

Aprendizaje No Supervisado – Memoria

Aprendizaje Automático I

ELENA CAMPANERO BELDA

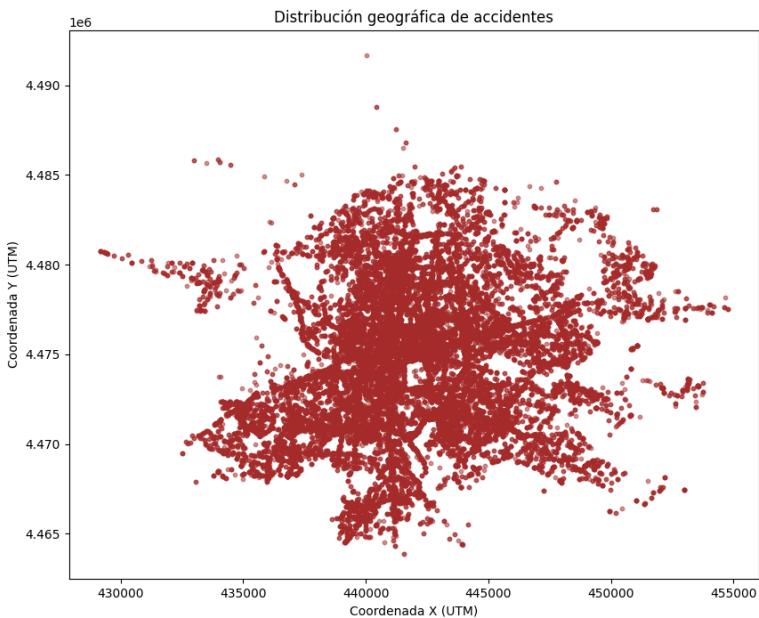
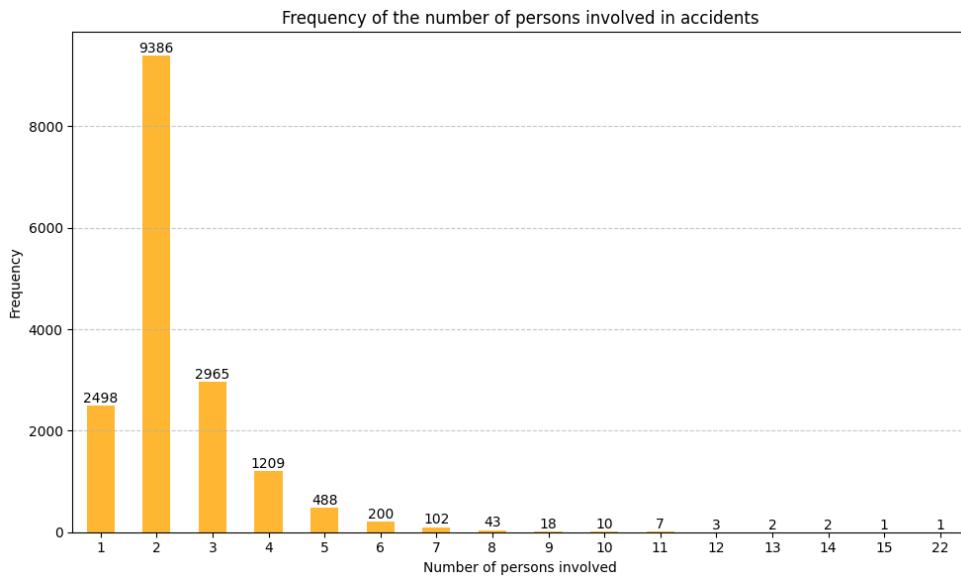
NATALIA KLINIK

1. Contexto.

Tenemos proporcionado un conjunto de datos que contiene información sobre los accidentes de ciudad de Madrid. El objetivo de esta práctica es extraer información sobre las condiciones en las que se producen accidentes en Madrid, tales como el efecto de la lluvia, determinadas localizaciones o determinados tipos de vehículos.

2. Descripción del dataset.

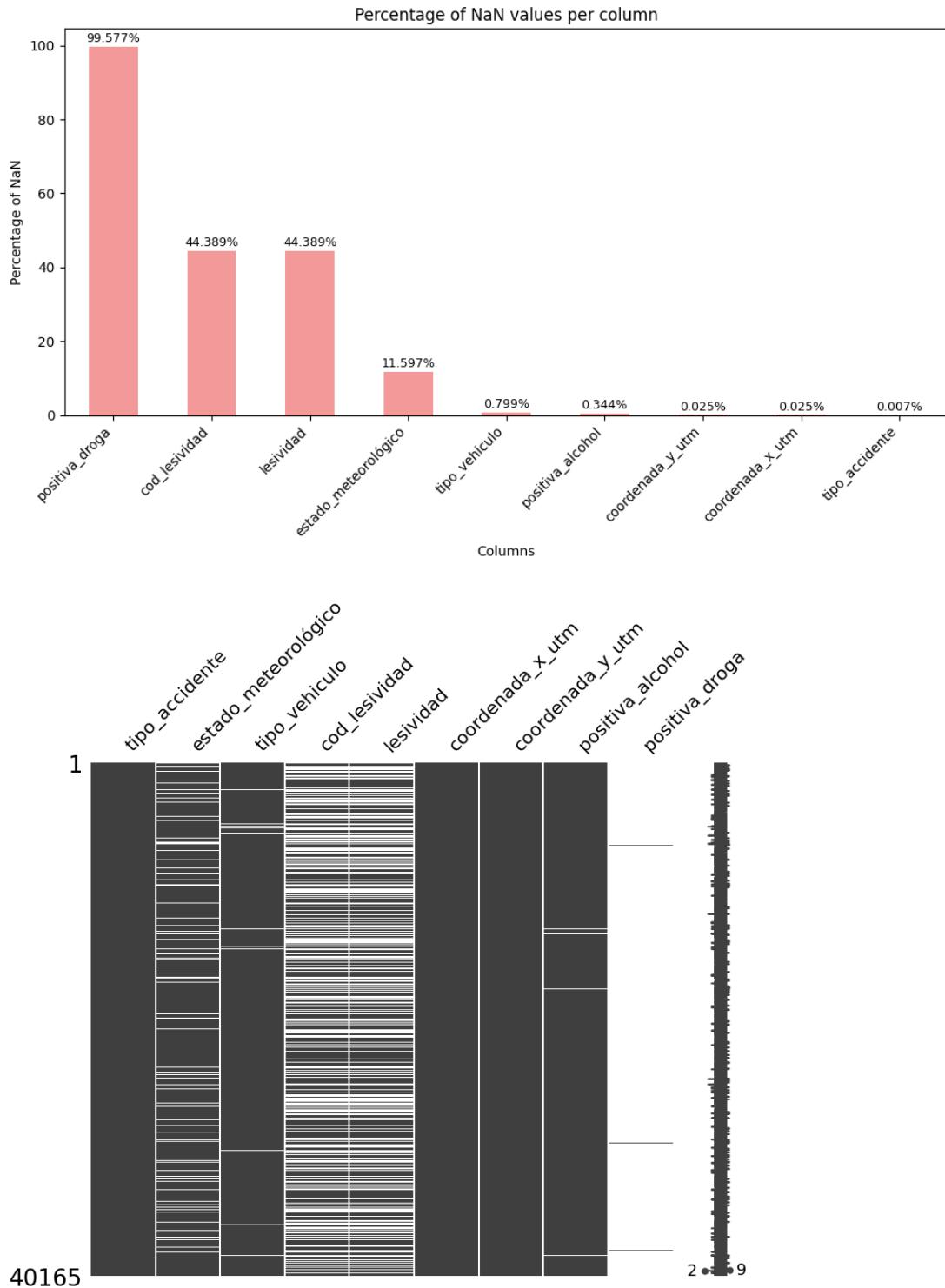
Hemos empezado con hacer un análisis descriptivo de nuestro dataset. Primero, hemos comprobado los tipos de datos que tenemos. Luego hemos visualizado la distribución de cada variable, obteniendo las gráficas, que presentamos debajo y mas tarde en la memoria.



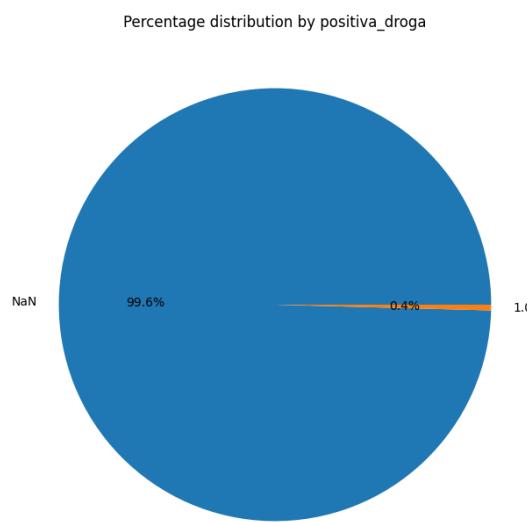
3. Preprocesamiento del dataset.

3.1. Valores nulos.

Mirando las distribuciones se puede ver que muchas columnas contienen valores nulos. Hemos visualizado la distribución de estas variables para saber qué hacer con esos.

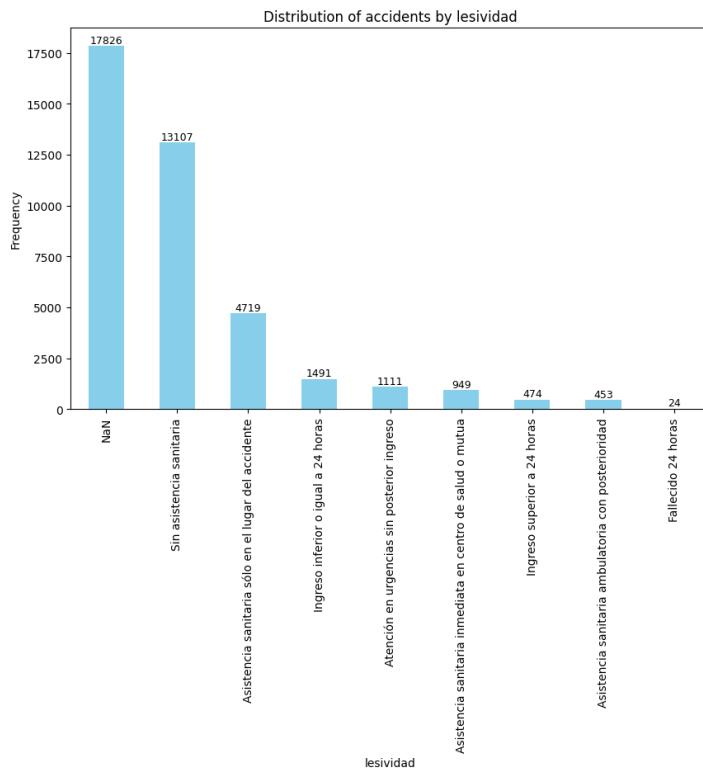


3.1.1. Columna positiva_droga.



Podemos ver que en caso de esta columna casi todos los valores son nulos y solo disponemos de 170 filas con valores positivos. Pensamos que llenar los NaNs con otro valor, como N para droga negativa, no tendrá sentido, ya que la columna estaba totalmente artificial. En ese caso decidimos a eliminar la columna particular.

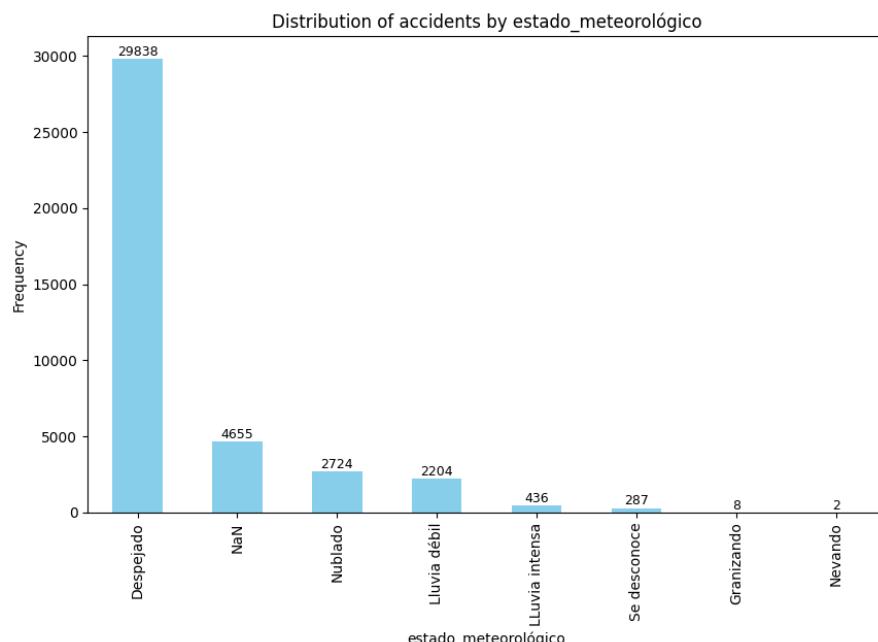
3.1.2. Columnas lesividad y cod_lesividad.



Lo primero que hemos visto es que esas columnas importan exactamente la misma información. La única diferencia es que una contiene de información codificada y la otra no. Decidimos a eliminar la columna lesividad y usar solo cod_lesividad.

En caso de los NaNs, decidimos a llenar los valores vacíos con la nueva categoría, nombrada “Sin especificar”. Pensamos que eso es la mejor opción, ya que una imputación de 40% de los datos con otro 60% no tiene mucho sentido. Tampoco tiene eliminar las filas con valores vacíos o cada columna.

3.1.3. Columna estado_meteorologico.



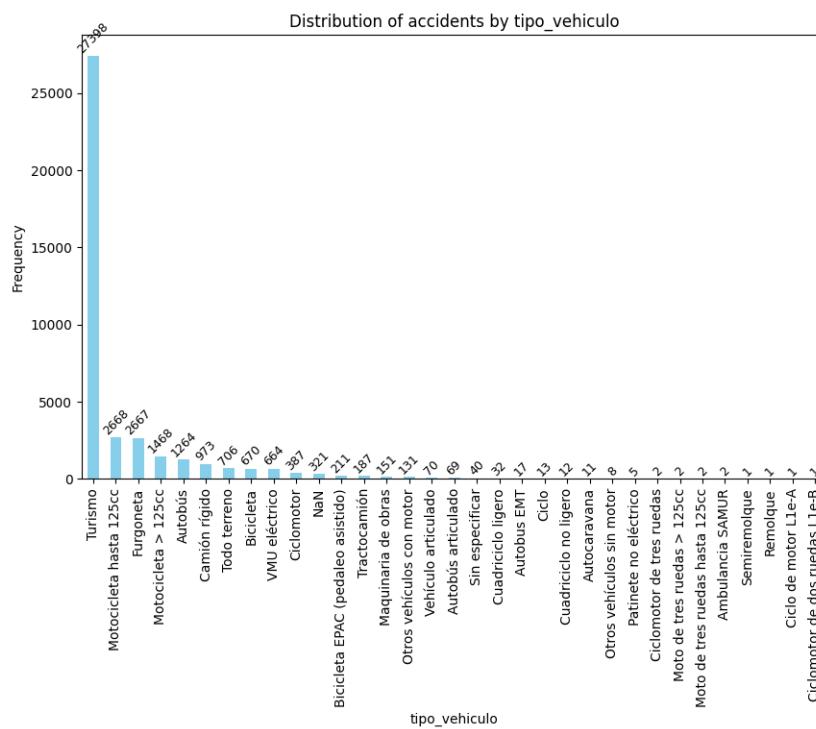
En ese caso no solo tenemos NaNs, pero también una categoría “Se desconoce”. Gracias a eso podemos imputar los valores vacíos directamente.

Sin embargo, en primer lugar, hemos intentado a buscar si en caso cuando para un accidente tenemos mas que una fila, podemos llenar un valor vacío con valor de otra fila. Desgraciadamente, no había sido ninguna situación así.

Luego hemos probado una imputación basada en el estado meteorológico de otros accidentes el mismo día, a la misma hora y en una zona cercana. No hemos avanzado mucho, como solo se han imputado 321 de las filas y otras 4334 todavía estaban vacías.

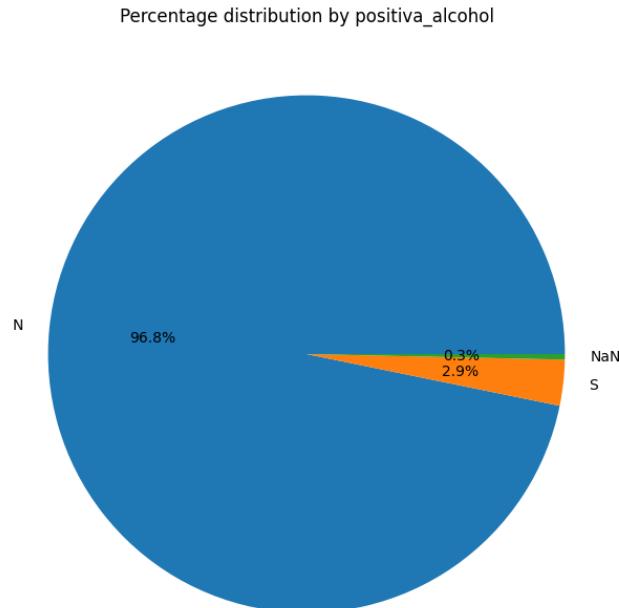
Al final, llenamos los valores restantes, como hemos dicho antes – codificando NaN en la categoría “Se desconoce”.

3.1.4. Columna tipo_vehiculo.



Como existe una categoría “Sin especificar”, podemos cambiar los NaNs directamente a esa.

3.1.5. Columna positiva_alcohol.



En muchos casos hay mas que un coche involvido en el mismo accidente, decidimos a comprobar si las filas con los valores vacíos tienen los mismos num_expediente. Parece que tenemos el mismo número de datos vacíos ya que de num_expediente. En ese caso no podemos eliminarlas, como aportan información relevante.

Luego hemos pensado de comprobar si la persona con valor vacío de positiva_alcohol es siempre un conductor de coche. Si la persona era un niño o el pasajero, el uso de alcohol no influye los datos. Desgraciadamente, todos los valores vacíos están asignados a conductores de los choches.

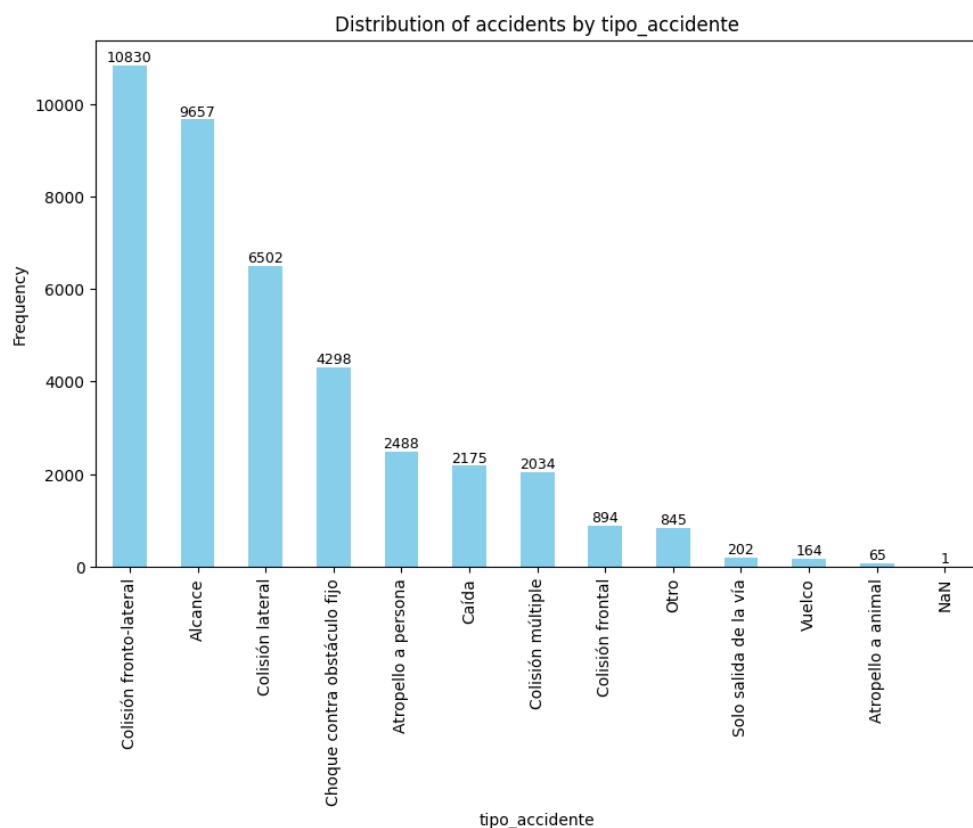
Al final decidimos tratar los valores nulos como la influencia de alcohol negativa ('N'), teniendo en cuenta que esta imputación puede resultar en la mal clasificación de los datos.

3.1.6. Columnas coordenada_x_utm y coordenada_y_utm.

El dataset principal contiene de las columnas localización y numero. Hemos visto que esas dos aportan exactamente la misma información que las coordenadas. En caso de las coordenadas, la distancia esta preservada, entonces decidimos a eliminar columnas dichas antes.

El porcentaje de los valores vacíos es mismo en ambos casos y es muy bajo. Sin embargo, tenemos que comprobar si cuando falta coordenada x, siempre falta coordenada y, porque eso facilitaba el proceso. Eso resultó a ser verdadero, entonces, como solo faltan 10 valores, decidimos a eliminar las filas que los contienen.

3.1.7. Columna tipo_accidente.

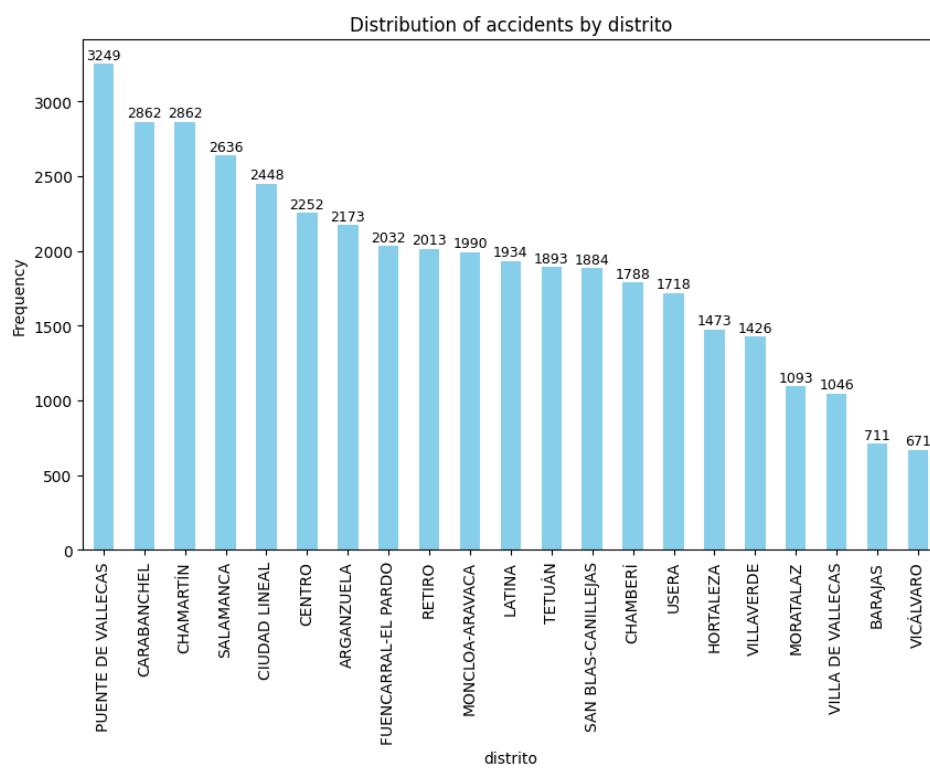


Como es solo una fila, comprobamos si existe otro implicado en el accidente para tratarlo y evitar perder información. Resulto que en esa fila tenemos también los valores desconocidos en otras columnas y no aporta mucha información. Decidimos eliminarla.

3.2. Datos redundantes.

Hemos dicho antes, que lesividad y cod_lesividad importan la misma información, y que con localización y las coordenadas la situación es similar. Lo mismo pasa para las columnas distrito y cod_distrito.

3.2.1. Columnas distrito y cod_distrito.

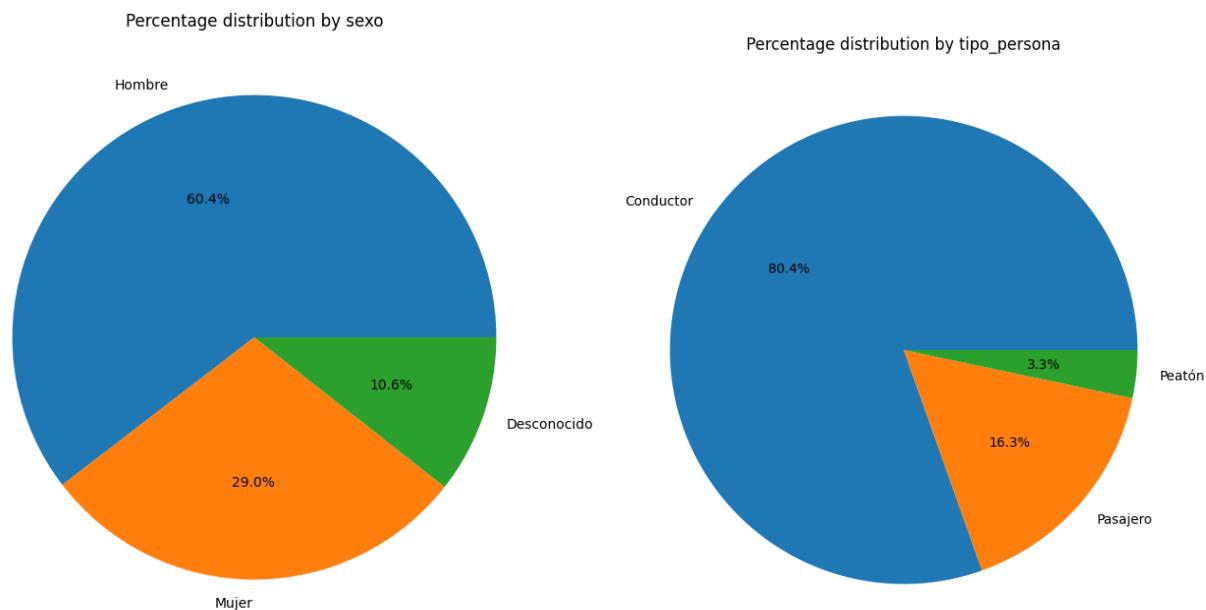


Ambas variables tienen la misma distribución. Entonces como cod_distrito ya está codificado, decidimos a usar esa y eliminar la otra.

3.3. Transformación de los datos.

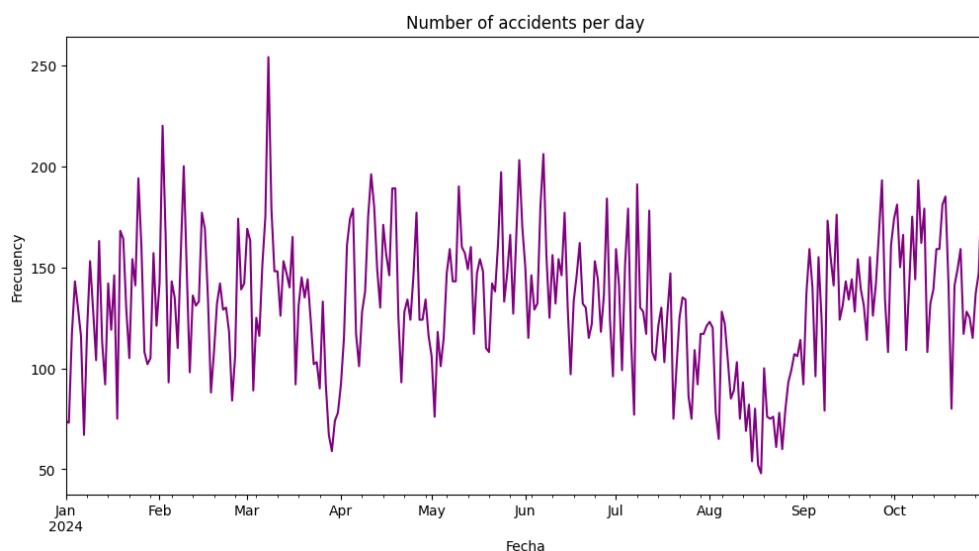
3.3.1. Variables categóricas.

Presentamos la distribución de los variables categóricas, que nos quedan.



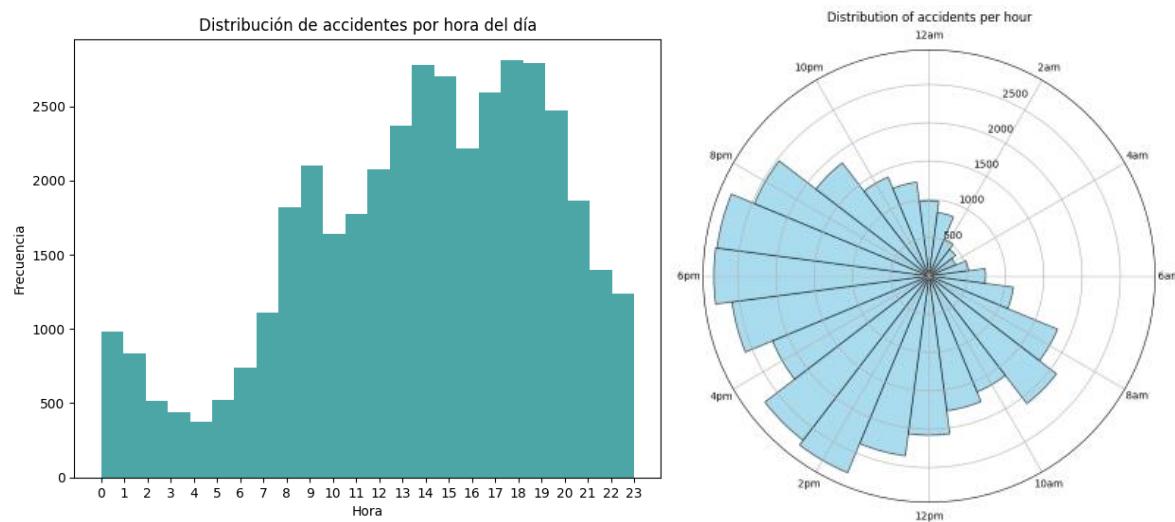
En caso de variables categóricas, hemos usado el Label Encoder (para los variables con mas de dos categorías) y el One-Hot Encoding (para los variables con solo dos categorías), para transformarles a los valores numéricos.

