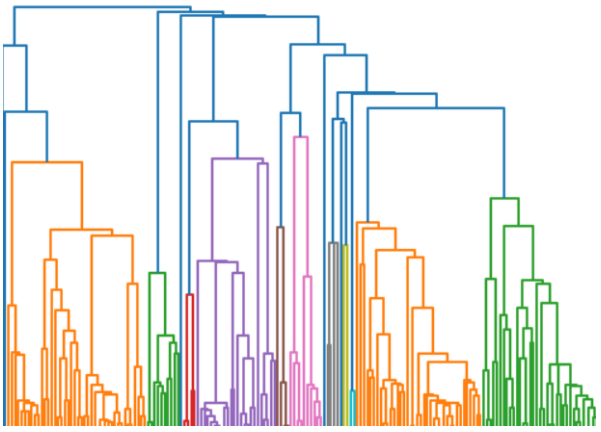
Proyecto Práctico

Aplicación de Técnicas de Preprocesado y Clustering



Autores:

- Mario García Berenguer
- Eder Tarifa Fernández

ÍNDICE

1. Presentación de los Datos

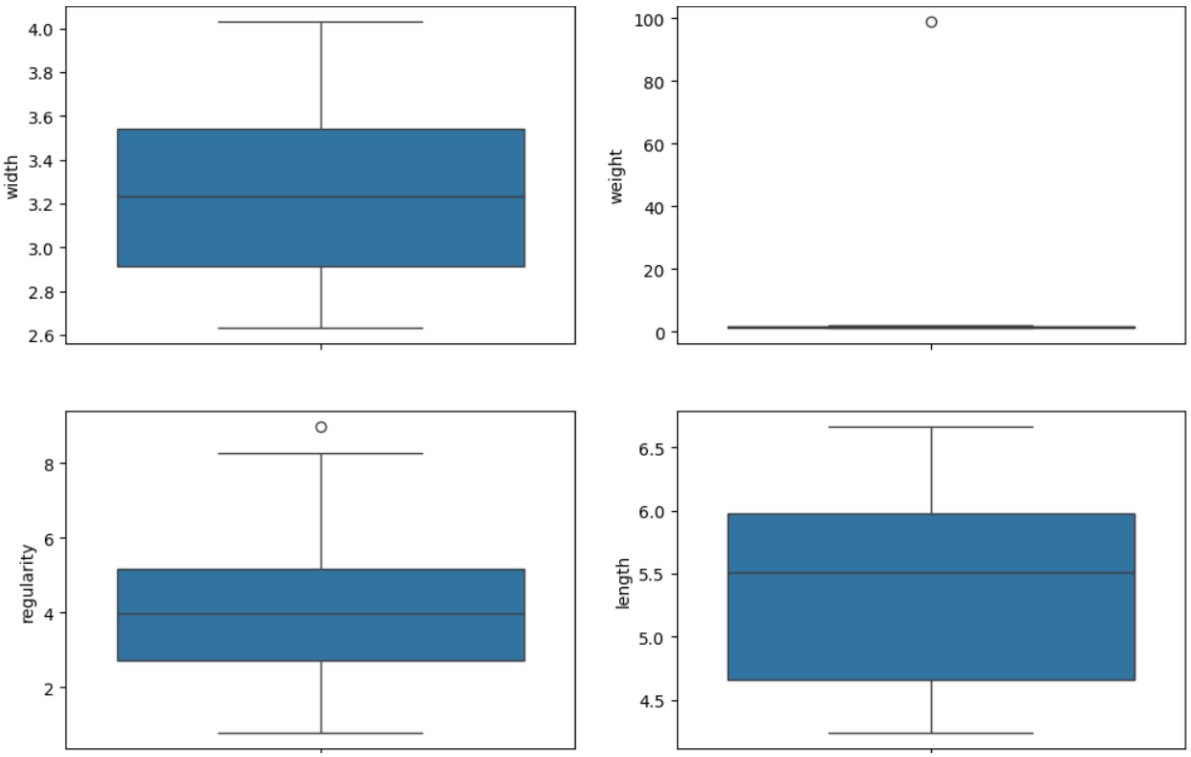
En este proyecto vamos a trabajar con un csv con datos de distintas instancias de frutas, aunque sin saber de qué tipo es cada una. Solo conocemos las siguientes variables:

- Weight: Esta variable representa el peso de la fruta en kg.
- Length: Esta variable representa lo que mide la fruta de largo en cm.
- Width: Esta variable representa el ancho de la fruta en cm.
- Regularity: Esta variable representa el grado de simetría de la fruta.
- Cleft: Esta variable representa el tamaño de la hendidura que tiene la fruta.

	weight	length	width	regularity	cleft
0	1.205	4.603915	2.847	5.691634	Small
1	1.726	5.978000	3.594	4.539000	Large
2	1.126	4.516534	2.710	5.965993	Average
3	1.755	5.791000	3.690	5.366000	Large
4	1.238	4.666888	2.989	6.153947	Small
...	...	...	...	...	...
175	1.345	NaN	3.065	3.531000	Very small
176	1.237	4.508595	2.960	4.655452	Small
177	1.437	5.569000	3.153	1.464000	Average
178	1.273	5.412000	2.882	3.533000	Very small
179	1.480	5.656000	3.288	3.112000	Average

Todas estas variables son variables numéricas y continuas, a excepción de cleft. Cleft se trata de una variable categórica, que puede tomar 5 valores distintos: “Very small”, “Small”, “Average”, “Large”, “Very large”. De la misma forma podemos suponer que se trata de una variable ordinal, ya que estos valores sí siguen en cierto modo un orden, un orden del tamaño de la hendidura.

