UNIVERSIDAD DE GRANADA

PRÁCTICA 2 INTELIGENCIA DE NEGOCIO

Visualización y segmentación

David Alberto Martín Vela

Grupo 2 de prácticas
davidmv1996@correo.ugr.es
Doble Grado Ingeniería Informática y Matemáticas

Curso 2020-2021

Contents

List of Figures

List of Tables

Chapter 1

Visualización

Veremos el uso de visualización para analizar un dataset, concretamente el de mamografías proporcionado en la práctica anterior. Se aplicarán los conceptos del seminario de visualización se muestrarán distintas visualizaciones y se procederá a un análisis de las mismas. [1]

1.1 Visualización de las medidas

En esta sección mostramos de forma visual los resultados de las medidas para cada preprocesamiento de la práctica anterior, analizando las diferencias entre modelos dado un preprocesamiento, y también analizando las diferencias entre preprocesamientos de un mismo modelo. Partimos de los casos bases sin preprocesamiento de la figura [1.1], haremos mediciones utilizando como medida f1 utilizando validación cruzada de 5 particiones (debido a la importancia en el caso de detección de cáncer de medir utilizando esta medida en vez de una accuracy normal como se ha explicado en la práctica anterior) y entre los distintos tipos de preprocesamiento distinguiremos maneras de tratar con los datos perdidos y con la escala de los datos. Entre ellas tendremos directamente eliminar todos los valores, **Drop missing** , la sustitución por la media, **Mean Imputation**, (solo valido para variables numéricas), mediana, **Median Imputation** (la mediana es un estimador má robusto para datos con variables de gran magnitud que podrían dominar los resultados, también conocido como long tail) y moda, **Mode Imputation** (solo válido para variables categóricas), del tipo k-vecinos más cercanos, **KNN Imputation** que consisten en buscar los k valores más próximos al que queremos sustituir. Una vez identificados se puede sustituir por la media (algoritmo k-medias) o por la moda (algoritmo k-modas) e Iterative Imputer, **Iterative Imputation** (usa regresión lineal round-robin, modelando cada característuca con valores perdidos como una función de otras características, la version implementada asume variables gaussianass, hay que considerar transformarlas para que parezcan mas normales y que mejore potencialmente el rendimiento). Cuando hablamos de normalizar nos referimos a escalar una variable para que tenga valores

[1] Todo lo relevante a este caso de estudio se recopila en la carpeta de notebooks en el fichero llamado **1_visualization_mamographic_case_1.ipynb**, las imagenes generadas pueden encontrarse en **notebooks/figures/1_visualization**

entre 0 y 1, mientras que la estandarización transforma los datos para que tengan una media de cero y una desviación típica de 1. Aunque vemos cada configuración para todos los valores, habría que estudiar cada característica por separado, por ejemplo, no tiene sentido **Mean Imputation** para valores no numéricos.

También cabe comentar, del feedback recibido en la práctica anterior, la validación cruzada (CV) generalmente significa que divide algún conjunto de datos de entrenamiento en k piezas para generar diferentes conjuntos de entrenamiento / validación. Al hacerlo, se puede ver qué tan bien rinde un modelo (y es capaz de hacer predicciones) en diferentes muestras de un conjunto de datos de entrenamiento. En la práctica anterior hemos combinado los datos de prueba en nuestra validación cruzada, lo cuál es un **error**.

Durante el entrenamiento y el ajuste del modelo, no debería ver los datos de prueba. La idea es que reserve un conjunto de datos verdaderamente ”exógeno” (nunca utilizado durante el entrenamiento) para probar qué tan bien funciona al final. Si usamos los datos de prueba durante el entrenamiento, la información puede filtrarse desde los datos de prueba al modelo y ya no podrá demostrar la validez del modelo (porque la información de los datos de prueba se combinaron con el modelo). En esta práctica eso se ha corregido y se ha dividido el conjunto en datos de entrenamiento y datos de prueba (en un 80% - 20%) y se ha aplicado validación cruzada posteriormente con los datos de entrenamiento.

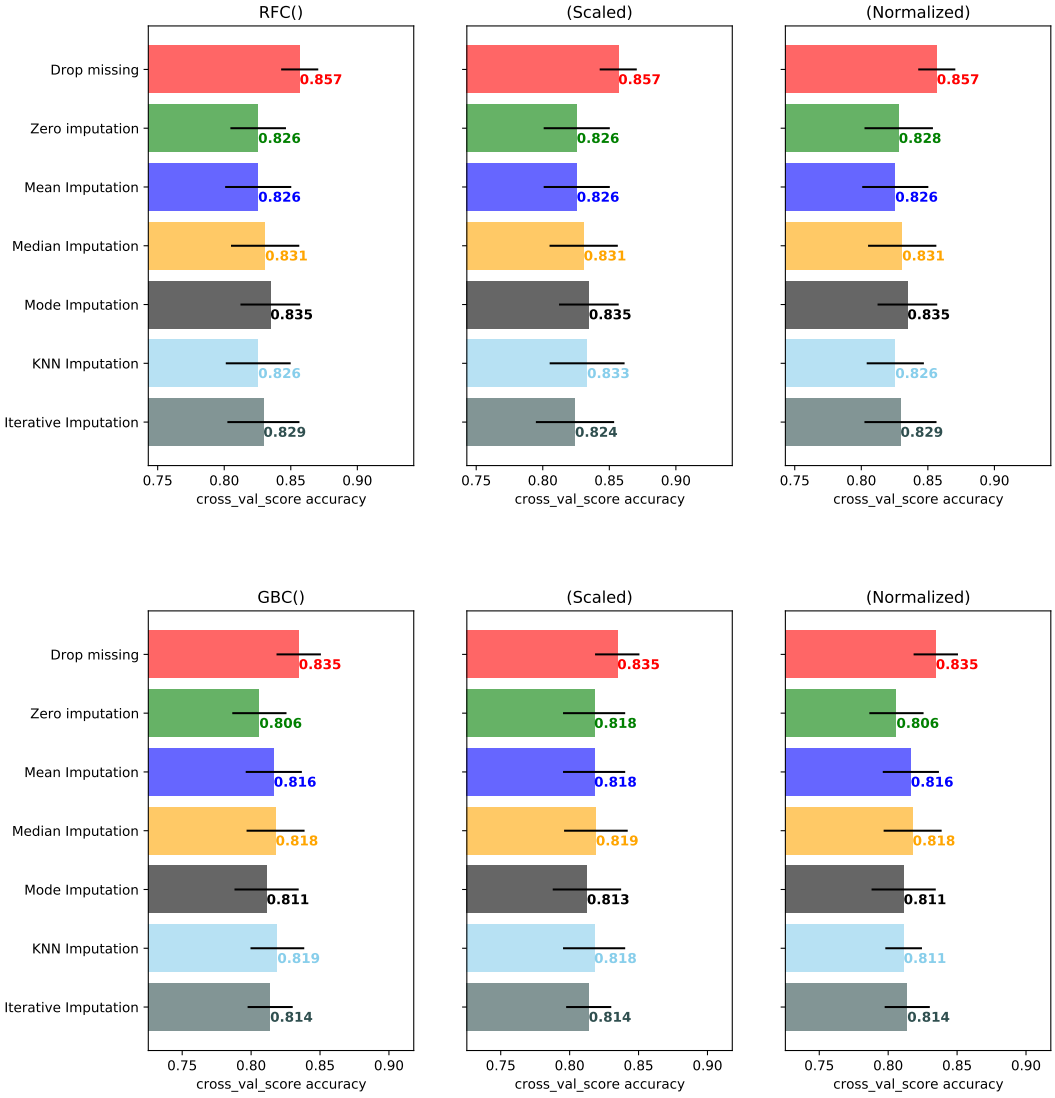
```
lr = LogisticRegression(C=0.01,penalty='l2', solver='newton-cg',
    random_state = 10)
knn = KNeighborsClassifier(n_neighbors = 6, metric = 'manhattan')
svc = SVC(C=1, gamma='auto', random_state=10)
rfc = RandomForestClassifier(max_depth=3, n_estimators=100, random_state=10)
gbc = GradientBoostingClassifier(n_estimators=1, max_depth=3, random_state
    = 10)
```

Figure 1.1: Estimadores sin preprocesamiento

En las figuras [1.2] y [1.3] vemos los resultados de nuestra validación cruzada de 5 particiones teniendo en cuenta como medida f1 score sobre los preprocesamientos comentados anteriormente. A priori, no se aprecia gran diferencia entre los tipos de imputaciones y los resultados obtenidos, si que se ve claro que escalar los datos siempre beneficia ya que la algunos de nuestros algoritmos utilizados se basa en distanciasm luego o beneficia o simplemente no perjudica como es el ejemplo de RandomForest. También cabe comentar que a veces obtenemos mejores resultados simplemente eliminando los valores perdidos, lo cuál hace pensar que nuestros valores producen mucho ”ruido”, incluso influye mucho más el tipo de escalado en nuestros datos. Aunque con algunos algoritmos la diferencia de escalar o no los datos es despreciable, en general se obtienen mejores resultados escalando nuestros datos (KNN, SVC y Logistic Regression, figura [1.2])

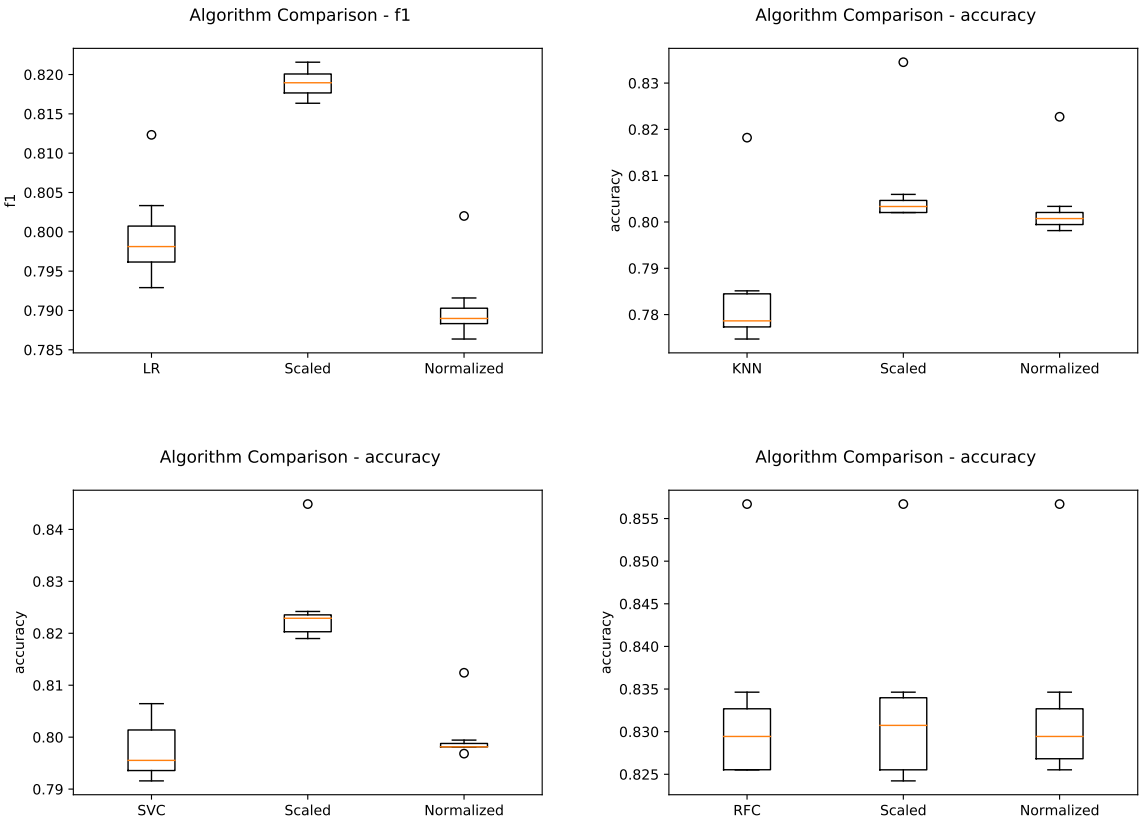
Figure 1.2: Preprocesamientos LR, KNN y SVC

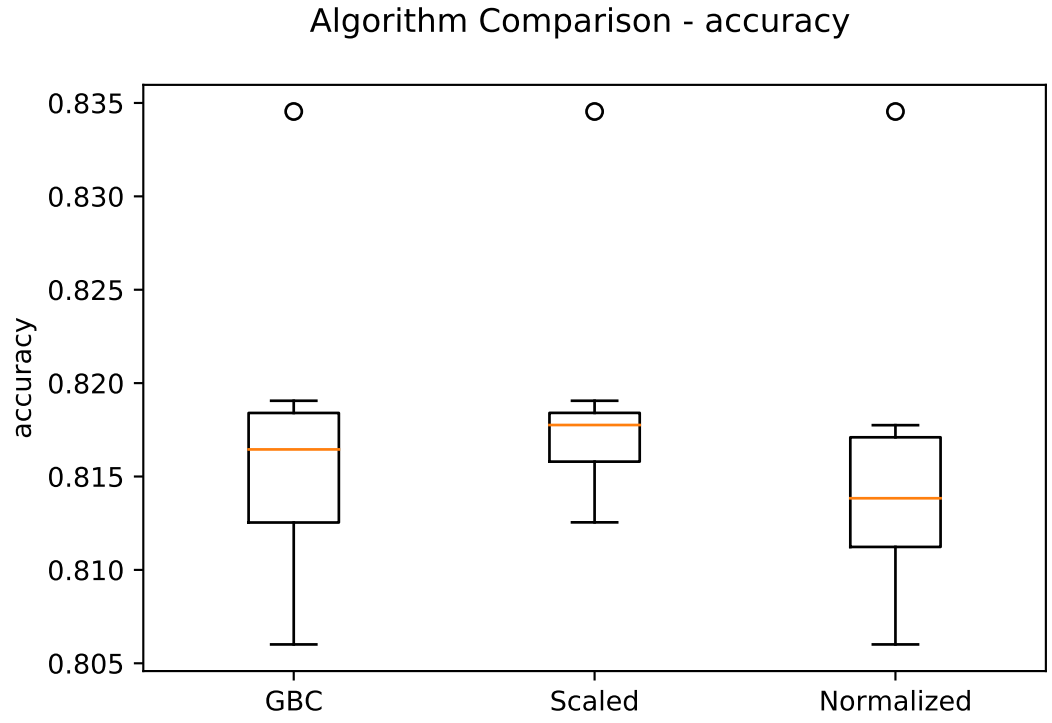
Figure 1.3: Preprocesamientos RFC y GBC



También mostramos un par de boxplots donde visualizamos las scores de nuestros algoritmos en la figura [1.4], la ventaja que tiene este boxplot frente a otro tipo de visualización, es que podemos ver sobre que intervalo de valores circula nuestro score, viendo que en algunos casos aunque no notamos una gran diferencia frente a la media, nuestro intervalo se reduce disminuyendo la varianza y por ende, siendo más preciso.

Figure 1.4: Boxplot accuracy LR, KNN, SVC, RFC y GBC





Como extra y del feedback recibido en la práctica anterior, también visualizamos un par de estimadores del RandomForestClasssifier en la figura [1.5], solo mostramos hasta 5 estimadores ya que no tendría sentido mostrar los 100, es para hacerse una idea de como se dividen las ramas con una profundidad máxima de 3, que es nuestra configuración óptima elegida.

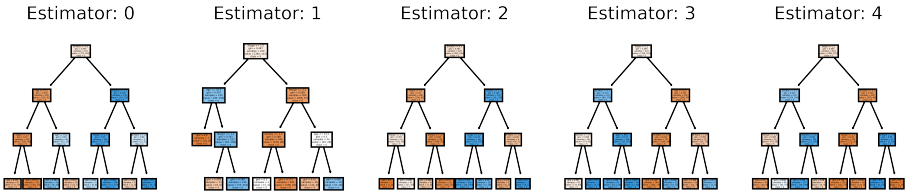


Figure 1.5: 5 estimadores RFC.

1.2 Gráficas de curva ROC

Inluimos una curva ROC de los distintos modelos presentados en la práctica anterior, pero con las condiciones y parámetros que mejor resultado nos da, y realizando validación cruzada para los resultados y cogiendo la media, es por eso que nos da una curva tan similar para todos los casos. Nuestras combinaciones de parámetros para cada estimador puede verse en la figura [1.6].

Figure 1.6: Mejores combinaciones de estimadores elegida

```
# LR (Best estimator)
Pipeline(steps=[('pre',ColumnTransformer(transformers=[('cat',Pipeline(steps=[
('imputer_cat',SimpleImputer(strategy='most_frequent')), ('onehot',
    OneHotEncoder(handle_unknown='ignore'))]), ['Shape']),
('num', Pipeline(steps=[ ('imputer_num',SimpleImputer()),
('scaler',StandardScaler())]),['BI-RADS', 'Age', 'Margin', 'Density'])])),
('clf', LogisticRegression(C=0.01, random_state=10,solver='newton-cg'))])
cross_val score y std: (0.8335276338514681, 0.032319847105984616)

# KNN (Best estimator)
Pipeline(steps=[('pre',ColumnTransformer(transformers=[('cat',Pipeline(steps=[
('imputer_cat',SimpleImputer(strategy='most_frequent')), ('onehot',
    OneHotEncoder(handle_unknown='ignore'))]), ['Shape']),
('num', Pipeline(steps=[ ('imputer_num',SimpleImputer()),
('scaler',StandardScaler())]),['BI-RADS', 'Age', 'Margin', 'Density'])])),
('clf', KNeighborsClassifier(metric='manhattan', n_neighbors=6))])
cross_val score y std: (0.8179134283246977, 0.026233632645637096)

#SVC (Best estimator)
Pipeline(steps=[('pre',ColumnTransformer(transformers=[('cat',Pipeline(steps=[
('imputer_cat',SimpleImputer(strategy='most_frequent')), ('onehot',
    OneHotEncoder(handle_unknown='ignore'))]), ['Shape']),
('num', Pipeline(steps=[ ('imputer_num',SimpleImputer()),
('scaler',StandardScaler())]),['BI-RADS', 'Age', 'Margin', 'Density'])])),
('clf', SVC(C=1, gamma='auto', random_state=10))])
cross_val score y std: (0.8283246977547496, 0.03887076020885105)

#GBC (Best estimator)
Pipeline(steps=[('pre',ColumnTransformer(transformers=[('cat',Pipeline(steps=[
('imputer_cat',SimpleImputer(strategy='most_frequent')), ('onehot',
    OneHotEncoder(handle_unknown='ignore'))]), ['Shape']),
('num', Pipeline(steps=[ ('imputer_num',SimpleImputer(strategy='median')),
('scaler',StandardScaler())]),['BI-RADS', 'Age', 'Margin', 'Density'])])),
('clf',GradientBoostingClassifier(n_estimators=1, random_state=10))])
cross_val score y std: (0.785627158894646, 0.03710160085379686)

#RFC (Best estimator)
Pipeline(steps=[('pre',ColumnTransformer(transformers=[('cat',Pipeline(steps=[
('imputer_cat',SimpleImputer(strategy='most_frequent')), ('onehot',
    OneHotEncoder(handle_unknown='ignore'))]), ['Shape']),
('num', Pipeline(steps=[ ('imputer_num',SimpleImputer(strategy='median')),
('scaler',StandardScaler())]),['BI-RADS', 'Age', 'Margin', 'Density'])])),
('clf', RandomForestClassifier(max_depth=1, random_state=10))])
cross_val score y std: (0.8314389032815198, 0.041161211805022364)
```

En la figura [1.7] mostramos la curva con todos los modelos de la práctica anteriors con la mejor combinación de parámetros de la figura [1.6].

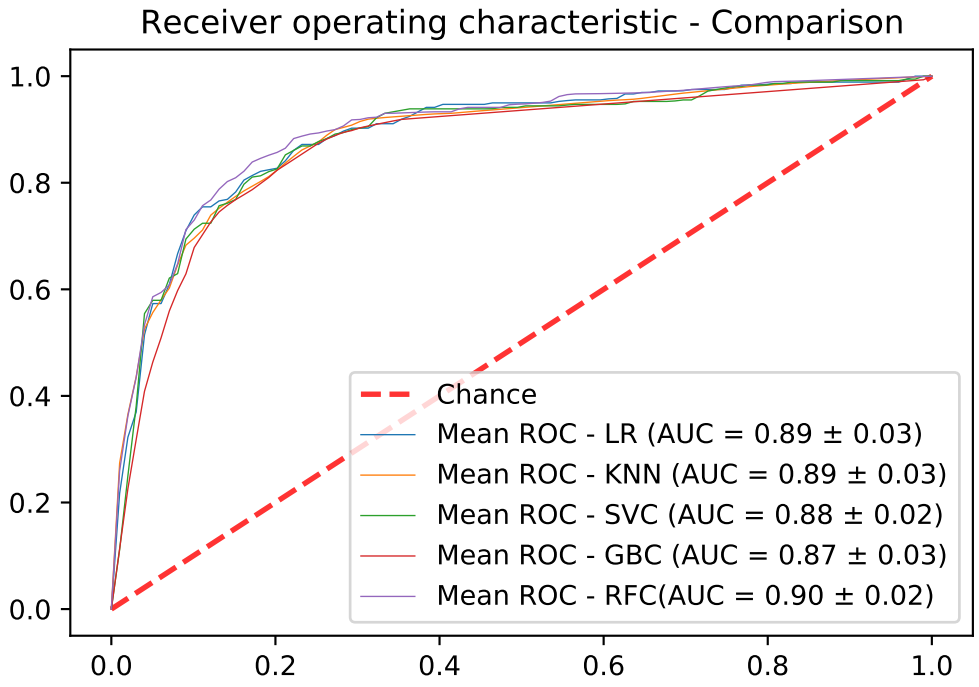


Figure 1.7: ROC con todos los modelos

1.3 Análisis de los atributos

En esta sección se muestra una serie de representaciones para mostrar la relación entre los distintos atributos y el atributo objetivo (la severidad). Paralelamente, se usarán dichas representaciones para analizar la posible relación entre ellos. Partimos de la tabla [1.1] donde mostramos las correlaciones entre nuestras variables y con nuestro atributo objetivo. Para calcular estas correlaciones antes se ha codificado la variable Shape y Severity ya que son categóricas, para el caso de la variable Shape se ha codificado forzando el orden siguiente:

```
{'N': 1.0, 'R': 2.0, 'O': 3.0, 'L': 4.0, 'I': 5.0})
```

	BI-RADS	Age	Shape	Margin	Density	Severity
BI-RADS	1.000000	0.288232	-0.398170	0.412952	0.082207	0.505132
Age	0.288232	1.000000	-0.366142	0.411355	0.028954	0.432066
Shape	-0.398170	-0.366142	1.000000	-0.736550	-0.077698	-0.561569
Margin	0.412952	0.411355	-0.736550	1.000000	0.109392	0.574919
Density	0.082207	0.028954	-0.077698	0.109392	1.000000	0.064010
Severity	0.505132	0.432066	-0.561569	0.574919	0.064010	1.000000

Table 1.1: Tabla de Correlaciones

Excepto **Density**, nuestras variables están correlacionadas con nuestro objetivo **Severity**, hacemos un plot general para ver nuestras distribuciones en la figura [1.8].

