Proyecto sobre aprendizaje no supervisado

Aprendizaje Automático I

CDIA 3

(AI Generated)

Hecho por:

Alonso Ruiz Palomo

Juan Moreno Segura

Índice

Accidentes de Tráfico en Madrid

- Tarea 3.5: Compara los resultados obtenidos con K-means y el clustering aglomerativo/jerárquico. Discute las ventajas y desventajas de cada método en diferentes tipos de datos.

- Aplicación de algoritmos de clustering de densidad (en 3_densidad.ipynb)
  - Tarea 4.1: Realiza el pre-procesamiento necesario para poder aplicar algoritmos de densidad.
  - Tarea 4.2: Establece el radio (eps) y número de puntos mínimo número más adecuado de clusters para el dataset proporcionado.
  - Tarea 4.3: ¿Cómo varía la calidad del clustering con diferentes valores de 'eps' y de minpoints?
  - Tarea 4.4 Utiliza por lo menos dos índices de calidad de clustering y analiza sus resultados.
  - Tarea 4.5 ¿Cuál es el número óptimo de clusters? ¿por qué?
- Aplicación de otros algoritmos
  - Tarea 5.1: Emplea otros algoritmos como HDBScan y compara con otros algoritmos su rendimiento.
  - Tarea 5.2: Emplea otros algoritmos como K-modes y compara con otros algoritmos su rendimiento.

Accidentes de Tráfico en Madrid

- Descripción del dataset (en 1_jerarquico.ipynb)

- Tarea 1: Análisis descriptivo del dataset y preproceso.

Cargamos el dataset y hacemos primero un análisis cualitativo.

Podemos ver que, por las columnas del dataset tenemos datos relevantes sobre los accidentes como por ejemplo la fecha y hora en la que ocurrió, localización, distrito, estado meteorológico y también sobre el conductor o pasajeros como rango de edad, sexo, si era positivo en drogas o alcohol, etc.

Vemos que por la forma del dataset tenemos más de 40000 datos de accidentes con 19 características cada uno. Haciendo un info(), vemos que tenemos valores faltantes y que la mayoría de nuestras columnas son tipo object, por lo que tendremos que pasarlas más adelante a categóricas. También tenemos tipos numéricos.

Hacemos describe de los tipos numéricos para ver estadísticas relevantes sobre las columnas numéricas, y sobre las de tipo object, en las que vemos que tenemos muchas con pocos valores distintos, por lo que haremos un one-hot encoding.

A continuación, vamos a empezar con el tratamiento de las columnas:

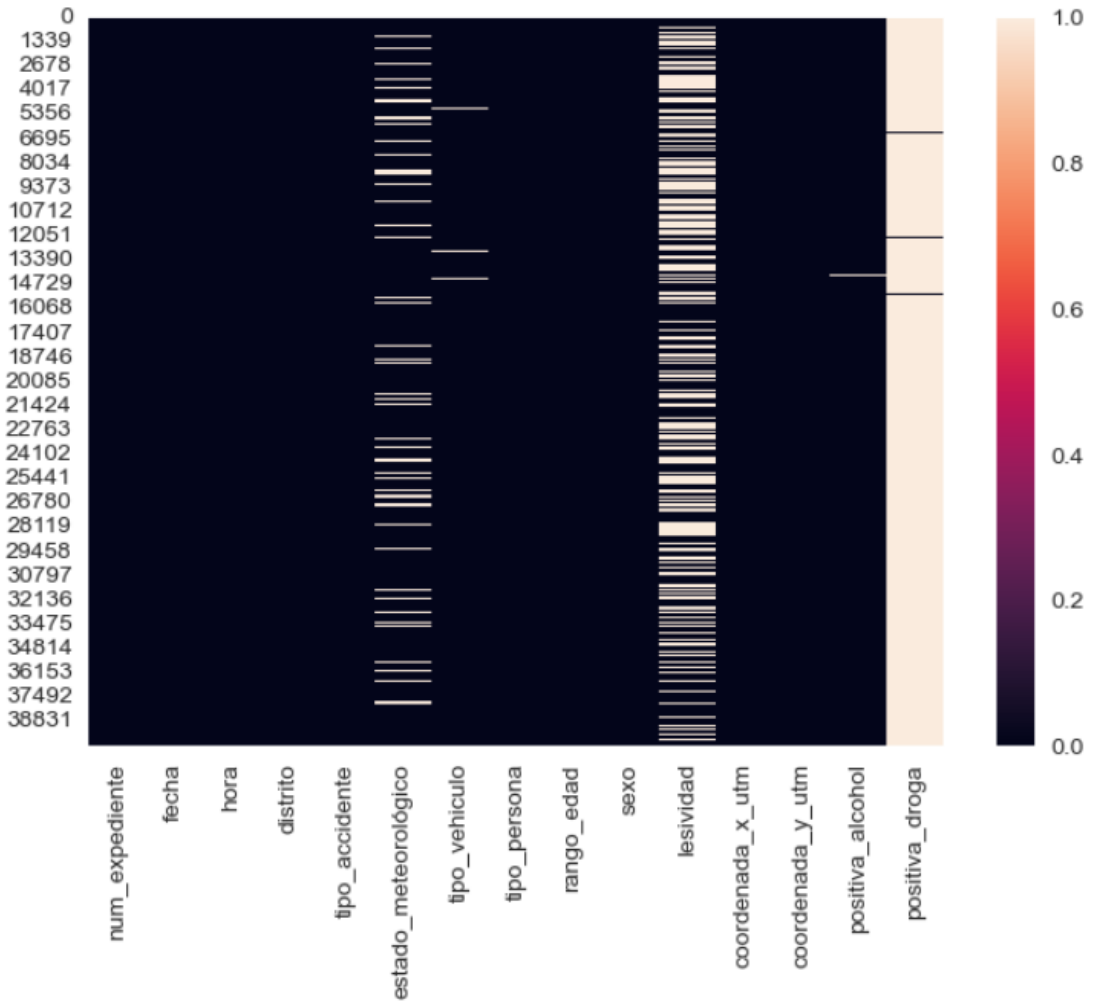
De la columna num_expediente vamos a quedarnos solo con los números que aparecen tras la letra S, para pasarla a numérica, ya que el resto es siempre igual y el año ya lo tenemos en la columna “fecha”.

Vemos que tenemos el mismo número de valores para las columnas “cod_distrito” y “distrito”. Lo mismo sucede en “cod_lesividad” y “lesividad”. Por lo que vamos a quedarnos con la columna categórica, eliminando la que empieza por “cod” ya que ambas hacen referencia a la misma información, por lo que es redundante una de ellas. Eliminamos la numérica ya que, por ejemplo, en cod_distrito, el modelo interpretaría que “1” y “2” son distritos más cercanos que “1” y “4”, lo cual no tiene por qué ser así. Lo mismo sucede con la lesividad.

También vamos a eliminar la columna “localización” y “número” que indican la calle y el número de esta en la que ha sucedido el accidente, ya que tenemos otras 2 columnas con las coordenadas del accidente, en tipo numérico, mucho más fácil para meterlo a un modelo de aprendizaje.

Más adelante seguiremos con el tratamiento de las columnas, por ahora lo dejamos así.

Ahora nos vamos a centrar en el tratamiento de valores faltantes.



Vemos que los valores faltantes se centran en 8 columnas, por lo que vamos a ir tratándolas una a una.

Primero en la columna tipo_accident" tenemos 3 valores faltantes, por lo que vamos a convertirlos al valor "Otro" (ya existente en la columna).

Después encontramos 4658 valores faltantes en la columna estado_meteorológico, por lo que vamos a convertirlos a "Se desconoce" (valor ya existente en la columna).

Luego tenemos 321 valores faltantes en tipo_vehículo, que vamos a cambiarlos a "Sin especificar" (ya existente en la columna).

Encontramos más tarde 17829 valores faltantes en la columna lesividad, que los convertimos a "Desconocido", añadiendo este valor a la columna.

En las columnas de las coordenadas x e y del accidente, tenemos 10 valores faltantes en cada una, que vemos que coinciden con el mismo registro de accidente porque tienen el mismo índice. Vamos a rellenar estos valores con la media de cada columna, ya que es numérica.

Hay 138 valores faltantes en positiva_alcohol. Como vemos que tenemos valores "S" o "N" en esa columna, asumimos que los nan son porque no se hizo control de

alcoholemia a la persona involucrada en el accidente, así que los convertiremos a “Desconocido”.

Ya la última columna que vemos con valores faltantes es positiva_droga con 39995 de estos, la que más tiene. El único valor que tiene esta columna es 1, por lo que asumimos que los valores faltantes son porque no se hizo test de drogas, o dio negativo, así que los convertimos a 0.

Ya no tenemos valores faltantes.

Vamos a hacer un cambio de tipos.

Primero vamos a pasar a formato datetime la columna fecha, y vamos a separar la columna hora en otras 2 columnas para tener en una la hora y en otra el minuto, y así eliminar la columa hora.

Guardamos el dataframe actual en df_raw, para mantener las columnas tipo object para las visualizaciones y otros apartados en las que las necesitemos.

Exportamos este df_raw en pickle por si lo necesitáramos más adelante en otros apartados. Exportamos en pickle para guardar los tipos.

A continuación, nos quedamos con las columnas tipo object, y vemos que la que más valores distintos tiene es tipo_vehículo, con 32 valores distintos, pero el resto tiene pocos valores distintos. Como hemos dicho antes, si hacemos un LabelEncoder, el modelo puede interpretar que dos categorías son más similares por su cercanía numérica y esto no tiene por qué ser así, asi que hacemos un one-hot encoding con pd.get_dummies.

Ahora eliminamos las columnas tipo object del dataframe original y concatenamos las del dummies. Ahora tenemos 115 columnas, son demasiadas.

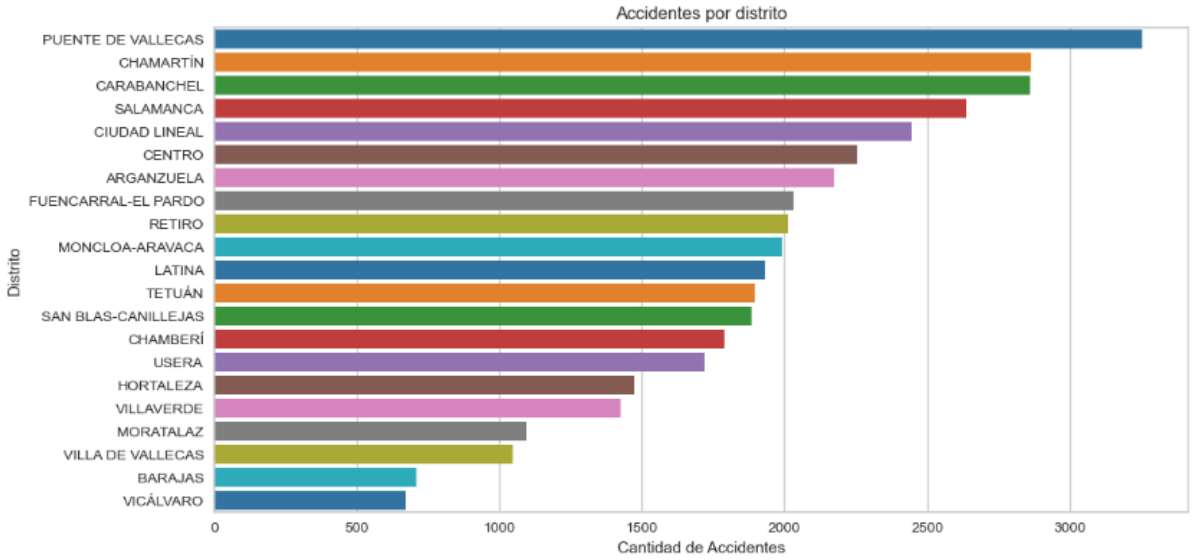
Vamos a eliminar las columnas que tengan una varianza menor que 0.001 ya que estas no van a aportar casi información al entrenamiento porque sus valores prácticamente no varían. Nos quedamos con 97 columnas.

Sacamos 2 columnas de la columna fecha: mes y dia, para separarlo y así tener columnas numéricas y poder eliminar la columna fecha (tipo datetime).

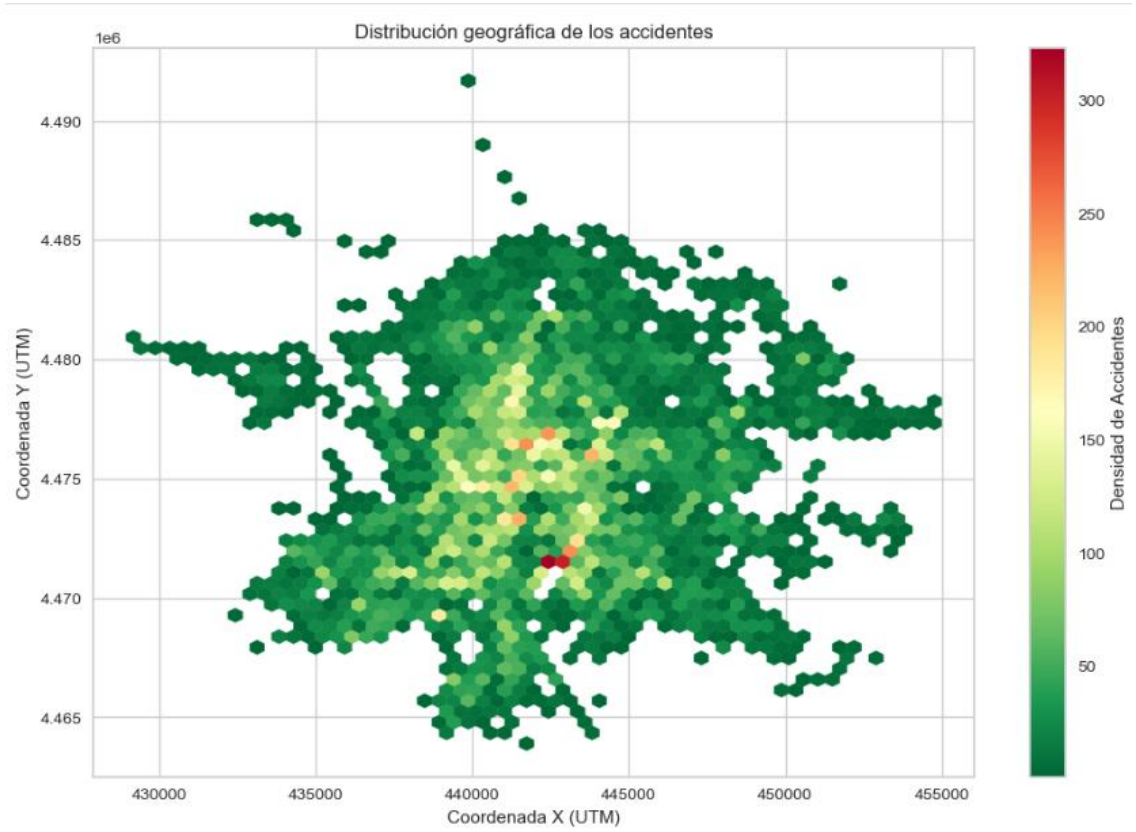
Por último, vamos a estandarizar las columnas con valores superiores a 2 (para no seleccionar columnas booleanas) con el StandardScaler para nuestro entrenamiento de modelos. Convertimos todos nuestros datos a tipo float64 para el entrenamiento y guardamos el dataframe actual en un pickle para importarlo en el resto de cuadernos.