Si realizamos un boxplot de las variables numéricas para analizar posibles datos atípicos, nos encontramos con que existe uno muy alejado del resto en la variable Weight y otro no tan alejado en la variable regularity.



También podemos ver que todas las variables a excepción de cleft toman valores Nan en alguna instancia.

```
Number of NAN in weight : 12
Number of NAN in length : 24
Number of NAN in width : 17
Number of NAN in regularity : 10
Number of NAN in cleft : 0
```

Viendo todo esto, podemos ya comenzar a pensar en la fase de preprocesado de datos. Sabemos que vamos a tener que hacer una imputación de valores para aquellas instancias que tengan un Nan en alguna variable, que vamos a tener que codificar la variable cleft para que tome valores numéricos, que vamos a tener que eliminar los datos más atípicos, que vamos a escalar todas las variables para dejarlas en una escala lo más parecida posible y así evitar futuros problemas por distancias entre variables.

Una vez planteado todo, pasamos al preprocesado.

2. Preprocesado

Comenzaremos con la imputación de los Nan. Además, como el dato atípico de la variable Weight es claramente un error, y no parece que sea algo con lo que podamos trabajar, también lo vamos a imputar. Para ello lo pasaremos a Nan.

2.1 Imputación de valores

Para esta tarea nos surgen dos principales herramientas, el SimpleImputer o el KNNImputer de la librería sklearn.impute.

Hemos realizado pruebas con ambos, teniendo en cuenta varios valores para K en el caso del KNNImputer, y este ha sido el resultado para una de las instancias, ya que no podemos mostrar todos los casos:

```
Simple Imputer:  5.3824124004494776
```

```
KNN Imputer with K =  3 :  5.315678295661525
KNN Imputer with K =  5 :  5.372606977396915
KNN Imputer with K =  7 :  5.385147840997796
KNN Imputer with K =  9 :  5.207621692075332
KNN Imputer with K =  11 :  5.181689660442592
KNN Imputer with K =  13 :  5.259352789605269
KNN Imputer with K =  15 :  5.216447020866943
```

Cuando hemos repasado varias instancias nos hemos dado cuenta de que el mejor método de imputación era el KNNImputer, ya que permitía fijarnos en más de una variable a la hora de imputar. Además, el valor que hemos decidido escoger para K es 5, ya que hemos visto que suele generar valores intermedios entre los máximos y los mínimos que generan el resto de valores de K, además de que nos parece un buen número de vecinos en el que fijarnos.

2.2 Codificación de la variable “Cleft”

De nuevo aquí se nos abrían otras dos opciones para utilizar. Podíamos elegir entre un OrdinalEncoder o un OneHotEncoder, de la librería sklearn.preprocessing. Vamos a estudiar los pros y los contras de cada método de codificación.

Pros de Ordinal Encoder	Contras de Ordinal Encoder	Pros de OneHotEncoder	Contras de OneHotEncoder
Aprovechamos el orden de los datos	No se pueden aplicar operadores suma, resta etc	Forma muy simple de codificar	No aprovechamos el orden de los datos
Mantenemos una sola variable			Creamos mínimo 4 variables más

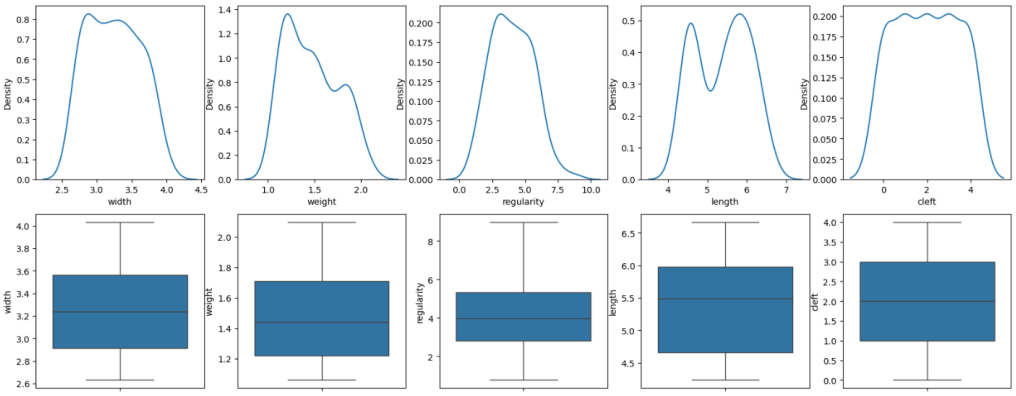
En conclusión, tenemos que:

1. Tenemos 5 posibles valores para la variable, lo que nos traduciría un OneHotEncoder en mínimo 4 columnas más. No obstante, un OrdinalEncoder nos mantendría una sola variable.
2. Los valores parece que siguen un orden, de menor tamaño a mayor tamaño, por lo que un OrdinalEncoder adquiere más sentido.

