Universidad de Sevilla

Facultad de Matemáticas

Dto. Estadística e Investigación Operativa

Doble Grado en Matemáticas y Estadística

Análisis de conglomerados con series temporales

Trabajo Fin de Grado

Autora:

Mencía Veas Lerdo de Tejada

Tutor:

Antonio Beato Moreno

Junio 2023

A todas las personas que forman parte de mi vida.

A mi familia, especialmente.

Índice general

Resumen

El análisis de conglomerados con series temporales es una técnica estadística novedosa y aplicable a numerosos campos como las finanzas y las telecomunicaciones. Se trata de agrupar series temporales semejantes. Aquí radica el problema, pues es necesario definir una distancia entre series temporales que minimice la pérdida de información. Sin embargo, a priori es posible no saber qué metodología se adaptaría mejor a los datos, por ello es esencial hacer un estudio de diferentes distancias y aplicar diferentes métodos de análisis cluster para hacer una comparación de resultados.

Para desarrollar esta investigación se han de asentar las bases teóricas mínimas en las que se fundamenta. Por ello, se introducen conceptos de análisis de conglomerdos, de series temporales, así como de distancias y teoría de conjuntos.

A continuación, se propone una relación de distancias entre series temporales que tiene en cuenta diferentes aspectos de las mismas, de modo que ninguna de ellas aporta resultados exactamente iguales.

Por último, se muestra el código heurístico de cómo aplicar todo lo explicado sobre un conjunto de datos inventado en Python. Además, a título ilustrativo, también se incluye la aplicación real de este procedimiento con datos del Índice de Precios del Consumo (IPC) con el correspondiente análisis de los resultados.

Abstract

Cluster analysis with time series is a novel statistical technique applicable to many fields such as Finance and Telecommunications. It invloves grouping time series that are similar. Herein lies the problem: it is necessary to define a distance between time series that minimizes the loss of information. However, a priori it is possible not to know which methodology would best suit the data, so it is essential to study different distances and apply some different cluster analysis methods to make a comparison of results.

In order to develop this research, the minimum theoretical bases on which it is based must be established. Therefore, concepts of cluster analysis, time series, distances and set theory are introduced.

Then, a relation of distances between time series is proposed, taking into account different aspects of them, so that none of them gives exactly the same results.

Finally, the heuristic code of how to apply everything explained on an invented data set in Python is shown. In addition, for illustrative purposes, the actual application of this procedure with Consumer Price Index data is also included with the corresponding analysis of the results.

Índice de figuras

Capítulo 1

Introducción

El análisis de conglomerados con datos temporales engloba numerosos conceptos relativos a las series temporales y al análisis de conglomerados que han de ser aclarados antes de empezar a desarrollar toda la teoría en la que se centra el interés de este documento. Por ello, se dedica el capítulo introductorio a exponer los conceptos básicos en los que se basan las técnicas que se usan para hacer análisis cluster con series temporales.

1.1. La técnica del análisis por conglomerados

Dada una población de individuos $P = \{p_1, p_2, \ldots, p_n\}$, el análisis por conglomerados, o análisis cluster, es una técnica que trata de particionar esta población en grupos de observaciones que cumplen [10]:

- Existe una alta similaridad entre los individuos de un mismo grupo.
- Existe una diferencia significante entre observaciones que pertenecen a conglomerados diferentes.

Definición. Partición de un conjunto [14].

Dado un conjunto P, se dice que una familia $\{P_1, P_2, \ldots, P_s\}$, de subconjuntos de P, es una partición de P si se verifica que:

$$P_1 \cup P_2 \cup \cdots \cup P_s = P \text{ y } P_i \cap P_j = \emptyset \text{ para } i \neq j$$

Como a apriori no conocemos el verdadero grupo al que pertenece cada observación para evaluar la bondad de la clasificación que se realiza, se dice que el análisis cluster es una técnica de aprendizaje no supervisado [10]. Esto también implica que no existe una

solución única ni óptima, sino que depende del rendimiento que se desee conseguir. Es decir, la mejor solución es la que al investigador le resulte más interpretable y lógica en base al estudio que está realizando.

Por ejemplo, pensemos en el siguiente conjunto de datos: {rectángulo azul, rectángulo rojo, rectángulo verde, triángulo azul, triángulo amarillo, círculo amarillo, círculo rojo}. Se debe llegar a resultados diferentes si el interés se centra en agrupar observaciones por color, por forma, o por ambos. Incluso el número de conglomerados cambia en cada caso. Véanse las figuras (1.1), (1.2), (1.3), (1.4), (1.5).

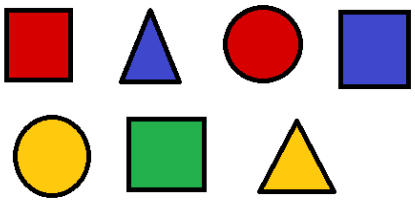


Figura 1.1: Ejemplo. Conjunto de figuras geométricas de distintos colores

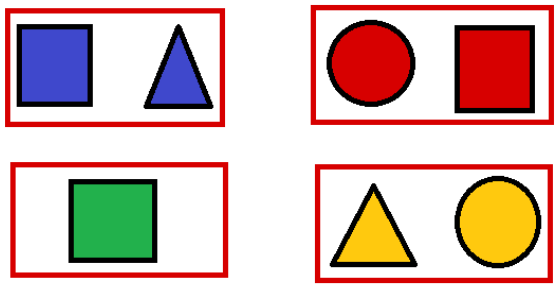


Figura 1.2: Agrupación por colores

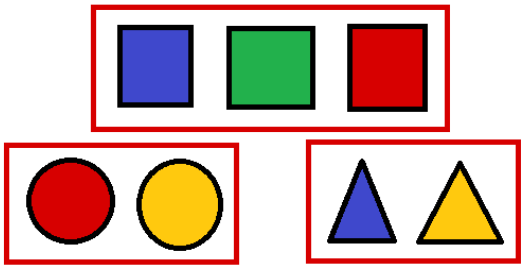


Figura 1.3: Agrupación por formas

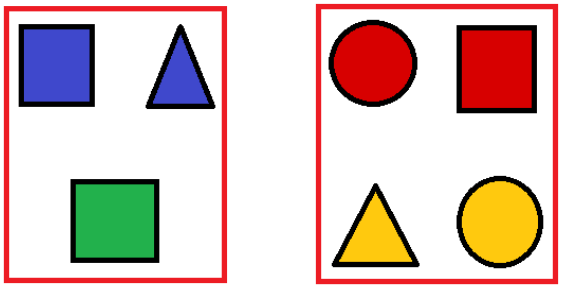


Figura 1.4: Agrupación por colores fríos/cálidos

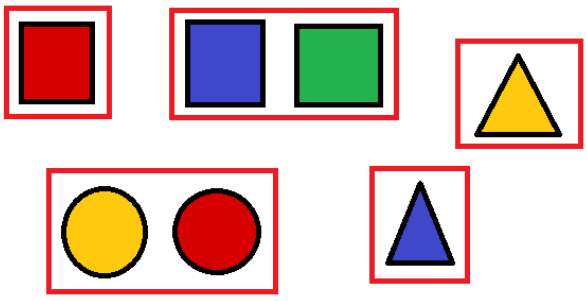


Figura 1.5: Agrupación por formas y color frío/cálido

Esto mismo ocurre con cualquier tipo de datos que se desea clasificar: datos numéricos, cualitativos, matrices, series, figuras, imágenes, etcétera; por lo que siempre es necesario definir claramente el motivo de la clasificación. Tener esto claro facilita la comprensión e interpretación de los resultados, incluso, y más aún, si son inesperados.

El número mínimo de conglomerados es uno, compuesto por todas las observaciones; y el número máximo es igual al número de observaciones, en cuyo caso cada conglomerado estaría formado por una única observación. Ninguno de estos casos extremos tiene especial interés porque no aportan información adicional.

El objetivo principal del análisis cluster es agrupar observaciones cercanas respecto de algún factor de interés, por ejemplo: separar observaciones de personas por sexo, por edad, por altura o por nivel de ingresos. Las formas de atacar este problema son muy variadas y dependen principalmente de la finalidad al segmentar la población, lo cual es fijado por dicho factor de interés.

Ahora bien, también es de gran importancia la función elegida para medir la distancia entre observaciones, pero esto está altamente ligado a la finalidad del estudio. Las distancias determinan en el algoritmo a qué grupo pertenece la observación, por lo que tiene sentido que en la definición de la distancia escogida vaya intrínseco el objetivo del análisis cluster. Es decir, continuando con el ejemplo anterior, si el objetivo del estudio es analizar la pigmentación de las figuras geométricas de la imagen 1.1, se ha de elegir una distancia que se base en cuantificar el pigmento de cada observación.

Definición. Distancia [15].

Dado un conjunto $X \neq \emptyset$, una distancia es una función $d : X\text{x}X \to \mathbb{R}$ que verifica:

1. $d(x, y) \geq 0 \ \forall x, y \in \mathbb{R}$ y $d(x, y) = 0 \Leftrightarrow x = y$,
2. $d(x, y) = d(y, x) \ \forall x, y \in \mathbb{R}$,
3. $d(x, z) \leq d(y, x) + d(y, z) \ \forall x, y, z \in \mathbb{R}$.

1.1.1. Clustering jerárquico aglomerativo

El clustering jerárquico aglomerativo es un proceso de análisis cluster que empieza con tantos clusters como observaciones. Es un proceso en varias etapas, y en cada una evalúa la distancia entre los datos y los clusters para incluir cada dato en el cluster más cercano. El proceso finaliza con un único cluster en el que se encuentran todas las observaciones.

Es un método muy visual, ya que todos los pasos quedan representados en el dendograma, por lo que es posible escoger visualmente el número k de conglomerados adecuado para el problema en cuestión, en base a las ramas y las distancias representadas en el dendograma.

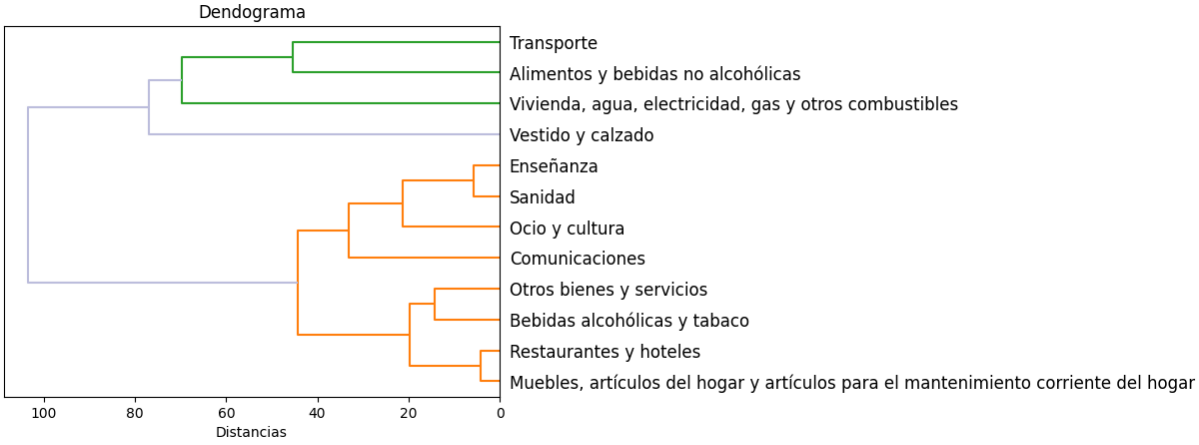


Figura 1.6: Ejemplo de dendograma

En este caso, la misma función implementada recomienda tomar tres conglomerados, pero eso depende de la finalidad del estudio.

1.1.2. Técnica Kmeans

La técnica K-means es un proceso iterativo al que hay que dar el número de conglomerados, k, como parámetro de entrada. Se toman tantas observaciones como conglomerados, a las que llamaremos centroides. Esta elección puede ser al azar o de forma intencionada, pero en cada paso se recalcularán. Incluso, es posible tomar como centroides datos no observados.

Una vez que se han calculado los centroides, se forman los conglomerados en función de las distancias entre las observaciones y los centroides. En cada iteración se recalculan los centroides como el elemento central del cluster y se reasignan las observaciones al centroide más cercano. El proceso finaliza cuando no se observa ningún cambio en los conglomerados entre dos pasos.

A diferencia que la técnica del clustering jerárquico aglomerativo, no existe ningún gráfico que recoja todo el proceso de reubicación de las observaciones hasta la posición final. En este caso, sólo se observan los conglomerados que se obtienen como resultado.

1.1.3. Selección del número de clusters

Fijar el número de conglomerados en que se dividirán las observaciones a priori no es un problema trivial, porque se pueden tomar desde un cluster hasta tantos como observaciones. Además, con el hándicap de que no se tienen datos iniciales del grupo verdadero al que pertenece el dato, como sí pasa en análisis supervisados. Por lo que, o bien se prueban todos los casos posibles y se escoge el mejor de entre todos los resultados, o bien se escoge un número al azar.

Sin embargo, existen medidas a través de las que es posible realizar este análisis antes de aplicar los métodos de análisis cluster. Por ejemplo, el Índice de Calinski-Harabasz.

Definición. Índice de Calinski-Harabasz [23].

Sea el conjunto de datos D, con n observaciones, que ha sido particionado en k conglomerados. Sea C_q el cluster q, c_q el centro de D, y n_q el cardinal del q-ésimo conglomerado. La puntuación Calinski-Harabasz, p, es el resultado de:

$$p = \frac{\mathrm{tr}(B_k)}{\mathrm{tr}(W_k)} \mathrm{x} \frac{n-k}{k-1}, \tag{1.1}$$

donde tr() es la función *traza* de una matriz, y

$$B_k = \sum_{q=1}^{k} \sum_{x \in C_q} (x - c_q)(x - c_q)' \tag{1.2}$$

$$W_k = \sum_{q=1}^{k} n_q (c_q - c_D)(c_q - c_D)' \tag{1.3}$$

1.2. Nociones sobre series temporales

Definición. Serie temporal [5].

Una serie temporal es una sucesión de observaciones correspondientes a una variable en distintos momentos de tiempo.

El enfoque en el que se basa el estudio clásico de las series temporales es el siguiente: el comportamiento de una variable a lo largo del tiempo es el resultado de la integración de cuatro componentes:

- La tendencia, que es la curva suave y regular que sigue la serie a largo plazo;

- El ciclo, que describe los movimientos recurrentes en torno a la tendencia que se repiten cada varios años;
- La componente estacional, que son los movimientos periódicos y regulares de la serie.
- La componente irregular, que no se puede predecir porque incluye todas las variaciones no explicadas por las componentes anteriores.

Siguiendo este enfoque, se puede expresar la serie temporal, X_t, como función de sus componentes, que varían con el tiempo:

$$X_t = f(T_t, E_t, I_t), \text{ donde}$$

- T_t es el valor de la tendencia en el instante t,
- E_t es el valor de la componente estacional en el instante t,
- I_t es el valor de la componente regular en el instante t.

Los modelos que se pueden definir para describir X_t son muy diversos, combinando operaciones entre componentes, por ejemplo:

- Modelo aditivo: $f(T_t, E_t, I_t) = T_t + E_t + I_t$
- Modelo multiplicativo: $f(T_t, E_t, I_t) = T_t$ x E_t x I_t
- $f(T_t, E_t, I_t) = T_t^2 + \frac{E_t}{I_t}$

Veamos la diferencia al emplear un modelo u otro, y la importacia de clasificar el modelo correctamente en las imágenes 1.7, 1.8, 1.9. La mayor diferencia se observa en la componente estacional y en los residuos.

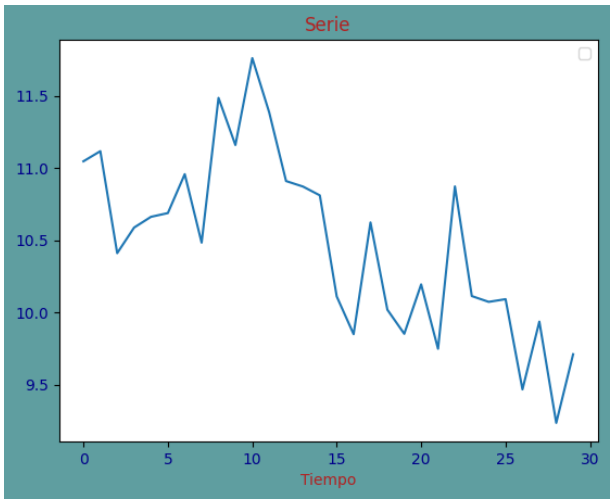


Figura 1.7: Ejemplo. Serie temporal

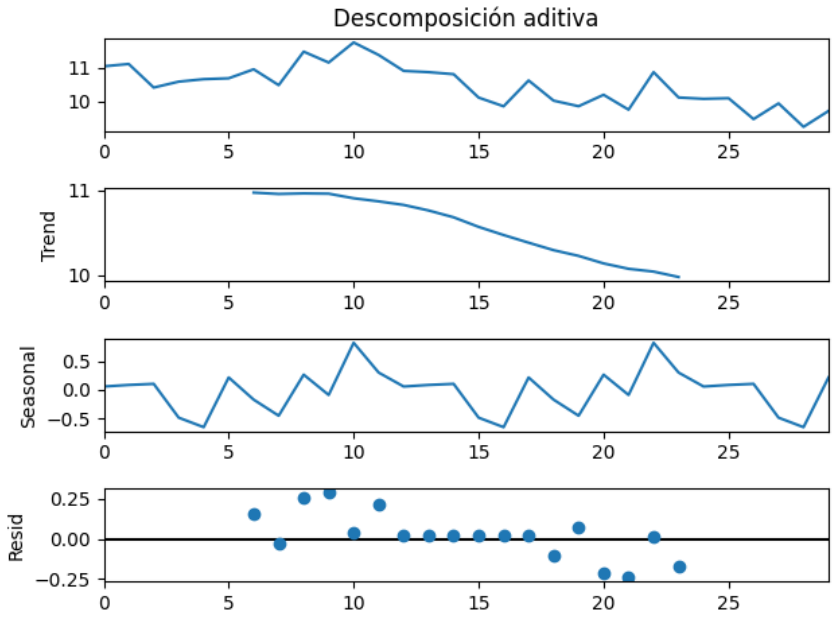


Figura 1.8: Ejemplo. Descomposición con modelo aditivo

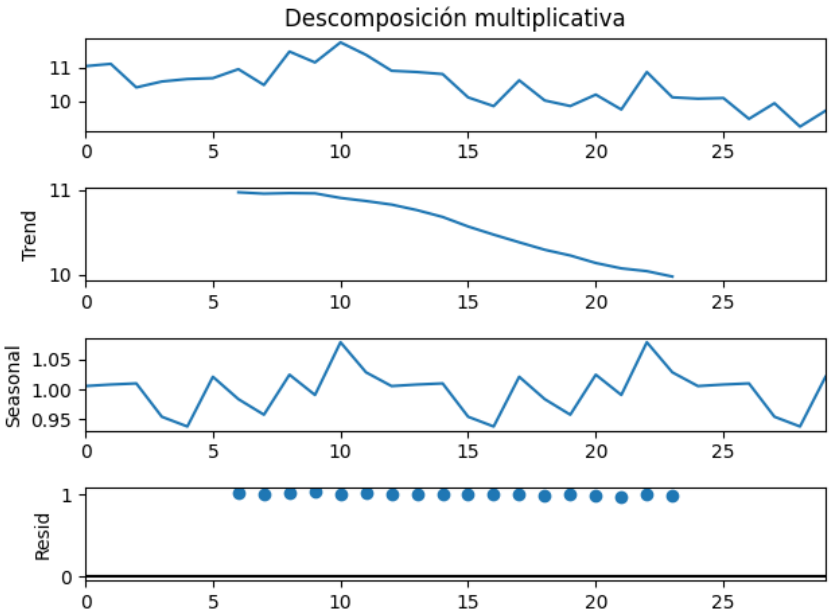


Figura 1.9: Ejemplo. Descomposición con modelo multiplicativo

Claro que para cada modelo la forma de predicción es diferente. Una forma de afrontar el problema de ajustar un modelo idóneo para predecir es la siguiente:

Supongamos que $X_t = f(T_t, E_t, I_t) = T_t + E_t + I_t$ y que hay L estaciones. Todo se basa en aceptar que $I_t \approx 0$, porque en ese caso $X_t \approx T_t + E_t$.

A continuación, se estima T_t mediante el cálculo de medias móviles.

Definición. Media móvil (MM).

Es una transformación lineal de un conjunto de datos $\{X_t\}_{t=1}^{n}$ en $\{Z_t\}_{t=q+1}^{n-s}$, donde

$$Z_t = \sum_{j=-q}^{s} a_j X_{t+j}, t = q+1, \ldots, n-s,$$

con s y q núemros no negativos verificando $q+1 \leq n+1$, y $\{a_j\}_{j=-q}^{s}$ constantes reales tales que

$$\sum_{j=-q}^{s} a_j = 1$$

Con esto, $\hat{T}_t = MM_t$, y $X_t \approx MM_t + E_t \Rightarrow \hat{E}_t \approx X_t - MM_t$.

Ahora, se ha de estimar E_t. Para ello, se calcula $\hat{s}_i, i = 1, \ldots, n$, que es la media de las estacionalidades obtenidas en la estación i.

Sea $\hat{e}_i = \hat{s}_i - \bar{s}, i = 1, \ldots, n$, el índice variacional estacional en cada instante, donde $\bar{s}$ es la media de los $\hat{s}_i$. Entonces, $\hat{e}_i$ es una estimación de E_i.

Con esto, es posible reestimar la tendencia como $T_t \approx X_t - \hat{e}_i$. Y eso no es más que la serie temporal inicial desestacionalizada. Si estos datos desestacionalizados tienen una varianza muy pequeña, el ajuste es muy bueno.

Los cálculos son así en base al modelo aditivo que se ha supuesto, pero habría que adaptarlo en cada modelo diferente, en función de cómo esté definido.

Hay otros métodos que emplean ideas parecidas a ésta, pero en este trabajo emplearemos los modelos ARIMA, que han sido definidos y explicados en la sección 2.4.2.

Analizar series temporales es muy útil a la hora entender su evolución en el tiempo e intentar hacer buenas predicciones más o menos cercano. De ahí que existan muchas técnicas de predicción y análisis de datos temporales, que intentan mejorarse unas a otras en función de los datos que se manejan.

Aplicar análisis cluster a las series temporales refuerza esta finalidad, ya que comparar el comportamiento de varias series puede dar pistas sobre la evolución las mismas. Incluso se podrían construir regiones de confianza de predicciones en base a las series temporales que se han agrupado en un mismo conglomerado.

Para profundizar más en series temporales, consultar las referencias fuente de la sección: [5], [7], [9], [22].

1.3. ¿Por qué hacer clustering con series temporales?

Clasificar observaciones es un proceso sobre el que se ha estudiado mucho en Estadística: árboles de decisión, Naïve Bayes, análisis de conglomerados, regresión logística, redes neuronales. Todos ellos son ejemplos de métodos de clasificación de datos. Ninguno es mejor que otro, porque cada caso es diferente. Pero todos ellos sirven para clasificar.