- Visualización de los datos:

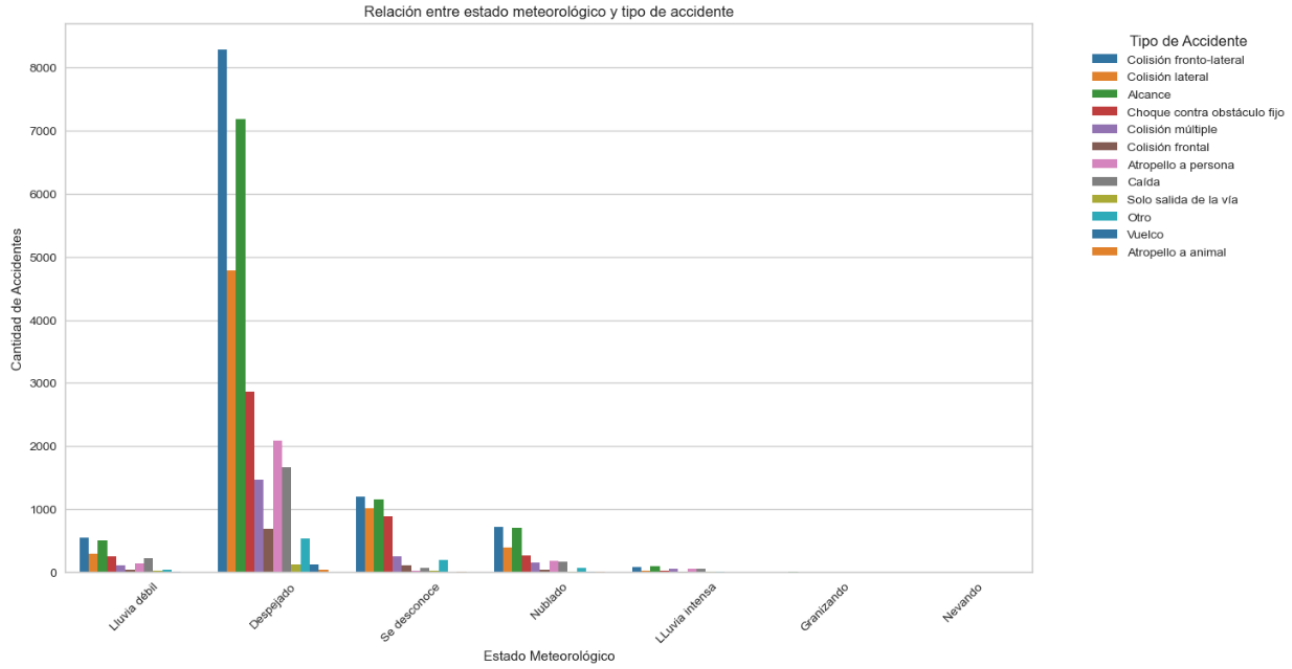
Vamos a hacer visualizaciones de nuestros datos para entenderlos mejor utilzando el dataframe df_raw, que conserva las columnas tipo object:



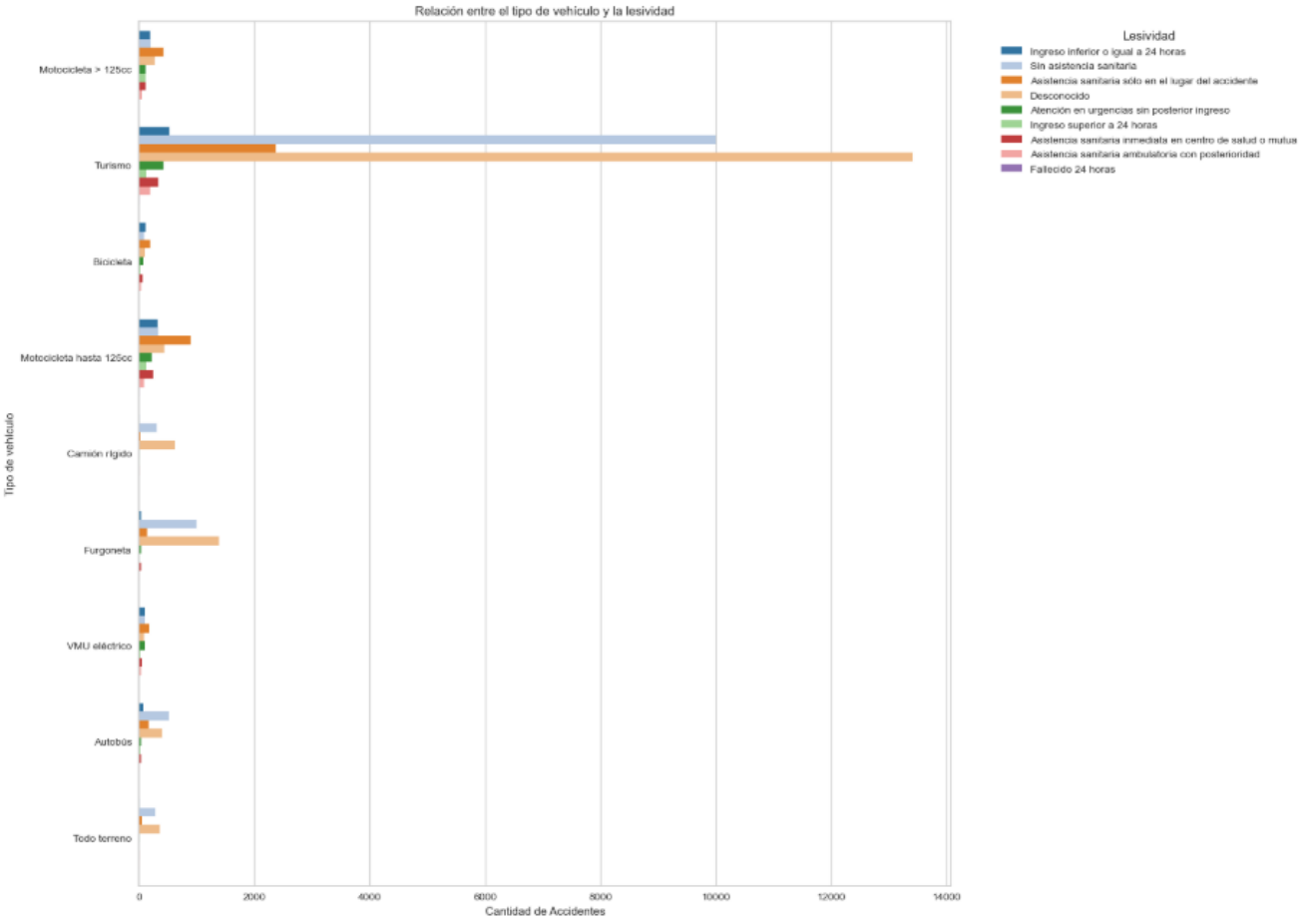
Vemos que el distrito en el que más accidentes se han producido es Puente de Vallecas, y el que menos es Vicálvaro.



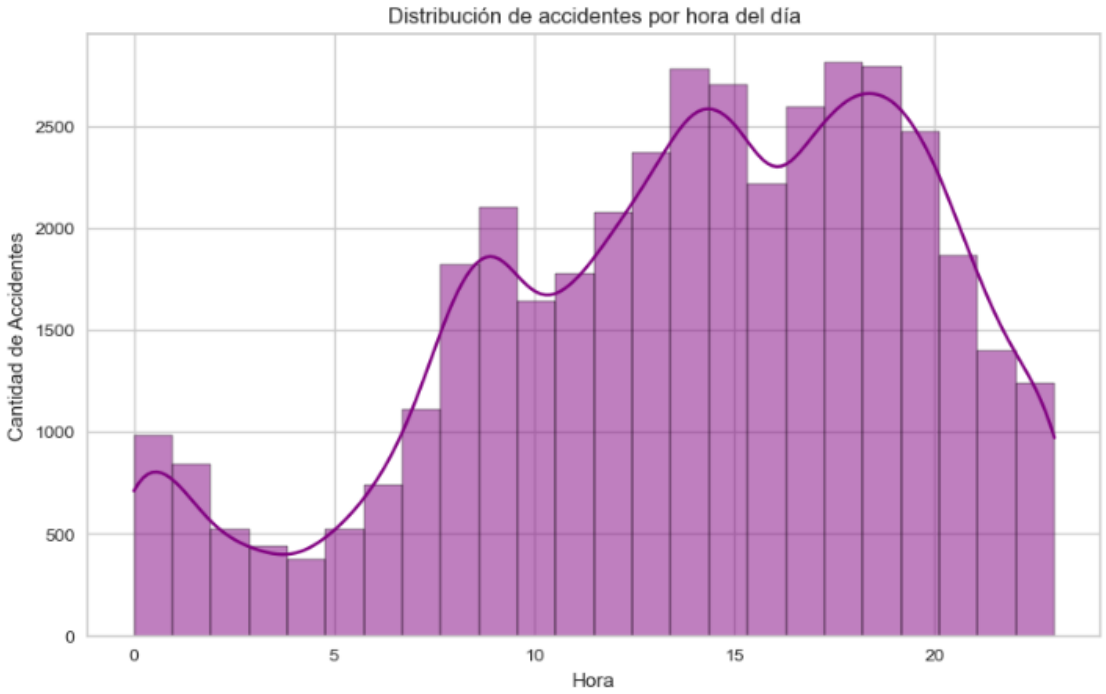
Este mapa indica la distribución por coordenadas con mapa de calor según la densidad de accidentes. Como era de esperar, hay más por el centro, y se extienden al extrarradio por las autopistas (A1-A6), con zonas sin accidentes como la casa de campo o barajas ya que no discurren vehículos por esas zonas.



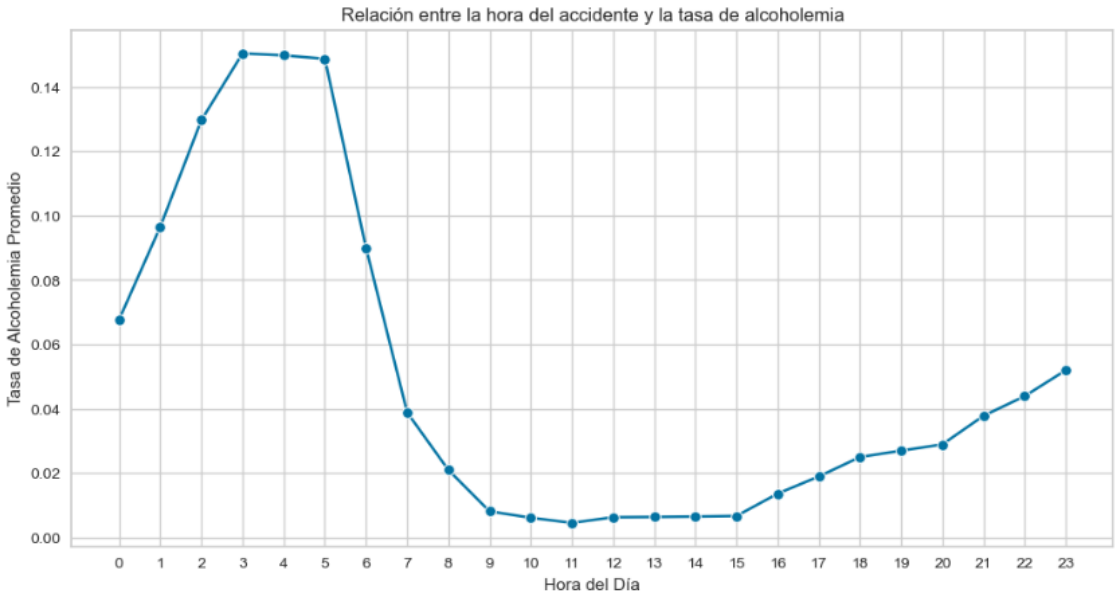
La mayor parte de los accidentes ocurren con cielo despejado por el clima que tiene Madrid, bastante soleado, pero podemos ver que cuando llueve o hay niebla también tenemos accidentes, elevándose los que son por alcance debido a la humedad de la calzada que la hace más deslizante. Los accidentes más comunes son por colisión fronto-lateral, lateral y por alcance.



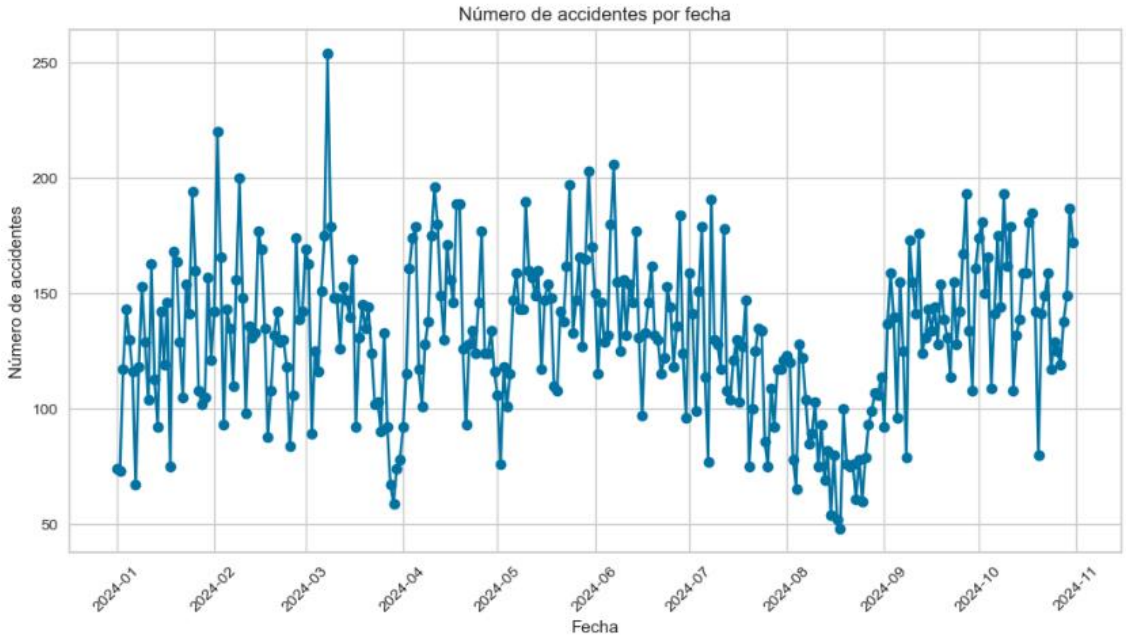
Podemos ver que los turismos son los que más involucrados están en accidentes, porque son los más comunes en las carreteras. Además, las motocicletas suelen necesitar asistencia sanitaria debido a que son más inseguras que un coche o furgoneta por la exposición que suponen, lo mismo sucede con las bicicletas. En cambio, las furgonetas casi nunca necesitan asistencia sanitaria por su robustez.



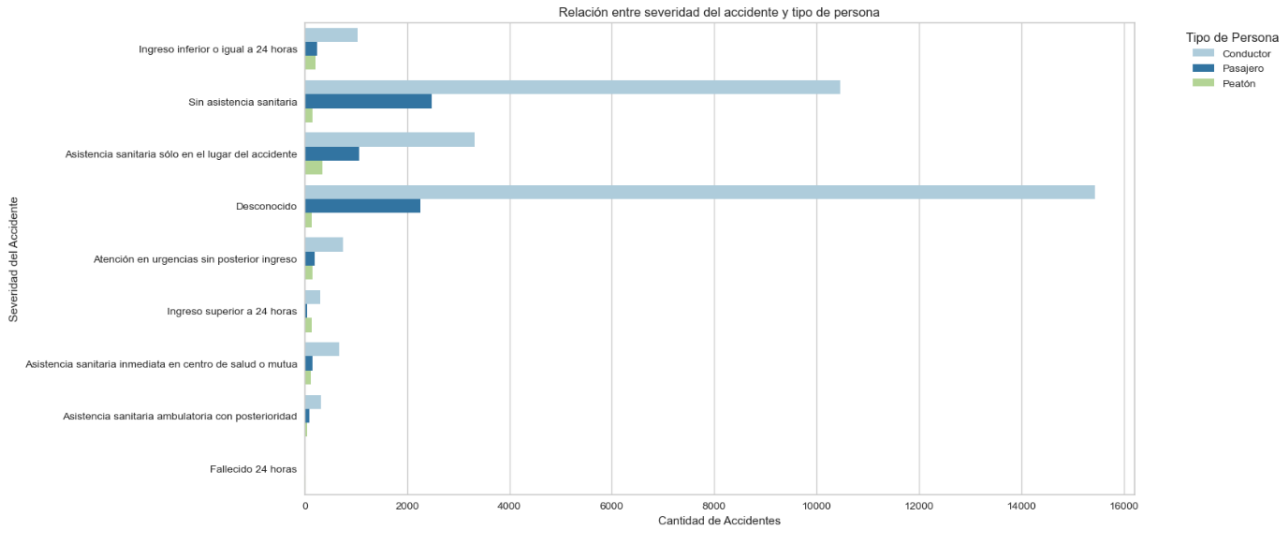
Podemos ver que, como era de esperar, a altas horas de la madrugada hay menos accidentes. Hay un pico cuando la gente está yendo a trabajar (8-9 AM), a la hora de comer (2-3 PM) y por la tarde pronto (6-7 PM), pues es cuando la gente utiliza vehículos para transportarse, para ir y volver de trabajar/estudiar. Por la noche siempre hay menos tráfico.