Una vez decidida la técnica pasamos a la codificación. Les vamos a dar valores del 0 al 4, siendo 0 “Very small” y 4 “Very large”. El resultado sería algo tal que así.

	cleft
0	1.0
1	3.0
2	2.0
3	3.0
4	1.0
...	...
175	0.0
176	1.0
177	2.0
178	0.0
179	2.0

Vemos que ya tenemos la variable cleft codificada, por lo que ahora vamos a ver todas las variables juntas a ver cómo se comportan.



Podemos apreciar que han desaparecido todos los atípicos de los boxplot, incluido el de la variable regularity. Parece ser que al imputar este ha dejado de ser atípico. También vemos que valores suele tomar cada variable, aunque los máximos y los mínimos los podemos ver mejor aquí, así como la diferencia.

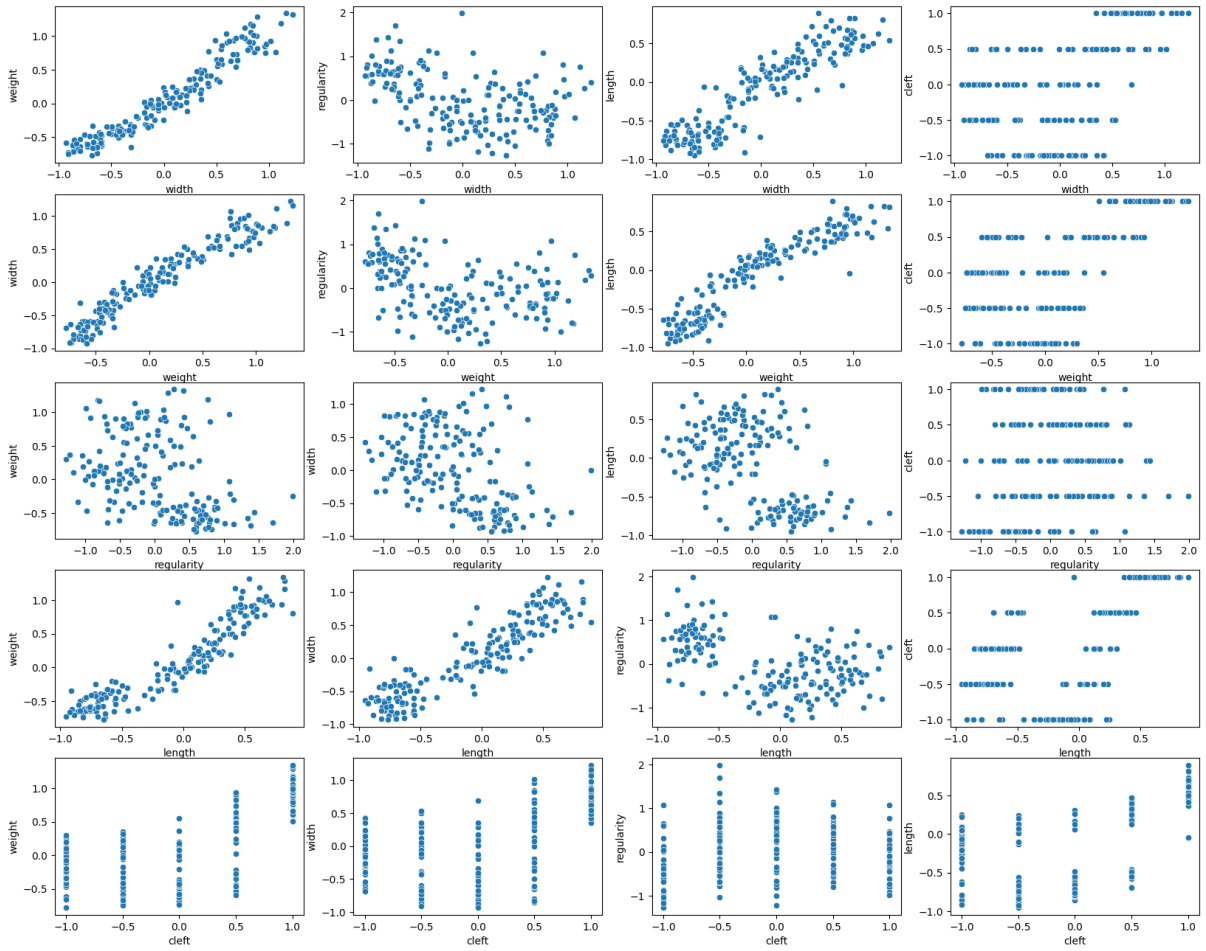
```
'weight': (1.059, 2.097, 1.038),
'length': (4.236811412670006, 6.666, 2.4291885873299943),
'width': (2.63, 4.032, 1.4020000000000001),
'regularity': (0.7651, 8.986146203041347, 8.221046203041347),
'cleft': (0.0, 4.0, 4.0)}
```

Vemos que aunque los rangos de las variables no se distancian mucho, lo más seguro es escalarlas para que los métodos de clustering puedan obtener mejores resultados.