El hecho de que se desarrollen tantas técnicas con un mismo objetivo hace patente la gran utilidad que tiene clasificar nuevas observaciones: correos no deseados, movimientos fraudulentos en los bancos, piezas defectuosas, tipo de virus del que se ha contagiado una persona, niveles socio-económicos, marca de deportivas preferida. Lo cierto es que una buena clasificación hace que las predicciones estén bien fundamentadas. Incluso permite planear estrategias de mercado más acordes a la población. Por ejemplo, reconociendo los niveles socio-económicos que priman entre las personas que principalmente utilizan un producto. Son planteamientos muy lógicos: segmentar la población según características similares y en base a otras observaciones ya clasificadas.

Sin embargo, cuando los datos son series temporales, un objeto más complejo que sólo observaciones multidimensionales, no es tan clara la utilidad de segmentar tal población.

El sentido de agrupar series temporales se basa en responder a la siguiente pregunta: ¿Qué quiere decir que dos series temporales se parecen?

Porque una vez que tenemos una repuesta, existe un motivo para segmentar un conjunto de series. Bien sea porque tienen un perfil similar, bien porque se les pueden ajustar modelos semejantes, bien porque los modelos ajustados predicen de forma parecida, bien porque siguen los mismos ciclos, bien porque las funciones de autocorrelación son muy cercanas.

Ahora que sabemos que tiene sentido clasificar series de tiempo, ¿qué utilidad real y práctica tiene? Comparar la inflación en el tiempo entre comunidades autónomas, detectar crisis económicas en base series pasadas, estudiar patrones entre series, desarrollar técnicas de reconocimiento de voz. Todas las utilidades necesitan, además, un estudio exhaustivo para terminar de entender el porqué de los resultados.

Para entenderlo mejor, tomemos el ejemplo del reconocimiento de voz. Al dictar un texto al teléfono móvil, se espera que el aparato reconozca las ondas de sonido que produce la persona y las procese adecuadamente. Pero esto no sólo debe hacerlo con una persona, sino con todas las personas que le puedan dictar. De modo que si dos personas, que tienen tonalidades de voz y formas de hablar diferentes, dictan la misma frase, el móvil debe clasificar ambas ondas como iguales, aunque no lo sean, porque el mensaje es el mismo.

1.4. Identificación del problema

En el análisis de conglomerados es una técnica que se puede utilizar tanto con datos univariantes como con datos multivariantes. Es posible tratar los datos espacio-temporales como datos multivariantes, es decir, como puntos del espacio. Sin embargo, mucha información importante se perdería, principalmente las relaciones entre observaciones cercanas en el tiempo. Aquí reside el mayor problema al que se ha de hacer frente a la hora aplicar el análisis cluster: las coordenadas de los datos espacio-temporales están relacionadas y el orden importa. Por tanto, conviene no prescindir de este detalle porque, de otra forma, despojaríamos a las series temporales de su característica principal: la correlación entre observaciones cercanas en el tiempo.

Las observaciones de una serie de tiempo no son independientes unas de otras, sino que influye lo que ha ocurrido en el pasado.

En las series temporales tiene sentido pensar que dos observaciones cercanas en el tiempo están más correlacionadas que otras más alejadas. Igualmente ocurre en el análisis de datos espaciales: es lógico que observaciones que han sido tomadas de lugares cercanos

estén más correlacionadas que si fueran de lugares alejados.

En particular, es importante resaltar que las series temporales no pueden tener observaciones en dos grupos diferentes. Cada serie es un objeto indivisible, no un conglomerado inicial, y hay que tratarla como una observación *per se*, no como subconjunto de observaciones. Es decir, hay que hacer un análisis de conglomerados con datos multivariantes de forma que intervenga la correlación de las observaciones.

En las series temporales, además, es necesario tener en cuenta que, aunque las tratemos como una serie de observaciones, el orden de las coordenadas es el que es y no otro. Por tanto, las series no sólo son indivisibles sino invariantes, aunque pueden verse ampliadas con nuevas observaciones.

En definitiva, como se mencionó en la sección 1.1, el foco de atención es: qué finalidad tiene el estudio. Porque ello determinará el factor de interés respecto del que homogeneizar las series y las distancias más adecuadas para llevar esto a término.

En el siguiente capítulo se desarrollarán varias formas de medir la diferencia o la similitud entre series. Veremos que depende de cada caso en concreto: las series con las que se trabaja, el factor de interés y la finalidad del estudio.

Capítulo 2

Distancias entre series temporales

2.1. Preámbulo

En esencia, el análisis de conglomerados con series temporales sigue siendo el mismo problema que el descrito en la sección 1.1: cómo agrupar datos de este tipo de forma que los grupos finales tenga una varianza pequeña internamente y sean significativamente distintos entre ellos.

Por tanto, queda por aclarar cómo adecuar el tratamiento de este tipo de datos convenientemente para desarrollar el análisis cluster.

En este caso también es primordial el factor de interés respecto del que se define la homogeneidad y la distancia idónea entre las series de tiempo y espacio. Por ejemplo, los resultados de un análisis de conglomerados aplicado al conjunto de series temporales del Índice de Actividad del Sector Servicios (IASS) diferirán si se hace, por ejemplo, un estudio para comparar los resultados de las comunidades autónomas o para hacer una comparación entre periodos. Así mismo, influirá la función distancia entre series.

Es evidente que tener claro el motivo del análisis también permite delimitar un conjunto de distancias apropiadas para el caso, y que las distacias entre conglomerados se pueden definir a partir de las distancias entre individuos.

Todas las distancias serán utilizadas para implementar las técnicas de agrupación que trataremos a lo largo de este trabajo: clustering jerárquico aglomerativo y la técnica K-means.

Al algoritmo de clustering jerárquico únicamente necesita una matriz de distancias como parámetro de entrada para obtener el dendograma y ajustar el modelo en base a dicha matriz y al número de conglomerados apropiados, según lo observado en el dendograma.

La técnica K-means, por el contrario, necesita el número de conglomerados deseados como parámetro de entrada y el conjunto de series que queramos comparar: el conjunto de datos inicial o el conjunto de distancias, para que el algoritmos tenga en cuenta la distancia escogida.

El mayor interés del estudio es obtener una clasificación no evidente a simple vista, o confirmar el resultado esperado. Por lo general no será posible dar una respuesta trivial, véanse, por ejemplo, las figuras 2.1, 2.2 y 2.3. En cada caso concreto será necesario identificar el mejor método para medir distancias porque el análisis visual de los datos es insuficiente e inexacto.

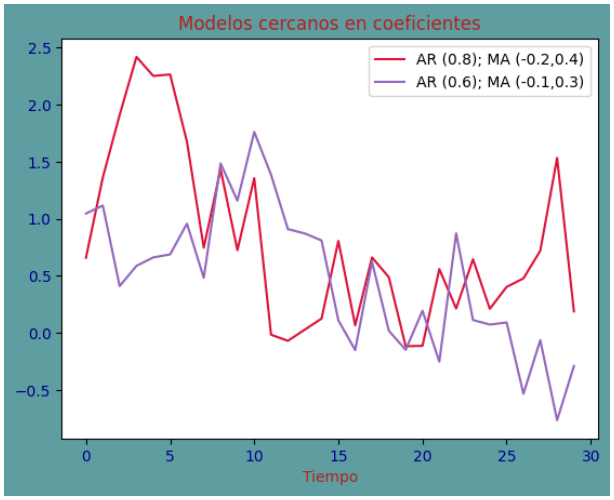


Figura 2.1: Modelos similares en coeficientes

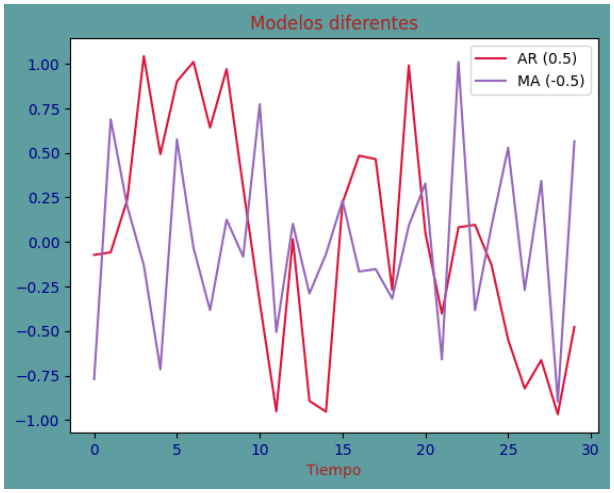


Figura 2.2: Modelos diferentes

Conviene, por tanto, mostrar una clasificación de factores clave en los que basar el análisis que darán pie definir distancias o similaridades entre las series. De forma que los resultado no se basen en la apariencia, sino en los datos observados. El análisis de conglomerados permite descubrir qué hay detrás de la apariencia.

La clasificación de distancias que se empleará en este trabajo es la que se introduce en el capítulo 12 de [13]. Por lo que, para un estudio más detallado se recomienda su lectura.

2.2. Clustering basado en las observaciones

El clustering basado en las observaciones es una forma de proceder directa, que se basa en la comparación de las series observadas o transformaciones de las mismas. Este método es útil sobre todo cuando las series no son muy largas. Es decir, cuando no hay mucha información recavada.

¿Por qué no es bueno agrupar de esta forma cuando las series tienen muchas observacio-

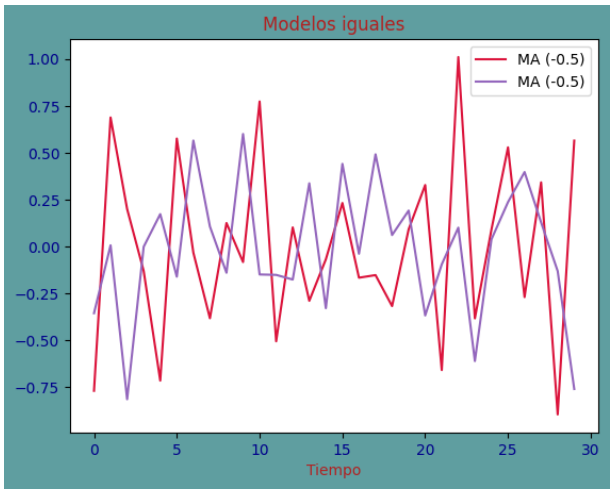


Figura 2.3: Modelos iguales

nes? Porque en ese caso, debe existir una estructura de autocorrelaciones que caracterice a la serie, cosa que no es posible captar al comparar observaciones. En cambio, cuando las series son cortas, no merece la pena comparar la estructura de autocorrelaciones porque las estimaciones de las autocorrelaciones serán poco fiables. Este hecho se desarrolla en la sección 2.3.

En este tipo de análisis se le da gran importancia al dato recogido en cada momento *ipso facto*. Por ejemplo, si la investigación es un estudio médico sobre el efecto de una droga en pacientes que padezcan una enfermedad o no, se tomarán medidas de los pacientes en varios intervalos temporales, y comparar los valores obtenidos tendrán una gran importancia. Es cierto que también se podrían hacer estudios evolutivos del efecto de la droga. Pero, por eso, es necesario concretar la finalidad del análisis.

Algunas distancias acordes a este planteamiento del problema son:

- Distancia euclídea.
  - Comparar las observaciones que se encuentran en la misma posición. (Ver apartado 2.2.1)
  - Comparar las velocidades. (Ver apartado 2.2.2)
  - Comparar las aceleraciones. (Ver apartado 2.2.3)
- Comparar el área entre pares de series. (Ver apartado 2.2.4)
- Distancia Dynamic Time Warping (DTW). (Ver apartado 2.2.5)

Las tres primeras opciones son distancias muy similares porque todas capturan el mismo tipo de información. La distancia DTW, en cambio, es más elaborada pero también más flexible.

Sean $x = (x_1, x_2, \ldots, x_m)$ e $y = (y_1, y_2, \ldots, y_n)$ dos series temporales con m y n observaciones, respectivamente.

Si $m = n$, es posible implementar la distancia euclídea y calcular el área entre dos series.

2.2.1. Distancia entre observaciones

La distancia entre dos series temporales basada en sus observaciones se define como [13]:

$$d_{\text{obs}}(x, y) = \sqrt{\sum_{t=1}^{m} (x_t - y_t)^2 u_t} \tag{2.1}$$

donde u_t es un peso adecuado para las observaciones del instante t.

Los pesos son una forma de ponderar la importancia de las observaciones. Al utilizar la distancia euclídea, es esencial utilizarlos porque esta es la forma de hacer intervenir, en cierto sentido, a la estructura de autocorrelaciones. De otro modo, no existiría diferecia entre x como serie de tiempo y x como un punto de un espacio m-dimensional.

Los pesos deben cumplir:

1. $u_t \geq 0, \forall 1 \leq t \leq m$;
2. $\sum_{t=1}^{m} u_t = 1$

Sin embargo, cada observación no deja de ser considerada independiente de las anteriores, lo cual desprovee a las series temporales de su principal característica. Lo único que se tiene en cuenta es el instante de la observación, pero es invariante frente a permutaciones temporales.

Si bien es cierto que hay otro inconveniente importante: la distancia es invariante frente a las permutaciones en el orden de las observaciones. Aunque se puede evitar esta traba si se escogen adecuadamente los pesos asociados a cada instante.

Con las distancias que se proponen a continuación, se pretende sortear estas limitaciones.

2.2.2. Distancia entre velocidades

La definición de esta distancia se apoya en la siguiente idea de velocidad [13]:

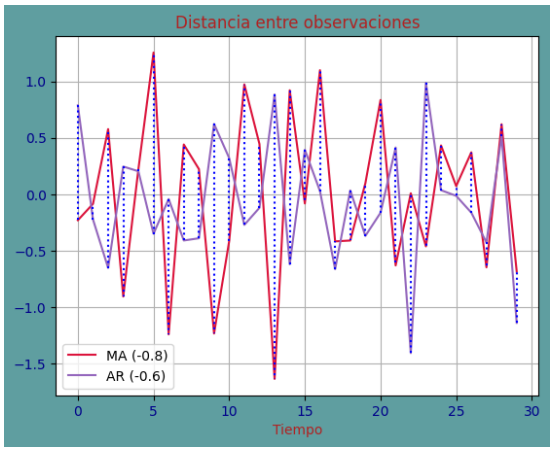


Figura 2.4: Distancia entre observaciones

Sean $v_t^x = x_t - x_{t-1}$ y $v_t^y = y_t - y_{t-1}$ las velocidades de las series x e y, respectivamente, en el instante t.

La distancia entre dos series temporales basada en sus velocidades se define como [13]:

$$d_{\text{vel}}(x,y) = \sqrt{\sum_{t=2}^{m}(v_t^x - v_t^y)^2 w_t} \tag{2.2}$$

desarrollando con la definición de velocidad, se tiene:

$$\begin{aligned}
d_{\text{vel}}(x,y) &= \sqrt{\sum_{t=2}^{m}(v_t^x - v_t^y)^2 w_t} = \\
&= \sqrt{\sum_{t=2}^{m}((x_t - x_{t-1}) - (y_t - y_{t-1}))^2 w_t} = \\
&= \sqrt{\sum_{t=2}^{m}((x_t - y_t) - (x_{t-1} - y_{t-1}))^2 w_t} = \\
&= \sqrt{\sum_{t=2}^{m}(x_t - y_t)^2 w_t - 2\sum_{t=2}^{m}(x_t - y_t)(x_{t-1} - y_{t-1})w_t + \sum_{t=2}^{m}(x_{t-1} - y_{t-1})^2 w_t} = \\
&= \sqrt{d_{obs}(x^*, y^*)^2 - 2\sum_{t=2}^{m}(x_t - y_t)(x_{t-1} - y_{t-1})w_t + d_{obs}(x,y)^2}
\end{aligned} \tag{2.3}$$

donde w_t es un peso adecuado para las velocidades del intervalo $[t-1, t]$, y x^*, y^* son las series originales sin la primera observación.

Esta distancia consigue comparar los perfiles de las series tramo por tramo. Y, tomando unos pesos apropiados, esta distancia plasma de forma más certera la estructura de dependencia temporal entre observaciones consecutivas.

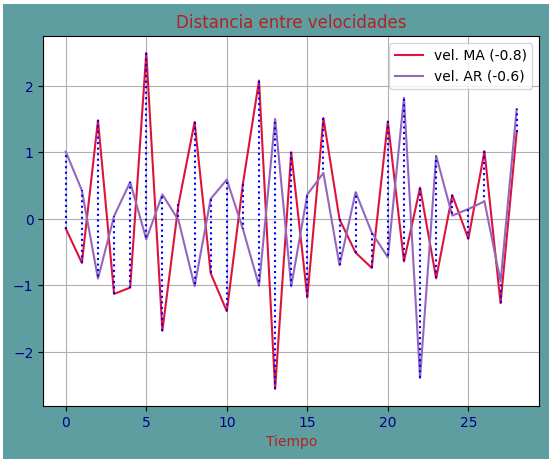


Figura 2.5: Distancia entre velocidades

2.2.3. Distancia entre aceleraciones

La definición de esta distancia se apoya en la siguiente idea de aceleración [13]:

Sean $a_t^x = v_t^x - v_{t-1}^x$ y $a_t^y = v_t^y - v_{t-1}^y$ las aceleraciones de la serie temporal x e y, respectivamente, en el instante t. Es decir, $a_t^x = x_t - 2x_{t-1} + x_{t-2}$ y $a_t^y = y_t - 2y_{t-1} + y_{t-2}$

La distancia entre dos series temporales basada en sus aceleraciones se define como [13]:

$$d_{\text{acc}}(x, y) = \sqrt{\sum_{t=3}^{m} (a_t^x - a_t^y)^2 z_t} \tag{2.4}$$

donde z_t es un peso adecuado para las aceleraciones del intervalo $[t-2, t]$.

Esta medida es aún más cercana a captar la dependencia temporal de la serie.

Observamos que las medidas basadas en las obervaciones se basan todas en la misma idea, pero se aplican a datos diferentes.

2.2.4. Área entre series

Dadas dos series temporales, calcular el área entre ellas no es complicado, pero es necesario hacerlo por intervalos porque se trata de una función definida a trozos. Una serie temporal es una función lineal en cada intervalo de tiempo observado.

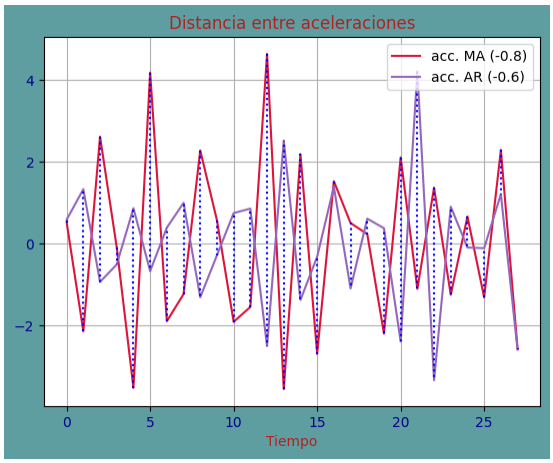


Figura 2.6: Distancia entre aceleraciones

No es necesario que las series compartan los instantes de recogida de datos, pero sí deben compartir el instante inicial y el final. En definitiva, todo se reducirá al cálculo integral, o al cálculo de superficies; y esto es independiente de los límites de integración o la forma del contorno creado. Si bien es cierto que los instantes extremos han de coincidir para que el área calculada sea interpretable para el caso que nos atañe.

Sean $\{X_t\} = x_{i_1}, \ldots, x_{i_m}$ e $\{Y_t\} = y_{j_1}, \ldots, y_{j_n}$ dos series temporales, donde $i_1 = j_1$ e $i_m = j_n$.

Se definen las funciones asociadas como:

$$f_x(t) = \begin{cases} \frac{t-i_1}{i_2-i_1}(x_{i_2} - x_{i_1}) + x_{i_1} & \text{for } t \in (i_1, i_2) \\ \ldots \\ \frac{t-i_{m-1}}{i_m-i_{m-1}}(x_{i_m} - x_{i_{m-1}}) + x_{i_{m-1}} & \text{for } t \in (i_{m-1}, i_m) \end{cases}$$

$$f_y(t) = \begin{cases} \frac{t-i_1}{i_2-i_1}(y_{i_2} - y_{i_1}) + y_{i_1} & \text{for } t \in (i_1, i_2) \\ \ldots \\ \frac{t-i_{m-1}}{i_m-i_{m-1}}(y_{i_m} - y_{i_{m-1}}) + y_{i_{m-1}} & \text{for } t \in (i_{m-1}, i_m) \end{cases}$$

Se define la distancia de área encerrada entre $\{X_t\}$ e $\{Y_t\}$ como:

$$d_{\text{area}}(x, y) = \int_{i_1}^{i_m} (f_x(t) - f_y(t)) \tag{2.5}$$

Otra forma de calcular este área es calcular, para cada serie, el polinomio que interpola todos los puntos observados y, a continuación, hallar el área entre los polinomios desde el instante inicial observado hasta el final.

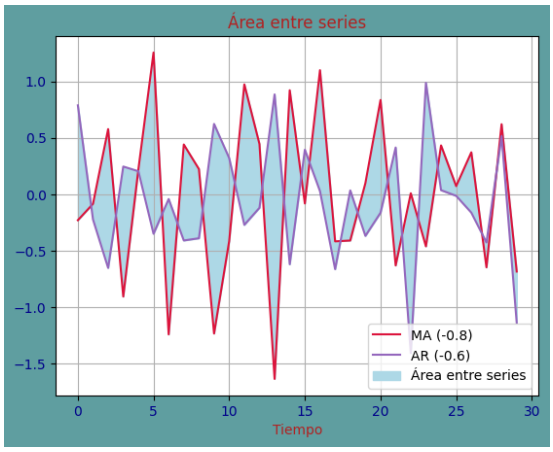


Figura 2.7: Ejemplo. Área entre dos series

2.2.5. Distancia DTW

En el caso de que $m \neq n$, una posible distancia es la Dynamic Time Warping, que se traduce como deformación dinámica del tiempo. Es una medida de similitud entre dos secuencias temporales que permite encontrar el alineamiento óptimo entre dos series temporales dadas incluso si hay un desfase en la velocidad o en el tiempo, cosa que la distancia euclídea no es capaz de medir [6]. Las imágenes 2.10 y 2.11 son ejemplo de cómo se genera el alineamiento óptimo entre dos series X e Y.

Para definir formalmente esta distancia, debemos ver x e y como una malla de tamaño m x n en la que cada elemento de es un punto (i,j) del plano, que representa una conexión cualquiera entre los elementos s_i y t_j. Es decir, hemos de ver s_i y t_j como dos rectas perpendiculares que dan pie a cuatro caminos diferentes desde su punto intersección.

Necesitamos también la noción de *camino deformado*. Un camino deformado, W, es una secuencia $[w_0, w_1, \ldots, x_k]$ de los puntos del mallado m x n que cumplen [6]:

- $w_0 = (0,0)$ y $w_k = (m,n)$
- $w_{l+1} - w_l \in \{(1,0),(0,1),(1,1)\} \forall l \in \{0,1,\ldots,k-1\}$

$W = [w_0, w_1, \ldots, x_k]$. Donde $w_k = (i,j)_k$.