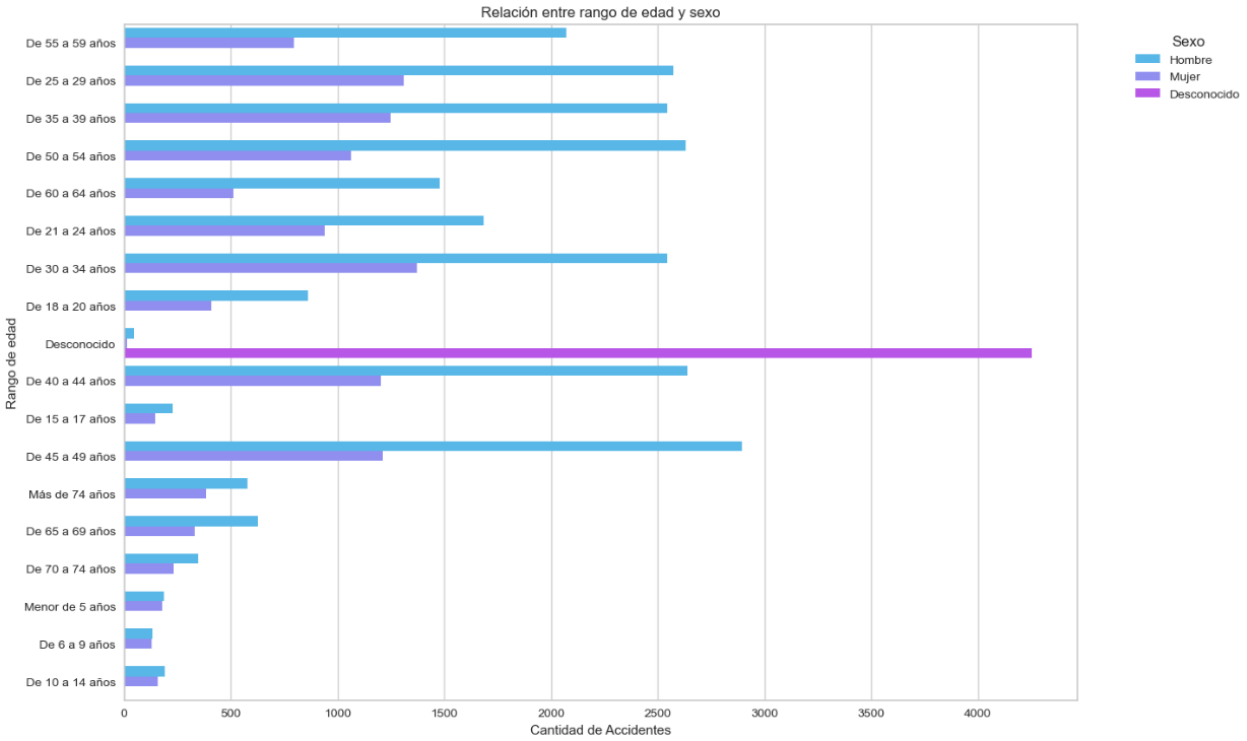
La tasa de alcoholemia positiva es mayor a las horas de la madrugada, por la tendencia de beber cuando es de noche y salir de fiesta con un vehículo. Durante el día es muy bajo y comienza a ascender a partir de las 16H.



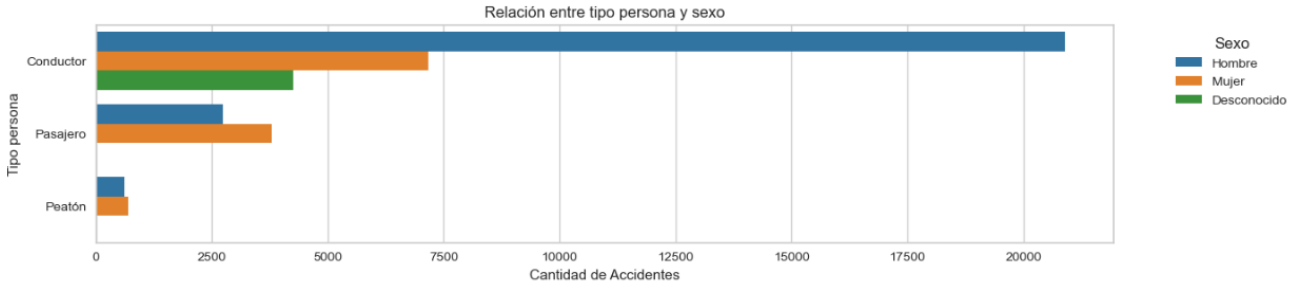
Vemos que en Julio y Agosto descienden los accidentes, seguramente porque al ser periodo vacacional de verano, los madrileños se van fuera de veraneo y hay menos vehículos en la ciudad. El resto del año permanecen bastante uniformes. Con algún pico de accidentes, como en Marzo (salida de Semana Santa).



Vemos que lo usual para los conductores es que no reciban asistencia sanitaria, en cambio los peatones como van más expuestos sí que reciben más, los pasajeros al igual que los conductores.



Vemos que la mayoría de accidentes les sucede a personas entre los 30 y 60 años, ya que son las edades comunes de los conductores, y que los hombres suelen duplicar a las mujeres en los accidentes, también porque aparecen más en los registros. Los niños suelen estar menos involucrados debido a que no conducen, al igual que las personas mayores.



La mayor parte de los conductores involucrados en accidentes son hombres, las mujeres predominan más como pasajeras o peatones.

- Aplicación de algoritmos de clustering jerárquico (en 1_jerarquico.ipynb)

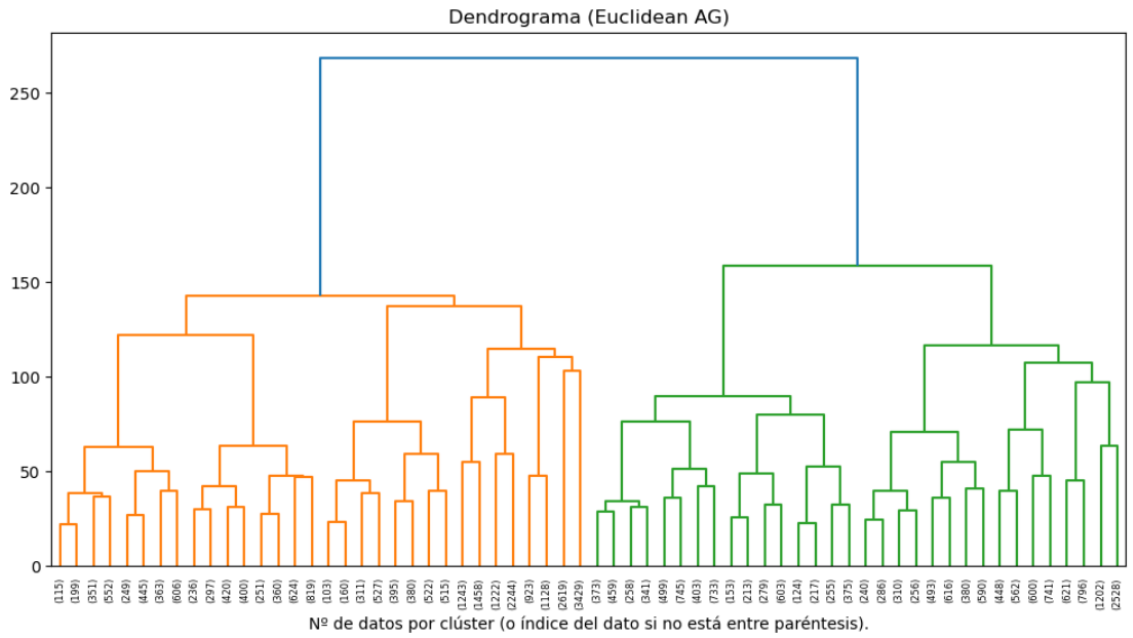
- Tarea 2.1: Aplicar 2 métodos de clustering jerárquico con distintas distancias.
- Tarea 2.2: Analiza si la distribución entre los clústers es uniforme con distintas profundidades.

Voy a responder a las 2 preguntas bajo cada gráfico para tener el apoyo visual.

El primer modelo de clustering jerárquico que vamos a emplear es el Agglomerative Clustering (bottom-up) con distancias euclídeas, coseno y manhattan.

Utilizamos hiperparámetros n_clusters = None para que nos haga el dendograma completo y distance_threshold = 0 (ya que el anterior hiperparámetro está a cero).

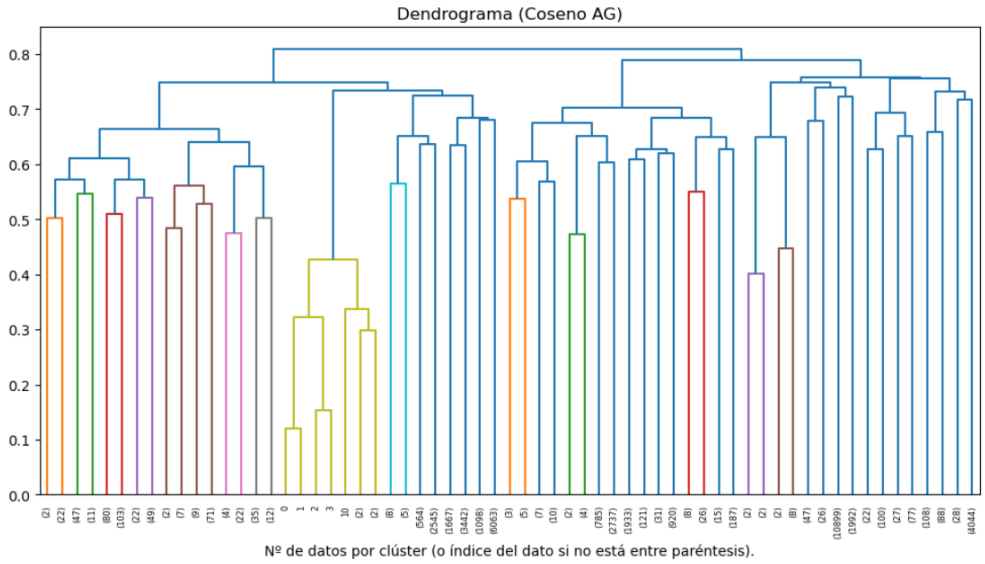
Con distancia euclídea y profundidad igual a 5 obtenemos el dendograma:



Este es el método que mejor hace el clustering, ya que como podemos ver, la división entre los distintos clústers utilizando la distancia euclidea consigue que los clústers estén balanceados en número de puntos. No tenemos ningun clúster con un único dato.

Variando la profundidad vemos que el tamaño de los clústers se divide de manera uniforme, separando por agrupaciones de datos, no por datos individuales.

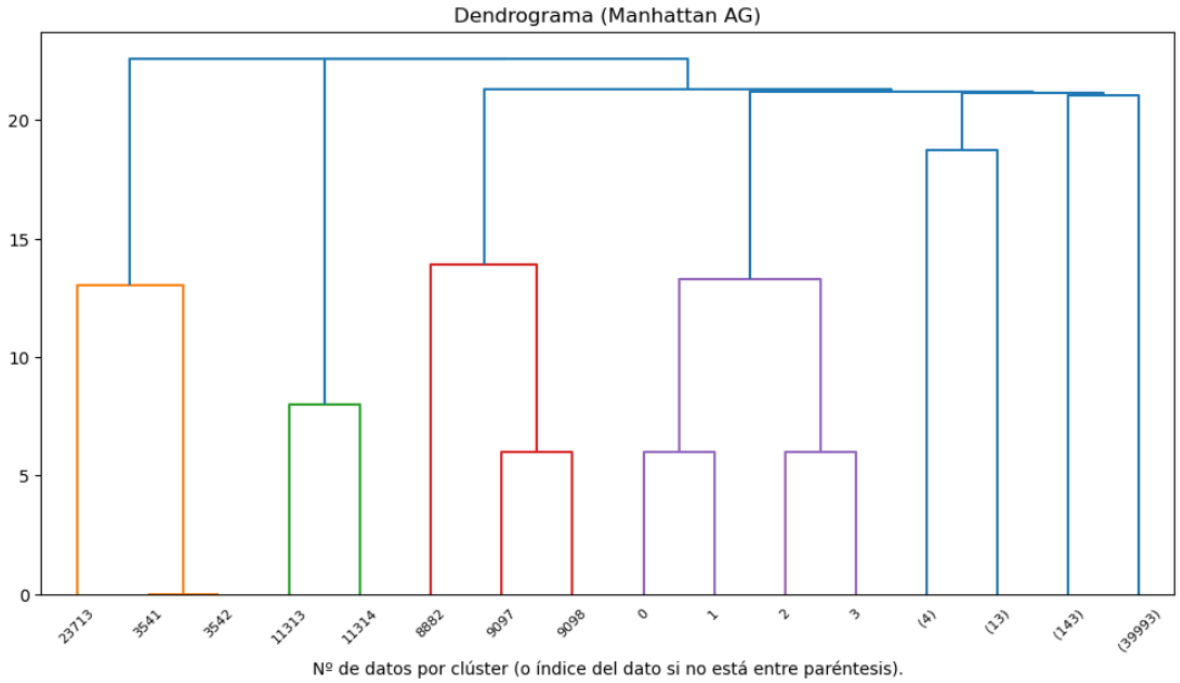
Ahora con la distancia coseno, cambiamos el linkage a "average" que utiliza el enfoque de "enlace promedio":



Con la distancia coseno, vemos que la división de datos entre los distintos clústers se realiza peor que con la euclídea, puesto que unos pocos clústers son los que tienen una gran cantidad de datos, y el resto tienen muy pocos, o incluso se forman clústers compuestos por un solo dato.

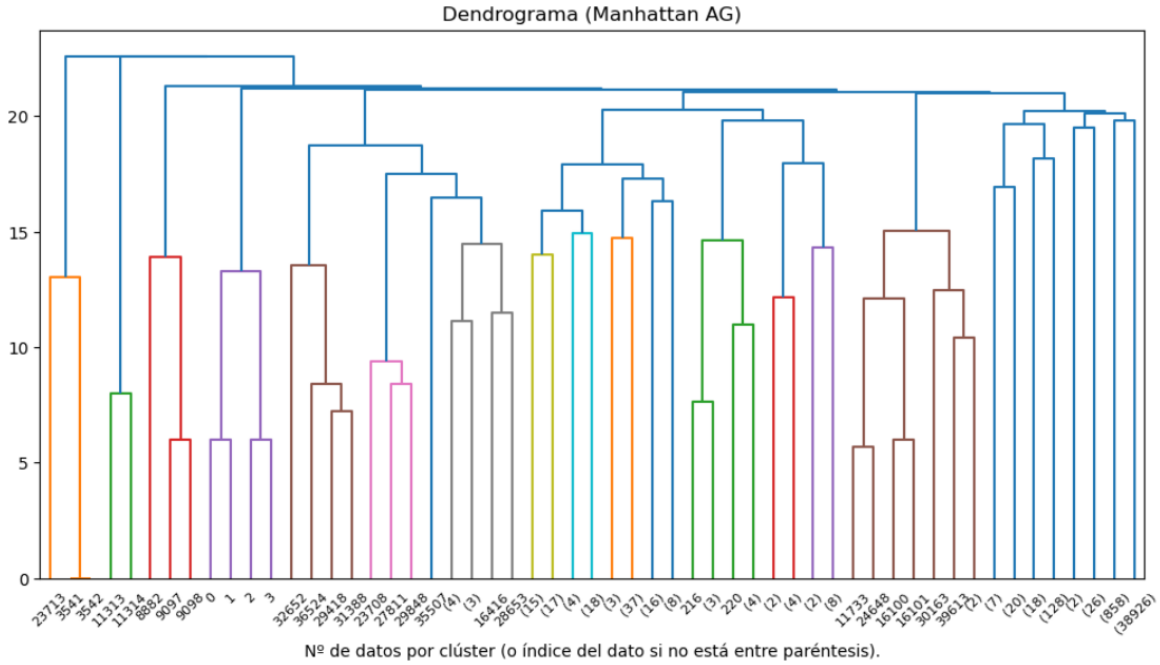
En este caso, variando la profundidad del dendograma vemos que la distribución no es uniforme, vemos el ejemplo aquí con la ramificación color mostaza, que resalta a la vista en comparación con el resto de ramificaciones.

Ahora con distancia manhattan (city block), y linkage = average.



Con la distancia Manhattan vemos que funciona horriblemente, tenemos prácticamente todos los datos en un clúster, y el resto tienen o muy pocos datos, o son directamente un punto por clúster.

Probando con distintas profundidades, vemos que las distribuciones de los datos en los clústers claramente no son nada uniformes, pues, como hemos dicho, casi todos los puntos se concentran en solo una agrupación. Como podemos ver en el siguiente gráfico, aumentando la profundidad permanece esta tendencia:



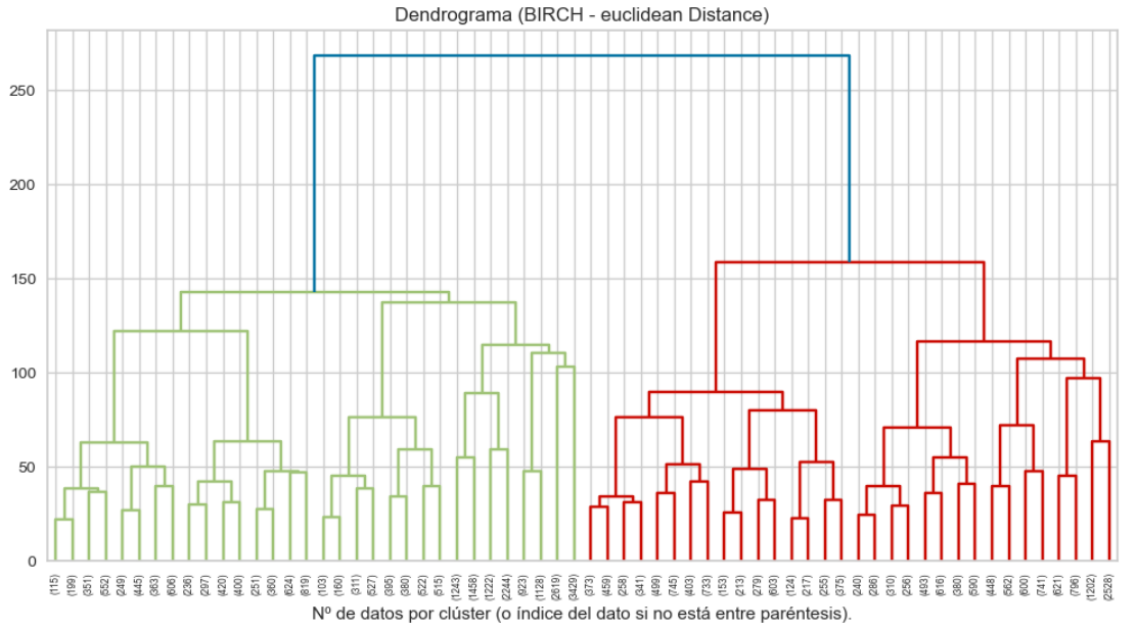
Ahora vamos a emplear BIRCH:

Cremos una función para que nos muestre el dendograma del clustering BIRCH según los datos pasados, el tipo de distancia utilizada, el método de linkage y la profundidad del dendograma (5 por defecto).