2.3 Escalado de las Variables

Para asegurarnos de que nuestro escalado es correcto y es el más óptimo para realizar clustering, vamos a escalar los datos con distintas técnicas. Para ello vamos a usar las siguientes clases de la librería sklearn.preprocessing: StandarScaler, MinMaxScaler y RobustScaler.

Una vez realizados los tres escalados, guardamos todos estos modelos, incluido el sin escalar, para más adelante probarlos y decidir cual utilizaremos.
Si realizamos un scatterplot de los datos podemos obtener algo así.



Este es el gráfico del modelo en el que aplicamos RobustScalar.
Claramente vemos relaciones lineales entre muchas variables, como entre Weight y Width o Weight y Length. También vemos que la variable “Cleft” sigue tomando únicamente los 5 valores que podía tomar inicialmente, aunque ahora estén codificados numéricamente. Esto son cosas que tienen sentido y que hasta el momento nos podíamos imaginar, por lo que parece que vamos por buen camino y no hemos estropeado los datos, sino que los hemos mejorado y enriquecido.

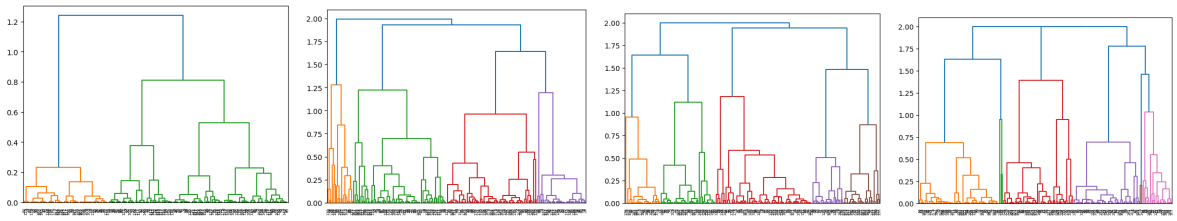
3. Clustering Jerárquico

Cuando nos planteamos por primera vez que método y que disimilaridad utilizar en nuestro clustering jerárquico, nos surgen muchas dudas y cuestiones que debemos probar. Es por ello que hemos decidido tener en cuenta todos los tipos de linkage, y así probar cuál es más fiable o cuál nos convence más. En el caso de las disimilaridades utilizadas, hemos elegido la distancia euclídea, la distancia Manhattan, la distancia de Chebyshev y la correlación. De la misma forma, hemos probado con entre 2 y 7 clusters, a ver qué modelo daba mejores resultados.
Para poder hacer todo esto hemos creado una serie de funciones auxiliares que nos ayuden a mostrar los dendrogramas de los clusters que generemos.

Vamos a dividir esta sección según características que hemos ido observando.

3.1 Respecto a los Datos Utilizados

Cuando nos referimos a los datos utilizados hablamos de los datos con un escalado específico, o sin escalar incluso. En esta sección queremos remarcar que los modelos por lo general ofrecen resultados parecidos. Si que es cierto que si tuviéramos que eliminar uno de ellos, este sería el modelo de los datos sin escalar, ya que en algunos casos se diferencia bastante de los otros 3 en el número de clusters que general. Estas diferencias se pueden ver por ejemplo si usamos como disimilaridad la correlación y como linkage el tipo complete.

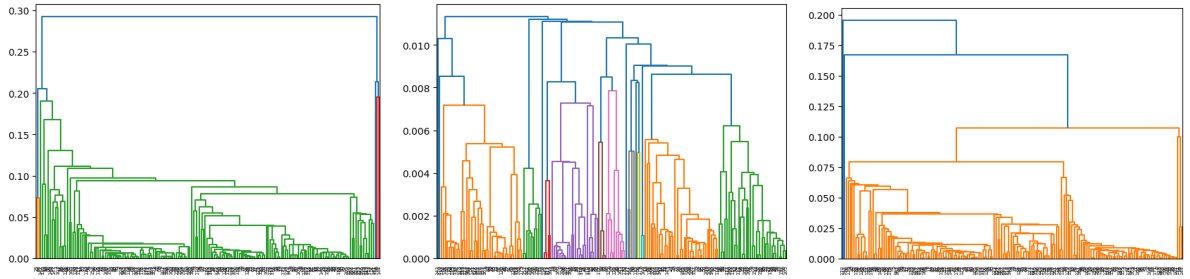


Vemos que el primero, que es el que usa datos sin escalar, solo genera 2 clusters. Sin embargo, los otros tres dataframes escalados generan 4 o 5 clusters, por lo que se ve una clara diferencia en el número. También se ve una clara diferencia en la jerarquía que estos siguen.