3.3.2. Columna fecha.



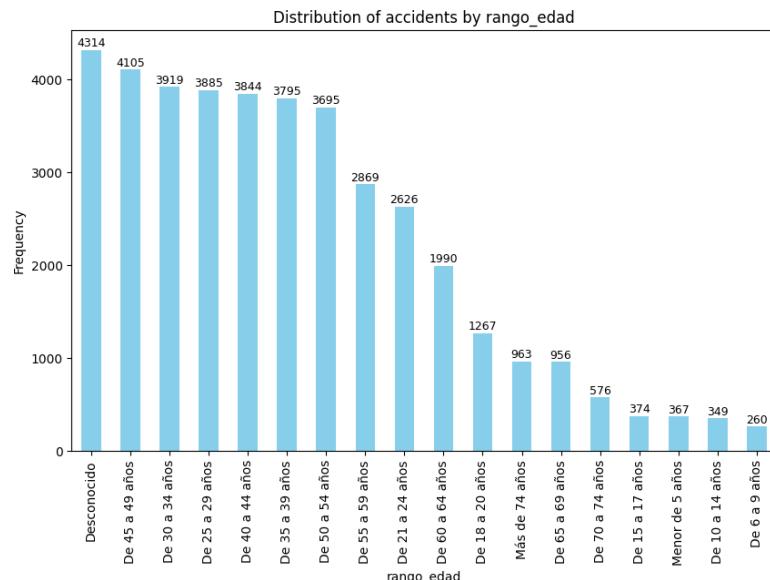
Hemos transformado esa variable a tipo Datetime y luego visualizamos la distribución de los accidentes por día. También hemos extraído el mes y el día de nuestra variable y como no tenía sentido usar todas las columnas, hemos eliminado la fecha.

3.3.3. Columna hora.



Hemos realizado los mismos pasos. Hemos convertido la columna en tipo Datetime y hemos extraído los valores de las horas y los minutos. Luego hemos eliminado la columna hora. También hemos visualizado la distribución de los accidentes por la hora.

3.3.4. Columna rango_edad.



En caso de columna rango_edad, hemos hecho un diccionario, para codificar la columna manualmente. Gracias a esto tendremos los datos más comprensibles y con más sentido a las distancias entre los valores.

3.4. Agrupación de los datos.

Como sabemos, en nuestro dataframe, existen más de una fila para un mismo expediente y por ello debemos de analizar las posibles soluciones:

3.4.1. Mantener todas las filas.

Eso nos puede servir cuando queremos identificar patrones a nivel individual (persona), como el perfil de las personas involucradas en accidentes. Una desventaja es que los accidentes con más personas involucradas no se analizan en conjunto, lo que puede sesgar los resultados del clustering.

3.4.2. Agrupar las filas por expediente.

Eso nos puede servir para analizar los accidentes en conjunto y no individualmente. Una desventaja de esta solución es que las columnas resultantes de la fusión puede no reflejar la realidad de los datos originales ya que nosotros la creamos en base a una regla que nosotros definimos.

3.4.3. Decisión final.

Como el objetivo es caracterizar accidentes según la ubicación, condiciones meteorológicas, tipo de accidente, etc., decidimos agrupar las filas por expediente.

Hemos buscado los variables que siguen mismos entre el mismo tipo_expediente, como cod_distrito, coordenadas, fecha, hora, estado_meteo, tipo_accidente. Las variables que no coinciden son tipo_persona, sexo, rango_edad, lesividad, vehiculo, positivo_alcohol.

Para agrupar los datos correctamente, hemos tomado siguientes pasos.

3.4.3.1. Columna rango_edad.

Reducimos los rangos y creamos una columna para cada rango que reflejará la cantidad de personas que hay de cada rango.

3.4.3.2. Columna tipo_persona.

Haremos una columna para conductor, pasajero y peatón que reflejará la cantidad de personas que hay de cada tipo.

3.4.3.3. Columna sexo.

Haremos una columna para hombre, mujer y otra para desconocido que reflejará la cantidad de personas que hay de cada sexo.

3.4.3.4. Columna lesividad.

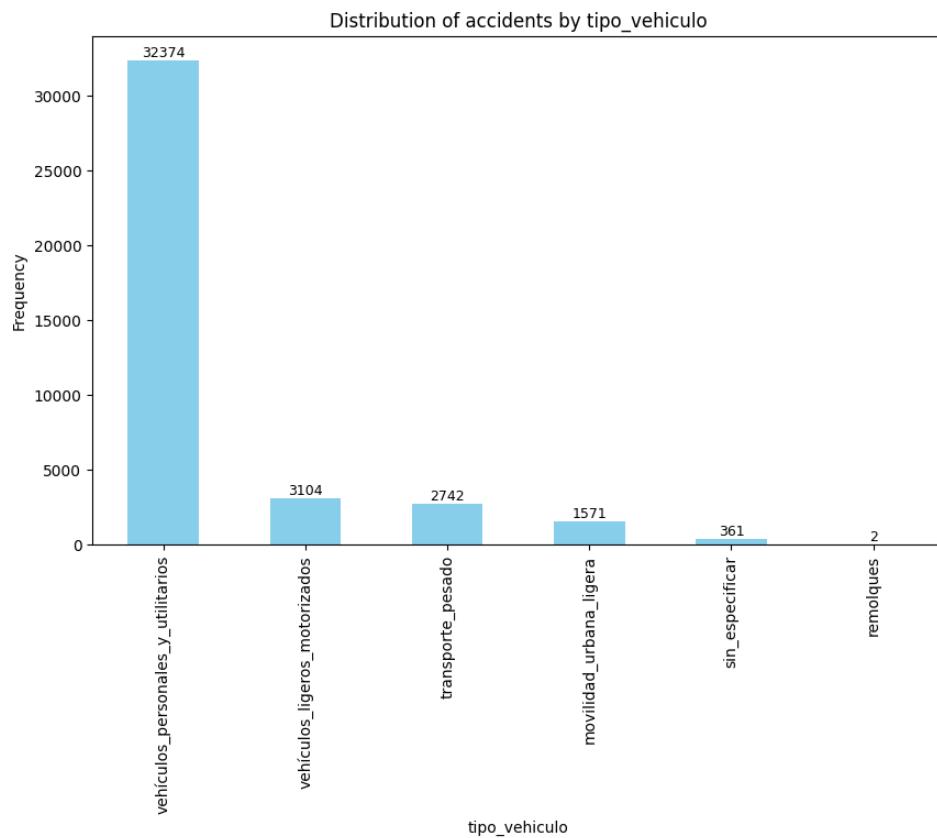
Nos quedaremos con la lesión más grave.

3.4.3.5. Columna positivo_alcohol.

Haremos una columna que represente si existe algún implicado que haya consumido alcohol.

3.4.3.6. Columna tipo_vehículo.

Simplificamos los tipos de vehículos y hacemos una columna por tipo de vehículo que reflejará la cantidad de vehículos de cada tipo.



4. Aplicación de algoritmos de clustering.

4.1. Algoritmos jerárquicos.

En este punto, visualizamos los distintos dataframes (el siguiente apartado) para decidir si usar los datos originales, escalados o normalizados ya que no nos daban resultados satisfactorios. Por ello decidimos explorar otras posibilidades como un escalado y un posterior normalizado para buscar el mejor modelo.

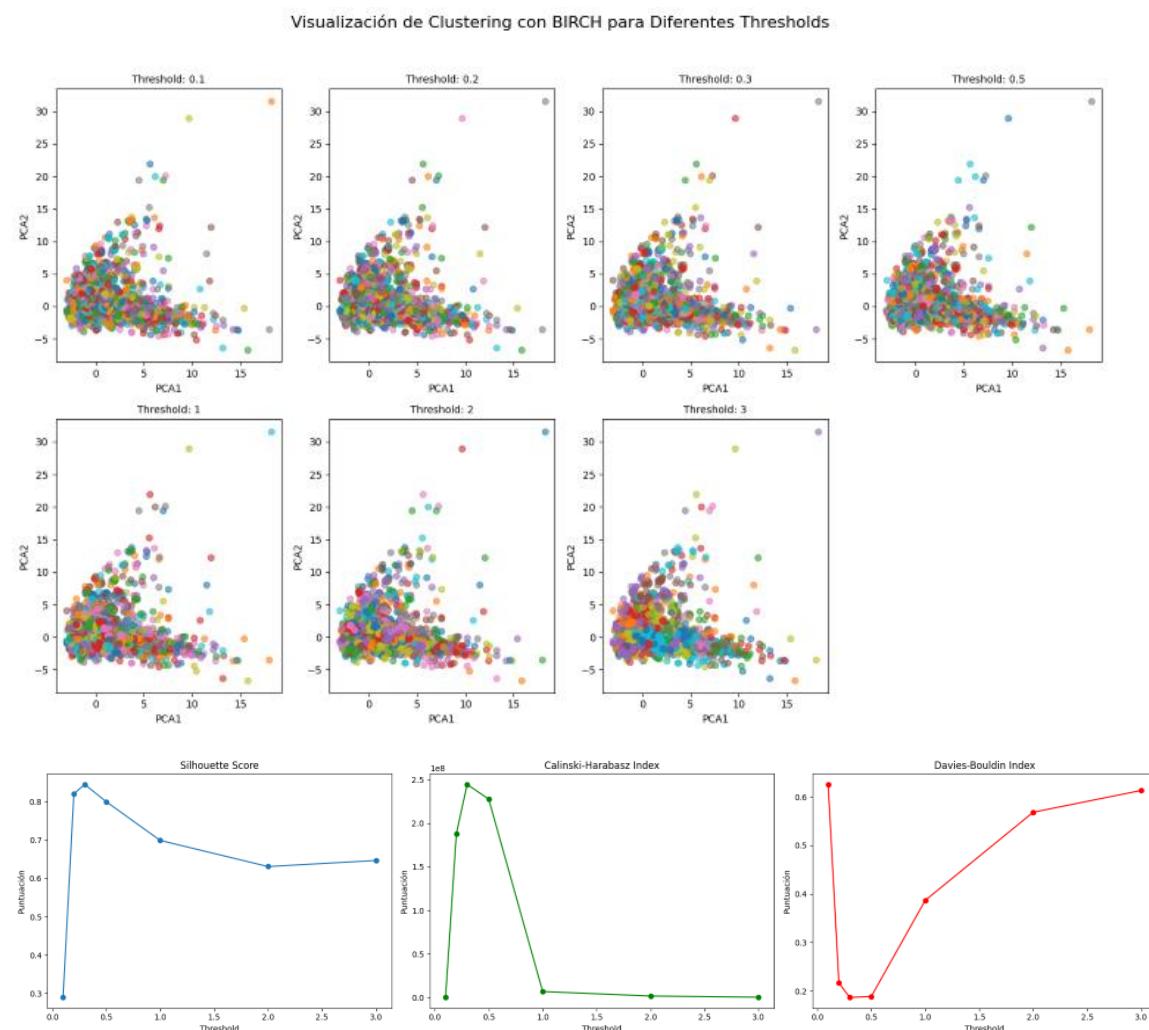
4.1.1. Aplicación de algoritmos.

Vamos a usar el BIRCH y Linkage.

4.1.1.1. BIRCH.

Hay que ajustar el threshold para que se ajuste a la cantidad de clusters que queremos. Ese método solo sirve para escalados / normalizados.

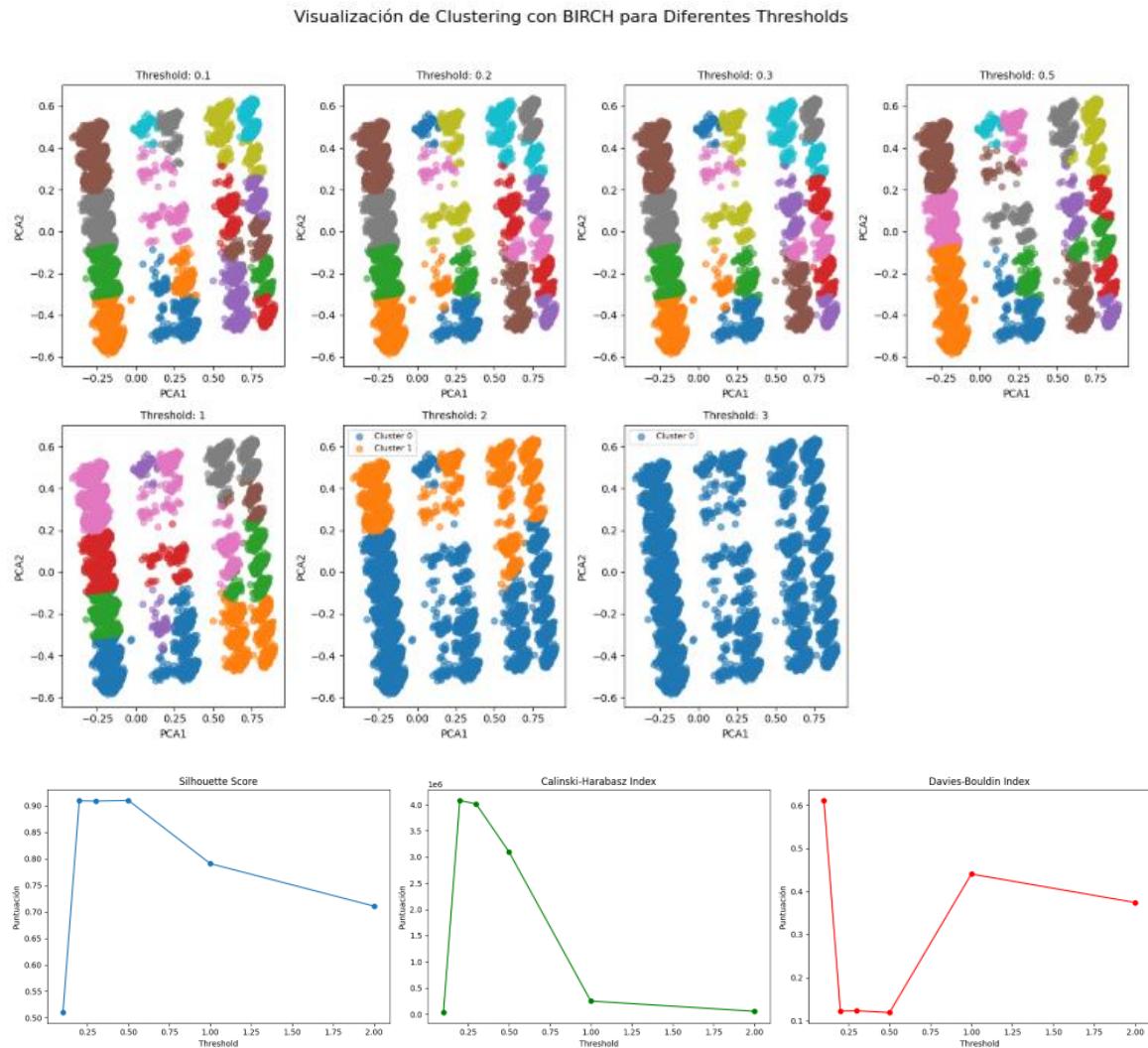
4.1.1.1.1. Datos escalados.



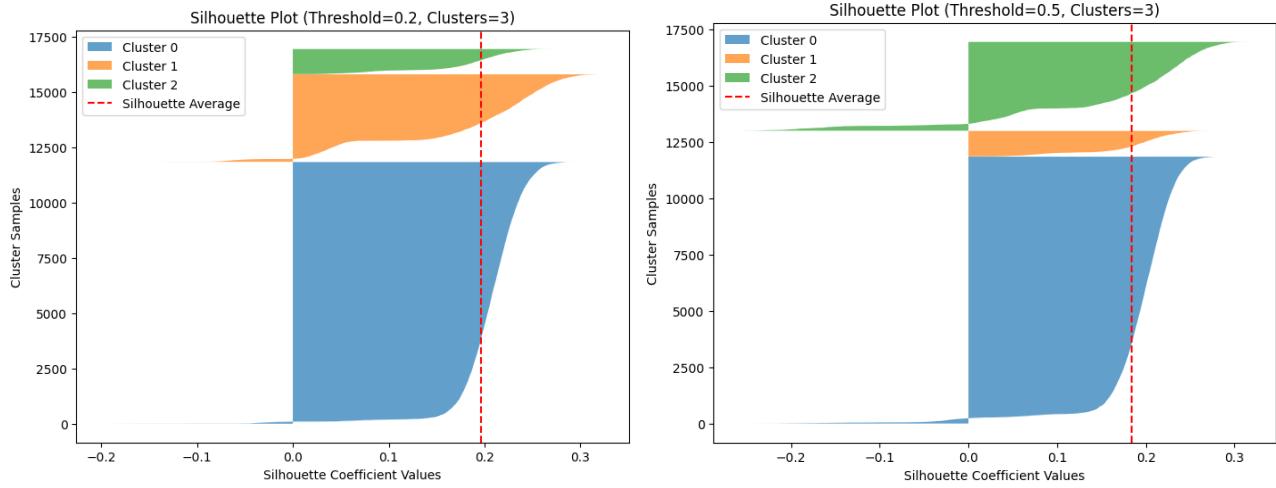
En la visualización de los datos escalados, podemos ver que los datos están más dispersos y no se aprecian bien los clusters, no tienen una forma clara. En consecuencia, tenemos un índice desproporcionado en Calinski ($2.5 * 10^8$). Eso puede indicar un número excesivo de clusters o datos artificialmente separados. Extrañamente tenemos buenos resultados en Silhouette y Davies-Bouldin.

En general, este algoritmo no es capaz de encontrar clusters en los datos escalados.

4.1.1.1.2. Datos normalizados.



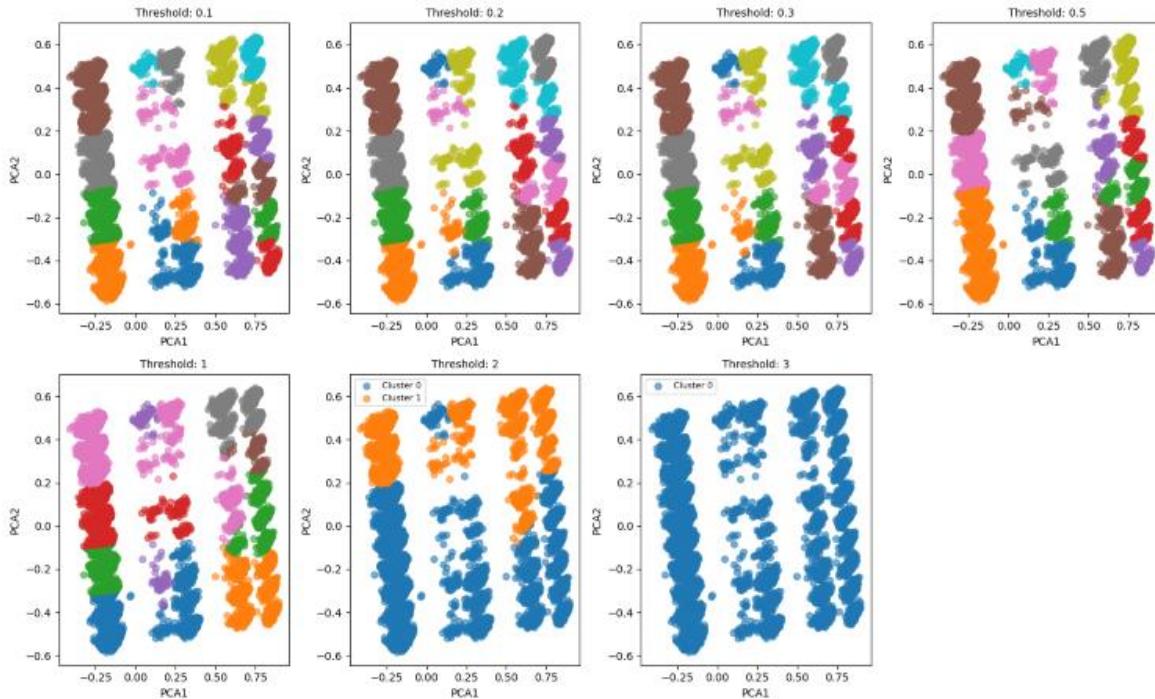
A simple vista, el dataframe normalizado parece tener una forma más clara de clusters. Los índices reflejan buenos resultados tanto en Silhouette como en Calinski y en Davies-Bouldin para valores de threshold de 0.2 e incluso 0.5. Este sería por ello un buen dataframe para aplicar el algoritmo BIRCH y sería interesante analizar las agrupaciones que se han hecho.

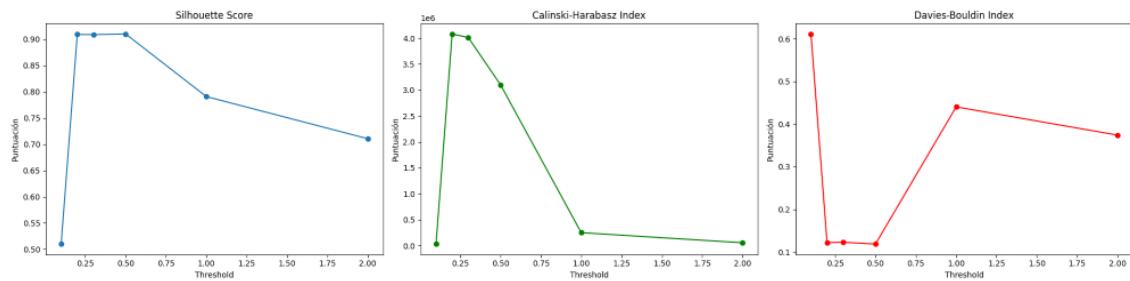


Da resultados buenos para 0.2. Tiene varios puntos mal clasificados en los clusters 0 y 2 y la forma del cluster 2 en esta gráfica refleja que hay puntos muy alejados del centro del cluster dentro del mismo. Además, el coeficiente de silhouete es muy bajo para todos los puntos en general, lo que indica que no es el algoritmo perfecto para ese dataframe.

4.1.1.1.3. Datos escalados y normalizados.

Visualización de Clustering con BIRCH para Diferentes Thresholds





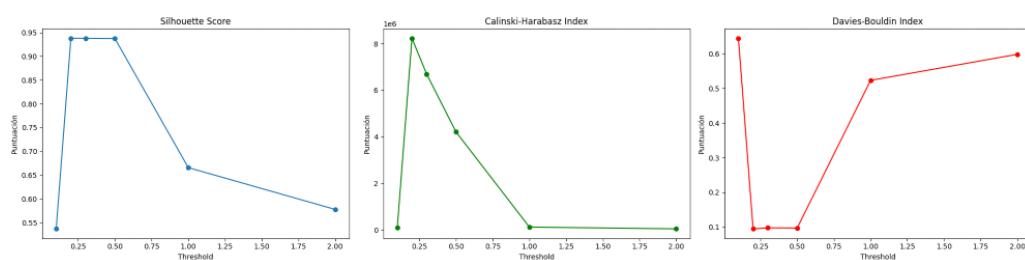
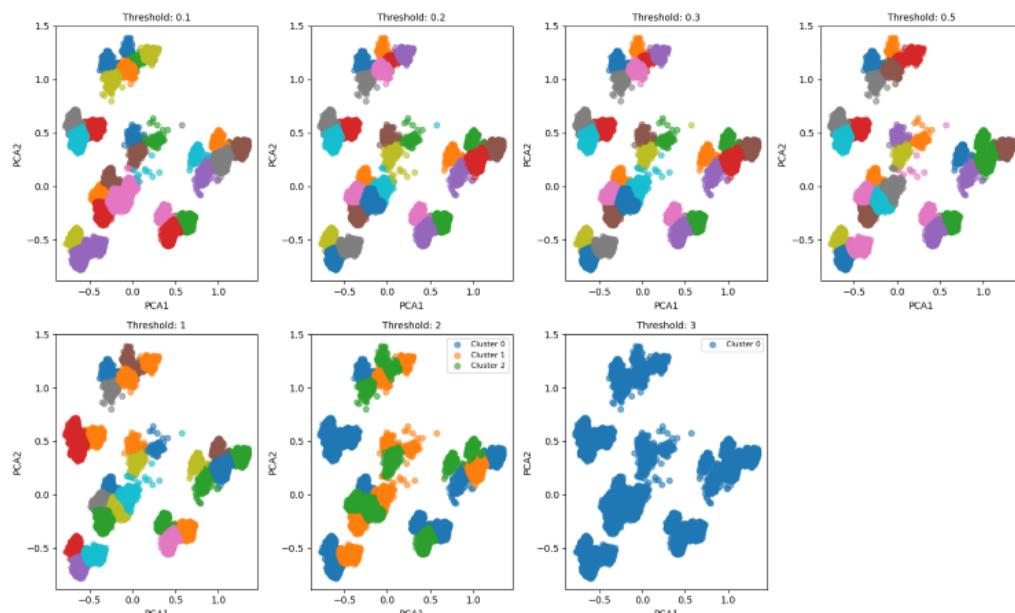
No presenta ninguna diferencia del dataframe normalizado, por lo que queda claro que no es necesario aplicar el escalado antes del normalizado y que da lugar a los mismos resultados que normalized.

4.1.1.1.4. Dataset “prueba101”.

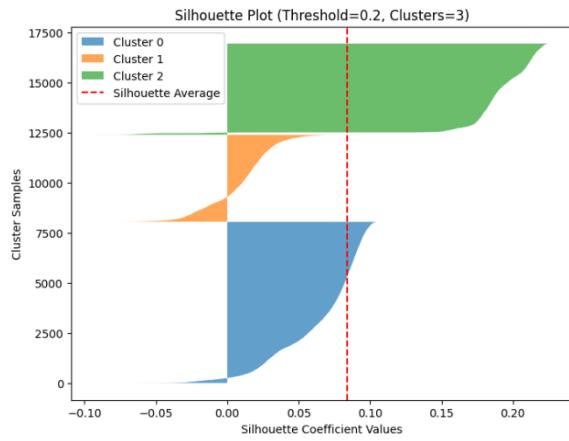
Como ninguno de los conjuntos nos daba resultados preferidos, decidimos a experimentar y crear otro dataframe. Las columnas principalmente codificadas con label encoder (tipo_accidente y estado_meterologico), cambiadas a one-hot encoding. También hemos cambiado las columnas de fecha y hora en funciones cíclicas y hemos eliminado la columna cod_distrito.

4.1.1.1.4.1. Datos normalizados.

Visualización de Clustering con BIRCH para Diferentes Thresholds



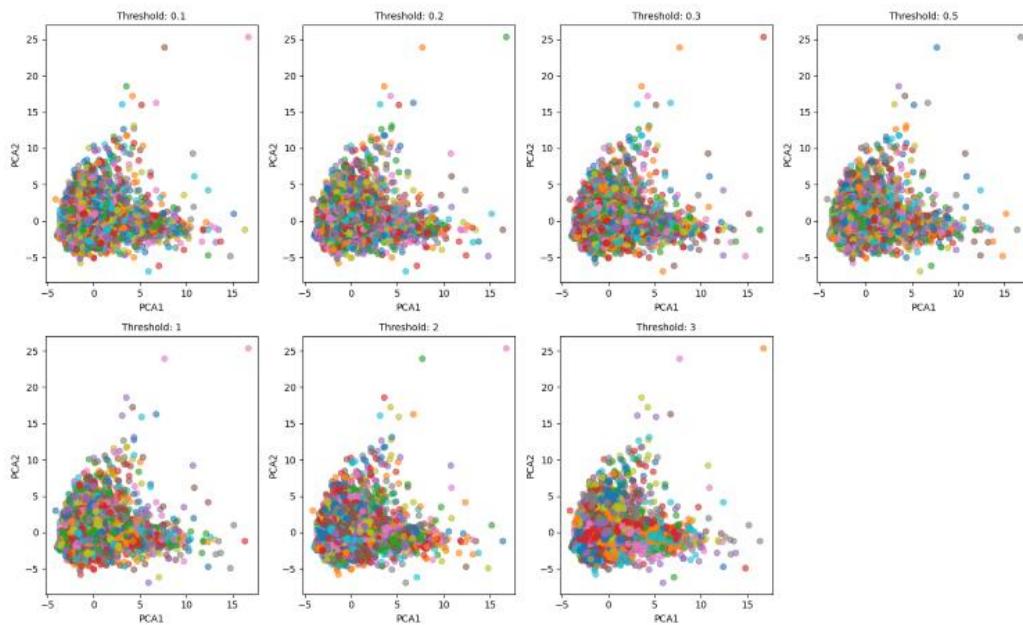
De la misma manera que ha pasado con normalized, al tener una distribución más agrupable a simple vista, los índices son buenos en Silhouette, Calinski y Davies-Bouldin. Es incluso mejor que normalized.

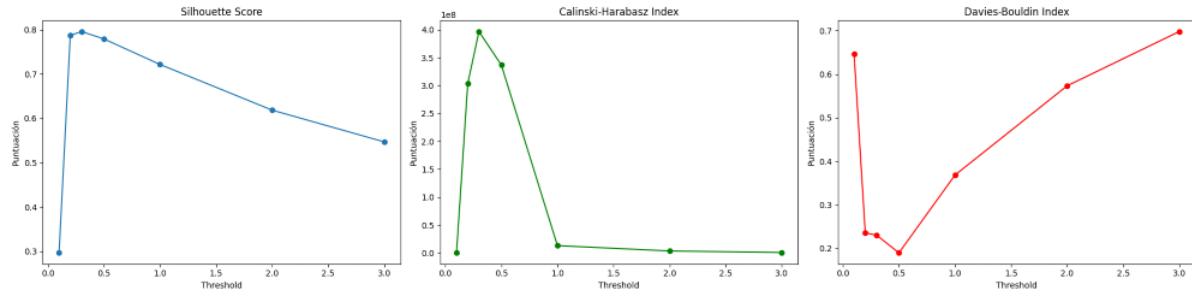


En este caso no da resultados tan buenos como normalized, hay muchos datos que se clasifican mal en el cluster 1 y los clusters tienen puntos muy dispersos en el mismo cluster. Además, el cluster 1 tiene un coeficiente de silhouette promedio muy bajo.