Utilizaremos un branching_factor = 50 (número máximo de nodos por hoja), n_clusters = None para generar el dendograma completo, threshold = 0.5 que define la distancia máxima para considerar un punto en un clúster.

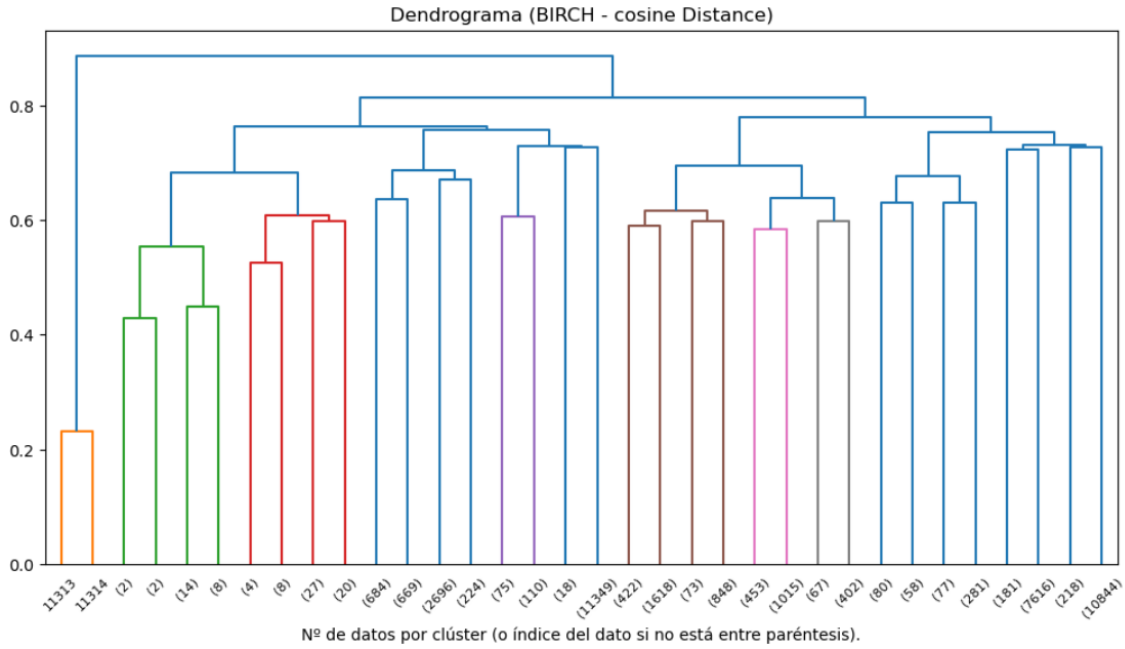
También nos va a devolver el modelo y las labels para más adelante calcular los Índices de calidad.

BIRCH con distancia euclidea y linkage = Ward:



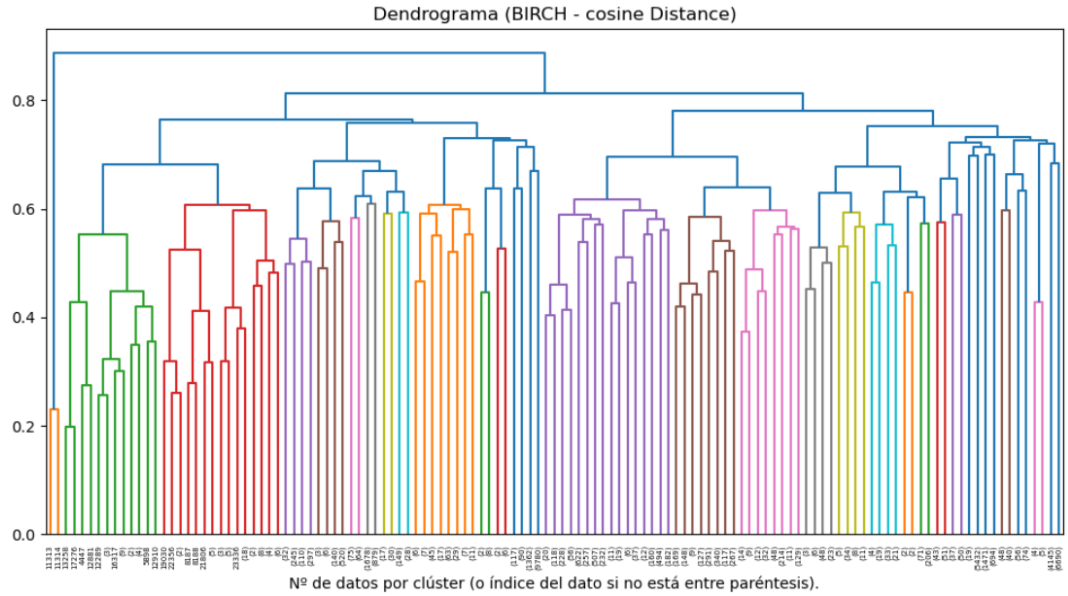
Con el método BIRCH vemos que tenemos prácticamente los mismos resultados que con AgglomerativeClustering empleando la distancia euclídea. Por lo que las conclusiones son las mismas.

Con distancia coseno (normalizamos antes el dataset con la norma l2):

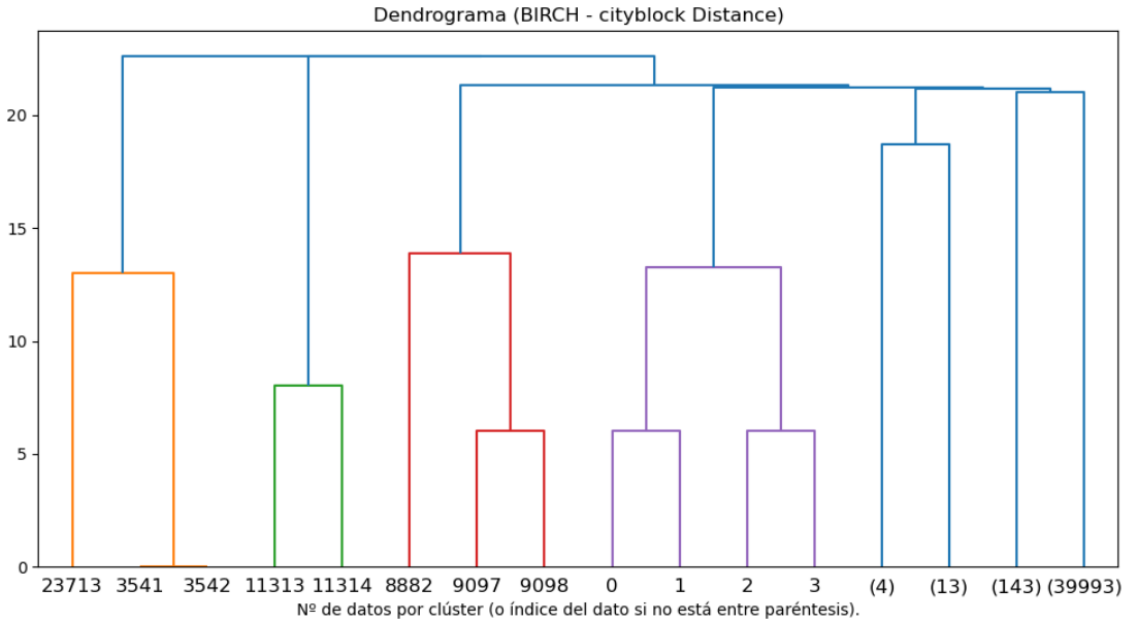


Con la distancia coseno, vemos diferencias con el del AgglomerativeClustering pero también similitudes. Ambas tienen clústers que condensan gran parte de los datos, y otros más pequeños, incluso alguno que solamente tiene un dato. Como diferencias, en el dendograma del BIRCH tenemos un número final de clústeres menor que en el del AgglomerativeClustering.

Variando la profundidad del dendograma vemos que, como con la distancia coseno en el AgglomerativeClustering, la distribución entre los clústers no es uniforme, pues se crean clústers formados por un solo dato, otros muy pequeños y pocos más grandes:



Por último, probamos con distancia cityblock (manhattan) y linkage = average.



Otra vez, tenemos los mismos resultados que con el AgglomerativeClustering con distancia Manhattan. Por lo que las conclusiones son las mismas.

- Tarea 2.3: Cómo afectan las distintas distancias al dendograma.

Vemos que la mejor distancia para nuestro clustering es la euclidea, ya que mantiene una distribución uniforme de los datos entre los distintos clústers. La coseno no funciona mal del todo, y la manhattan es pésima. Además, para la distancia coseno vemos que el dendograma,el eje está comprendido entre valores 0 y 1, lo cual ayuda más a la

comparación con otros métodos. En cambio las distancias euclídea y manhattan utilizan una escala no acotada superiormente, más difícil de interpretar.

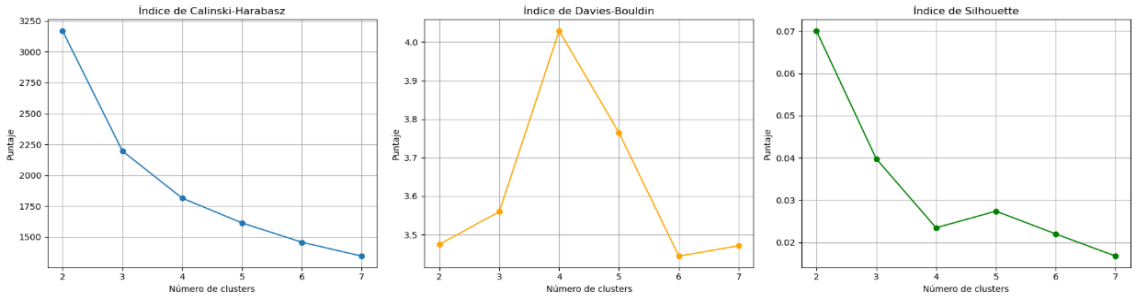
- Tarea 2.4: Emplea 3 índices de calidad y analiza resultados.

Vamos a evaluar con los índices de Calinski-Harabasz, Davies Bouldin y Silueta.

Primero, vemos con Agglomerative Clustering:

Creamos una función para encapsular el código. Con el Agglomerative Clustering no podemos visualizar los diagramas de Silueta por la naturaleza de este algoritmo, pero sí que lo haremos para el BIRCH más adelante.

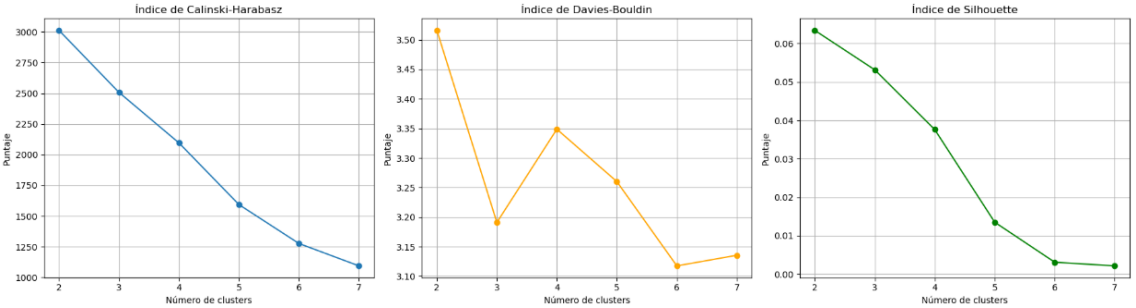
Con distancia euclídea:



Para analizar los resultados tenemos que tener en cuenta que el Índice de Calinski-Harabasz da las puntuaciones mayores al mejor número de clusters, al igual que el Índice de Silueta, y al contrario que el de Davies Bouldin, que da puntuaciones más bajas a la mejor separación en números de clusters.

Así, vemos que nuestro número óptimo de clusters con las 3 métricas es 2, aunque es cierto que con el Índice de Davies Bouldin, 6 clusters obtiene la menor puntuación por lo que sería el número óptimo. Tanto en la gráfica de Davies Bouldin como en la de Silueta vemos que 4 clústers sería un número muy malo para separación de los datos.

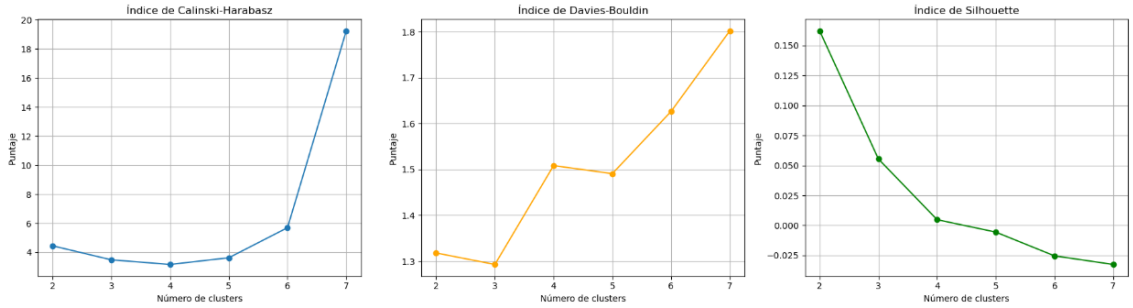
Con distancia coseno:



Con las distancias coseno vemos que el Índice de Calinski-Harabasz no varía casi en comparación con la anterior distancia euclídea, al igual que el Índice de Silueta, con la leve diferencia de que esta vez la bajada de la puntuación a medida que aumentamos el número de clústers es más suave, no tan pronunciada como antes.

El Índice de Davies Bouldin sí que vemos que cambia bastante, sigue apareciendo 6 como el número óptimo de clústers pero esta vez seguido de 7, y obteniendo 2 clusters como el peor número de agrupaciones al tener la puntuación más alta.

Con distancia manhattan:



Encontramos un Índice de Silueta muy similar al anterior, indicando 2 clústers como lo óptimo.

El Índice de Davies Bouldin tiene una tendencia creciente de puntuación a medida que aumentamos el número de clústers, obteniendo 3 clústers como el número óptimo, y después 2.

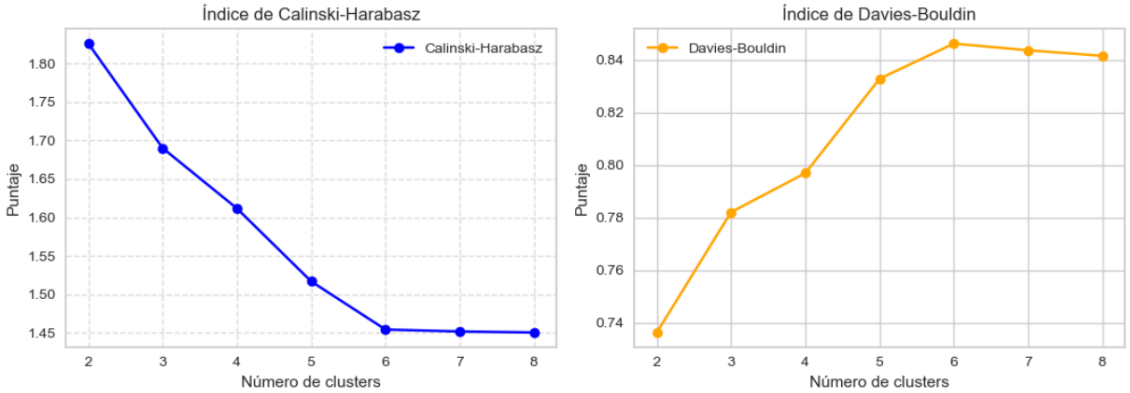
Por último, vemos que el índice de Calinski-Harabasz cambia por completo esta vez, indicando que 7 clústers es la mejor división de los datos, y que el resto de divisiones son muy malas.

Recordemos que, cuando analizamos los dendogramas anteriormente, la distancia manhattan daba resultados muy malos en cuanto a la división de datos en los distintos clústers, por lo que no nos fiaremos mucho de estos Índices de calidad.

