra bastante dispersión y que algunos puntos están menos cohesionados dentro de sus clusters.

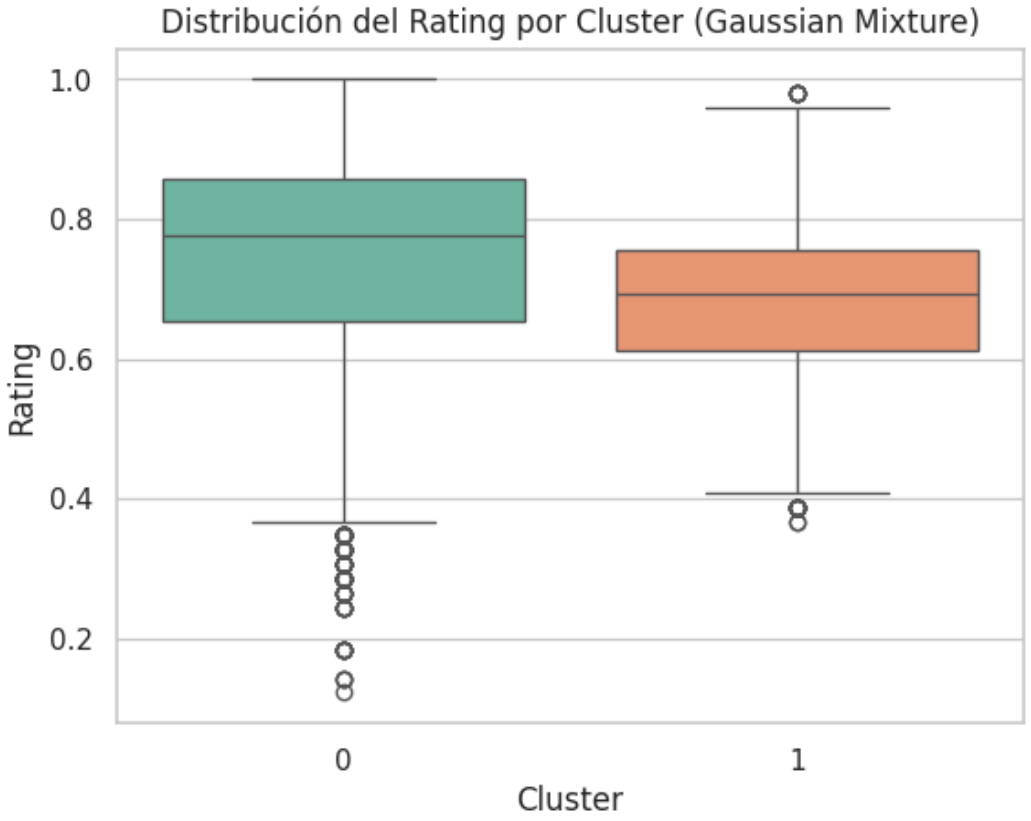
Analizamos ahora los promedios de los clusters que crean el método OPTICS y el método Gaussian Mixture, para poder sacar conclusiones y compararlas con las obtenidas con el método de K-Means.

```
Promedios OPTICS:
                    Price    Rating  Distance  Ranking position
OPTICS_Cluster
-1               0.026501  0.684220  0.013895          0.515754
 0               0.023214  0.753252  0.025871          0.482800
 1               0.013007  0.674603  0.006671          0.909928
```