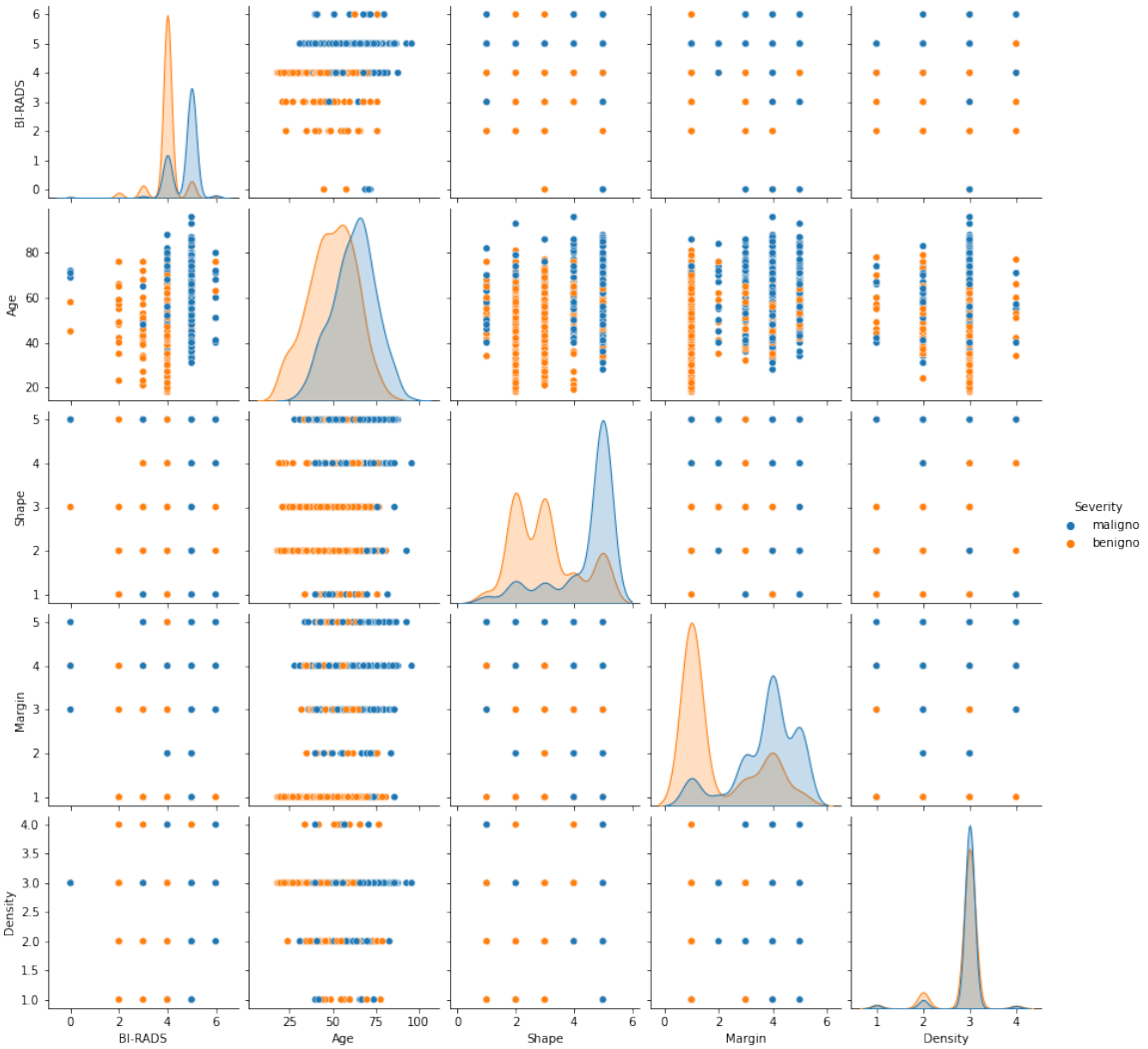


Figure 1.8: Visualización pairplot con el atributo Severity

Vemos que en general, las distribuciones de nuestras características consigo mismas, a mayores valores de las mismas mayor concentración de **maligno** obtenemos, excepto en **Density** que la cantidad de maligno y benigno son las mismas, estudianto con más detalle esto, mostramos un Catplot en la figura [1.9], donde tienen más "fuerza" mayores valores de **BI-RADS** y **Margin** que **Shape**.

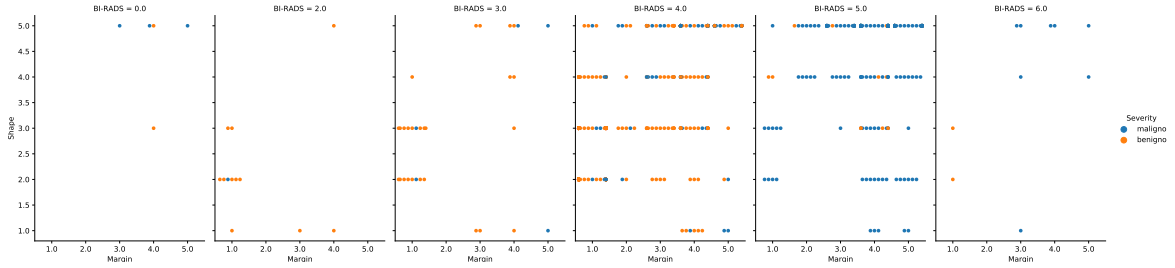


Figure 1.9: Catplot BI-RADS, Shape y Margin con Severity

También vemos el número de **BI-RADS** con **Severity** en la figura [1.10] donde debería haber un aumento de malignos conforme aumenta el número de **BI-RADS**, y lo hay, en el máximo nivel (6), son lo que la malignidad esta comprobada mediante biopsia, por ello el poco número , los demás son solo sospechas, lo cuál puede dar lugar a error, de ello que haya también benignos y malignos entre el 4 y el 5.

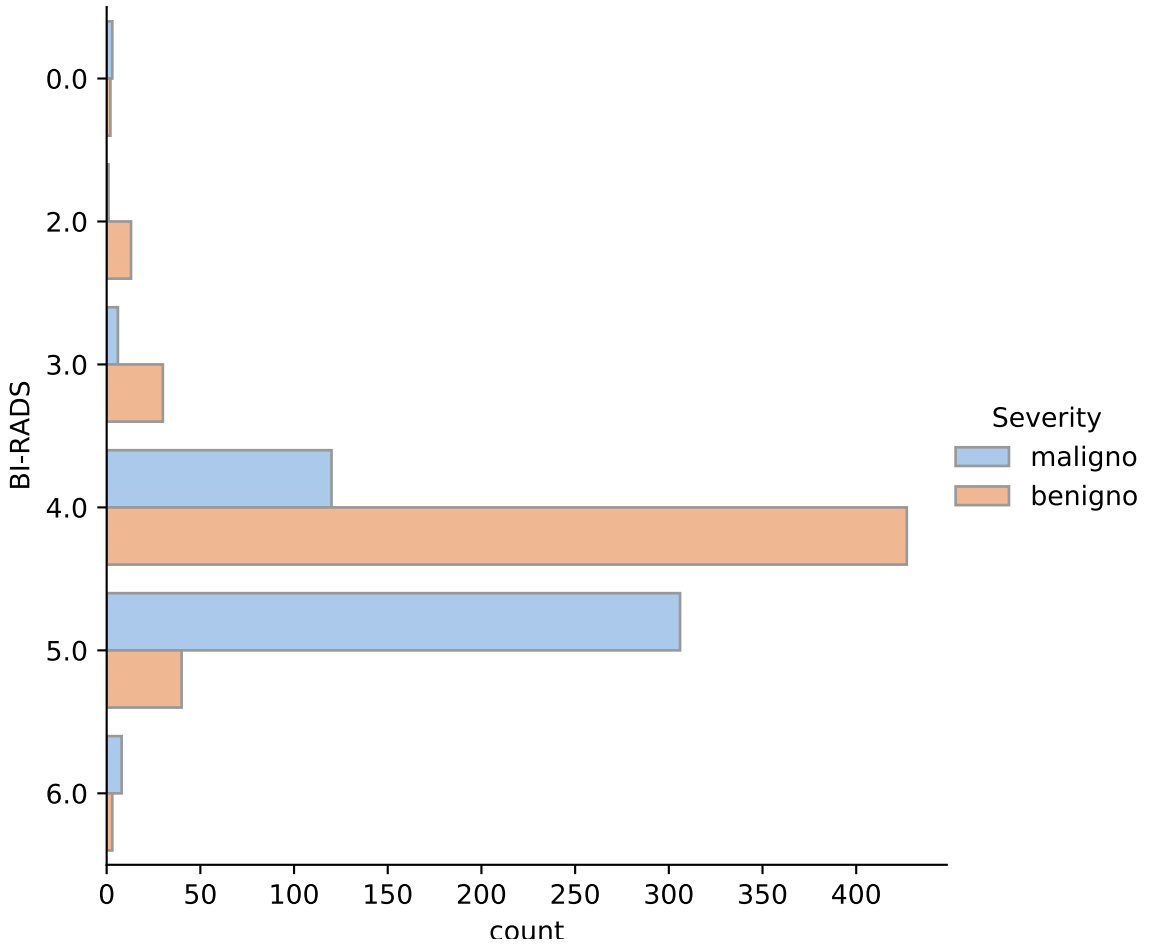


Figure 1.10: BI-RADS con Severity

También podemos hacer dos boxtplot en la figura [1.11] viendo la concentración de malignos y benignos comparando Shape y Margin junto a la atributo Age, para ver la variación de concentración de estos. Se encuentra una mayor concentración de

malignos a mayores edades, independientemente de los otros atributos, siempre hay mayor concentración en mayores edades.

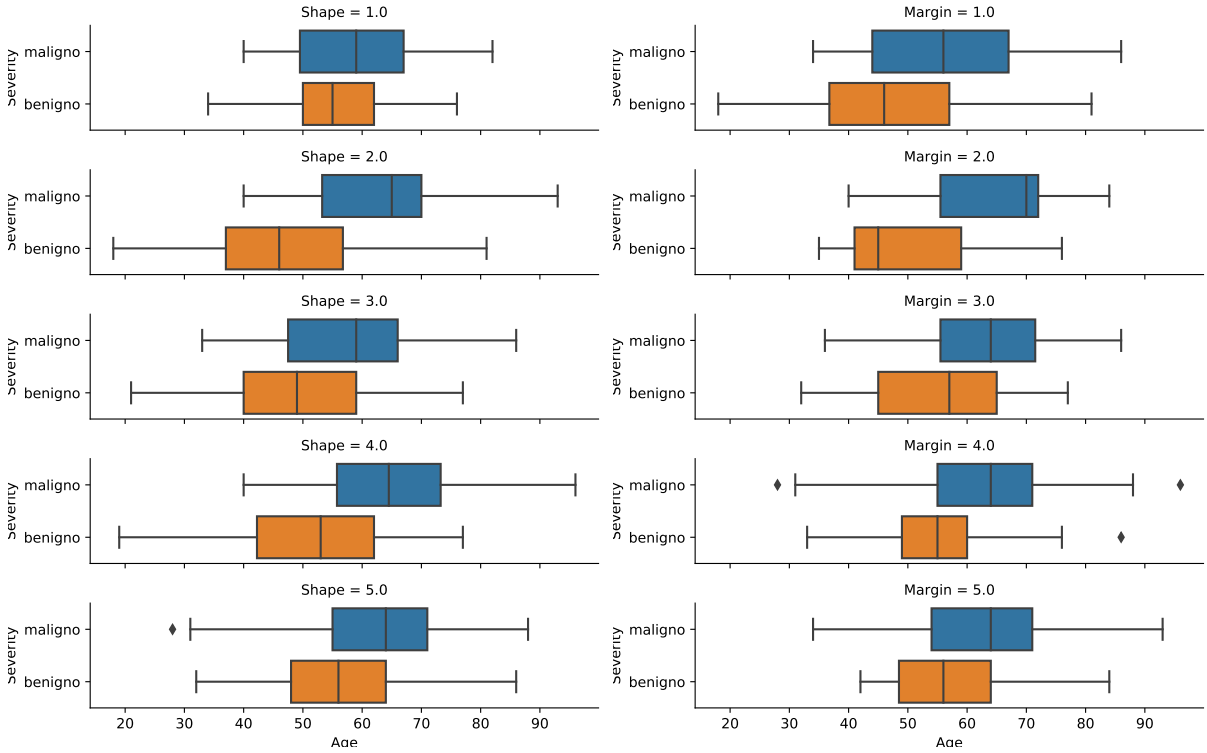


Figure 1.11: Boxplot Shape y Margin con Age

Con esto concluimos el capítulo 1 donde mostramos distintas maneras de visualizar los resultados de la práctica anterior, junto a gráficos visuales para ver la similitud/diferencia de nuestros atributos con el atributo objetivo y entre ellos combinando distintas técnicas junto a enlaces proporcionados en clase de prácticas.

Chapter 2

Segmentación

2.1 Introducción

En esta capítulo de la memoria vamos a abordar un problema de aprendizaje no supervisado, clustering. Una compañia aseguradora quiere comprender mejor las dinámicas en accidentes de tráfico en España. Para ello, a partir de diversas variables que caracterizan el accidente, se pretende encontrar los grupos de accidentes similares y relaciones de causalidad que expliquen tipos y gravedad de los accidentes. Como *dataset* se utilizarán los datos publicados por la **Dirección General de Tráfico (DGT)**, donde se recoge información de más de 30 variables entre los años 2008 y 2015. en nuestro caso, utilizaremos los datos del año 2013, por tanto utilizaremos una base que consiste en 89.519 accidentes.
Nos interesaremos en segmentar los accidentes seleccionando previamente grupos de interés (concretamente 4 para compararlos 2 a 2, el caso 1 con el 2, y el caso 3 con el 4) según las variables categóricas y realizar el estudio sobre ellos. Será necesario también aplicar una normalización para que las métricas de distancia y la visualización funcionen correctamente. Justificarse las decisiones tomadas respecto al tratamiento de las variables.
En cada caso de estudio se analizarán 5 algoritmos distintos de agrupamiento (siendo al menos uno de ellos K-means) obteniendose el tiempo de ejecución y métricas de rendimiento, como son las siguientes medidas de error:

- **Calinski-Harabaz**: se define como la razón entre la dispersión interior de los clústeres y la dispersión entre los clústeres.

- **Silhouette**: es un método de interpretación y validación de la consistencia en un problema de clustering. Mide como de bien colocado está cada objeto en su clase. Este método devuelve un valor entre -1 y 1, siendo una medida de como de similar es un objeto a los elementos de su clúster (cohesión) comparado con el resto de clústeres (separación). Los valores más altos indican que el objeto está bien colocado en su clúster y bien separado del resto de clústeres, mientras que un valor bajo o negativo significa que estamos considerando un número erróneo de clústeresen el algoritmo.

Además, se analizará el efecto de algunos parámetros determinantes en al menos 2 algoritmos distintos para cada caso de estudio.
Entre los 5 algoritmos que se escogeran para cada de estudio se elegiran entre los siguientes [Alg]:

1. **K-Means**: Este algoritmo se hará en todos los casos de estudio y será tomado como algoritmo de referencia. Es un algoritmo de particionamiento iterativo en el que las instancias se van moviendo entre clústeres minimizando el error cuadrático entre clústeres. Necesita ser especificado el número de clústeres. Para su funcionamiento el método utiliza el concepto de centroide (media de los puntos de un mismo clúster) que no tiene por qué pertenecer al conjunto de muestra. El algoritmo finaliza cuando el centroide más cercano a cada punto de la muestra sea el del clúster al que pertenece. El centroide se actualiza en cada iteración del algoritmo. Entre los parámetros más significativos del método en la libreria *scikit-learn* encontramos:

   - **n_clusters**: Número de clústeres(8 por defecto).
   - **init**: Método de inicialización, toma los siguientes valores:
     - **k_means++**: Opción por defecto. Selecciona los centroides de forma que se acelere la convegencia.
     - **random**: Elige los centroides aleatoriamente de entre los datos de la muestra.
     - **ndarray**: Para pasar como argumento los centroides.
   - **max_iter**: Máximo número de iteraciones en una misma ejecución, por defecto 300.

2. **Mini Batch K-Means**: Variación de **K-means** que usa mini-batches para reducir el tiempo de computación mientras que busca optimizar la misma función. Los mini-batches son subconjuntos de los datos de entrada aleatoriamente generados en cada iteración de entrenamiento. Estos subconjuntos disminuyen radicalmente el tiempo de computación requerido para converger a una solución local. A diferencia de otros algoritmos que mejoran el tiempo de cómputo de **K-Means**, por regla general Mini-Batch K-Means produce resultados poco peores que los que produce **K-Means**. Aunque los parámetros son muchos iguales a los que necesita el métodos **K-Means**, destacamos:

   - **batch-size**: Tamaño de los mini-batches (100 por defecto).

3. **Mean-Shift**: Este algoritmo busca encontrar máximos en función de la densidad regular de las muestras, se basa en centroides, que actualiza los candidatos a centroides de forma que sean la media entre los puntos pertenecientes a una misma región. Posteriormente, se eligen los centroides de entre los candidatos de forma que no haya dos centroides muy cercanos.

   El algoritmo fija el número de clústeres, en lugar de depender del parámetro bandwidth, que puede ser fijado manualmente o estimado.

   Entre los parámetros más significativos del método en *scikit-learn* tenemos:

- **bandwidth**: Dicta el tamaño de la región por la que buscar. Si no se introduce el parámetro, se llama al método *estimate bandwidth*.

4. **SpectralClustering**: Hace uso del radio espectral de la matriz de similitud de la muestra para reducir la dimensionalidad antes de aplicar clustering en dimensiones más bajas. Luego, aplica **K-Means** en este espacio de dimensión reducida. Este algoritmo es especialmente eficiente si la matriz de similitud contiene muchos ceros. Tiene un buen funcoinamiento cuando utilizamos un número pequeño de clústers y no se aconseja su uso para un alto número de estos.

   Entre los parámetros más significativos del método en *scikit-learn* tenemos:

   - **n_clusters**: Número de clusters
   - **eigen_solver**: Método para buscar autovalores (*arpack*, *obpcg* o *amg*)
   - **n_init**: Número de veces que aplicamos el algoritmo de **K-Means** con diferentes centroides.
   - **affinity**: Manera de construir la matriz de afinidad (*nearest_neighbors*, *precomputed* o *rbf*)
   - **n_neigbors**: Número de vecinos a utilizar para construir la matriz de similitud usando el métodos de vecino más cercano (ignorado cuando usamos affinity = ”rbf”).
   - **assign_labels**: Método a utilizar en el espacio de dimensión reducida. Puede tomar los valores *kmeans* o *discretize*

5. **AgglomerativeClustering**: Implementa clustering jerárquico usando aproximación ascendete, cada observación comienza en el propio clúster y los clusters se mezclan sucesivamente. El criterio de conexión determina la métrica usada para cada estrategia de mezcla y podemos elegir entre las siguientes opciones configuracion el parámetro *linkage*:

   - **Ward**: minimiza la suma de las diferencias al cuadrado entre todos los clusters. Al igual que **K-Means** minimiza la varianza.
   - **Maximum**:minimiza la máxima distancia entre pares de clusters.
   - **Average**: minimiza la media de las distancias entre pares de clusters.

   Entre los parámetros más significativos del método en *scikit-learn* tenemos:

   - **n_clusters**: Número de clusters
   - **linkage**: Comentado antes
   - **affinity**: Métrica a utilizar

6. **DBSCAN**: considera los clusters como áreas de alta densidad separadas por otras de de baja densidad, de esta manera, las formas de los clusters pueden ser de cualquier manera. A las muestras que están en áreas de alta densidad se les denomina *core samples*.

   Entre los parámetros más significativos del método en *scikit-learn* tenemos:

   - **eps**: distancia permitida entre dos muestras para poder pertenecer al mismo cluster.
   - **min_samples**: Múmero de muestras necesarias en un vecindario para poder considerar un punto como *core point*.
   - **algorithm**: algoritmo que utiliza el vecino más cercano.
   - **metric**: Métrica usada para calcular la distancia entre instancias.

7. **Birch**: Este algoritmo se utiliza para aplicar clustering jerárquico a conjuntos de datos grandes donde uno jerárquico tardaría demasiado. Entre los parámetros más significativos del método en *scikit-learn* tenemos:

   - **n_clusters**: Número de clusters
   - **thershold**: Por defecto toma el valor de 0.5.
   - **branching factor**: Maximum number of CF subclusters in each node

Finalmente, el análisis se apoyará en visualizaciones tales como nubes de puntos (scatter matrix), dendrogramas y mapas de temperatura (heatmap). También se mostraran gráficas de los centroides para ayudar a interpretar el significado de cada grupo.
A partir de los resultados obtenidos se deberán extraer conclusiones sobre los grupos de población. Se valorará el acierto en la selección de casos de estudio que mejor reflejen los grupos encontrados en los datos.
Se incluye una sección por cada caso de estudio analizado. En cada una de ellas se explicará qué caso se analiza, se mostrará una tabla comparativa con los resultados de los algoritmos de clustering y tantas otras tablas para el análisis de los parámetros (una tabla por algoritmo) junto a visualizaciones necesarias para analizar el problema. Además, se añadirá un apartado final titulado **Interpretación de la segmentación** donde se incluirán conclusiones generales. Aparte en los casos pares se incluirán comentarios de comparación respecto al caso anterior (Recordamos que comparamos los casos 1 y 2, y los casos 3 y 4).
Por comodidad personal todos los atributos de nuestro conjunto utilizado serán transformados a lowercase, también comentar que se trabajara siempre con los datos normalizados para trabajar en la misma escala ya que muchos algoritmos de clustering se basan en distancias y no queremos que unos atributos tengan más peso que otros. (Cuando hablamos de normalización estamos diciendo reescalar nuestros atributos en el rango entre 0 y 1, no confundir con estandardization)

2.2 Caso de estudio 1

2.2.1 Descripción

Para el primer caso de estudio, hemos escogido los accidentes que se producen a la hora de entrada y salida del trabajo y colegio (entre las 8-10 horas y 13-16 horas), escogiendo como días de lunes a viernes donde además, para reducir el número se estudiará en una proviciona concreta, por ejemplo Madrid, que tiene la mayor densidad de población de España.[1] Nos quedaría un conjunto total de 4010 muestras.

La selección de características para el análisis es:

1. tot_victimas
2. tot_muertos
3. tot_heridos_graves
4. tot_heridos_leves
5. tot_vehiculos_implicados

Vemos una primera visualización de los datos en nuestras variables a estudiar en la figura [2.1], donde apreciamos que la mayoría de los valores de nuestras variables se concentran en los menores valores aunque en general, se encuentra todo tipo de valores.

[1] Todo lo relevante a este caso de estudio se recopila en la carpeta de notebooks en el fichero llamado **2_segmentation_accidents_case_1.ipynb** junto al archivo **case_hours.py** donde recopilamos todas las funciones utilizadas.

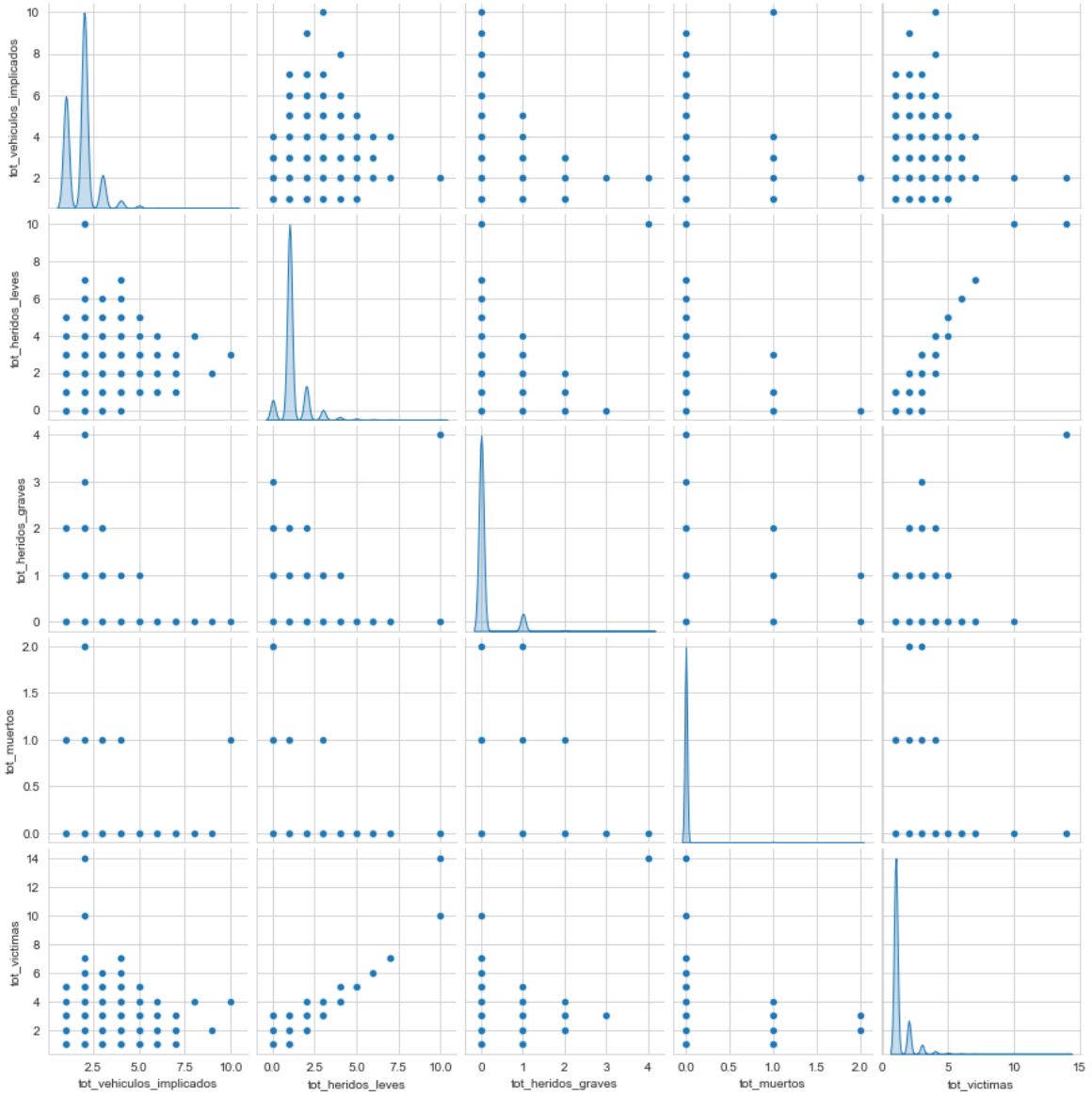


Figure 2.1: Primera visualización de los datos caso 1.