El objetivo es llegar de (0,0) a (n,m) a través de los segmentos posibles que forman el mallado [6]. Por tanto, los posibles segmentos que se pueden tomar desde un punto intersección son los que permiten avanzar en la malla. Véase gráficamente en la imagen 2.8 la forma de avanzar.

Para entenderlo mejor, la imagen 2.9 muestra un camino deformado posible, que no es único.

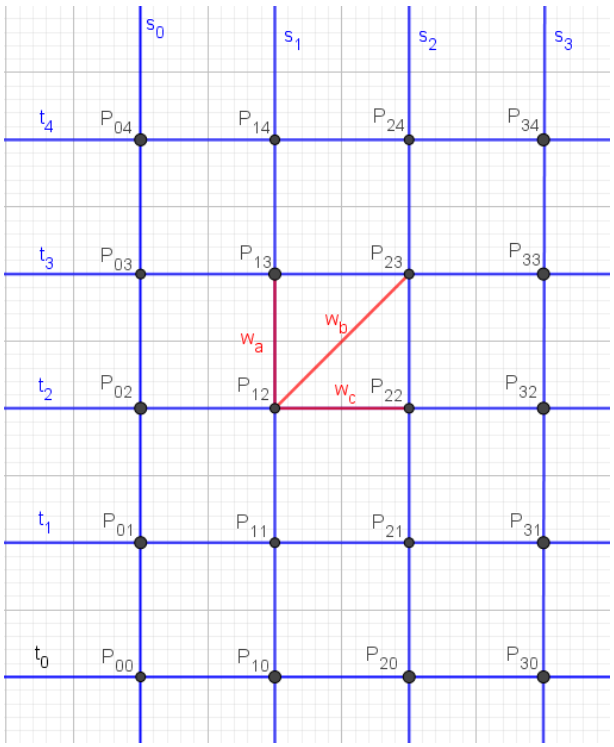


Figura 2.8: Segmentos factibles

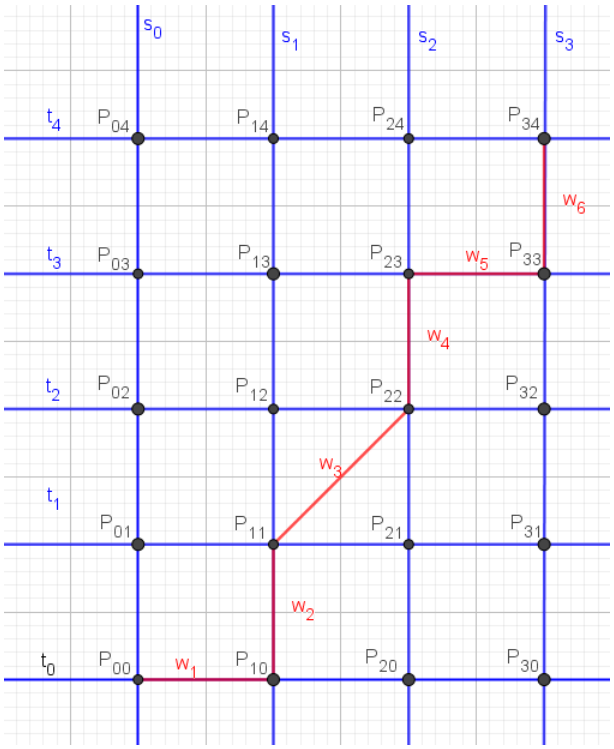


Figura 2.9: Ejemplo de camino deformado

Un ejemplo real de alineamientos y camino deformado entre series los encontramos en las imágenes 2.10 y 2.11 :

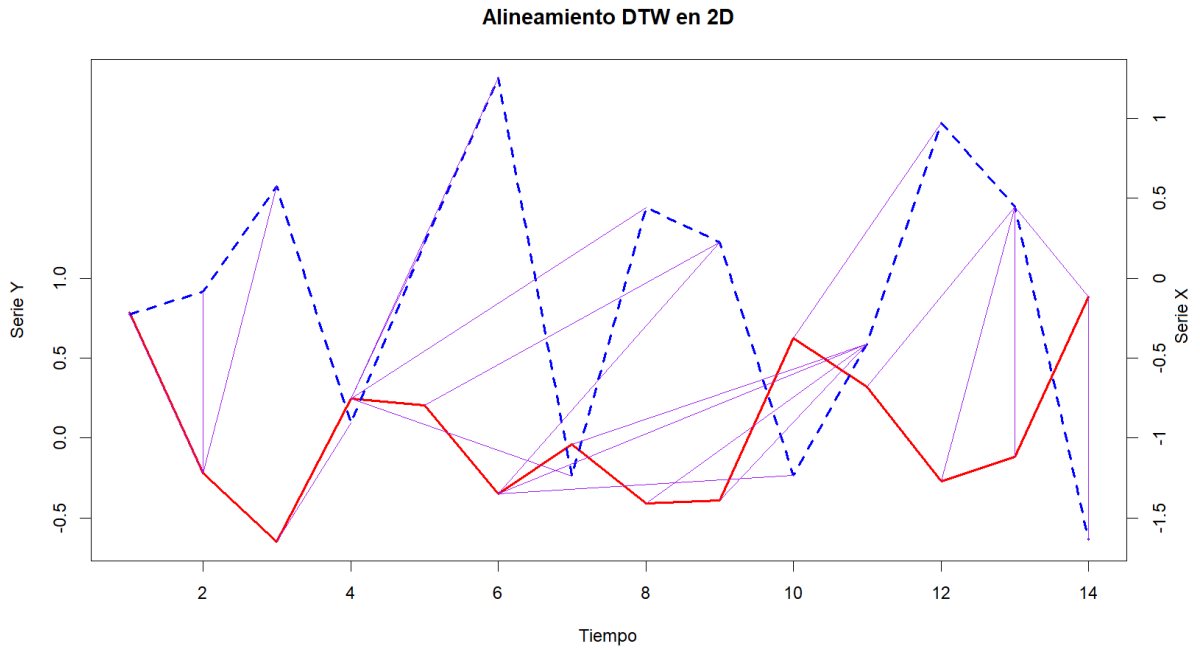


Figura 2.10: Alineamientos entre las series

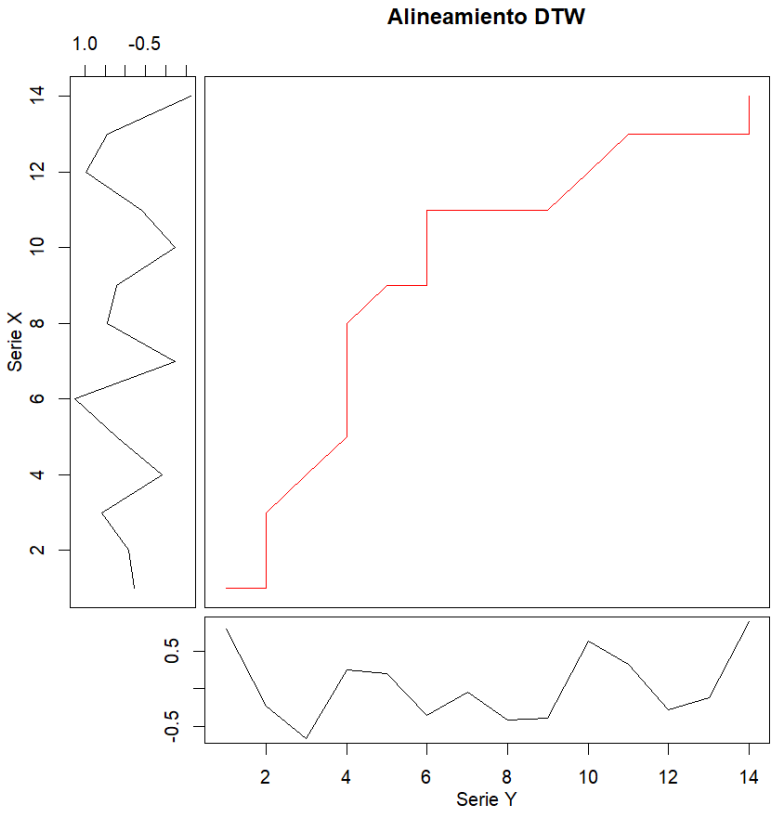


Figura 2.11: Camino deformado

2.3. Clustering basado en las características de las series

Si las series están formadas por muchas observaciones, no es una buena idea compararlas en base a las observaciones por el ruido y el hecho de que la estructura de autocorrelaciones (ver definición 2.3) de la serie temporal es ignorada, lo cual es una pérdida grande de información.

El teorema central del límite (ver teorema 2.3.1) aplicado a la función de autocorrelación muestral aporta luz en este sentido, porque cuantas más observaciones formen la serie, más exacta será la estructura de autocorrelaciones. Y, por tanto, es lógico hacer uso de ella cuando un número suficientemente grande de observaciones ha sido observado.

Una forma de proceder es comparar todas las características de las series. ¿Cuáles? La tendencia, la estacionalidad, los picos, la varianza de las observaciones, la media, las autocorrelaciones, la frecuencia, los outliers. Pero puede resultar un método rudimentario y será muy difícil definir una distancia en base a todos estos parámetros. Que, además,

esté bien definida y se pueda implementar en un algoritmo.

Pero esto resulta muy tedioso y demasiado fundamentado en la apariencia. Esto significa que los resultados no serán determinantes porque, como se comentó en la introducción, uno de los fines del análisis de conglomerados es descubrir grupos que a priori no eran evidentes.

Antes de continuar con el desarrollo de las distancias de esta sección, aclaremos los conceptos básicos para entender en qué se basan.

Definición. Función de autocovarianzas en general [18].

Sea T un conjunto de puntos en el tiempo, por ejemplo $\{1, 2, 3, \dots\}$. La función de autocovarianzas de un proceso estocástico $\{X_t\}$ tal que $Var(X_t) < \infty$ para cada $t \in T$, se denota por:

$\{\gamma(r,s), r, s \in T\}$, *donde* $\gamma(r,s) = E[(Y_r - EY_r)(Y_s - EY_s)]$

Definición. Proceso estocástico estacionario [18].

El proceso estocástico $\{X_t\}$ se dice un proceso estacionario si

$$\begin{aligned} E|X_t|^2 < \infty \forall t \in \mathbb{Z} \\ EX_t = m = \forall r, t \in \mathbb{Z} \\ \gamma_k = \gamma_{k+l} \forall k, l \in \mathbb{Z} \end{aligned} \tag{2.6}$$

Definición. Función de autocovarianzas (FACV) [18].

La FACV de un proceso estocástico estacionario es una función del retardo k, que es el número de periodos de separación entre las variables, que recoge el conjunto de las autocovarianzas del proceso. Se denota por:

$\{\gamma_k, k = 0, 1, 2, \dots\}$, *donde* $\gamma_k = E(Y_t - EY_t)(Y_{t+k} - EY_{t+k})$

Definición. Función de autocorrelación (FAC) [18].

La FAC de un proceso estocástico estacionario es una función del retardo k que recoge el conjunto de coeficientes de autocorrelación del proceso. Se denota por:

$\{\rho_k, k = 0, 1, 2, \dots\}$, donde $\rho_k = \frac{cov(Y_t, Y_{t+k})}{\sqrt{V(Y_t)V(T_{t+k})}} = \frac{\gamma_k}{\gamma_0}$

El coeficiente de correlación no tiene unidades, es una medida cuantifica la dependencia que existe entre dos variables aleatorias del proceso separadas k periodos.

Definición. Correlograma.

El correlograma es la representación de los coeficientes de autocorrelación. En la gráfica, el punto (x,y) corresponde a y = ρ_x. Normalmente, es una gráfica que cuando $x \to \infty$, $y \to 0$, porque la dependencia entre observaciones disminuye con el transcurso del tiempo.

Como la función de autocorrelación es simétrica respecto del orden k, el correlograma sólo se representa en la parte de $x \geq 0$.

Las figuras 2.12 y 2.13 son ejemplos de correlogramas.

Teorema 2.3.1. *Teorema central del límite aplicado a las series temporales [18].*

Si $\{X_t\}$ *es un proceso estacionario, entonces para cada* $h \in \{1, 2, \dots\}$, $\hat{\rho}(h)$ *es asintóticamente* $N(\rho(h), n^{-1}W)$*, donde*

$$\begin{aligned} \hat{\rho}(h)' &= [\hat{\rho}(1), \hat{\rho}(2), \dots, \hat{\rho}(h)], \\ \rho(h)' &= [\rho(1), \rho(2), \dots, \rho(h)] \end{aligned} \tag{2.7}$$

y W *es la matriz de covarianzas cuyo elemento* (i, j) *viene dado por la fórmula de Barlett:*

$$\begin{aligned} w_{ij} = \sum_{k=-\infty}^{\infty} \{&\rho(k+i)\rho(k+j) + \rho(k-i)\rho(k+j) + \\ &+ 2\rho(i)\rho(j)\rho^2(k) - 2\rho(i)\rho(k)\rho(k+j) - 2\rho(j)\rho(k)\rho(k+i)\} \end{aligned} \tag{2.8}$$

Definamos varias distancias en base a estos conceptos. Sean $\hat{\gamma}_x$ y $\hat{\rho}_x$ el vector de las autocovarianzas y las autocorrelaciones estimadas de la serie $\{X_t\}$, respectivamente.

2.3.1. Comparar las FACV muestrales

La distancia entre dos series temporales basada en la FACV muestral se define como [13]:

$$d_{\text{FACV}}(x, y) = \sqrt{(\hat{\gamma}_x - \hat{\gamma}_y)'\omega(\hat{\gamma}_x - \hat{\gamma}_y)} \tag{2.9}$$

donde $\hat{\gamma}$ es la función de autocovarianzas muestrales, y ω es una matriz de pesos adecuada. Por ejemplo, la identidad o una matriz diagonal con pesos geométricos en la diagonal.

La matriz ω debe aportar más importancia a las autocovarianzas muestrales de observaciones cercanas en el tiempo. En primer lugar, porque, al utilizar más datos para ser estimadas, son estimaciones más fiables. Y, en segundo lugar, porque tiene mayor interés la dependencia entre observaciones cercanas en el tiempo.

Por otro lado, sabemos que la función de autocorrelación (FACV) de una serie temporal describe la dependencia temporal entre las observaciones. La FACV tiene una observación

menos que la serie original, lo que la hace poco tratable. Sin embargo, Box and Jenkins demostraron que la FACV estimada hasta la diferencia $N/4$, con N la longitud de la serie, es suficiente para describir la dinámica de las series temporales. Lo ideal, según estos autores, es que $N \geq 50$ [13].

2.3.2. Comparar las FAC muestrales

La distancia entre dos series temporales basada en la FAC muestral se define como [13]:

$$d_{\text{FAC}}(x, y) = \sqrt{(\hat{\rho}_x - \hat{\rho}_y)'\omega(\hat{\rho}_x - \hat{\rho}_y)} \tag{2.10}$$

donde $\hat{\phi}$ es el vector de autocorrelaciones parciales muestrales, y ω es una matriz de pesos adecuada. Por ejemplo, la identidad o una matriz diagonal con pesos geométricos en la diagonal.

Observación 2.3.2. Igual que para la distancia d_{FACV}, tiene sentido que la matriz ω aporte más importancia a las autocorrelaciones muestrales o las autocovarianzas muestrales de observaciones cercanas en el tiempo. En primer lugar, porque son estimaciones más fiables, ya que utilizan una cantidad mayor de datos para ser estimadas. Y, en segundo lugar, porque tiene mayor interés, ya que la dependencia entre observaciones suele aumentar con la cercanía temporal.

2.3.3. Comparar los correlogramas

Las distancias d_{FACV} y d_{FAC} están basadas en la distancia entre dos vectores sin ninguna transformación, el vector de las autocovarianzas y de las autocorrelaciones, respectivamente. Sin embargo, también sería interesante comparar dichos vectores tomando sus componentes en valor absoluto. De esta forma, se compara el grado de influencia de las observaciones precedentes sin tener en cuenta si la relación es directa o inversa.

Gráficamente, si los correlogramas son los de las imágenes 2.12 y 2.13, se trata de comparar los valores de las autocorrelaciones en valor absoluto, como se representa en la figura 2.14, para prescindir de la relación directa o inversa, y haciendo incapié en el nivel de influencia.

Para medir la diferencia entre dos correlogramas, entonces, se propone el siguiente cálculo:

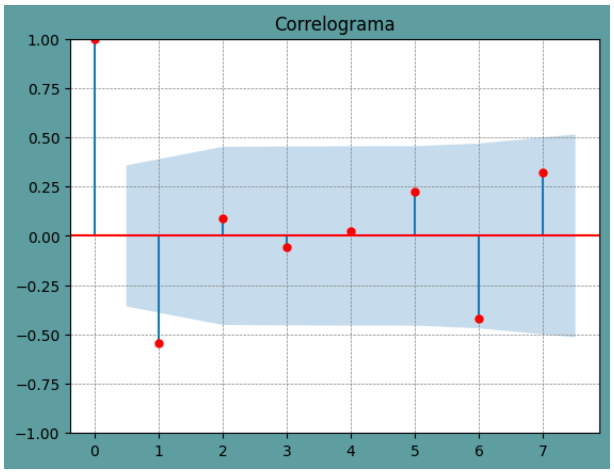


Figura 2.12: Correlograma de la serie X

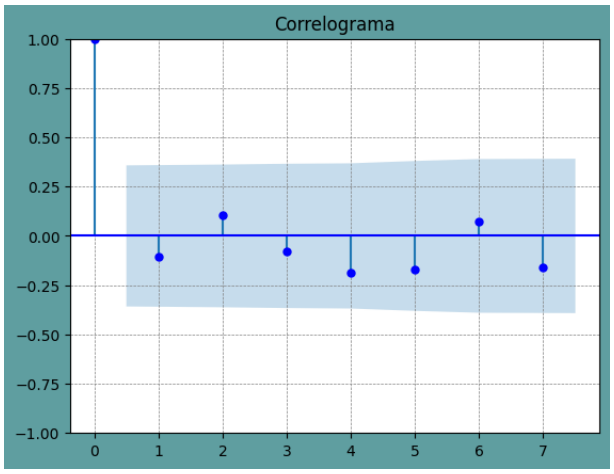


Figura 2.13: Correlograma de la serie Y

$$d_{\mathrm{COR}}(x,y) = \sqrt{(|\hat{\rho}_x| - |\hat{\rho}_y|)'(|\hat{\rho}_x| - |\hat{\rho}_y|)} \tag{2.11}$$

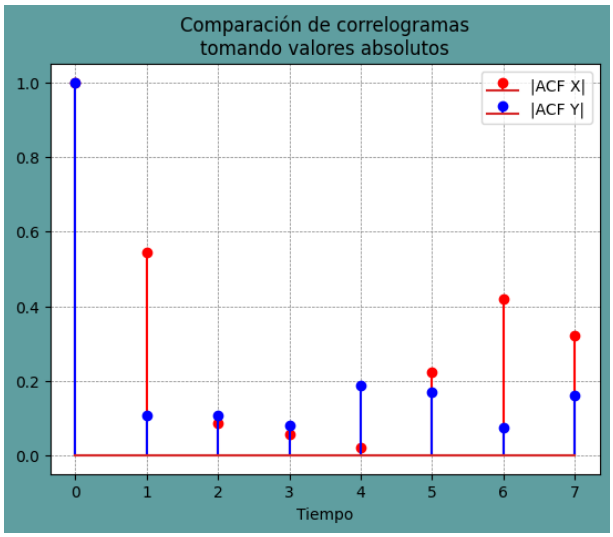


Figura 2.14: Comparación de correlogramas

2.4. Clustering basado en modelos ajustados a las series

Los métodos basados en modelos para hacer análisis por conglomerados de series temporales se fundamenta en que las series que han sido generadas por el mismo modelo deben ser similares [13]. Esta clasificación puede hacerse en términos de los parametros estimados para cada modelo o en función de los residuos de los modelos ajustados. Es decir, hemos de definir una distancia entre modelos estadísticos.

En esta sección se tratarán tres tipos de modelos: modelo de regresión lineal múltiple, modelo ARIMA y modelos de tipo cadenas de Markov.

Algunas formas de proceder:

- Ajustar un modelo de regresión lineal múltiple a cada serie y medir la distancia entre los coeficientes de los modelos. (Ver apartado 2.4.3)
- Ajustar un modelo ARIMA a cada serie y medir la distancia entre los coeficientes autorregresivos. (Ver apartado 2.4.4)
- Ajustar un modelo de regresión lineal múltiple a cada serie y calcular las distancias entre los residuos. (Ver apartado 2.4.5)
- Ajustar un modelo ARIMA a cada serie y calcular las distancias entre los residuos. (Ver apartado 2.4.6)
- Ajustar modelos de tipo cadena de Markov ocultas. (Ver apartado 2.4.7)

En primer lugar, se analizarán los modelos estadístico ARIMA y Regresión Lineal Múltiple (RLM).

2.4.1. Modelo de regresión lineal múltiple (RLM)

El modelo de regresión lineal múltiple es aquel en el que es posible definir cada observación y como:

$$y = \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p + \varepsilon \tag{2.12}$$

donde, $x_1, \ldots, x_p$ son valores conocidos, $\beta_1, \ldots, \beta_p$ son coeficientes desconocidos, y ε es el error aleatorio [24].

Si hay más de una observación, lo cual es el caso de las series temporales, entonces se tiene para cada observación $y_i, i \in \{1, \ldots, n\}$:

$$y_i = \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} + \varepsilon_i \tag{2.13}$$

Esta expresión se puede aplicar a las series temporales. Sea $Y_t = (y_1, \ldots, y_n)$ una serie temporal. Entonces, [24]:

$$Y_t = XB + \varepsilon \tag{2.14}$$

donde X es una matriz de dimensión n x p; y ε es el vector de errores aleatorios que debe satisfacer las siguientes hipótesis distribucionales:

1. Media nula. $E(\varepsilon_i) = 0, \forall i$
2. Incorrelación. $E(\varepsilon_i \varepsilon_j) = 0, \forall i \neq j$
3. Homocedasticidad. $V(\varepsilon) = \sigma^2 I_n$, donde I_n es la matriz identidad de dimensión n.

El vector B es la incógnita que se ha de estimar para definir el modelo.

2.4.2. Modelo ARIMA

Antes de definir el modelo ARIMA, son necesarios los siguientes conceptos:

Definición. Ruido blanco [11].

Un **ruido blanco (RB)**, α_t, es un proceso estocástico formado por una secuencia de variables aleatorias que cumplen:

- $E[Y_t] = 0, \forall t$
- $V(\alpha_t) = \sigma^2, \forall t$
- $Cov(\alpha_t, \alpha_s) = 0, \forall t \neq s$

Definición. Polinomio de medias móviles [9].

Sean $\theta_1, \theta_2, \ldots, \theta_q$ constantes reales, y sea L el operador retardo. Se define el proceso MA(q) como

$$Y_t = \alpha_t - \theta_1 \alpha_{t-1} - \theta_2 \alpha_{t-2} - \cdots - \theta_q \alpha_{t-q}, \text{ donde } \alpha_t \sim RB(0, \sigma^2) \tag{2.15}$$

$\theta_q(L) = 1 - \theta_1 L - \theta_2 L^2 - \cdots - \theta_q L^q$ es el polinomio de medias móviles del proceso Y_t, y $(\theta_1, \theta_2, \ldots, \theta_q)$ es el vector de parámetros de medias móviles.