3.2 Respecto a los linkages utilizados

Si nos fijamos en los dendrogramas que generamos podemos extraer varias conclusiones.
En primer lugar, vemos que cuando usamos “single” como linkage, generamos dendrogramas muy descompensados y extremos. Esto quiere decir que, por un lado, el número de clusters generados suele ser o muy alto (6 o más) o muy bajo (2 o menos); y que, por otro lado, los clusters generados son muy irregulares y

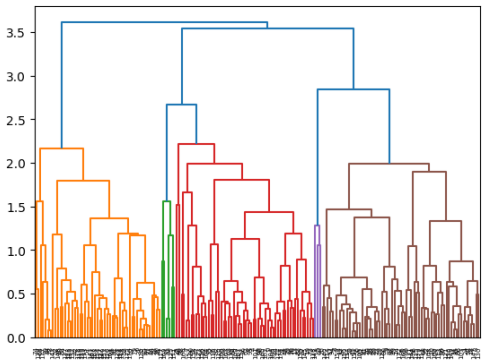
asimétricos, y tienden a tener tamaños muy variados y para nada compensados. Estos dos temas que acabamos de comentar se pueden apreciar en dendrogramas como los siguientes:



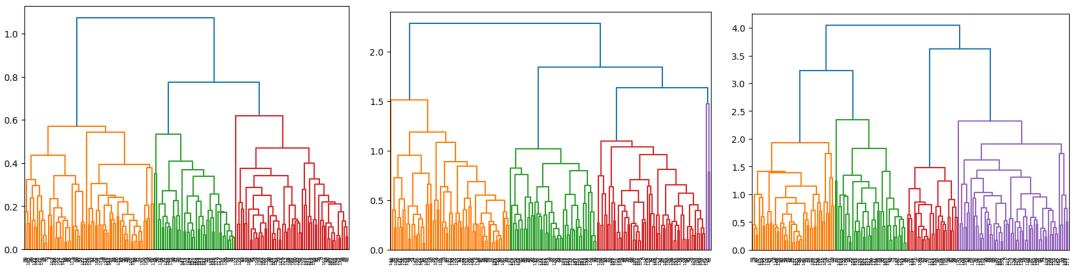
Esto tiene sentido con lo que hemos estudiado, ya que sabemos que el single linkage es muy extremo a la hora de balancear los clusters.

En segundo lugar, comentaremos los resultados obtenidos con el average linkage. Sabemos que el average linkage es sensible a las escalas. Esto es algo que podemos comprobar gracias a los dendrogramas, ya que cuando hacemos clustering con los datos sin escalar usando este linkage nos damos cuenta de que genera resultados con diferencias para los datos con escalado y sin escalado, sobre todo en la distancia euclídea:

- No escalado:



- Escalados:

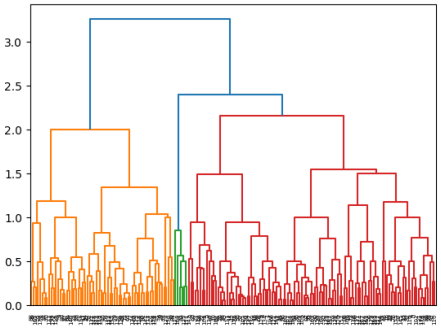


Estas diferencias no solo se dan en el número de clusters, sino que también lo vemos en la estructura de las jerarquías en los clusters. Cuando usamos datos escalados esta estructura es muy similar entre sí, y genera clusters de tamaño y forma similar también. Sin embargo, con los datos sin escalar vemos que tenemos dos clusters muy pequeños.
El ward linkage y el complete linkage parecen ser los que mejor resultados dan.

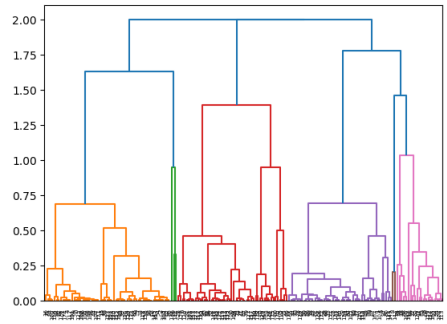
3.3 Respecto a las disimilaridades utilizadas

Con respecto a las métricas que utilizamos para crear los dendrogramas, vemos que las que mejores resultados dan son la distancia euclídea y la distancia Manhattan. La razón por la que llegamos a esta conclusión es porque estas dos suelen generar siempre modelos similares y más regulares, mientras que la distancia de Chebyshev y la correlación no nos proporcionan modelos tan parecidos. Estas diferencias se pueden ver fácilmente cuando usamos el linkage complete en el modelo robusto por ejemplo:

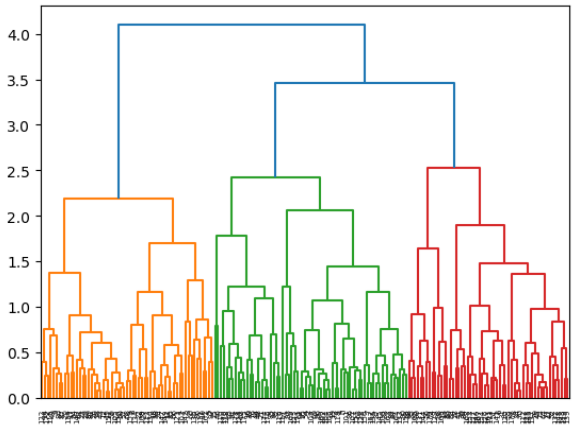
- Chebyshev: Genera 3 clusters bastante distintos en tamaño y estructura