4.1.1.1.4.2. Datos escalados.

Visualización de Clustering con BIRCH para Diferentes Thresholds





Al tener el mismo aspecto que scaled_df, tiene sentido que los índices sean similares. Simplemente se ha reducido el valor de Calinski pero sigue siendo desproporcionado y los clusters difusos.

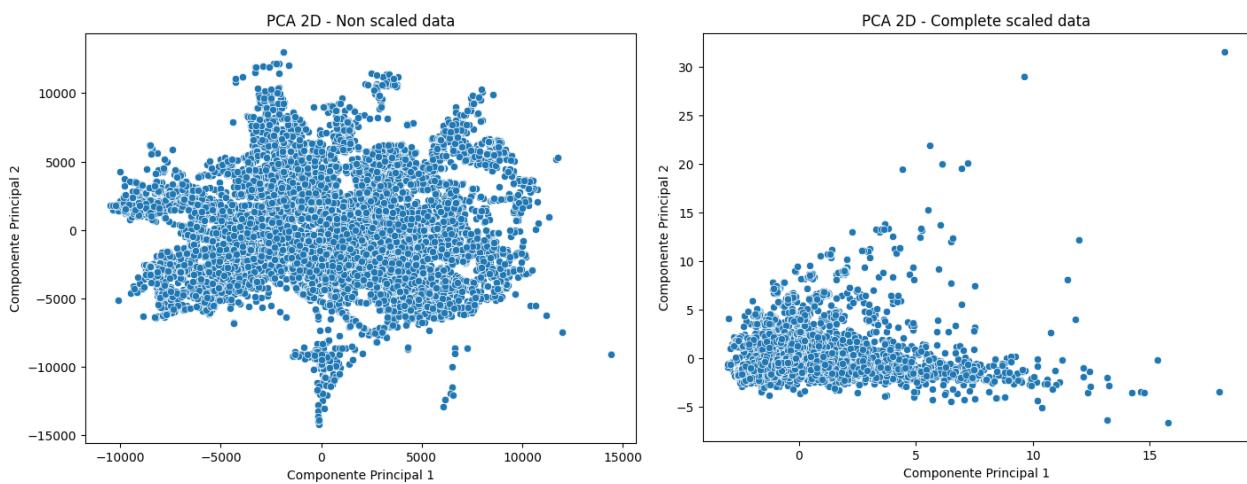
4.1.1.1.5. Conclusión.

BIRCH da muy buenos resultados en general pero da valores exageradamente altos en Calinski y en los casos en los que no se aprecian agrupaciones a primera vista, presenta clusters difusos. No es el algoritmo más adecuado para este dataset pero tras analizar el algoritmo de LINKAGE, muestra mejores resultados.

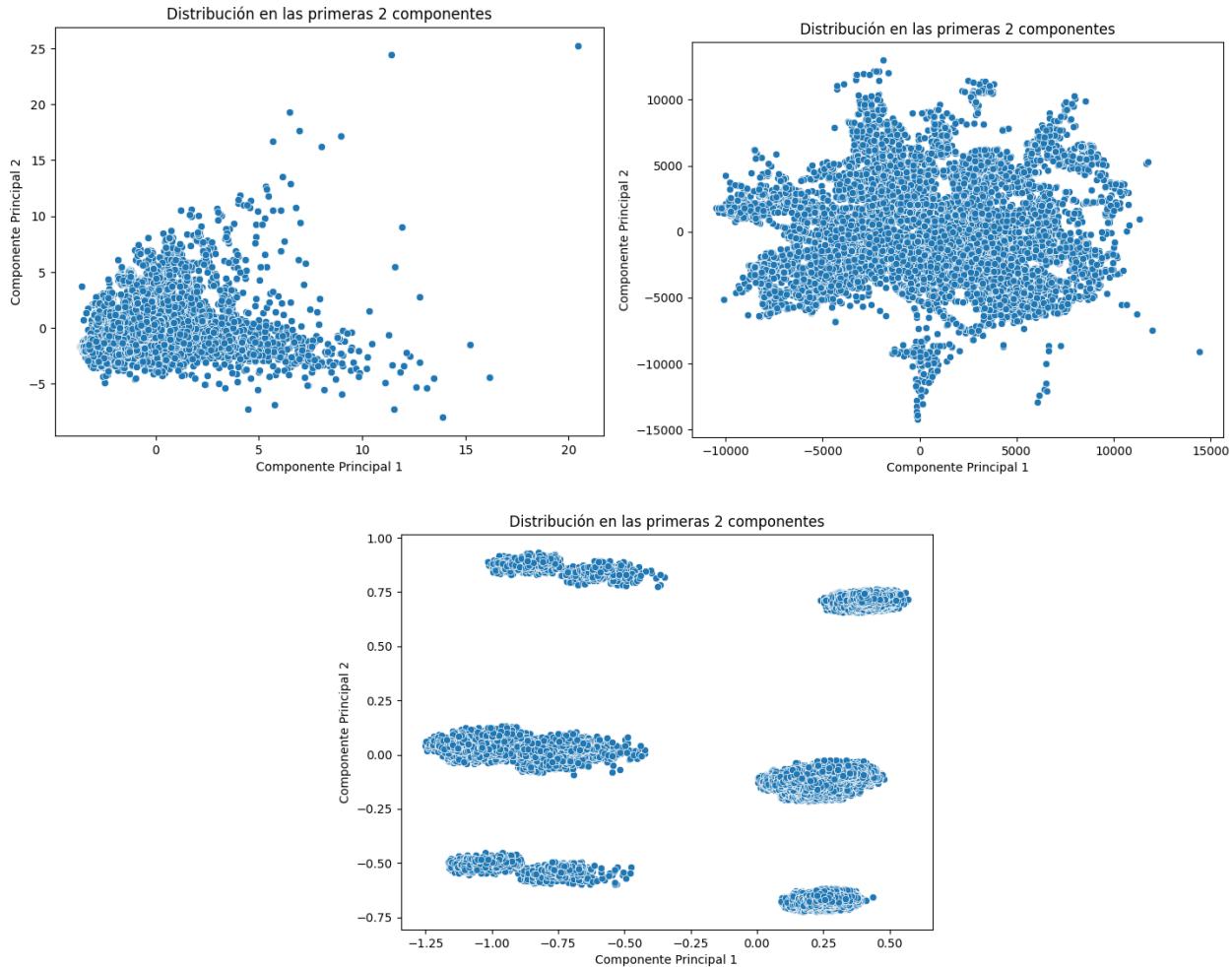
4.1.1.2. Linkage.

En un principio probamos directamente a estudiar los resultados con un dataframe naturalmente escalado ya que los algoritmos de clustering jerárquico son sensibles a la escala de los datos debido a que tratan con distancias. Llegamos incluso a realizar un escalado solo de las columnas de coordenadas por ser las únicas que no estaban en el mismo rango que el resto de columnas y porque temíamos que una distorsión de las variables categoricas codificadas podría suponer una pérdida de información o legibilidad.

Al final comprendimos que esto no tenía sentido ya que el objetivo de una estandarización es dar lugar a una distribución de datos con media 0 y desviación estándar 1, y solo modificando dos columnas, no es posible conseguir esto.



También hemos probado como varían los resultados para nuestro dataframe “prueba101” (en orden) no escalado, escalado y normalizado.

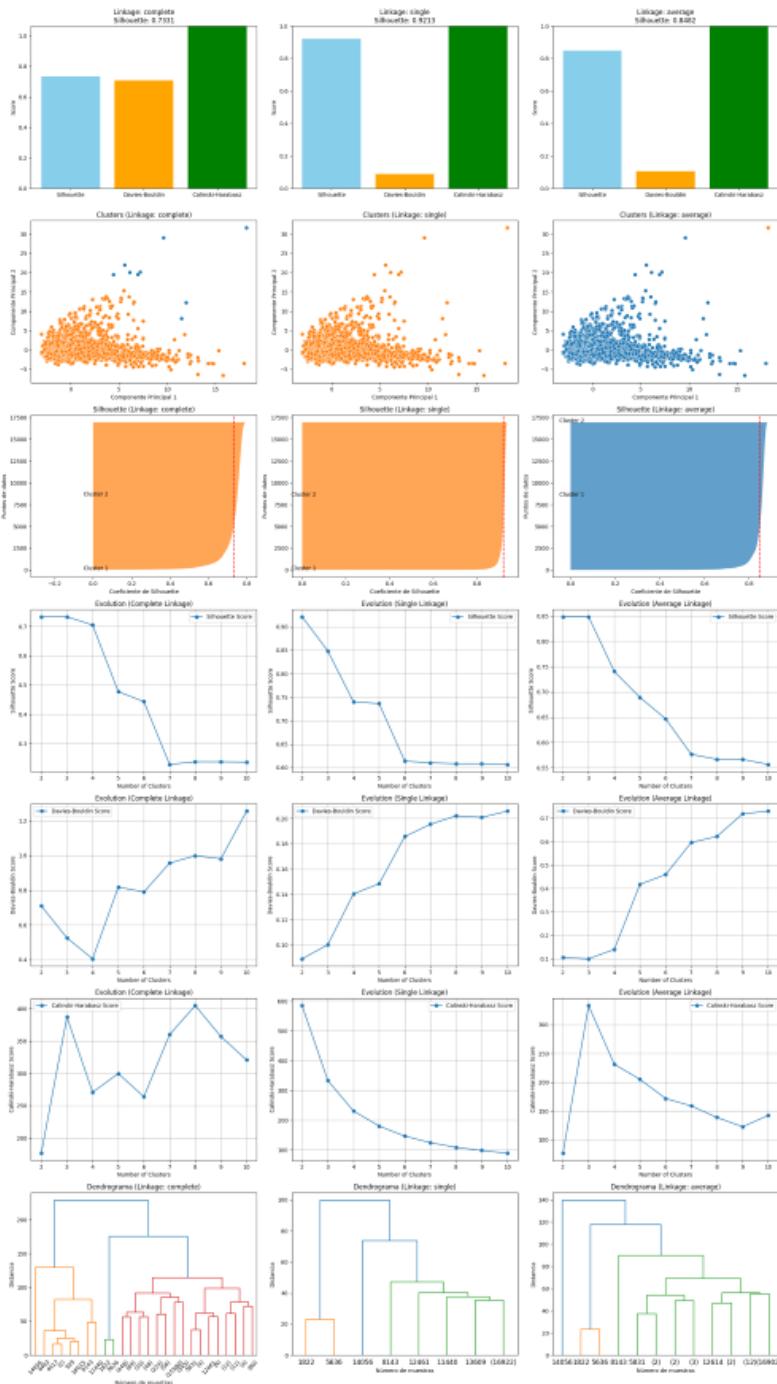


4.1.2. Número óptimo de clusters.

Hemos intentado a evaluar los resultados basándose en varios conjuntos de datos. Para cada dataframe y cada distancia hemos visualizado los valores de los índices de calidad, clusterización con varios tipos de **linkage**, la evaluación de los valores de los índices entre varios números de clusters y las diferencias entre dendrogramas para cada cluster.

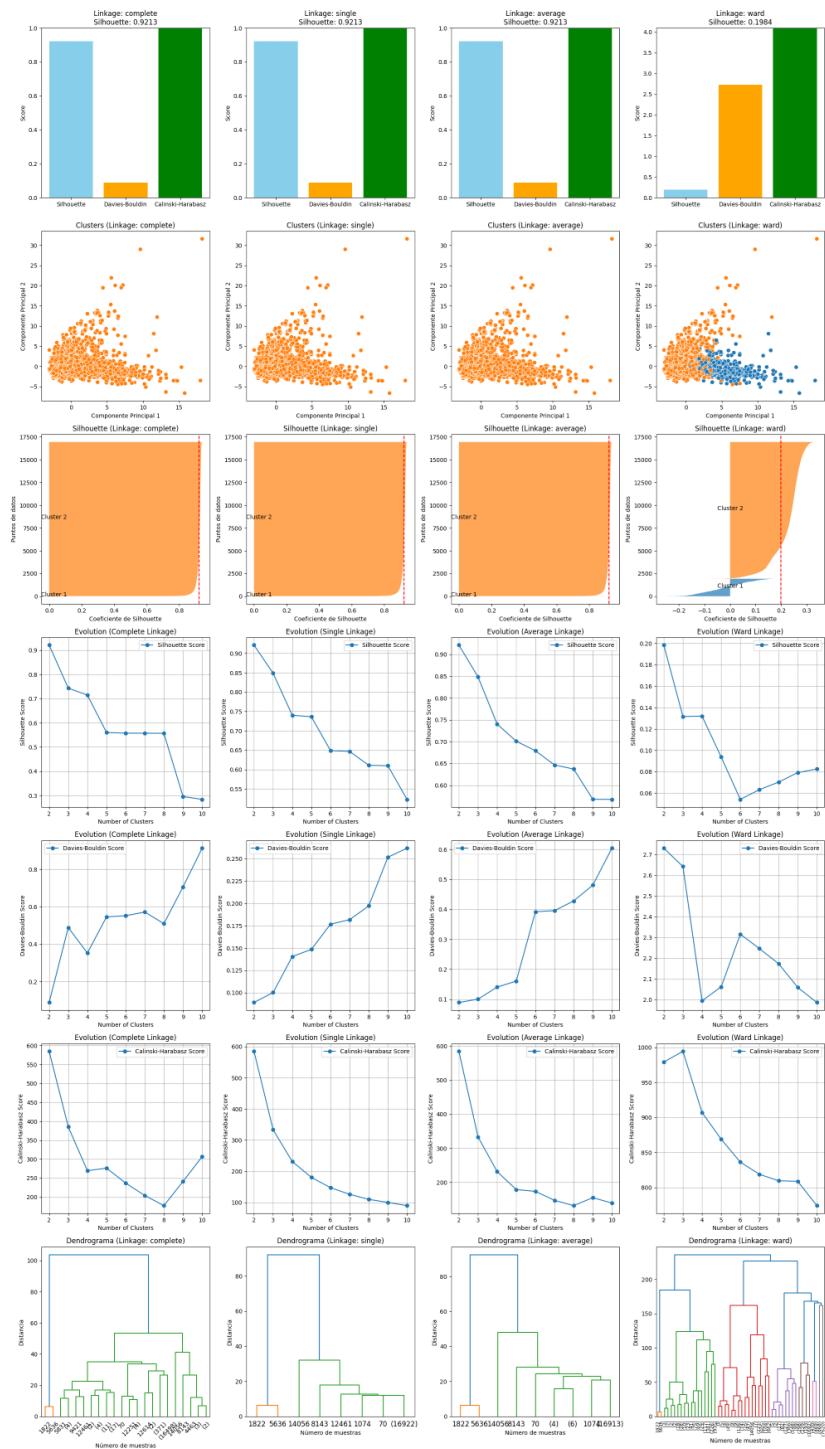
4.1.2.1. Datos escalados.

4.1.2.1.1. Distancia city_block.



Para la distancia city_block, los mejores resultados se obtienen con el método de linkage single, ya que produce el valor más alto del índice de Silhouette. Aunque la separación no es claramente visible en el gráfico de dispersión, el dendrograma muestra una estructura simple y separa claramente los datos en dos clusters. En todos los índices evaluados, el número óptimo de clusters es dos, ya que maximiza los valores de Silhouette y Calinski-Harabasz, y minimiza el valor de Davies-Bouldin.

4.1.2.1.2. Distancia euclídea.



En caso de distancia *euclidean*, los peores resultados se obtienen con linkage ward. Los otros linkages son realmente similares. Los dendrogramas de single y average son claros y bien separadas y el número óptimo de clusters también es 2. Los resultados no son tan buenos como para la distancia de *city_block*.

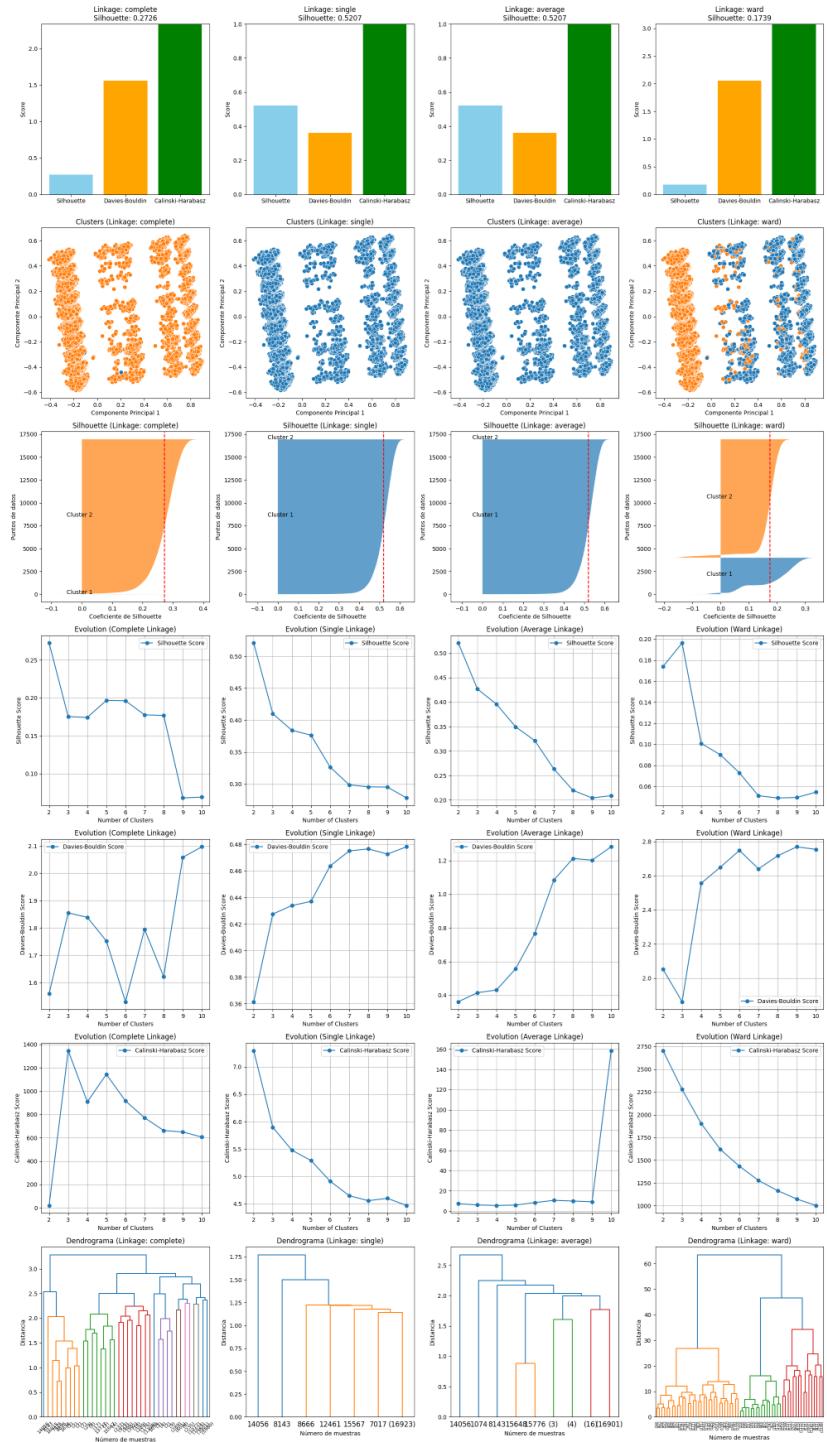
4.1.2.1.3. Distancia cosena.



La distancia **cosena** da peores resultados de todos. En caso de linkage complete y average, los valores de silhouette son muy bajos, cuando el valor de Davis-Bouldin sube mucho. Los dendrogramas son muy complejos y difíciles a leer. En caso de linkage single, los resultados son visiblemente mejores, pero la estructura del dendrograma muestra que no hay mucha diferencia entre grupos y que la separación no funciona bien.

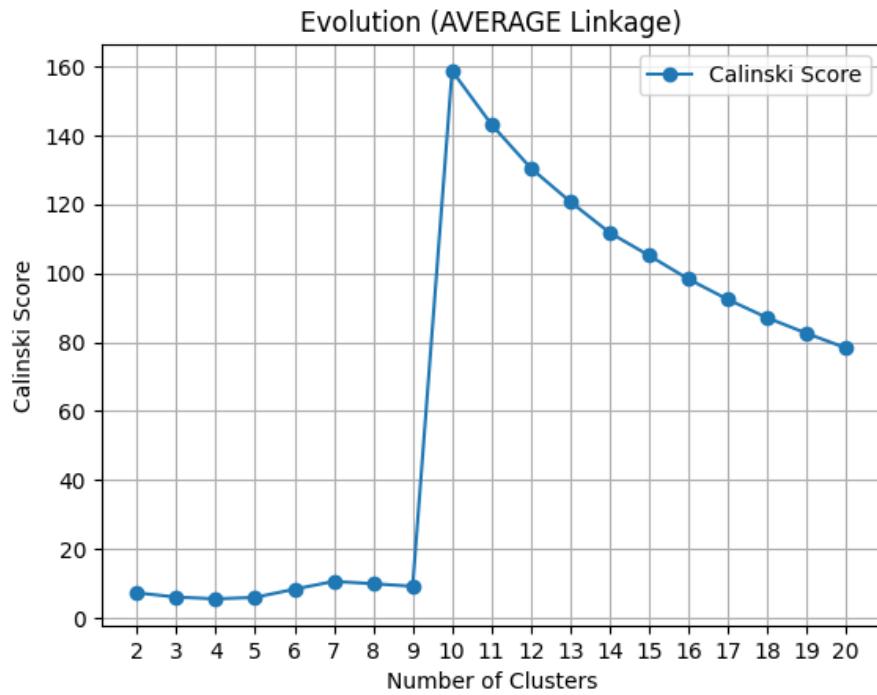
4.1.2.2. Datos normalizados.

4.1.2.2.1. Distancia euclídea.



Cuando usamos el dataframe normalizada y distancia euclídea, vemos que los resultados se peroran. Otra vez, es más lógico usar dos clusters, ya que para ese número obtenemos los valores de índices buenos. Linkages complete y ward, nos dan dendrogramas complejos. Para linkages single y average las agrupaciones son muy débiles.

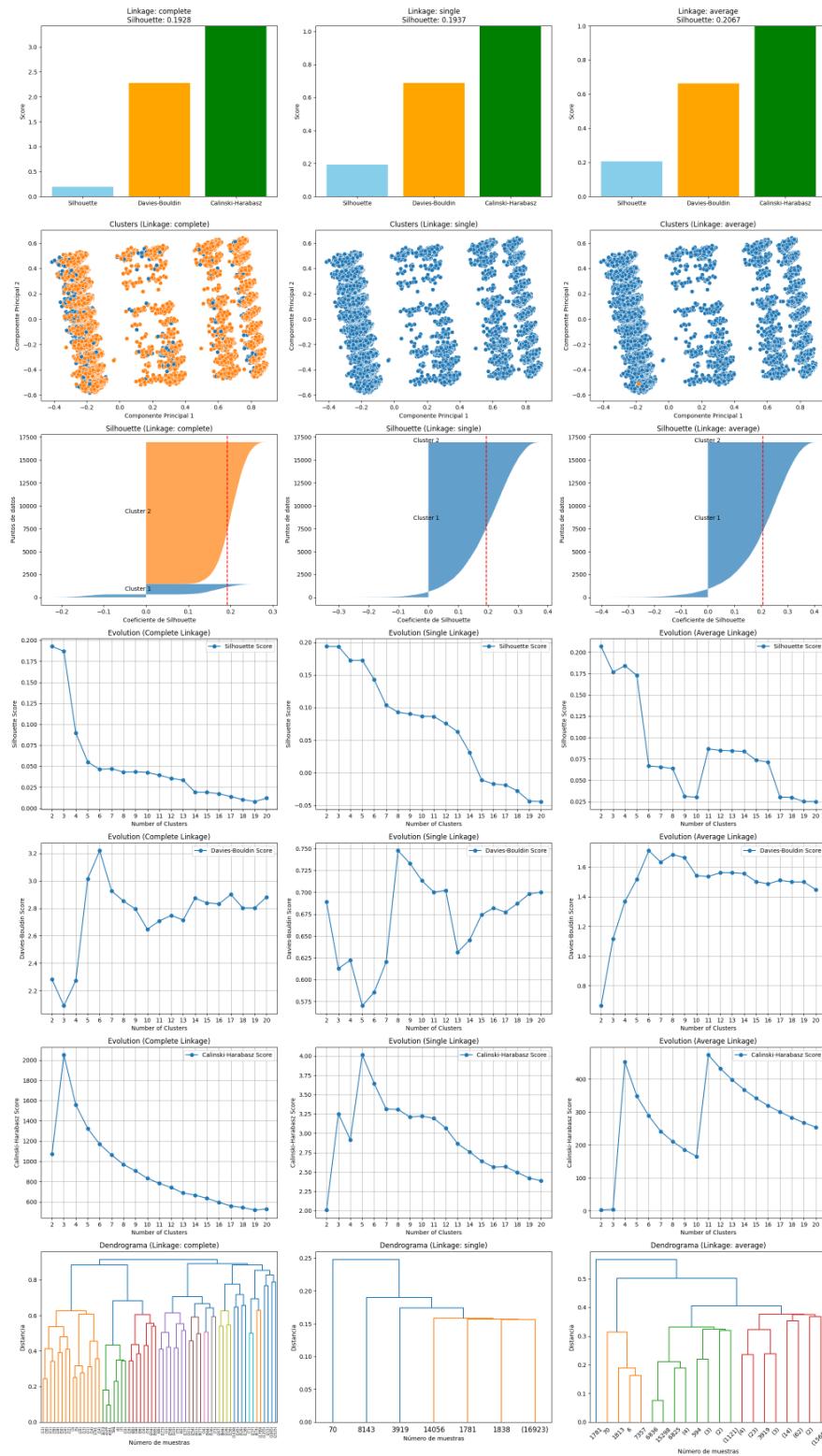
Como el valor del índice de Calinski-Harabasz sube muchísimo entre cluster 9 y 10, queremos ver si para el número de clusters más grande, la subida continua.



Pero mirando la gráfica podemos ver, que después del cluster 10 los valores vuelven a bajar.

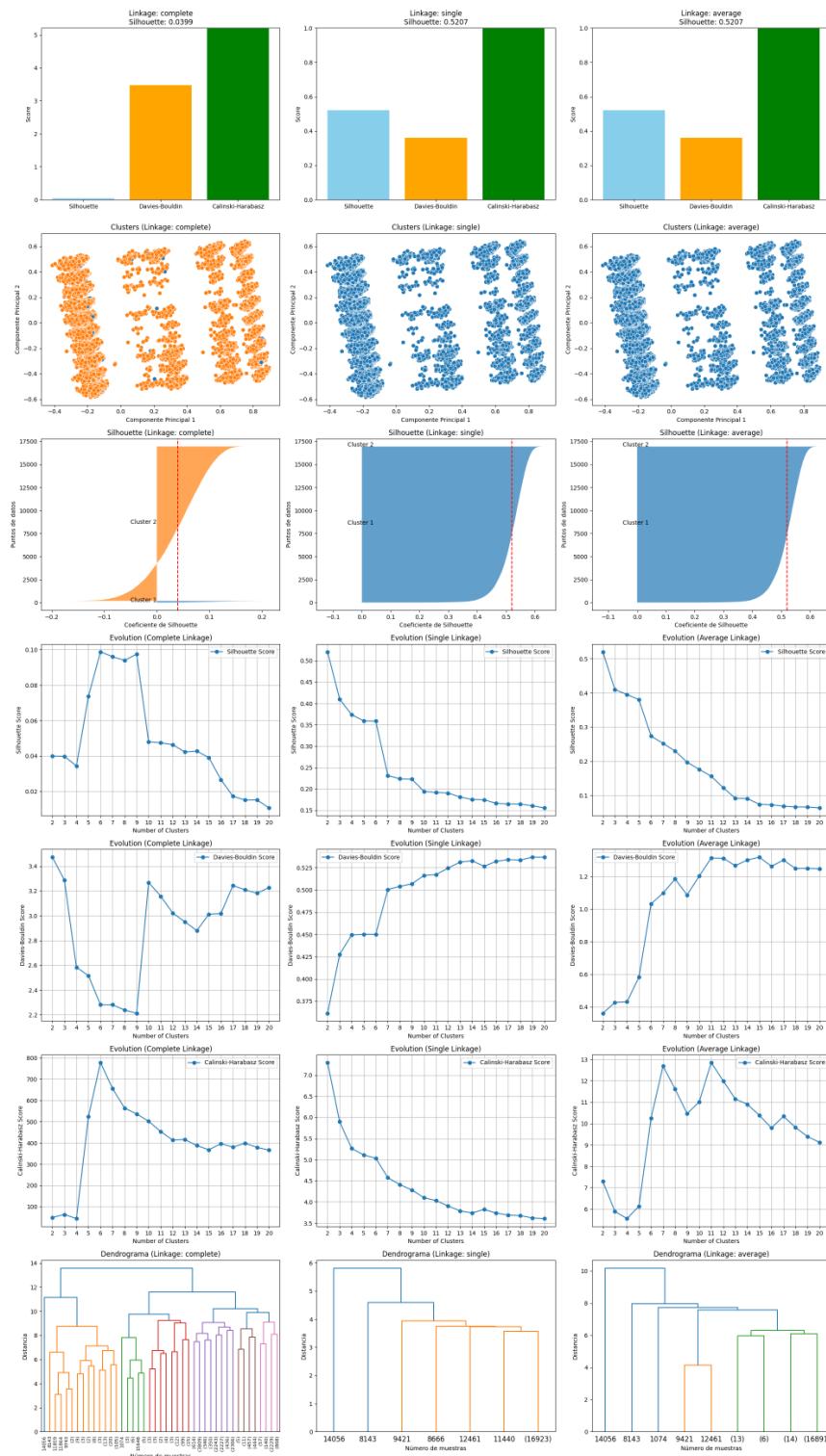
Vamos a usar el número máximo de clusters del valor 20 para otras distancias también, para ver si, como en caso de la distancia euclídea, los valores se cambian.

4.1.2.2. Distancia cosena.



La distancia cosena nos deja con muchos outliers, lo que se puede ver en las gráficas de fila 3. Los valores de índices de calidad varían mucho entre números de clusters y las agrupaciones no son buenas.