Definición. Polinomio autorregresivo [9].

Sean $\phi_1, \phi_2, \ldots, \phi_p$ constantes reales, y sea L el operador retardo. Se define el proceso AR(p) como

$$Y_t = \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \cdots + \phi_p Y_{t-p} + \alpha_t, \text{ donde } \alpha_t \sim RB(0, \sigma^2) \tag{2.16}$$

Despejando α_t, se tiene:

$$\begin{aligned}\alpha_t &= Y_t - \phi_1 Y_{t-1} - \phi_2 Y_{t-2} - \cdots - \phi_p Y_{t-p} = \\ &= (1 - \phi_1 L - \phi_2 L^2 - \cdots - \phi_p L^p) Y_t = \\ &= \phi_p(L) Y_t\end{aligned} \tag{2.17}$$

$\phi_p(L) = 1 - \phi_1 L - \phi_2 L^2 - \cdots - \phi_p L^p$ es el polinomio autorregresivo del proceso Y_t; y $(\phi_1, \phi_2, \ldots, \phi_q)$ es el vector de parámetros autorregresivos.

Definición. Modelo ARIMA [9].

Sea $d \in \mathbb{R}$. Una serie temporal $\{Y_t\}$ sigue un modelo integrado autorregresivo de medias móviles (modelo ARIMA) si la serie definida como el resultado de aplicar el operador diferencia d veces sobre Y_t, $W_t = \nabla^d Y_t$, es un proceso ARMA estacionario. Es decir, existe un polinomio autorregresivo, $\phi_p(L)$, y otro de medias móviles, $\theta_q(L)$, tales que es posible definir W_t como

$$\phi_p(L)\nabla^d Y_t = \theta_q(L)\alpha_t, \text{ donde } \alpha_t \sim RB(0, \sigma^2) \tag{2.18}$$

2.4.3. Distancia entre coeficientes. Modelo de RLM

La idea para definir esta matriz de distancias entre series temporales es ajustar un modelo de regresión para cada serie del conjunto observado.

Empleando la teoría de la sección 2.4.1, definimos las variables dependientes como vectores sombrero. Es decir, vectores con un único elemento no nulo: el de la posición que nos interesa.

Sea $Y_t' = (y_1, \ldots, y_n)$ una serie temporal. Entonces, los vectores sombrero son $v_k' = (v_1 k, \ldots, v_n k)$ con $v_i k = 0, \forall i \neq k$, siendo k el instante de interés, y $v_k k = 1$. Si la serie es estacional, sólo serán necesarios tantos vectores como estaciones. Por ejemplo, la serie temporal del IPC recoge datos cada mes, por lo que habrá doce vectores sombrero, para calcular los valores de la serie en cada estación. En cambio, si la serie recoge datos cada minuto del primer cuarto de un partido de baloncesto, necesitaremos un vector para cada instante de tiempo observado.

También será necesario un vector de instantes para que el modelo recoja el paso del tiempo. Este vector es $w' = (1, 2, \ldots, n)$.

Por tanto, las variables explicativas son: cada instante de tiempo, y el transcurso del tiempo.

Con las variables explicativas, se puede escribir la serie observada, Y_t, como:

$$Y_t = y_1 v_1 + y_2 v_2 + \cdots + y_n v_n \tag{2.19}$$

Y el modelo de regresión lineal (Ver apartado 2.14) se define como sigue:

$$\begin{aligned} &Y = XB + \varepsilon, \text{ donde} \\ &Y = Y_t; \\ &X = [I_n | w]; \\ &\varepsilon \text{ es desconocido ;} \\ &B = \text{ es la incógnita} \end{aligned} \tag{2.20}$$

Una vez que se ha calculado el modelo de cada serie, se han de comparar los coeficientes de todos los modelos.

En definitiva, lo que se consigue con esta manera de proceder es resolver el problema de análisis de conglomerados con datos serie temporales a través de otro problema de análisis de conglomerados con datos multidimensionales no temporales. Este estudio es más sencillo que el anterior porque la distancia euclídea es una buena opción en este caso, aunque se podrían implementar otras técnicas.

Por lo tanto, si M es la matriz de distancias, M_{ij} es la distancia euclídea entre el vector de coeficientes del modelo ajustado sobre la serie i y el vector de coeficientes del modelo ajustado sobre la serie j. Es decir, si $Y_i = \beta_{i0} + x_{i1}\beta_{i1} + x_2\beta_{i2} + \cdots + x_{ip}\beta_{ip} + \varepsilon$ $Y_j = \beta_{j0} + x_{j1}\beta_{j1} + x_{j2}\beta_{j2} + \cdots + x_{jp}\beta_{jp} + \varepsilon$

Entonces,

$$M_{ij} = \sqrt{\sum_{k=1}^{p} (\beta_{ik} - \beta_{jk})^2} \tag{2.21}$$

Observación 2.4.1. Es de interés apuntar que, en el caso de las series temporales, el hecho de incluir o no a la constante del modelo depende del objetivo perseguido con el análisis de conglomerados. Si los valores observado tienen mucha importancia, es conveniente tener en la constante del modelo. Si, de otro modo, lo que más importa es el perfil de la serie, entonces no es buena idea incluirlo porque aumentará las distancias entre series sin un motivo de peso.

2.4.4. Distancia entre coeficientes. Modelo ARIMA

El método para definir una matriz de distancias entre series temporales usando los coeficientes del modelo ARIMA es similar a la que hemos descrito para el modelo RLM en la sección 2.4.3: ajustar un modelo ARIMA para cada serie del conjunto observado y comparar los coeficientes. Es decir, hay que definir una distancia entre modelos ARIMA usando los coeficientes.

Sean X_t e Y_t dos series temporales. Supongamos que existen $\hat{\phi}_p(L)$, d y $\hat{\theta}_q(L)$ tales que se puede modelar X_t de la siguiente forma:

$$\hat{\phi}_p(L)\nabla^d X_t = \hat{\theta}_q(L)\alpha_t, \text{ donde } \alpha_t \sim RB(0, \sigma^2) \tag{2.22}$$

y análogamente para Y_t:

$$\hat{\psi}_r(L)\nabla^c X_t = \hat{\rho}_s(L)\beta_t, \text{ donde } \beta_t \sim RB(0, \sigma^2) \tag{2.23}$$

En esta ocasión, el problema se resume en la siguiente pregunta: ¿qué ocurre si $p \neq r$, o $q \neq s$, o $d \neq c$?

En realidad, si $p \neq r$, o $q \neq s$, lo único que ocurre es que los coeficientes de más que aparecen en uno se corresponden con coeficientes nulos en el otro. Por ejemplo, consideremos $p = 0 = r$ y $q = 2, s = 1$. Entonces sabemos que: $\hat{\theta}_1(L) \neq 0 \neq \hat{\theta}_2(L)$, $\hat{\theta}_k(L) = 0$ $\forall k > 2$, $\hat{\rho}_1 \neq 0$ y $\hat{\rho}_k = 0$ $\forall k > 1$.

Por ello, el problema se reduce a la situación $d \neq c$, que son los parámetros de la diferenciación para tener series estacionarias. Supongamos que $d = 2$ y $c = 1$, con los otros parámetros que han sido considerados como ejemplo.

$$\nabla^d X_t = \hat{\rho}_2(L)\alpha_t, \text{ donde } \alpha_t \sim RB(0, \sigma^2)$$

$$\nabla^c X_t = \hat{\rho}_1(L)\beta_t, \text{ donde } \beta_t \sim RB(0, \sigma^2)$$

Con los coeficientes del modelo lineal no existe este dilema porque para todas las series se ajusta un polinomio de grado 1. Habría un inconveniente si, en vez de simplificar y ajustar un modelo lineal, se hiciera una búsqueda previa del mejor ajuste de cada serie.

2.4.5. Distancia entre residuos. Modelo RLM

En esta sección se desarrolla una metodología muy novedosa para cuantificar la distancia que existe entre dos series. Se trata aprovechar la técnica de validación de modelos y la medición del rendimiento esperado de los modelos.

Supongamos que se han observado m series temporales $(Y_1, \ldots, Y_m)$, no necesariamente todas de la misma longitud. Cada serie temporal está formada por una serie de datos.

El proceso es el siguiente: Tomamos $Y_i = (y_{1i}, \ldots, y_{n_{ii}})$ y ajustamos un modelo de RLM, como se ha explicado en la sección 2.4.1. Hasta aquí, es el mismo proceso que hubiéramos seguido si quisiéramos ajustar un modelo RLM al conjunto de todos los datos observados, independientemente de la serie a la que pertenezcan, tomando el subconjunto de observaciones pertenecientes a Y_i como conjunto de entrenamiento. Se llega a la misma situación desde dos caminos totalmente independientes en cuanto a interpretación de los datos. A continuación, hemos de validar el modelo en las otras series observadas.

Para llevar a cabo todos estos pasos, se utiliza el mismo razonamiento que en la sección 2.4.1: crear variables explicativas y construir el modelo. La originalidad viene con la validación del modelo. Básicamente, se predicen los valores según el modelo en los instantes donde se tienen datos y se mide el grado de disparidad entre los resultados esperados y los observados, con todas las series. Esa medida, que bien puede ser la suma de residuos al cuadrado o la raíz cuadrada del error cuadrático medio, por ejemplo.

Si M es la matriz de disimilaridades, $M[i,j]$ es la raíz cuadrada del error cuadrático medio cometido tras predecir las observaciones de la serie j mediante el modelo de regresión ajustado sobre los datos de la serie i. Es decir, si

$$\begin{gathered} Y_i = \hat{\beta}_{i0} + x_{i1}\hat{\beta}_{i1} + x_2\hat{\beta}_{i2} + \cdots + x_{ip}\hat{\beta}_{ip} + \varepsilon; \\ \hat{Y}_i = (\hat{y}_{1i}, \ldots, \hat{y}_{n_{ii}}); \\ Y_j = (y_{1j}, \ldots, y_{n_{jj}}) \end{gathered}$$

Entonces,

$$M_{ij} = \sqrt{\sum_{k=1}^{p} (\hat{y}_{ki} - y_{kj})^2}, \text{ donde } p = min\{n_i, n_j\} \tag{2.24}$$

Sin embargo, como esta definición no crea una matriz simétrica ni una matriz cuya diagonal sean ceros, construiremos M_{sim} de la siguiente forma para que cumpla esos requisitos:

1. $M_{sim}[i,i] = 0, \forall i$
2. $M_{sim}[i,j] = M_{sim}[j,i] = \max\{M[i,j], M[j,i]\}, \forall i \neq j$

De esta forma, tenemos una matriz simétrica y cuya diagonal está compuesta por ceros.

2.4.6. Distancia entre residuos. Modelo ARIMA

El método para definir una matriz de distancias entre series temporales usando los residuos del modelo ARIMA es similar a la que hemos descrito en la sección 2.4.5: ajustar un modelo ARIMA para cada serie del conjunto observado y medir la validez del modelo en las demás series. Es decir, hay que definir una distancia entre modelos ARIMA usando los residuos.

En esta ocasión no se tienen los problemas descritos en la sección 2.4.2.

2.4.7. Modelo cadenas de Markov

Al igual que con los modelos anteriores, esta sección comienza con una introducción teórica sobre las cadenas de Markov. Para entrar en más detalle, leer las fuentes [2], [4], [16] y [19], que son los documentos base de esta sección.

Definición. Cadena de Markov [2].

Sea $\{X_n\}_{n\geq 0}$ una sucesión de variables aleatorias que toman valores en el espacio numerable de estados E. $\{X_n\}_{n\geq 0}$ se dice una cadena de Markov si cumple la propiedad de Markov:

$$P(X_{n+1} = k|X_n = e, X_{n-1} = e_{n-1}, \ldots, X_0 = e_0) = P(X_{n+1} = k|X_n = e), \quad \text{donde } e, e_{n-1}, \ldots, e_0 \in E \tag{2.25}$$

Si $P(X_{n+1} = k|X_n = e)$ no depende de n, entonces se llama cadena de Markov homogénea.

Definición. Matriz de transición [2].

La matriz $P = \{p_{ij}\}_{i,j\in E}$, donde $p_{ij} = P(X_{n+1} = j|X_n = i)$ es la probabilidad de transición de i a j. Se cumple que:

$$p_{ij} \geq 0, \text{ y } \sum_{k\in E} p_{ik} = 1 \text{ para todos los estados } i, j.$$

La matriz de transición es una forma de definir una cadena de Markov. También se puede representar en forma de grafos dirigidos con pesos. Las imágenes 2.15 y 2.16 son un ejemplo de ello.

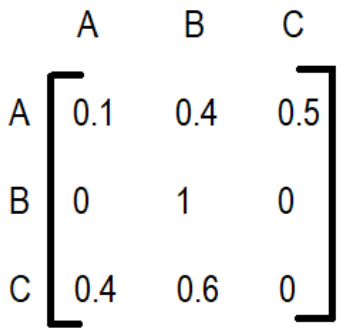


Figura 2.15: Matriz de transición

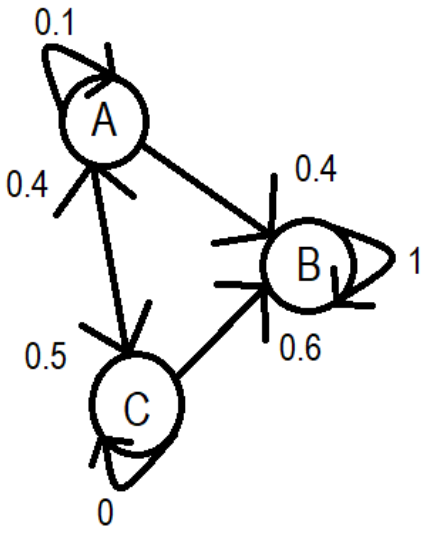


Figura 2.16: Grafo asociado a la matriz de transición

Definición. Cadena de Markov oculta [4].

Una cadena de Markov oculta (CMO) es un proceso estocástico doble, $(\{X_n\},\{Y_n\})_{n=1}^{T}$, donde:

- $\{X_n\}_{n\geq 0}$ es una cadena de Markov homogénea.
- $\{Y_n\}_{n\geq 0}$ es un proceso observable y que cumple:

$$P(\{X_n\}_{n\geq 0}, \{Y_n\}_{n\geq 0}) = \prod_k P(X_k|Y_k)$$

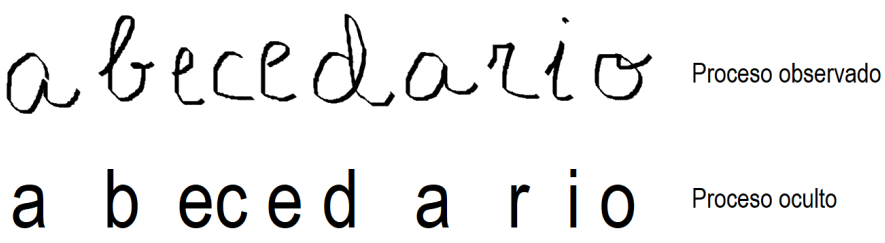


Figura 2.17: Ejemplo. Cadena de Markov oculta con datos categóricos

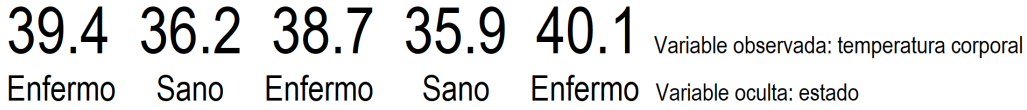


Figura 2.18: Ejemplo. Cadena de Markov oculta con datos continuos

Definición. Matriz de emisión [4].

Si la serie observada es discreta, la matriz $E = \{e_{it}\}_{i \in E}$, donde $e_{it} = P(Y_t = y_t | X_t = i)$ es la probabilidad de que la observación y_t haya sido ëmitidaïpor el estado i. Se cumple que:

$$e_{it} \geq 0, \text{ y } \sum_{k \in E} e_{kt} = 1 \text{ para todos los instantes } t.$$

En el caso de observar una serie temporal continua, sólo se puede hablar de $P(y_t|\Lambda)$, donde Λ es el conjunto de parámetros que definen la distribución de la cadena oculta.

Supongamos que $X = [x_1, \ldots, x_N]$ son los estados ocultos, A es la matriz de transición, $Y = [y_1, \ldots, y_T]$ son las observaciones, Λ es el conjunto de parámetros que define la CMO, y π es la probabilidad de emisión a priori.

Sea $\Theta = (\pi, A, \Lambda)$ la terna de parámetros que definen la distribución de la CMO [16].

Recordemos el teorema de Bayes:

Teorema 2.4.2. *Sean A y B dos sucesos factibles. Entonces,* $P(A|B) = \frac{P(A \cap B)}{P(B)}$

Teniendo en cuenta este teorema y la propiedad de Markov (Ver la propiedad 2.25), la distribución de probabilidad conjunta de (X, Y) viene dada por [16]:

$$P(X, Y|\Theta) = P(x_1|\pi) \left[\prod_{t=2}^{T} P(x_t|x_{t-1}, A) \right] \prod_{m=1}^{T} P(y_t|\Lambda) \tag{2.26}$$

El estado de una cadena de Markov está caracterizado por su distribución de transición entre estados ocultos, la distribución de emisión entre los estados observados y ocultos, y por la probabilidad a priori para el primer estado oculto.

Trasladando esta teoría al caso que nos ocupa, es posible considerar las series de tiempo como cadenas de Markov ocultas, tratando las series como secuencia de estados observados.

El primer problema que se ha de confrontar al construir un modelo HMM sobre un conjunto de observaciones es definir los estados. Si tomamos las series de observaciones

como los estados, el espacio de estados cambia con el tiempo y es cada vez más grande, lo que hace que la matriz de transición sea poco manejable.

Una vez definidos los estados, se han de encontrar los parámetros óptimos que describen el conjunto de datos. En el caso que nos ocupa, este conjunto de observaciones es una serie temporal. Esto se formula como sigue [4]:

$$\Lambda^* = \text{argmax}_\Lambda P(X|\Lambda)$$

Una vez que los modelos de cadenas de Markov ocultas están bien definidos, ¿cómo utilizarlo para el análisis de conglomerados con series temporales? Una posibilidad es utilizar las matrices de transición para comparar los modelos que definen. Por ejemplo, definir la distancia entre cadenas de Markov ocultas como la distancia de Frobenius de la diferencia de matrices de transición asociadas.

Es decir, si A y B son las matrices de transición asociadas a dos cadenas de Markov ocultas, X e Y, respectivamente, la distancia entre ellas se define como sigue:

$$d_{CMO}(x, y) = \sqrt{\text{tr}((A - B)^*(A - B))}, \tag{2.27}$$

donde tr() es la función *traza* de una matriz.

2.5. Validez de los resultados

El análisis cluster es un método no-supervisado, por lo que evaluar el resultado del algoritmo es muy difícil, aunque igualmente importante. Pero, ¿en base a qué se mide la bondad de los conglomerados?, ¿cuándo es aceptable o buena la solución?

El objetivo es seleccionar los conglomerados que mejor se ajustan a los datos.

Existen medidas que se calculan a partir de las distancias dentro de cada conglomerado y las distancias entre clusters. Por ejemplo, la puntuación de Davies-Bouldin, que está implementada en Python. Su cálculo se realiza de la siguiente forma:

[21] Sean C_i, $i \in 1, \ldots, k$. Sea s_i la distancia media entre los puntos del cluster i y el centroide del cluster i. Por último, sea d_{ij} la distancia entre los centroides de los clusters i y j.

$$R_{ij} = \frac{s_i + s_j}{d_{ij}} \tag{2.28}$$

Con esto, se define:

Definición. Índice de Davies-Bouldin (DB) [21].

$$DB = \frac{1}{k} \sum_{i=1}^{k} \underset{i \neq j}{max} R_{ij} \tag{2.29}$$

Por lo tanto, cuanto menor sea esta puntuación, mejor será la clasificación realizada, porque significa que se está consiguiendo homogeneidad dentro de los conglomerados y heterogeneidad entre conglomerados distintos.

Capítulo 3

Software disponible

A la hora de construir un algoritmo efectivo y eficiente hay que saber cómo tratar con:

- La alta dimensionalidad.
- El ruido.
- La alta correlación de características.
- Las diferentes longitudes de las series.

Del mismo modo, es necesario clasificar y saber elegir de entre las distancias entre series según el interés del análisis.

Para el desarrollo del caso práctico se ha empleado el lenguaje de programación Python, que cuenta con numerosos módulos tanto para el cálculo de distancias y ajuste de modelo, como para la implementación de algoritmos para el análisis de conglomerados. Incluso, existen módulos específicos para hacer el análisis completo de conglomerados entre series temporales. Es más, normalmente las distancias que usaremos están todas ya definidas y existe todo un paquete de funciones en torno a ella. Estas funciones se aprovecharán, pero también serán programadas otras a mano.

La ventaja principal que tiene trabajar con paquetes, módulos o librerías de diferentes lenguajes es que detrás de todos ellos hay un equipo de expertos programadores y conocedores del tema sobre el que programan. Además, cada funcionalidad está avalada por el mayor o menor uso que de ella hacen los beneficiarios de la aplicación. Por tanto, utilizar aquello que ya está implementado y bien documentado, de forma que no se utilice como caja negra sino conociendo los instrumentos, siempre es una garantía mayor de eficiencia y eficacia del código.

3.1. Desarrollo en R

El lenguaje R tiene funcionalidades y librerías específicas para el análisis cluster con datos temporales. La gran mayoría de distancias y formas de proceder que se han comentado en este trabajo están incluidas en los paquetes `TSclust` y `dtwclust`.

El paquete `TSclust` [17] es una recopilación de medidas de disimilaridad entre series temporales. Algunas de las que se han propuesto a lo largo de este trabajo ya están implementadas en este paquete, como las basadas en autocorrelaciones, autocorrelaciones parciales y en el modelo ARIMA. También está implementada la distancia DTW. Para leer el manual de referencia, se puede visitar la Página oficial. Incluso, incluye una función para evaluar los clusters finales, aunque ésta presupone que existen unas etiquetas iniciales reales y verdaderas con las que comparar el resultado final, lo cual choca con el hecho de que estemos empleando técnicas de aprendizaje automático no supervisado. Lo más normal en análisis cluster con series temporales es no disponer de esta información, porque justamente la finalidad del estudio es analizar diferentes formas de agrupación, que pueden no ser predecibles.

El paquete `dtwclust` [20] está más acotado porque gira exclusivamente en torno a la distancia DTW, y la implementa usando distintas técnicas de análisis cluster, entre ellas, el clustering jerárquico. Para leer el manual de referencia, se puede visitar la Página oficial.

3.2. Desarrollo en Python

3.2.1. Librerías

Python es un lenguaje con, al igual que R, cantidad de librerías para diferentes ámbitos. Para este caso en concreto, será fundamental y básico el manejo de los módulos `NumPy` y `Pandas`, incluso será muy útil hacer uso de `datetime` para una buena lectura y control de las fechas, ya que trabajaremos con series de tiempo.