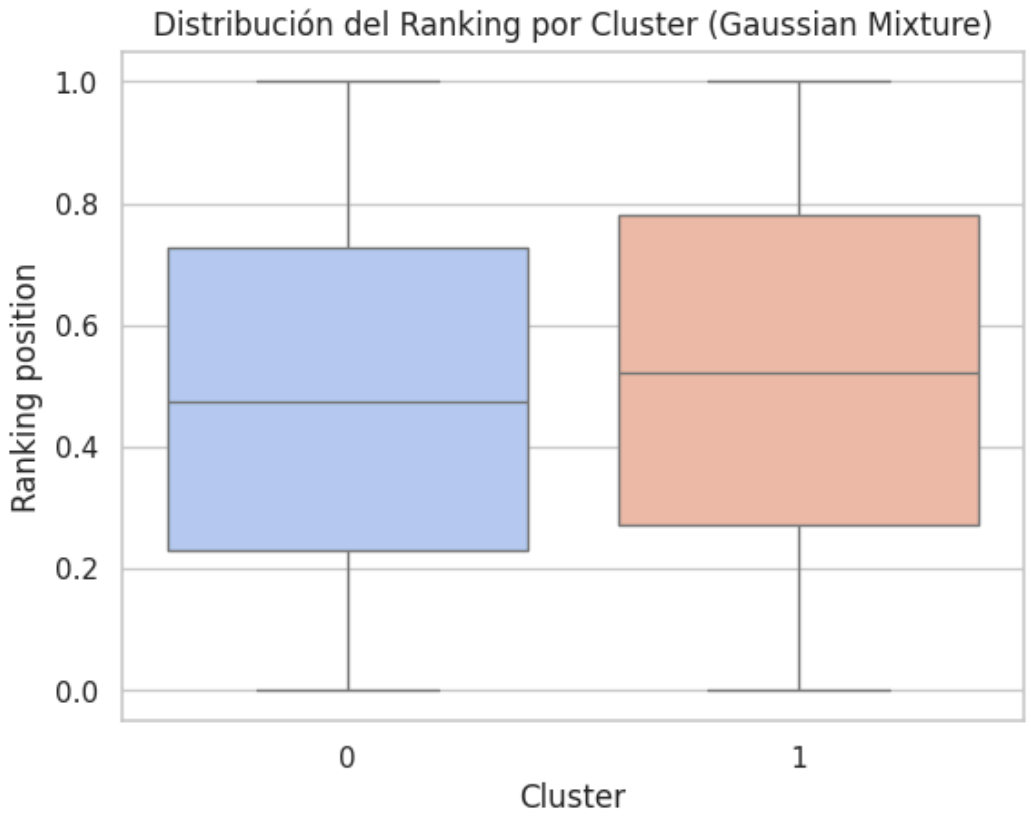
Con el método OPTICS, el cluster -1 se supone que contiene los puntos considerados como ruido, los que no encajan bien en ningún cluster. Estos resultan ser alojamientos más caros en general, y con valores de *Rating, Distance* y *Ranking position* intermedios. El cluster 0 tiene el mejor *Rating* pero los alojamientos más lejanos al centro. En el cluster 1 ocurre justo al contrario, tenemos alojamientos peor puntuados pero muy económicos y céntricos. El mejor ranking es el del cluster 1, demostrando que, a pesar de lo que personalmente creía, no es el rating lo que más influye en el ranking.

```
Promedios Gaussian Mixture:
                            Rating  Ranking position     Price  Distance
GaussianMixture_Cluster
0                         0.753252          0.482800  0.023214  0.025871
1                         0.684078          0.521608  0.026300  0.013788
```

Con el método Gaussian Mixture, se crean sólo 2 clusters. El 0 tiene el mejor rating pero la mayor distancia al centro, siendo alojamientos más alejados pero bien puntuados. El cluster 1 tiene peores ratings pero mayor centricidad, además de alojamientos más caros.

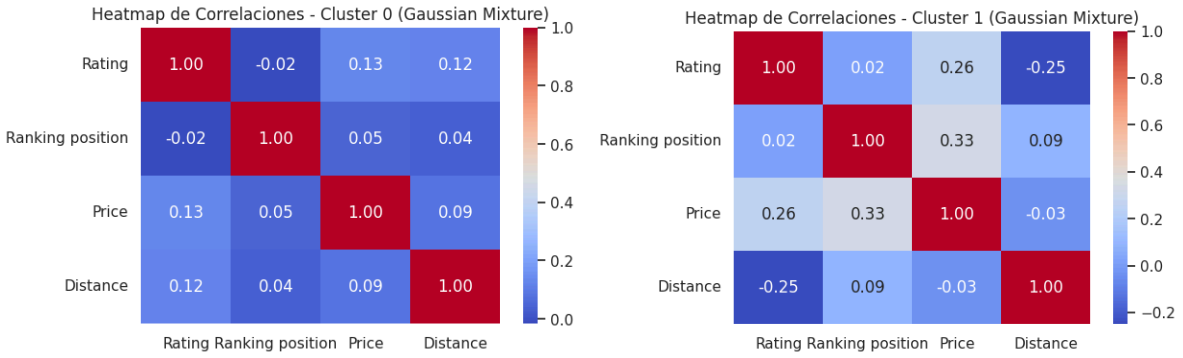


Este gráfico nos ayuda a ver que, aunque en el cluster 0 teníamos un rating promedio más alto, en realidad este cluster tiene alojamientos más variados, algunos con ratings muy altos, pero otros con una nota más baja que las peores notas del cluster 1, que es más compacto. Por lo tanto, si el objetivo fuese encontrar alojamientos de mejor rating, escogeríamos el cluster 0, pero si fuese excluir los alojamientos con rating menor que 0.4 sobre 1, nos iríamos al cluster 1, que es más restrictivo en cuanto a estos alojamientos con ratings bajos.. Recordamos que en este caso de estudio, sólo hemos escogido los alojamientos con más de un 5 en rating, por lo que 0.4/1 supone alojamientos notables, de más de un 7/10.