4.1.2.2.3. Distancia city_block.

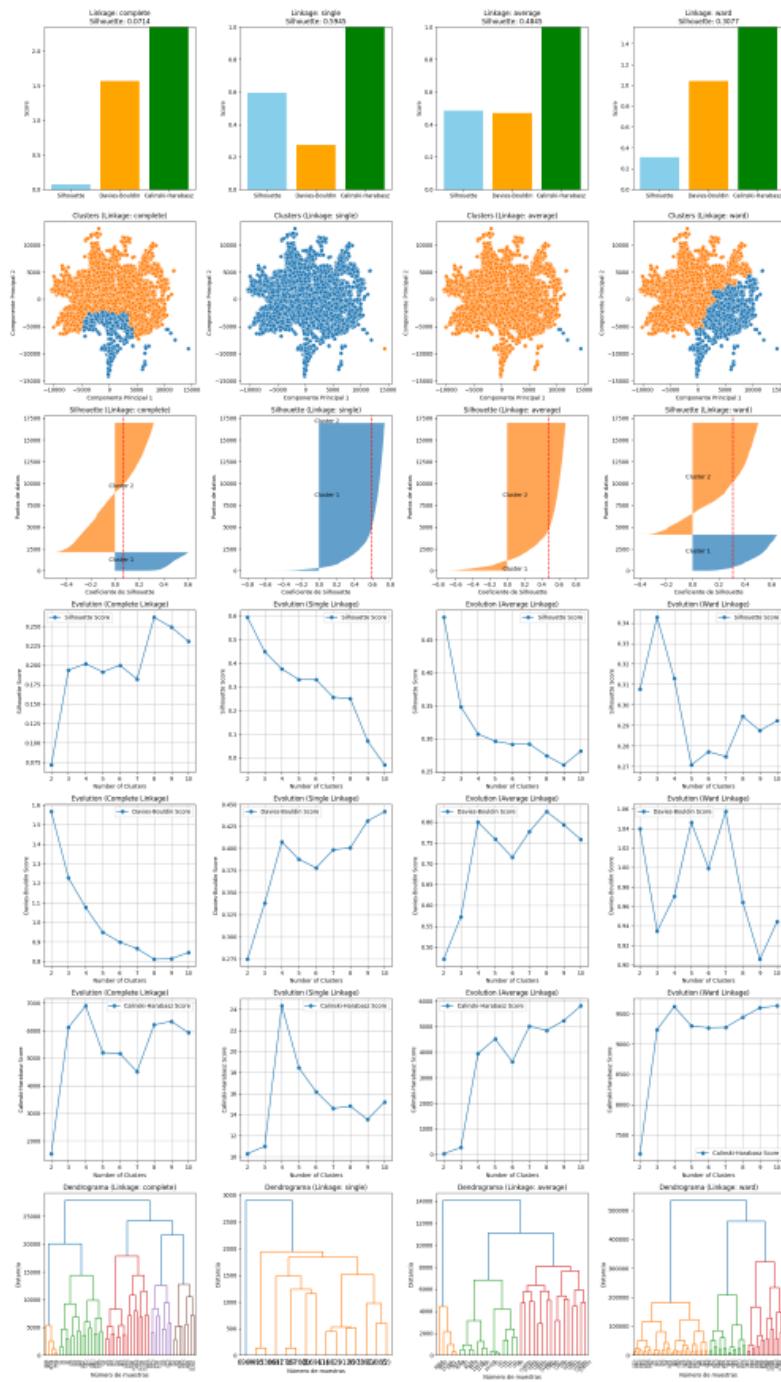


En caso de la distancia de city_block, los linkages single y average dan resultados bastante buenos. Sin embargo, los dendrogramas muestran agrupaciones débiles y los valores de índices, aunque bastante correctos, no llegan altos (con el silhouette) ni bajos (con Davies-Bouldin).

4.1.2.3. Datos agrupados.

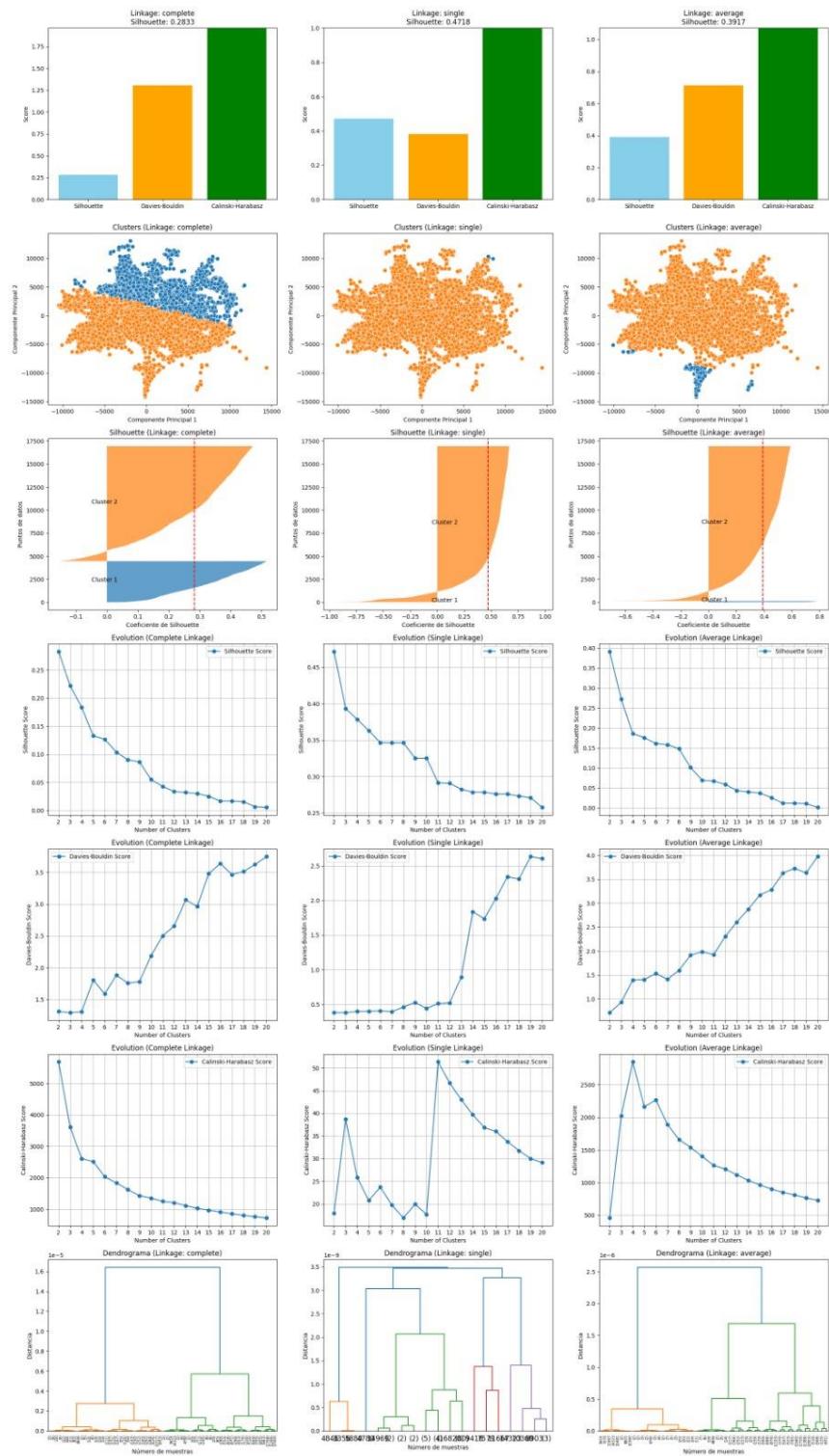
Como el conjunto agrupado no está ni escalado ni normalizado, esperamos que los resultados serán peores que en otros casos.

4.1.2.3.1. Distancia euclídea.



Para la distancia euclídea, tenemos muchos outliers, especialmente en el caso de linkage complete y link. Los dendrogramas están densos y, en el caso de linkage single - mal agrupado. Los valores de índices no son buenos. El número óptimo de clúster varía entre 2 y 3.

4.1.2.3.2. Distancia cosena.



Los resultados para la distancia cosena son mucho más peores. Ni los dendrogramas ni los índices parecen bien.

4.1.2.3.3. Distancia city_block.



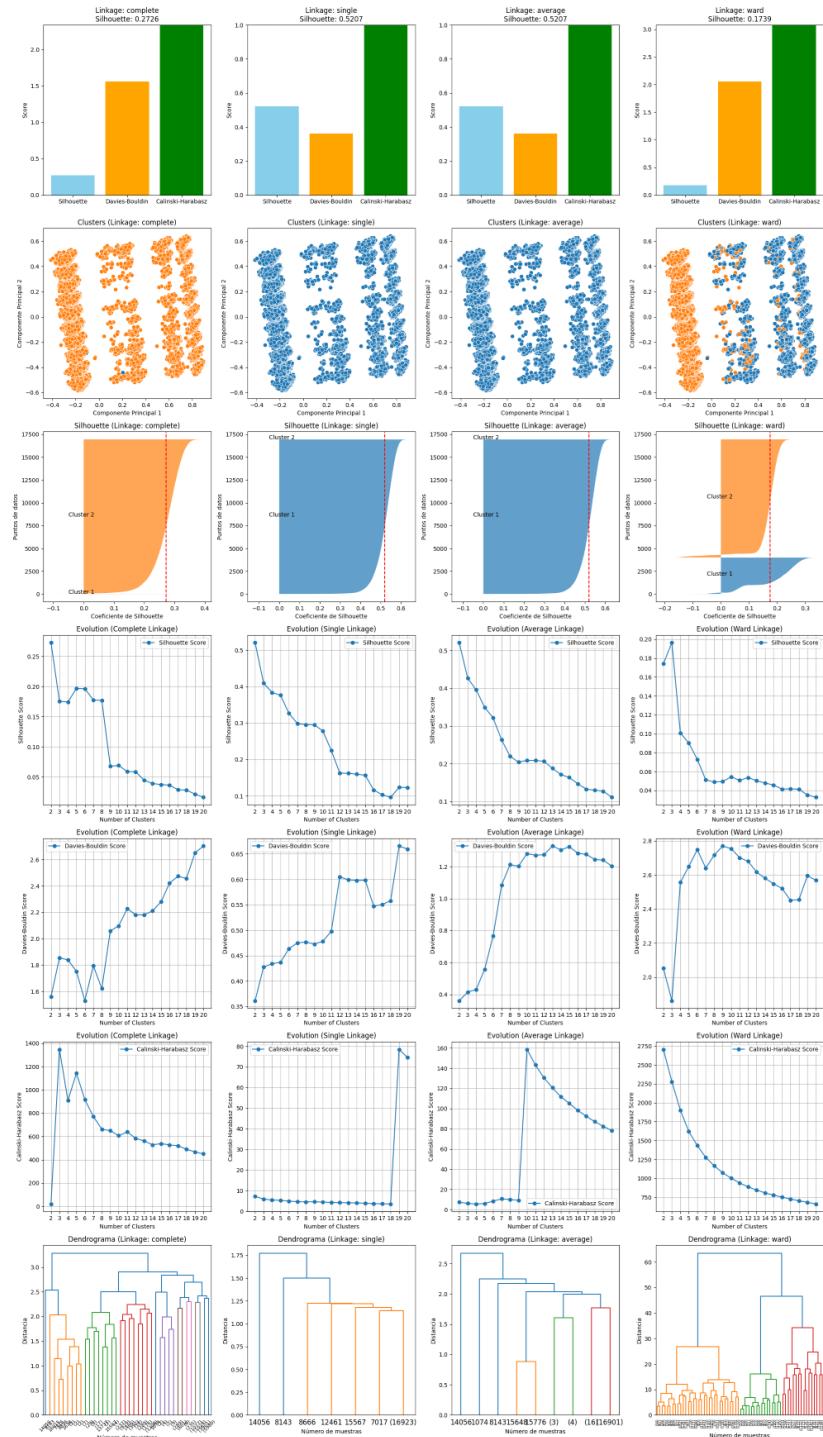
Podemos ver que los valores de los índices no son buenos, tampoco son las agrupaciones en los dendrogramas. Otra vez, el número de clusters óptimo varía entre 2 y 3.

Mirando los resultados obtenidos para la distancia de city_block y las otras distancias vistas antes, podemos decir que usar el dataframe normalizado no sea la mejor opción.

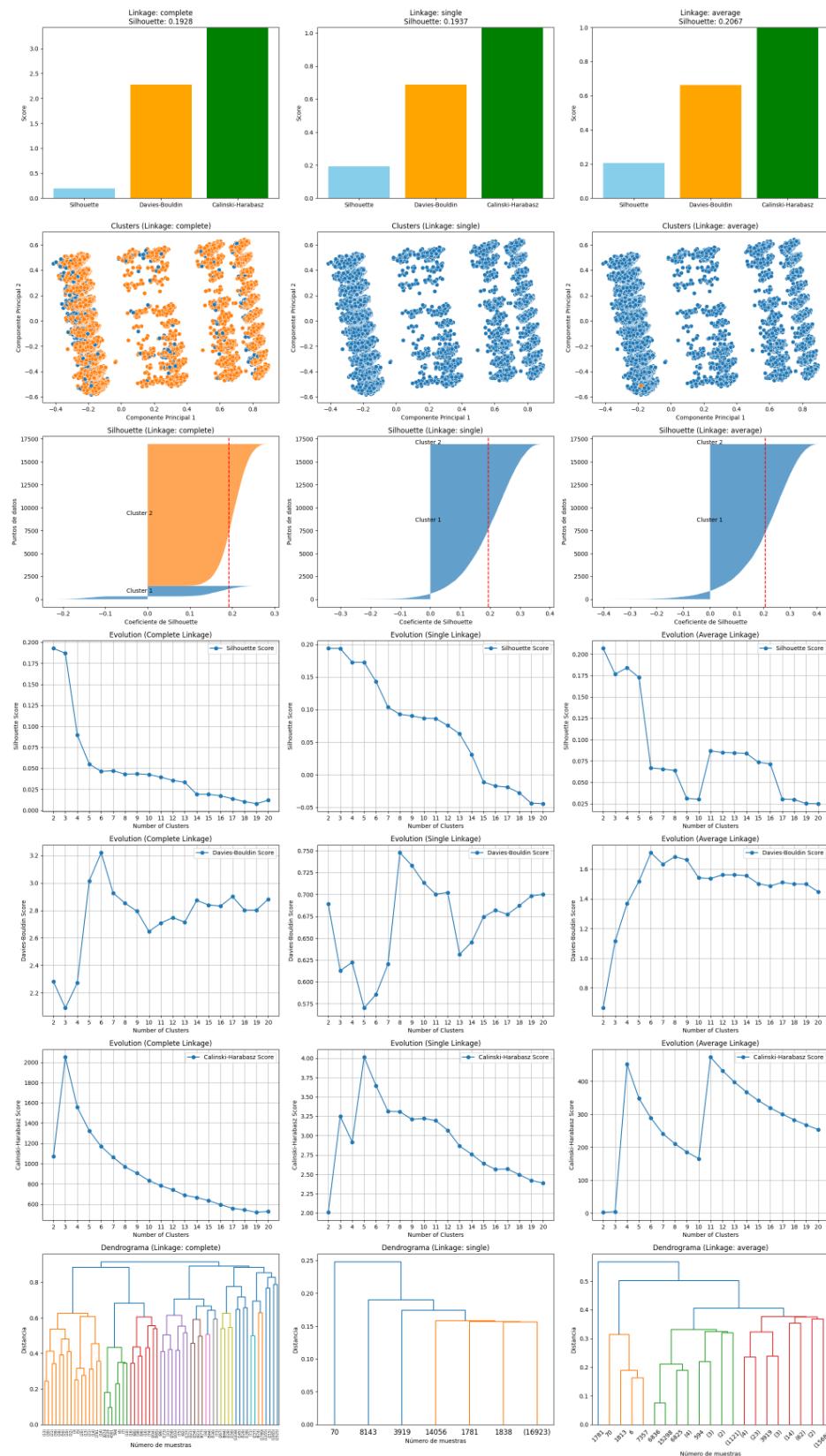
4.1.2.4. Datos escalados y normalizados.

Como queremos probar varias opciones, hemos creado también un conjunto de datos que está primero escalado y entonces normalizado. Sin embargo, los resultados obtenidos son casi los mismos como en caso del conjunto que solo está normalizado. Y como los resultados para el dataframe normalizado no son buenos, tampoco vamos a usar un conjunto escalado y normalizado.

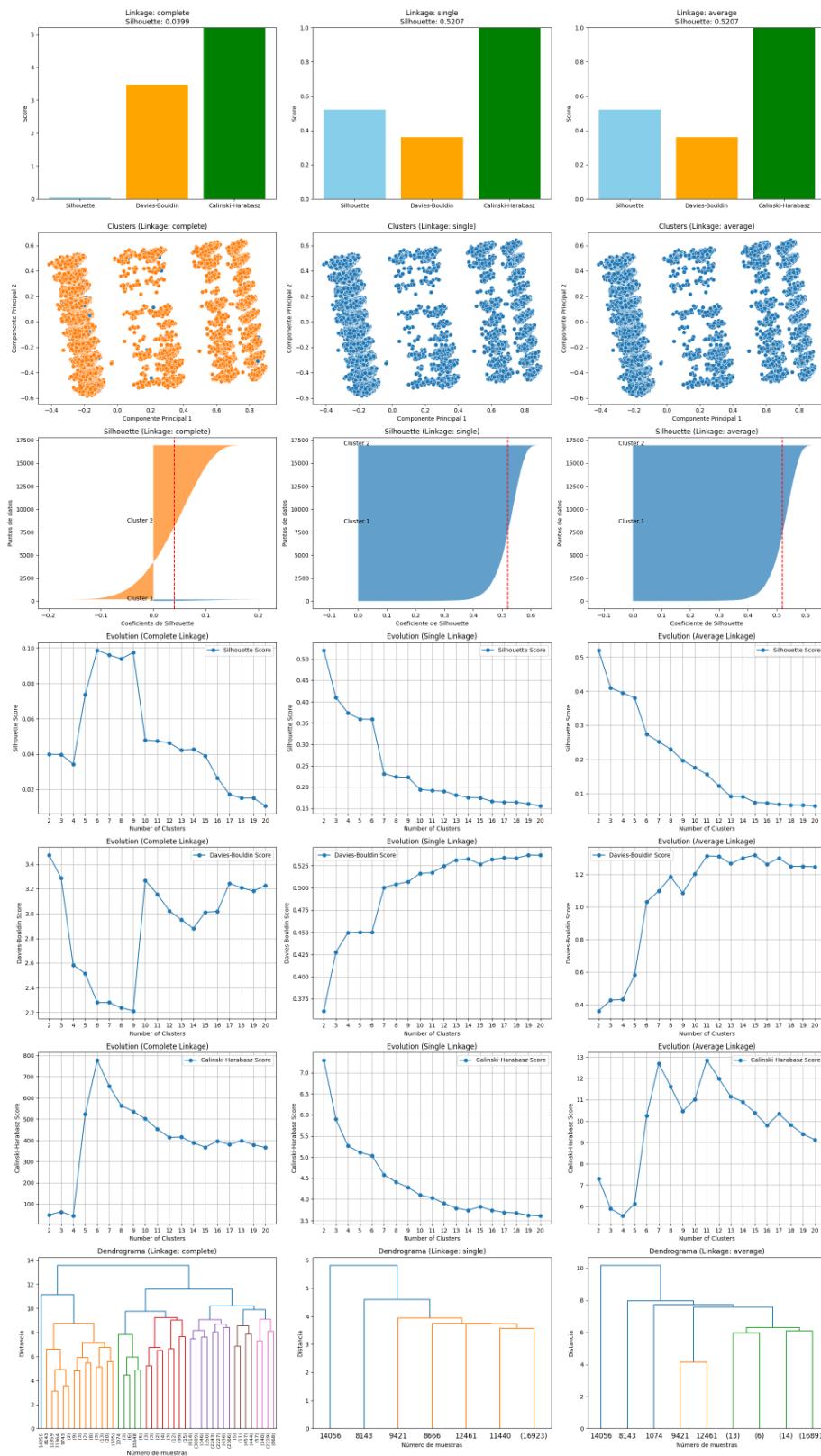
4.1.2.4.1. Distancia euclídea.



4.1.2.4.2. Distancia cosena.



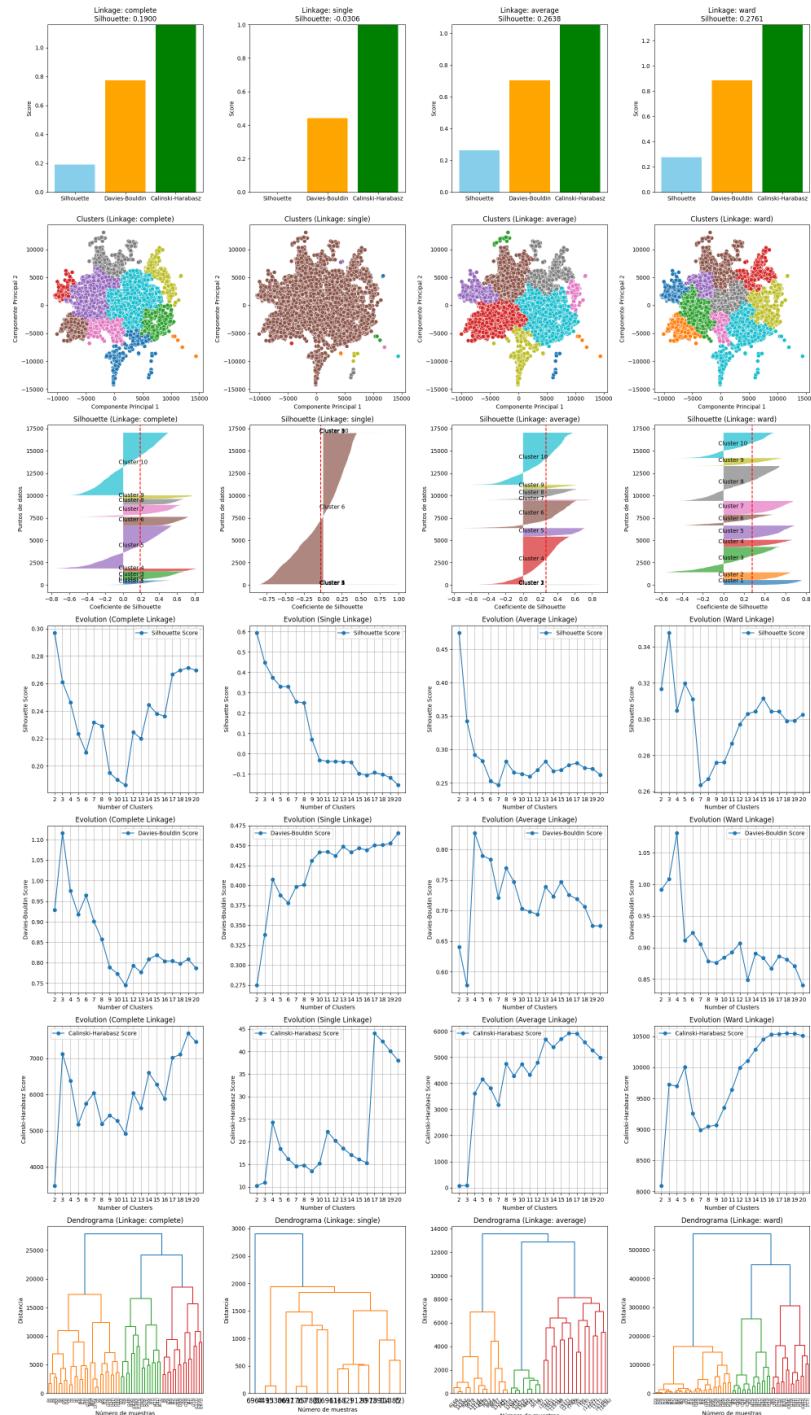
4.1.2.4.3. Distancia city_block.



4.1.2.5. Dataset “prueba101”.

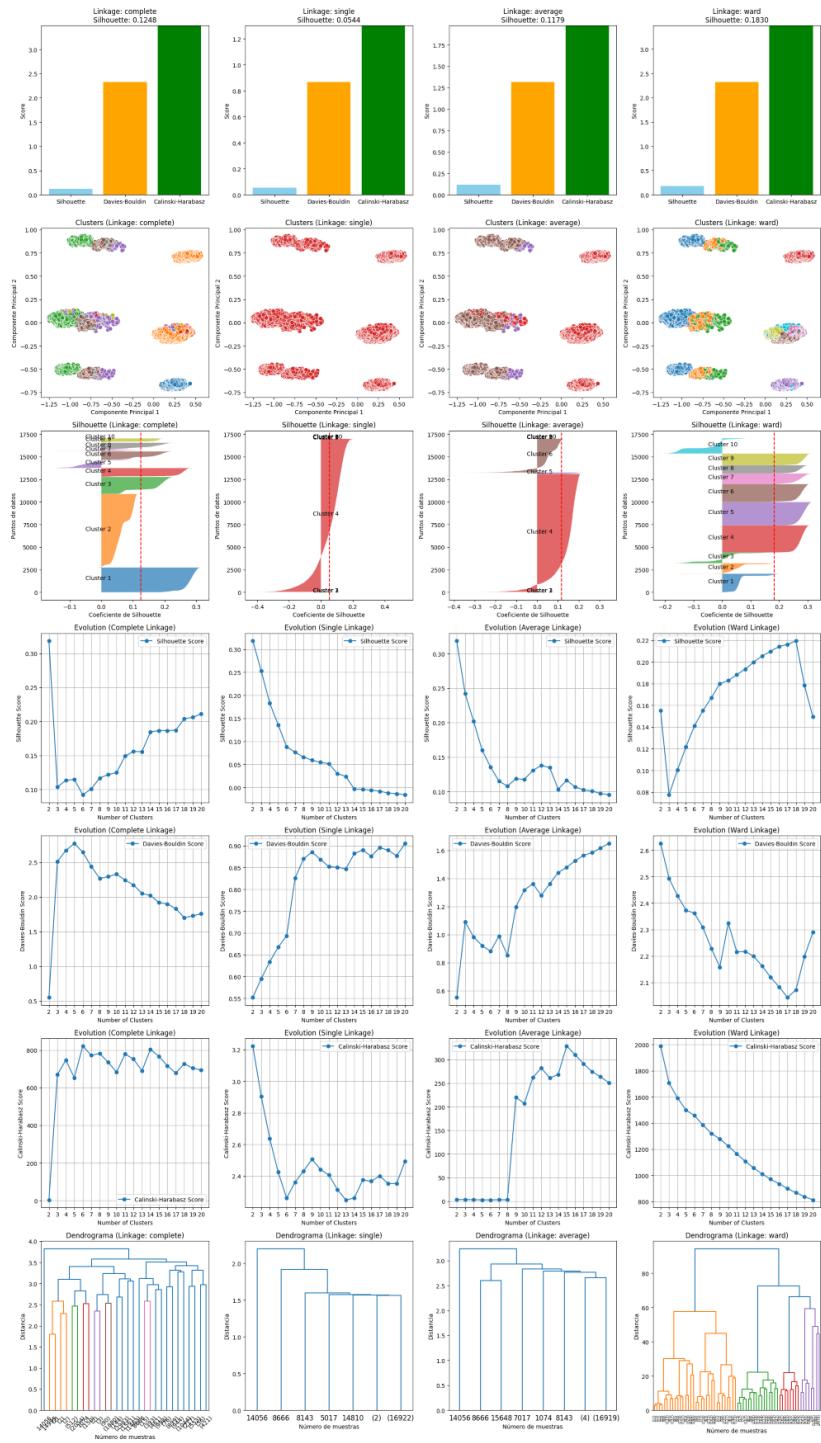
Hemos probado a usar dicho conjunto agrupado, escalado y normalizado con la distancia euclídea.

4.1.2.5.1. Datos agrupados.



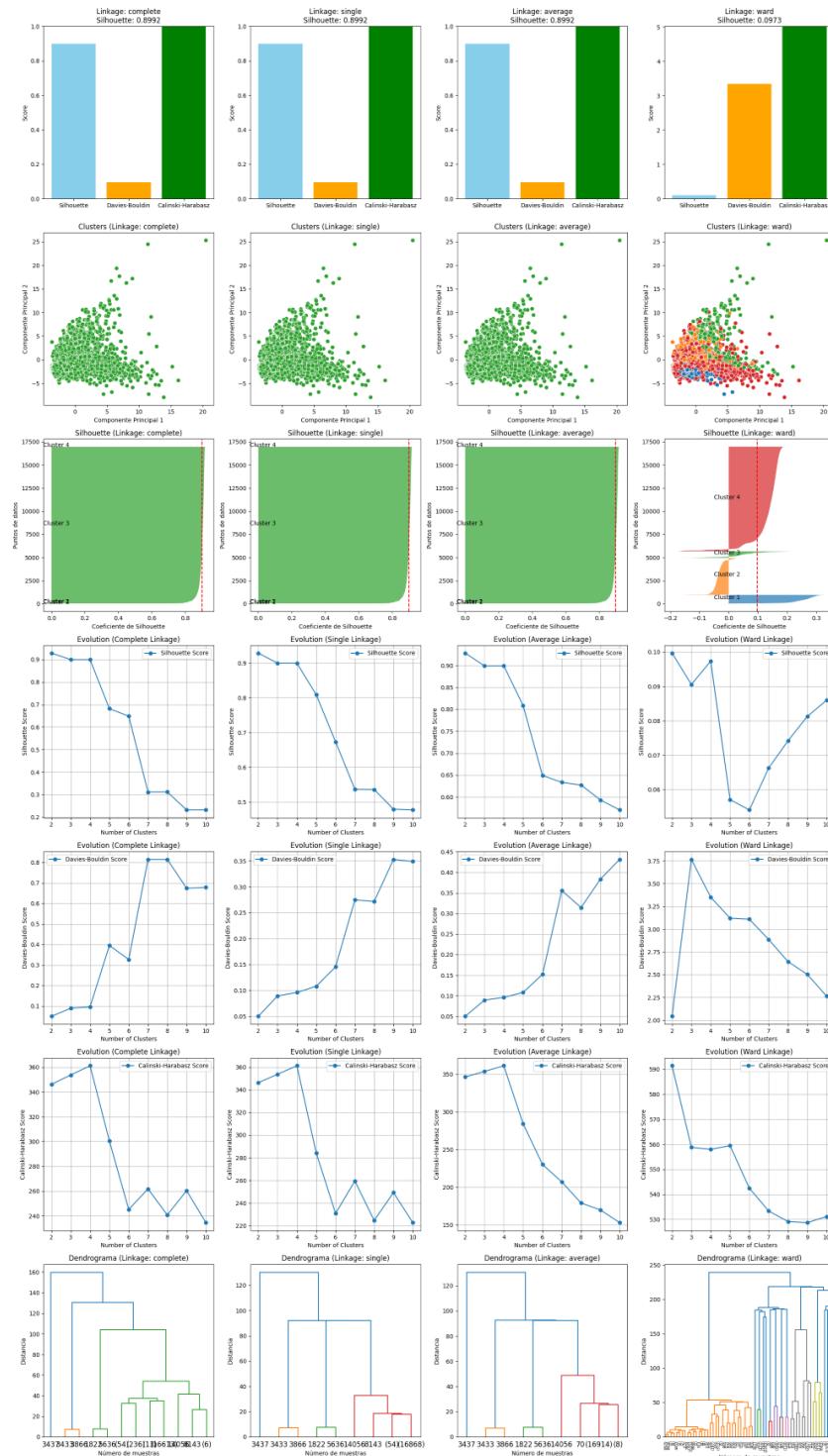
Aunque la separación en el diagrama de dispersión es muy clara, los valores de los índices no parecen buenos e indican que es mejor usar dos clústeres. Tenemos también muchos outliers y los dendrogramas realmente densos.