El nombre del módulo `NumPy` viene de Numerical Python. Es la principal librería matemática para manejar datos numéricos, principalmente en formatos de tipo matricial y arrays multidimensionales, que son las estructuras básicas de este paquete. Quizás, una desventaja de este módulo en el caso que nos ocupa es que las estructuras básicas de NumPy únicamente admiten guardar datos de un mismo tipo. Sin embargo, ha sido utilizada para desarrollar otros paquetes como `Pandas`, `scikit-learn` y `Matplotlib`, por lo que tiene una gran importancia y un gran valor. Además, como todos los paquetes de

tanta utilidad, no cesa de actualizarse e intentar mejorar para beneficio de los usuarios. En su página oficial se puede consultar la documentación oficial, que es muy clara y completa.

La librería `Pandas` está orientada al análisis y manipulación de datos. La estructura del DataFrame viene de este módulo, y supone un avance en Python porque tratar con un conjunto de datos en formato DataFrame es mucho más eficiente que cualquier otro tipo de datos. Será muy práctica para la lectura de datos y la manipulación de las series temporales que se organizarán en DataFrames. En este formato será posible guardar fechas e índices sin problema, ya que se pueden mezclar diferentes tipos de datos en un mismo DataFrame. Para conocer más en detalle todas las posibilidades y herramientas que `Pandas` ofrece, se puede acceder a la página web oficial.

Por otro lado, la librería `Matplotlib` sirve para crear gráficos estáticos, animados e, incluso, interactivos en Python. En particular, para poder utilizarla en línea, es necesario especificarlo con el comando `%matplotlib inline`. Es habitual utilizar la interfaz `Matplotlib.pyplot`, que proporciona una forma de representar similar a `MATLAB`, en el sentido que abre las figuras en la pantalla.

Para definiciones y funciones más concretas, el módulo `statsmodels` será de gran ayuda. Como su nombre indica, es fundamentalmente una librería que tiene implementados modelos estadísticos y métodos para estimar sus parámetros, por lo que todo lo que usemos que esté relacionado con modelos ARIMA, RLM o Hidden Markov Chaines, podrá ser implementado usando este módulo. Es posible consultar la documentación en la página web oficial.

También usaremos el módulo `scikit-learn`, con el que también se pueden ajustar modelos. Aunque ésta es una herramienta para desarrollar aprendizaje automático en Python, principalmente. `Scikit-learn` es una librería muy amplia que ha desarrollado técnicas de aprendizaje supervisado y no supervisado, selección y evaluación de modelos, procesamiento y visualización de datos, entre otros temas. Para profundizar en este paquete, se puede visitar la página web oficial.

Hasta aquí, se han comentado las principales librerías que se usarán en la aplicación del capítulo 4. A continuación, se exponen los módulos que también serán utilizados.

Al igual que R, existe un módulo que tiene implementada la distancia DTW, el módulo `dtaidistance`, y que se aprovechará en el caso práctico para aplicar análisis cluster con esta distancia.

Por otro lado, será interesante trabajar con el módulo `datetime` para manipular las fechas de las series temporales del problema.

3.2.2. ¿Por qué Python?

La elección de Python frente a R en el desarrollo del caso práctico es méramente una decisión de aprovechamiento académico. Este TFG es el final del Doble Grado en Matemáticas y Estadística, unos estudios de cinco años de duración en los que la herramienta básica de trabajo ha sido R, en materia estadística. Por lo que cambiar al lenguaje Python tiene la finalidad de iniciarse en la programación con un nuevo lenguaje que encabeza la lista de lenguajes de programación actualmente. En resumen, se trata de descubrir más otro lenguaje para estar lo más actualizada posible.

3.2.3. Otras herramientas

Para trabajar con Python se ha empleado Google Colab, que es un producto de Google Research que permite escribir código de Python en cuadernos Jupyter en línea, lo cual es muy práctico porque es posible acceder al código en cualquier momento. Además, los cuadernos Jupyter también incluyen la posibilidad de utilizar Markdown, por lo que es muy sencillo organizar el documento con una tabla de contenidos para facilitar la comprensión y el desplazamiento en el documento.

A continuación presentamos un caso práctico, ejemplo de la forma de proceder en el análisis de conglomerados con series temporales. Se tomarán datos inventados para mostrar un ejemplo. Se trata de las series que han sido representadas en las figuras 2.3 a 2.5 . Se pretende que el resultado sea que las series se agrupen en tres conglomerados tal y como aparece en las figuras mencionadas.

Importación de módulos

En primer lugar, se importan las librerías y los módulos principales que servirán a lo largo de toda la práctica. Más adelante, cuando sean necesarias funciones específicas de algún otro módulo, entonces se importarán. Así el proceso queda más claro.

Hay librerías que no vienen en el paquete base de Python, por lo que antes de importarlas para utilizarlas, es necesario instalarlas. Las forma de instalar una librería en los cuaderno de Jupyter es la siguiente:

```
# quiet sirve para ignorar los mensajes de instalación
!pip install datetime --quiet
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import datetime
import warnings
%matplotlib inline
```

Lectura y preparación de datos

A continuación, se lee la base de datos que contiene los valores de las series temporales.

Se define el nombre de la base de datos que se va a usar.

```
file = 'series.xlsx'
```

A continuación, se realiza la lectura de los datos de cada periodo especificando que la primera fila corresponde al nombre de las columnas.

```
datos = pd.read_excel(io = file,header = 0)
```

Como la primera columna indica la fecha en que ha sido observado cada dato, se puede utilizar como nombre de la fila, de modo que no se confunda con un grupo ECOICOP. Una forma de hacerlo es la siguiente.

```
datos.rename(index=datos["Fecha"],inplace=True)
datos = datos.drop(['Fecha'], axis=1)
```

Es conveniente comprobar que la lectura se ha realizado correctamente antes de continuar.

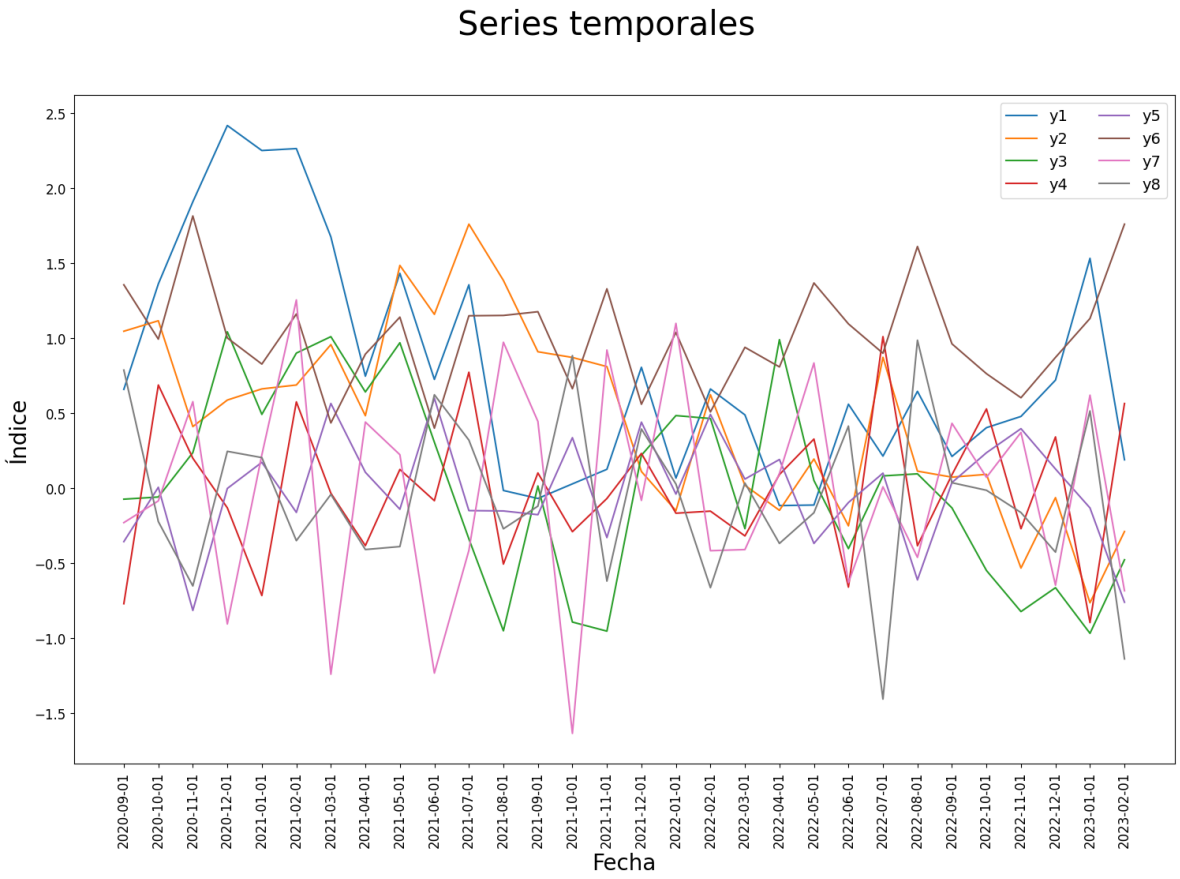
```
datos.head()
```

	y1	y2	y3	y4	y5	y6	y7	y8
2020-09-01	0.659087	1.046467	-0.073204	-0.770190	-0.356159	1.356159	-0.229547	0.787612
2020-10-01	1.361943	1.116432	-0.058704	0.687942	0.006143	0.993857	-0.084578	-0.222777
2020-11-01	1.909412	0.410758	0.237154	0.199685	-0.815080	1.815080	0.576937	-0.651905
2020-12-01	2.418062	0.587727	1.042446	-0.131086	-0.001646	1.001646	-0.905282	0.245830
2021-01-01	2.251217	0.662077	0.491974	-0.715647	0.172409	0.827591	0.222277	0.204014

Representación gráfica de las series variacionales por periodo

Es razonable interesarse por la forma de las series temporales que forman parte del conjunto que se quiere particionar. Así es posible anticipar el resultado, aunque el estudio aporte resultados no esperados. Por ello, se respresentan gráficamente.

```
plt.figure(figsize=(18,11))
x = range(0,datos.shape[0])
labels = [d.date() for d in datos.index]
plt.plot(x,datos)
plt.legend(datos.columns,fontsize = 14,ncol = 2)
plt.xticks(x, labels, rotation='vertical')
plt.tick_params(labelsize = 12)
plt.xlabel('Fecha',fontsize = 20)
plt.ylabel('Índice',fontsize = 20)
plt.suptitle("Series temporales",fontsize=30)
plt.show()
```



Matrices de distancias y disimilaridades

En este punto del desarrollo comienza el foco del problema en torno al que gira todo el trabajo: cuantificar la distancia entre dos series temporales. Como hay distancias que siguen un mismo patrón de código, únicamente se muestran las distancias que necesitan ser programadas con códigos significativamente diferentes.

Distancia entre observaciones

En primer lugar, se define la función que calcula la distancia d_{obs} entre dos series, con posibilidad a especificar los pesos.

```
def dobs(s1,s2,pesos):
  if len(s1) != len(s2):
    print('Método implementado para series de igual longitud.')
    return() # no sigue leyendo código
  l = len(s1)
  SumaPonderada = 0
  for i in range(l): # bucle para calcular la suma
    SumaPonderada += (s1[i]-s2[i])**2*pesos[i]
  res = np.sqrt(SumaPonderada)
  return(res)
```

En segundo lugar, se construye una función que, dado un conjunto de datos y un vector de pesos, calcule la matriz de distancias utilizando la función anterior.

```
def matriz(Datos,pesos):
  n = Datos.shape[1] # número de columnas
  # pesos = np.ones(n)
  M = np.ones((n,n)) # matriz de dimensiones adecuadas
  for i in range(n): # bucle para rellenar la matriz
    s1 = Datos[:,i]
    for j in range(i+1,n):
      s2 = Datos[:,j]
      d = dobs(s1,s2,pesos)
      M[i,j] = d
      M[j,i] = d
  return(M)
```

Por último, se define la matriz de distancias sobre los datos, con un vector de unos como pesos.

```
matdist = matriz(np.array(datos),np.ones(datos.shape[0]))
```

Distancia entre áreas

En este caso, es necesario utilizar una función para calcular integrales. Por ello, se importa la función *quad*. Como sólo se pretende usar una función, no es necesario importar todo el paquete que la contiene.

```
from scipy.integrate import quad
```

Como el objetivo es integrar con límites y sumar los resultados, también es fundamental construir una función capaz de realizar este proceso dadas dos series de datos de igual longitud, donde los datos han sido observados en los mismos instantes.

```
def area_entre_series(x, y):
  if len(x) != len(y):
    print('Método implementado para series de igual longitud.')
    return
  n = len(x)
  area = 0
  for i in range(n-1):
    # definición de la recta que une dos puntos consecutivos de x
    def f(z):
    # Los puntos a unir son: (i,x[i]), (i+1,x[i+1])
      sol = (z-x[i])/(x[i+1]-x[i]) + i
      return(sol)
    # definición de la recta que une dos puntos consecutivos de y
    def g(s):
    # Los puntos a unir son: (i,y[i]), (i+1,y[i+1])
      sol = (s-y[i])/(y[i+1]-y[i]) + i
      return(sol)
    # cálculo integral
    integral, error = quad(lambda t: f(t) - g(t), i, i+1)
    area += integral # añadimos las áreas que se van formando
    return area
```

Por último, se define la matriz de distancias con las dimensiones adecuadas y se rellena mediante un bucle.

```
matdist = np.zeros([len(datos.columns),len(datos.columns)])

for D,M in [[datos,matdist]]:
  for i in range(len(datos.columns)):
    for j in range(i+1,len(datos.columns)):
      dist = area_entre_series(D.iloc[:,i],D.iloc[:,j])
      M[i,j] = dist
      M[j,i] = dist
```

Distancia DTW

La distancia DTW ya está implementada en el módulo *dtaidistance*, por lo que se empleará para el cálculo de dicha distancia entre pares de series. Este módulo no se encuentra en el paquete básico de Jupyter, por lo que es necesario instalarlo. Además, como sólo se usará la función *dtw*, no se importará el módulo completo.

```
!pip install dtaidistance --quiet
from dtaidistance import dtw
```

La función *dtw* acepta como primer atributo un array. Además, para que funcione como se pretende, es necesario trasponer los datos.

Compact = False es para tener una matriz de distancias simétrica como se ha hecho con las otras deistancias, en vez de una matriz triangular superior.

```
matdist = dtw.distance_matrix_fast(np.array(datos.transpose()),
                                   compact = False)
```

Distancia entre correlogramas

Las funciones de autocovarianzas y autocorrelaciones, así como el correlograma, están disponibles en la librería *statsmodels*.

```
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.tsa.stattools import acf, acovf
```

Se define la matriz vacía con dimensiones adecuadas y se utiliza un bucle para rellenarla utilizando la distancia entre correlogramas.

```
matdist = np.zeros([len(datos.columns),len(datos.columns)])

for dat,mat in [[datos,matdist]]:
  for i in range(len(datos.columns)):
    # Calcular la función de autocorrelaciones para la serie i
    acr_i = acf(dat.iloc[:,i], adjusted=False,
                nlags=int(len(dat.iloc[:,i])/4),
                fft=False, bartlett_confint=False, missing='none')
    for j in range(i+1,len(datos.columns)):
      # Calcular la función de autocorrelaciones para la serie j
      acr_j = acf(dat.iloc[:,j], adjusted=False,
                  nlags=int(len(dat.iloc[:,j])/4),
                  fft=False, bartlett_confint=False, missing='none')
      # Calcular la distancia entre las autocorrelaciones
      mat[i,j] = np.sqrt(sum(abs(acr_i)-abs(acr_j))**2)
      mat[j,i] = mat[i,j]
```

Basada en los coeficientes del modelo RLM

El modelo de regresión lineal múltiple se puede ajustar usando la librería *sklearn*, y las medidas relacionadas con dicho modelo, también. Es necesario importar todas estas funciones.

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import  mean_squared_error, r2_score
```

Se guardarán las medidas MSE y R^2 en listas para visualizar la bondad de los modelos.

```
MSE = []
R2 = []
```

Así mismo, los coeficientes de cada modelo se guardarán en una lista para poder compararlos con facilidad.

```
modelos = []
```

Para ajustar el modelo, lo primero es configurar el tipo de modelo de interés. En este caso, modelo de regresion lineal múltiple con coeficiente independiente.

```
lm = LinearRegression(fit_intercept=True)
```

Se calculan todos los coeficientes y medidas de interés mediante bucles.

```
for d in [datos]:
  # necesario para no modificar el conjunto de datos inicial
  datos_P = d.copy()
  # Defnición de las variables dummy, que representan el mes
  #  (11 variables dummy), y la variable tiempo.
  # Se añaden estas variables al dataset inicial mediante un bucle
  for i in range(1,13):
    name = "dummy"+str(i)
    datos_P[name] = [int(date.month == i) for date in datos_P.index]
  datos_P["tiempo"] = [i for i in range(datos_P.shape[0])]

  # Regresión lineal
  for Y in datos_P.columns[range(len(datos.columns))]:
    data = datos_P[[Y,"dummy1","dummy2","dummy3","dummy4","dummy5",
                    "dummy6","dummy7","dummy8","dummy9","dummy10",
                    "dummy11","dummy12","tiempo"]]
    # Se ajusta el modelo a los datos
    reg = lm.fit(data.drop(columns = [Y]), data[Y])
    modelos.append(reg.coef_)
    # Se predice con el modelo ajustado en los datos para
    # analizar los residuos
    predictions = reg.predict(data.drop(columns = [Y]))
    # Calculamos métricas a partir de las predicciones para comprobar
    # la bondad del modelo
    MSE.append(np.sqrt(mean_squared_error(predictions, data[Y])))
    R2.append(r2_score(predictions, data[Y]))
```

Por último, se define la matriz de distancias y se completa con la distancia entre vectores de coeficientes de modelos.

```
matdist = np.zeros([len(datos.columns),len(datos.columns)])

p = 0
for mat in [matdist]:
  for i in range(len(datos.columns)):
    for j in range(i+1,len(datos.columns)):
      # cálculo de la distancia euclidea entre
      # los vectores de coeficientes
      mat[i,j] = np.linalg.norm(modelos[p+i]-modelos[p+j])
      mat[j,i] = mat[i,j]
  p += 12
```

Basada en los coeficientes del modelo de cadenas de Markov ocultas

La librería *hmmlearn* contiene la función *hmm*, para el ajuste de modelos de cadenas de Markov ocultas (Hidden Markov Models).

```
!pip install hmmlearn --quiet
from hmmlearn import hmm
```

El número de estados ocultos será el número de clusters finales.

```
n_states = 3
```

Se prepara un DataFrame para recoger los resultados.

```
conglomerados = pd.DataFrame({'Conglomerado': range(len(datos.columns)),
                              'Series': datos.columns})

# Se define el tipo de distribución: modelo Gaussiano
model = hmm.GaussianHMM(n_components=n_states)

# Entrenamos el modelo en los datos
model.fit(datos.transpose())

# Predecimos la secuencia de estados ocultos
hidden_states = model.predict(datos.transpose())

# Se guarda el resultado
conglomerados['Conglomerado'] = hidden_states
```

```
conglomerados = conglomerados[['Series','Conglomerado']]
conglomerados
```

	Series	Conglomerado
0	y1	1
1	y2	2
2	y3	2
3	y4	2
4	y5	0
5	y6	1
6	y7	2
7	y8	0

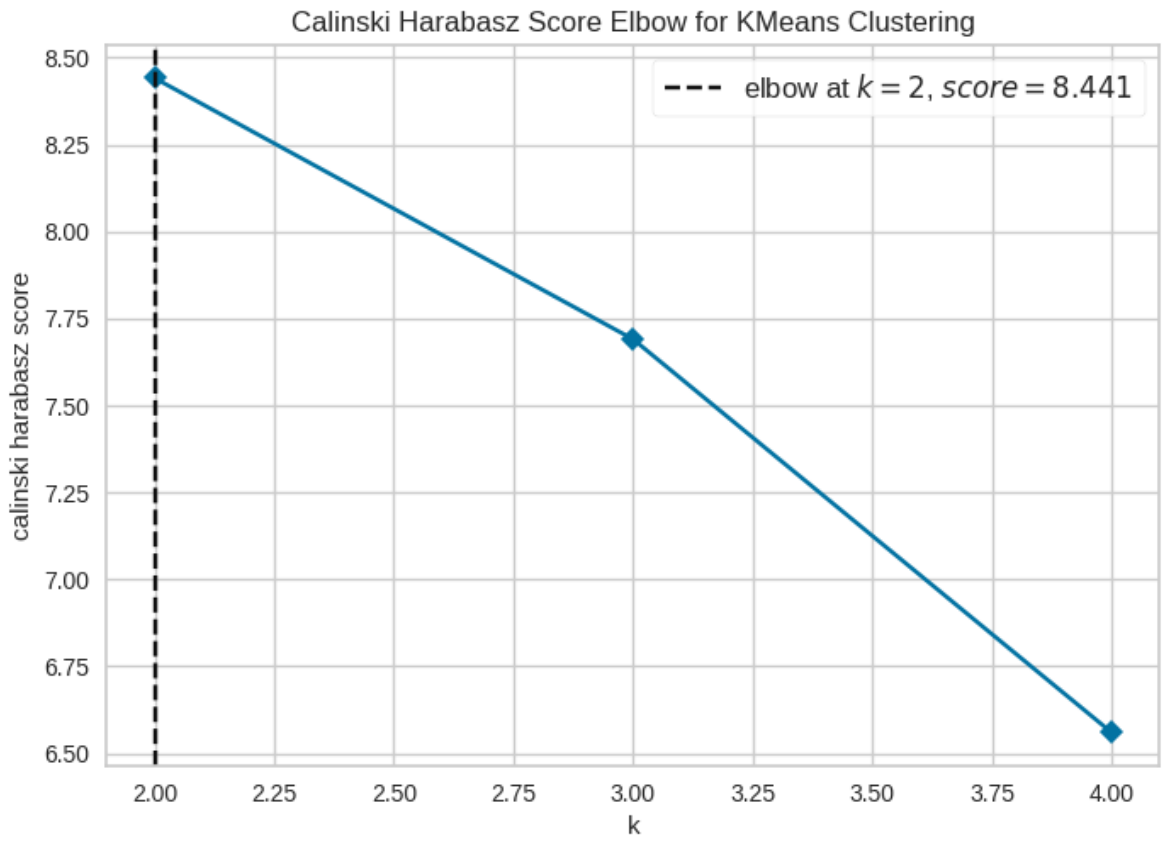
Clustering

Selección del número de conglomerados

Para seleccionar el número óptimo de clusters, se usará una función ya implementada en la librería *yellowbrick*.