También hacemos una pequeña visualización en el conjunto original para ver como influyen las variables que modificamos en este caso, por ejemplo **diasemana** y **hora** sobre los atributos de **tot_muertos** y **tot_victimas** en la figura [2.2]

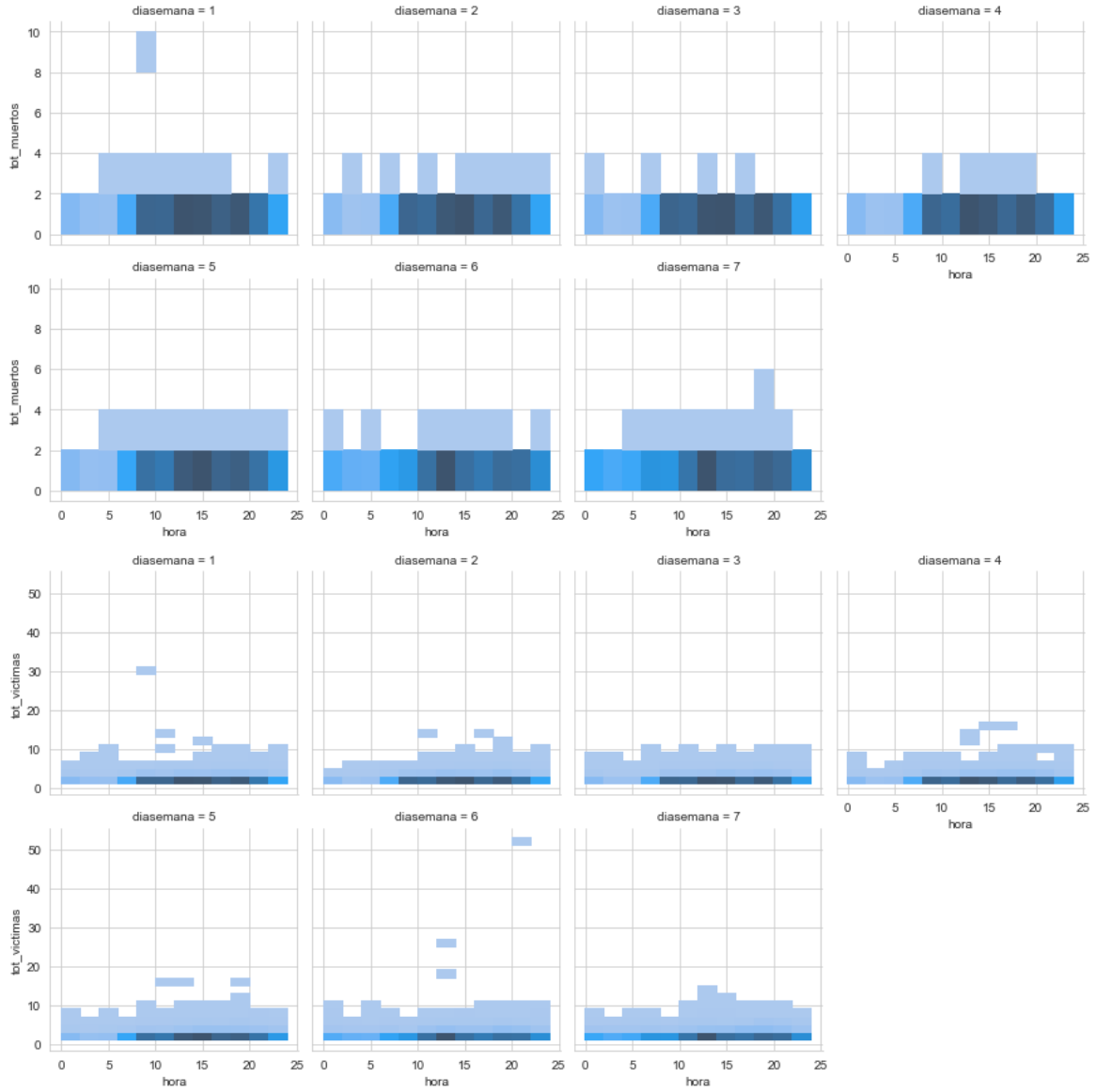


Figure 2.2: Visualización de tot_muertos y tot_victimas respecto días de la semana y horas.

2.2.2 Análisis del caso de estudio 1

Para el análisis de este caso de estudio hemos considerado 5 algoritmos de clustering diferentes:

- **K-Means** con parámetros **n_clusters** = 5, **n_init** = 100 e **init** = k-means++
- **MiniBatchKMeans** con **n_clusters** = 5 y **init** = k-means++.
- **MeanShift** con bandwidth estimado usando *estimate_bandwidth* usando como parámetro *quantile* = 0.3.
- **AgglomerativeClustering** con **n_clusters** = 100 y **linkage** = 'ward'.

- **AgglomerativeClustering** con **n_clusters** = 100 y **linkage** = 'average'.

En la tabla [2.1] vemos los resultados obtenidos para cada uno de estos algoritmos.

Algoritmo	Clusters	Calinski	Silhouette	time(s)
K-Means	5	2403.207134701694	0.737289	0.547624
MiniBatchMeans	5	2345.150007151071	0.670002	0.021241
MeanShift	70	1.790387e+29	0.992269	2.402508
AgglomerativeWard	100	1.255212e+29	0.740898	0.246929
AgglomerativeAverage	100	1.238418e+29	0.955112	0.221144

Table 2.1: Tabla comparativa caso 1.

En cuanto a los comentarios de los resultados obtenidos en la tabla [2.1], podemos destacar que **K- Means** y **MiniBatchKMeans** han obtenido en ambas medidas del error resultados muy parecidos, sin embargo el tiempo ha marcado la diferencia, siendo el tiempo para **K-Means** del orden de casi 25 veces más alto. No nos debe sorprender por lo comentado en la descripción de los algoritmos acerca de **MiniBatchKMeans** que no es más que un **K-Means** mejorado en eficiencia utilizando mini-batches para reducir el tiempo de computación, la reducción del tiempo es bastante significativa pudiendo ser su uso crucial en problemas de más altas dimensiones.
En cuanto al resto de algoritmos en comparación con nuestro algoritmo de referencia, **K-Means**, obtenemos que **MeanShift** ha obtenido mejores resultados tanto para el índice de **Calinski-Harabaz** y alcanzando un resultado bastante cercano a 1 en el coeficiente **Silhouette**, lo que nos indica que hemos obtenido un resultado muy bueno de agrupación a costa de mayor tiempo. Destacar que estos resultados se han obtenido utilizando un total de 70 clusters (fijados automáticamente por el algoritmo), bastantes más clusters que los 5 clusters fijados para las dos versiones del **K-means**, por tanto sería interesante probar a aplicar **K-Means** más clusters para ver si el resultado. Realizaremos este estudio en la parte de análisis de parámetros [2.2.3].
En cuanto a los algoritmos jerárquicos los comentarios que podemos realizar son comunes a ambos algoritmos y es que en ambos casos el de **Calinski-Harabaz** es muy alto y el coeficiente **Silhouette** en **AgglomerativeWard** bastante peor que con **AgglomerativeAverage**. Igual que con **MeanShift** esto se puede deber a que al haber considerado un número tan alto de clusters habrá varios de ellos que estén compuestos por un solo punto, siendo así la distancia a su propio cluster 0 y produciendo problemas en las medidas consideradas.

2.2.3 Análisis de parámetros caso 1

En esta sección vamos a probar distintas configuraciones de algunos algoritmos implementados en la sección anterior. Nos centraremos en K-Means donde hablaremos del método del codo y en algoritmos jerárquicos.

K-Means

Como ya hemos comentado en la sección anterior, los mejores resultados de otros algoritmo respecto a este se podría deber al número de clústeres elegido, al tener que elegir el número de clusteres, limitamos la potencialidad del método pues puede ser que el número introducido no sea optimo. La idea básica de los algoritmos de clustering es que cada observación se encuentre muy cerca a las de su mismo grupo y los grupos lo más lejos posible entre ellos.

Podemos ver el **método del codo** (elbow method) que utiliza la distancia media de las observaciones a su centroide. Es decir, se fija en las distancias intra-cluster. Cuanto más grande es el número de clusters **k**, la varianza intra-cluster tiende a disminuir. Cuanto menor es la distancia intra-cluster mejor, ya que significa que los clústers son más compactos. El método del codo busca el valor **k** que satisfaga que un incremento de **k**, no mejore sustancialmente la distancia media intra-cluster. Visualizamos esto en la figura [2.3].

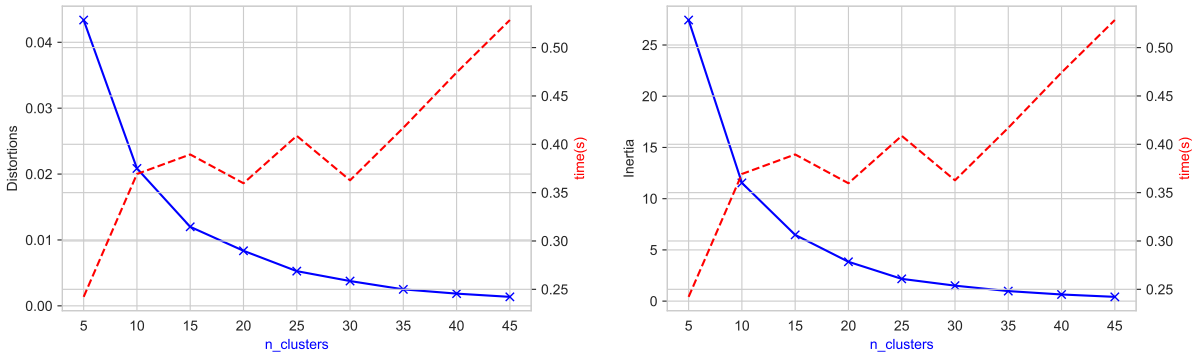


Figure 2.3: Elbow caso 1.

En este análisis, a partir de 15 clusters la reducción en la suma total de cuadrados internos parece estabilizarse, indicando que a partir k = 15 es una buena opción, visualizamos esto también en la tabla [2.2] y comparamos las medidas obtenidas con gráficas en la figura [2.4]

	Silhouette	Calinski-Harabaz	time(s)
5	0.737289	2403.207135	0.242206
10	0.874954	3147.826375	0.369276
15	0.918733	3830.826366	0.389491
20	0.942489	4891.703991	0.359697
25	0.961021	6963.955539	0.408744
30	0.971851	8338.228209	0.362750
35	0.980446	11022.318966	0.417055
40	0.983656	14544.026801	0.474238
45	0.986478	20097.199248	0.528528

Table 2.2: Tabla Elbow caso 1.

Se cumple lo que pensabamos y es que el resultado del **K-Means** para este caso de estudio mejora según el número de clusters considerado. Ahora bien, igual que aumentan tanto el índice de **Calinski-Harabaz** como el coeficiente **Silhouette**, también aumenta el tiempo de ejecución del algoritmo. Esto sería algo a tener en un problema de un tamaño suficiente como para que el tiempo nos causara problemas. De todas formas, el algoritmo más lento no ha tardado ni 1 segundo por lo que en nuestro caso no es un problema. También vemos que a partir de 15, como habiamos comentado con el método del codo, nuestros valores se ”estabilizan”. El coeficiente **Silhouette** crece muy rápido al principio quedando casi estancada al final, al contrario que él índice **Calinski-Harabaz**.

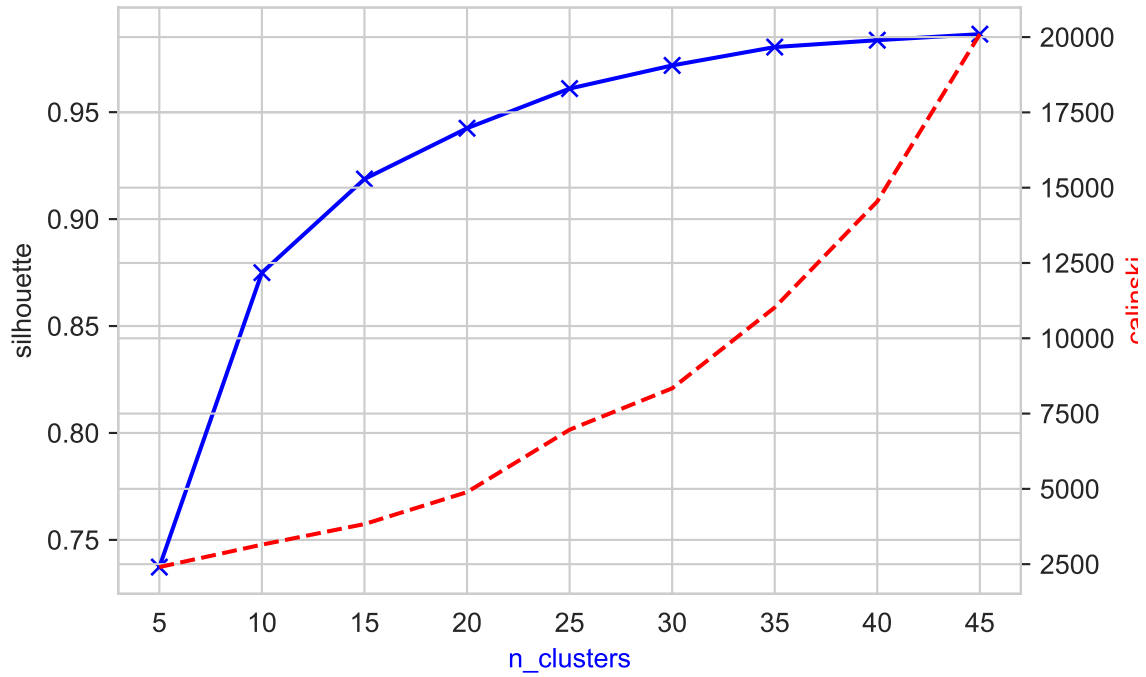


Figure 2.4: Comparación Silhouette y Calinski-Harabaz caso 1.

Aunque a partir de k= 15 se presenten mejores resultados, obtener una solución de este de más de 15 clusters es muy difícil de interpretar. Por tanto, aunque aumentemos los clusters para comprobar que **KMeans** obtiene buenos resultados, no tendría sentido solución con tantos clusters.

Algoritmos jerárquicos

Los extraños resultados de las medidas de error para estos algoritmos creemos que se deben a que existen varios clusters con un solo elemento. Por ello, de manera adicional vamos a realizar un preprocesado de los datos eliminando los outliers antes de realizar las diferentes configuraciones de este algoritmo. No debería ser necesario que hubiera clusters con un solo elemento dado que estamos considerando 100 clusters en un conjunto de 4010 muestras.

Vamos a considerar que todo clusters con menos de 2 datos es un clusters de valores perdidos. Eliminando estos outliers para 100 clusters obtenemos que **de los 100 clusters hay 34 con más de 2 elementos luego del total de 4010 elementos, seleccionamos 3939**. Realizaremos la comparación de parámetros con este conjunto de **3939** elementos.

Una vez eliminados los outliers, vamos a probar los algoritmos jerárquicos para diferentes clusters donde compararemos como antes los resultados. Obtenemos los resultados y los mostramos en la tabla [2.3].

Algoritmo	Clusters	Calinski	Silhouette	time(s)
Ward-5	5	3798.562107	0.750642	0.212561
Average-5	5	781.215053	0.659255	0.268287
Ward-10	10	5489.123402	0.888629	0.406322
Average-10	10	1024.420430	0.657322	0.186418
Ward-15	15	9181.137199	0.951418	0.307619
Average-15	15	1382.437352	0.648075	0.201964
Ward-20	20	20105.210888	0.974317	0.236620
Average-20	20	1715.888579	0.676291	0.128592
Ward-25	25	42511.555973	0.989642	0.157019
Average-25	25	1788.471953	0.699943	0.130088
Ward-30	30	124354.630222	0.996088	0.156460
Average-30	30	1577.598294	0.697332	0.129580

Table 2.3: Tabla comparación algoritmos jerárquicos caso 1.

A rasgos generales, observamos la mejora de los resultados. Tanto el valor del **Calinski-Harabaz** como el del coeficiente **Silhouette** aumentan en función del número de clusters considerados. También en general, Ward ha obtenido mejores resultados que Average, luego a partir de ahora consideraremos el algoritmo Ward cuando queramos aplicar un jerárquico. Al contrario que al principio, vemos que estos algoritmos no precisan de un número de clusters muy elevado como escogimos de 100 al principio para su buen funcionamiento, de hecho, con menos clusters han obtenido resultados incluso superiores eliminando los outliers. Esto es importante dada la importancia de la interpretabilidad en este tipo de problemas.

2.2.4 Interpretación de resultados

Para la interpretación de los resultados, vamos a utilizar solo algunos de los algoritmos implementados. Elegiremos aquellos que hayan funcionado bien además de aquellos que no hayan necesitado un alto número de clusters pues esto dificultaría la interpretación.

Primera Interpretación

Vamos a realizar la interpretación del primer algoritmo **K-Means** implementado, que usaba 5 clusters. Aunque como ya hemos visto anteriormente es el peor K-Means implementado el resultado no es malo y es más fácil de visualizar. Apoyaremos nuestro análisis en la scatter-matrix producida, que podemos observar en la figura [2.5]. Haciendo el pairplot, obtenemos una matriz simétrica donde tenemos representadas, tanto la relación entre atributos como el valor de cada una de las variables. así, podremos observar tanto tendencias de variables a comportarse interrelacionadas con otras para determinados atributos. En la diagonal podemos apreciar la cantidad de elementos que pertenecen a cada uno de los clusters.

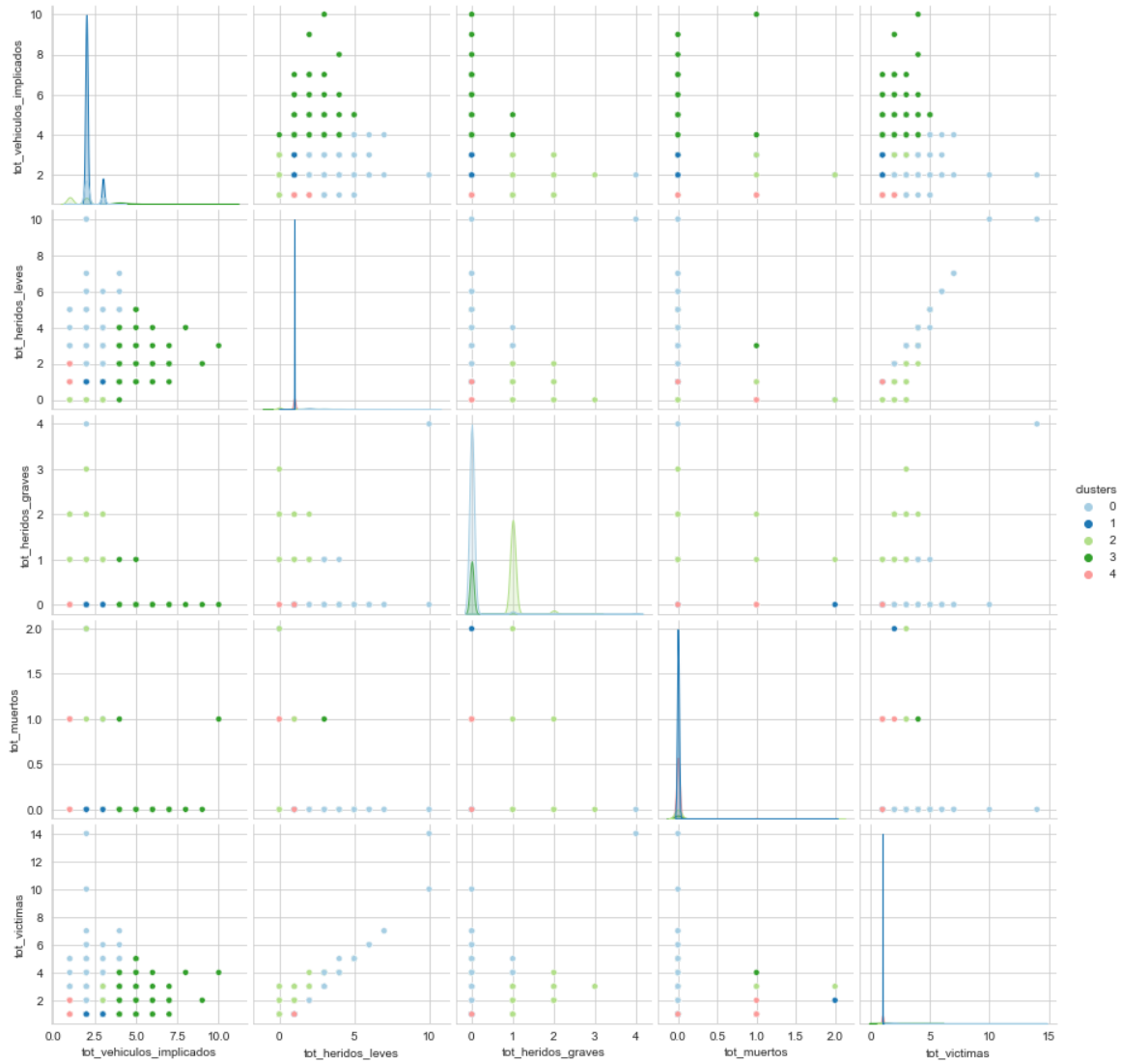


Figure 2.5: Pairplot K-Means 5 clusters caso 1.

Haciendo uso de esta visualización, vamos a estudiar qué características caracterizan a cada uno de los clusters generados. A rasgos generales, vamos a utilizar más la primera columna dado que esta representa las relaciones entre los vehículos implicados (**tot_vehiculos_implicados**) y las diferentes víctimas.

- **Cluster 0**: Incluye desde 1 a 4 vehículos implicados con gran número de víctimas entre 2 y hasta 14 cuando hay 2 vehículos implicados sin muertos pero con heridos graves y muchos heridos leves, entre 2 y 10.
- **Cluster 1**: Este cluster incluye la mayoría de vehículos implicados pero con número entre 1 y 3 sin muertos ni heridos graves pero con un herido leves siempre.
- **Cluster 2**: Con 3 vehículos implicados con siempre 1 o 2 muertos, muchos heridos graves entre 1 y 3 y ningún herido leve.

- **Cluster 3**: Contiene le mayor número de vehículos implicados entre 3 y hasta 10, sin muertos quizás con 1 herigo graves y entre 1 y 5 heridos leves.

- **Cluster 4**: En este clusters se encuentran aquellos accidentes en los que el número de vehículos implicados ha sido mínimo, con 1 o 2 heridos leves, nunca heridos graves y ninguno o 1 muerto, con 1 o 2 victimas.

A rasgos generales hemos podido agrupar los accidentes de tráfico que se producen en la provincia de Madrid durante las horas de salida y entrada del trabajo y colegio en 5 tipos.

1. Dado el bajo número de víctimas podemos deducir que son solo las que estaban en los coches (por regla general), luego la mayoría a accidentes fuera de la zona urbana.

2. Seguramente teniendo lugar en zonas de mucha concurrencia de pesonas en estas horas, centro o salida de colegios.

3. Pocos implicados y victimas asi que puede ser por tráfico en zona urbana

4. Puede deberse a accidentes en zona urbana pero de mayor gravedad.

5. Solo deja heridos y muchos vehículos implicados asi que puede deberse a caravanas o zonas urbanas por el tráfico

Casi todos los accidentes pueden haberse producido en zona urbana y varios accidentes con un número alto de coches implicados y de heridos graves. Además, los clusters con carácteristicas más determinantes para este tipo de accidentes son los más numerosos, clusters 1 y 4 (lo observamos en la diagonal). Por tanto, confirmamos que existe el peligro en las zonas urbanas de producirse accidentes. Mostramos los centroides en la figura [2.6] para apoyar este análisis.

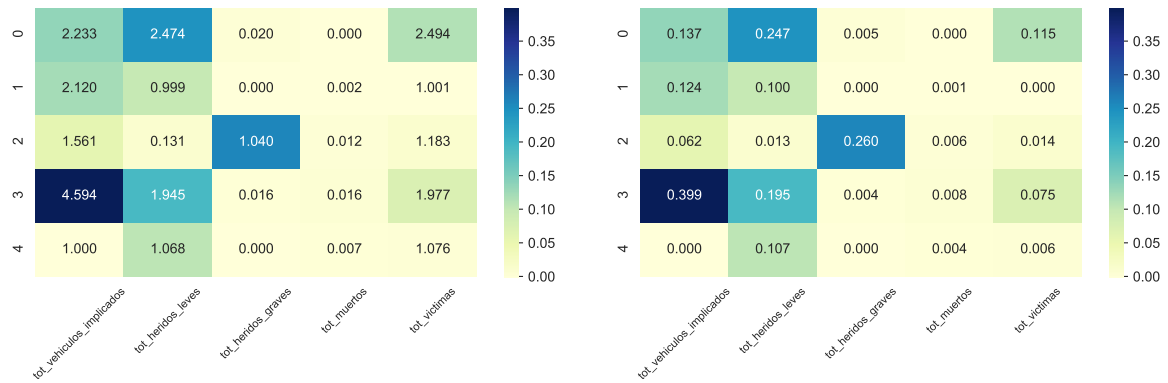


Figure 2.6: Centroides K-Means con datos sin normalizar y normalizados caso 1.