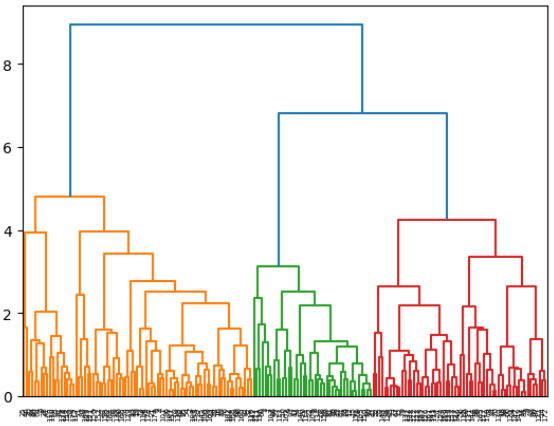


- Correlación: Genera 6 clusters muy distintos entre sí, claramente muy mal modelo



- Euclídea y Manhattan: Generan 3 clusters con tamaños parecidos y bastante similares en estructura.





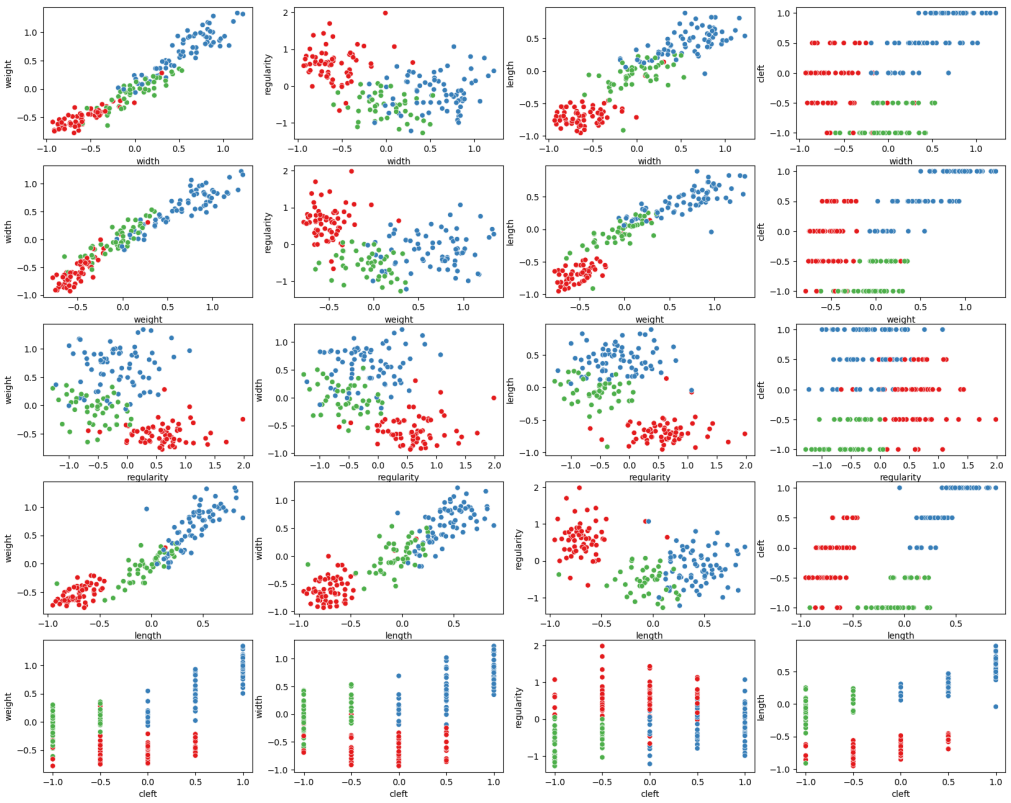
4. Clustering usando KMeans

A continuación, utilizaremos la función KMeans de sklearn.cluster para determinar qué número de clusters es el óptimo para nuestro conjunto de datos.

Para realizar esto hemos calculado los coeficientes de Silhouette y la inercia para los diferentes Ks de KMeans, fijándonos, utilizando las gráficas, en el coeficiente máximo y en el gráfico de la inercia en el k donde se crea una forma de codo. Además, hemos realizado este proceso para cada tipo de escalado de datos que hemos realizado. De esta forma, de forma indirecta estamos volviendo a comprobar que escalado obtiene el mejor resultado, como hemos hecho en el apartado anterior.

En los resultados, como era de esperar, obtenemos el mismo resultado que en el clustering jerárquico. El mejor coeficiente de Silhouette que obtenemos es un valor ligeramente inferior a 0,6 en K=3, y además, observamos que la forma de codo se crea también en este mismo K, para el RobustScalar.

Una vez concretado el número de clusters, aplicaremos la función mostrar_cluster que agrupa las variables de dos en dos y nos muestra las agrupaciones que realiza para cada una de ellas.