```
from yellowbrick.cluster import KElbowVisualizer
from sklearn.cluster import KMeans
```

```
# Para ignorar los warnings, se usa esta linea de comando
warnings.filterwarnings("ignore")

# Se inicia el modelo y el visualizador para
# después aplicarlo a los datos
model = KMeans()
visualizer = KElbowVisualizer(
    model, k=(2,5), metric='calinski_harabasz',
    timings=False, locate_elbow=True
)

visualizer.fit(datos) # Se ajustan los datos al visualizador
visualizer.show()
```




```
<Axes: title={'center': 'Calinski Harabasz Score Elbow for KMeans Clustering'}, xlabel='
k', ylabel='calinski harabasz score'>
```

Jerárquico aglomerativo

```
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.cluster import AgglomerativeClustering
```

```
model = linkage(matdist, method='average',
                metric='euclidean',
                optimal_ordering=False)
dendograma = dendrogram(model,labels = datos.columns,
                        leaf_rotation = 0,
                        above_threshold_color='#bcbddc',
                        orientation='left')
plt.title('Dendograma',fontsize=20)
plt.xlabel('Distancias',fontsize=15)
plt.tick_params(labelsize = 11)
plt.ylabel('')
```

Escogeremos 3 conglomerados.

```
nclusters = n_states
clust_model = AgglomerativeClustering(n_clusters = nclusters,
                                      affinity = 'precomputed',
                                      linkage = 'average')
clust_model.fit(matdist)

conglomerados = pd.DataFrame({'Series': datos.columns,
                              'Conglomerado': clust_model.labels_})
conglomerados
```

K-means

Lo interesante de esta sección es que debemos pasar las series de distancias al algoritmo en vez de las series originales.

Para utilizar KMeans del módulo sklearn.cluster es necesario tener las observaciones por líneas. ¿Qué conjunto de datos hemos de pasar al algoritmo? Porque está claro que no debemos pasarle el conjunto inicial, si no no tendría sentido diferenciar entre distancias.

Realmente deberíamos pasarle un conjunto de datos que represente las distancias entre las series, y en base a eso iniciar el clustering.

Aún así, hemos de definir mejor este proceso. Si pasamos directamente las matrices de distancias, tampoco llegaremos al objetivo que nos interesa porque cada entrada de la matriz tiene un significado diferente. En cambio, nosotros buscamos un conjunto de datos cuyas filas sean las observaciones y las columnas sean las variables.

Siguiendo este razonamiento, se propone la siguiente solución: tomamos como punto de referencia una de las series, supongamos la primera. A continuación calculamos todas las distancias únicamente respecto de esa serie. Así tendremos bien identificado el conjunto de datos y tendrá sentido hacer KMeans sobre él.

```
from sklearn.cluster import KMeans
```

```
n = len(datos.columns)
vecdist = np.zeros((n,1))

for v,m in [[vecdist,matdist[0,:]]]:
  for i in range(0,n):
    v[i,0] = m[i]

kmeans = KMeans(n_clusters=n_states, random_state=0, n_init="auto").fit(vecdist)
```

Se construye un conjunto de datos que contenga los clusters.

```
conglomerados = pd.DataFrame(
    {'Series': datos.columns,
     'Conglomerado': kmeans.labels_})
conglomerados
```

	Series	Conglomerado
0	y1	2
1	y2	1
2	y3	1
3	y4	1
4	y5	0
5	y6	0
6	y7	1
7	y8	1

Conclusiones

Las conclusiones se realizan una vez seleccionada la distancia, el número de conglomerados y la técnica de clustering, tras ejecutar el código correspondiente. En este caso, se ha elegido la distancia basada en los coeficientes del modelo RLM, tres conglomerados y el método K-Means.

Para evaluar la bondad del resultado, se utiliza la puntuación Davies-Bouldin, que está implementada en la librería *sklearn*.

```
from sklearn.metrics import davies_bouldin_score

davies_bouldin_score(datos.transpose(),
                     conglomerados['Conglomerado'])
```

```
1.9158112618959124
```

No es un resultado especialmente bueno, ya que cuanto más cercano a 0, mejor es la partición en términos de análisis cluster, donde se busca la mayor homogeneidad dentro de los grupos y la mayor heterogeneidad entre conglomerados.

Para ver las series temporales por conglomerados y entender mejor cuál ha sido el proceso de unión de las series, se procede como sigue:

En primer lugar, se crea un array que contenga los nombres de las series temporales para que sea más sencillo acceder a ellas.

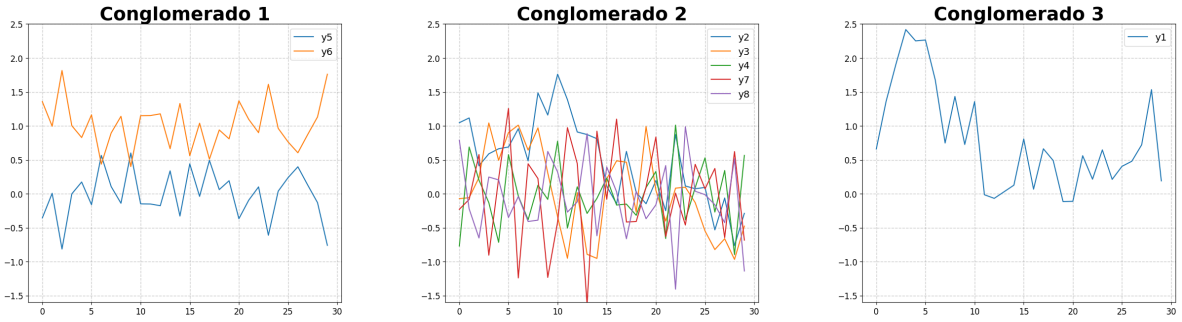
```
columnsPdf = np.array(datos.columns)
```

Se ha de importar la libreía Image para hacer la composición las imágenes de cada cluster.

```
from IPython.display import Image
```

A continuación, se representan en una sóla imagen los tres conglomerados resultados.

```
for datos,cluster,name in [[datos,conglomerados,'Conglomerado']]:

  k = 1
  fichero = 'Conglomerados'

  fig = plt.figure(figsize=(30,17))
  fig.subplots_adjust(hspace=0.33, wspace=0.33)

  for c in range(n_states):
    title = 'Conglomerado '+str(c+1)
    # se agrupan las series que pertenecen al mismo cluster
    idx = [i for i in range(cluster.shape[0]) if cluster[name][i] == c]
    # cong = cluster[name][idx]
    ax = fig.add_subplot(2,3,k)
    x = range(0,datos.shape[0])
    ax.plot(x,datos[datos.columns[idx]])
    ax.legend(columnsPdf[idx],fontsize = 14)
    plt.tick_params(labelsize = 12)
    ax.grid(color='gray', linestyle='dashed', linewidth=1, alpha=0.4)
    ax.set_title(title,size='large',weight='extra bold',
                 stretch='extra-expanded',fontsize=27)
    ax.set_ylim(-1.6,2.5)
    ax.tick_params(labelsize = 12)

    k += 1

  p += 1

  imagen = ax.get_figure()
  ax.plot()
  plt.savefig(fichero, bbox_inches='tight') # se guarda la imagen final

  plt.show()
```




```
# Se restaura la configuración de advertencias
warnings.resetwarnings()
```

Capítulo 4

Caso práctico

Con la finalidad de estudiar el comportamiento de todas las distancias descritas con las técnicas de análisis de conglomerados también presentadas, aplicaremos todo ello a un conjunto de datos temporales: las series del índice de precios del consumo.

Es una buena forma de proceder, sobre todo para mostrar cómo sería la implementación del algoritmo completo. Se ha programado únicamente en Python, mediante los cuadernos de Jupyter, que permiten programar en línea, sin necesidad de descargar la aplicación Spyder.

Ahora bien, los resultados que obtengamos son exclusivos para estos datos y no son determinantes de que un método es mejor que otro. Todo depende de los tipos de datos que se manejen y de la finalidad del estudio.

4.1. El Índice de precios del consumo (IPC)

El Instituto Nacional de Estadística (INE) explica a través de píldoras informativas lo que es el IPC. De éstas, se concluye que:

El IPC es una medida estadística de la evolución mensual de los precios de los bienes y servicios que consume la población residente en viviendas familiares en España. Es una medida que nos ayuda a saber cómo varían los precios de cada producto y en qué momento. Principalmente, sirve como medida de la inflación. Pero también sirve para actualizar los contratos de arrendamiento de inmuebles y como punto de referencia en la negociación salarial.

El conjunto de bienes y servicios, que conforman la cesta de la compra, se obtiene básicamente del consumo de las familias y la importancia de cada uno de ellos en el cálculo del IPC está determinada por dicho consumo. Hay cerca de 500 artículos incluidos

en la cesta de la compra, que se distribuyen en 12 grupos, que le denominan grupos ECOICOP (European Classification of Individual Consumption by Purpose). La tabla 4.1 es una leyenda para identificar los grupos ECOICOP con imágenes que se utilizarán en el código, para que los resultados sean muy visuales y sencillos de entender.

Grupo ECOICOP	Representación en Python
Alimentos y bebidas no alcohólicas	
Bebidas alcohólicas y tabaco	
Vestido y calzado	
Vivienda	
Menaje	
Medicina	
Transporte	
Comunicaciones	
Ocio y cultura	
Enseñanza	
Hoteles, cafés y restaurantes	
Otros bienes y servicios	

Tabla 4.1: Grupos ECOICOP. Imágenes de la plataforma Pixabay.

El cálculo del IPC lo lleva a cabo el INE. Es un índice representativo porque se incluyen en la cesta de la compra los nuevos bienes y servicios que aparecen en el mercado y se eliminan los que son poco significativos. Por ejemplo, tiene sentido que en 2023 entren en la cesta de la compra los reloges inteligentes y no se tengan en cuenta los tocadiscos de vinilos. También es comparable, pues el índice se calcula de la misma forma en todas las regiones y todos los meses.

4.2. Análisis del IPC en Python

Clustering de series IPC por grupos ECOICOP

Análisis por periodos

Este caso práctico ejemplifica la forma de proceder en el análisis de conglomerados con series temporales. Se utilizarán los datos del Índice de Precios de Consumo (IPC) desde 2002 hasta 2023. Se trata de hacer un análisis cluster de series temporales por grupos ECOICOP en varios periodos para estudiar también la evolución de los conglomerados en el tiempo. Otro motivo de separar en periodos es que no conviene que las series sean excesivamente largas.

Los grupos ECOICOP son:

- Alimentos y bebidas no alcohólicas.
- Bebidas alcohólicas y tabaco.
- Vestido y calzado.
- Vivienda.
- Menaje.
- Medicina.
- Transporte.
- Comunicaciones.
- Ocio y cultura.
- Enseñanza.
- Hoteles, cafés y restaurantes.
- Otros bienes y servicios.

El clustering de series temporales está basado fundamentalmente en la definición de distancias entre series. Una vez definidas estas distancias, sólo hay que aplicar el algoritmos de análisis de conglomerados más conveniente.

Los resultados mostrados corresponden a la elección de la distancia DTW y la técnica de clustering jerárquico aglomerativo, con cuatro conglomerados, porque es la combinación que mejores resultados ha mostrado en el Índice de Davies-Bouldin.

Importación de módulos

En primer lugar, se importan las librerías y los módulos principales que servirán a lo largo de toda la práctica. Más adelante, cuando sean necesarias funciones específicas de algún otro módulo, entonces se importarán. Así el proceso queda más claro.

```
# quiet sirve para ignorar los mensajes de instalación
!pip install datetime --quiet
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import datetime
%matplotlib inline
```

Además, para evitar los mensajes *Warning*, que muchas veces son extensos, se compila la siguiente casilla.

```
import warnings
warnings.filterwarnings('ignore')
```

Lectura y preparación de datos

A continuación, se lee la base de datos de las estadísticas IPC a la que se puede acceder a través de la página web del Instituto Nacional de Estadística (https://ine.es/).

Para hacer un estudio más exhaustivo y acertado, y evitando el ruido que puede producir una serie temporal excesivamente larga en la que se observan diferentes comportammientos, se dividen los datos en cuatro periodos:

Periodo 1: Enero 2002 a julio 2008 (antes de la crisis económica de la burbuja inmobiliaria).

Periodo 2: Agosto 2008 a febrero 2013 (la crisis económica).

Periodo 3: Marzo 2013 a febrero 2020 (el período de recuperación de esta crisis hasta que se decretó el estado de alarma por la pandemia).

Periodo 4: Marzo 2020 a agosto 2023 (desde el inicio de la pandemia por COVID-19 hasta enero de 2023).

Para una lectura correcta de las fechas, la función *read_excel* permite especificar los campos que tienen un formato, así como una función para procesar dicho tipo de dato.

La siguiente función transforma una cadena de formato "%YM%m" en una fecha.

```
# Función que transforma una cadena de formato "%YM%m", en una fecha
def f(string):
  d = datetime.datetime.strptime(string,"%YM%m").date()
  return(d)
```

Se define el nombre de la base de datos que se va a usar.

```
file = 'IPC_database.xls'
```

En la base de datos se ha incluido una variable que indica el periodo al que pertenece cada observación, y se leerá por separado porque simplemente es una variable auxiliar que permite organizar los datos.

```
periodos = pd.read_excel(io = file,sheet_name = 'Datos',
                         header = 0,usecols="B")
```

Para indicar las filas que han de ser leídas en cada periodo, la función pd.read_excel tiene un argumento el el que se puede especificar las filas que no hay que incluir. Por ello, se crean los siguientes conjuntos que indican las observaciones a desechar en cada periodo.

```
skipP1 = [i+1 for i in periodos.loc[periodos.iloc[:,0] != 1].index]
skipP2 = [i+1 for i in periodos.loc[periodos.iloc[:,0] != 2].index]
skipP3 = [i+1 for i in periodos.loc[periodos.iloc[:,0] != 3].index]
skipP4 = [i+1 for i in periodos.loc[periodos.iloc[:,0] != 4].index]
```

A continuación, se realiza la lectura de los datos de cada periodo especificando todos los argumentos convenientes.

```
datos_P1 = pd.read_excel(io = file,sheet_name = 'Datos',header = 0,
                         usecols="A,C:N", parse_dates=["Fecha"],
                         date_parser=f,skiprows=[i for i in skipP1])
datos_P2 = pd.read_excel(io = file,sheet_name = 'Datos',header = 0,
                         usecols="A,C:N", parse_dates=["Fecha"],
                         date_parser=f,skiprows=[i for i in skipP2])
datos_P3 = pd.read_excel(io = file,sheet_name = 'Datos',header = 0,
                         usecols="A,C:N", parse_dates=["Fecha"],
                         date_parser=f,skiprows=[i for i in skipP3])
datos_P4 = pd.read_excel(io = file,sheet_name = 'Datos',header = 0,
                         usecols="A,C:N", parse_dates=["Fecha"],
                         date_parser=f,skiprows=[i for i in skipP4])
```

Como la primera columna indica la fecha en que ha sido observado cada dato, se puede utilizar como nombre de la fila, de modo que no se confunda con un grupo ECOICOP. Una forma de hacerlo es la siguiente.

```
datos_P1.rename(index=datos_P1["Fecha"],inplace=True)
datos_P1 = datos_P1.drop(['Fecha'], axis=1)
datos_P2.rename(index=datos_P2["Fecha"],inplace=True)
datos_P2 = datos_P2.drop(['Fecha'], axis=1)
datos_P3.rename(index=datos_P3["Fecha"],inplace=True)
datos_P3 = datos_P3.drop(['Fecha'], axis=1)
datos_P4.rename(index=datos_P4["Fecha"],inplace=True)
datos_P4 = datos_P4.drop(['Fecha'], axis=1)
```

Para tener un acceso sencillo a los nombres de los grupos, también se hará la siguiente lectura de los grupos ECOICOP, que se encuentran en la base de datos, y se transformará a formato Lista. Se recomienda ver el Anexo para visualizar mejor el proceso.

```
# Lectura de los grupos ECOICOP, que están guardados en la base de datos
ECOICOP = pd.read_excel(io = file,sheet_name = 'Metadatos',header = 0,usecols="B")
# Se toman únicamente los grupos ECOICOP
ECOICOP = ECOICOP[2:14]
# Se guarda en formato lista
ECOICOP = [ecoicop[0] for ecoicop in ECOICOP.to_numpy()]
ECOICOP
```

```
['Alimentos y bebidas no alcohólicas',
 'Bebidas alcohólicas y tabaco',
 'Vestido y calzado',
 'Vivienda, agua, electricidad, gas y otros combustibles',
 'Muebles, artículos del hogar y artículos para el mantenimiento corriente del hogar',
 'Sanidad',
 'Transporte',
 'Comunicaciones',
 'Ocio y cultura',
 'Enseñanza',
 'Restaurantes y hoteles',
 'Otros bienes y servicios ']
```

Es conveniente comprobar que todo avanza tal y como se ha previsto.

```
datos_P1.head()
```

	IPC_01	IPC_02	IPC_03	IPC_04	IPC_05	IPC_06	IPC_07	IPC_08	IPC_09	IPC_10	IPC_11	II
2002-01-01	65.859	46.340	82.717	58.763	81.364	86.313	63.805	128.593	101.553	58.352	63.523	6
2002-02-01	65.836	46.348	81.782	58.888	81.475	86.844	64.089	126.667	101.580	58.430	64.123	6
2002-03-01	66.185	46.436	83.519	59.041	81.819	87.210	65.007	125.098	103.157	58.496	64.756	6
2002-04-01	66.614	48.321	89.456	59.223	82.336	87.665	66.018	125.098	102.658	58.556	65.182	6
2002-05-01	66.902	48.446	90.427	59.324	82.686	87.212	66.191	125.098	102.968	58.563	65.416	6

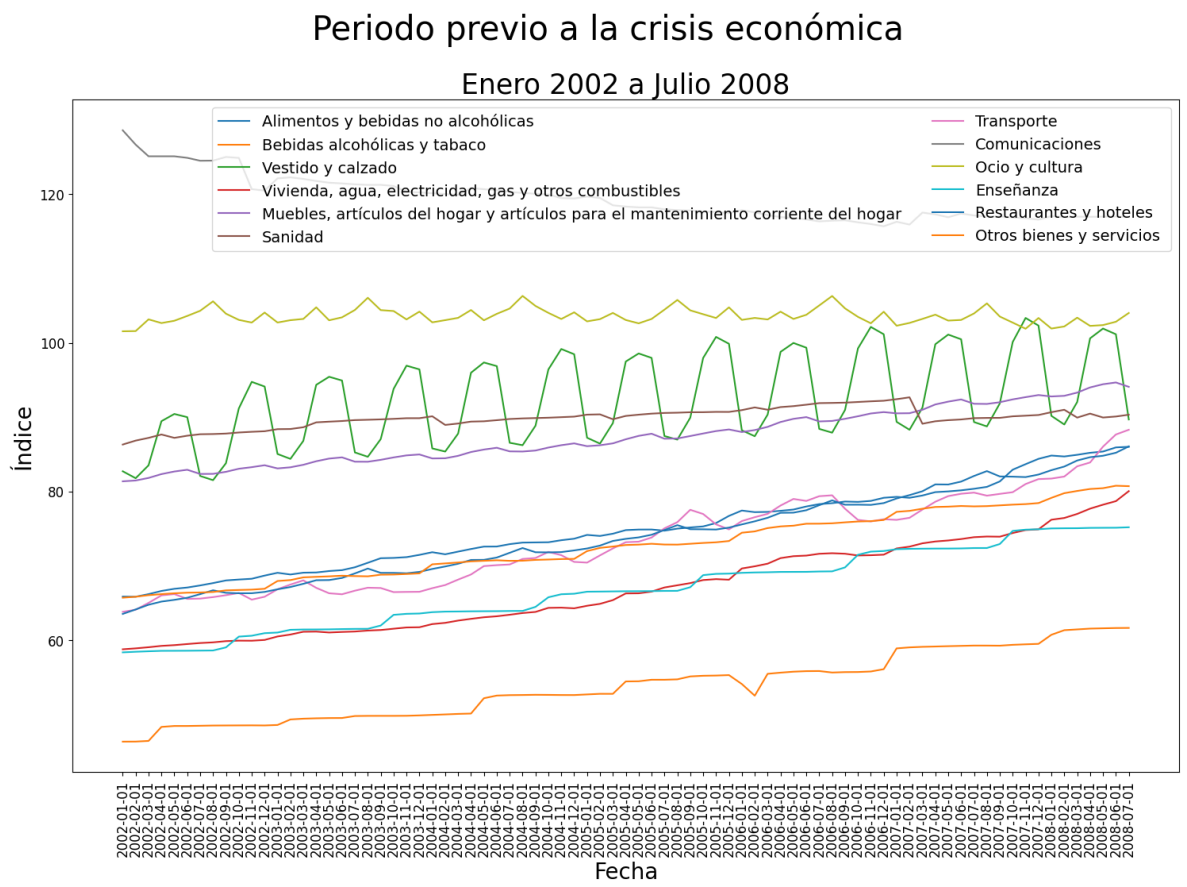
Además, se cargan las imágenes que identificarán a los grupos ECOICOP, como se ha mostrado en la tabla 4.1, y se guardan en el formato adecuado para usarlas posteriormente.