Ahora vamos a analizar los índices para BIRCH. Utilizaremos los de Calinski-Harabasz y Davies Bouldin, y después visualizamos los diagramas de Silueta, compatibles con BIRCH.

Encapsulamos el código en funciones al igual que antes.

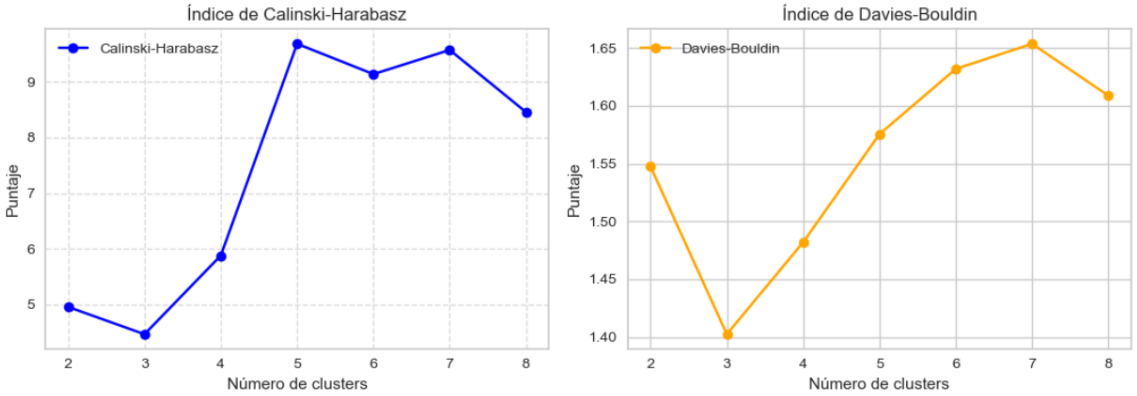
Con distancia euclídea:



Tanto el índice de Calinski-Harabasz como el de Davies Bouldin muestran tendencias similares, es verdad que no se leen igual las dos gráficas porque son métricas

inversamente proporcionales. Justamente por eso tiene sentido que el valor más alto del primero sea para 2 clústers al igual que el más bajo para el segundo índice, indicando el número óptimo de estos. A medida que aumentan el número de clústers, los índices empeoran, indicando que cuantas más divisiones de los datos, peores son estas.

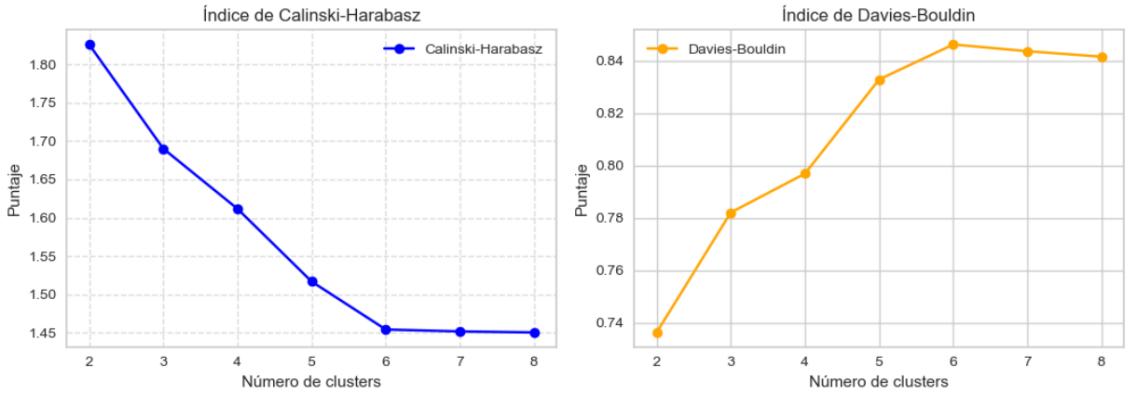
Con distancia coseno:



Para las distancias coseno, vemos que el Índice de Davies Bouldin mantiene su tendencia parecida a la de la distancia euclídea pero esta vez el número óptimo de clústers es 3 y después 4.

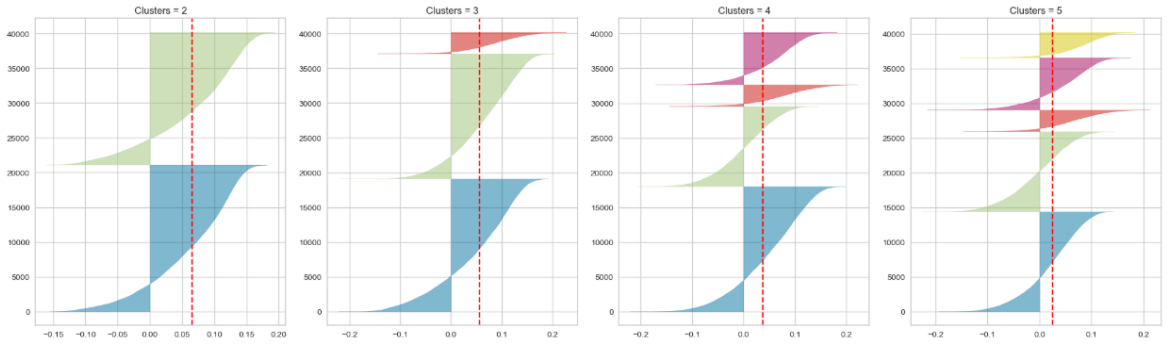
Esta vez se contradice totalmente con el Índice de Calinski-Harabasz ya que este muestra que 3 clústers es la peor división, seguida de 2, y que la mejor serían 5 clústers.

Con distancias manhattan:



Sorprendentemente, para la distancia manhattan (que era la peor) obtenemos prácticamente los mismos índices que con la euclídea.

Diagramas de Silueta para BIRCH, lo hacemos desde 2 a 5 clústers puesto que es muy costoso. Visualizamos el Índice de Silueta únicamente para la distancia euclídea ya que es muy costoso hacerlo y esta es la que mejor separa los datos:



Observamos que en todas las posibles divisiones encontramos datos mal clasificados, aunque sean muy pocos como vemos con el clúster que aparece de color rojo, o el amarillo.

Se ve que tenemos 2 clústers mayoritarios que son el gris y el azul, que, hasta con 5 clústers condensan más de la mitad de los datos. Estos dos clústers son los que más número de datos mal clasificados tienen, lo que tiene sentido al ser los más grandes.

El mejor Índice de Silueta se lo lleva la división en 2 clústers, muy poco por delante de la de 3 clústers. La peor separación sería en 5 clústers.

Aun así, el Índice de Silueta para 2 clústers es muy bajo (aprox un 0.07), lo que indica que, en verdad, no se encuentran estructuras de clústers sólidas. La puntuación cercana a cero indica datos solapados (frontera).

- Tarea 2.5: Cuál es el número óptimo de clústers.

Habiendo analizado todos los Índices de calidad con los distintos modelos y distancias, lo más seguro sería decir que el número óptimo de clústers es 2.

Teniendo en cuenta principalmente los modelos con distancia euclídeas y coseno (vimos antes que eran las que mejor dividían los datos en los clústers), el valor más repetido de número óptimo de clústers es 2, aunque es cierto que este número difiere en ocasiones, principalmente empleando la distancia coseno y evaluando con Davies Bouldin.

En verdad, tampoco tenemos otro candidato tan claro como este al número óptimo de clústers.

Como hemos dicho anteriormente analizando el Índice de Silueta, no tenemos estructuras sólidas por lo que este método de clustering jerárquico no funciona bien con nuestros datos, por lo que tendremos que probar con otros más adelante para conseguir mejores resultados. Por lo que no nos vamos a molestar en intentar encontrar una interpretación de esta separación debido a lo mala que es.

- Tarea 2.6: Analiza, con clustering, las zonas con mayor índice de siniestralidad para cada tipo de vehículo.

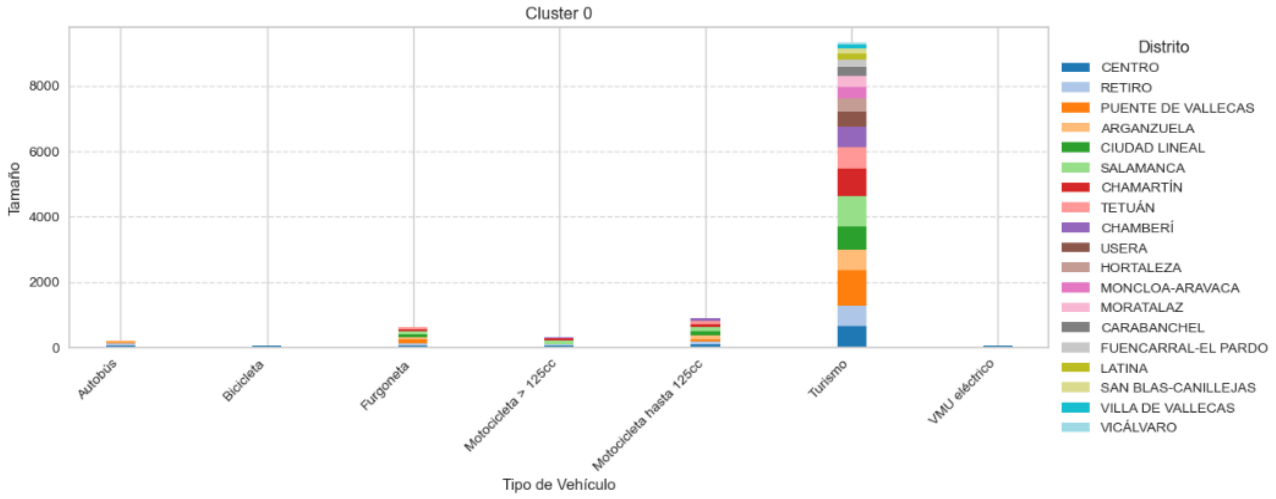
Para ello vamos a crear 2 funciones que muestren las zonas con mayores accidentes por cada tipo de vehículo, una con agglomerative clustering y otra con BIRCH.

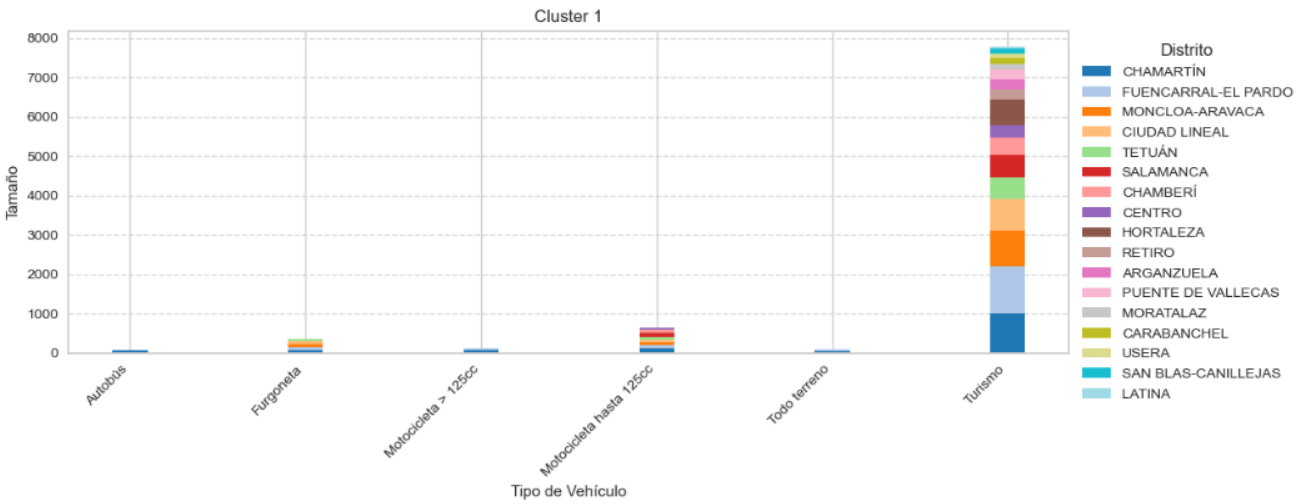
Vamos a utilizar el dataset preprocesado, y también el que no tiene las columnas categóricas pasadas a one-hot-encoding, ya que necesitamos los distritos y tipos de vehículos.

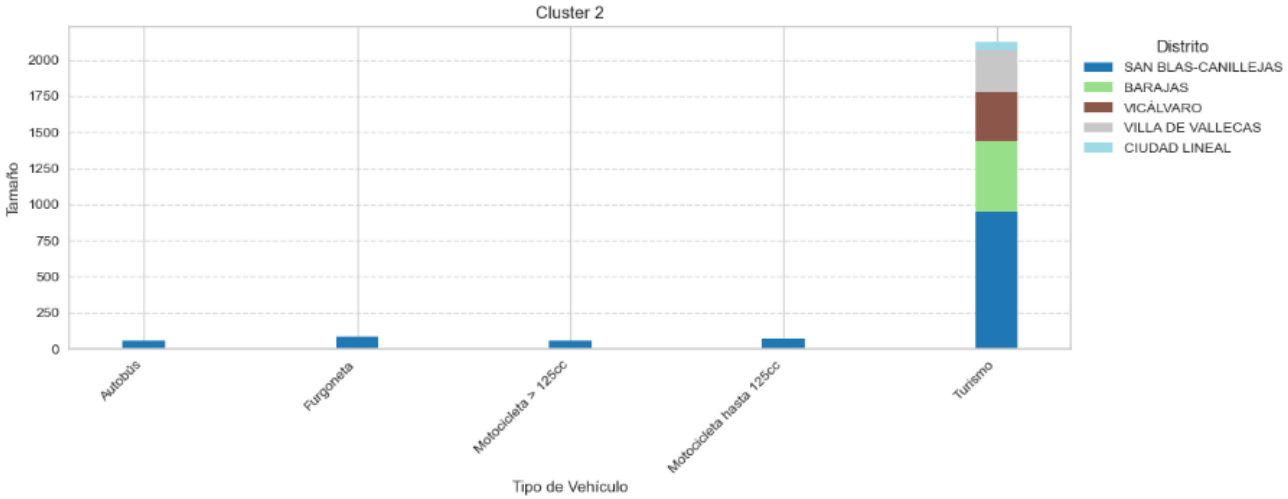
Vamos a emplear 5 clusters ya que este es el número que detecta BIRCH y hacer sólo 2 clústers no nos serviría mucho para esta tarea puesto que, aparentemente no separa los datos según un criterio visible. Si lo hacemos con 5 clústers sí que podemos encontrar una separación marcada que comentaremos a continuación.

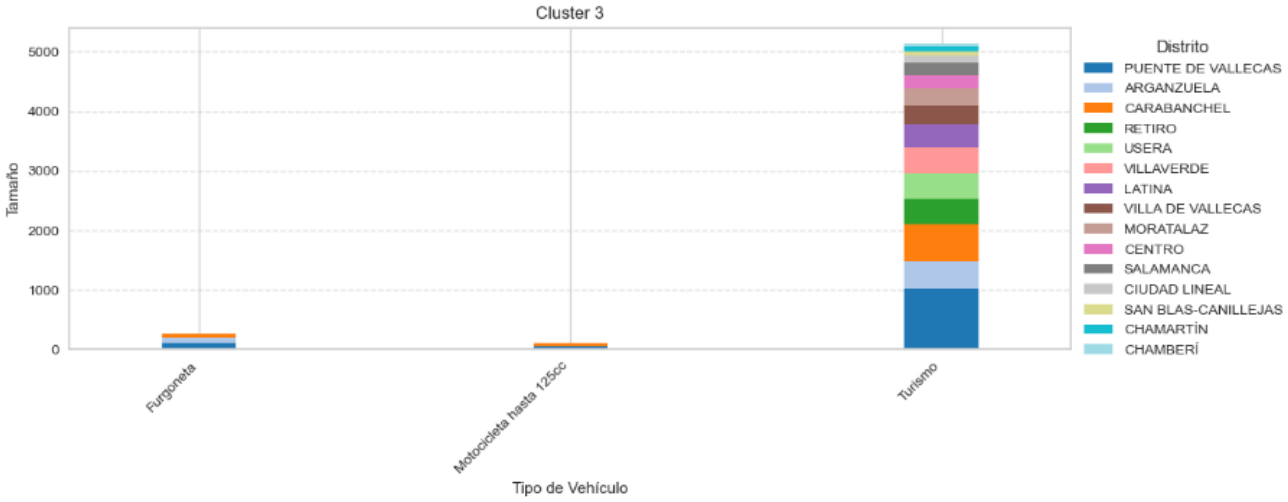
Hemos hecho el clustering con Agglomerative Clustering y con BIRCH, y la división entre clústers sale mejor con BIRCH, así que vamos a visualizar estos resultados. Con Agglomerative Clustering, la división en distintos clústers es principalmente por el tipo de vehículo, puesto que sale algún clúster con un único tipo (turismo principalmente) aunque el turismo sigue apareciendo en el resto de clústers puesto que es el vehículo que más involucrado está en accidentes.