Segunda Interpretación

Para esta segunda interpretación, vamos a considerar un algoritmo jerárquico para así poder apoyar nuestro análsis en un heatmap y un dendograma.
Escogemos un algoritmo que nos haya proporcionado buenos resultados pero con un número de de clusters bajo. Por ello, vamos a utilizar el Ward AgglomerativeClustering sin outliers definido en la parte de análisis de parámetros de los algoritmos jerárquicos. Analizamos la Scatter Matrix producida en la figura [2.7].

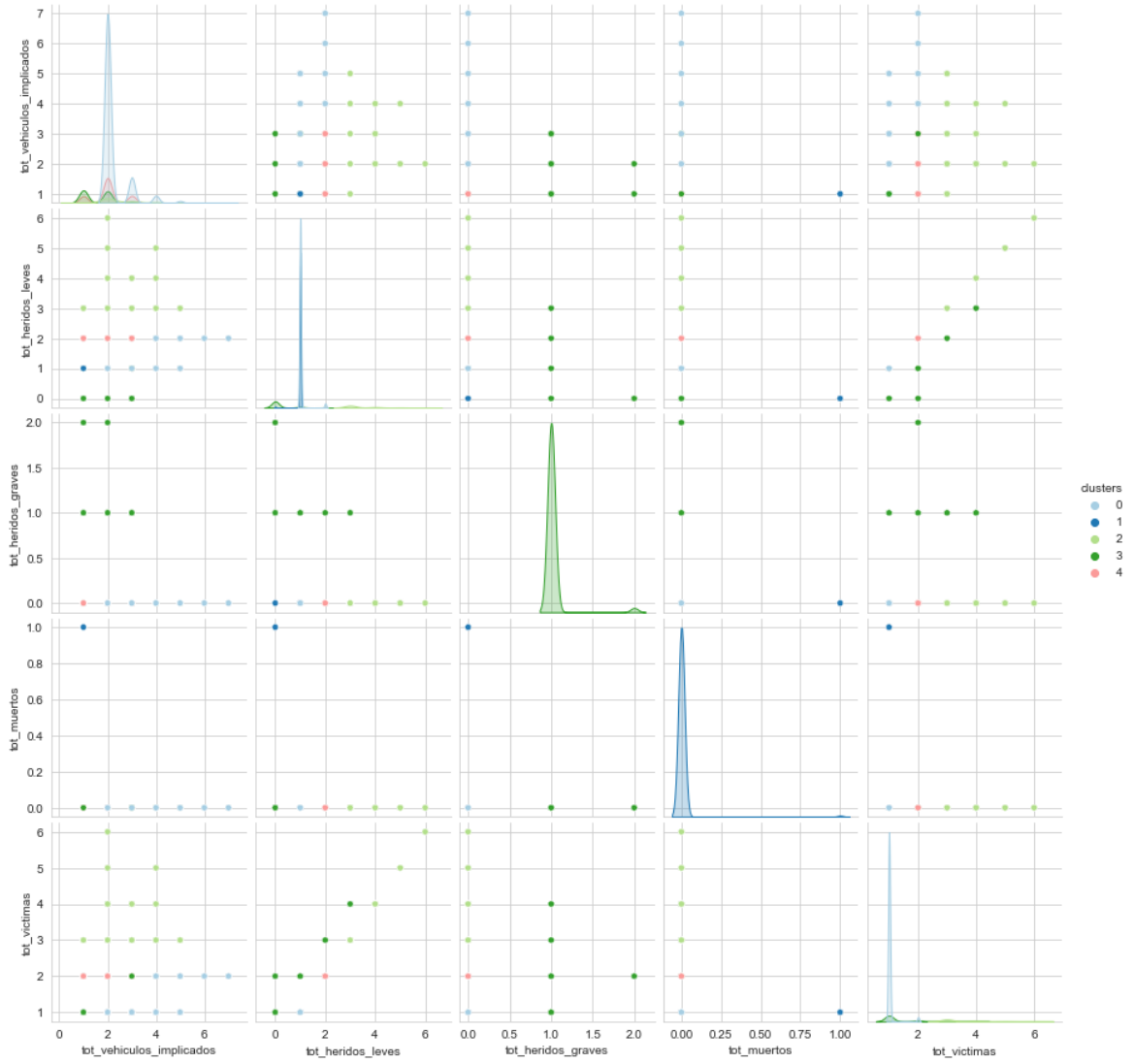


Figure 2.7: Pairplot AgglomerativeClustering 5 clusters caso 1.

- **Cluster 0**: Se concetran todo número de vehículos implicados, como mínimo 2, siempre con algún herido leve pero nunca heridos graves ni muertos pero siempre 1 o 2 víctimas.
- **Cluster 1**: Cluster con siempre 1 vehículo implicado y 1 muerto, sin heridos graves ni leves ni víctimas.

- **Cluster 2**: Mínimo algún vehículo implicado, donde se concentran la mayoría de heridos leves y víctimas, mínimo 3, sin muertos ni heridos graves.

- **Cluster 3**: Siempre con heridos graves, como mínimo 2, sin muertos y con 1 o dos víctimas.

- **Cluster 4**: Siempre con 2 heridos leves, entre 1 o 3 vehículos implicados sin heridos graves ni muertos pero con 2 víctimas.

En este caso observamos en que en la mayoría de los clusters hay una relación entre el número de vehículos implicados y el número de de heridos leves. Es natural pensar ya que cuanto más vehículos implicados hayan, más personas pueden sufrir daños. También vemos una relación entre heridos leves y víctimas, lo cuál también es normal.

Complementamos con una visualización de los centroides en la figura [2.8] y un dendograma junto con un heatmat [2.9]. Para ello utilizamos el paquete *seaborn* [Sea] que nos ofrece ambas representaciones en una. Estas representaciones tienen sentido para un número pequeño de muestras. Utilizamos el método ward pues ha sido el que ha demostrado proporcionar mejores resultados en este caso de estudio concreto. También vemos el árbol que representa la fusión jerárquica de grupos como un dendrograma. La inspección visual a menudo puede ser útil para comprender la estructura de los datos, aunque más en el caso de muestras pequeñas. Entre paréntesis tenemos el número de puntos en el nodo (o índice de punto si no hay paréntesis).

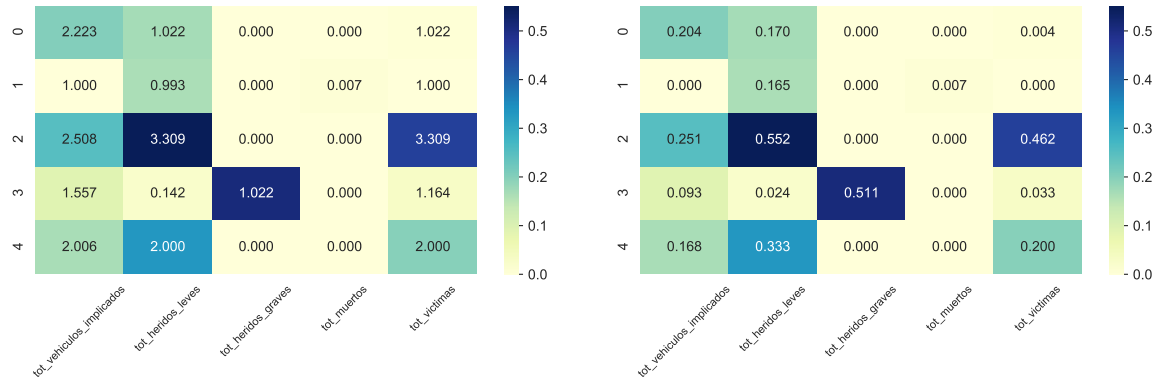


Figure 2.8: Centroides AgglomerativeClustering con datos sin normalizar y normalizados caso 1.

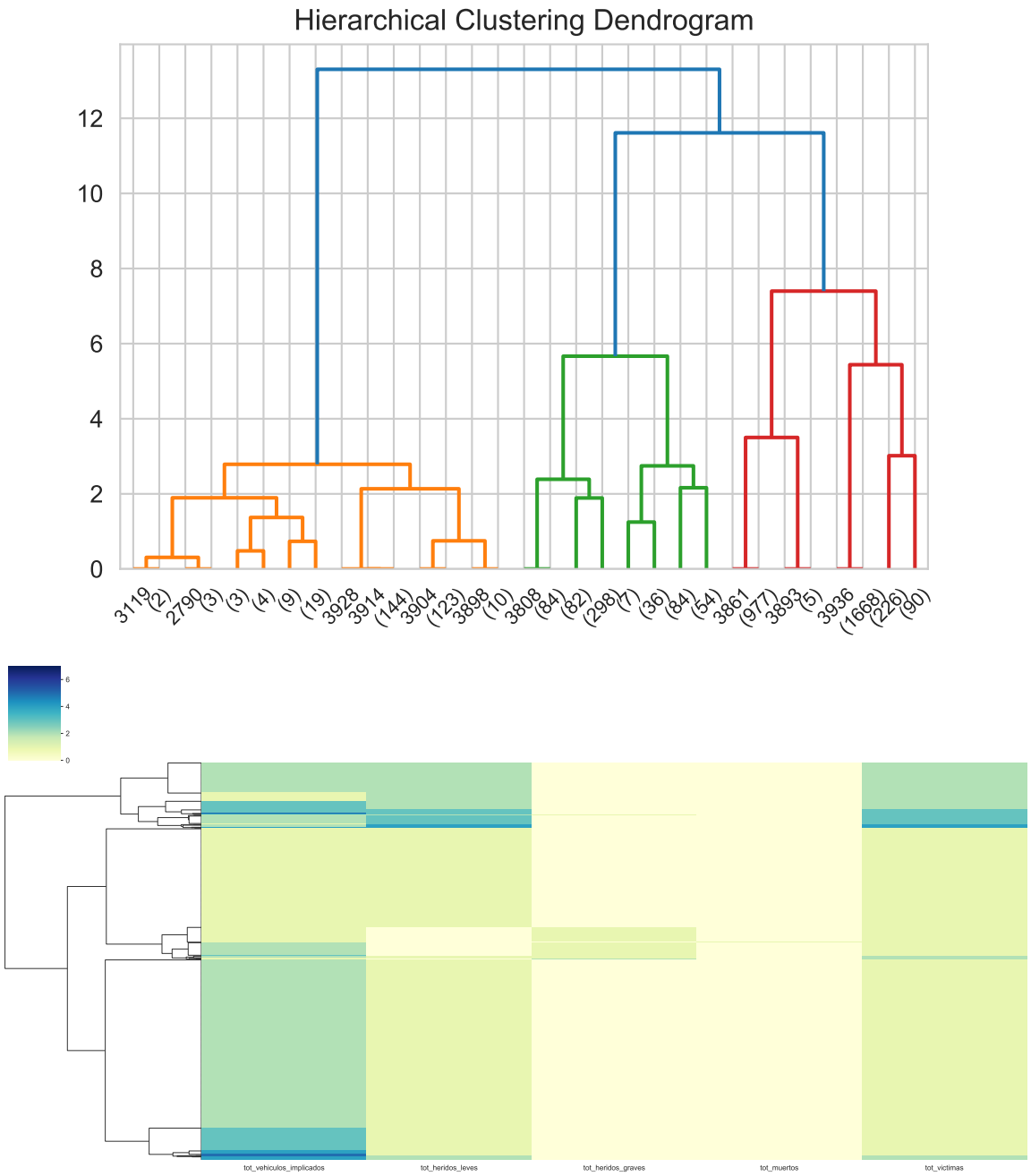


Figure 2.9: Dendograma y Heatmap AgglomerativeClustering caso 1.

A rasgos generales, podemos comprobar que en la mayoría de los clusters generados hay muy pocos muertos y muy pocos heridos graves (solo en dos de ellos hay más heridos graves). En cuanto a los heridos leves, en los mismos clusters que había muchos heridos graves, hay pocos leves, concentrándose el resto de heridos leves en el resto de clusters. En cuanto a los vehículos implicados, es por regla general muy alto.
El cluster más grande es aquel en el que hay muchos vehículos implicados, bastantes heridos leves y muy pocos o ningun herido grave y muertos. El segundo más grande

recoge los accidentes que han implicado un menor número de vehículos pero que han implicado más heridos leves. Entre estos dos clusters se recoge a gran parte de la muestra considerada.
Llegamos a la misma conclusión que en la interpretación anterior y es que por el tipo de accidente, muchos vehículos involucrados y muchos heridos leves, seguramente se traten de accidentes ocurridos en zona urbana. Por tanto hay cierto peligro en la franja horaria de entrada y salida del colegio y del trabajo dada la cantidad de peatones que hay a esa hora por la calle y al tráfico que los trabajadores producen.

2.3 Caso de estudio 2

2.3.1 Descripción

En el segundo caso de estudio, y para poder compararlo con el caso 1, hemos escogido el mismo caso que antes solo que en un intervalo de horas distinto, en la noche (entre las 19 horas y las 00:00) y de madrugada (desde las 00:00 a las 6 de la mañana), y en la misma comunidad autónoma, Madrid.[2] Nos quedaría un conjunto total de 4291 muestras.
De forma análoga al caso anterior la selección de características para el análisis es:

1. tot_victimas
2. tot_muertos
3. tot_heridos_graves
4. tot_heridos_leves
5. tot_vehiculos_implicados

Vemos una primera visualización de los datos en nuestras variables a estudiar en la figura [2.10], donde apreciamos que la mayoría de los valores de nuestras variables se concentran en los menores valores aunque en general, se encuentra todo tipo de valores, a diferencia que en la visualización del caso 1, nuestros valores ahora están mucho más juntos y concentrados.

[2]Todo lo relevante a este caso de estudio se recopila en la carpeta de notebooks en el fichero llamado **2_segmentation_accidents_case_2.ipynb** junto al archivo **case_hours.py** donde recopilamos todas las funciones utilizadas.

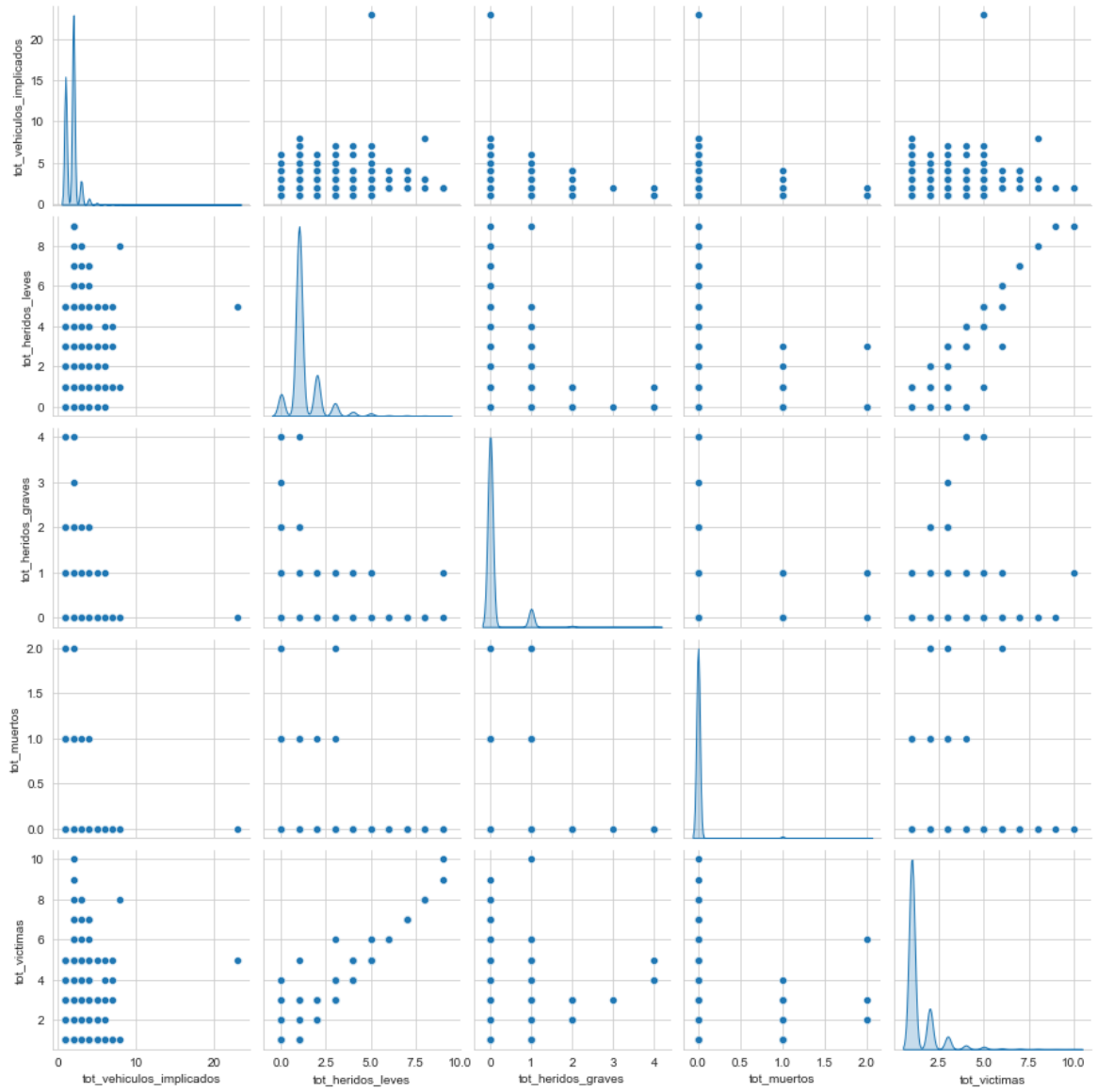


Figure 2.10: Primera visualización de los datos caso1.

Recordamos la visualización donde comentamos como afectan nuestro caso elegido sobre las carácterísticas a estudiar en la figura [2.2].

2.3.2 Análisis del caso de estudio 2

Para el análisis de este caso de estudio hemos considerado 5 algoritmos de clustering diferentes, los mismos que en el caso anterior para poder comparar:

- **K-Means** con parámetros **n_clusters** = 5, **n_init** = 100 e **init** = k-means++
- **MiniBatchKMeans** con **n_clusters** = 5 y **init** = k-means++.
- **MeanShift** con bandwidth estimado usando *estimate_bandwidth* usando como parámetro *quantile* = 0.3.

- **AgglomerativeClustering** con **n_clusters** = 100 y **linkage** = 'ward'.
- **AgglomerativeClustering** con **n_clusters** = 100 y **linkage** = 'average'.

En la tabla [2.4] vemos los resultados obtenidos para cada uno de estos algoritmos.

Algoritmo	Clusters	Calinski	Silhouette	time(s)
K-Means	5	5070.608040	0.770733	0.658824
MiniBatchMeans	5	5070.608040	0.770733	0.078821
MeanShift	34	3.528619e+03	0.812838	6.339370
AgglomerativeWard	100	2.954202e+29	0.983920	0.291820
AgglomerativeAverage	100	2.954184e+29	0.984852	0.295632

Table 2.4: Tabla comparativa caso 2.

A grandes rasgos vemos que todos los algoritmos se comportan de forma relativamente similar a como se comportabane en el caso 1. En cuanto a los tiempos de ejecución, **Meanshift** es el que más consume comparado con los demás. El motivo de que el índice de **Calinski-Harabaz** sea más alto que en la mayoría del resto de algoritmos es que este índice funciona especialmente bien en conjuntos convexos, en aquellos algoritmos basados en distancias. El problema en la medida de nuestros algoritmos jerárquicos puede ser como ya hemos comentado antes debido a la presencia de **outliers**, de nuevo profundizaremos sobre ello en la sección de análisis de parámetros. También destacar el rendimiento de **MiniBatchMeans** en tiempo en contraste de **K-Means** consiguiendo los mismos resultados.

2.3.3 Análisis de parámetros caso 2

En esta sección vamos a probar distintas configuraciones de algunos algoritmos implementados en la sección anterior del caso 2. Nos centraremos en K-Means donde hablaremos del método del codo y en algoritmos jerárquicos.

K-Means

Los resultados de unos algoritmos respecto a **K-Means** se podría deber al número de clústeres elegido, al tener que elegir el número de clusteres, limitamos la potencialidad del método pues puede ser que el número introducido no sea optimo. La idea básica de los algoritmos de clustering es que cada observación se encuentre muy cerca a las de su mismo grupo y los grupos lo más lejos posible entre ellos.
Podemos ver el **método del codo** (elbow method) que utiliza la distancia media de las observaciones a su centroide. Es decir, se fija en las distancias intra-cluster. Cuanto más grande es el número de clusters **k**, la varianza intra-cluster tiende a disminuir. Cuanto menor es la distancia intra-cluster mejor, ya que significa que los clústers son más compactos. El método del codo busca el valor **k** que satisfaga que un incremento de **k**, no mejore sustancialmente la distancia media intra-cluster. Visualizamos esto en la figura [2.11].

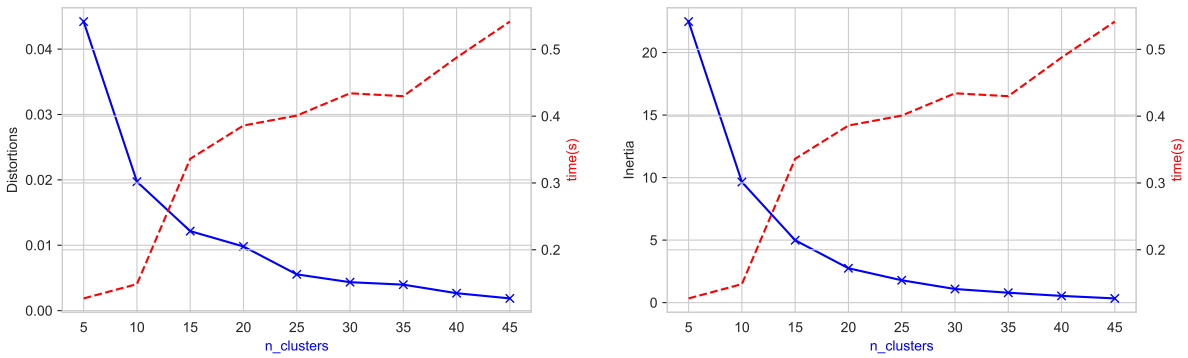


Figure 2.11: Elbow caso 2.

En este análisis, a partir de 10 clusters la reducción en la suma total de cuadrados internos parece estabilizarse, luego es una buena opción, visualizamos esto también en la tabla [2.5] y comparamos las medidas obtenidas con gráficas en la figura [2.12]

	Silhouette	Calinski-Harabaz	time(s)
5	0.770733	5070.608040	0.127164
10	0.825913	5878.782160	0.148570
15	0.905514	7580.415483	0.336152
20	0.916037	10311.729405	0.386008
25	0.948496	12623.225669	0.400867
30	0.951447	17229.540155	0.434225
35	0.951896	20238.943913	0.429920
40	0.967707	26010.334075	0.487753
45	0.974761	36789.381029	0.541600

Table 2.5: Tabla Elbow caso 1.

Se cumple lo que pensabamos y es que el resultado del **K-Means** para este caso de estudio mejora según el número de clusters considerado. Ahora bien, igual que aumentan tanto el índice de **Calinski-Harabaz** como el coeficiente **Silhouette**, también aumenta el tiempo de ejecución del algoritmo. Esto sería algo a tener en un problema de un tamaño suficiente como para que el tiempo nos causara problemas. De todas formas, el algoritmo más lento no ha tardado ni 1 segundo por lo que en nuestro caso no es un problema. También vemos que a partir de 15, como habiamos comentado con el método del codo, nuestros valores se ”estabilizan”. El coeficiente **Silhouette** crece muy rápido al principio quedando casi estancada al final, al contrario que él índice **Calinski-Harabaz**. Al contrario que en el caso anterior, vemos que ahora el valor de **Silhouette** sube algo más ”lineal” que logarítmico.