```
from PIL import Image
i1 = Image.open("/content/apple.png")
i1.save("/content/apple.png")
i2 = Image.open("/content/beer.png")
i2.save("/content/beer.png")
i3 = Image.open("/content/chaqueta.png")
i3.save("/content/chaqueta.png")
i4 = Image.open("/content/house.png")
i4.save("/content/house.png")
i5 = Image.open("/content/closet.png")
i5.save("/content/closet.png")
i6 = Image.open("/content/medical.png")
i6.save("/content/medical.png")
i7 = Image.open("/content/car.png")
i7.save("/content/car.png")
i8 = Image.open("/content/antena.png")
i8.save("/content/antena.png")
i9 = Image.open("/content/basketball.png")
i9.save("/content/basketball.png")
i10 = Image.open("/content/books.png")
i10.save("/content/books.png")
i11 = Image.open("/content/chef.png")
i11.save("/content/chef.png")
i12 = Image.open("/content/suma.png")
i12.save("/content/suma.png")
```

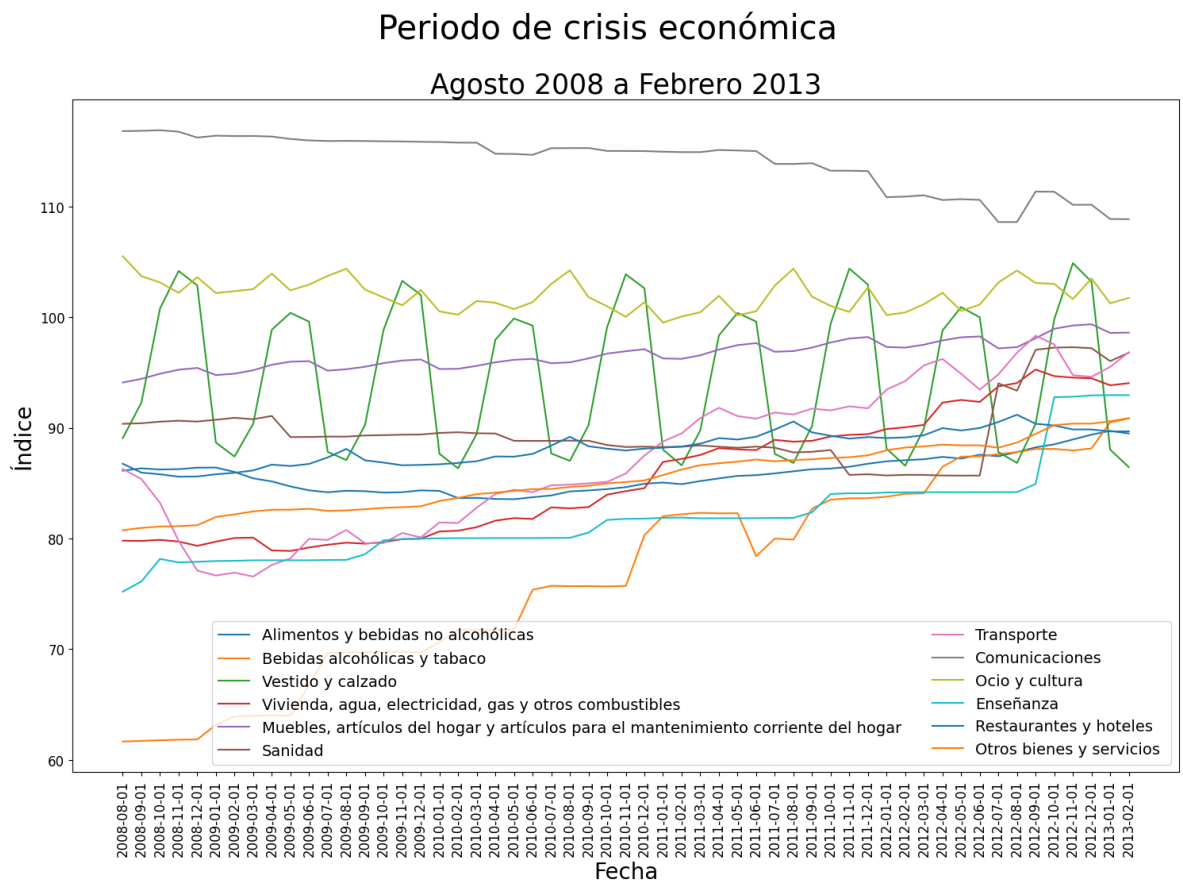
Representación gráfica de las series variacionales por periodo

Para proceder, es razonable tener una idea de la forma de las series temporales que forman parte del conjunto que se quiere particionar. La principal finalidad es intentar prever cuál será el resultado final, pero conviene proceder con el anális porque probablemente haya algunos resultados no esperados interesantes.

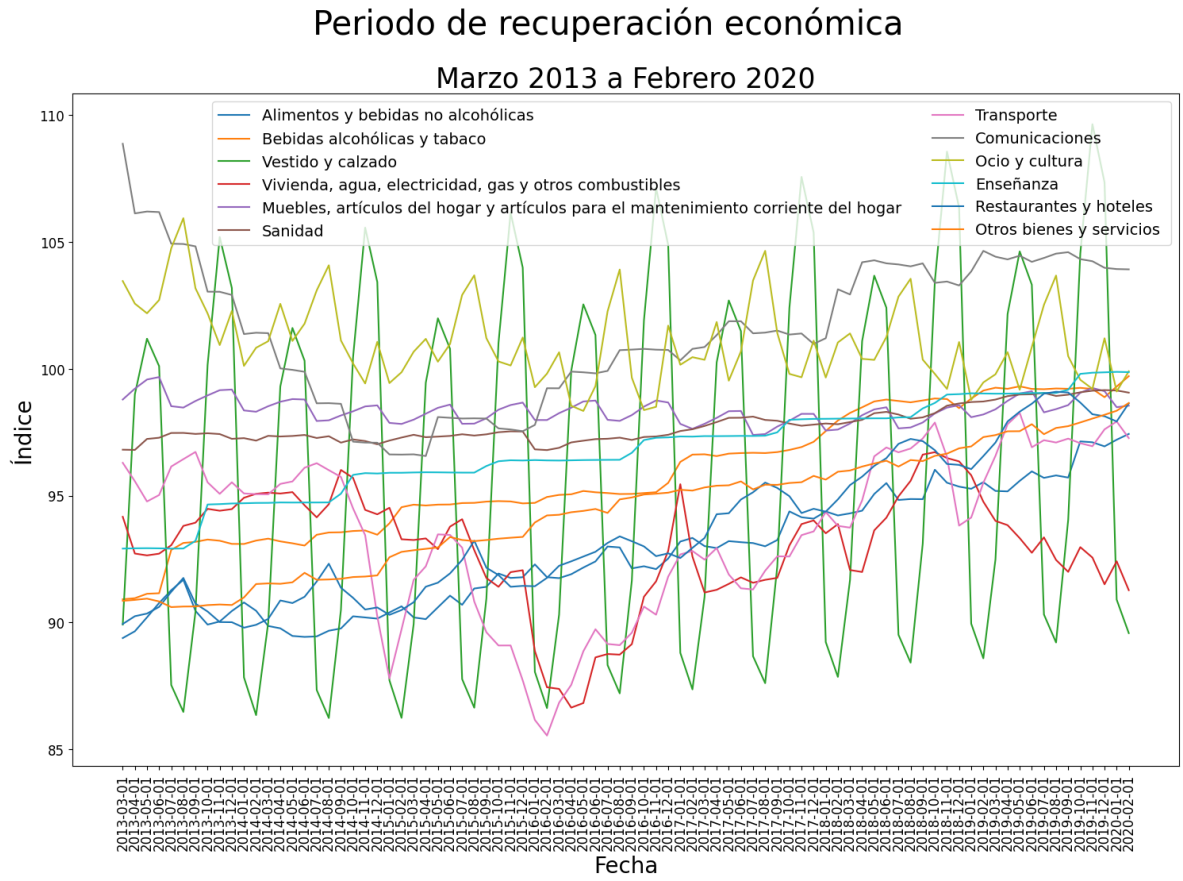
```
plt.figure(figsize=(18,11))
x = range(0,datos_P1.shape[0])
labels = [d.date() for d in datos_P1.index]
plt.plot(x,datos_P1)
plt.legend(ECOICOP,fontsize = 14,ncol = 2)
plt.xticks(x, labels, rotation='vertical')
plt.tick_params(labelsize = 12)
plt.xlabel('Fecha',fontsize = 20)
plt.ylabel('Índice',fontsize = 20)
plt.title("Enero 2002 a Julio 2008",fontsize = 25)
plt.suptitle("Periodo previo a la crisis económica",fontsize = 30)
plt.show()
```



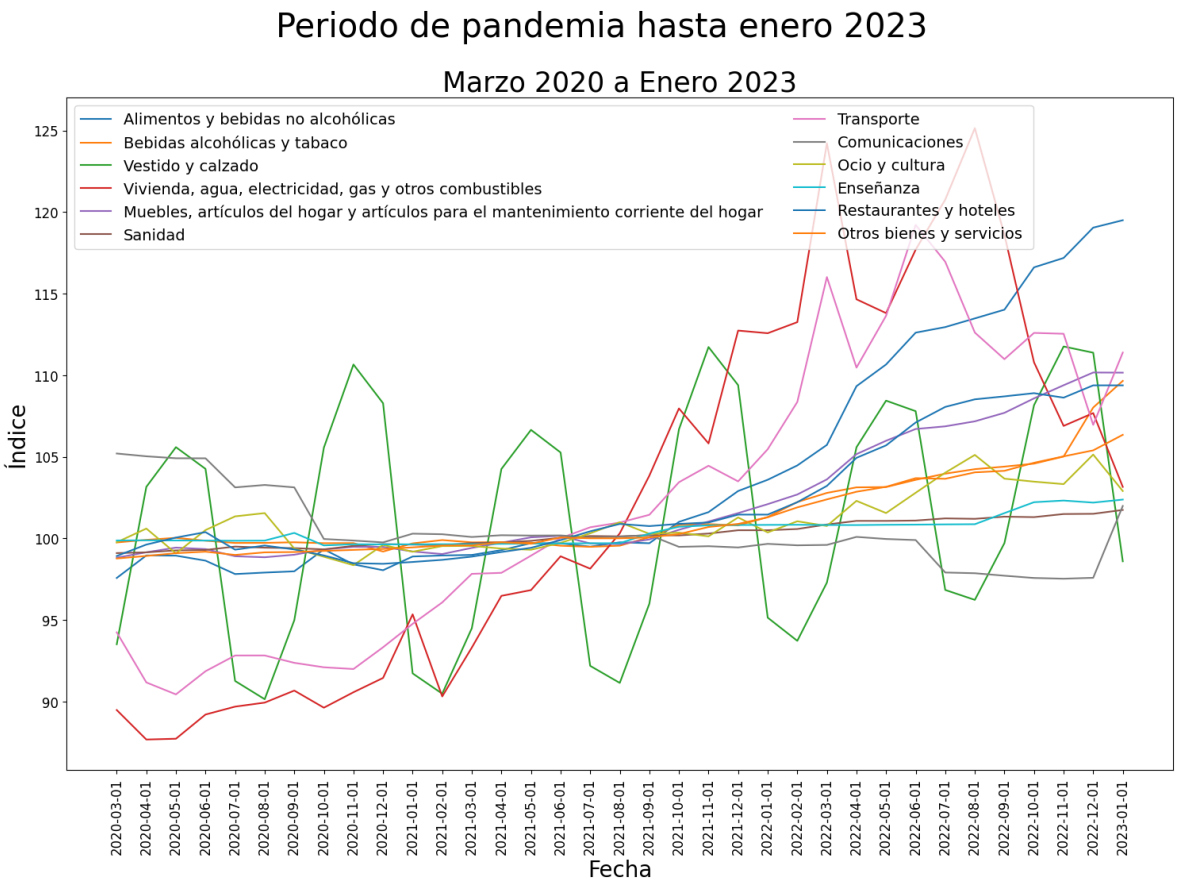

```
plt.figure(figsize=(18,11))
x = range(0,datos_P2.shape[0])
labels = [d.date() for d in datos_P2.index]
plt.plot(x,datos_P2)
plt.legend(ECOICOP,fontsize = 14,ncol = 2)
plt.xticks(x, labels, rotation='vertical')
plt.tick_params(labelsize = 12)
plt.xlabel('Fecha',fontsize = 20)
plt.ylabel('Índice',fontsize = 20)
plt.title("Agosto 2008 a Febrero 2013",fontsize=25)
plt.suptitle("Periodo de crisis económica",fontsize=30)
plt.show()
```




```
plt.figure(figsize=(18,11))
x = range(0,datos_P3.shape[0])
labels = [d.date() for d in datos_P3.index]
plt.plot(x,datos_P3)
plt.legend(ECOICOP,fontsize = 14,ncol = 2)
plt.xticks(x, labels, rotation='vertical')
plt.tick_params(labelsize = 12)
plt.xlabel('Fecha',fontsize = 20)
plt.ylabel('Índice',fontsize = 20)
plt.title("Marzo 2013 a Febrero 2020",fontsize=25)
plt.suptitle("Periodo de recuperación económica",fontsize=30)
plt.show()
```




```
plt.figure(figsize=(18,11))
x = range(0,datos_P4.shape[0])
labels = [d.date() for d in datos_P4.index]
plt.plot(x,datos_P4)
plt.legend(ECOICOP,fontsize = 14,ncol = 2)
plt.xticks(x, labels, rotation='vertical')
plt.tick_params(labelsize = 12)
plt.xlabel('Fecha',fontsize = 20)
plt.ylabel('Índice',fontsize = 20)
plt.title("Marzo 2020 a Enero 2023",fontsize=25)
plt.suptitle("Periodo de pandemia hasta enero 2023",fontsize=30)
plt.show()
```



Matrices de distancias y disimilaridades

Esta sección que comienza del desarrollo del análisis, empieza la parte central del problema a resolver, en torno al que gira todo el trabajo: cuantificar la distancia entre dos series temporales. Se usará el código presentado en el capítulo anterior adaptado a este conjunto de datos.

Distancia entre observaciones

En primer lugar, se define la función que calcula la distancia d_{obs} entre dos series, con posibilidad a especificar los pesos.

```
def dobs(s1,s2,pesos):
  if len(s1) != len(s2):
    print('Método implementado para series de igual longitud.')
    return()
  l = len(s1)
  SumaPonderada = 0
  for i in range(l):
    SumaPonderada += (s1[i]-s2[i])**2*pesos[i]
  res = np.sqrt(SumaPonderada)
  return(res)
```

En segundo lugar, se construye una función que, dado un conjunto de datos y un vector de pesos, calcule la matriz de distancias utilizando la función anterior.

```
def matriz(Datos,pesos):
  n = Datos.shape[1]
  M = np.ones((n,n))
  for i in range(n):
    s1 = Datos[:,i]
    for j in range(i+1,n):
      s2 = Datos[:,j]
      d = dobs(s1,s2,pesos)
      M[i,j] = d
      M[j,i] = d
  return(M)
```

Por último, se definen las matrices de distancias sobre los datos de esta práctica, con un vector de unos como pesos.

```
matdist_P1 = matriz(np.array(datos_P1),np.ones(datos_P1.shape[0]))
matdist_P2 = matriz(np.array(datos_P2),np.ones(datos_P2.shape[0]))
matdist_P3 = matriz(np.array(datos_P3),np.ones(datos_P3.shape[0]))
matdist_P4 = matriz(np.array(datos_P4),np.ones(datos_P4.shape[0]))
```

Distancia entre velocidades

Aplicando los mismos pasos que para d_{obs}, se calculan las matrices de distancias con la distancia d_{vel}. Será necesaria también una función para definir qué se entiende por *velocidad*.

Las series tienen que tener al menor 2 observaciones y deben ser series de igual longitud.

```
# Función para calcular la 'Velocidad de una serie'
def vel(serie):
  velocidad = []
  # Python empieza a contar en 0
  for i in range(1,len(serie)):
    v = serie[i] - serie[i-1]
    velocidad.append(v)
  return(velocidad)

# Función 'Distancia entre dos series',
# con posibilidad a especificar los pesos
def dvel(s1,s2,pesos):
  if len(s1) != len(s2):
    print('Método implementado para series de igual longitud.')
    return()
  l = len(s1)
  v1 = vel(s1)
  v2 = vel(s2)
  SumaPonderada = 0
  for i in range(l-1):
    SumaPonderada += (v1[i]-v2[i])**2*pesos[i]
  res = np.sqrt(SumaPonderada)
  return(res)

# Función para definir la matriz de distancias
def matriz(Datos,pesos):
  n = Datos.shape[1]
  M = np.ones((n,n))
  for i in range(n):
    s1 = Datos[:,i]
    for j in range(i+1,n):
      s2 = Datos[:,j]
      d = dvel(s1,s2,pesos)
      M[i,j] = d
      M[j,i] = d
  return(M)

# Definición de las matrices de distancias para el conjunto de datos escogido
matdist_P1 = matriz(np.array(datos_P1),np.ones(datos_P1.shape[0]))
matdist_P2 = matriz(np.array(datos_P2),np.ones(datos_P2.shape[0]))
matdist_P3 = matriz(np.array(datos_P3),np.ones(datos_P3.shape[0]))
matdist_P4 = matriz(np.array(datos_P4),np.ones(datos_P4.shape[0]))
```

Distancia entre aceleraciones

Aplicando los mismos pasos que para d_{vel}, se calculan las matrices de distancias con la distancia d_{acc}. Será necesaria también una función para definir qué se entiende por *aceleración*.

Las series tienen que tener al menor 3 observaciones y deben ser series de igual longitud.

```
# Función para calcular la 'Aceleración de una serie'
def acc(serie):
  aceleracion = []
  # Python empieza a contar en 0
  for i in range(2,len(serie)):
    a = serie[i] - 2*serie[i-1] + serie[i-2]
    aceleracion.append(a)
  return(aceleracion)

# Función 'Distancia entre dos series',
# con posibilidad a especificar los pesos
def dacc(s1,s2,pesos):
  if len(s1) != len(s2):
    print('Método implementado para series de igual longitud.')
    return()
  l = len(s1)
  a1 = acc(s1)
  a2 = acc(s2)
  SumaPonderada = 0
  for i in range(l-2):
    SumaPonderada += (a1[i]-a2[i])**2*pesos[i]
  res = np.sqrt(SumaPonderada)
  return(res)

# Función para definir la matriz de distancias
def matriz(Datos,pesos):
  n = Datos.shape[1]
  M = np.ones((n,n))
  for i in range(n):
    s1 = Datos[:,i]
    for j in range(i+1,n):
      s2 = Datos[:,j]
      d = dacc(s1,s2,pesos)
      M[i,j] = d
      M[j,i] = d
  return(M)

# Definición de las matrices de distancias
# para el conjunto de datos escogido
matdist_P1 = matriz(np.array(datos_P1),np.ones(datos_P1.shape[0]))
matdist_P2 = matriz(np.array(datos_P2),np.ones(datos_P2.shape[0]))
matdist_P3 = matriz(np.array(datos_P3),np.ones(datos_P3.shape[0]))
matdist_P4 = matriz(np.array(datos_P4),np.ones(datos_P4.shape[0]))
```

Distancia entre áreas

Las áreas serán calculadas a través de integrales, por ello es necesario importar una librería que tiene implementada la función que calcula integrales, quad.

```
from scipy.integrate import quad
```

La series son de la misma longitud y los datos deben haber sido observados en los mismos instantes.

Se define una función para calcular el área entre dos series de estas características. Una forma de hacerlo es iterativamente, a trozos, suponiendo que la serie está bien definida si los puntos se unen utilizando rectas.

```
def area_entre_series(x, y):
  if len(x) != len(y):
    print('Método implementado para series de igual longitud.')
    return
  n = len(x)
  area = 0
  for i in range(n-1):
    def f(z):
      # Los puntos a unir son: (i,x[i]), (i+1,x[i+1])
      sol = (z-x[i])/(x[i+1]-x[i]) + i
      return(sol)
    def g(s):
      # Los puntos a unir son: (i,y[i]), (i+1,y[i+1])
      sol = (s-y[i])/(y[i+1]-y[i]) + i
      return(sol)
    integral, error = quad(lambda t: f(t) - g(t), i, i+1)
    area += integral
    return area
```

Por último, se definen matrices vacías de las dimensiones adecuadas y se utiliza un bucle para rellenarlas utilizando la distancia area entre series.

```
matdist_P1 = np.zeros([len(ECOICOP),len(ECOICOP)])
matdist_P2 = np.zeros([len(ECOICOP),len(ECOICOP)])
matdist_P3 = np.zeros([len(ECOICOP),len(ECOICOP)])
matdist_P4 = np.zeros([len(ECOICOP),len(ECOICOP)])

for D,M in [
    [datos_P1,matdist_P1],
     [datos_P2,matdist_P2],
      [datos_P3,matdist_P3],
       [datos_P4,matdist_P4]]:
  for i in range(len(ECOICOP)):
    for j in range(i+1,len(ECOICOP)):
      dist = area_entre_series(D.iloc[:,i],D.iloc[:,j])
      M[i,j] = dist
      M[j,i] = dist
```

Distancia DTW

La distancia DTW ya está implementada en el módulo *dtaidistance*, por lo que se empleará para el cálculo de la distancia entre series.

```
!pip install dtaidistance --quiet
from dtaidistance import dtw
```

Para utilizar esta función correctamente con el conjunto de datos que se ha definido, se requiere trasponer la matriz de datos y pasarla a formato array.

Además, se utiliza el argumento compact = False para obtener un matriz simétrica, en vez de triangular superior.

```
matdist_P1 = dtw.distance_matrix_fast(
    np.array(datos_P1.transpose()),
    compact = False)
matdist_P2 = dtw.distance_matrix_fast(
    np.array(datos_P2.transpose()),
    compact = False)
matdist_P3 = dtw.distance_matrix_fast(
    np.array(datos_P3.transpose()),
    compact = False)
matdist_P4 = dtw.distance_matrix_fast(
    np.array(datos_P4.transpose()),
    compact = False)
```

Distancia entre FACV

Las funciones de autocovarianzas y autocorrelaciones, así como el correlograma, están disponibles en la librería statsmodels, por lo que es necesario importarlas para usarlas.

```
from statsmodels.tsa.stattools import acovf
```

Se definen matrices vacías de las dimensiones adecuadas y se utiliza un bucle para rellenarlas utilizando la distancia entre FACV.

```
matdist_P1 = np.zeros([len(ECOICOP),len(ECOICOP)])
matdist_P2 = np.zeros([len(ECOICOP),len(ECOICOP)])
matdist_P3 = np.zeros([len(ECOICOP),len(ECOICOP)])
matdist_P4 = np.zeros([len(ECOICOP),len(ECOICOP)])

for dat,mat in [
    [datos_P1,matdist_P1],
     [datos_P2,matdist_P2],
      [datos_P3,matdist_P3],
       [datos_P4,matdist_P4]]:
  for i in range(len(ECOICOP)):
    # Calcular la función de autocovarianzas
    acvs_i = acovf(dat.iloc[:,i])
    for j in range(i+1,len(ECOICOP)):
      acvs_j = acovf(dat.iloc[:,j])
      # como matriz pesos se toma la identidad
      mat[i,j] = np.sqrt(sum((acvs_i-acvs_j)**2))
      mat[j,i] = mat[i,j]
```

Distancia entre FAC

Este caso es análogo al anterior, pero la función a usar es *acf*, que también ha de ser importada.

```
from statsmodels.tsa.stattools import acf
```

Se definen matrices vacías de las dimensiones adecuadas y se utiliza un bucle para rellenarlas utilizando la distancia entre FACV.

```
matdist_P1 = np.zeros([len(ECOICOP),len(ECOICOP)])
matdist_P2 = np.zeros([len(ECOICOP),len(ECOICOP)])
matdist_P3 = np.zeros([len(ECOICOP),len(ECOICOP)])
matdist_P4 = np.zeros([len(ECOICOP),len(ECOICOP)])

for dat,mat in [
    [datos_P1,matdist_P1],
     [datos_P2,matdist_P2],
      [datos_P3,matdist_P3],
       [datos_P4,matdist_P4]]:
  for i in range(len(ECOICOP)):
    # Calcular la función de autocorrelaciones
    acr_i = acf(dat.iloc[:,i], adjusted=False,
                nlags=int(len(dat.iloc[:,i])/4), fft=False,
                bartlett_confint=False, missing='none')
    for j in range(i+1,len(ECOICOP)):
      acr_j = acf(dat.iloc[:,j], adjusted=False,
                  nlags=int(len(dat.iloc[:,j])/4), fft=False,
                  bartlett_confint=False, missing='none')
      # como matriz pesos se elige la identidad
      mat[i,j] = np.sqrt(sum((acr_i-acr_j)**2))
      mat[j,i] = mat[i,j]
```

Distancia entre correlogramas

Igual que en el caso anterior, se utilizará la función *acf*, pero el cálculo de la distancia es diferente.