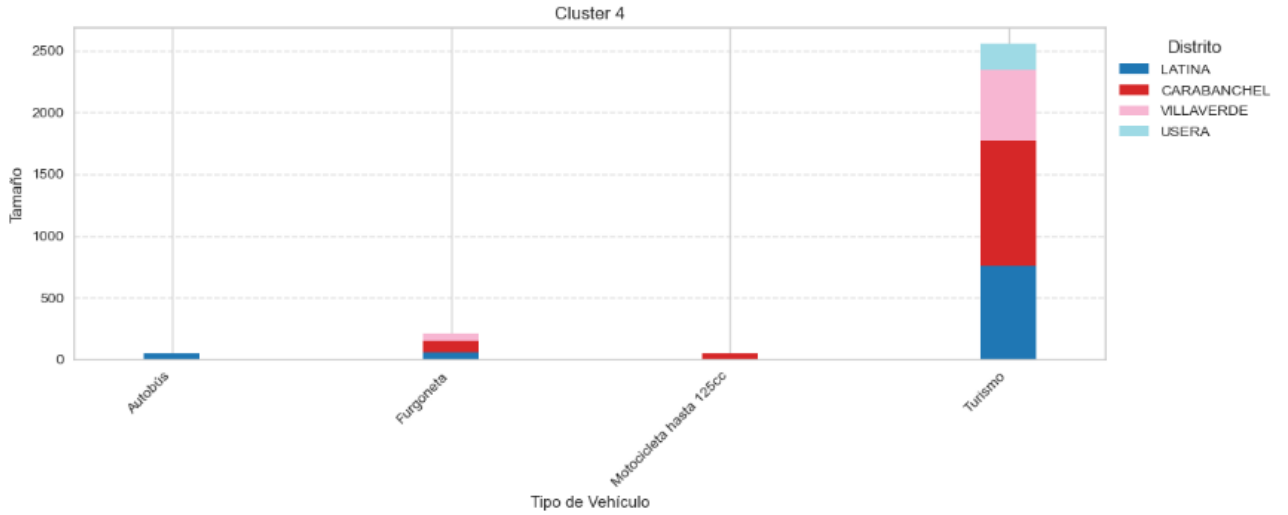
Vamos a ver los resultados con BIRCH:











BIRCH hace una mejor separación en clústers por distritos, por ejemplo, en el clúster 4 vemos datos únicamente de los distritos de Latina, Carbanchel, Villaverde o Usera, los cuales son barrios colindantes. Esto sucede también en el clúster 2, que contiene distritos del este de Madrid.

El turismo al ser el vehículo más utilizado también es el que más accidentes tiene, vemos aparece predominante en todos los clústers.

La bicicleta, solamente aparece en accidentes del distrito centro, al igual que sucede con el VMU eléctrico, pues son vehículos que se utilizan mucho por alquiler en esa zona y no están disponibles en las afueras de la ciudad.

Los accidentes de furgoneta vemos que nunca suceden en el distrito Centro ni en los cercanos a él debido a que las furgonetas no circulan por esas zonas, sino principalmente por el extrarradio y zonas más industriales, como tiene sentido, ya que se utilizan para movimiento de mercancías.

Los accidentes de todo terreno aparecen por Fuencarral-El Pardo debido a que es una zona de campo y es más transitada por vehículos equipados para ese tipo de terreno, estos no aparecen por el centro.

Los de autobús los vemos en distritos como Latina, San Blas-Canillejas o Puente de Vallecas, distritos por los que pasan muchas líneas de autobuses.

Finalmente, las motocicletas aparecen en distritos de todo tipo como Chamartín, Retiro, Carabanchel, Moncloa-Aravaca... ya que son vehículos muy utilizados en la ciudad por su tamaño y sencillez de conducción. Además, son probablemente los más propensos a tener accidentes en los que haya heridos, principalmente el conductor, debido a la exposición de este.

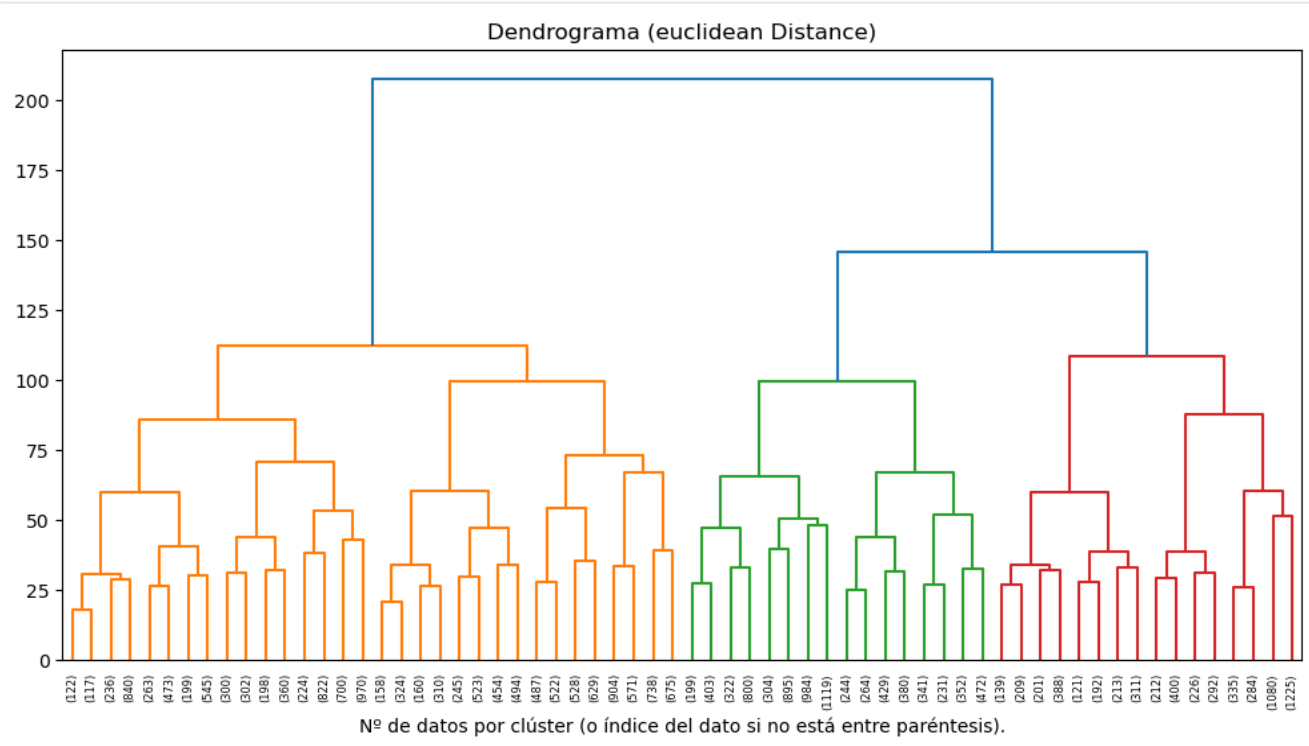
- Aplicación de algoritmos de clustering particional (en 2_particional.ipynb)
  - Tarea 3.1: Realiza el pre-procesamiento necesario para poder aplicar algoritmos de clustering particional

Ya teníamos el preproceso hecho de la tarea anterior, pero algoritmos como K-means necesitarían un paso extra, una eliminación de ruido; ya que son especialmente débiles ante este. Así, nuestro preproceso extra será el estudio y eliminación de valores atípicos por medio de un estadístico. Desde un principio probamos varias combinaciones como borrar todo lo que esté fuera de cuantil 1 y 3, pero eso solo nos devolvía una tabla vacía; fuimos agrandando los valores hasta el percentil 2 y 98, que nos devolvía poco más de la mitad de los datos; selección con la que finalmente nos quedamos para trabajar y que nos dejaría con una hipótesis a ser comprobada en futuros estudios; los datos son de naturaleza muy extrema y están fuertemente dispersos.

- Tarea 3.2: Establece el número más adecuado de clústeres para el dataset proporcionado. Ayudate de los metodos vistos (al menos 2) en la asignatura, asi como de gráficas para justificar la decisión. Compara los resultados que obtienes con cada métododo.

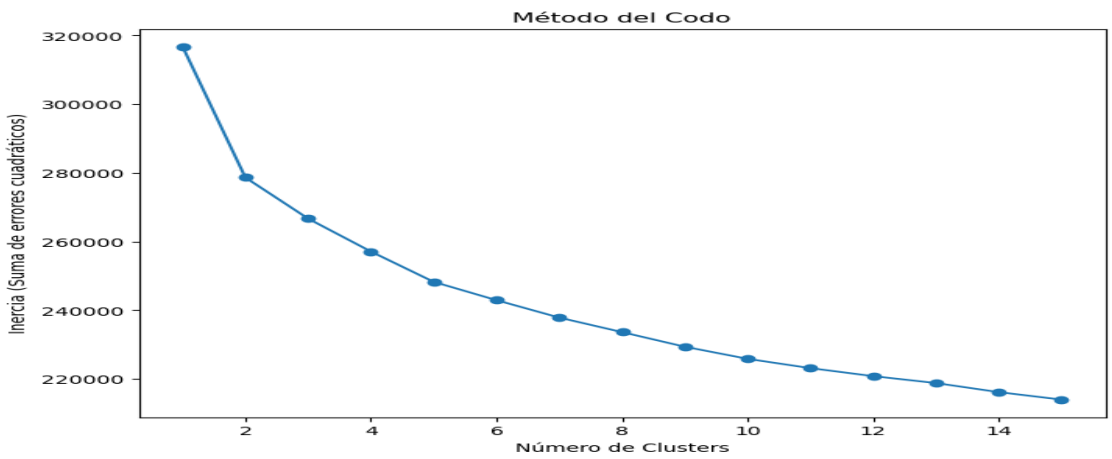
Para maximizar nuestras posibilidades de encontrar el número de clústeres que mejor se ajuste a nuestros datos, empleamos los 3 métodos que conocemos:

- Dendrograma: ya utilizado previamente para visualización con diferentes tipos de distancias y algoritmos, empleamos el pre implementado de scipy para obtener el siguiente gráfico:

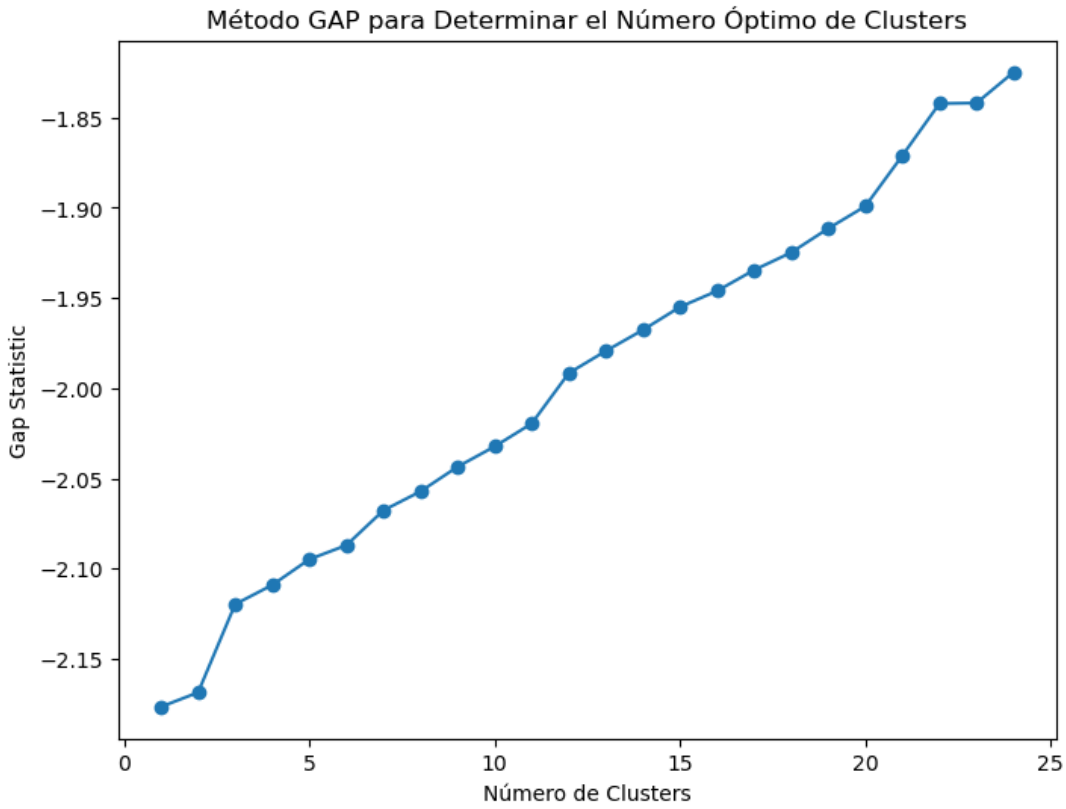


El cual nos deja con resultados un tanto ambiguos, pues depende de donde empecemos a contar sus ramas, las posibilidades varían de 2 a 21 (o influso más) clústeres diferentes, asique nos apoyamos en los siguientes métodos para tratar de acertar el número optimo de agrupamientos.

- Método del codo: estudiamos la inercia en un rango de 1 a 15 clusters empleando el algoritmo Kmeans, obteniendo que el mayor descenso de la inercia y por tanto el número óptimo de clusters dictado por este método sería en el número 2, lo cual puede visualizarse en el gráfico adjunto



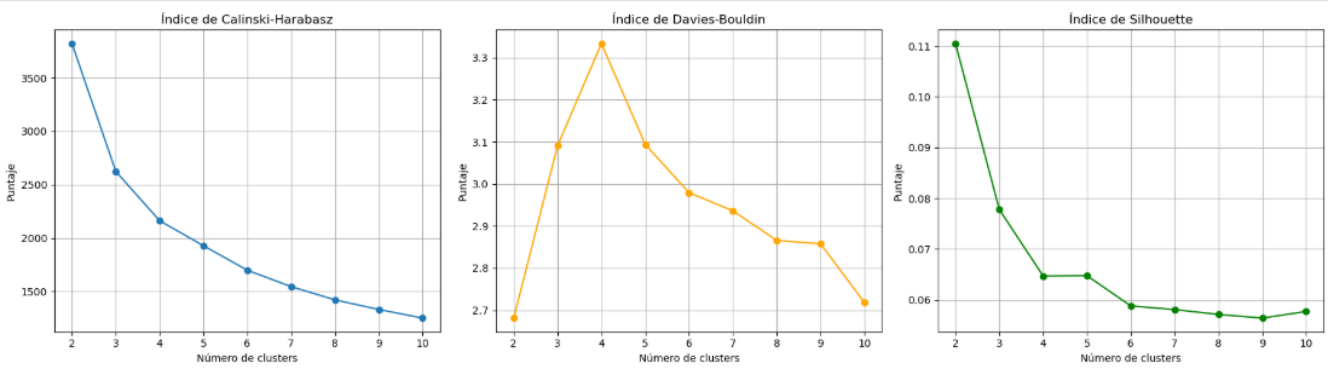
- Método GAP: método similar pero contrario al del codo, nos muestra un ascenso inigualado de dos a tres, luego podríamos decir que, según GAP, la mejor distribución de valores sería en 3 grupos distintos.



Luego, juntando los resultados, y por cercanía a la conclusión del trabajo previamente realizado, es casi seguro concluir que un número bajo de clusters, siendo exactos, tres, realizaría la mejor separación de datos posible

- Tarea 3.3 ¿Cómo varía la calidad del clustering con diferentes valores de 'k'?

Para esta tarea reciclamos código de un ejercicio previo, empleamos las métricas de calinski-harabasz, Davies-bouldin y el índice de silueta para analizar la calidad del agrupamiento en un rango desde el 2 hasta el 10



Como podemos observar, el algoritmo funciona mejor con indices bajos, con un rango entre el 2 y el 4, siendo el 3 el que mejor media tiene en estas métricas, lo que refuerza nuestra teoría de 3 clusters siendo el número más adecuado que hay.