4.1.2.5.2. Datos normalizados.



En el caso del conjunto normalizado, las agrupaciones son muy claras y separadas, pero los resultados no son buenos. El índice de silhouette es muy bajo y el índice de Davies-Bouldin relativamente alto para todos los linkages. En caso de linkage complete y ward, los dendrogramas son densas y en otros casos las agrupaciones son débiles.

4.1.2.5.3. Datos escalados.



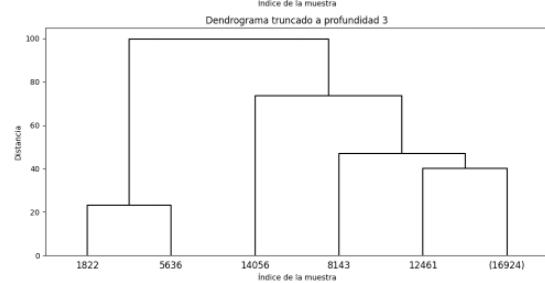
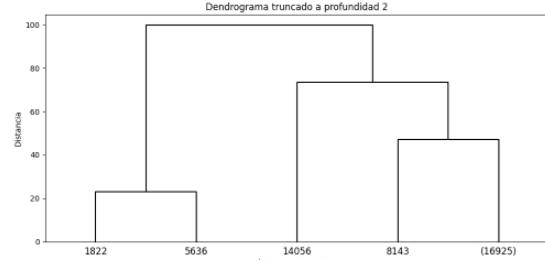
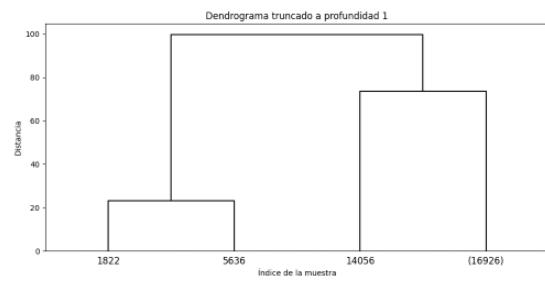
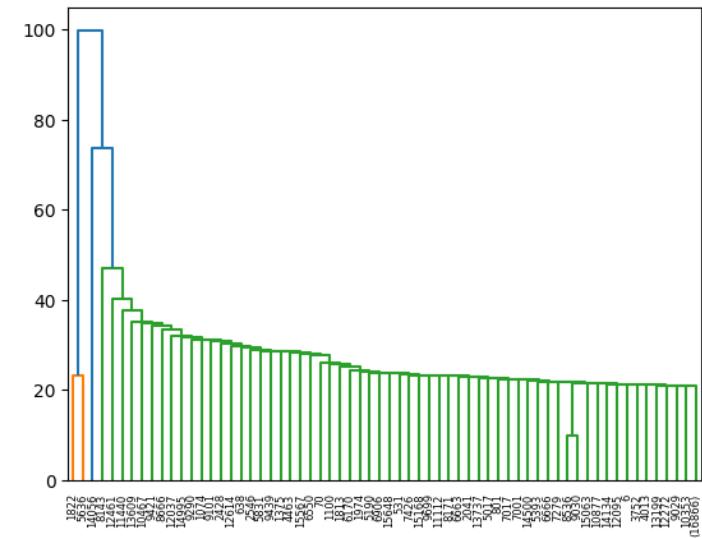
Datos escalados muestran los índices muy buenos para todas las linkages. Linkage complete nos da un dendrograma bien estructurado. El número óptimo de clusters se cambia entre 2 y 3.

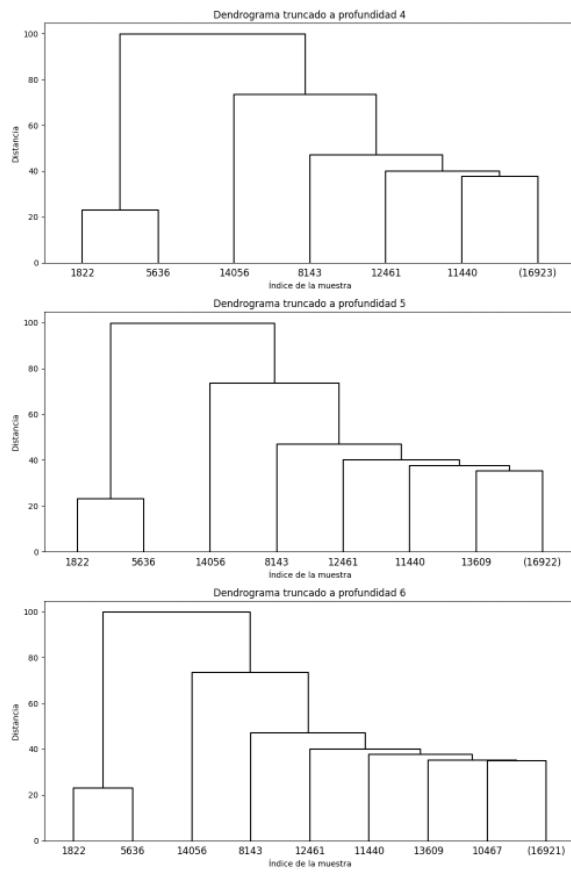
Al final hemos decidido usar el conjunto de datos *escalado** con la distancia de *city_block* y el linkage *single*.

4.1.3. Análisis de los dendrogramas.

Tenemos un pequeño problema porque single tiende a generar árboles altamente desequilibrados y nos da un "Recursion error". Esto ocurre porque el criterio se basa en la distancia mínima entre los puntos de dos clústeres, lo que puede dar lugar a muchas combinaciones en niveles bajos, provocando problemas en el cálculo del dendrograma.

Por ello vamos a truncar el dendrograma.



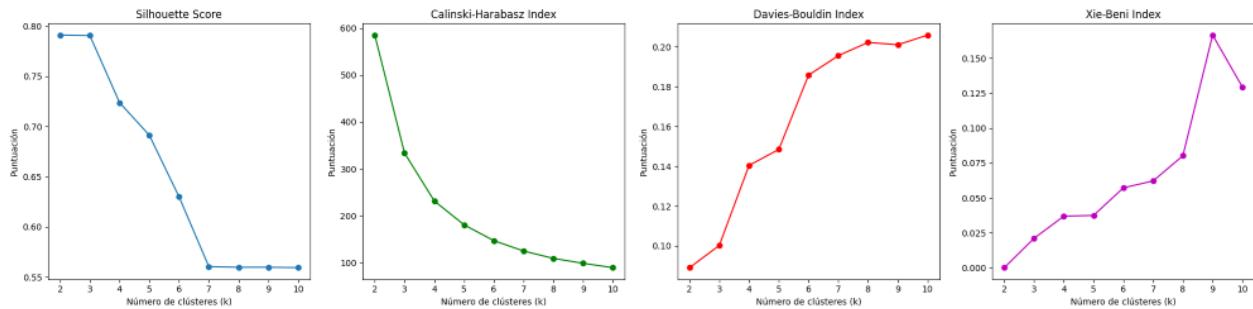


En el dendrograma, las primeras fusiones (azules) muestran grandes diferencias en las distancias de enlace, reflejando alta variabilidad y clusters iniciales alejados. A medida que avanzamos, las fusiones (verdes) tienen una reducción más gradual en altura, pero se estancan alrededor de una distancia de 30, sin grandes saltos que indiquen nuevos clusters bien definidos. Esto sugiere que los datos son homogéneos en esta etapa y que la estructura jerárquica ha alcanzado su límite, sin aportar nueva información significativa.

En general, la distribución es uniforme: las primeras fusiones son variables, mientras que las últimas son más homogéneas, especialmente en el lado derecho, donde los datos están más compactos en el espacio. Solo hay una división clara que produce un cluster bien separado a la izquierda (con baja distancia interna y mayor separación), mientras que los clusters del lado derecho son débiles, con puntos dispersos y distancias altas constantes. A simple vista podemos deducir que la mejor versión del dendrograma es a profundidad 1 o 2 (entre 4 y 5 clusters) ya que a partir de este punto, el hecho de añadir más divisiones no genera clusters relevantes o de calidad.

4.1.4. Índices de calidad.

Después de analizar todas las gráficas previamente, hemos decidido usar el dataframe scaled_df con distancia "cityblock", linkage "single". A partir de este punto solo utilizaremos estos datos.



Como podemos ver, los índices dan mejores resultados usando 2 clusters ya que Silhouette y Calinski Harabasz se maximizan y Davies Bouldin y Xie-Beni tienen valores mínimos.

Esto es interesante ya que en el análisis del dendrograma derivado de este dataframe, habíamos llegado a la conclusión de que un número óptimo de clusters era entre 4 y 5.

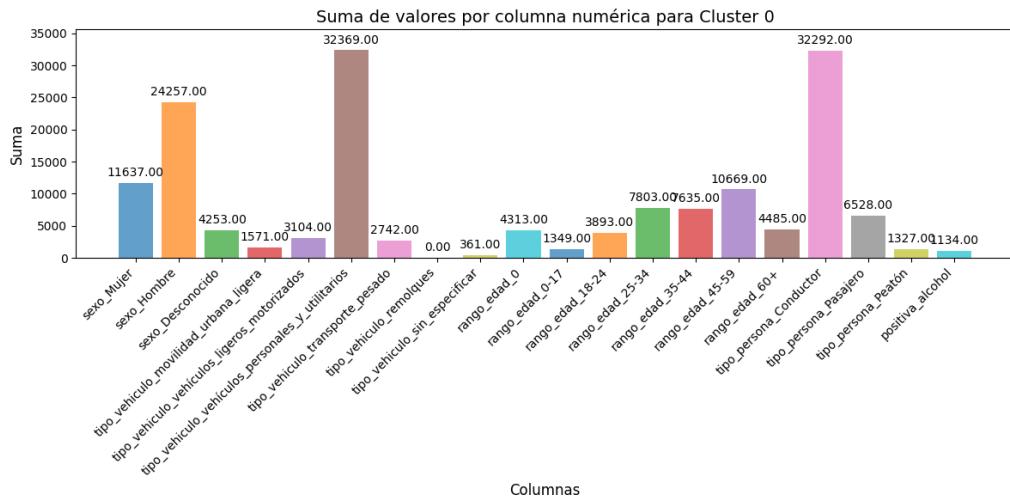
En este caso hemos decidido confiar en los resultados de los índices de calidad y estableceremos el número de clusters en 2.

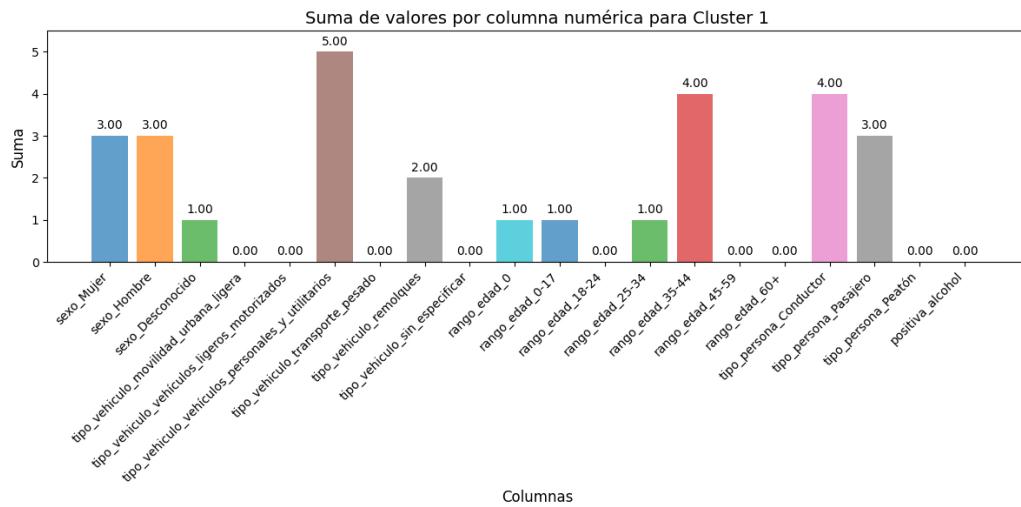
4.1.4. Análisis de clusters.

Para analizar los clusters hemos usado el Agglomerative Clustering y hemos recibido el índice de Silhouette promedio: 0.9213.

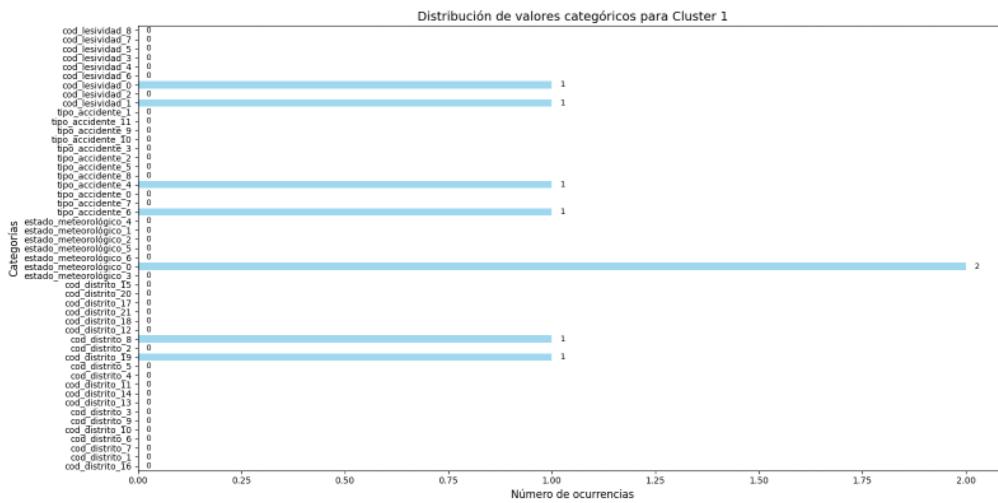
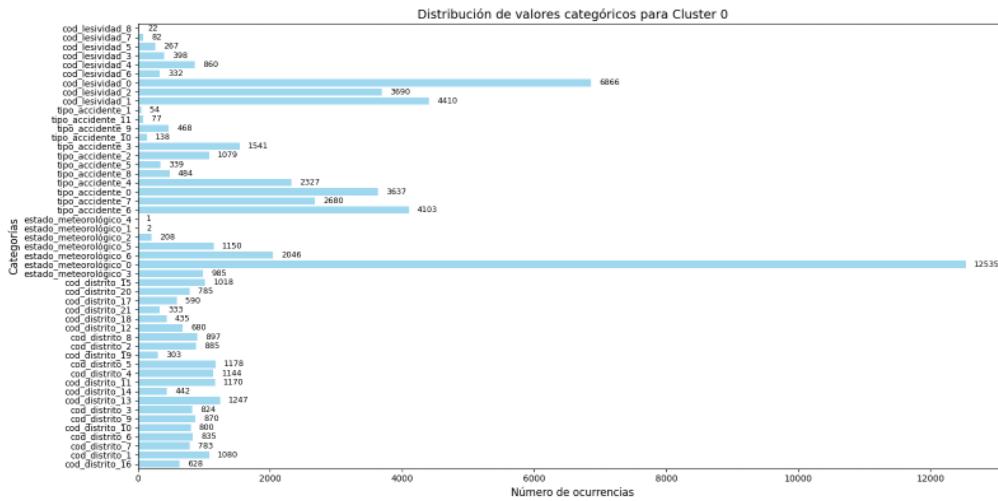
Luego visualizamos las distribuciones de las variables entre clusters.

4.1.4.1. Valores numéricos.

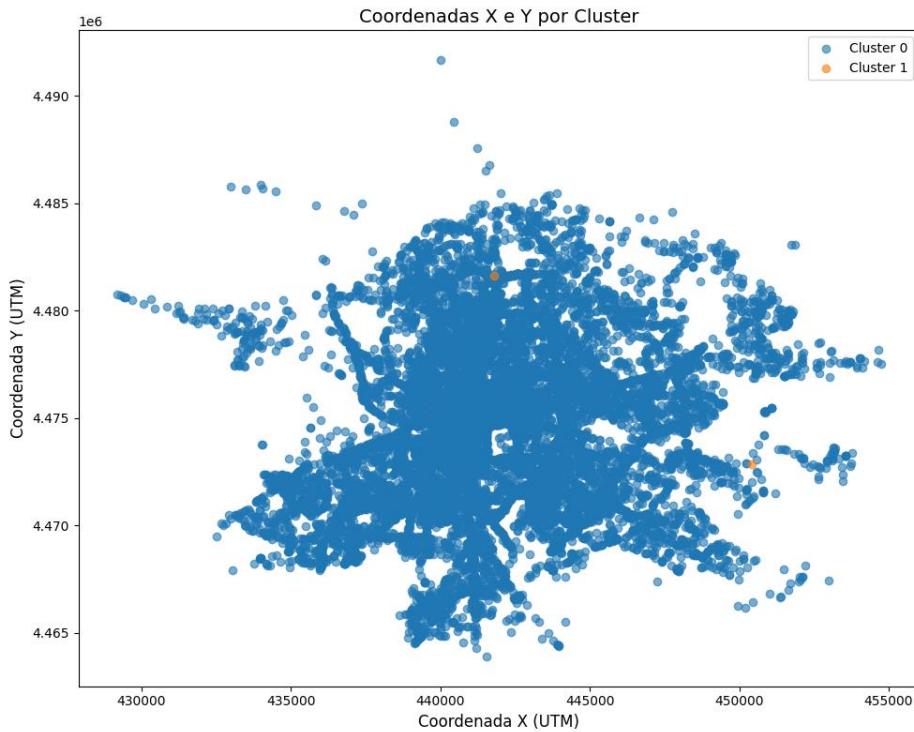




4.1.4.2. Valores categóricos.



4.1.4.3. Coordenadas.



4.1.4.4. Observación.

Pese a tener buenos resultados en los índices de calidad, la distribución de datos entre clusters es de baja calidad y no refleja ninguna estructura consistente en cuanto a la distribución de las variables.

Podemos ver que hay un cluster muy grande que abarca la mayoría de los datos y otro muy pequeño que ni si quiera tiene un conjunto de datos que se distinga del resto por alguna característica especial. Esto nos indica que la distribución de los datos no es homogénea y que los clusters no están bien definidos. Viendo las gráficas tal vez se puede deducir que hay valores que empañan las posibles agupaciones como el desbalanceo de aparición de determinadas características como por ejemplo el estado meteorológico 0 que aparece en la mayoría de los datos o la distribución de tipo de vehículo que no es homogénea.

En este caso una manera de mejorar podría ser estudiar la posibilidad de reducir el número de filas con `estado_meteorológico_0` y eliminar o ajustar las categorías con poca representación como `estado_meteorológico_4`.

4.1.4.5. Análisis de los clusters de BIRCH.

Hemos intentado analizar los clusters obtenidos de BIRCH, pero desgraciadamente no era posible.

Es imposible estudiar los clusters que genera BIRCH por lo comentado anteriormente. Con `threshold = 0.2`, el algoritmo genera 10649 clusters, lo que es inviable para su estudio.

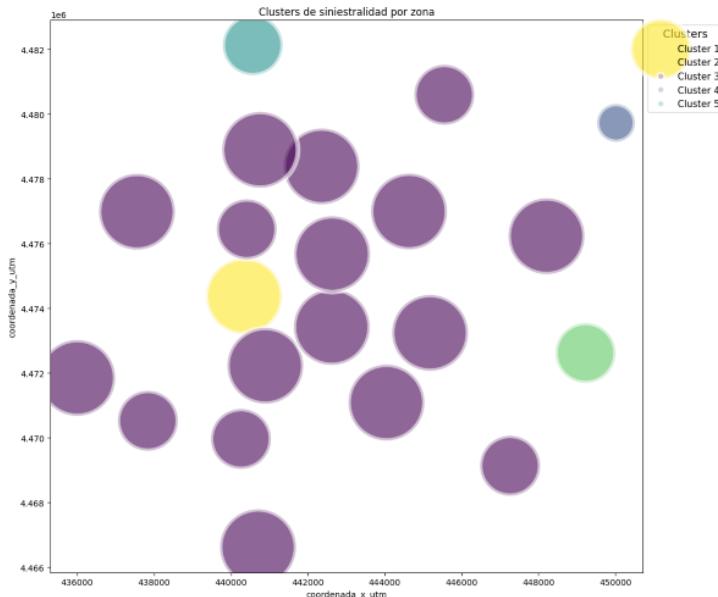
4.1.6. Zonas con mayor índice de siniestralidad para cada tipo de vehículo

4.1.4.6. Zonas con mayor índice de siniestralidad para cada tipo de vehículo.

4.1.4.6.1. Enfoque 1.

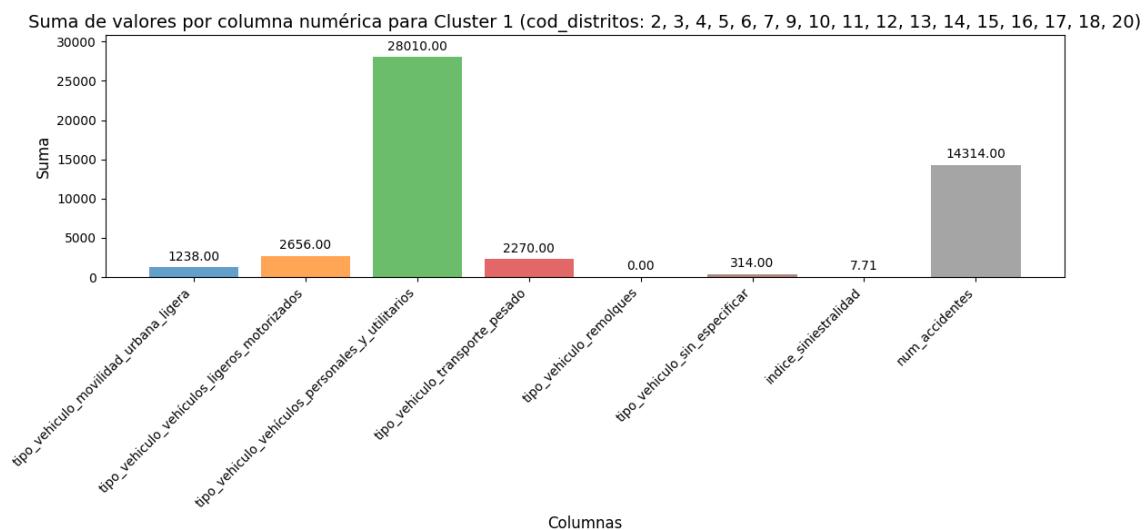
Para poder estudiar las zonas con mayor índice de siniestralidad para cada tipo de vehículo, hemos decidido modificar los pesos de las columnas de tipo de vehículo para entrenar de nuevo el algoritmo. Como índice de siniestralidad, usaremos el número de accidentes por código de distrito combinado con el código de lesividad

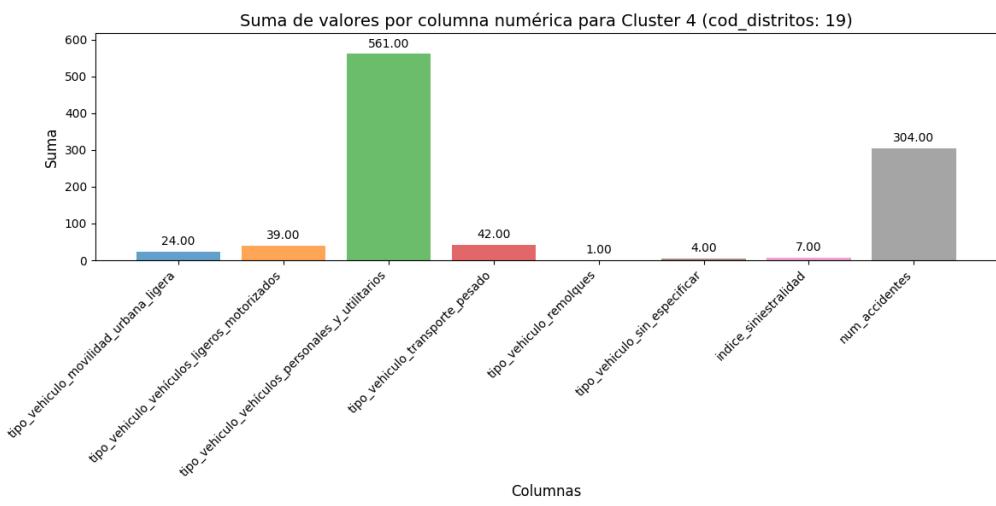
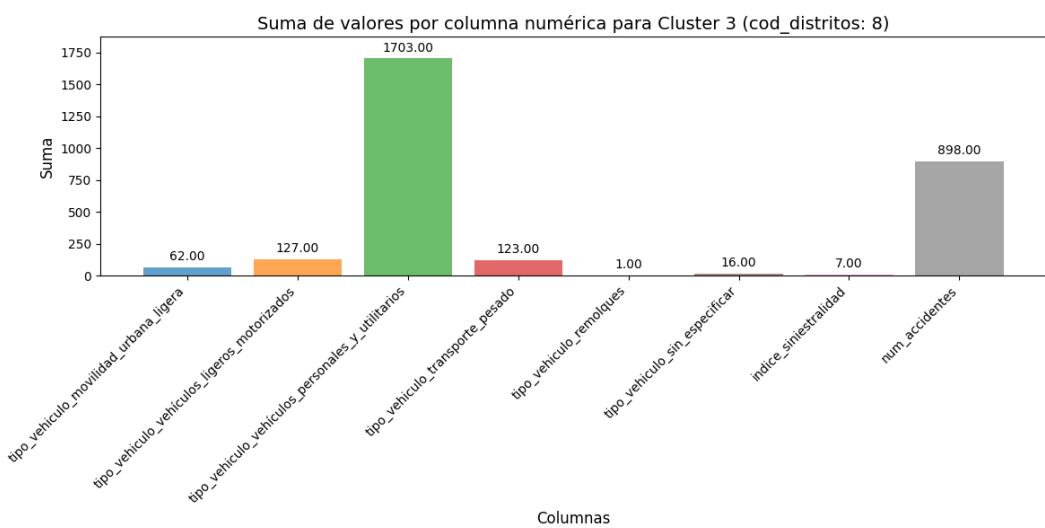
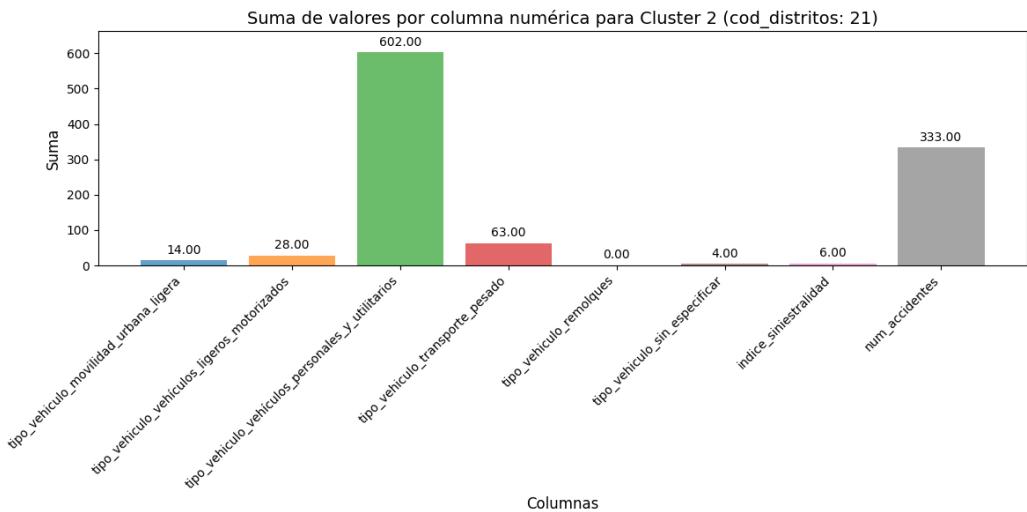
Vamos a generar un nuevo dataframe que por cada distrito tenga la cantidad de vehículos de cada tipo involucrados en esta zona, el número de accidentes y el índice de siniestralidad.