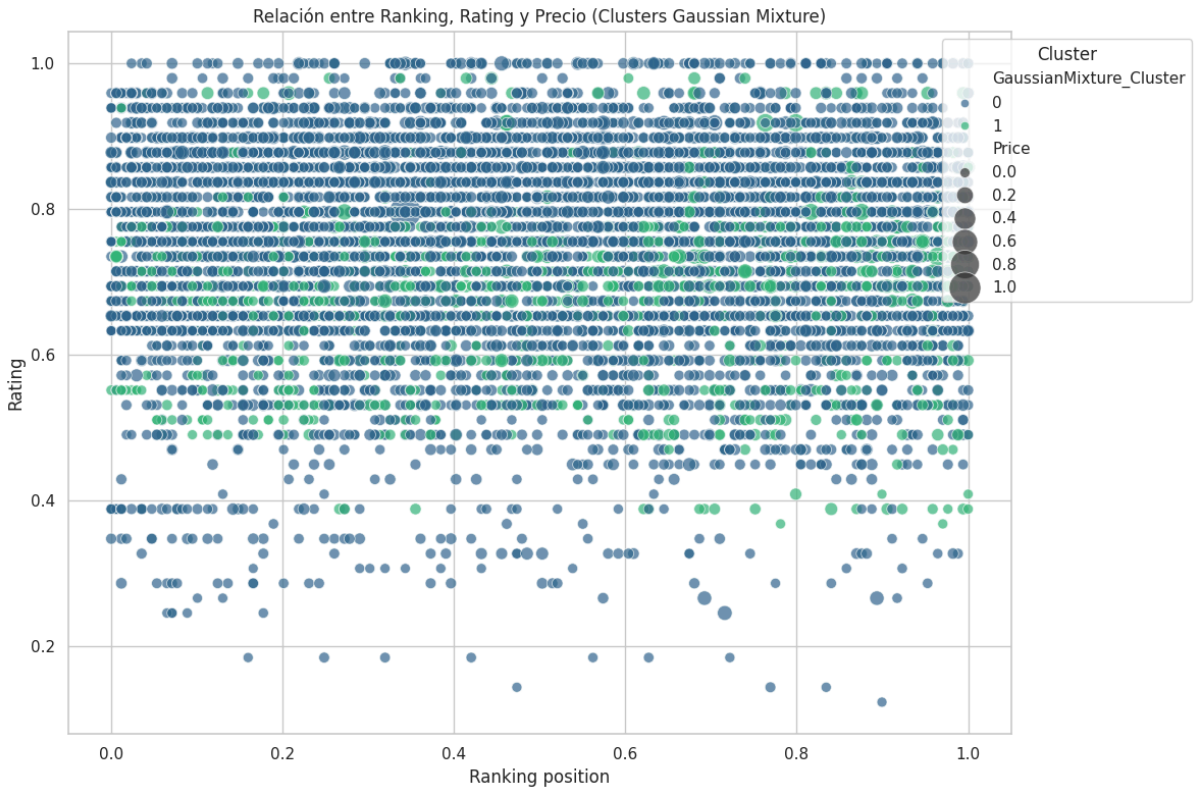


Analizando este boxplot, vemos que el ranking de alojamientos en cada cluster está muy igualado. No podemos sacar conclusiones demasiado útiles de este atributo.

Analizo ahora la correlación de variables en cada cluster del Gaussian Mixture con un heatmap:



Concluimos que, en el cluster 0, tenemos alojamientos donde el ranking y el rating no van de la mano, pues tienen una ligera correlación negativa. En el cluster 1, tenemos alojamientos donde la distancia al centro y el precio influyen mucho en el rating que tienen. Asimismo, el precio es un gran influyente en la colocación del alojamiento en el Ranking. Estos datos concuerdan con los que obtuvimos al analizar la tabla de promedios.



Con este gráfico de burbujas del algoritmo Gaussian Mixture, podemos analizar la relación entre *Ranking position* y *Rating* en los dos clusters que se crean usando distintos colores para cada uno. Además, podemos relacionarlo también con el precio del alojamiento, con el tamaño de las burbujas.

En el cluster 1, el verde, los datos se distribuyen ampliamente en términos de Ranking y Rating. Hay una mayor densidad de puntos con valores altos de Rating, pero con posiciones de ranking distribuidas de forma diversa. El cluster 0, el azul, tiene valores más variados para ambas variables. Los puntos más grandes, que representan precios altos, están distribuidos entre ambos clusters. Sin embargo, no parecen seguir un patrón claro ni estar significativamente correlacionados con el ranking o el rating.