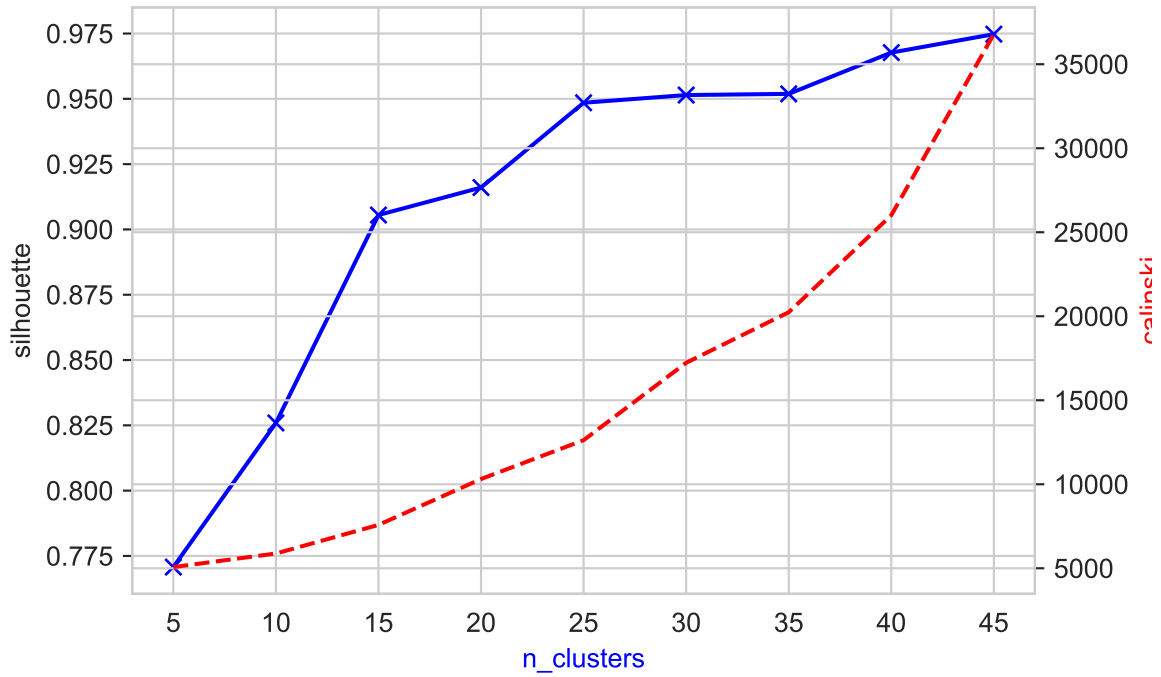


Figure 2.12: Comparación Silhouette y Calinski-Harabaz caso 2.

Aunque a partir de k= 10 se presenten mejores resultados, obtener una solución de este número es muy difícil de interpretar. Por tanto, aunque aumentemos los clusters para comprobar que **K-Means** obtiene buenos resultados, no nos sería útil una solución con tantos clusters.

Algoritmos jerárquicos

Los extraños resultados de las medidas de error para estos algoritmos creemos que se deben a que existen varios clusters con un solo elemento. Como hemos hecho en el caso anterior, vamos a realizar un preprocesado de los datos eliminando los outliers antes de realizar las diferentes configuraciones de este algoritmo. No debería ser necesario que hubiera clusters con un solo elemento dado que estamos considerando 100 clusters en un conjunto de 4291 muestras.

Vamos a considerar que todo clusters con menos de 2 datos es un clusters de valores perdidos. Eliminando estos outliers para 100 clusters obtenemos que **de los 100 clusters hay 44 con más de 2 elementos luego del total de 4291 elementos, seleccionamos 4225**. Realizaremos la comparación de parámetros con este conjunto de **4225** elementos.

Vamos a probar los algoritmos jerárquicos para diferentes clusters donde compararemos como antes los resultados. Obtenemos los resultados y los mostramos en la tabla [2.6].

Algoritmo	Clusters	Calinski	Silhouette	time(s)
Ward-5	5	3158.105007	0.701912	0.449514
Average-5	5	1003.017025	0.652483	0.179692
Ward-10	10	5259.847903	0.849512	0.398993
Average-10	10	1273.633809	0.633926	0.199937
Ward-15	15	7729.315708	0.943058	0.272679
Average-15	15	1194.704065	0.529241	0.241230
Ward-20	20	13275.376273	0.958528	0.281966
Average-20	20	1839.611994	0.652115	0.292453
Ward-25	25	20854.500117	0.976483	0.194090
Average-25	25	1692.560346	0.639964	0.164377
Ward-30	30	35602.242577	0.987464	0.199421
Average-30	30	1883.674949	0.680803	0.160824

Table 2.6: Tabla comparación algoritmos jerárquicos caso 2.

Observamos la mejora de los resultados en la precisión de nuestras medidas. Tanto el valor de **Calinski-Harabaz** como el del coeficiente **Silhouette** aumentan en función del número de clusters considerados. Ward ha obtenido mejores resultados que Average, luego a partir de ahora consideraremos el algoritmo Ward cuando queramos aplicar un jerárquico. A modo de conclusión de esta sección, al contrario que al principio, vemos que estos algoritmos no precisan de un número de clusters muy elevado como escogimos de 100 al principio para su buen funcionamiento, de hecho, con menos clusters han obtenido resultados incluso superiores eliminando los outliers. Esto es importante dada la importancia de la interpretabilidad en este tipo de problemas.

2.3.4 Interpretación de resultados

Para la interpretación de los resultados, vamos a utilizar solo algunos de los algoritmos implementados. Elegiremos aquellos que hayan funcionado bien además de aquellos que no hayan necesitado un alto número de clusters pues esto dificultaría la interpretación.

Primera interpretación

Vamos a realizar la interpretación del primer algoritmo **K-Means** implementado, que usaba 5 clusters pues de las configuraciones iniciales es el que mejores resultados ha obtenido, podemos observar en la figura [2.13] el Scatter Matrix del mismo.
Como en el caso de estudio anterior, vamos a sacar las caracterizaciones de cada cluster generado además de buscar relaciones entre los atributos. A rasgos generales, vamos a utilizar más la primera columna dado que esta representa las relaciones entre los vehículos implicados (**tot_vehiculos_implicados**) y las diferentes víctimas.

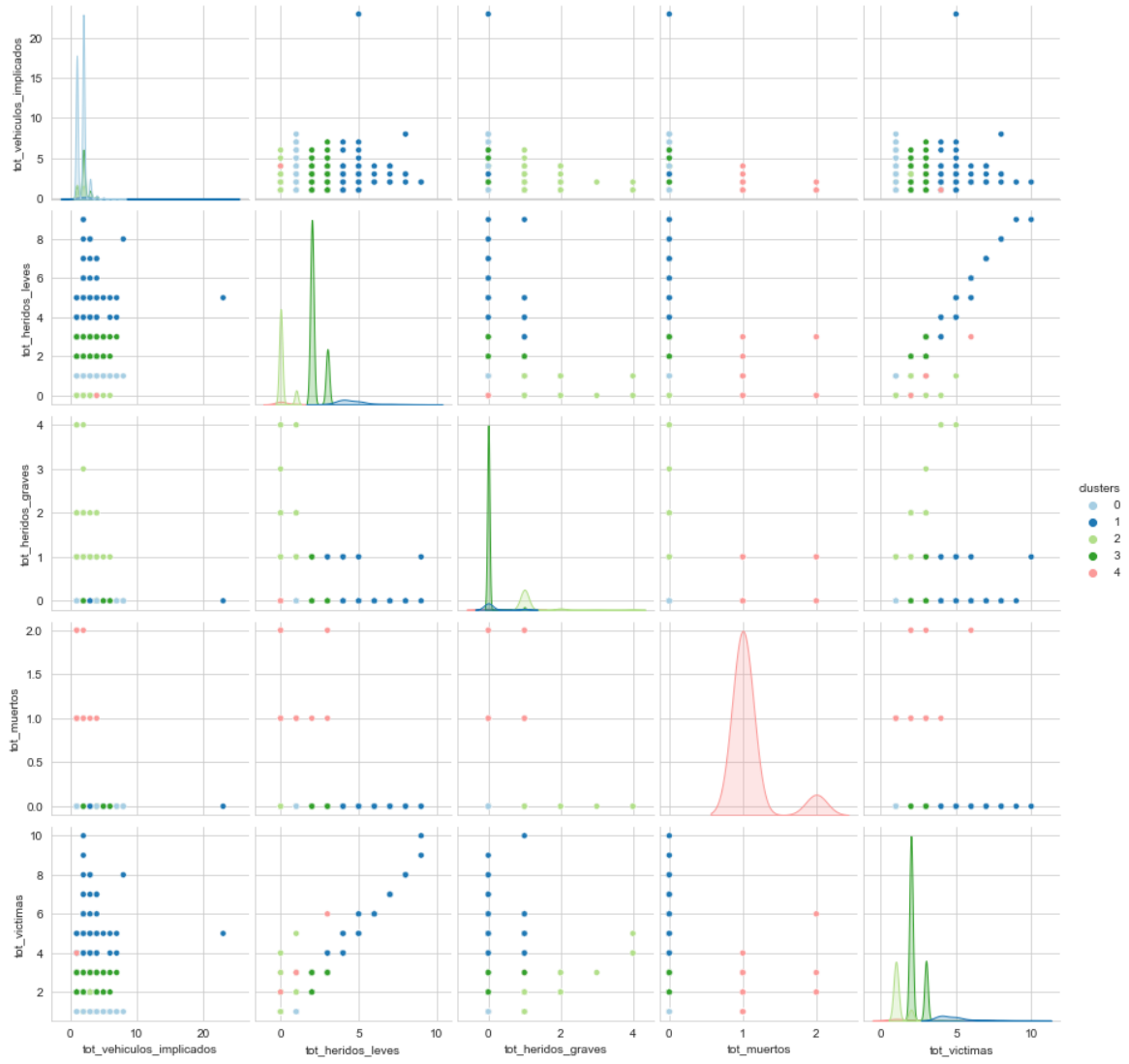


Figure 2.13: Pairplot K-Means 5 clusters caso 2.

De forma general, podemos caracterizar a cada uno de los clusters de la siguiente manera:

- **Cluster 0**: Accidentes que concentran la mayor cantidad de vehículos implicados pero entre 1 y 8 de total de los mismos, sin muertos ni heridos graves pero siempre con algún herido leve.
- **Cluster 1**: Contiene el mayor número de vehículos implicados, desde 0 hasta más de 20, sin muertos ni heridos graves pero con un monton de heridos leve,s, desde 4 hasta 10
- **Cluster 2**: Entre 3 y 4 vehículos implicados, sin muertos pero con muchos heridos graves y sin heridos leves.
- **Cluster 3**: Desde 0 hasta 8 vehículos implicados con 2 o 3 víctimas sin muertos ni heridos graves pero con 2 o 3 heridos leves.

- **Cluster 4**: Solo un vehículo implicados siempre con 4 víctimas, de 1 a 2 muertos (el único que contiene muertos)sin heridos graves ni leves.

En este caso vemos que el cluster 4 destaca por ser el único que contiene fallecidos, luego todos los accidentes mortales están en este cluster y están compuestos por 1 o 2 muertos, seguramente pueden ser debido a una persona o pareja de persona volviendo de madrugada tal vez bajo los efectos del alcohol. También comentar que en la tabla de **tot_vehiculos_implicados** y **tot_heridos_leves** encontramos una relación casi lineal debido a la colocación de los puntos. Complementamos con una visualización de los centroides en la figura [2.14]. Finalmente nos lleva a pensar que la gravedad y cantidad de los heridos es más baja que en otros tipos de accidentes como por ejemplo en el caso anterior.

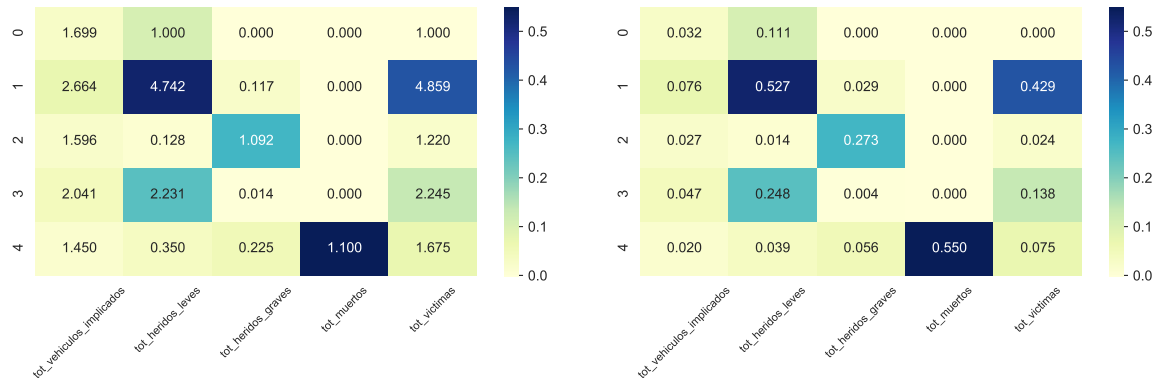


Figure 2.14: Centroides K-Means con datos sin normalizar y normalizados caso 2.

Segunda interpretación

Para esta segunda interpretación, vamos a considerar un algoritmo jerárquico para así poder apoyar nuestro análsis en un heatmap y un dendograma.

Escogemos un algoritmo que nos haya proporcionado buenos resultados pero con un número de de clusters bajo. Por ello, vamos a utilizar el Ward AgglomerativeClustering sin outliers definido en la parte de análisis de parámetros de los algoritmos jerárquicos. Analizamos la Scatter Matrix producida en la figura [2.15].

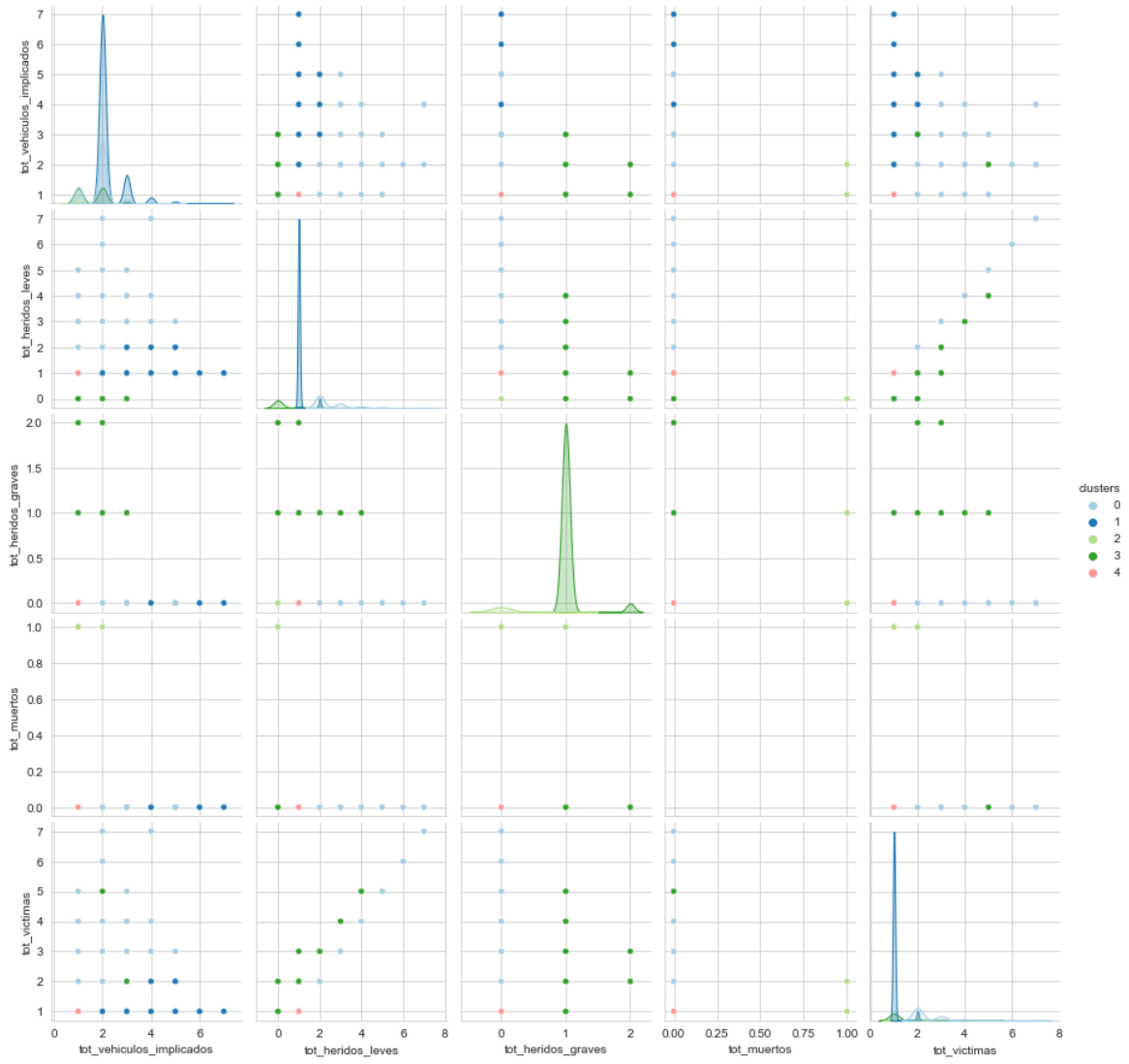


Figure 2.15: Pairplot AgglomerativeClustering 5 clusters caso 2.

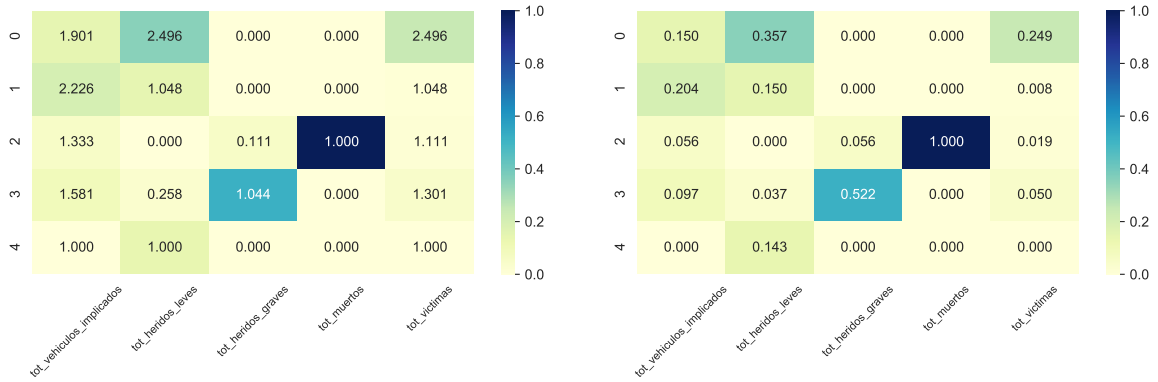


Figure 2.16: Centroides AgglomerativeClustering con datos sin normalizar y normalizados caso 2.

Construimos también un dendrograma para analizar la similitud de los clusters resultantes y cómo se agrupan. Esta será la gráfica principal en la que centraremos nuestro estudio del algoritmo, ya que seguramente es la que más interpretabilidad tiene para un algoritmo jerárquico, lo vemos en la figura [2.17]

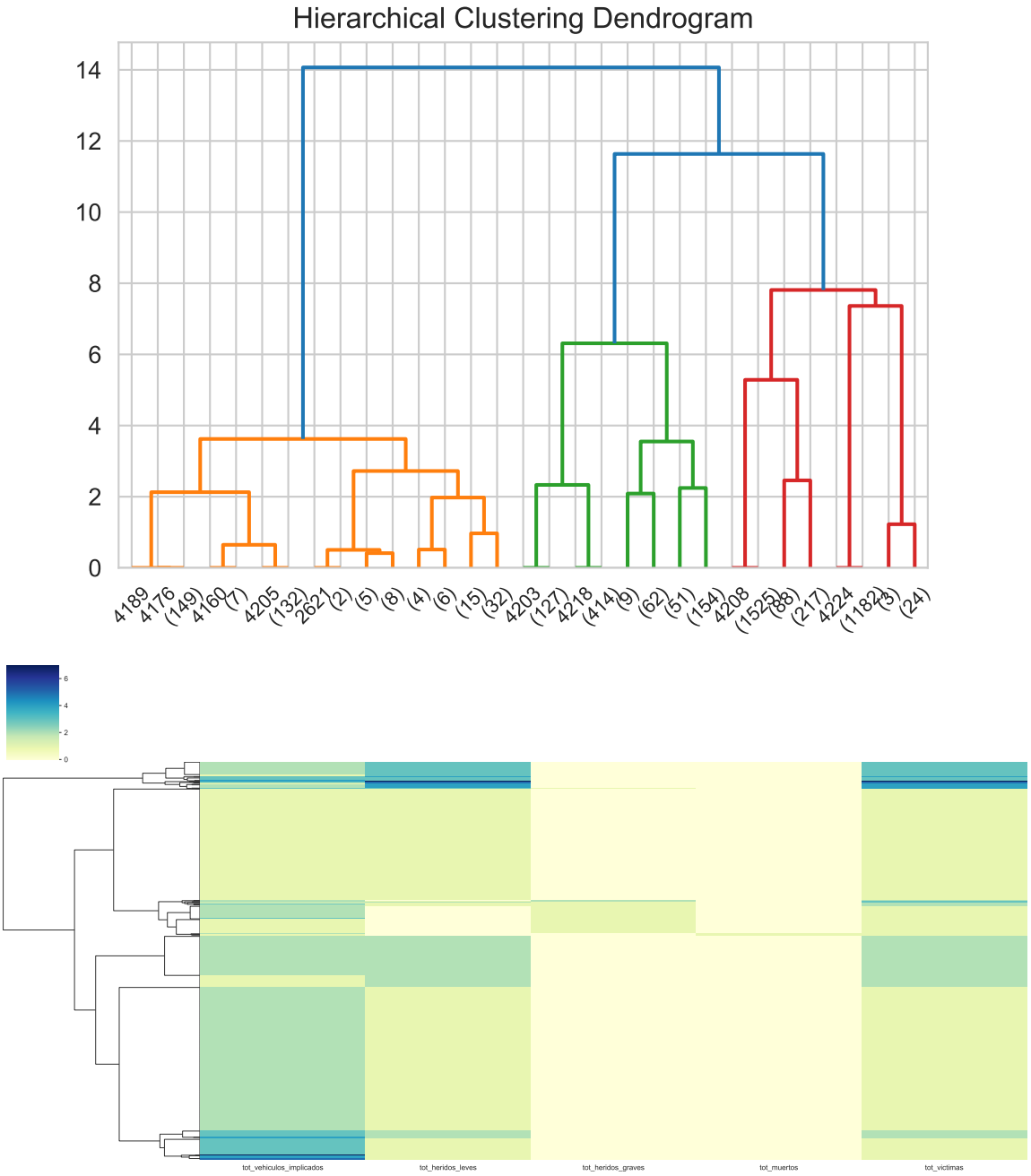


Figure 2.17: Dendograma y Heatmap AgglomerativeClustering caso 2.

Lo primero que llama la atención de este caso de estudio es que la gran mayoría de víctimas mortales se reúnen en uno de los clusters, de ahí la gran diferencia de color en el heatmap del cluster 2 en la columna de muertes con respecto al resto de clus-

ters. También es destacable que este algoritmo para este caso de estudio es capaz de dividir los ejemplos de forma que en el cluster 2 queden aquellos accidentes mortales en los que apenas hay heridos graves ni leves (las víctimas casi en su totalidad fallecen) y sin embargo en el cluster 1 se agrupen los que tienen heridos graves pero apenas fallecidos. Es un comportamiento llamativo del caso de estudio, en el que parece que no hay punto medio a la hora de hacer el recuento de víctimas.
Si observamos la matriz de dispersión se puede apreciar que sí que existen casos de muerte en cada uno de los clusters (en la representación de total de muertos contra total de heridos leves por ejemplo), sin embargo sí es cierto que para el resto de clusters la cantidad de fallecimientos es bastante inferior al de este cluster en cuestión. El resto de clusters siguen una agrupación natural, y básicamente similar a la que encontrábamos en otros ejemplos anteriores, pues la distribución de los accidentes resulta ser similar; existe un cluster para aquellos accidentes graves con varios heridos graves pero pocos o ningún fallecido, y dos para accidentes con heridos leves, uno de ellos con más vehículos implicados y pocas víctimas (el caso del choque en cadena indicado anteriormente) y otro con menos vehículos y más heridos leves, y por consecuente más víctimas.
La agrupación de este algoritmo es bastante buena y sigue unos criterios bastante lógicos para distinguir unos accidentes de otros. Al contar con solo 5 clusters los accidentes están suficientemente bien definidos y el algoritmo se refleja positivamente en las métricas e interpretabilidad de nuestras gráficas.

2.4 Caso de estudio 3

2.4.1 Descripción

Para el tercer caso de estudio, hemos escogido los accidentes que tengan que ver con curvas (tantos fuertes como suaves señalizadas y sin señalizar) con visibilidad restringida (Debido a la configuración del terreno, factores atmosféricos, vegetación y otras causas).[3]
Nos quedaría un conjunto total de 3469 muestras.
De forma análoga al caso anterior la selección de características para el análisis es:

1. tot_victimas
2. tot_muertos
3. tot_heridos_graves
4. tot_heridos_leves
5. tot_vehiculos_implicados

[3]Todo lo relevante a este caso de estudio se recopila en la carpeta de notebooks en el fichero llamado **2_segmentation_accidents_case_3.ipynb** junto al archivo **case_curves.py** donde recopilamos todas las funciones utilizadas.

Vemos una primera visualización de los datos en nuestras variables a estudiar en la figura [2.18], donde apreciamos que la mayoría de los valores de nuestras variables se concentran en los menores valores aunque en general, se encuentra todo tipo de valores.