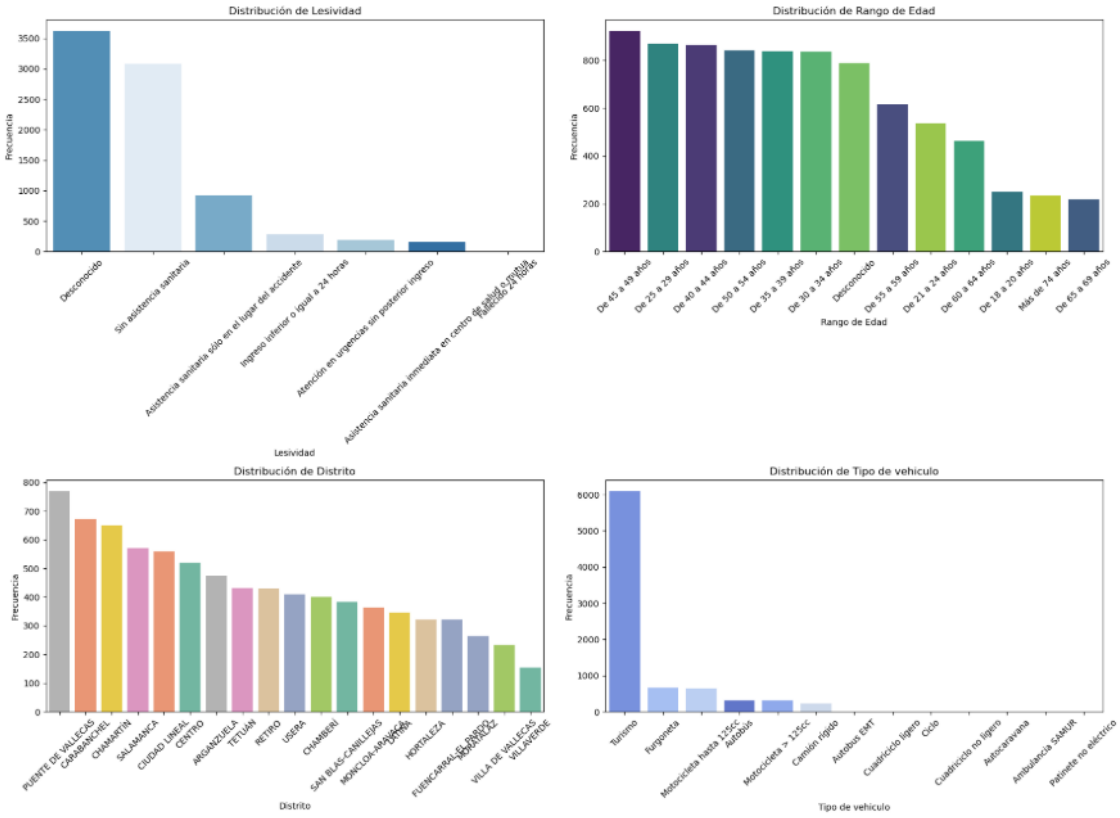
Cabe destacar que con todos los números de clusters el índice de silueta es extremadamente bajos, lo cual solo aumenta según sube el número de estos. Esto podría significar que el algoritmo no está entendiendo los datos o que los grupos que se forman no están muy densos o separados entre sí.

- Tarea 3.4: Con el número más adecuado de clusters, ayúdate de estadísticas para analizar a los viajeros incluidos en cada cluster.

Utilizando los índices de las entradas resultantes de cada uno de los 3 clusteres, hemos recuperado las entradas originales, antes de realizar todas las transformaciones que nos dieron nuestro dataframe limpio, y hemos hecho un estudio de: tipo de vehículo, el tipo de lesividad, el rango de edad y el distrito en el que tuvo lugar el accidente para observar alguna diferencia

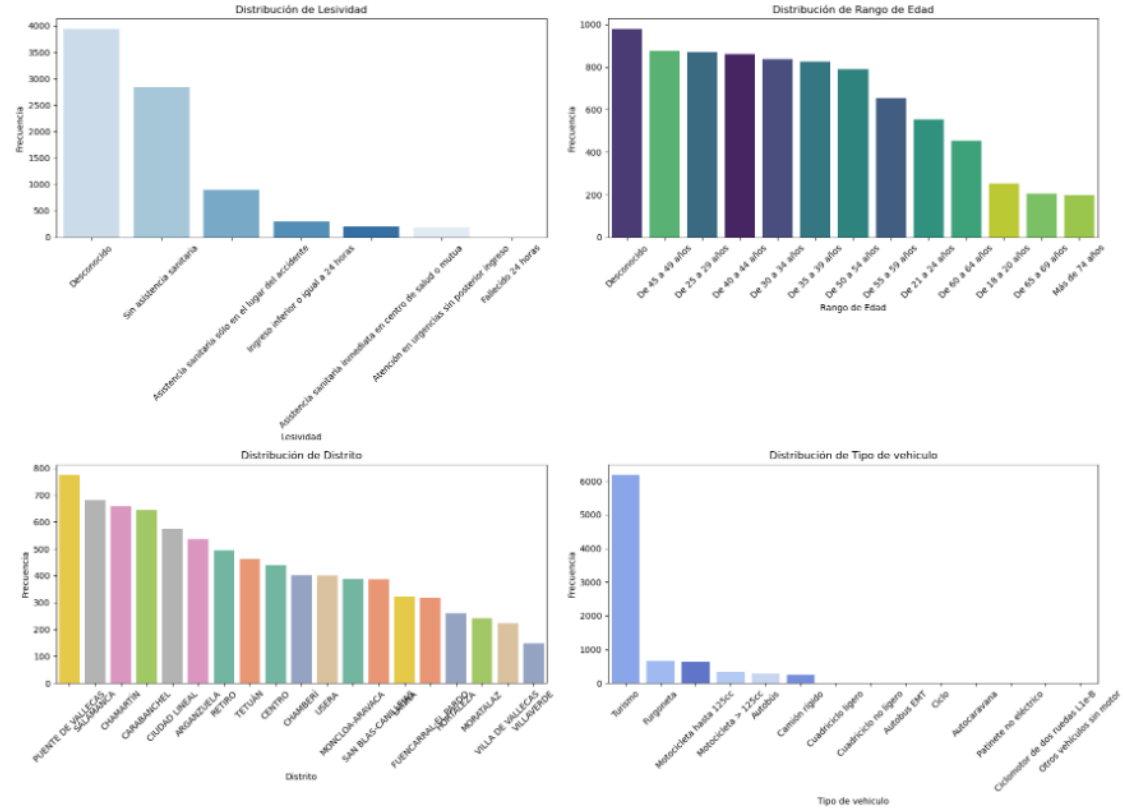
--- Gráficos del dataframe: DataFrame 1 ---

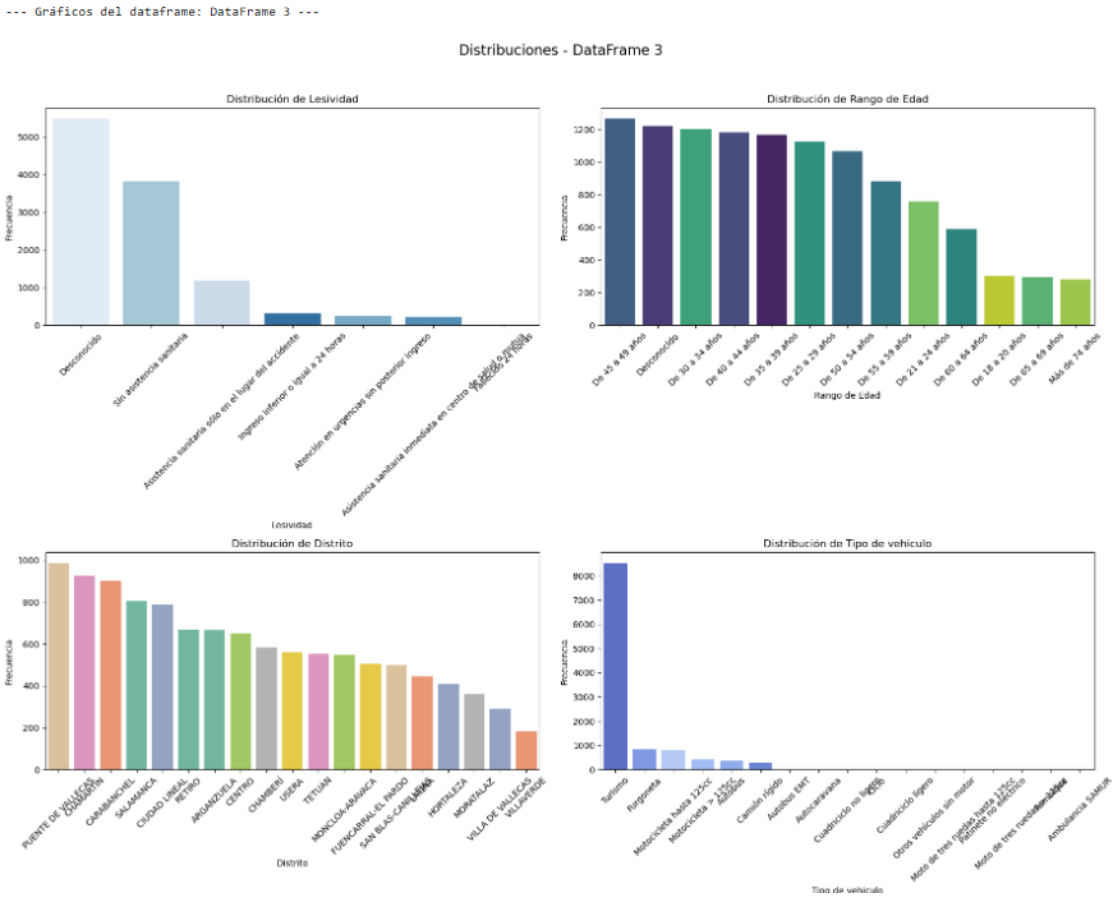
Distribuciones - DataFrame 1



--- Gráficos del dataframe: DataFrame 2 ---

Distribuciones - DataFrame 2





Podemos observar que los primeros elementos de todos los clusters son siempre parecidos; los vehículos son turismos, los accidentes son en puente de Vallecas, el rango de edad o indeterminado o 45 años... Luego es probable que para la agrupación se estén usando otros elementos que no hemos estudiado, como sus coordenadas, la presencia de estupefacientes (ya sea alcohol u otro tipo de drogas), el tipo de persona (conductor, peatón…) o el género.

- Tarea 3.5: **:** Compara los resultados obtenidos con K-means y el clustering aglomerativo/jerárquico. Discute las ventajas y desventajas de cada método en diferentes tipos de datos.

Los resultados son realmente parecidos, pues si vemos las gráficas, las de clustering aglomerativo son casi identicas a las que se generan con K-means. Si miramos Birch, este también nos indica lo mismo, que el número de clusters es bajo (de dos o tres).

- K-means: tiene ventaja en grupos pequeños de datos, siempre y cuando sean densos y de tamaños más o menos parecidos. En este caso, los datos son grandes pero al haberse

borrado tantos solamente usando los cuantiles 2 y 98, creemos que los datos están fuertemente dispersos, y de haber grupos, serán pequeños. Otra posible explicación sería que de haber clusters, estos sean de tamaños muy variantes o sean no convexos.

- Jerarquico: al igual que el anterior, son gravemente afectados por el ruido, de ahí que nos salgan resultados tan pobres y un número de clusters tan extraño como 2 o 3. Estos trabajan de forma pobre cuando el número de datos o las dimensiones escalan, por lo que era de esperar un comportamiento débil.

- Aplicación de algoritmos de clustering de densidad (en 3_densidad.ipynb)

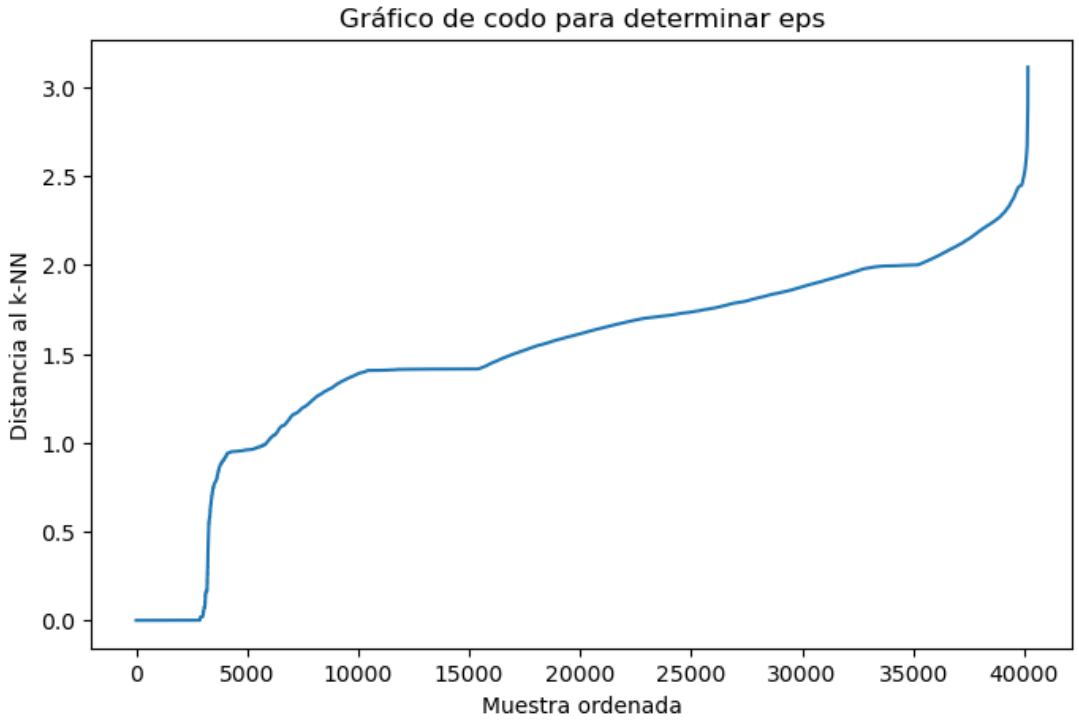
- Tarea 4.1: Realiza el pre-procesamiento necesario para poder aplicar algoritmos de densidad.

Utilizaremos el preprocesamiento realizado anteriormente, ya que también resulta adecuado para este caso, pero añadiremos algunos ajustes. En particular, implementaremos una reducción de dimensionalidad utilizando algoritmos como PCA. Para ello, conservaremos el 95% de la varianza explicada, con el fin de minimizar la pérdida de información. Esto es especialmente importante, ya que en espacios de alta dimensionalidad, los algoritmos basados en densidad tienden a no desempeñarse de manera óptima. Además, la selección de parámetros como el epsilon y el mínimo de puntos se complica en sobremanera en escenarios parecidos a estos.

- Tarea 4.2: Establece el radio (eps) y número de puntos mínimo número más adecuado de clusters para el dataset proporcionado.

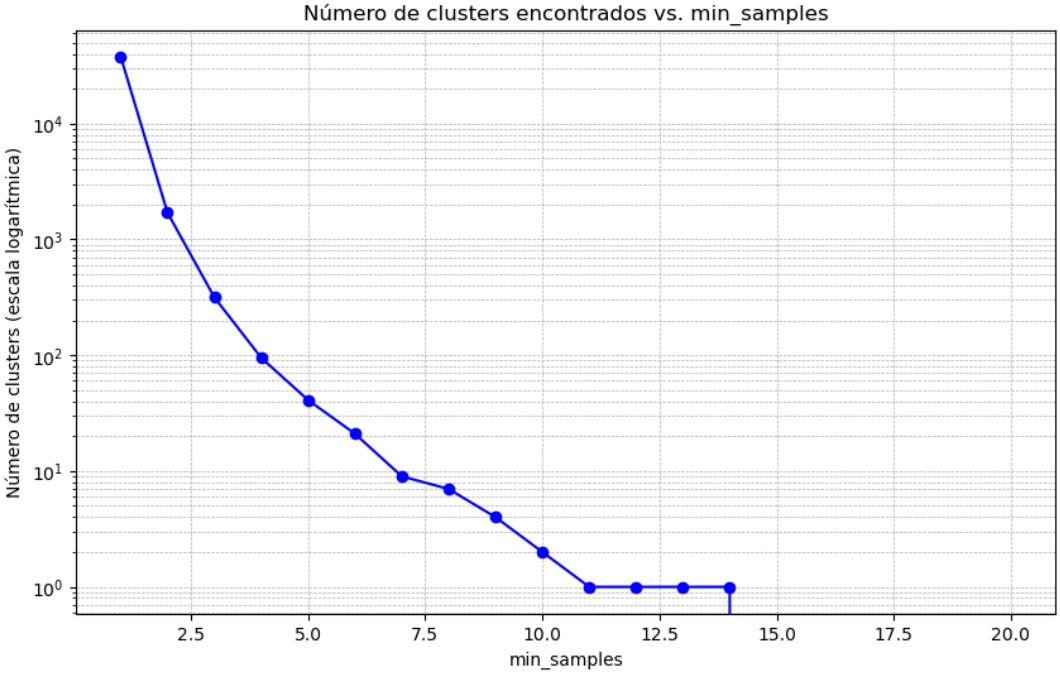
Esta tarea se dividirá en dos pasos:

- 1: Establecer el radio epsilon: Estudiaremos un gráfico de codo, el cual proviene del algoritmo KNN, ordenadas las distancias entre los puntos de menor a mayor. Los resultados que tengan valores muy bajos nos dirán como de densamente poblado están los gráficos. Si la pendiente cambia, sabremos que la distancia entre puntos ha aumentado; por lo que tomaremos ese valor (aproximadamente) y ese será nuestro epsilon



Podemos observar que la pendiente empieza a cambiar algo más abajo de 1 y el cambio de pendiente se acentúa mucho tras este, tomemos un epsilon de 0.9, pues a partir de este rango los puntos están muy separados

- 2: Establecer el número de puntos minimo mas adecuado: Del análisis y visualización realizados en el apartado 1, sabemos que hay 21 clusters, luego haremos un bucle que prediga los datos, el que más se acerque a este número será el más adecuado. El siguiente grafico está en escala logarítmica para poder apreciar todos los puntos:



- Tarea 4.3: ¿Cómo varía la calidad del clustering con diferentes valores de 'eps' y de minpoints?