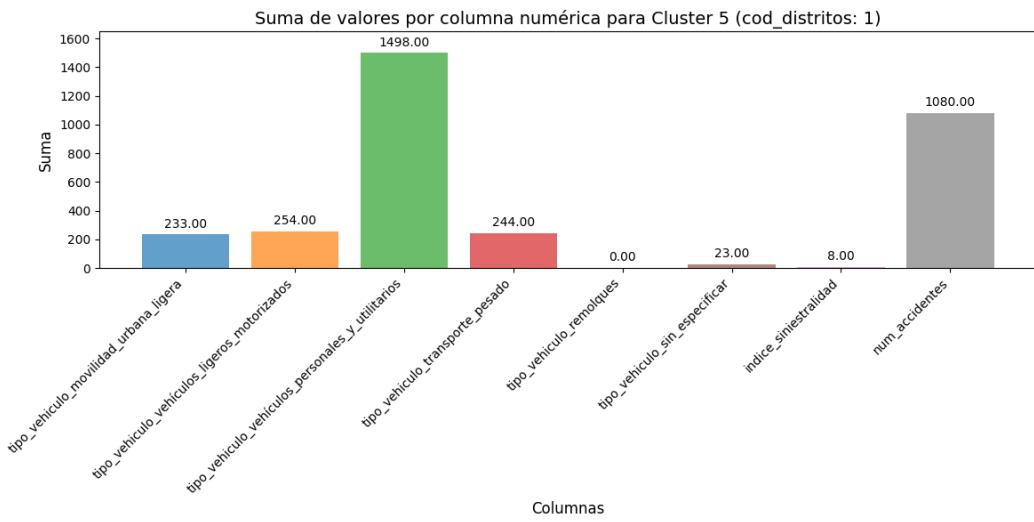


Con el código proporcionado hemos obtenido los clusters separados para los distritos de numero 21, 8, 19, 1. Sin embargo todos los distritos restantes están agrupados en el mismo cluster, que no nos parece bien.

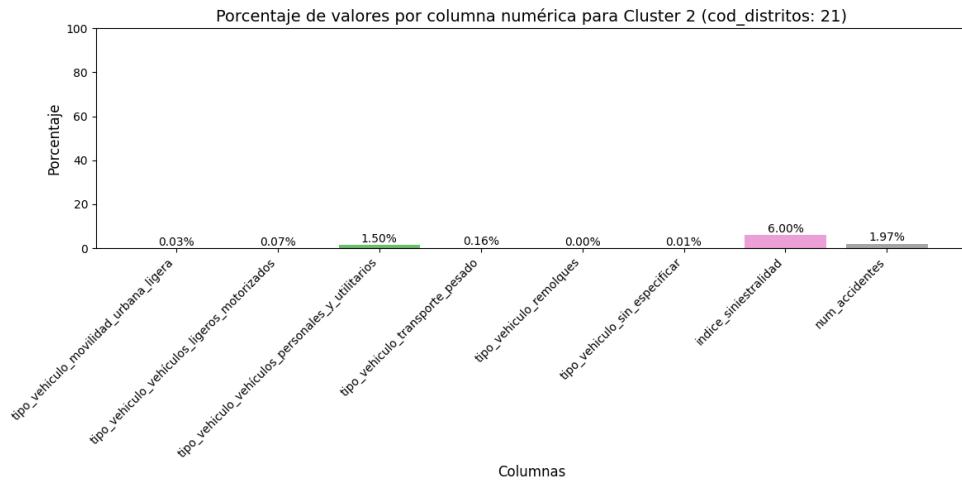
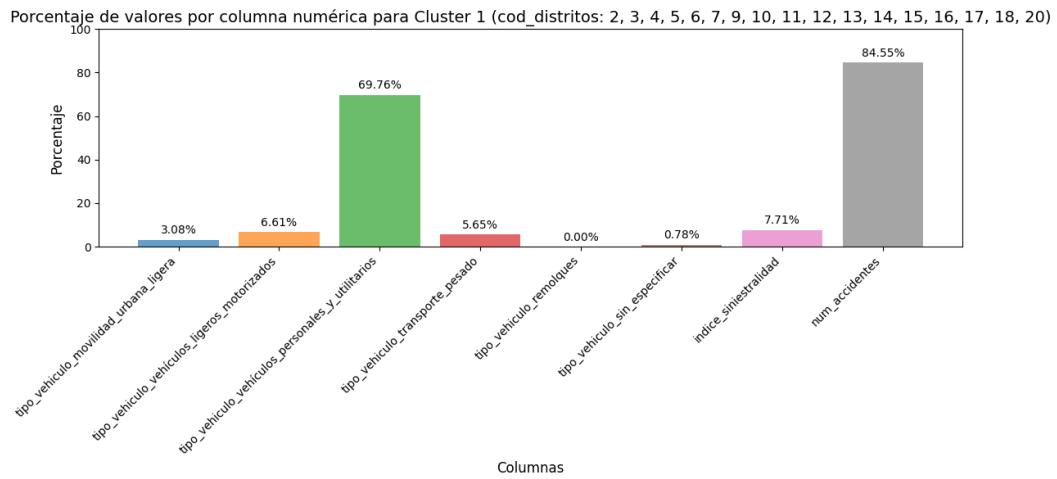
Podemos ver la distribución de los variables entre los clusters.

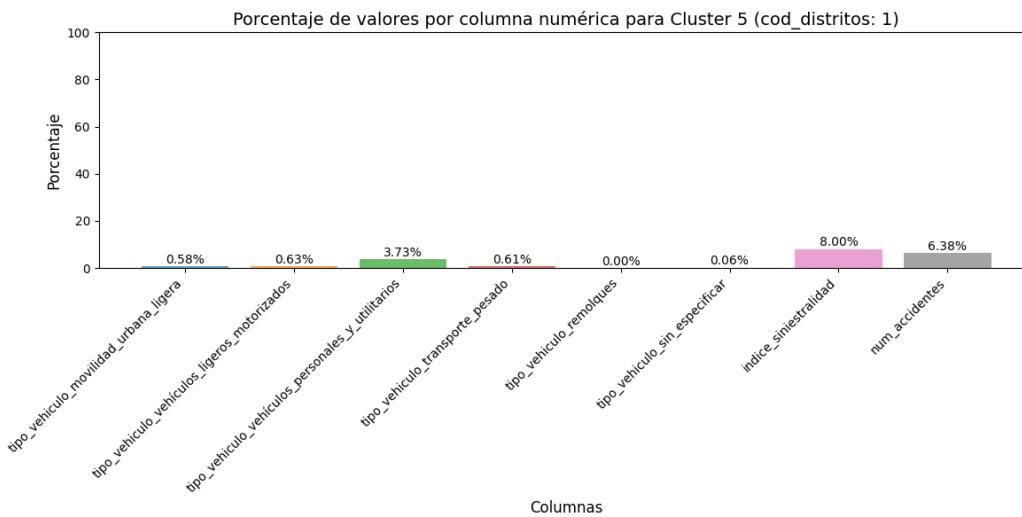
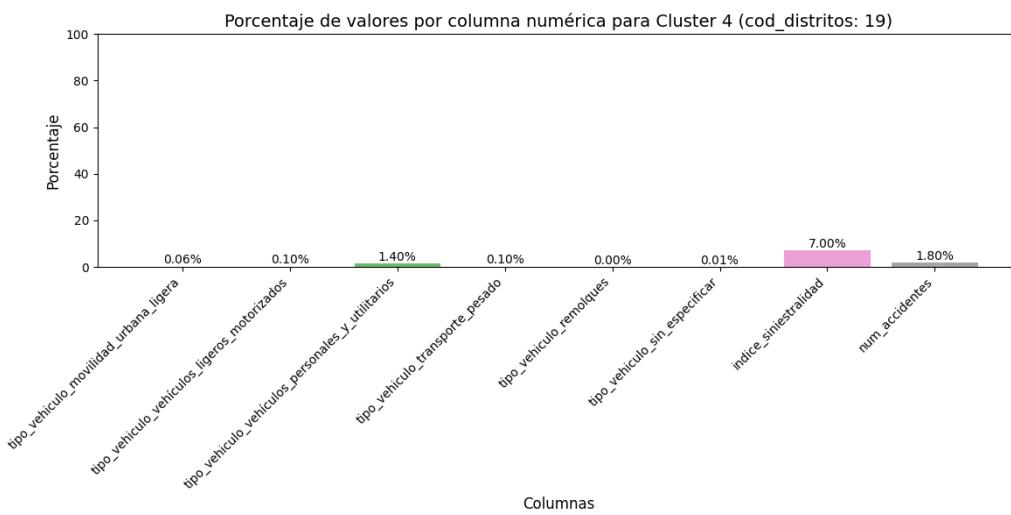
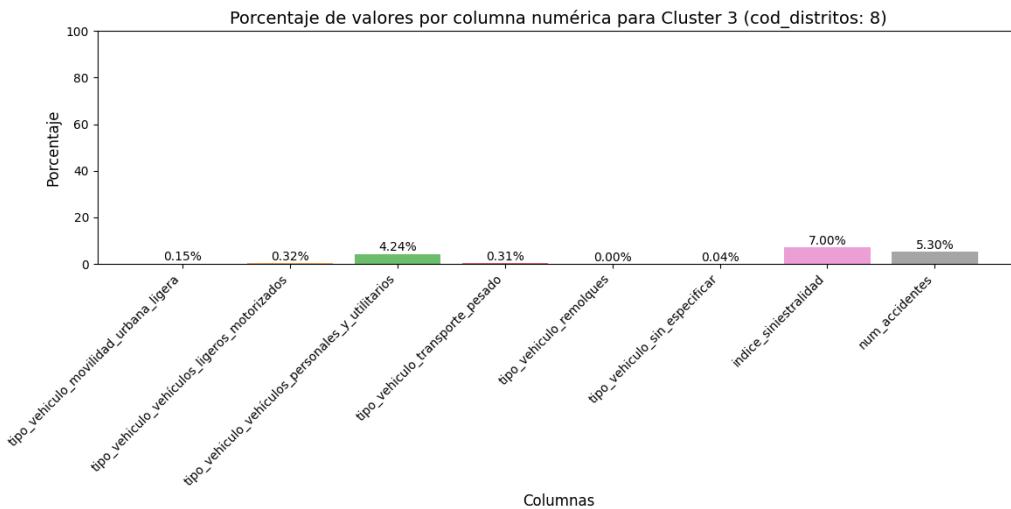






Los porcentajes tampoco se miran bien.

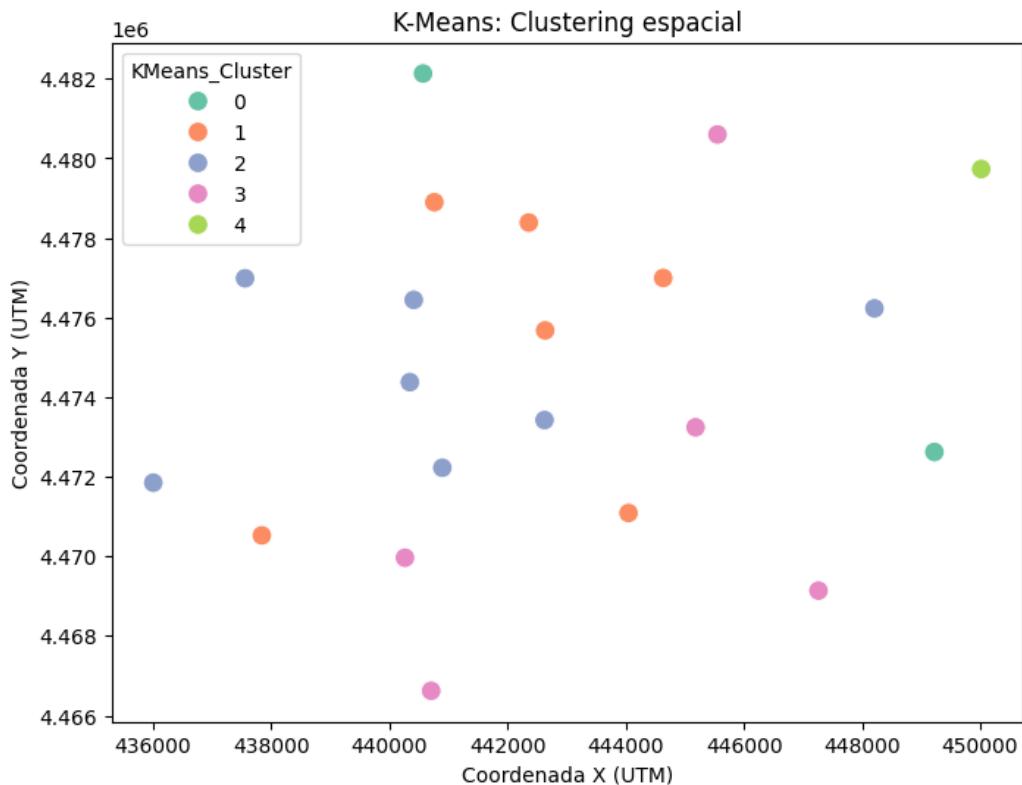




4.1.4.6.2. Enfoque 2.

Pensamos que los agrupaciones no son buenos, como no aportan ninguna opción. Entonces hemos probado otro enfoque.

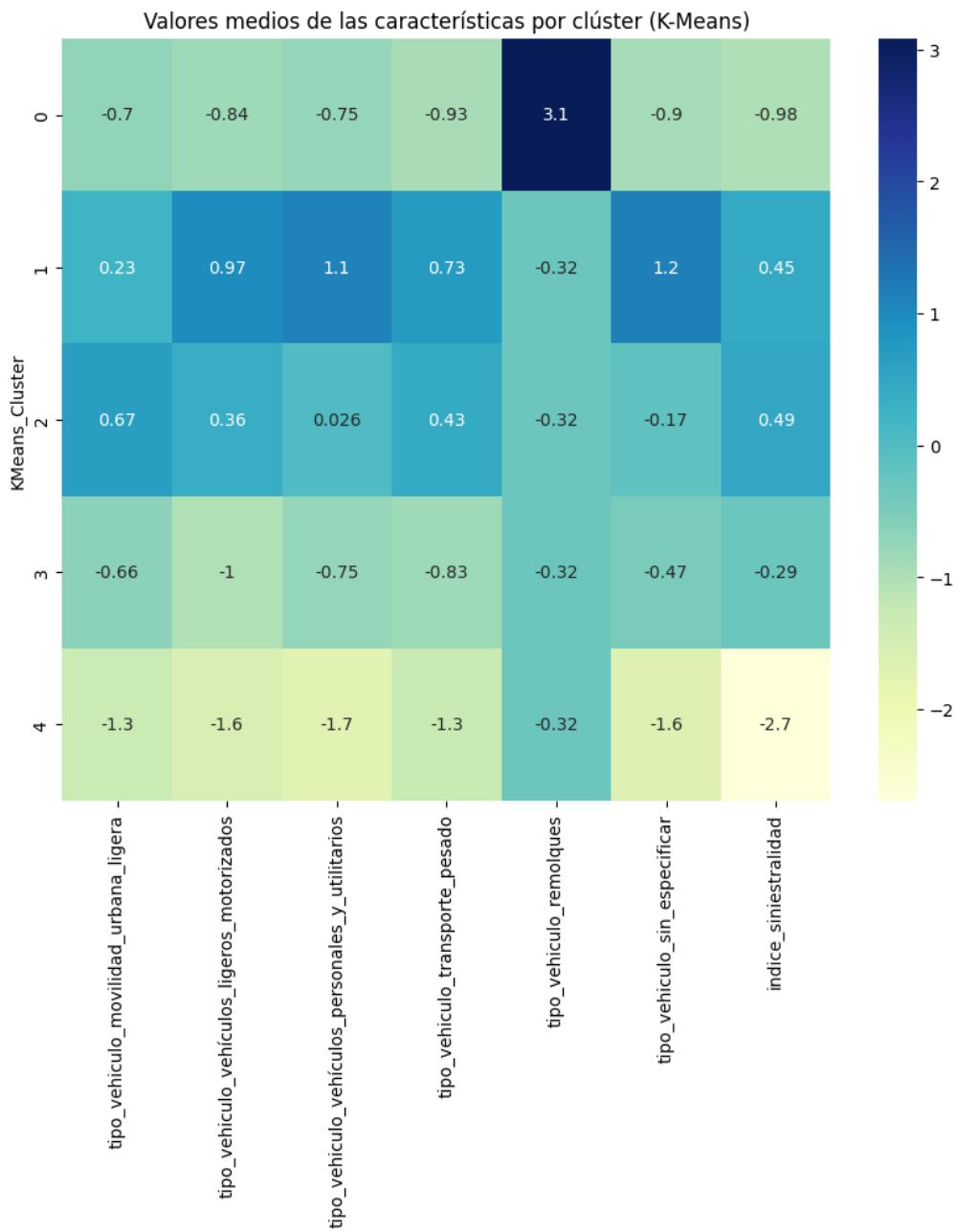
Solo hemos seleccionado las columnas relevantes, es decir esas que aportan información que nos interesa (tipo_vehiculo, índice_sinestralidad y las coordenadas). Luego hemos normalizado los datos y aplicamos el algoritmo de k-means. Este método agrupa los datos en clústeres esféricos basados en la similitud de las características seleccionadas.



Hemos obtenido una grafica parecida a la mapa de Madrid.

- Algunos clústeres (por ejemplo, el clúster 4) están más dispersos, lo que indica que los distritos asociados a este grupo no están geográficamente cercanos.
- Por otro lado, otros clústeres, como el clúster 1 o el clúster 2, parecen concentrarse en regiones más específicas.
- Es posible que los clústeres reflejen patrones espaciales relacionados con factores como el tráfico, densidad de población, o características específicas de los vehículos involucrados.

Hemos también obtenido un heatmap de las variables en cada cluster.



La grafica muestra los valores promedio de las características normalizadas para cada clúster. Los valores están codificados por colores, donde:

- Valores cercanos a 0 indican características promedio.
- Valores positivos altos indican que la característica es significativamente alta en ese clúster.
- Valores negativos bajos indican que la característica es significativamente baja.

Observaciones por cluster:

- **Cluster 0:**
 - Destaca por un valor muy alto en tipo_vehiculo_remolques (+3.1).
 - En el resto de características, tiene valores negativos, lo que sugiere que es un clúster asociado principalmente a accidentes con remolques.
- **Cluster 1:**
 - Tiene valores altos en tipo_vehiculo_movilidad_urbana_ligera, tipo_vehiculo_vehículos_ligeros_motorizados y tipo_vehiculo_vehículos_personales_y_utilitarios.
 - Este clúster parece estar relacionado con accidentes que involucran vehículos ligeros y movilidad urbana.
- **Cluster 2:**
 - Valores moderados en todas las características, lo que sugiere que es un clúster con un perfil balanceado.
 - Este clúster podría representar distritos con un nivel promedio de siniestralidad y participación de distintos tipos de vehículos.
- **Cluster 3:**
 - Valores bajos en todas las características. Este clúster podría representar distritos con un índice de siniestralidad bajo y poca participación de vehículos específicos.
- **Cluster 4:**
 - Valores significativamente bajos en todas las características, especialmente en indice_siniestralidad (-2.7).
 - Este clúster parece representar distritos con muy baja siniestralidad.

Relación entre los clusters:

- Los clústeres 0 y 1 parecen ser los más especializados:
 - El clúster 0 está dominado por accidentes con remolques.
 - El clúster 1 se asocia a vehículos ligeros.
- Los clústeres 3 y 4 representan distritos con menor nivel de siniestralidad.
- El clúster 2 actúa como un clúster intermedio con características más equilibradas.

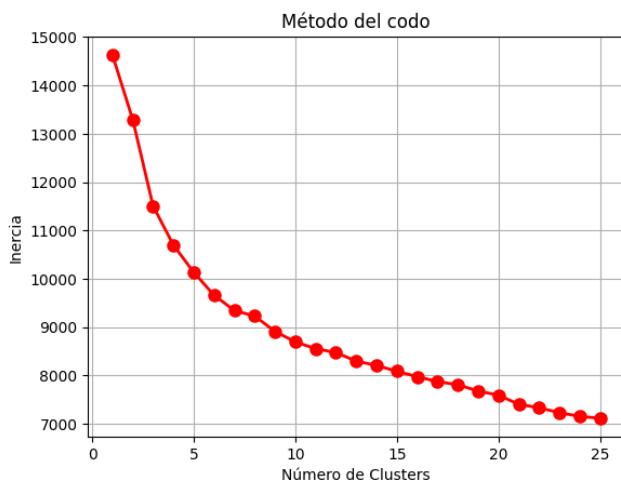
4.2. Clustering particional.

4.2.1. Preprocesamiento.

El "preprocesamiento necesario para poder aplicar algoritmos de clustering particional" es básicamente la normalización de los datos ya que k-means se basa en distancias. En este caso, ya los hemos normalizado con MinMax scaler en el notebook anterior y por ello vamos a usar `normalized_df` en este apartado.

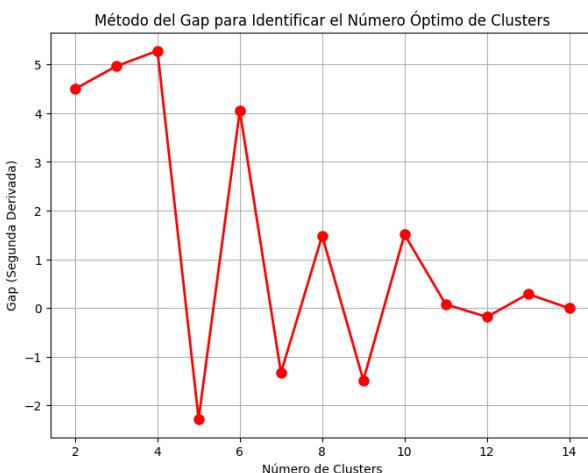
4.2.2. Número más adecuado de clusters.

4.2.2.1. Método del codo.



Esta gráfica da lugar a dudas sobre el número óptimo de clusters. Se observa un cambio pronunciado en la pendiente entre 3 y 5 clusters, a partir de 5 clusters la inercia disminuye más lentamente, indicando una posible saturación en la mejora de la partición. El número óptimo de clusters sugerido por este método es 4 o 5, ya que estos puntos reflejan el equilibrio entre una mejor separación y evitar el exceso de clusters.

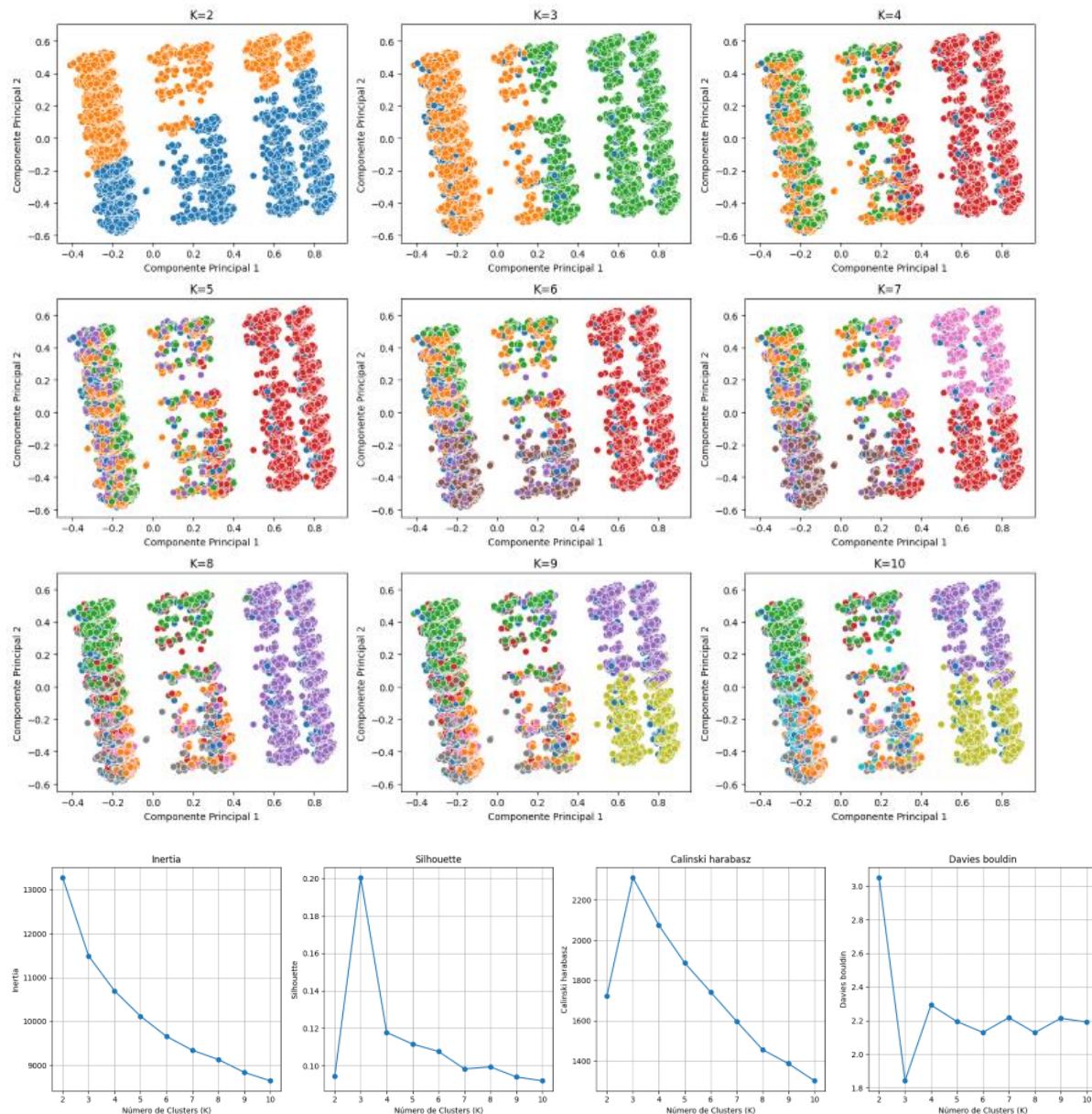
4.2.2.2. Método gap.



Este método calcula la brecha entre la variabilidad observada en los datos originales y una referencia aleatoria, utilizando la segunda derivada para identificar picos. Los picos más altos indican el número óptimo de clusters. Hay un pico significativo en el valor de gap en 4 clusters. También hay fluctuaciones en valores secundarios para 6 y 10 clusters, pero estos son menos pronunciados.

Con la información de Elbow y Gap, podemos decir que el número óptimo de clusters sugerido por es claramente 4.

4.2.3. Calidad del clustering con diferentes valores de k.



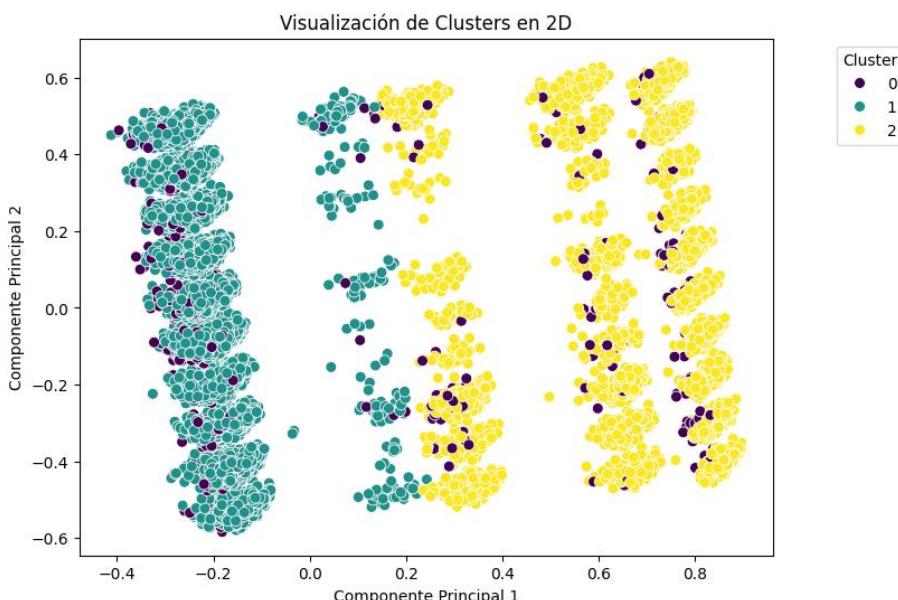
Las gráficas muestran cómo evolucionan los clusters con k:

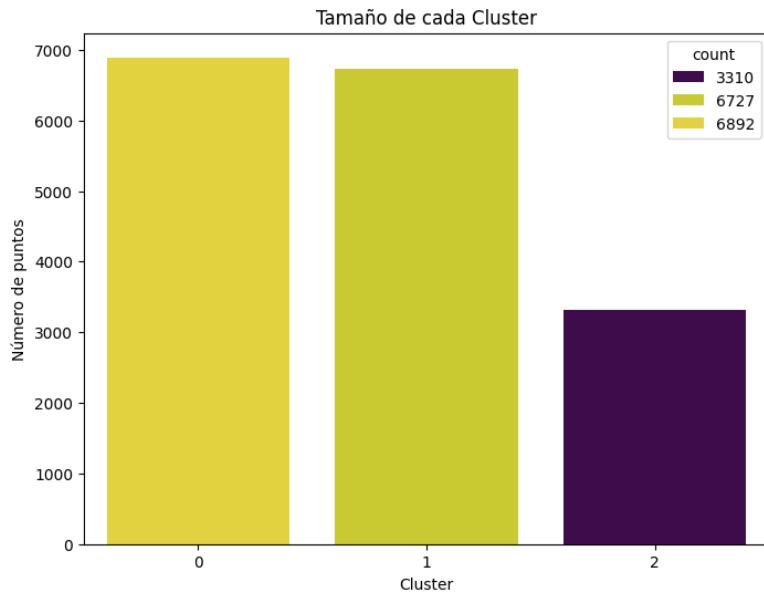
- En k=2, los clusters están bien definidos pero no parecen adaptarse adecuadamente a la naturaleza del dataframe.
- En k=3, se observan grupos mejor adaptados, pero con muchos puntos que permanecen no son clasificados adecuadamente.
- En k=4, los clusters son menos claros y se introduce más ruido del esperado.
- Para k>4, los clusters comienzan a dividirse en demasiados subgrupos poco definidos, introduciendo ruido y redundancia.

De los índices podemos observar lo siguiente:

- La inercia disminuye a medida que aumenta k, lo cual es esperado porque al aumentar el número de clusters, los puntos están más cerca de sus centroides. Entre k=3 y k=4, la disminución de la inercia es significativa (11500 a 10800). Sin embargo, después de k=4, la disminución es menor, lo que confirma que k=4 es un buen punto de equilibrio pero no demiente que k=3 también lo sea.
- Se ve una clara predominancia en k=3 y una mejora en el índice de silhouette. Esto nos lleva a replantearnos el número óptimo de clusters.
- Como k=3 tiene el valor más alto de Calinski, reforzamos la idea de cambiar el número óptimo de clusters. Es un valor bastante alto y supuestamente sugiere que los clusters son densos y bien separados. No obstante no podemos obviar la mala puntuación de silhouette.
- El índice Davies-Bouldin sugiere que k=4 no es ideal en términos de compacidad, por ello y apoyando en el resto de métricas, podemos concluir que k=3 es la mejor elección.