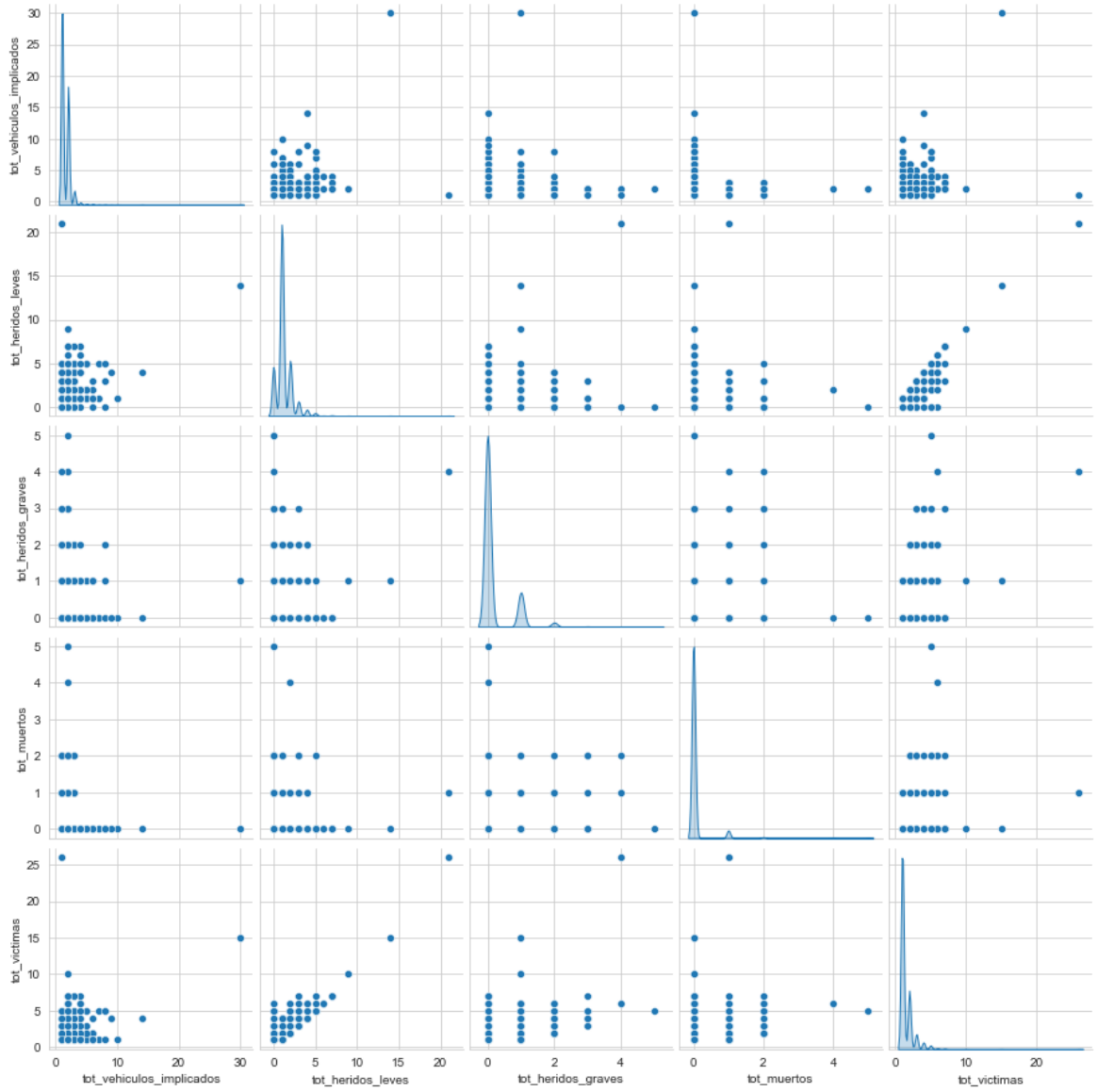


Figure 2.18: Primera visualización de los datos caso 3.

También hacemos una pequeña visualización en el conjunto original para ver como influyen las variables que modificamos en este caso, por ejemplo el **tipo de trazado** y **visibilidad** sobre los atributos de **tot_muertos** y **tot_victimas** en la figura [2.19]

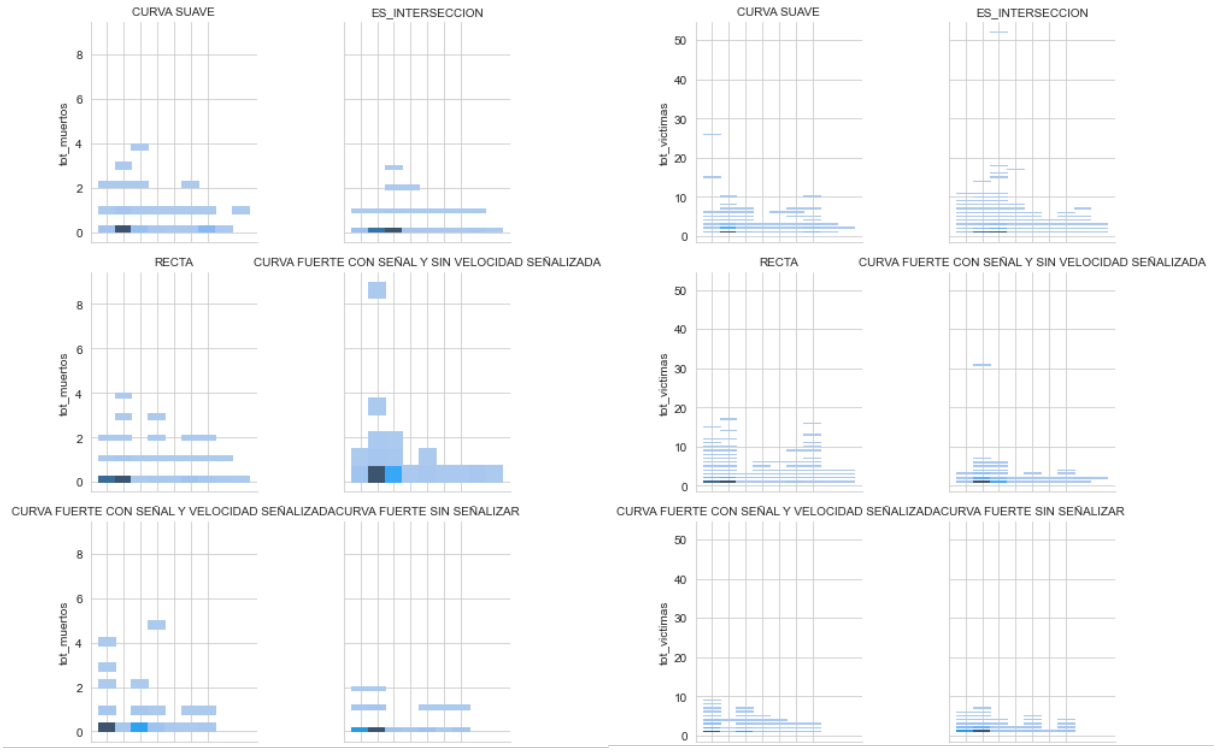


Figure 2.19: Visualización de tot_muertos y tot_victimas respecto el ktrazado y visibilidad.

2.4.2 Análisis del caso de estudio 3

Para el análisis de este caso de estudio hemos considerado 5 algoritmos de clustering diferentes:

- **K-Means** con parámetros **n_clusters** = 5, **n_init** = 100 e **init** = k-means++
- **MiniBatchKMeans** con **n_clusters** = 5 y **init** = k-means++.
- **MeanShift** con bandwidth estimado usando *estimate_bandwidth* usando como parámetro *quantile* = 0.3.
- **AgglomerativeClustering** con **n_clusters** = 5 y **linkage** = 'ward'.
- **Birch** con **n_clusters** = 5 y **threshold** = 0.1

En la tabla [2.7] vemos los resultados obtenidos para cada uno de estos algoritmos.

Algoritmo	Clusters	Calinski	Silhouette	time(s)
K-Means	5	2692.953520	0.719600	2.530842
MiniBatchKMeans	5	2651.924240	0.670398	0.028871
MeanShift	34	1324.001177	0.703094	8.998035
Agglomerative	5	2641.686463	0.671854	0.235597
Birch	5	160.052353	0.825009	0.067555

Table 2.7: Tabla comparativa caso 3.

En cuanto al algoritmo de referencia, **K-Means**, ha obtenido resultados relativamente buenos para el coeficiente de Silhouette pero como siempre ha obtenido un resultado bastante mejor en el índice de Calinski-Harabaz, consiguiendo un valor superior al del resto de algoritmos mostrados.
El resto de algoritmos tienen un comportamiento similar exceptuando el algoritmo **Birch** con el coeficiente de **Calinski**, aunque en general, hemos conseguido malos coeficientes de Silhouette, el que mejor coeficiente proporciona es el algoritmo Birch a costa de una muy mala puntuación para el indice de Calinski-Harabaz.

2.4.3 Análisis de parámetros caso 3

En esta sección vamos a realizar diferentes configuraciones de algunos de los algoritmos de la sección anterior. Con las configuraciones buscamos mejorar aquellos métodos que nos han dado un resultado peor del esperado o estudiar el comportamiento de estos al cambiar algunos parámetros.

K-Means

Los resultados de unos algoritmos respecto a **K-Means** se podría deber al número de clústeres elegido, al tener que elegir el número de clusteres, limitamos la potencialidad del método pues puede ser que el número introducido no sea optimo. La idea básica de los algoritmos de clustering es que cada observación se encuentre muy cerca a las de su mismo grupo y los grupos lo más lejos posible entre ellos. Vamos a probar utilizar el método del codo para ver cuál es el número óptimo de clusters de este algoritmo, de momento con el número utilizado, vemos que el algoritmo es costoso a nivel de tiempo.
Podemos ver el **método del codo** (elbow method) que utiliza la distancia media de las observaciones a su centroide. Es decir, se fija en las distancias intra-cluster. Cuanto más grande es el número de clusters **k**, la varianza intra-cluster tiende a disminuir. Cuanto menor es la distancia intra-cluster mejor, ya que significa que los clústers son más compactos. El método del codo busca el valor **k** que satisfaga que un incremento de **k**, no mejore sustancialmente la distancia media intra-cluster. Visualizamos esto en la figura [2.20]. Recordamos que el tiempo calculado es orientativo y depende de de muchos factores, lo que esta claro es que a más clusters, más aumenta el tiempo que tarda el algoritmo.

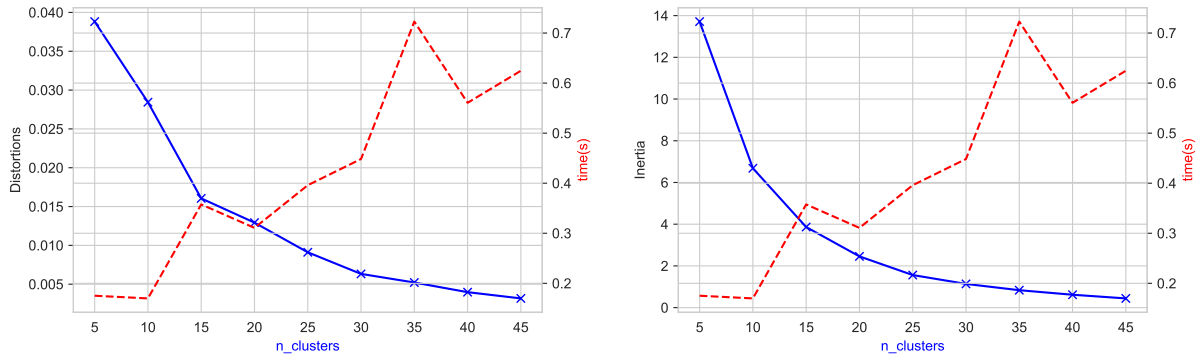


Figure 2.20: Elbow caso 3.

En este análisis, a partir de 10 clusters la reducción en la suma total de cuadrados internos parece estabilizarse, indicando que a partir k = 10 es una buena opción, visualizamos esto también en la tabla [2.8] y comparamos las medidas obtenidas con gráficas en la figura [2.21]

Clusters	Silhouette	Calinski-Harabaz	time(s)
5	0.719600	2692.953520	0.932284
10	0.701475	2857.443413	0.473481
15	0.814644	3351.410114	0.282286
20	0.833584	3984.041599	0.411960
25	0.869885	5034.601789	0.362263
30	0.911997	5759.570238	0.387246
35	0.918180	6714.757559	0.524351
40	0.937985	7930.292564	0.542527
45	0.947254	9849.629927	0.615982

Table 2.8: Tabla Elbow caso 3.

Se cumple lo que pensabamos y es que el resultado del **K-Means** para este caso de estudio mejora según el número de clusters considerado. Ahora bien, igual que aumentan tanto el índice de **Calinski-Harabaz** como el coeficiente **Silhouette**, también aumenta el tiempo de ejecución del algoritmo. Esto sería algo a tener en un problema de un tamaño suficiente como para que el tiempo nos causara problemas. De todas formas, el algoritmo más lento no ha tardado ni 1 segundo por lo que en nuestro caso no es un problema. También vemos que a partir de 10, como habiamos comentado con el método del codo, nuestros valores se ”estabilizan”. Vemos que a diferencia de los casos anteriores, ambos coeficientes crecen muy rápido.

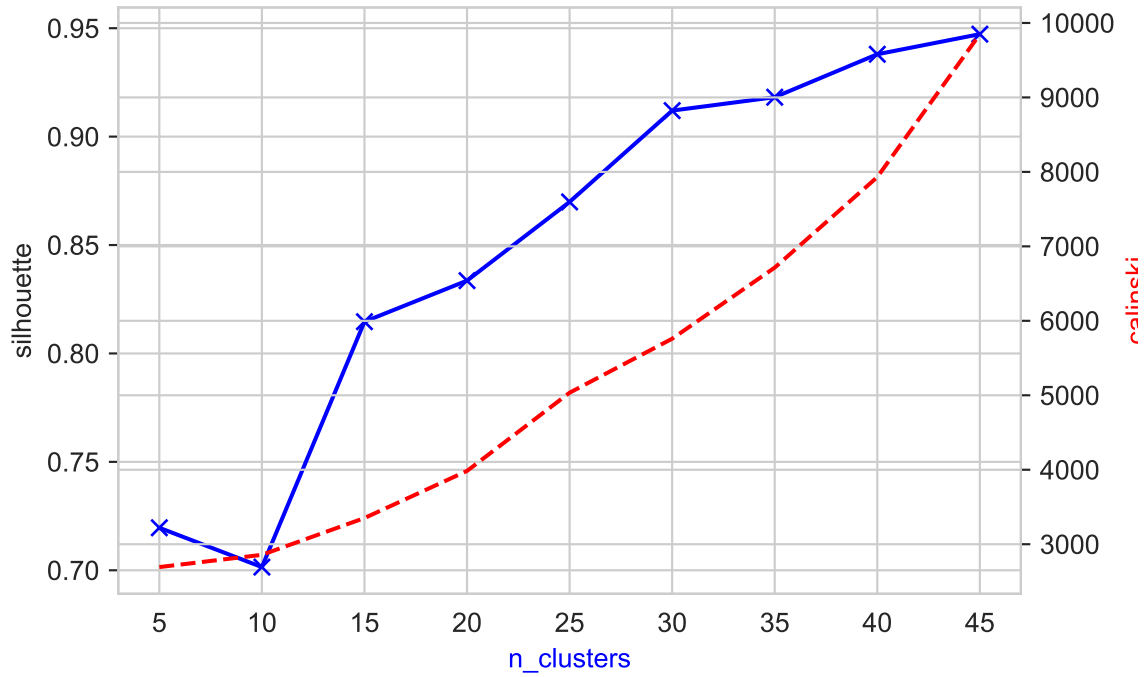


Figure 2.21: Comparación Silhouette y Calinski-Harabaz caso 3.

Birch

Como ya hemos visto en los comentarios generales, este algoritmo es el que obtiene peores resultados en ambas medidas del error. Por ello, nos planteamos si aumentando el número de clusters podría llegar a ser competitivo con el resto de algoritmos. En la figura [2.9] vemos distintos resultados para nuestro algoritmo Birch manteniendo el **threshold** a 0.01.

Algoritmo	Clusters	Calinski	Silhouette	time(s)
Birch-10	10	1914.874268	0.735079	0.190431
Birch-20	20	1525.303095	0.555075	0.121747
Birch-30	30	1132.892304	0.548451	0.164765
Birch-40	40	1405.511458	0.571084	0.284661
Birch-50	50	1319.315083	0.522287	0.124964
Birch-60	60	2927.739050	0.698816	0.115032

Table 2.9: Tabla comparativa clusters Birch caso 3.

Los datos obtenidos son curiosos pues vemos que, en las primeras configuraciones ambos coeficientes de error oscilan aumentando o disminuyendo. Esto se puede deber a la influencia negativa que tiene el número de clusters utilizado en el índice de Calinski-Harabaz, y es que los resultados no parecen mejorar en exceso según el

coeficiente Silhouette, pero sin embargo estamos aumentando en 10 el número de clusters considerados. También vemos que igualmente respecto a nuestra configuración inicia de 5 clusters, la mejora es clara en el índice de Calinski-Harabaz aunque disminuya Silhouette.
Analizamos tambíen la influencia del threshold para Birch-10, ya que ha conllevado mejoras con respecto a Birch-5 (implementado inicialmente) y sigue siendo interpretable. Recordamos que en apartado anterior, una disminución del valor de threshold nos proporcionaba peores resultados, veamos si aumentando el threshold en este caso conseguimos mejorar los resultados ya obtenidos. Vemos los resultados en la figura [2.10], los valores de threshold que elegimos son 0.01, 0.05 y 0.07.

Algoritmo	Clusters	Calinski	Silhouette	time(s)
Birch-01	10	1914.874268	0.735079	0.411858
Birch-05	10	226.310697	0.581081	0.236941
Birch-07	10	322.887305	0.609452	0.113330

Table 2.10: Tabla comparativa threshold Birch caso 3.

Nuestros resultados empeoran en todos los sentidos.

MeanShift

En todos los casos en los que hemos utilizado alguna configuración del MeanShift en general hemos obtenido muy buenos resultados. Sin embargo, el número de clusters utilizado ha sido demasiado alto. En el resto de ocasiones, lo que hemos hecho ha sido aumentar el número de clusters en otros métodos para conseguir superar a este algoritmo, sin embargo, esto no es útil en problemas de agrupamiento pues necesitamos en la mayoría de los casos obtener conclusiones de los grupos formados. Nos disponemos a configurar el parámetro del **bandwidth** (ancho de banda) en el algoritmo de MeanShift, para esto, vamos a variar el parámetro del **quantile** en el método que lo estima. Tomaremos valor de **quantile** = 0.2, 0.3, 0.4 y 0.5, obteniendo los resultados de la figura [2.17]

Algoritmo	Clusters	Calinski	Silhouette	time(s)
MeanShift-02	35	1306.491296	0.700947	6.026729
MeanShift-03	34	1324.001177	0.703094	5.910125
MeanShift-04	31	861.350696	0.710210	5.891991
MeanShift-05	30	870.245175	0.711891	5.833468

Table 2.11: Tabla comparativa quantile Meanshift caso 3.

Ocurre lo que esperabamos y es que el número de clusters necesitados ha descendido. Sin embargo, no hemos obtenido un método competitivo con el resto de algoritmos implementados hasta el momento pues el coeficiente Silhouette supera mínimanete 0.7 y el índice de Calinski-Harabaz es excesivamente alto. Lo que es

más, el tiempo de ejecución ha aumentado bastante, por lo que no hemos conseguido ninguna configuración del MeanShift que merezca la pena pues no hemos obtenido el equilibro entre buenos resultados e interpretabilidad.

AgglomerativeClustering

Como último algoritmo jerárquico, vamos a probar a implementar una matriz de conectividad [Con].
Esta matriz se define utilizando un grafo de vecinos con la función **kneighbors_graph** que recibe como parámetro el número de vecinos que consideramos y si queremos que se le incluya a sí mismo como vecino más cercano. Posteriormente, se hace esta matriz simétrica y se introduce como connectivity.
Vamos a comparar la diferencia en cuanto a las medidas consideradas si introducimos este parámetro como si no lo introducimos para valores de clusters 5 y 10. Además, vamos a mantener el método considerado la configuración inicial, el Ward y estableceremos como número de vecinos a considerar 10 en ambos casos. Obtenemos la tabla [2.12]

Algoritmo	Clusters	Calinski	Silhouette	time(s)
Ward-10	10	2701.013198	0.683822	1.023627
Ward-10-con	10	2727.413198	0.684460	4.269531
Ward-20	20	3760.914458	0.842622	0.230655
Ward-20-con	20	3420.257939	0.842291	3.821585

Table 2.12: Tabla AgglomerativeClustering conectividad caso 3.

Vemos que no se han producido mejoras al utilizar esta matriz de conectividad. Además, el tiempo necesitado se ha disparado siendo de los algoritmos más lentos que hemos utilizado a lo largo de esta práctica. Por tanto, podemos concluir categóricamente que mejor no utilizar este parámetro y dejar funcionar al método por si solo.
Además, comprobamos que las dos configuraciones implementadas utilizando más número de clusters mejoran bastante a la configurada anteriormente, por lo que, sí ha sido buena idea aumentar los clusters a considerar para la solución.

2.4.4 Interpretación de resultados

Para la interpretación de la segmentación, vamos a utilizar solo algunos de los algoritmos utilizados. Para la elección, usamos aquellos que hayan funcionado bien además de tener un bajo número de clusters por la interpretabilidad.

Primera Interpretación

Para la primera interpretación elegimos, como siempre, el algoritmo **K-Means** con la configuración inicial de 5 clusters. Utilizaremos la Scatter Matrix de la figura [2.22]

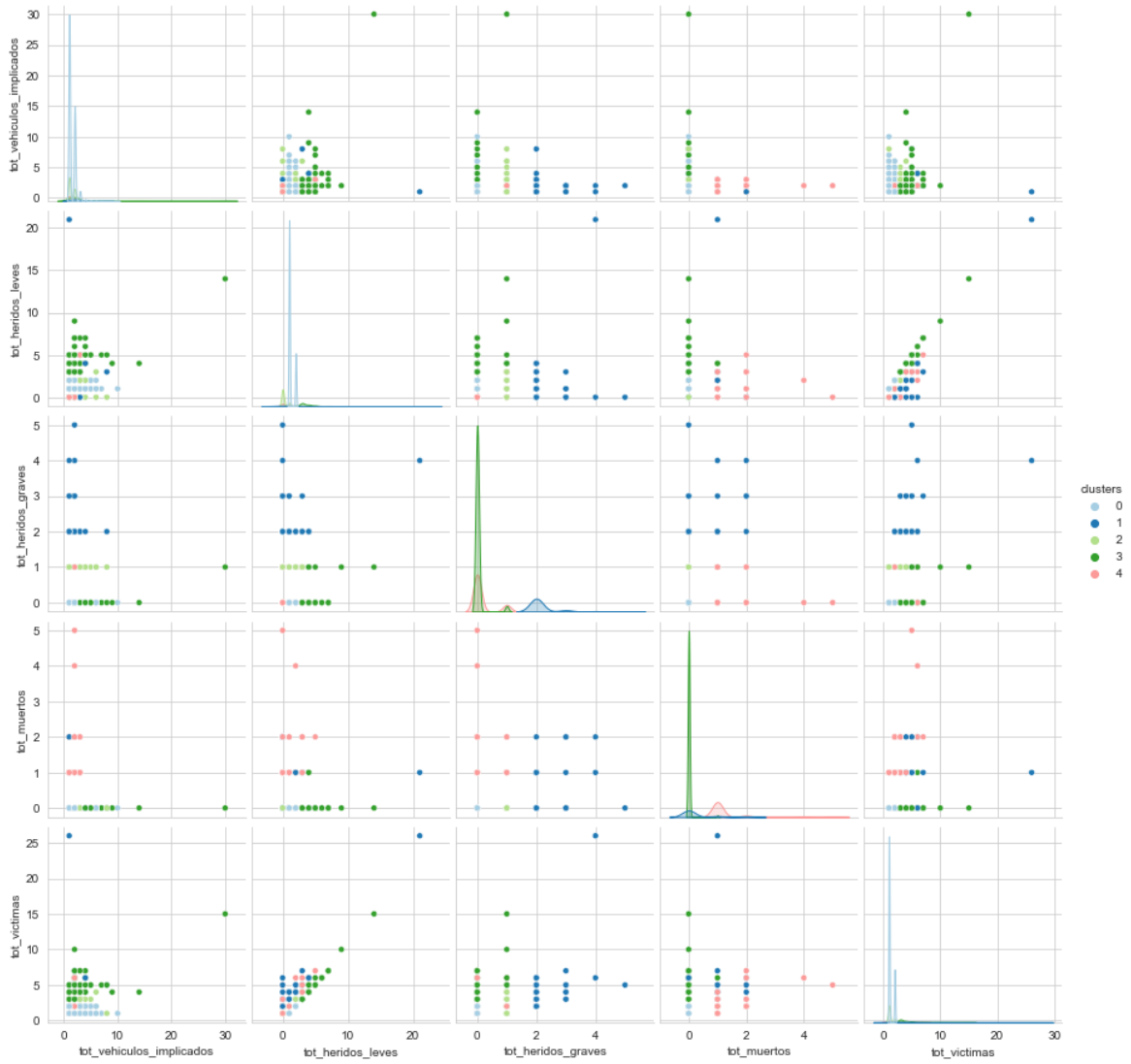


Figure 2.22: Pairplot K-Means 5 clusters caso 3.