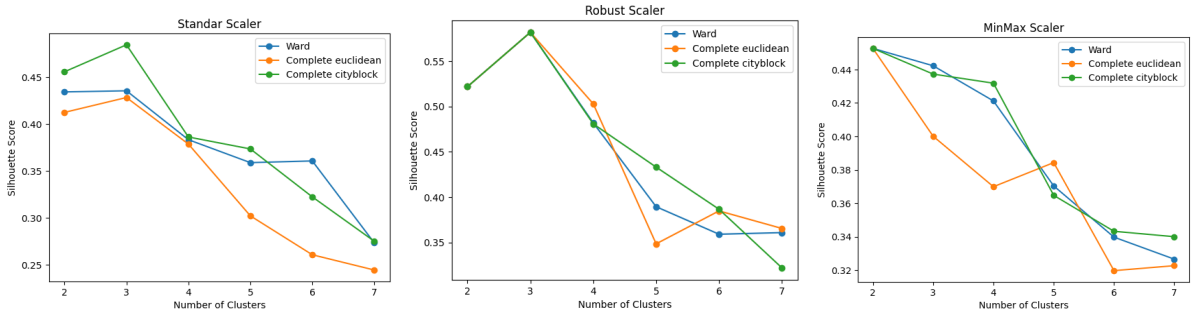
5. Calificación de los Clusterings

5.1 Calificación de Clustering Jerárquicos Aglomerativos

Para poder decidir cuál es el clustering jerárquico que mejor divide nuestros datos hemos decidido valorarlos mediante el coeficiente de Silhouette. No obstante no hemos probado todas las combinaciones, sino que nos hemos quedado con los que mejores dendrogramas nos daban. Estos son:

- Linkages: Ward y Complete
- Disimilaridades: Euclídea y Manhattan
- Datos escalados: Standar Scalar, MinMax Scalar, Robust Scalar

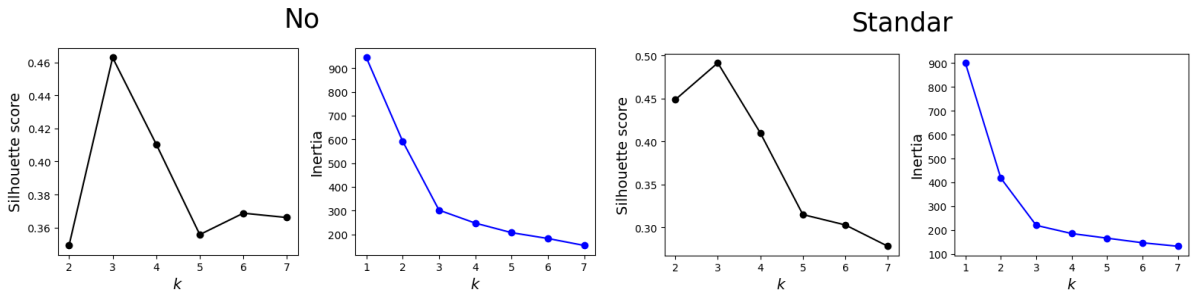
Los coeficientes de Silhouette que obtenemos son los siguientes:

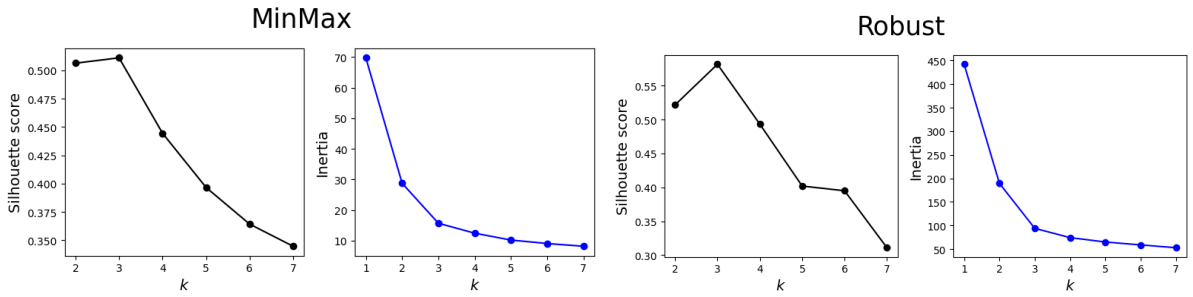


Vemos que el mejor modelo que obtenemos es el cluster en el que usamos el Robust Scaler con diferencia. Además, las combinaciones de linkages y disimilaridades usadas dan resultados muy parecidos, por lo que no sabríamos bien cual escoger al ser las 3 igual de buenas. No obstante, finalmente nos decantamos por el ward linkage, ya que nos parece la opción más balanceada.