4.2.4. Viajeros incluidos en cada cluster.

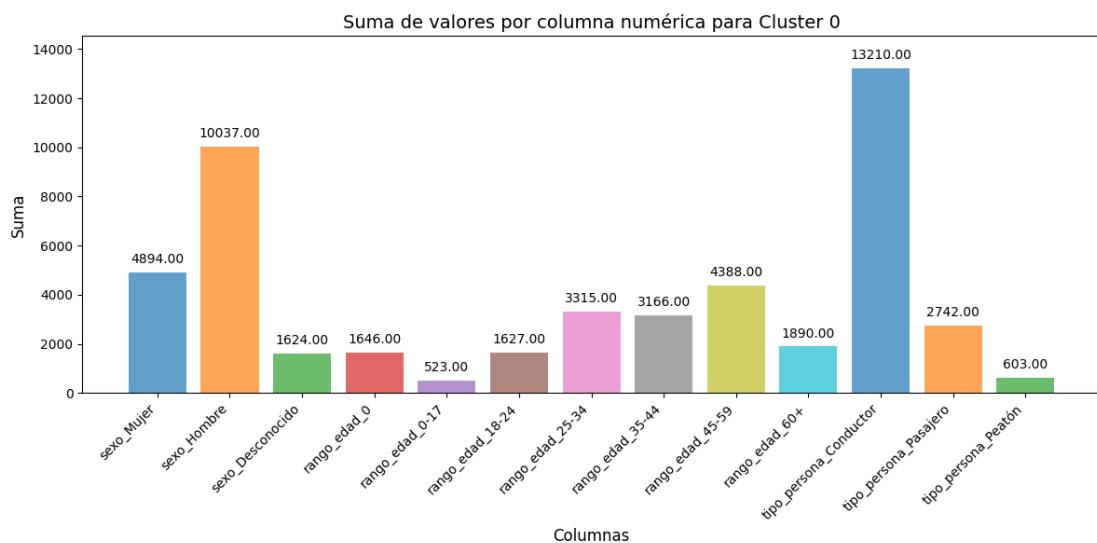


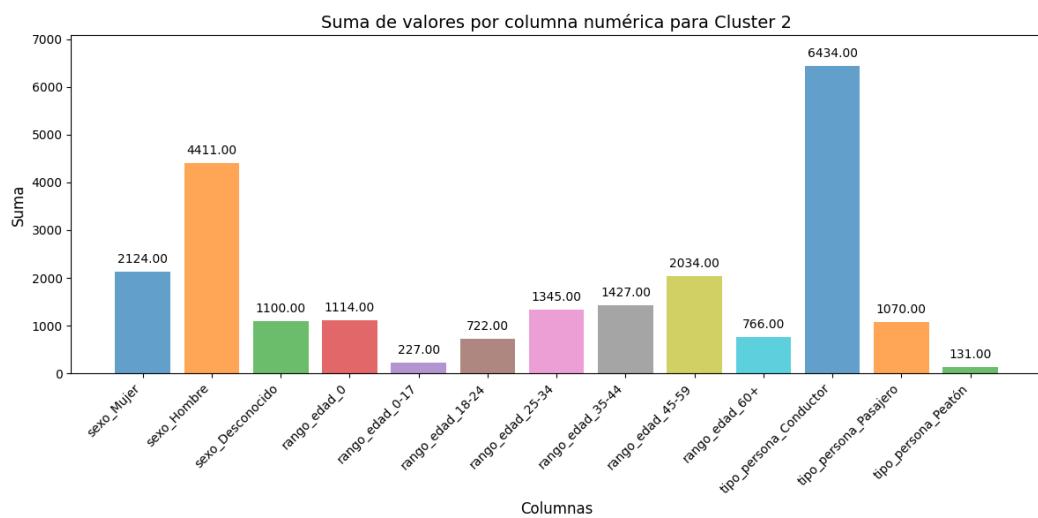
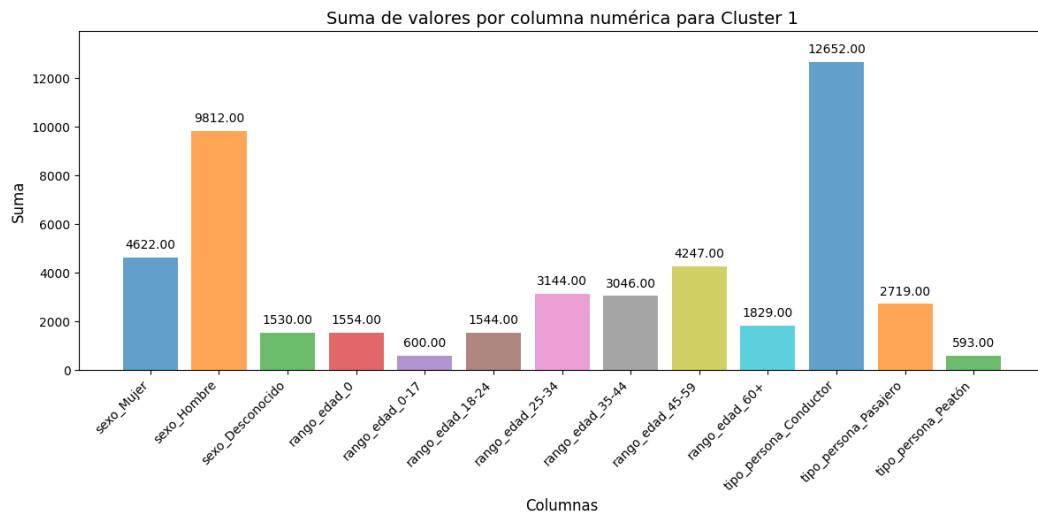


Clusters de tamaños desbalanceados podría ser un signo de agrupaciones desequilibradas o incluso de ruido en los datos. En este caso el cluster 2 es proporcionalmente más pequeño que los otros dos clusters.

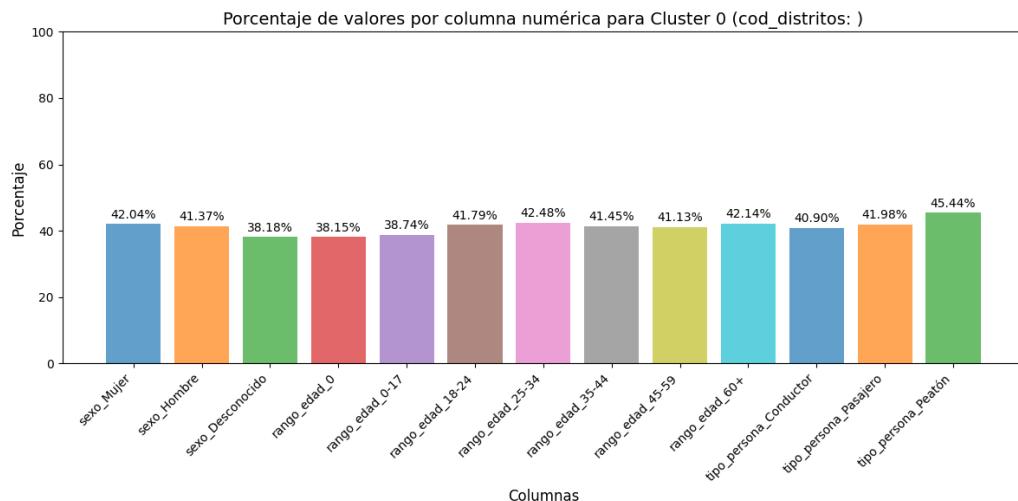
Como vamos a analizar a los viajeros, nos interesan solo las columnas de sexo, tipo de persona y edad.

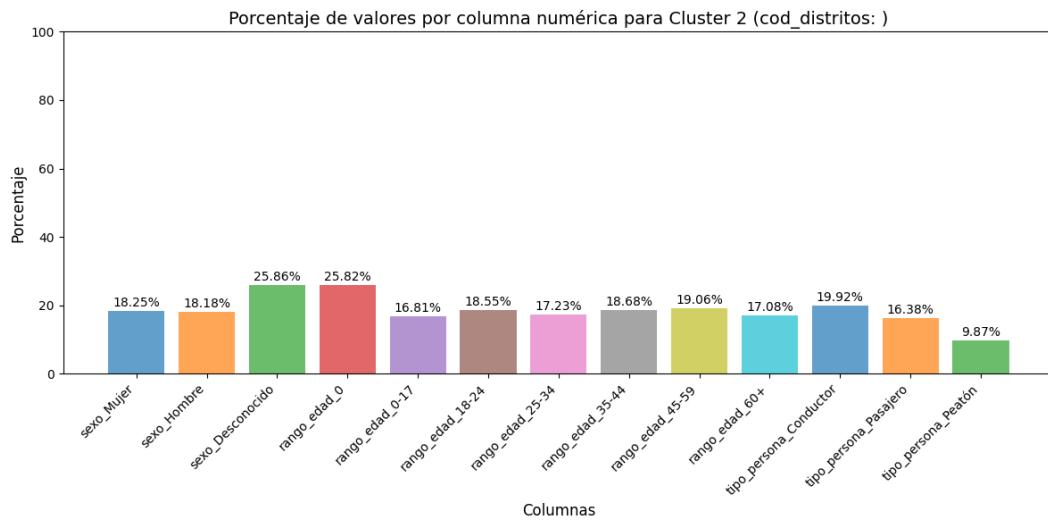
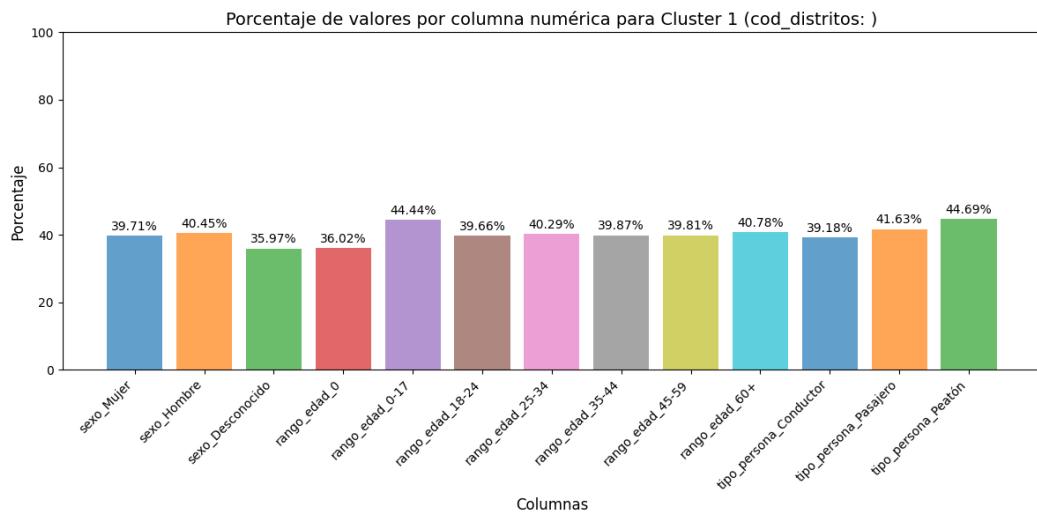
Al principio vamos a visualizar la suma de valores por columnas numéricas para cada cluster.





Y luego el porcentaje de valores por columnas numéricas para cada cluster.





Las gráficas de barras muestran la distribución de las variables categóricas en cada cluster. Se observa que todos los clusters tienen una distribución demasiado similar y no se distingue un patrón claro que los diferencie. Esto sugiere que los clusters no están bien definidos y que la segmentación no es adecuada o que los datos están demasiado desbalanceados para las características de las columnas.

4.2.5. Comparación de los resultados.

4.2.5.1. Comparación general.

Desgraciadamente con ambos algoritmos nos han salido clusters demasiado generalistas y poco específicos. En el caso de k-means hemos obtenido índices mucho más bajos en silhouette pese a estar utilizando un dataframe más cercano a las distribuciones esféricas que requiere k-means. En el caso de clustering jerárquico, hemos obtenido clusters con tamaños mucho más desbalanceados pero igualmente con una distribución de variables categóricas similar entre ellos.

Por otro lado, BIRCH nos llegó a dar buenos valores para los índices pero nos generaba un número muy elevado de clustes, lo que nos impedía poder analizarlos de manera individual.

4.2.5.2. Ventajas y desventajas de ambos algoritmos.

La primera diferencia que podemos apreciar es que el clustering jerárquico no necesita un número de clusters como parámetro en cambio, el k-means sí. Al trabajar con diferentes linkages y distancias hemos podido ver que el jerárquico trabaja mejor con datos circulares y agrupados naturalmente que con datos más densos y menos dispersos. Los diferentes parámetros que acepta, da la posibilidad de ajustar el clustering a los datos. Además una ventaja es que los resultados son reproducibles y no dependen de un random_state para obtener los mismos resultados. Como desventaja, el jerárquico es más lento y los dendrogramas que genera pueden llegar a ser difíciles de interpretar, sobretodo cuando las agrupaciones son complejas.

Por otro lado, k-means es más rápido y tiene un funcionamiento más dinámico por su capacidad de ajustarse iterativamente. Está recomendado para datasets más grandes de los que el jerárquico es capaz de procesar. Sin embargo, necesita que los datos estén normalizados, que se establezca el número de clusters de antemano, asume que los clusters son esféricos y es muy sensible a outliers y al ruido.

En resumen, para datos pequeños y o de formas complejas, el jerárquico es más recomendable. Para datos más grandes pero casi obligatoriamente esféricos, el k-means es más adecuado.

4.3. Algoritmos de densidad.

4.3.1. Preprocesamiento.

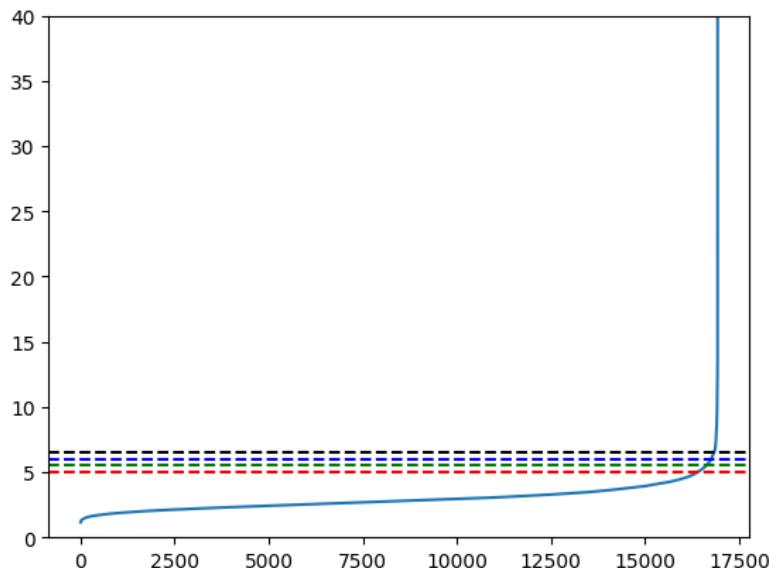
Creemos que no es necesario aplicar más preprocesamiento que para las otras partes de esta tarea.

Al principio, resolvimos esta tarea utilizando los datos normalizados, ya que son más recomendables para los algoritmos de densidad. Sin embargo, tras utilizar el mismo código para los datos normalizados, decidimos utilizar estos últimos, ya que dan resultados ligeramente mejores que los normalizados.

4.3.2. Buscando el épsilon y número mínimo de puntos.

4.3.2.1. Épsilon.

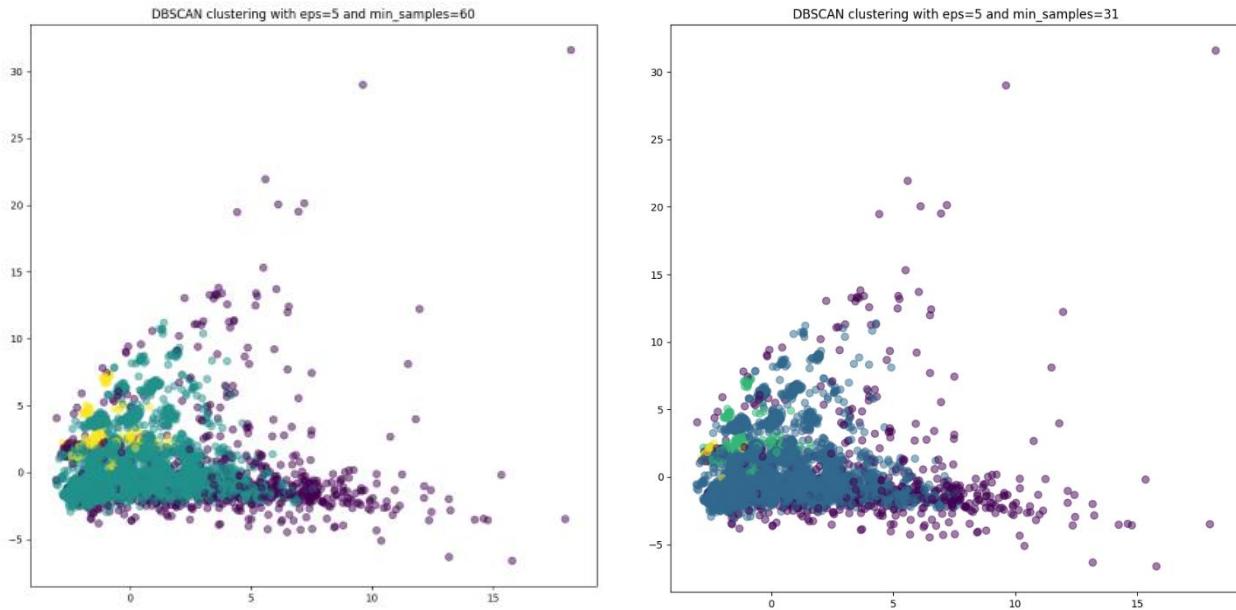
En primer lugar, trazamos la distancia a los vecinos más próximos, para establecer el valor de épsilon. Vamos a utilizar 60 vecinos más cercanos, ya que es el doble del número de características que tenemos en nuestro marco de datos.



Podemos ver en el gráfico que lo mejor sería utilizar 5 como valor épsilon, aunque también podemos probar con otros que están marcados en el gráfico.

4.3.2.2. MinPts.

Como MinPts podemos probar dos métodos recomendados - 2^*dimensión o dimensión + 1. Para ambos usaremos el primer épsilon elegido de valor 0.5.



La puntuación de silueta llega a las 0.224 para el MinPts=60 y a las 0.2270 para el MinPts=31. Los valores son muy similares. Podemos ver que la primera da una puntuación de siluetas más baja y un menor número de conglomerados (3), mientras que la segunda da una puntuación de siluetas más alta y un mayor número de conglomerados (4).

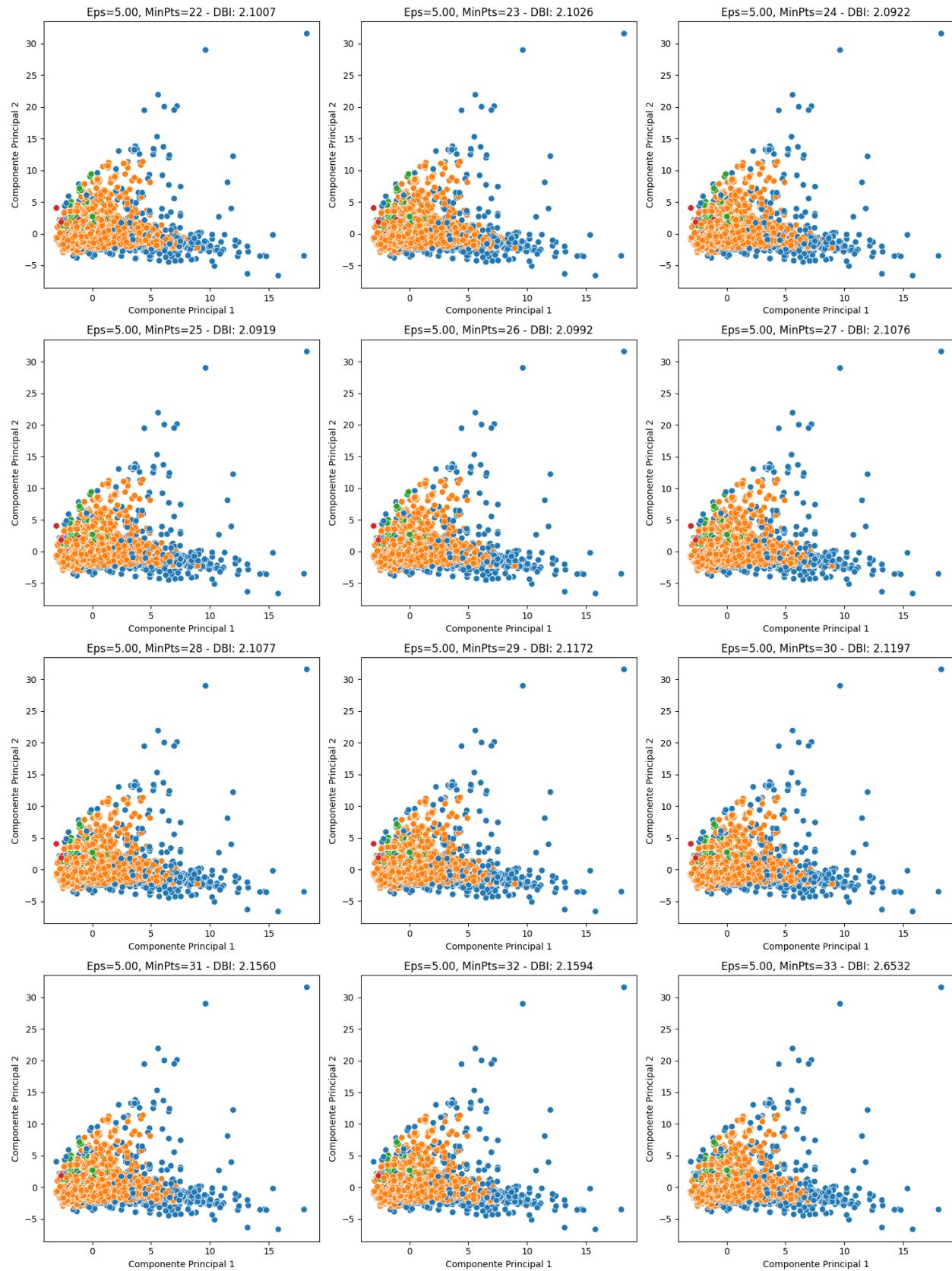
Para obtener el mejor resultado, analizaremos varias configuraciones de épsilon y MinPts. Las evaluaremos utilizando el índice de Davies-Bouldin, ya que es el recomendado para los algoritmos de densidad.

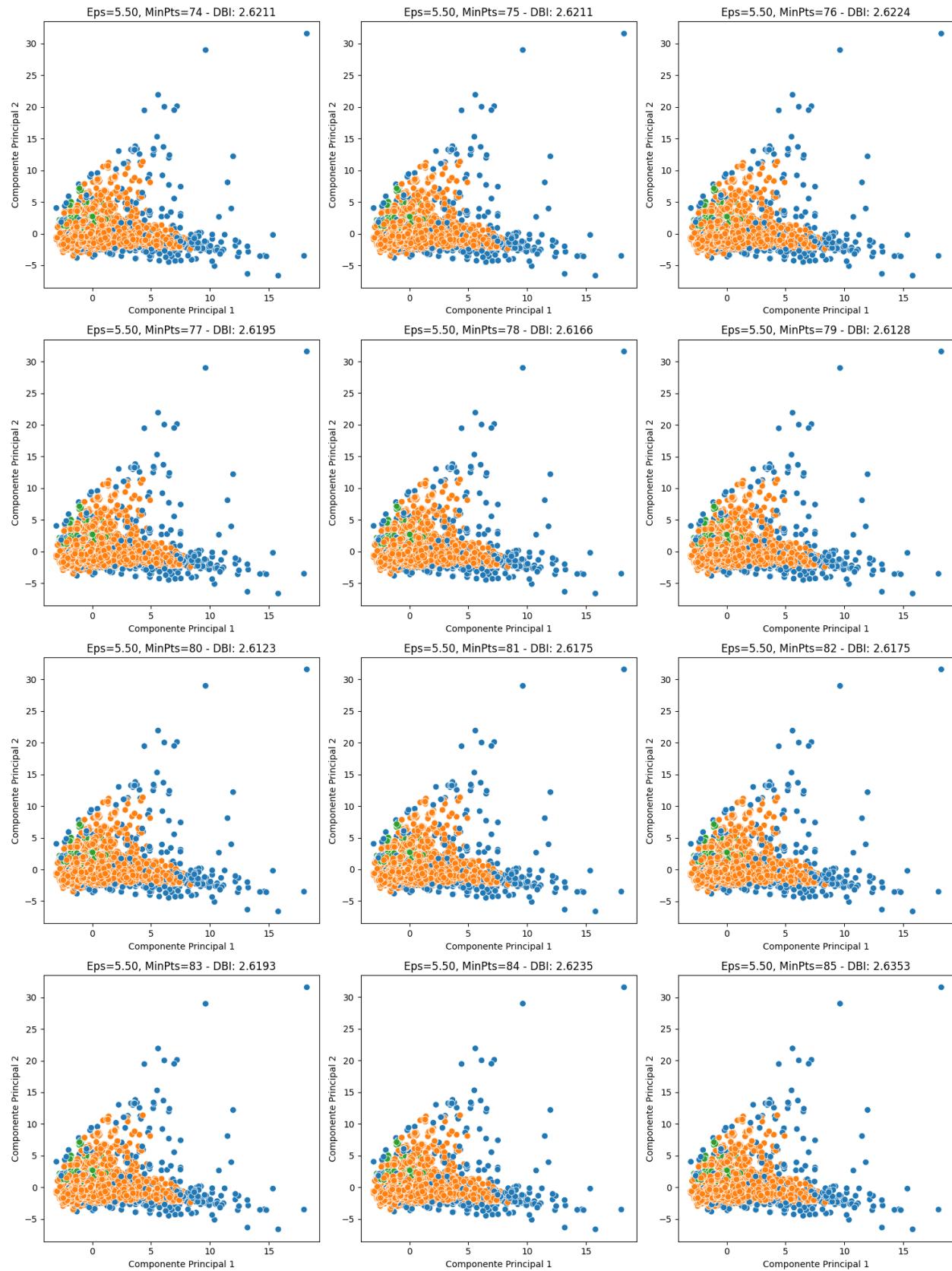
4.3.3. Calidad de clustering con diferentes valores de épsilon y MinPts.

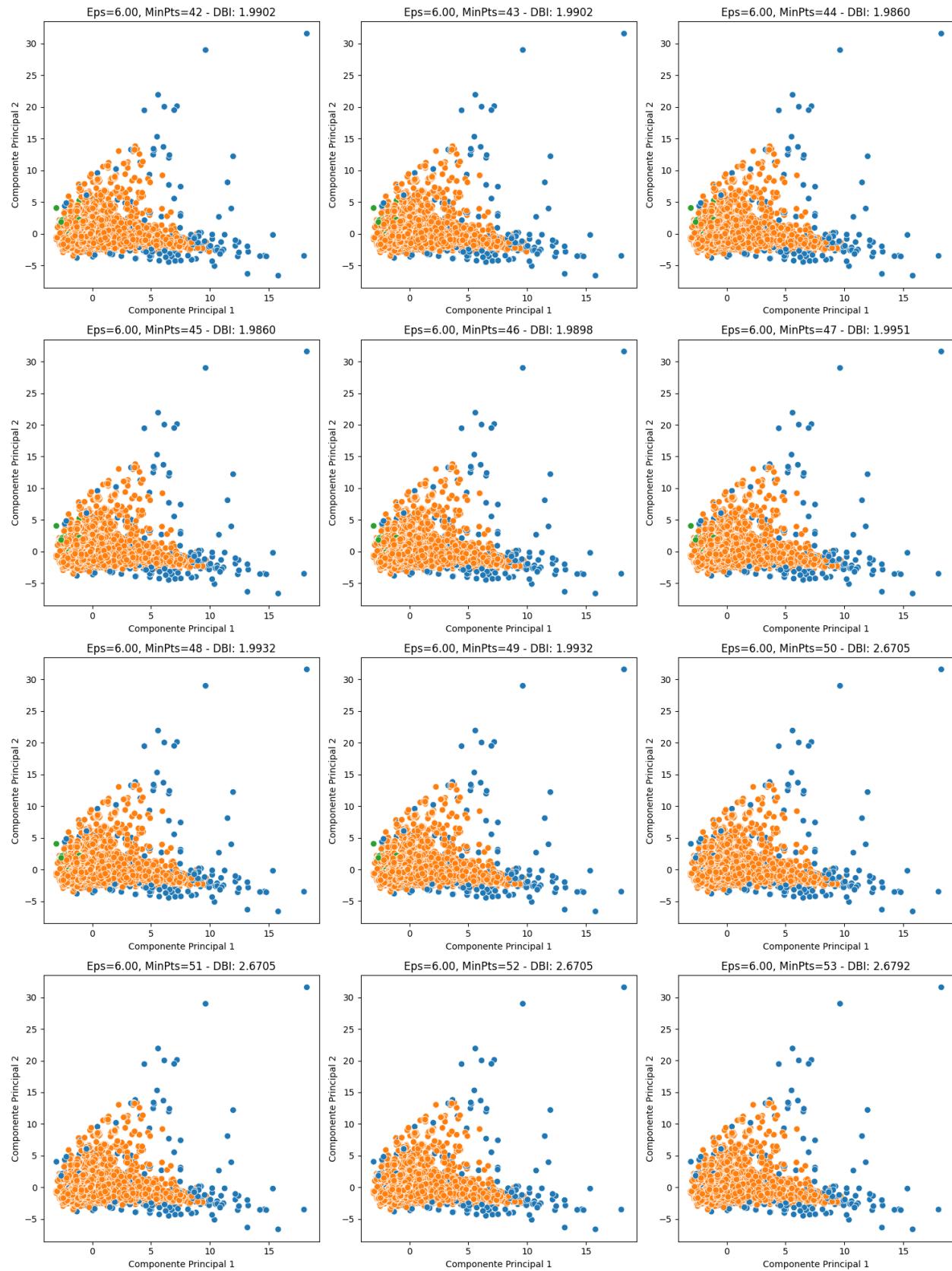
Primero definimos la función que nos ayudará a trazar y evaluar los resultados. Lo usamos para varios valores de épsilon (rango de 5 a 7.5) para ver como dichos parámetros influyen la calidad de clustering.