Como en los casos anteriores, podemos caracterizar a cada uno de los clusters de la siguiente manera centrandonos en la primera columna y relacionandolo con vehículos implicados:

- **Cluster 0**: En este cluster encontramos entre 1 y 9 vehículos implicados, siempre con víctimas entre 1 y 4, sin muertos pero siempre con 1 herido grave y 0 o 2-3 heridos leves, estando la mayoría de vehículos implicados entre 1 y 4.
- **Cluster 1**: Como antes, tenemos entre 1 y 10 vehículos implicados, entre 1 y 3 víctimas, sin muertos ni heridos graves pero siempre con heridos leves entre 1 y 3.
- **Cluster 2**: El cluster que contiene el mayor número de vehículos implicados, entre 1 y 15, con muchas víctimas entre 3 y 10, sin muertos ni víctimas pero con muchos heridos leves entre 3 y 15.

- **Cluster 3**: Agrupa las muestras de 2 vehículos implicados, pero siempre con muertos entre 1 o 5, 1 heridos grave y 0 o 5 heridos leves.

- **Cluster 4**: Contiene los grupos de 5 vehículos implicados con mayor cantidad de víctimas, 6 o más de 25, con 2 muertos y el que más cantidad de heridos graves posee, entre 2 y 5 con 0 o 4 heridos leves.

En cuanto a las relaciones entre las variables, observamos una dependencia lineal entre tot_victimas y tot_heridos_leves, de forma que cuanto más aumenta el número de víctimas más aumenta el número de heridos leves. Esto nos indica que la mayoría de las víctimas producidas en estos accidentes son leves.

Destacamos el caso de pocos vehículos implicados pero gran cantidad de víctimas, como puede ser el cluster 4, ya que puede ser el caso de vehículos grandes como autobuses, furgonetas y derivados, vehículos que transporten gran cantidad de gente.

Aunque en general los clusters son bastante similares, (vemos la agrupación de los puntos) en el sentido de más coches implicados aumentan las demás variables, excepto por el caso del cluster 4 como hemos comentado antes. Mostramos los centroides en la figura [2.23] para apoyar este análisis. Vemos que hay un cluster que sobresale que es el que contiene la mayor cantidad de heridos graves, el cluster 1 que ya hemos comentado anteriormente.

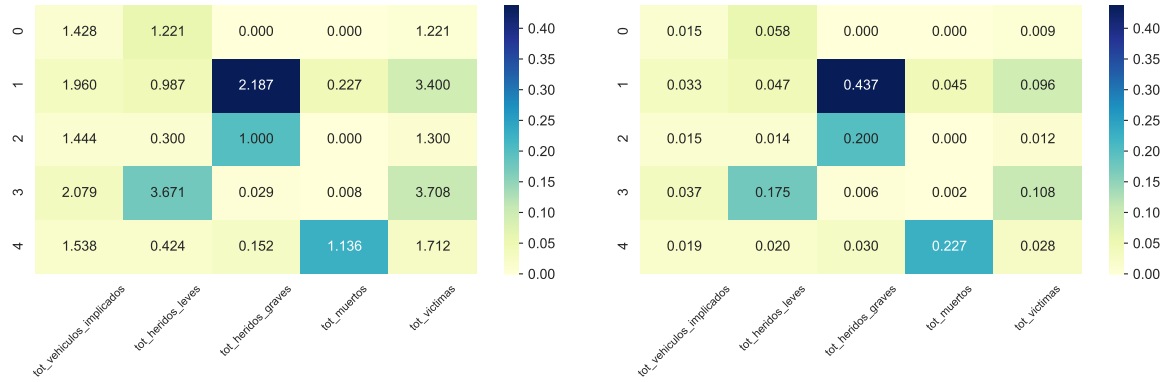


Figure 2.23: Centroides K-Means con datos sin normalizar y normalizados caso 3.

Segunda Interpretación

Para esta interpretación vamos a utilizar el algoritmo jerárquico implementado en los 5 algoritmos iniciales, un AgglomerativeClustering Ward con 5 clusters. Visualizamos la Scatter Matrix en la figura [2.24]

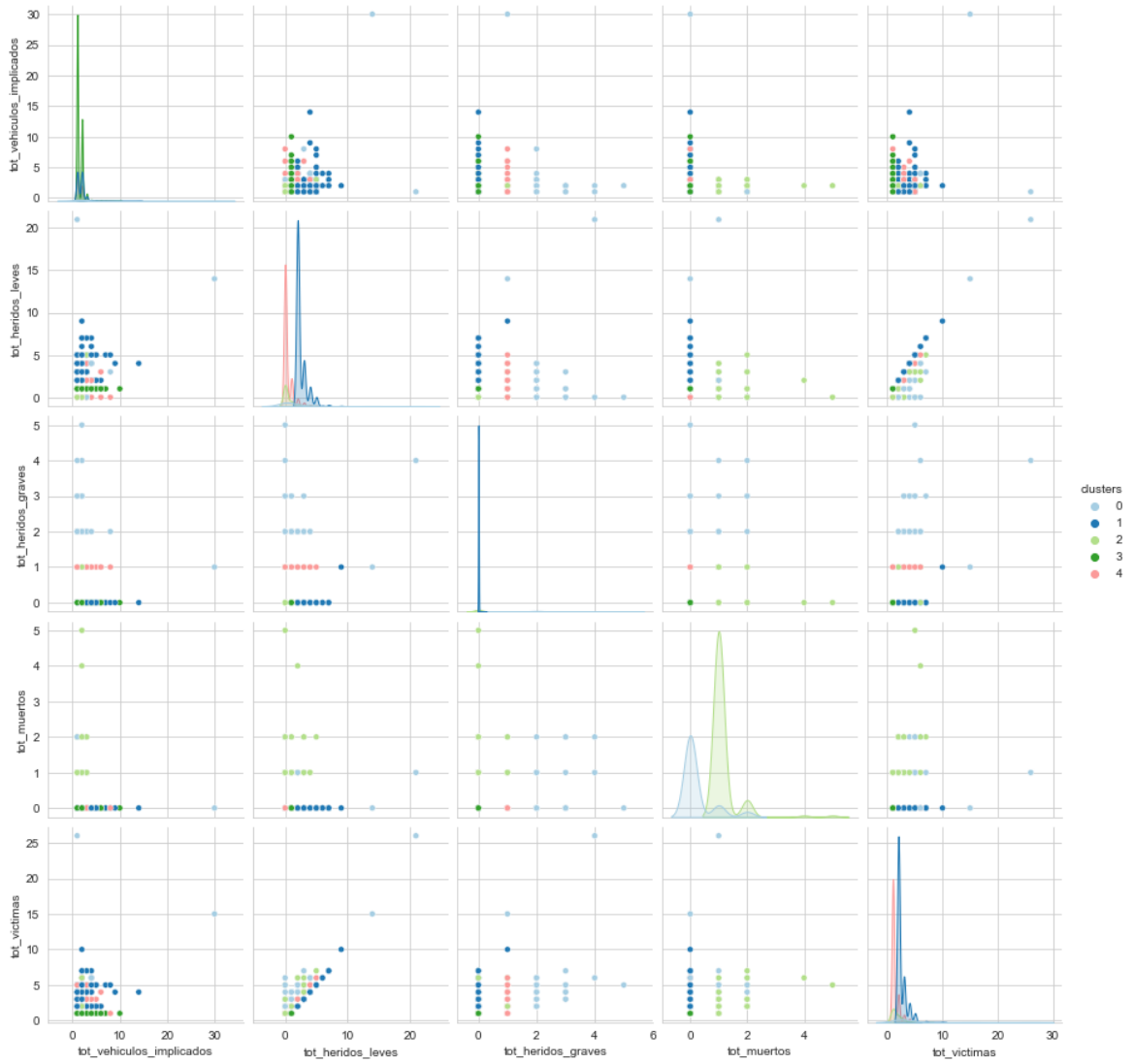


Figure 2.24: Pairplot AgglomerativeClustering 5 clusters caso 3.

Como hasta el momento, obtenedremos caracterizaciones de los diferentes clusters además de buscar relaciones entre las variables utilizadas. Se han generado 5 clusters, y las características generales de estos son:

- **Cluster 0**: Tiene todo número de vehículos implicados, desde 1 hasta 30 no en grandes cantidades, y con totales de víctimas por encima de 5, 15 o 26, sin muertos pero con muchos heridos graves entre 2 y 5 con bastantes heridos leves entre 3, 14 o 22.
- **Cluster 1**: Bastantes vehículos implicados entre 1 y 15 estando mas condensado entre 1 y 10, víctimas entre 1 y 5, 10 no tantas como en el cluster 1, sin muertos ni heridos graves pero con heridos leves.
- **Cluster 2**: Pocos vehículos implicados, solo 1 caso de muestras y es menos de 5, el cluster con más número de muertos entre 1 y 5, con 1 herido grave y sin heridos leves

- **Cluster 3**: Casos de vehículos implicados entre 1 y 10, con el menor número de víctimas, sin muertos ni heridos graves pero con algunos heridos leves (menos de 5).

- **Cluster 4**: Vehículos implicados en el intervalo entre 1 y 9, con víctimas entre 1 y 5, sin muertos pero siempre con 1 herido grave y a veces algún herido leve pero siempre menos de 5 heridos leves.

En cuanto a las conclusiones generales en función del tamaño y las dependencias entre variables, son las mismas que en la interpretación anterior. De igual manera vemos que el número de víctimas aumenta conforme aumentan los heridos leves, esto puede ser debido a que parte de las víctimas en estos accidentes no pertenecen a los vehículos o que pertecían a un vehículo bastante grande como hemos comentado en la primera interpretación, reforzando así el comentario. De nuevo nos apoyamos en la visualización de los centroides en la figura [2.25] junto a un dendográma en la figura[2.26]

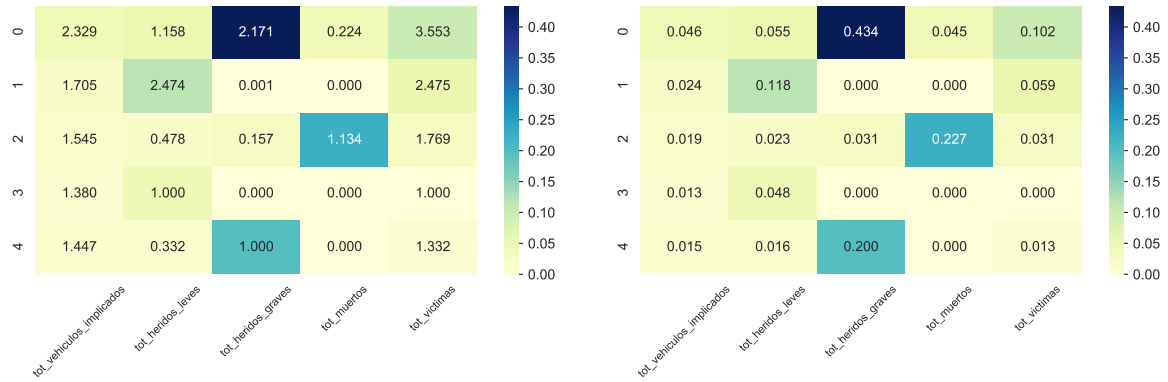


Figure 2.25: Centroides AgglomerativeClustering con datos sin normalizar y normalizados caso 3.

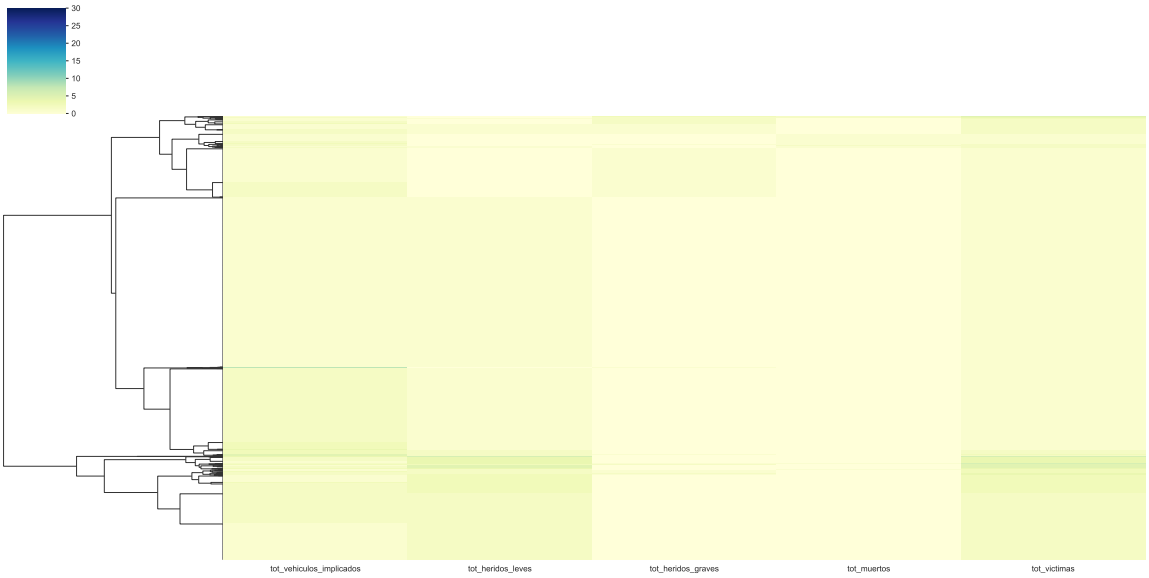


Figure 2.26: Dendograma y Heatmap AgglomerativeClustering caso 3.

2.5 Caso de estudio 4

2.5.1 Descripción

Para el tercer cuarto de estudio, de forma análoga a l caso de estudio hemos escogido los accidentes que tengan que ver con curvas (tantos fuertes como suaves señalizadas y sin señalizar) pero en esta ocasión sin visibilidad restringida para comparar [4]
Nos quedaría un conjunto total de 9249 muestras, bastante más grande que el caso 3.
De forma análoga al caso anterior la selección de características para el análisis es:

1. tot_victimas
2. tot_muertos
3. tot_heridos_graves
4. tot_heridos_leves
5. tot_vehiculos_implicados

Vemos una primera visualización de los datos en nuestras variables a estudiar en la figura [2.27], donde apreciamos que la mayoría de los valores de nuestras variables se concentran en los menores valores aunque en general, se encuentra todo tipo de valores. Vemos que a diferencia que en el caso anterior, ahora abarcamos valores no tan agrupados y si más alejados.

[4]Todo lo relevante a este caso de estudio se recopila en la carpeta de notebooks en el fichero llamado **2_segmentation_accidents_case_4.ipynb** junto al archivo **case_curves.py** donde recopilamos todas las funciones utilizadas.

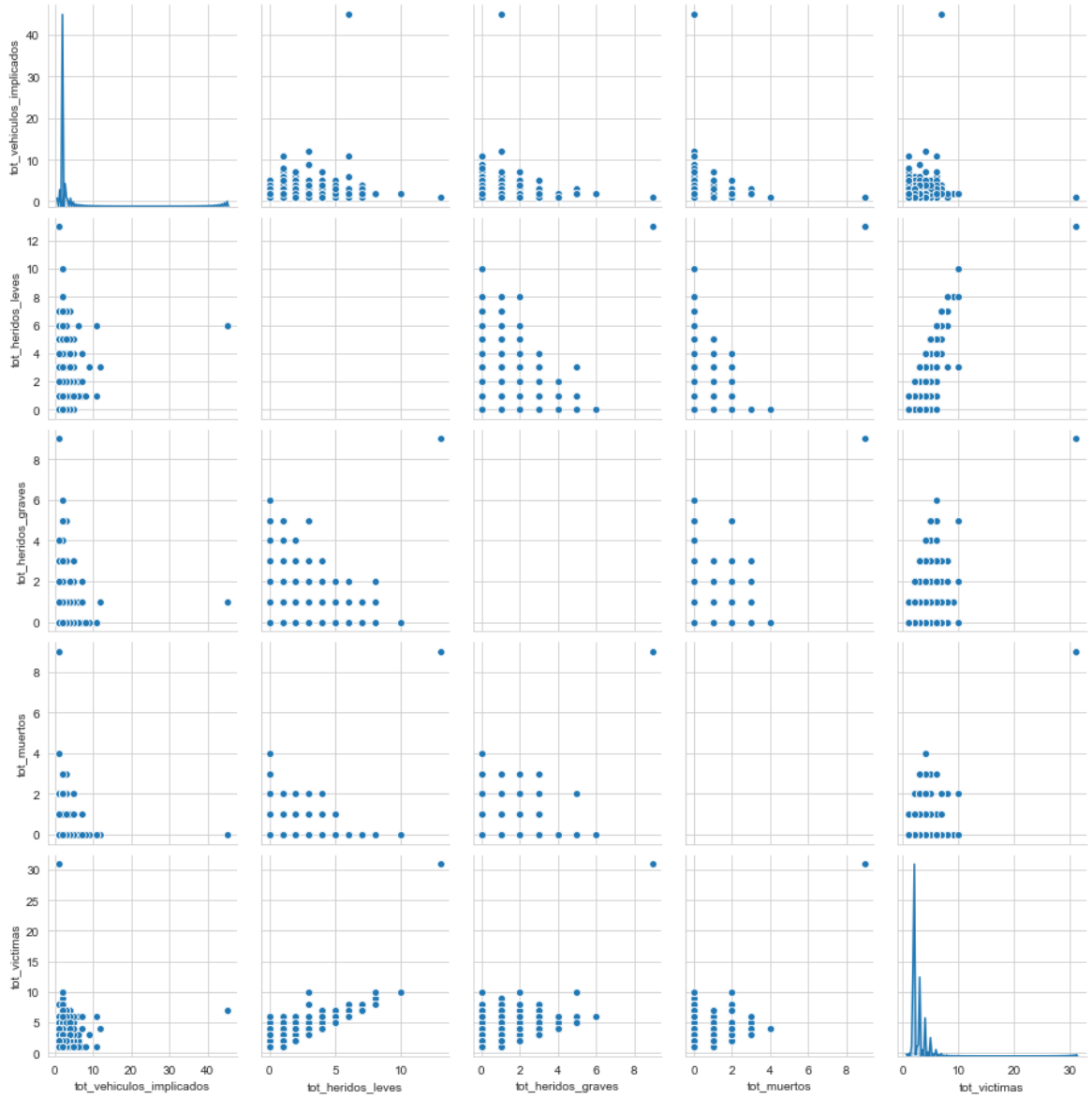


Figure 2.27: Primera visualización de los datos caso 4.

Recordamos la visualización en el conjunto original para ver como influyen las variables que modificamos en este caso, en la figura [2.19].

2.5.2 Análisis del caso de estudio 4

Para el análisis de este caso de estudio consideramos los mismos 5 algoritmos de clustering que en el caso anterior, con la misma configuración inicial de parámetros:

- **K-Means** con parámetros **n_clusters** = 5, **n_init** = 100 e **init** = k-means++
- **MiniBatchKMeans** con **n_clusters** = 5 y **init** = k-means++.
- **MeanShift** con bandwidth estimado usando *estimate_bandwidth* usando como parámetro *quantile* = 0.3.

- **AgglomerativeClustering** con **n_clusters** = 5 y **linkage** = 'ward'.
- **Birch** con **n_clusters** = 5 y **threshold** = 0.1

En la tabla [2.13] vemos los resultados obtenidos para cada uno de estos algoritmos.

Algoritmo	Clusters	Calinski	Silhouette	time(s)
K-Means	5	7190.384916	0.748319	4.120552
MiniBatchKMeans	5	6841.426275	0.726542	0.038931
MeanShift	80	5958.743235	0.856150	46.143938
Agglomerative	5	6675.223218	0.740896	3.174479
Birch	5	352.947363	0.777686	0.230513

Table 2.13: Tabla comparativa caso 4.

Vemos que respecto al caso anterior, se nota bastante la mejoría al tener un conjunto de datos mayor, ya que en tiempos relativamente similares conseguimos medidas mucho más prometedoras. **K-Means** ha obtenido los mejores resultados relativamente buenos para el coeficiente de Silhouette pero como siempre ha obtenido un resultado bastante mejor en el índice de Calinski-Harabaz, consiguiendo un valor superior al del resto de algoritmos mostrados a costa de un tiempo de ejecución considerablemente mayor para haber utilizado 5 clusters.
Todos los algoritmos han obtenido puntuaciones aceptables excepto **Birch**, que incluso comparandolo con el caso anterior que ya daba malos resultados ahora vuelve a dar malos resultados. También destacar que los buenos resultados de **Meanshift** no pueden tenerse muy en cuenta debido a que ha utilizado 80 clusters, en la sección siguiente veremos si usando un número mucho menor de clusters podemos alcanzar resultados similares.

2.5.3 Análisis de parámetros caso 4

K-Means

Los resultados de unos algoritmos respecto a **K-Means** se podría deber al número de clústeres elegido, al tener que elegir el número de clusteres, limitamos la potencialidad del método pues puede ser que el número introducido no sea optimo. La idea básica de los algoritmos de clustering es que cada observación se encuentre muy cerca a las de su mismo grupo y los grupos lo más lejos posible entre ellos. Vamos a utilizar el método del codo para ver cuál es el número óptimo de clusters de este algoritmo, de momento con el número utilizado, vemos que el algoritmo es costoso a nivel de tiempo.
Como en el caso anterior usamos de nuevo el **método del codo** (elbow method) que utiliza la distancia media de las observaciones a su centroide. Es decir, se fija en las distancias intra-cluster. Cuanto más grande es el número de clusters **k**, la varianza intra-cluster tiende a disminuir. Cuanto menor es la distancia intra-cluster mejor, ya que significa que los clústers son más compactos. El método del codo busca el

valor **k** que satisfaga que un incremento de **k**, no mejore sustancialmente la distancia media intra-cluster. Visualizamos esto en la figura [2.28]. Recordamos que el tiempo calculado es orientativo y depende de de muchos factores, lo que esta claro es que a más clusters, más aumenta el tiempo que tarda el algoritmo.

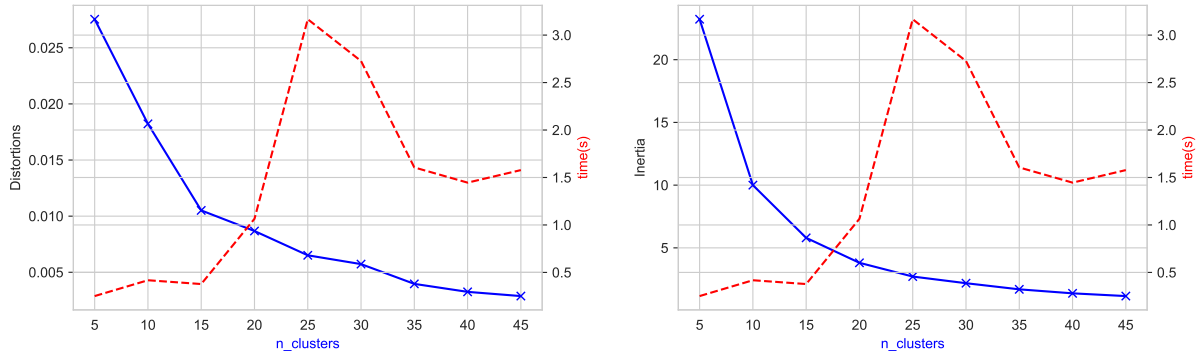


Figure 2.28: Elbow caso .

En este análisis, a partir de 10 clusters la reducción en la suma total de cuadrados internos parece estabilizarse, indicando que a partir k = 10 es una buena opción, visualizamos esto también en la tabla [2.14] y comparamos las medidas obtenidas con gráficas en la figura [2.29] vemos que comparado con el caso anterior, en este caso las medidas aumentan casi más lineal que en el caso anterior. Aunque al final parece que nuestro valor de **Silhouette** empieza a estabilizarse

Clusters	Silhouette	Calinski-Harabaz	time(s)
5	0.748319	7190.384916	0.250879
10	0.802218	8775.695554	0.417586
15	0.863620	10201.536284	0.377539
20	0.877710	11690.595530	1.064875
25	0.893110	13137.339275	3.166977
30	0.895435	13584.042354	2.726878
35	0.928683	15000.817778	1.605816
40	0.944435	16157.238143	1.445833
45	0.943412	17042.840381	1.577876

Table 2.14: Tabla Elbow caso 4.