Interpretación de la segmentación:

De lo que hemos analizado en este caso de estudio, en K-Means podemos decir que el cluster 0 contiene alojamientos muy económicos y relativamente céntricos, con muy buena nota, aunque un poco más abajo en el ranking. Esto es perfecto para turistas que valoren la opinión de otros clientes por encima del sistema de valoración de Booking, y que quieran quedarse relativamente cerca del centro sin gastar demasiado. El cluster 1 tiene alojamientos más lujosos, peor puntuados pero muy céntricos y mejor colocados en el ranking. Esto es perfecto para clientes cuya prioridad sea quedarse en el centro de la ciudad, aunque suponga invertir un poco más en la estancia. Por último, el cluster 2 tiene alojamientos de precio medio, más

periféricos, pero muy bien situados en el ranking. Esto es perfecto para clientes que no quieran estar comparando la calidad de los alojamientos, sino que prefieren fiarse del ranking de Booking directamente, pero no quieren gastar demasiado dinero en alojamientos de calidad.

Contenido Adicional:

```
#Función para evaluar los clusters
def evaluar_clusters(data, labels, nombre_algoritmo):
   if len(set(labels)) > 1:  # Sólo si hay más de un cluster
         sample_size = min(1000, len(data))   # Máx 1000 muestras si el
conjunto es grande
       try:
                    metric_SC  =  metrics.silhouette_score(data,  labels,
metric='euclidean', sample_size=sample_size, random_state=123456)
       except ValueError:
             metric_SC = None  # Si no se puede calcular el silhouette
score
       metric_CH = metrics.calinski_harabasz_score(data, labels)

       # Imprimir resultados
       print(f"{nombre_algoritmo}:")
       if metric_SC is not None:
           print("Silhouette Coefficient: {:.5f}".format(metric_SC))
       else:
           print("Silhouette Coefficient: No calculable")
       print("Calinski-Harabasz Index: {:.3f}".format(metric_CH))
   else:
        print(f"El algoritmo {nombre_algoritmo} no pudo generar clusters
válidos.")
```

He creado esta función para utilizarla con todos los casos de estudio. Se encarga de evaluar los clusters y mostrar los valores que obtienen para el Silhouette Score y el Calinski-Harabasz Score ya explicados.

```
#Función para visualizar los clusters
def graficar_clusters(data, labels, title, ax=None):
 if ax is None:
   plt.figure(figsize=(8, 6))
   ax = plt.gca()
 scatter = ax.scatter(data[:, 0], data[:, 1], c=labels, cmap='viridis',
s=50)
 ax.set_title(title)
 legend1 = ax.legend(*scatter.legend_elements(), title="Clusters")
 ax.add_artist(legend1)
```

También he creado esta función para mostrar los clusters visualmente y que sea más sencilla su interpretación. Los muestro como subgráficos dentro de un gráfico más grande, para que estén agrupados y sea más fácil visualizarlos, ya que si estaba uno debajo del otro me costaba más compararlos.

Bibliografía:

[1]. Uribe, L. G. (2021, 15 diciembre). *Aprendizaje no supervisado, clustering K-Means, para todos. . .* Medium. https://novagenio.medium.com/aprendizaje-no-supervisado-clustering-para-todos-con-python-2cc2bcaafae9

[2]. Jarroba, R. [. (2017, 2 junio). Selección del número óptimo de Clusters - Jarroba. Jarroba. https://jarroba.com/seleccion-del-numero-optimo-clusters/

[3]. *¿Cuáles son algunas métricas alternativas para evaluar la agrupación de K-means además de la suma de errores al cuadrado?* (2023, 13 abril). www.linkedin.com. https://es.linkedin.com/advice/0/what-some-alternative-metrics-evaluate-k-means?lang=es#:~:text=El%20%C3%ADndice%20de%20Calinski%2DHarabasz,y%20el%20n%C3%BAmero%20de%20puntos.

[4]. J, E. (2023, 27 septiembre). *OPTICS Clustering: From Novice to Expert in Simple Steps*. Datarundown. https://datarundown.com/optics-clustering/

[5]. *¿Cómo se compara el rendimiento y la escalabilidad de HDBSCAN y OPTICS para agrupar grandes conjuntos de datos?* (2023, 10 abril). www.linkedin.com. https://es.linkedin.com/advice/1/how-do-you-compare-performance-scalability-hdbscan?lang=es

[6]. *Machine Learning Evaluation Mastery: How to Use Silhouette Score and Calinski-Harabasz Index for Clustering Problems*. (2024, 25 febrero). GPTutorPro. https://gpttutorpro.com/machine-learning-evaluation-mastery-how-to-use-silhouette-score-and-calinski-harabasz-index-for-clustering-problems/