Para afrontar esta tarea, estudiamos valores cercanos a los ya elegidos y vimos cómo afecta esto al número de clusters. El método de evaluación es el mismo, vemos cuan cercano es el resultado al numero de clusters que buscamos.

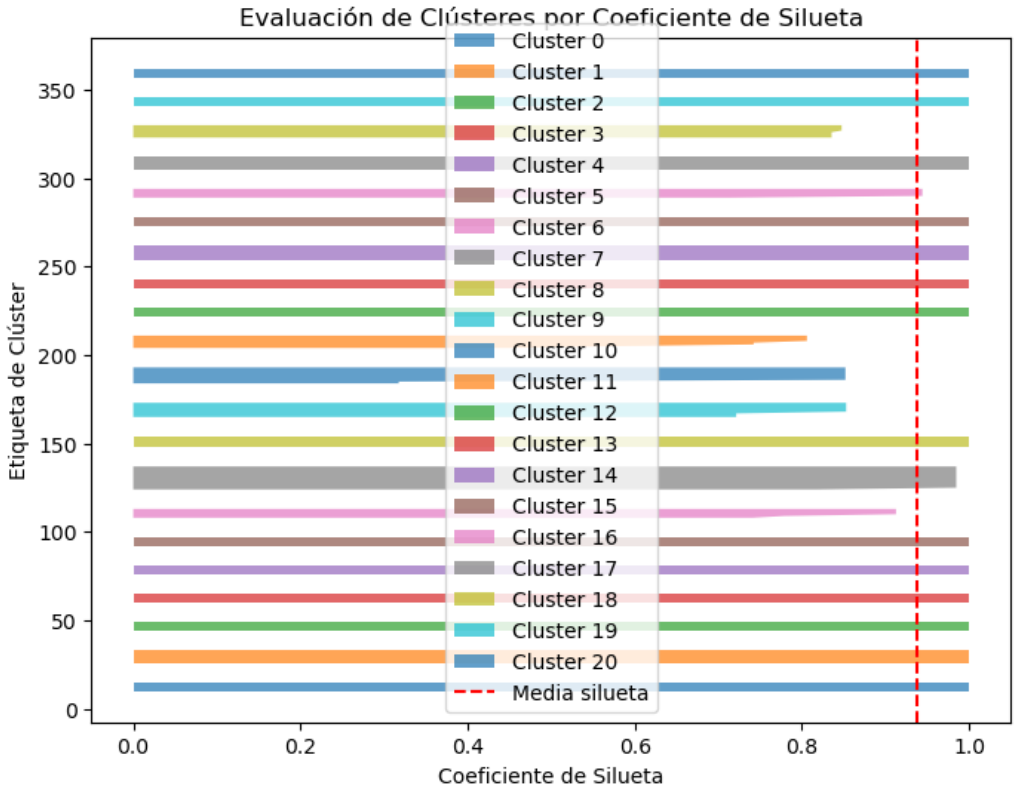
```
-----------------------------------------
Número de clusters encontrados para eps=0.5 y min=3: 275
Número de clusters encontrados para eps=0.5 y min=4: 88
Número de clusters encontrados para eps=0.5 y min=5: 35
Número de clusters encontrados para eps=0.5 y min=6: 17
Número de clusters encontrados para eps=0.5 y min=7: 6
Número de clusters encontrados para eps=0.5 y min=8: 5
Número de clusters encontrados para eps=0.5 y min=9: 2
-----------------------------------------
Número de clusters encontrados para eps=1.0 y min=3: 525
Número de clusters encontrados para eps=1.0 y min=4: 148
Número de clusters encontrados para eps=1.0 y min=5: 65
Número de clusters encontrados para eps=1.0 y min=6: 28
Número de clusters encontrados para eps=1.0 y min=7: 13
Número de clusters encontrados para eps=1.0 y min=8: 9
Número de clusters encontrados para eps=1.0 y min=9: 6
-----------------------------------------
Número de clusters encontrados para eps=1.5 y min=3: 1893
Número de clusters encontrados para eps=1.5 y min=4: 824
Número de clusters encontrados para eps=1.5 y min=5: 377
Número de clusters encontrados para eps=1.5 y min=6: 186
Número de clusters encontrados para eps=1.5 y min=7: 96
Número de clusters encontrados para eps=1.5 y min=8: 49
Número de clusters encontrados para eps=1.5 y min=9: 35
-----------------------------------------
Número de clusters encontrados para eps=2.0 y min=3: 510
Número de clusters encontrados para eps=2.0 y min=4: 298
Número de clusters encontrados para eps=2.0 y min=5: 205
Número de clusters encontrados para eps=2.0 y min=6: 125
Número de clusters encontrados para eps=2.0 y min=7: 100
Número de clusters encontrados para eps=2.0 y min=8: 76
Número de clusters encontrados para eps=2.0 y min=9: 54
-----------------------------------------
```

Basándonos en estos resultados y el gráfico de la tarea anterior, vemos que, para radios menores, salen menor número de clusters, esto se debe al comportamiento del algoritmo, al haber una distancia tan pequeña, muchos puntos pueden quedar fuera de cualquier cluster y que sean catalogados de ruido; mientras que con valores por encima de este, el número de clusters aumenta, pues se pueden considerar parte de muchos más clusters. Por otro lado, si miramos min_samples; observamos que cuando este crece el número de clusters decrece, esto se debe a que obligamoms a que haya más puntos muy cercanos para poder clasificarlos como clusters, luego el número de datos tratados de ruido aumenta de forma muy rápida. También deberíamos decir que nuestra hipótesis acerca de los datos desperdigados es probablemente cierta, pues a más radio y menos puntos cercanos, el número de clusters se dispara de manera exponencial.

- Tarea 4.4 Utiliza por lo menos dos índices de calidad de clustering y analiza sus resultados.

En este apartado estudiamos el índice de silueta (cuanto más cercano al 100% mejor) para comprobar como de similares son los puntos de un cluster entre sí;

encontrándonos con casi un 94% de coeficiente medio. Al ser un número tan alto, quisimos estudiar la varianza que había dentro de los diferentes grupos y encontramos lo siguiente: no existe gran varianza desde la media, los datos solo varían hasta un 20%, por lo que el agrupamiento, en términos de la métrica silueta es sólida.



También usamos la métrica de Davies-Bouldin, para estudiar la relación que existe entre la distancia inter e intra cluster (cuanto más bajo mejor), obteniendo cerca de un 16%, lo que nos dice que los grupos están bien separados y compactados, siendo poco cercanos o superpuestos; luego podría decirse que nuestra separación es ciertamente buena y que tal vez, el haber reducido la dimensionalidad ha impactado positivamente en nuestros datos. También podemos deducir que nuestros datos no son de forma convexa (de ahí que los algoritmos jerárquicos funcionen tan mal), si no que son grupos densos en los que existe una cantidad muy grande de ruido (pues al estudiar los labels que asigna el algoritmo, vemos que la lista no tiene una longitud muy extensa)

- Tarea 4.5 ¿Cuál es el número óptimo de clusters? ¿por qué?

La lógica nos lleva a pensar que 21, pues si miramos el problema tenemos 21 distritos diferentes y tras estudiar los parámetros y los resultados del algoritmo, concluimos que la separación es buena. Aunque no podemos decir exactamente que sea esta, así que vamos a hacer un estudio, agrupando el código anterior en una función que nos muestre los resultados variando un poco los parámetros.

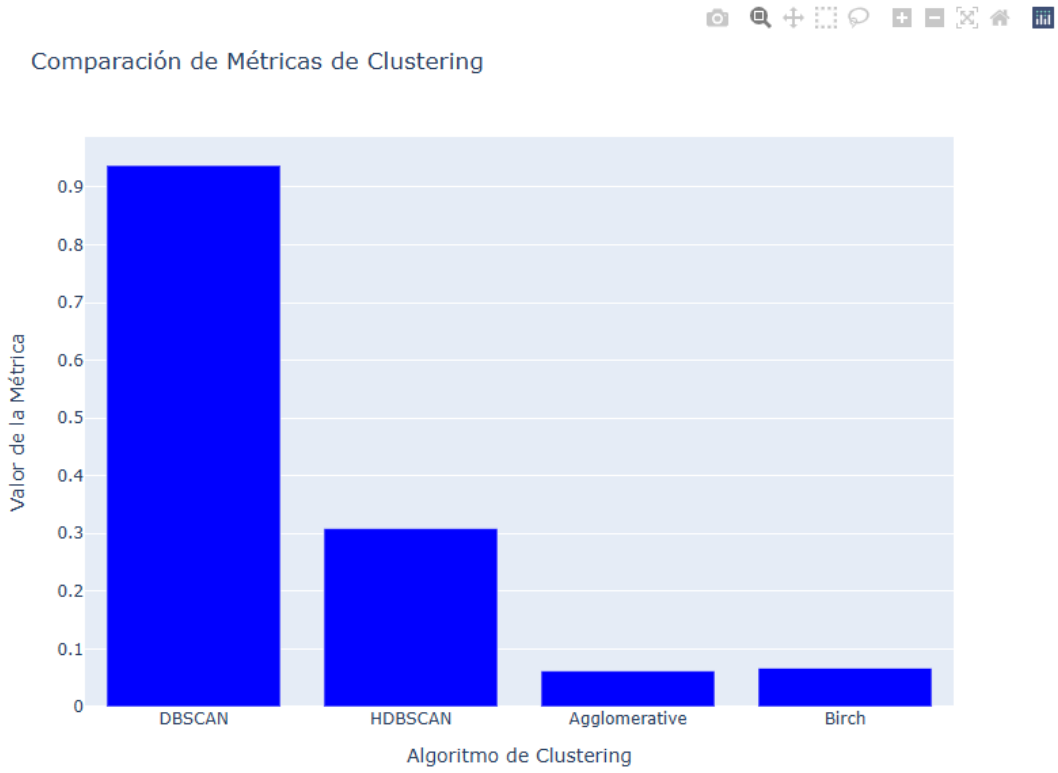
```
--------------------------------------
Resultados para el epsilon 0.8
Resultado para el min_sample 5
Silueta:0.9496657043032491, dbi:0.18273306091791783, clusters:41
Resultado para el min_sample 6
Silueta:0.9483446703110546, dbi:0.14680410265275828, clusters:20
Resultado para el min_sample 7
Silueta:0.9204364505082833, dbi:0.18257603498014985, clusters:8
--------------------------------------
Resultados para el epsilon 1
Resultado para el min_sample 5
Silueta:0.838629484653652, dbi:0.3337445287742009, clusters:65
Resultado para el min_sample 6
Silueta:0.8607781628117214, dbi:0.2980893817942101, clusters:28
Resultado para el min_sample 7
Silueta:0.8276635902380022, dbi:0.3234043164467148, clusters:13
```

Podemos observar que el más parecido a nuestro resultado (y el mejor de toda la prueba), de acuerdo con lo anteriormente explicado, es el epsilon 0.8 con min_sample 6, que da 20 clusters. Este da un resultado muy similar al nuestro, con unos parámetros escasamente mejores que los ya obtenidos, por lo que es casi seguro decir que alrededor de 20 grupos diferentes serían la mejor opción.

- Aplicación de otros algoritmos (en 5_otros.ipynb)

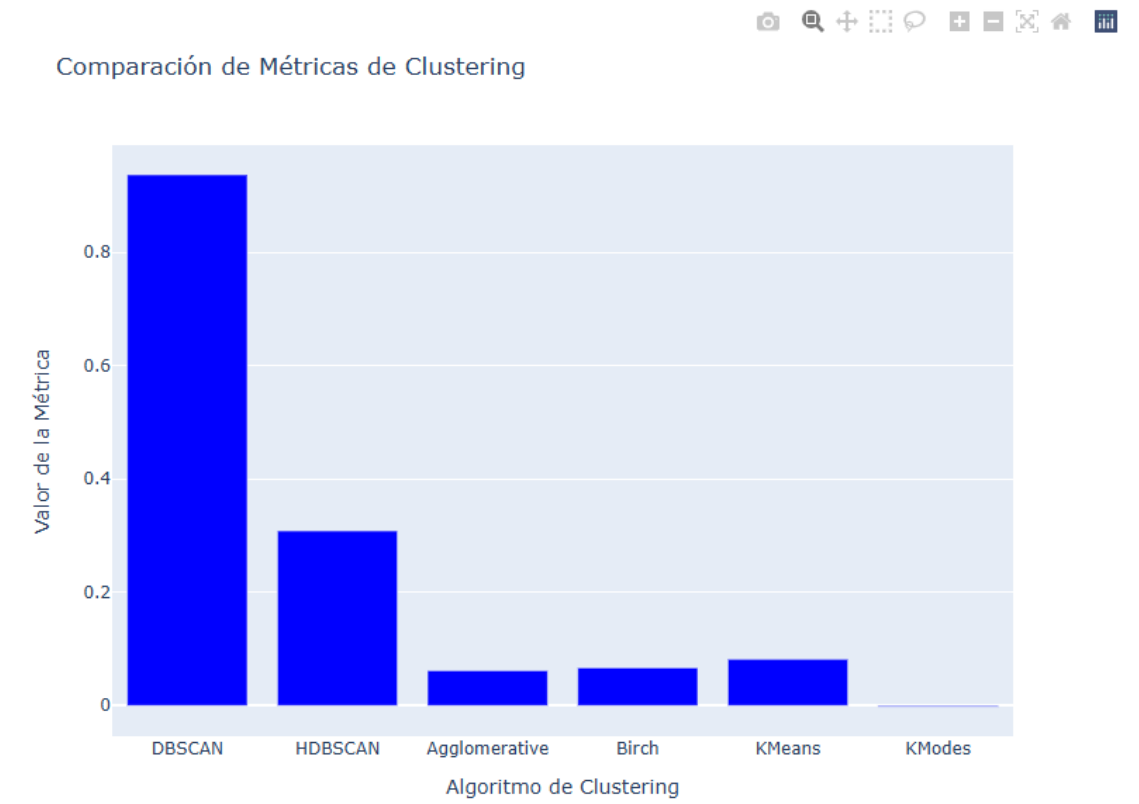
- Tarea 5.1: Emplea otros algoritmos como HDBScan y compara con otros algoritmos su rendimiento.

HDBSCANS es la ampliación jerárquica de DBSCANS, luego, para realizar la comparación, usaremos los algoritmos y una métrica que usamos tanto en el primer como tercer notebook, el índice de silueta (que nos explicará como de bien formados están los grupos). Cabe esperar que al también funcionar por densidad (propiedad la cual, por pruebas anteriores conocemos que poseen nuestros datos), hdbscans sea bastante mejor algoritmo que los primeros, y algo mejor que dbscans al ser una mejora a este. Para garantizar una mejor actuación de los algoritmos de densidad, volveremos a aplicar el preprocesado que ya hicimos en su notebook, realizando un pca al 95% de varianza explicada.



Como ya vimos con anterioridad, DBSCAN funciona a la perfección pues nuestros clusters son densos y compactos, superando a los jerárquicos básicos por su resistencia al ruido. Lo que no esperábamos era este desempeño tan pobre por parte de su extensión HDBSCAN. Esto puede deberse a que el parámetro de este algoritmo necesita un mejor tuning y estudio, o que la densidad de los clusters no sea variable y la complejidad añadida de este algoritmo sea incapaz de comprender la forma de los datos.

- Tarea 5.2: Emplea otros algoritmos como K-modes y compara con otros algoritmos su rendimiento.



Como conclusiones finales:

- DBSCAN funciona tan perfectamente por la forma de los datos en los clusters, siendo estos densos y bien diferenciados a lo largo del espacio, además de tener díametros bastante similares

- HDBSCAN podría ofrecer mejores resultados, pero parece que necesita un ajuste más preciso de los parámetros (tuning), como el tamaño mínimo de los clusters (min_cluster_size) y el tamaño mínimo de la muestra (min_samples), para adaptarse mejor a la estructura de los datos.

- Agglomerative clustering y birch no logran un desempeño óptimo porque, al ser métodos jerárquicos, tienden a ser más efectivos cuando los clusters tienen una estructura claramente jerárquica o formas regulares, la cual, como ya hemos comprobado, no es el caso.

- K-Means muestra un desempeño limitado porque los clusters tienen formas irregulares o porque hay mucho ruido en los datos, lo cual podría influir en los resultados.

- K-Modes, al estar diseñado para datos categóricos, tiene un desempeño nefasto debido a que los datos actuales ya fueron normalizados y tienen un enfoque continuo. Esto hace que K-Modes no sea adecuado para este caso, ya que su algoritmo no puede aprovechar la estructura de los datos proporcionados.