Se cumple lo que pensabamos y es que el resultado del **K-Means** para este caso de estudio mejora según el número de clusters considerado. Ahora bien, igual que aumentan tanto el índice de **Calinski-Harabaz** como el coeficiente **Silhouette**, también aumenta el tiempo de ejecución del algoritmo. Esto sería algo a tener en un problema de un tamaño suficiente como para que el tiempo nos causara problemas. De todas formas, el algoritmo más lento ha tardado como máximo 3 segundos por lo

que en nuestro caso no es un problema. También vemos que a partir de 10, como habiamos comentado con el método del codo, nuestros valores se ”estabilizan”. Vemos que a diferencia de los casos anteriores, ambos coeficientes crecen muy rápido.

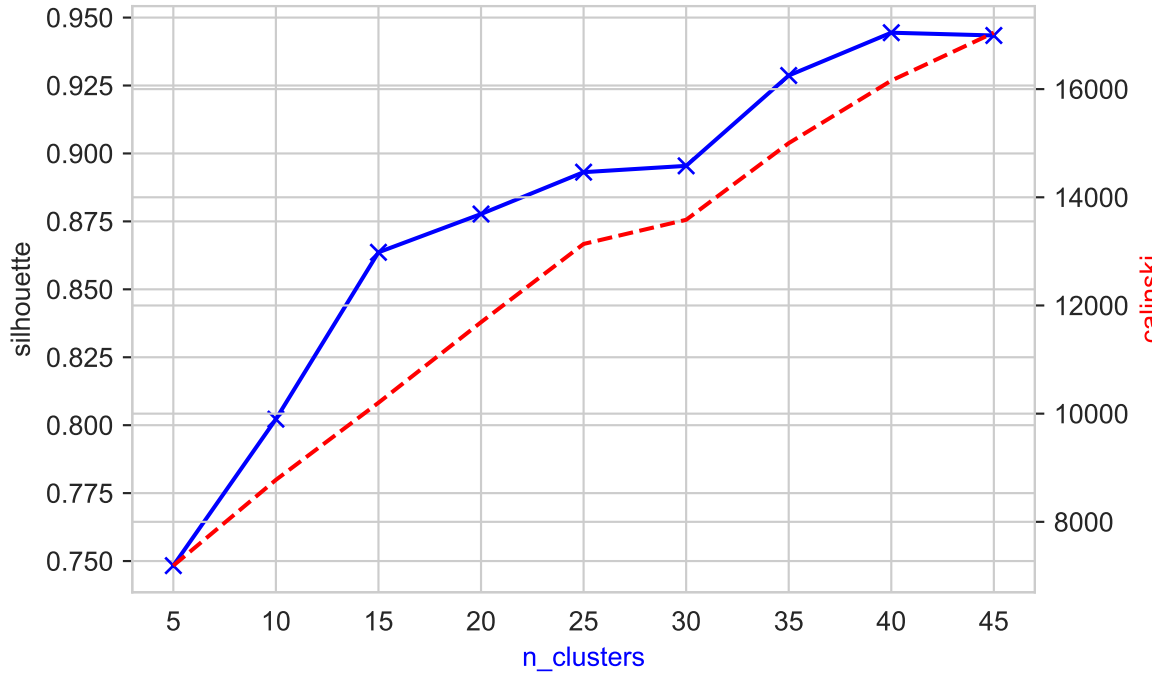


Figure 2.29: Comparación Silhouette y Calinski-Harabaz caso 4.

Birch

Como ya hemos visto en los comentarios generales, una vez más este algoritmo obtiene peores resultados en el coeficiente de **Calinski**. Por ello, nos planteamos si aumentando el número de clusters podría llegar a ser competitivo con el resto de algoritmos. En la figura [2.15] vemos distintos resultados para nuestro algoritmo Birch manteniendo el **threshold** a 0.01.

Algoritmo	Clusters	Calinski	Silhouette	time(s)
Birch-10	10	3544.429248	0.697959	0.585045
Birch-20	20	2687.237847	0.709917	0.737437
Birch-30	30	2070.812063	0.701047	0.425660
Birch-40	40	1781.324542	0.713947	1.512000
Birch-50	50	6477.617557	0.842701	1.136284
Birch-60	60	7370.865778	0.854710	0.543521

Table 2.15: Tabla comparativa clusters Birch caso 4.

En las primeras configuraciones el coeficiente de **Calinski** oscila aumentando o disminuyendo mientras que **Silhouette** solo aumenta. Esto se puede deber a la influencia negativa que tiene el número de clusters utilizado en el índice de Calinski-Harabaz, y es que los resultados no parecen mejorar en exceso según el coeficiente Silhouette (aunque hay un salto entre los clusters 40 y 50), pero sin embargo estamos aumentando en 10 el número de clusters considerados. También vemos que igualmente respecto a nuestra configuración inicia de 5 clusters, la mejora es clara en el índice de Calinski-Harabaz.
Analizamos tambíen la influencia del threshold para Birch-10, ya que ha conllevado mejoras con respecto a Birch-5 (implementado inicialmente) y sigue siendo interpretable. Recordamos que en apartado anterior, una disminución del valor de threshold nos proporcionaba peores resultados, veamos si aumentando el threshold en este caso conseguimos mejorar los resultados ya obtenidos. Vemos los resultados en la figura [2.16], los valores de threshold que elegimos son 0.01, 0.05 y 0.07.

Algoritmo	Clusters	Calinski	Silhouette	time(s)
Birch-01	10	3544.429248	0.697959	0.971139
Birch-05	10	2576.272947	0.654145	0.606884
Birch-07	10	1350.941220	0.599513	0.280413

Table 2.16: Tabla comparativa threshold Birch caso 3.

En comparación con nuestro resultado inicial, vemos una clara mejoría simplemente cambiando el threshold de 0.1 a 0.01.

Meanshift

El número de clusters utilizado en este algoritmo ha sido demasiado alto (80). En el resto de ocasiones, lo que hemos hecho ha sido aumentar el número de clusters en otros métodos para conseguir superar a este algoritmo, sin embargo, esto no es útil en problemas de agrupamiento pues necesitamos en la mayoría de los casos obtener conclusiones de los grupos formados.
Probamos distintas configuraciones del parámetro **bandwidth** (ancho de banda) en el algoritmo de MeanShift, para esto, vamos a variar el parámetro del **quantile** en el método que lo estima. Tomaremos valor de **quantile** = 0.2, 0.3, 0.4 y 0.5, obteniendo los resultados de la figura [2.17]

Algoritmo	Clusters	Calinski	Silhouette	time(s)
MeanShift-02	81	5904.067211	0.856255	13.723967
MeanShift-03	80	5958.743235	0.856150	19.589461
MeanShift-04	80	5958.743235	0.856150	31.827256
MeanShift-05	78	5926.095194	0.856006	25.699231

Table 2.17: Tabla comparativa quantile Meanshift caso 3.

El número de clusters necesitados no ha descendido, es más, ni siquiera hemos conseguido un método competitivo con el resto de algoritmos implementados hasta el momento pues ni el coeficiente **Silhouette** ni el índice de **Calinski-Harabaz** ha cambiado mucho. Lo que es más, el tiempo de ejecución ha aumentado bastante, por lo que no hemos conseguido ninguna configuración del MeanShift que merezca la pena ya que no hemos obtenido el equilibro entre buenos resultados e interpretabilidad.

AgglomerativeClustering

Como último algoritmo jerárquico, vamos a implementar una matriz de conectividad [Con].
Vamos a comparar la diferencia en cuanto a las medidas consideradas si introducimos este parámetro como si no lo introducimos para valores de clusters 5 y 10. Además, vamos a mantener el método considerado la configuración inicial, el Ward y estableceremos como número de vecinos a considerar 10 en ambos casos. Obtenemos la tabla [2.18]

Algoritmo	Clusters	Calinski	Silhouette	time(s)
Ward-10	10	7841.130154	0.807468	2.959856
Ward-10-con	10	7094.702364	0.802186	93.380960
Ward-20	20	10696.742580	0.877254	4.670297
Ward-20-con	20	10017.991953	0.872649	97.679698

Table 2.18: Tabla AgglomerativeClustering conectividad caso 4.

Vemos que no se han producido mejoras al utilizar esta matriz de conectividad. Además, el tiempo necesitado se ha disparado siendo de los algoritmos más lentos (sino el que más) que hemos utilizado a lo largo de esta práctica. Por tanto, podemos concluir categóricamente que mejor no utilizar este parámetro y dejar funcionar al método por si solo.
Además, comprobamos que las dos configuraciones implementadas utilizando más número de clusters mejoran bastante a la configurada anteriormente, por lo que, sí ha sido buena idea aumentar los clusters a considerar para la solución.

2.5.4 Interpretación de resultados

Para la interpretación de los resultados, vamos a utilizar solo algunos de los algoritmos implementados. Elegiremos aquellos que hayan funcionado bien además de aquellos que no hayan necesitado un alto número de clusters pues esto dificultaría la interpretación.

Primera interpretación

Para la primera interpretación elegimos, como siempre, el algoritmo **K-Means** con la configuración inicial de 5 clusters. Utilizaremos la Scatter Matrix de la figura [2.30],

al contrario que en los casos anteriores, los valores se encuentran más agrupados pero con un intervalo grande de valores (hasta 40 vehículos implicados en ciertos casos particulares), también comentar que hay cierta dependencia lineal entre tot_victimas y tot_heridos_leves.

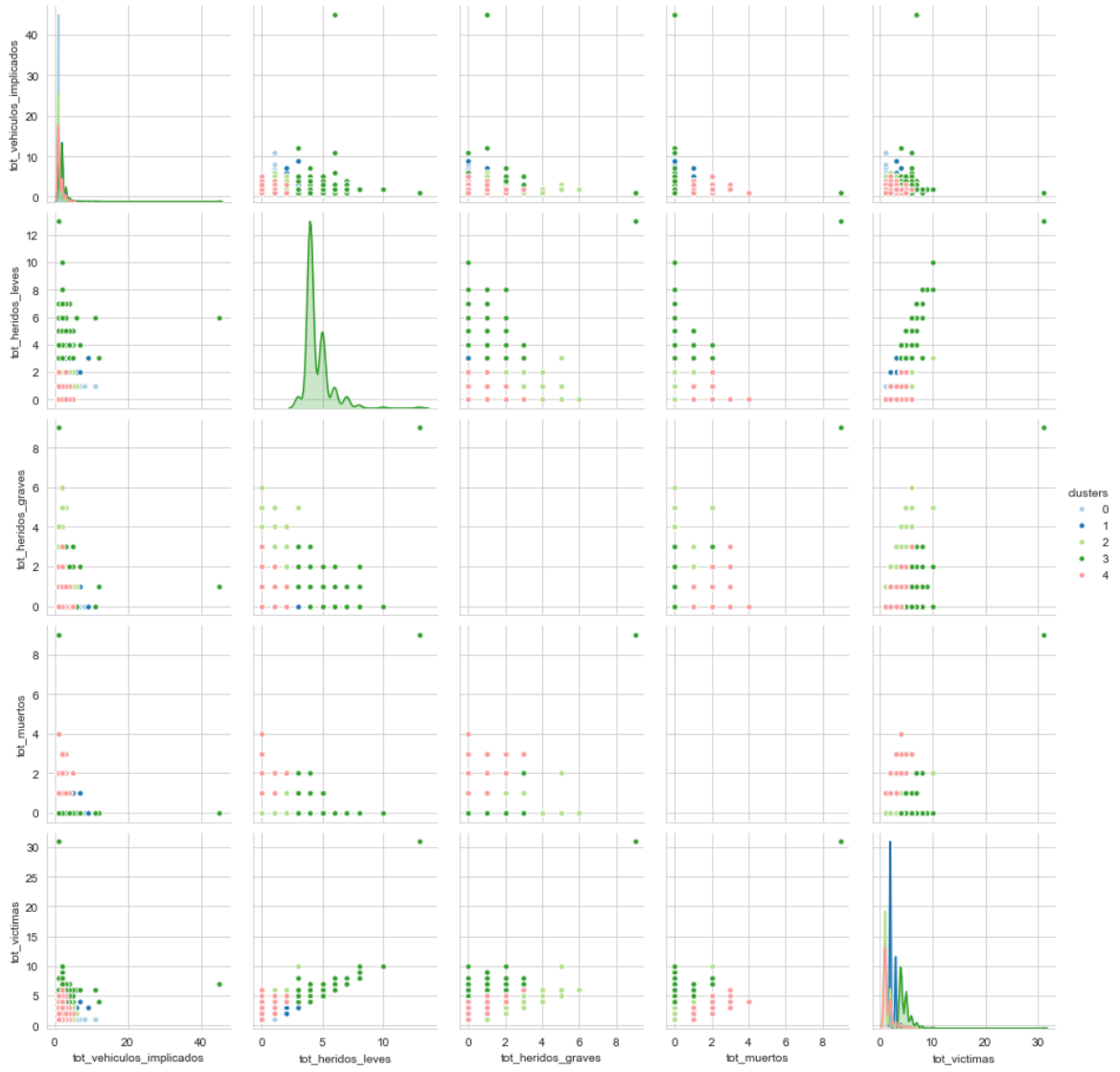


Figure 2.30: Pairplot K-Means 5 clusters caso 4.

Como en los casos anteriores, podemos caracterizar a cada uno de los clusters de la siguiente manera centrandonos en la primera columna y relacionandolo con vehículos implicados:

- **Cluster 0**: Concentra la mayor cantidad de vehículos implicados cuyos valores están entre 5 y 15, con un número mínimo de víctimas, con 1 fallecido. sin heridos graves y con 1 herido leve.
- **Cluster 1**: Como antes con números de vehículos implicados entre 9 y 10, esta vez con más víctimas, casi hasta 5, con 0 o 1 fallecido, sin heridos graves y

entre 2 y 3 heridos leves.

- **Cluster 2**: Menos de 5 vehículos implicados, con menos de 5 víctimas, sin fallecidos con bastantes heridos graves entre 4 y 6 pero con 1 o 2 heridos leves.

- **Cluster 3**: El cluster más interesante, agrupa la mayoría de heridos leves, y con la mayor variedad de vehículos implicados, desde 0 hasta 40, o 0 o 9 fallecidos y curiosamente solo 1 herido grave con 40 vehículos implicados pero hasta 9 heridos graves con menos de 2 vehículos implicados, y con la mayor cantidad de heridos leves desde 3 hasta 13.

- **Cluster 4**: Agrupa la mayoría de todos los menores valores de nuestros atributos estudiados, de 0 a 5 vehículos implicados, de 0 a 6 víctimas, de 1 a 4 fallecidos, de 0 a 3 heridos graves y de 0 a 2 heridos leves.

Vemos que en este caso, al contrario que en el caso anterior la cantidad de fallecidos y heridos graves es muchísimo menor, debido a que en esta ocasión no hay visibilidad restringida, por lo tanto vemos una clara relación en la importancia de visibilizar todo tipo de curvas. En la figura [2.31] de los centroides también destacamos el cluster 3, que agrupa gran cantidad de heridos leves, curiosamente hasta 40 vehículos implicados con únicamente 6 heridos leves y hasta 9 heridos graves con menos de 2 vehículos implicados, pudiendo ser un autobús o un vehículo grande que transporte a bastante cantidad de gente. De nuevo, parece que el factor de visibilidad en la curva es muy alentador en el tema de salvar vidas.

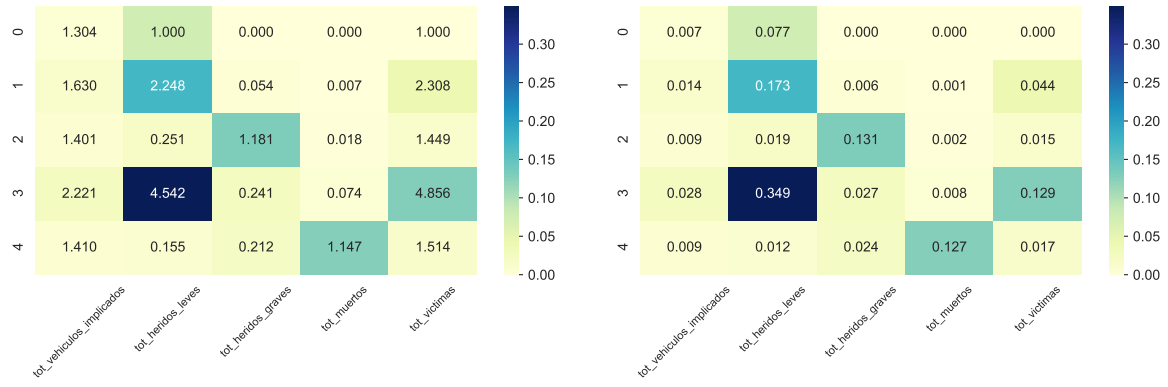


Figure 2.31: Centroides K-Means con datos sin normalizar y normalizados caso 4.

Segunda interpretación

Para esta interpretación vamos a utilizar el algoritmo jerárquico implementado en los 5 algoritmos iniciales, un AgglomerativeClustering Ward con 5 clusters. Visualizamos la Scatter Matrix en la figura [2.32]

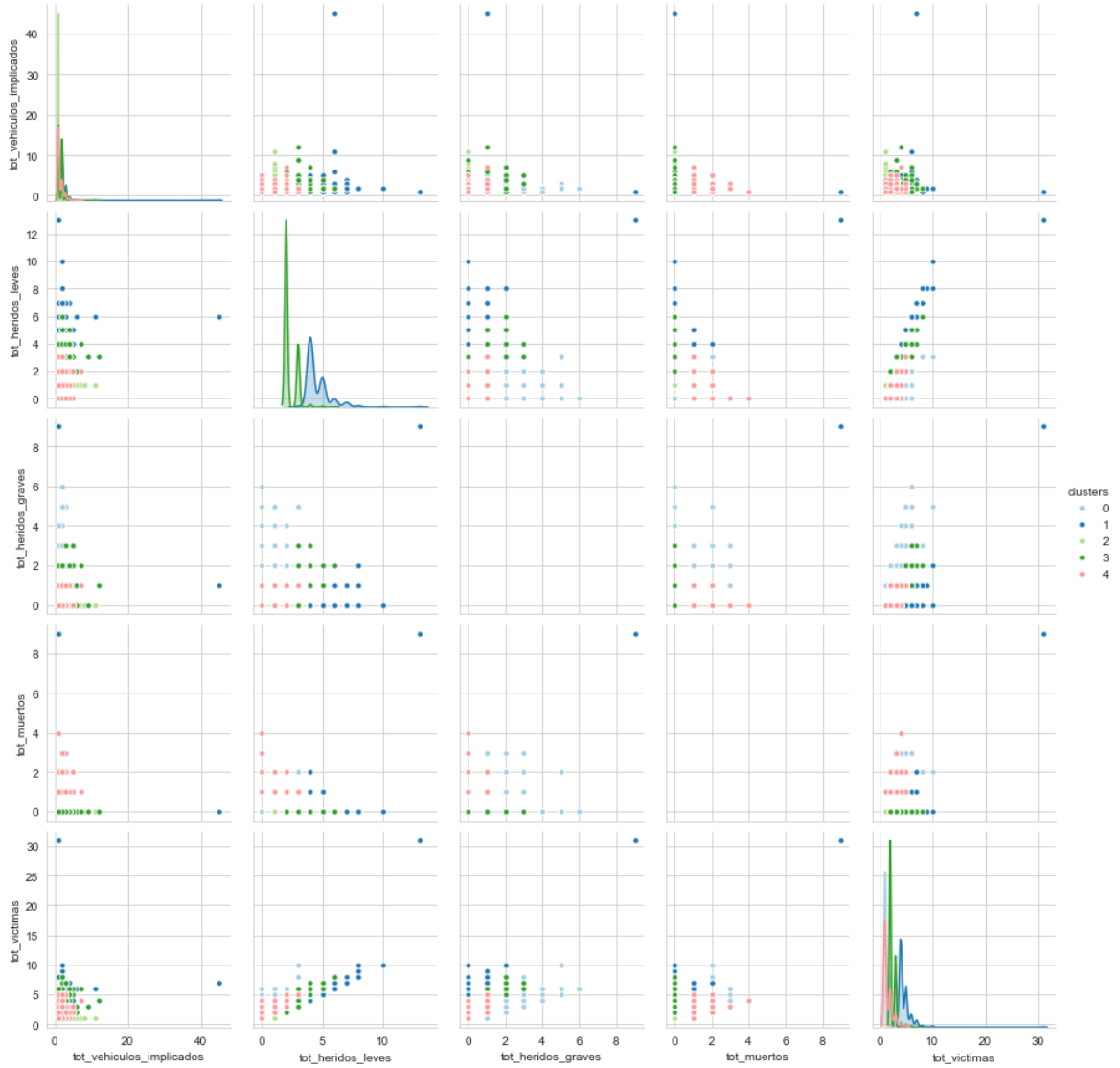


Figure 2.32: Pairplot AgglomerativeClustering 5 clusters caso 4.

Como hasta el momento, obtenedremos caracterizaciones de los diferentes clusters además de buscar relaciones entre las variables utilizadas. Se han generado 5 clusters, y las características generales de estos son:

- **Cluster 0**: El número de víctimas y vehículos implicados en este caso es mínimo, casi ni se aprecia en el Scatter Matrix, sin fallecidos y de 3 a 6 heridos graves sin heridos leves.
- **Cluster 1**: Contiene el número más grande de vehículos implicados, pero no el más concentrado, hasta 45 vehículos implicados, entre 5 y 10 víctimas, y sin término medio hablando de fallecidos o 0 o 9, con 1 herido grave o 9 y gran cantidad de heridos leves, como mínimo 6 hasta 13.
- **Cluster 2**: Entre 5 y 10 vehículos implicados, sin víctimas ni fallecidos, ni heridos graves pero siempre con algún herido leve, concentrando la mayor cantidad de vehículos implicados entre los comentados.

- **Cluster 3**: Entre 0 y 11 vehículos implicados, con gran variedad de víctimas concentrando la mayor cantidad de ellas entre 2 y 8, sin fallecidos y con gran variedad de heridos graves, de 0 a 5 y heridos leves, de 3 a 6.
- **Cluster 4**: En este cluster los valores se encuentran bastante concentrados, de 0 a 10 vehículos implicados, de 0 a 5 víctimas, de 1 a 4 fallecidos, de ninguno a 1 herido grave y de 0 a 3 heridos leves.

En cuanto a las conclusiones generales en función del tamaño y las dependencias entre variables, son las mismas que en la interpretación anterior. De igual manera vemos que el número de víctimas aumenta conforme aumentan los heridos leves, esto puede ser debido a que parte de las víctimas en estos accidentes no pertenecen a los vehículos o que pertecían a un vehículo bastante grande como hemos comentado en la primera interpretación, reforzando así el comentario. De nuevo nos apoyamos en la visualización de los centroides en la figura [2.33], destacamos que debido a que la cantidad de fallecidos y heridos leves es menor, los clusters ahora se concentran más en los otros atributos, viendo el centroide bastante grande de heridos leves.

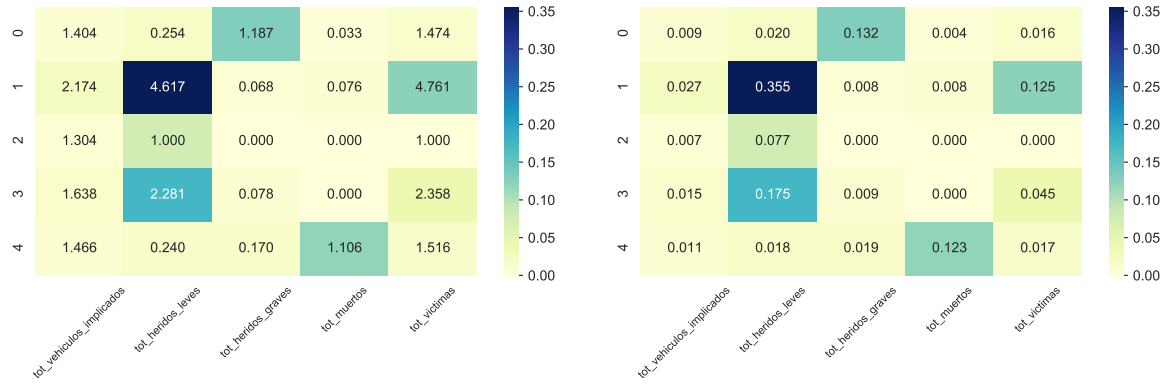


Figure 2.33: Centroides AgglomerativeClustering con datos sin normalizar y normalizados caso 4.

De nuevo vemos que no hay apenas variables respecto a nuestro atributo de fallecidos, esto puede ser de nuevo como hemos comentado en la primera interpretación debido a que en este caso no tenemos visibilidad restringida, y por ello repercute en nuestras variables a estudiar en el sentido lógico, ya que respecto al caso anterior, aunque respecto al caso anterior hay mas o menos los mismos vehículos implicados, la cantidad de heridos graves y fallecidos es muchísimo menor.

Referencias

[Alg] *Algoritmos de Clustering*. URL: https://scikit-learn.org/stable/modules/clustering.html.

[Con] *Connectivity matrix*. URL: http://scikit-learn.org/stable/auto_examples/cluster/plot_agglomerative_clustering.html.

[Sea] *Seaborn*. URL: http://seaborn.pydata.org/generated/seaborn.clustermap.html.