5.2 Calificación de Clustering particional K means

Como previamente hemos comentado, para decidir el número K óptimo para K means, hemos utilizado el coeficiente de Silhouette y la inercia (fijándonos en la forma de codo). Estos han sido los resultados que hemos obtenido:





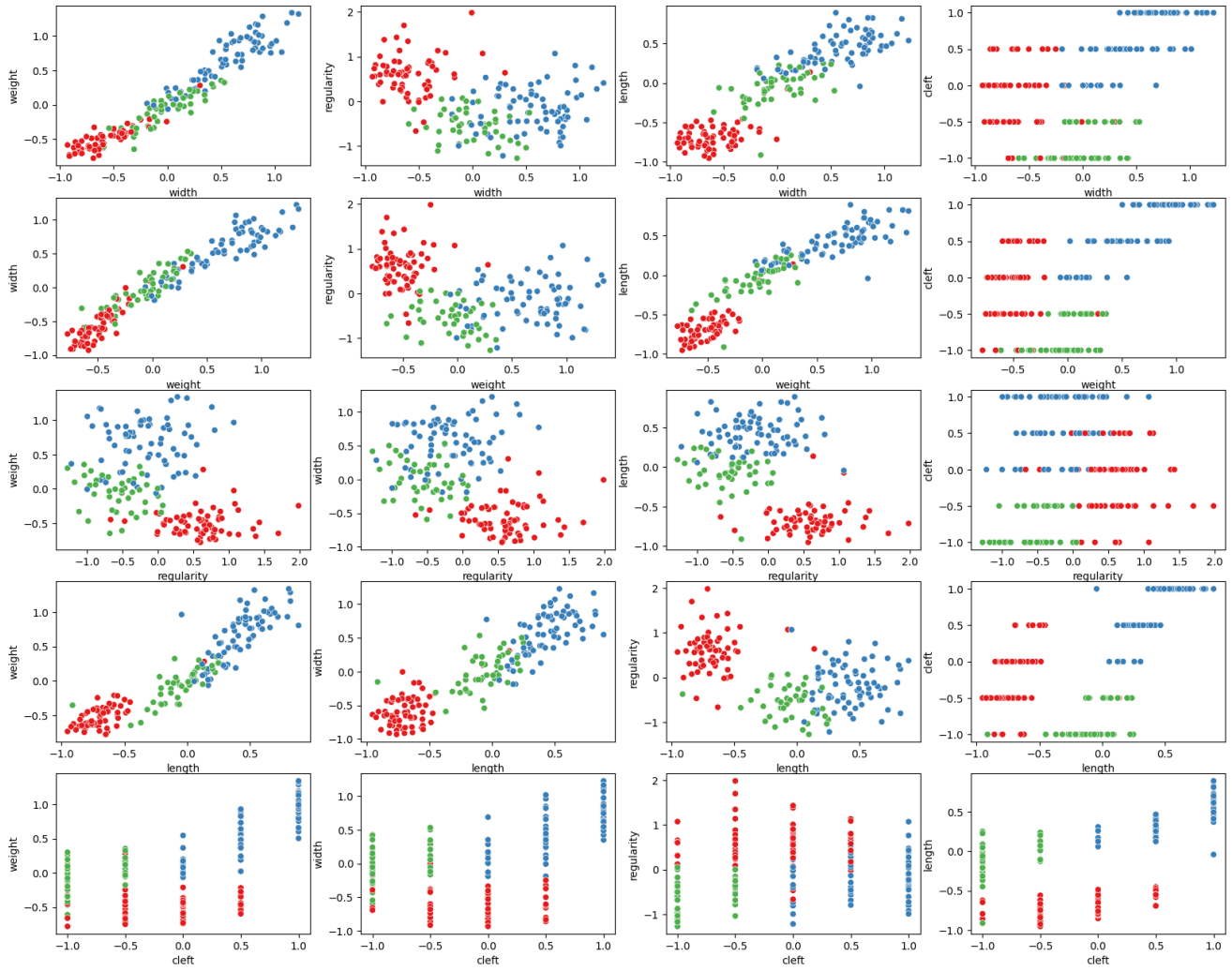
Podemos ver como en general todos los resultados, tanto el que está sin escalar y los escalados, tienen coeficientes de Silhouette bastante buenos, y además, conseguimos la forma de codo en el gráfico de la inercia en K = 3. Por tanto, podemos concluir que el K óptimo es 3 y, seleccionando el coeficiente más alto, el mejor escalado es el robusto.

6. Conclusiones

Una vez realizados tanto los clustering jerárquicos como los particionales utilizando KMeans hemos seleccionado el que nos daba como resultado el coeficiente de Silhouette más alto, que es el que utiliza KMeans con un escalado robusto con un coeficiente muy cercano a 0,6. Sin embargo, el clustering jerárquico aglomerativo también nos ha dado un buen coeficiente de Silhouette, ligeramente inferior al de KMeans, por lo que sabemos que este también es un buen modelo para nuestros datos.

A continuación, observamos el modelo creado con KMeans, separado en variables de dos en dos para poder graficarlas. Tenemos tres clusters diferentes, pero podemos ver como el azul es el que tiende a tener los valores más altos en todas las variables, a excepción de regularity. Como contraparte, el cluster rojo tiene todas las variables con valores bajos menos regularity que son más bien altos y cleft que es tanto alta como baja. Por último vemos que el cluster verde es el que tiene en todas las variables valores intermedios, menos en cleft, que es el cluster que solo toma valores bajos. Este gráfico nos ayuda a ver como ha interpretados nuestro

modelo qué variables son más determinantes para cada cluster y entre que rango de valores pueden estar.



Link para el vídeo de la presentación aquí
Link para el vídeo sobre nuestra opinión aquí