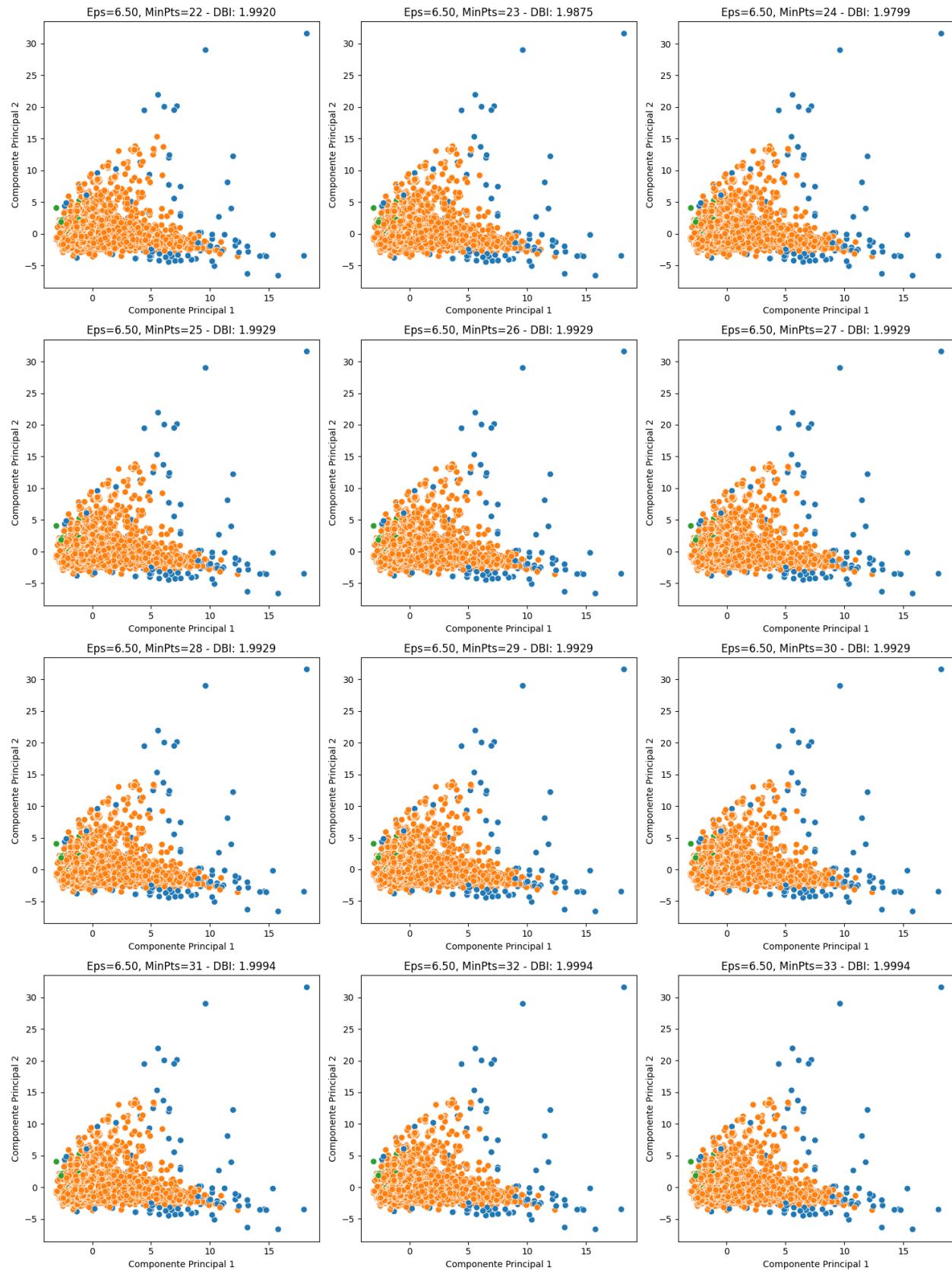
Empezamos con visualizar los resultados con rango de MinPts muy grande. A continuación visualizamos los resultados para la gama de mejores puntuaciones obtenidas en el código anterior.

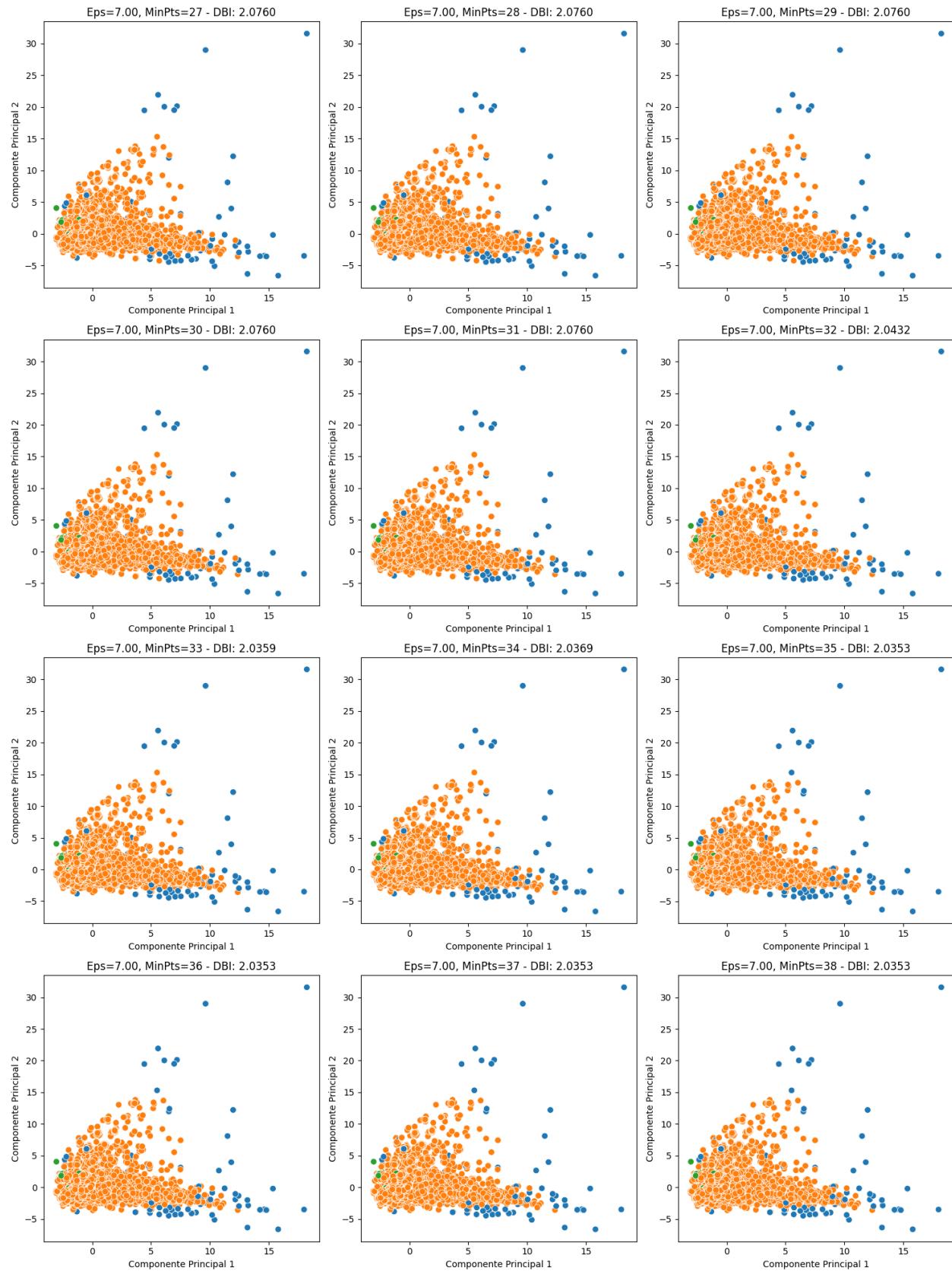
Analizando los resultados podemos elegir la mejor opción.

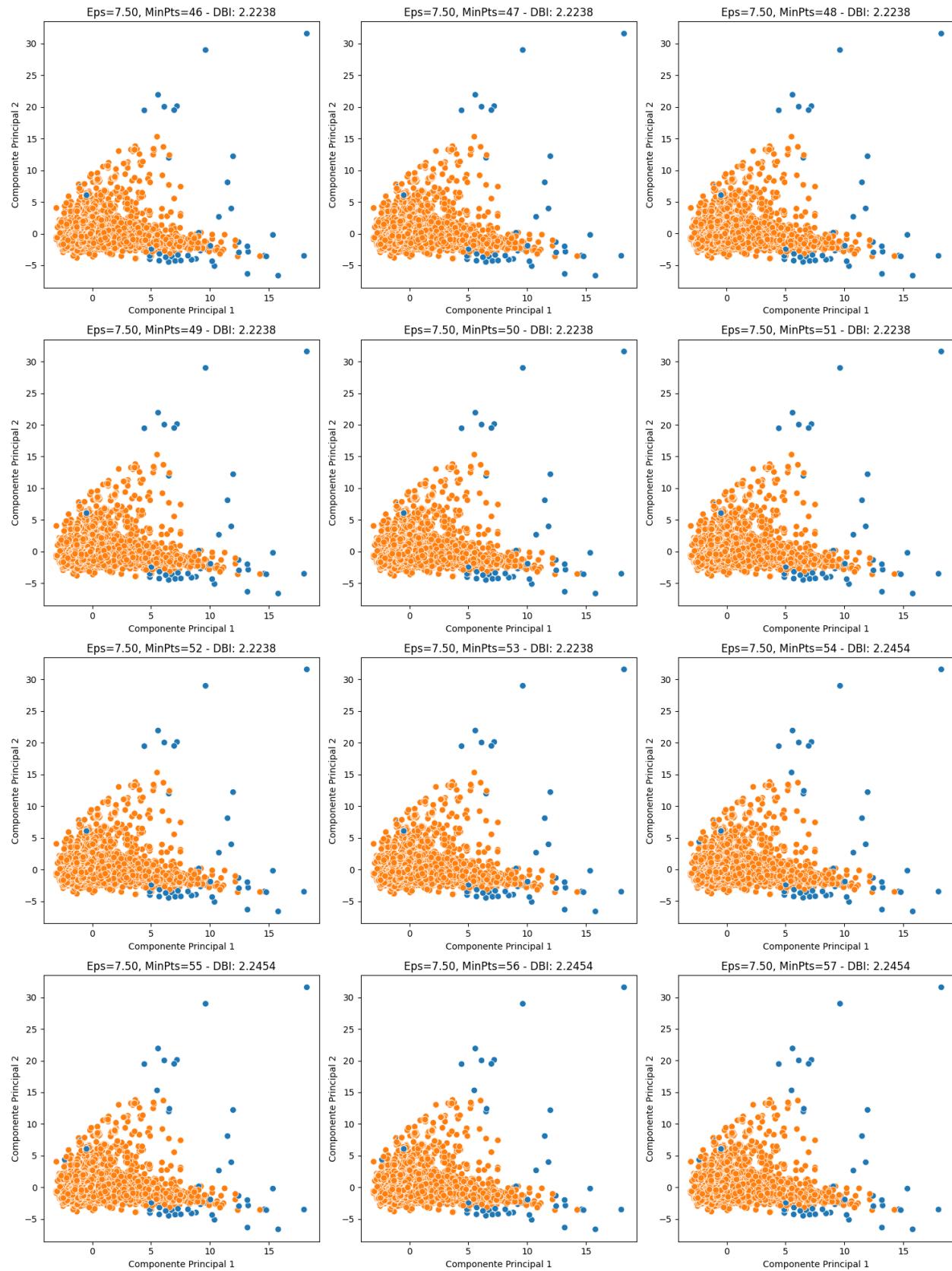












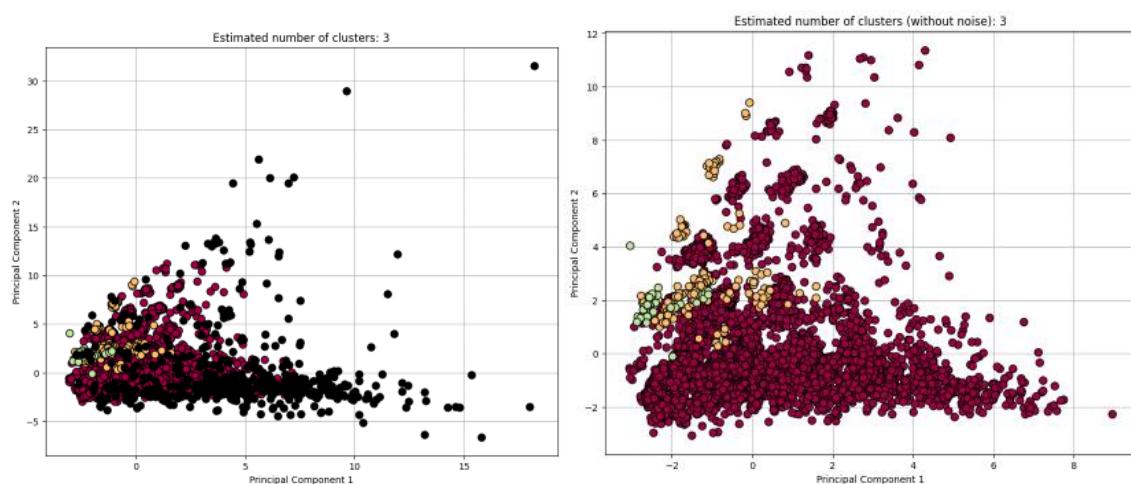
4.3.4. Índices de calidad.

Como las tareas 4.3 y 4.4 están conectados, vamos a hacer el análisis de clustering tras realizar la próxima tarea.

Tras analizar los resultados de la tarea 4.3, elegimos el mejor resultado para cada épsilon y lo evaluamos utilizando la puntuación de silueta, la puntuación calinski-harabasz y la puntuación davies-bouldin.

En primer lugar, perfeccionamos DBSCAN para obtener el número estimado de clusters y el número de puntos de ruido. A continuación, visualizamos los resultados. Primero, con el ruido representado en negro. Y luego, sin él.

4.3.4.1. Épsilon = 5



Silhouette Coefficient: 0.231

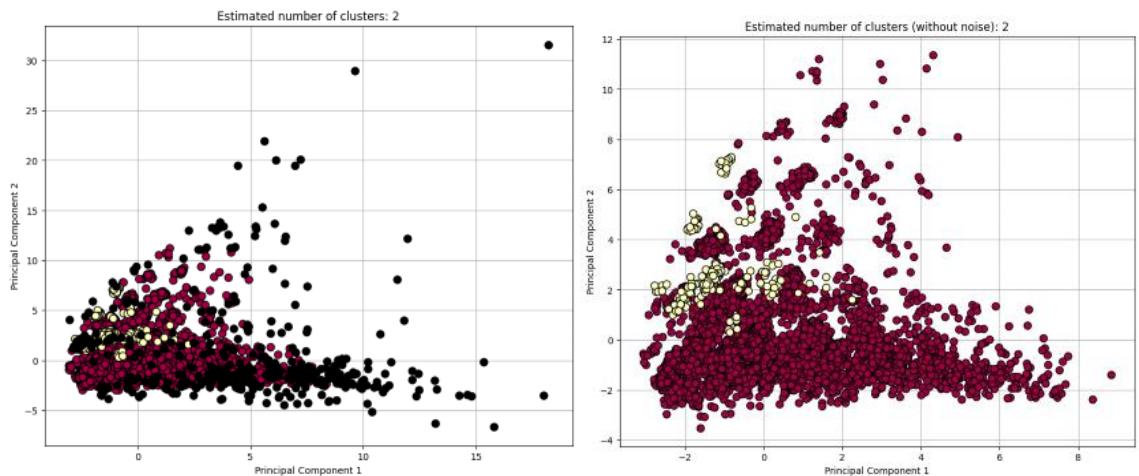
Calinski-Harabasz Index: 351.893

Davies-Bouldin Index: 2.092

En ese caso tenemos 3 clústeres, lo que podría parecer interesante, pero:

- El coeficiente de Silhouette es solo 0.231, lo que sugiere una separación deficiente de los clústeres.
- El DBI es 2.092, indicando una calidad de clústeres inferior a otras configuraciones.
- La cantidad de puntos ruido es la más alta (485).

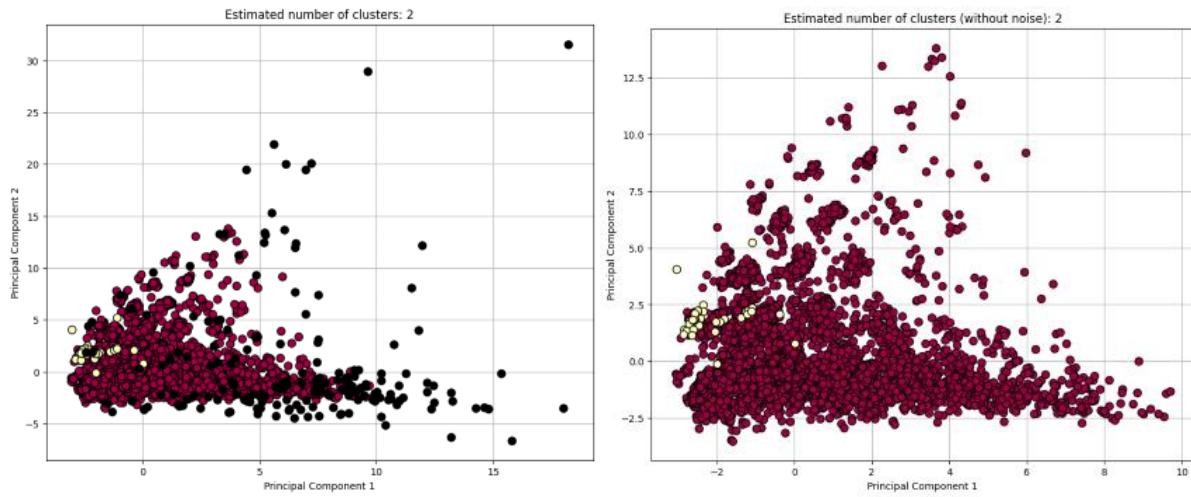
4.3.4.2. Épsilon = 5.5



Silhouette Coefficient: 0.226 Calinski-Harabasz Index: 374.994 Davies-Bouldin Index: 2.612

- El coeficiente de Silhouette es bajo (0.226), lo que indica que los clústeres están menos definidos en comparación con $\text{eps} = 6$ o $\text{eps} = 6.5$.
- El DBI es el más alto (2.612), lo que indica una calidad de clústeres inferior.
- La cantidad de puntos de ruido sigue siendo alta (411).

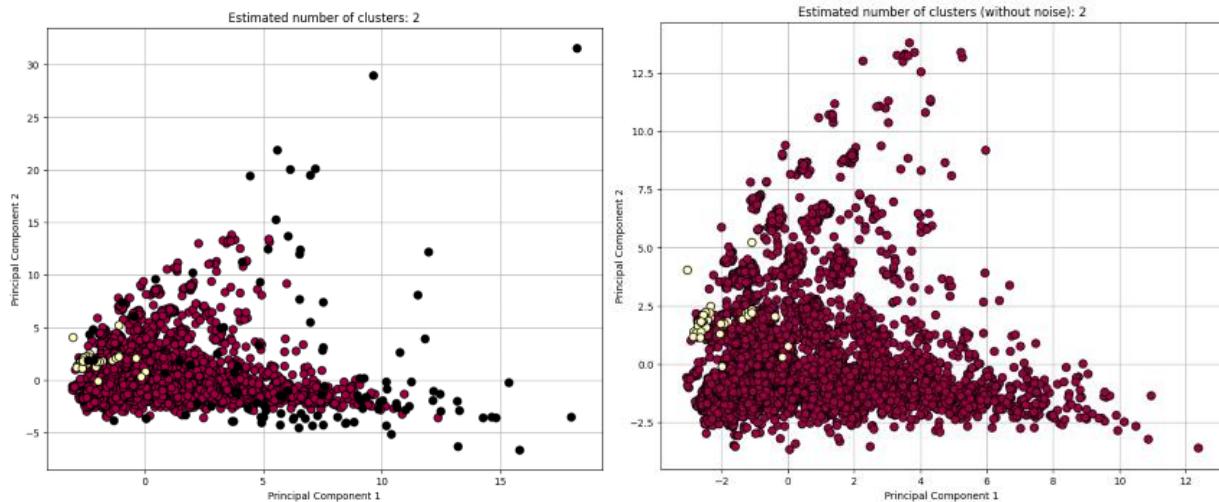
4.3.4.3. Épsilon = 6



Silhouette Coefficient: 0.477 Calinski-Harabasz Index: 310.894 Davies-Bouldin Index: 1.986

- El DBI es 1.986, ligeramente más alto que el de $\text{eps} = 6.5$ (1.980).
- La cantidad de puntos ruido es mayor (179) que con $\text{eps} = 6.5$.

4.3.4.4. ϵ psilon = 6.5 – la mejor opción

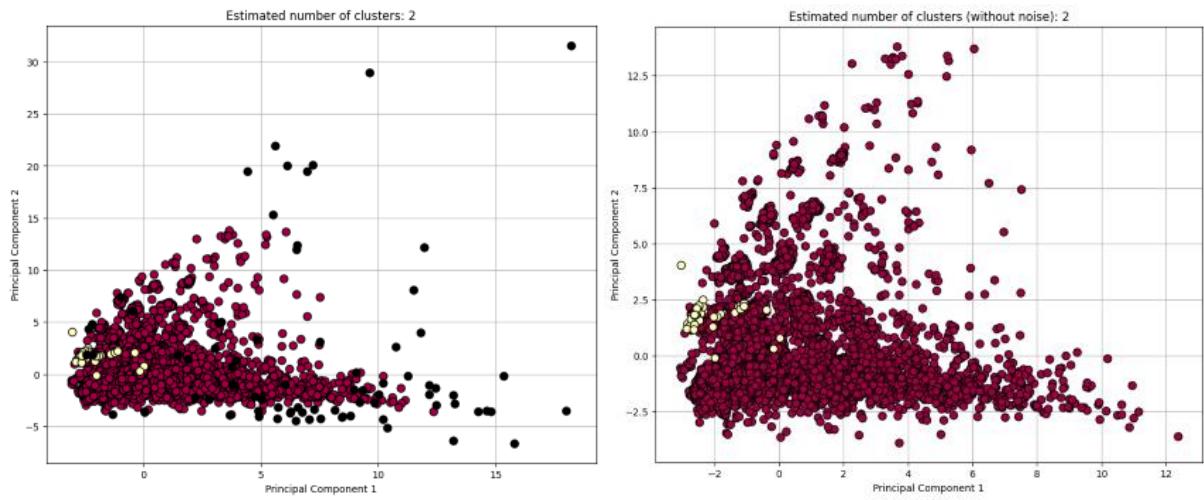


Silhouette Coefficient: 0.477

Calinski-Harabasz Index: 275.341

Davies-Bouldin Index: 1.980

4.3.4.5. ϵ psilon = 7



Silhouette Coefficient: 0.477

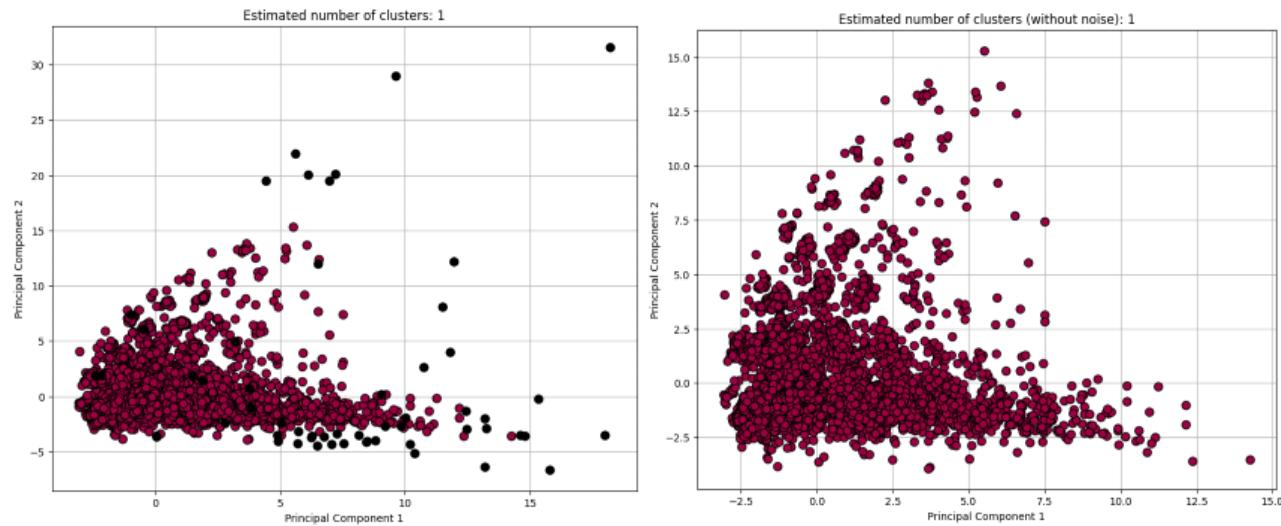
Calinski-Harabasz Index: 248.804

Davies-Bouldin Index: 2.035

- El DBI aumenta a 2.035, lo que sugiere una disminución en la calidad de los clústeres en comparación con ϵ ps = 6.5.

- La cantidad de puntos ruido es menor (77), pero a costa de un DBI y CH peores.

4.3.4.6. ϵ psilon = 7.5



Silhouette Coefficient: 0.680 Calinski-Harabasz Index: 220.695 Davies-Bouldin Index: 2.224

- Solo 1 clúster, lo que significa que no hay una división útil.
- Esta opción no es útil para analizar la estructura de los datos proporcionados.

4.3.4.7. Conclusión.

A la hora de elegir los mejores resultados, deberíamos maximizar la puntuación Silhouette y la puntuación Calinski-Harabasz, y minimizar el índice Davies-Bouldin.

Para ϵ s = 6, ϵ s = 6.5 y ϵ s = 7, el valor de Silhouette es 0.477, que es significativamente más alto en comparación con otras configuraciones, lo que indica que los clústeres están mejor definidos y claramente separados.

El valor más bajo de DBI=1.980 se obtuvo con ϵ s = 6.5. Ese índice mide la calidad de los clústeres, entonces se puede decir que la configuración col el ϵ s = 6.5 nos da los mejores resultados.

En caso de índice de Calinski-Harabasz, el valor más alto se obtuvo para el ϵ s = 5.5. Sin embargo, para ese valor de ϵ s, los valores de otros índices no son tan buenos.

Para ϵ s = 6.5, el valor de CH es 275.341, lo que sigue siendo aceptable y está equilibrado con otros indicadores. También, la cantidad de puntos etiquetados como ruido disminuye a 106.

Al final, pensamos que es mejor usar el DBSCAN con ϵ s = 6.5, porque:

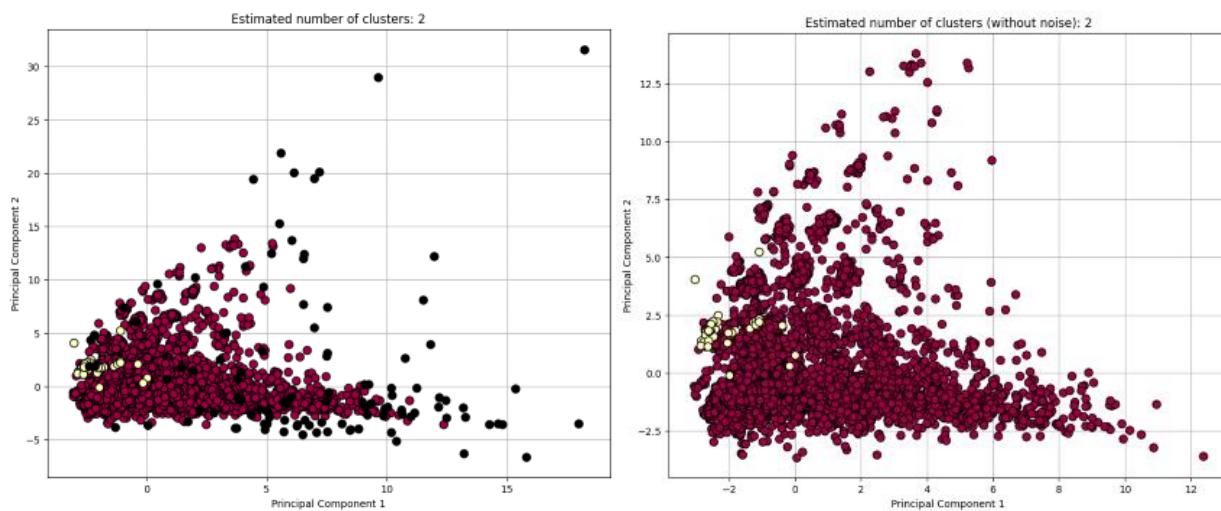
Maximiza el coeficiente de Silhouette (0.477) y minimiza el índice de Davies-Bouldin (1.980). La cantidad de puntos de ruido (106) es aceptable, y los clústeres están claramente definidos. Por lo tanto, elegir ϵ s = 6.5 proporciona el mejor equilibrio entre calidad, separación de los clústeres y cantidad de puntos de ruido.

4.3.5. Número óptimo de clusters.

Basándonos en los resultados anteriores, los datos deberían dividirse en **2 clústeres** con $\text{eps} = 6.5$ y $\text{min_samples} = 24$.

Conseguimos esta conclusión después de analizar varias opciones. En caso de casi todos de varios epsilon, el número óptimo de clusters es 2. Eso significa, que en DBSCAN, como los algoritmos de apartados anteriores, divide los datos proporcionados en dos grupos visibles

Los resultados finales se presentan así:



Estimated number of clusters: 2

Estimated number of noise points: 106

Silhouette Coefficient: 0.477

Calinski-Harabasz Index: 275.341

Davies-Bouldin Index: 1.980