```
matdist_P1 = np.zeros([len(ECOICOP),len(ECOICOP)])
matdist_P2 = np.zeros([len(ECOICOP),len(ECOICOP)])
matdist_P3 = np.zeros([len(ECOICOP),len(ECOICOP)])
matdist_P4 = np.zeros([len(ECOICOP),len(ECOICOP)])

# Se rellenan las matrices de distancias
for dat,mat in [
    [datos_P1,matdist_P1],
     [datos_P2,matdist_P2],
      [datos_P3,matdist_P3],
       [datos_P4,matdist_P4]]:
  for i in range(len(ECOICOP)):
    # Calcular la función de autocorrelaciones
    acr_i = acf(dat.iloc[:,i], adjusted=False,
                nlags=int(len(dat.iloc[:,i])/4), fft=False,
                bartlett_confint=False, missing='none')
    for j in range(i+1,len(ECOICOP)):
      acr_j = acf(dat.iloc[:,j], adjusted=False,
                  nlags=int(len(dat.iloc[:,j])/4), fft=False,
                  bartlett_confint=False, missing='none')
      mat[i,j] = np.sqrt(sum(abs(acr_i)-abs(acr_j))**2)
      mat[j,i] = mat[i,j]
```

Basada en los coeficientes del modelo RLM

El modelo de regresión lineal múltiple se puede ajustar usando la librería *sklearn*, y las medidas relacionadas con dicho modelo, también. Para utilizarlas, es necesario importarlas.

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import  mean_squared_error, explained_variance_score
```

Para recopilar las medidas MSE y R^2, se crean listas vacías que se irán rellenando mediante un bucle. El objetivo de esto es visualizar la bondad de los modelos contruidos.

```
MSE = []
R2 = []
```

También es necesario guardar los coeficientes de los modelos ajustados para compararlos posteriormente.

```
modelos = []
```

Para ajustar el modelo, lo primero es configurar el tipo de modelo de interés. En este caso, modelo de regresion lineal múltiple con coeficiente independiente.

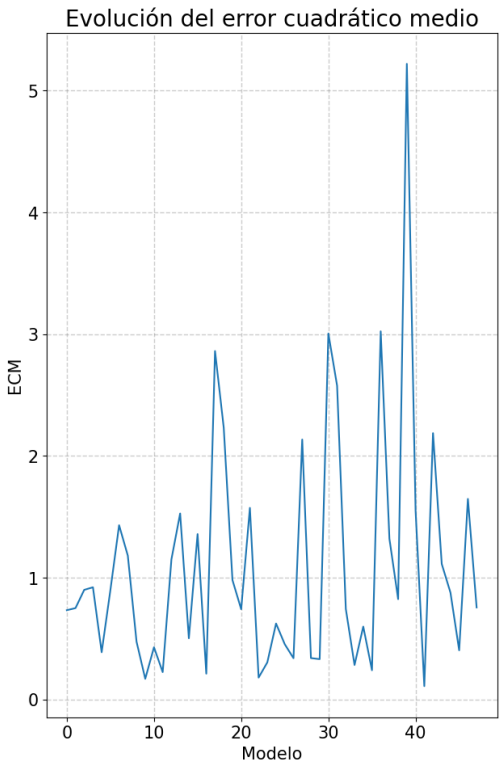
```
lm = LinearRegression(fit_intercept=True)
```

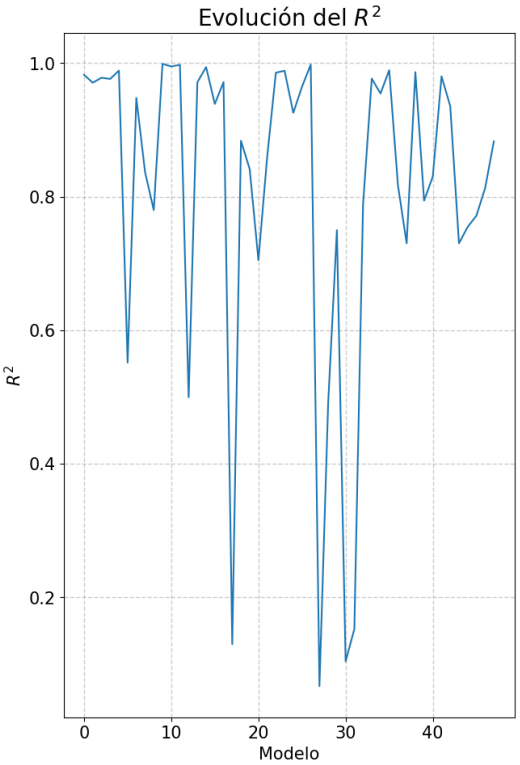
Por último, para cada periodo se calculan todos los coeficiente y medidas interesantes mediante un bucle.

```
for d in [datos_P1,datos_P2,datos_P3,datos_P4]:
  # Como se va a transformar el conjunto de datos,
  # para que no afecte al cuaderno entero se hará una
  # copia independiente del mismo
  datos_P = d.copy()
  # Definición de las variables dummy, que representan el mes
  # (11 variables dummy), y la variable tiempo.
  # Se añaden estas variables al dataset inicial mediante un bucle
  for i in range(1,13):
    name = "dummy"+str(i)
    datos_P[name] = [int(date.month == i) for date in datos_P.index]
  datos_P["tiempo"] = [i for i in range(datos_P.shape[0])]

  # Regresión lineal
  for Y in datos_P.columns[range(len(ECOICOP))]:
    datos = datos_P[[Y,"dummy1","dummy2","dummy3","dummy4","dummy5",
                     "dummy6","dummy7","dummy8","dummy9","dummy10",
                     "dummy11","dummy12","tiempo"]]
    # Se ajusta el modelo a los datos
    reg = lm.fit(datos.drop(columns = [Y]), datos[Y])
    modelos.append(reg.coef_)
    # Se predice con el modelo ajustado en los datos
    # para analizar los residuos
    predictions = reg.predict(datos.drop(columns = [Y]))
    # Cálculo de métricas a partir de las predicciones
    # para comprobar la bondad del modelo
    MSE.append(np.sqrt(mean_squared_error(predictions, datos[Y])))
    R2.append(explained_variance_score(datos[Y],predictions))
```

Para estudiar el comportamiento de los modelos ajustados, se representan la evolución del error cuadrático medio para cada modelo y la evolución del R^2.

```
fig1 = plt.figure(figsize=(18,11))
fig1.subplots_adjust(hspace=0.5, wspace=0.5)

ax = fig1.add_subplot(1,2,1)
ax.plot(MSE)
ax.set_title('Evolución del error cuadrático medio',size = 20)
ax.set_xlabel('Modelo',size = 15)
ax.set_ylabel('ECM',size = 15)
ax.tick_params(labelsize = 15)
ax.grid(color='gray', linestyle='dashed', linewidth=1, alpha=0.4)

ax = fig1.add_subplot(1,2,2)
ax.plot(R2)
ax.set_title('Evolución del $R^2$',size = 20)
ax.set_xlabel('Modelo',size = 15)
ax.set_ylabel('$R^2$',size = 15)
ax.tick_params(labelsize = 15)
ax.grid(color='gray', linestyle='dashed', linewidth=1, alpha=0.4)
```





Algunos modelos llaman la atención por su baja calidad, pero eso no es motivo para frenar la investigación. Por ello, se definen matrices vacías de las dimensiones adecuadas y se utiliza un bucle para rellenarlas utilizando la distancia entre coeficientes de modelos.

```
matdist_P1 = np.zeros([len(ECOICOP),len(ECOICOP)])
matdist_P2 = np.zeros([len(ECOICOP),len(ECOICOP)])
matdist_P3 = np.zeros([len(ECOICOP),len(ECOICOP)])
matdist_P4 = np.zeros([len(ECOICOP),len(ECOICOP)])

p = 0
for mat in [matdist_P1,matdist_P2,matdist_P3,matdist_P4]:
  for i in range(len(ECOICOP)):
    for j in range(i+1,len(ECOICOP)):
      # cálculo de la distancia euclidea entre los
      # vectores de coeficientes
      mat[i,j] = np.linalg.norm(modelos[p+i]-modelos[p+j])
      mat[j,i] = mat[i,j]
  p += 12
```

Basada en los coeficientes del modelo ARIMA

La librería *pmdarima* contiene la función *auto_arima*, que conviene usar para el ajuste automático de modelos ARIMA porque de otra forma habría que ajustar cada modelo a mano. Y, teniendo en cuenta que son $12x4 = 48$ modelos en total los que hay que construir, el estudio se puede alargar innecesariamente.

Para utilizar la función, es necesario importarla.

```
!pip install pmdarima --quiet
from pmdarima.arima import auto_arima
```

Para recopilar las medidas MSE y AIC, se crean listas vacías que se irán rellenando mediante un bucle. El objetivo de esto es visualizar la bondad de los modelos contruidos.

```
MSE = []
AIC = []
```

También es necesario guardar los coeficientes de los modelos ajustados para compararlos posteriormente.

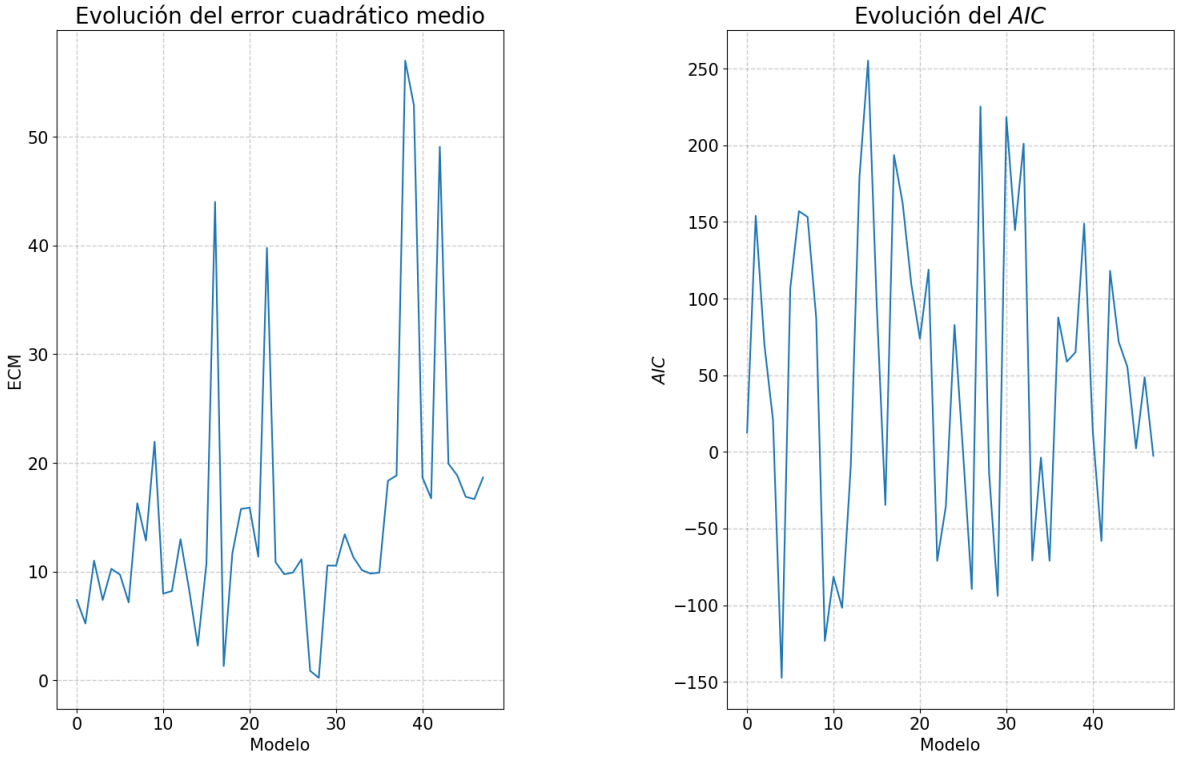
```
modelos = []
```

Por último, para cada periodo se calculan todos los coeficiente y medidas interesantes mediante un bucle.

```
for d in [
    datos_P1,
    datos_P2,
    datos_P3,
    datos_P4]:
  for k in range(len(d.columns)):
    # ARIMA
    model = auto_arima(d.iloc[:,k], seasonal=True, m=12,
                  start_p=0, start_q=0, max_p=2, max_q=2,
                  start_P=0, start_Q=0, max_P=2, max_Q=2,
                  trace=True, error_action='ignore',
                  suppress_warnings=True, stepwise=True)
    modelos.append(model.params())
    predictions = model.predict_in_sample(d.iloc[:,k])
    MSE.append(np.sqrt(mean_squared_error(predictions,
                                          d.iloc[:,k])))
    AIC.append(model.aic())
```

Para estudiar el comportamiento de los modelos ajustados, se representan la evolución del error cuadrático medio para cada modelo y la evolución del Criterio de Información de Akaike (AIC).

```
fig1 = plt.figure(figsize=(18,11))
fig1.subplots_adjust(hspace=0.5, wspace=0.5)

ax = fig1.add_subplot(1,2,1)
ax.plot(MSE)
ax.set_title('Evolución del error cuadrático medio',size = 20)
ax.set_xlabel('Modelo',size = 15)
ax.set_ylabel('ECM',size = 15)
ax.tick_params(labelsize = 15)
ax.grid(color='gray', linestyle='dashed', linewidth=1, alpha=0.4)

ax = fig1.add_subplot(1,2,2)
ax.plot(AIC)
ax.set_title('Evolución del $AIC$',size = 20)
ax.set_xlabel('Modelo',size = 15)
ax.set_ylabel('$AIC$',size = 15)
ax.tick_params(labelsize = 15)
ax.grid(color='gray', linestyle='dashed', linewidth=1, alpha=0.4)
```



A pesar de que hay modelos con error cuadrático medio o con AIC bastante alto, se definen matrices vacías de las dimensiones adecuadas y se utiliza un bucle para rellenarlas utilizando la distancia entre coeficientes de modelos ARIMA.

```
matdist_P1 = np.zeros([len(ECOICOP),len(ECOICOP)])
matdist_P2 = np.zeros([len(ECOICOP),len(ECOICOP)])
matdist_P3 = np.zeros([len(ECOICOP),len(ECOICOP)])
matdist_P4 = np.zeros([len(ECOICOP),len(ECOICOP)])

p = 0
for mat in [
    matdist_P1,
    matdist_P2,
    matdist_P3,
    matdist_P4]:
  for i in range(len(ECOICOP)):
    for j in range(i+1,len(ECOICOP)):
      x = modelos[i]-modelos[j]
      x = x[~np.isnan(x)]
      mat[i,j] = np.linalg.norm(x)
      mat[j,i] = mat[i,j]
  p += 12
```

Basada en el error de predicción con modelos RLM

El modelo de regresión lineal múltiple se puede ajustar usando la librería *sklearn*, y las medidas relacionadas con dicho modelo, también. Para utilizarlas es necesario importarlas.

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import  mean_squared_error
```

Se definen matrices vacías de las dimensiones adecuadas y se utiliza un bucle para rellenarlas utilizando la distancia basada en el error de predicción con modelos RLM.

```
matdist_P1 = np.zeros([len(ECOICOP),len(ECOICOP)])
matdist_P2 = np.zeros([len(ECOICOP),len(ECOICOP)])
matdist_P3 = np.zeros([len(ECOICOP),len(ECOICOP)])
matdist_P4 = np.zeros([len(ECOICOP),len(ECOICOP)])
```

Para ajustar el modelo, lo primero es configurar el tipo de modelo que se pretende ajustar. En este caso, modelo de regresion lineal múltiple con coeficiente independiente.

```
lm = LinearRegression(fit_intercept=True)
```

Por último, para cada periodo se calculan todos los coeficiente y medidas interesantes mediante un bucle.

```
for d,matdist_P in [
    [datos_P1,matdist_P1],
     [datos_P2,matdist_P2],
      [datos_P3,matdist_P3],
       [datos_P4,matdist_P4]]:
  # Como se va a transformar el conjunto de datos,
  # para que no afecte al cuaderno entero se hará una
  # copia independiente del mismo
  datos_P = d.copy()
  # Definición de las variables dummy, que representen el mes
  #  (11 variables dummy), y la variable tiempo.
  # Se añaden las variables al dataset inicial mediante un bucle
  for i in range(1,13):
    name = "dummy"+str(i)
    datos_P[name] = [int(date.month == i) for date in datos_P.index]
  datos_P["tiempo"] = [i for i in range(datos_P.shape[0])]

  # Regresión lineal
  linea = 0 # para movernos en la matriz de distancias
  for Y in datos_P.columns[range(len(ECOICOP))]:
    datos = datos_P[[Y,"dummy1","dummy2","dummy3","dummy4","dummy5",
                     "dummy6","dummy7","dummy8","dummy9","dummy10",
                     "dummy11","dummy12","tiempo"]]
    # Se ajusta el modelo a los datos
    reg = lm.fit(datos.drop(columns = [Y]), datos[Y])
    columna = 0 # para movernos en la matriz de distancias
    predictions = reg.predict(datos.drop(columns = [Y]))
    # Se compara la predicción con los valores observados de cada serie
    for Z in datos_P.columns[range(len(ECOICOP))]:
      # Se rellena la matriz de distancias con
      # la raiz error cuadratico medio
      matdist_P[linea,columna] = np.sqrt(mean_squared_error(predictions,
                                                            datos_P[Z]))
      columna += 1
    linea += 1
```

Como las matrices construidas no son simétricas ni tienen la diagonal compuesta por ceros, en general, es necesario adaptarlas para que cumplan estos requisitos.

```
# Se completa la matriz de distancias
for matdist in [
    matdist_P1,
    matdist_P2,
    matdist_P3,
    matdist_P4]:
  for i in range(matdist.shape[0]):
    matdist[i,i] = 0
    for j in range(i+1,matdist.shape[1]):
      m = max(matdist[i,j],matdist[j,i])
      # tomamos el maximo y no el minimo para que las diferencias
      # sean más significantes
      matdist[i,j] = m
      matdist[j,i] = m
```

Basada en el error de predicción con modelos ARIMA

El modelo ARIMA se puede ajustar usando la función *auto_arima* de la librería *pmdarima*. Para utilizarla es necesario importarla.

```
!pip install pmdarima --quiet
from pmdarima.arima import auto_arima
```

Se definen matrices vacías de las dimensiones adecuadas y se utiliza un bucle para rellenarlas utilizando la distancia basada en el error de predicción con modelos ARIMA.

```
matdist_P1 = np.zeros([len(ECOICOP),len(ECOICOP)])
matdist_P2 = np.zeros([len(ECOICOP),len(ECOICOP)])
matdist_P3 = np.zeros([len(ECOICOP),len(ECOICOP)])
matdist_P4 = np.zeros([len(ECOICOP),len(ECOICOP)])
```

A través de un bucle se busca automáticamente el modelo que mejor se ajusta a cada serie en cada periodo.

```
for d,m in [
    [datos_P1,matdist_P1],
     [datos_P2,matdist_P2],
      [datos_P3,matdist_P3],
       [datos_P4,matdist_P4]]:
  for k in range(len(d.columns)):
    # ARIMA
    model = auto_arima(d.iloc[:,k], seasonal=True, m=12,
                  start_p=0, start_q=0, max_p=2, max_q=2,
                  start_P=0, start_Q=0, max_P=2, max_Q=2,
                  trace=True, error_action='ignore',
                  suppress_warnings=True, stepwise=True)
    for s in range(len(d.columns)):
      predictions = model.predict_in_sample(d.iloc[:,s])
      m[k,s] = np.sqrt(mean_squared_error(predictions, d.iloc[:,s]))
```

Por último, se calculan las disimilaridades y se rellenan las matrices de forma que sean simétricas y con la diagonal de unos.

```
for matdist in [matdist_P1,matdist_P2,matdist_P3,matdist_P4]:
  for i in range(matdist.shape[0]):
    matdist[i,i] = 0
    for j in range(i+1,matdist.shape[1]):
      m = max(matdist[i,j],matdist[j,i])
      # tomamos el maximo y no el minimo para que las diferencias
      # sean más significantes
      matdist[i,j] = m
      matdist[j,i] = m
```

Basada en los coeficientes del modelo de cadenas de Markov ocultas

La librería *hmmlearn* contiene la función *hmm*, para el ajuste de modelos de cadenas de Markov ocultas (Hidden Markov Models).

```
!pip install hmmlearn --quiet
from hmmlearn import hmm
```

Se fija el número de estados ocultos, que serán los diferentes clusters en que se dividan las series. Este número ha sido escogido en base al método del codo, que ha sido desarrollado en la siguiente sección del código.

```
n_states = 4
```

Se preparan DataFrames para recoger los resultados en cada periodo.

```
conglomerados1 = pd.DataFrame({'Conglomerado1': range(len(ECOICOP)),
                               'ECOICOP': ECOICOP})
conglomerados2 = pd.DataFrame({'Conglomerado2': range(len(ECOICOP)),
                               'ECOICOP': ECOICOP})
conglomerados3 = pd.DataFrame({'Conglomerado3': range(len(ECOICOP)),
                               'ECOICOP': ECOICOP})
conglomerados4 = pd.DataFrame({'Conglomerado4': range(len(ECOICOP)),
                               'ECOICOP': ECOICOP})
```

Por último, se implementa el modelo de cadenas de Markov ocultas con 4 estados ocultos y se ajusta a los datos de cada periodo.

```
count = 1
# Para cada conjunto de datos se repite el mismo proceso:
for D,C in [
    [datos_P1.transpose(),conglomerados1],
     [datos_P2.transpose(),conglomerados2],
      [datos_P3.transpose(),conglomerados3],
       [datos_P4.transpose(),conglomerados4]]:
  name = 'Conglomerado'+str(count)

  # Definicion del tipo de distribución
  model = hmm.GaussianHMM(n_components=n_states)

  # Entrenar el modelo en los datos
  model.fit(D)

  # Se predicen la secuencia de estados ocultos
  hidden_states = model.predict(D)

  # Se guarda el resultado
  C[name] = hidden_states
  count += 1
```

Se unifican los resultados en un DataFrame que represente la evolución temporal que sufren los conglomerados.

```
conglomerados12 = conglomerados1.merge(conglomerados2,
                                       on = "ECOICOP")
conglomerados123 = conglomerados12.merge(conglomerados3,
                                         on = "ECOICOP")
conglomerados_evolucion = conglomerados123.merge(conglomerados4,
                                                 on = "ECOICOP")
```

Por último, se estructura bien el DataFrame para que el acceso sea lo más sencillo posible.

```
conglomerados_evolucion = pd.DataFrame({
    'Periodo1':[c for c in conglomerados_evolucion['Conglomerado1']],
    'Periodo2':[c for c in conglomerados_evolucion['Conglomerado2']],
    'Periodo3':[c for c in conglomerados_evolucion['Conglomerado3']],
    'Periodo4':[c for c in conglomerados_evolucion['Conglomerado4']]},
    index = conglomerados_evolucion['ECOICOP'])
conglomerados_evolucion.index.rename("Grupos ECOICOP", inplace = True)
conglomerados_evolucion['ECOICOP'] = [i for i in range(len(ECOICOP))]
conglomerados_evolucion
```

Grupos ECOICOP	Periodo1	Periodo2	Periodo3	Periodo4	ECOICOP
Alimentos y bebidas no alcohólicas	1	0	0	1	0
Bebidas alcohólicas y tabaco	3	3	3	0	1
Vestido y calzado	0	2	2	2	2
Vivienda, agua, electricidad, gas y otros combustibles	1	0	0	3	3
Muebles, artículos del hogar y artículos para el mantenimiento corriente del hogar	0	2	3	0	4
Sanidad	0	0	3	0	5
Transporte	1	0	0	1	6
Comunicaciones	2	1	1	0	7
Ocio y cultura	2	2	1	0	8
Enseñanza	1	0	3	0	9
Restaurantes y hoteles	1	0	0	1	10
Otros bienes y servicios	1	0	0	0	11

Clustering

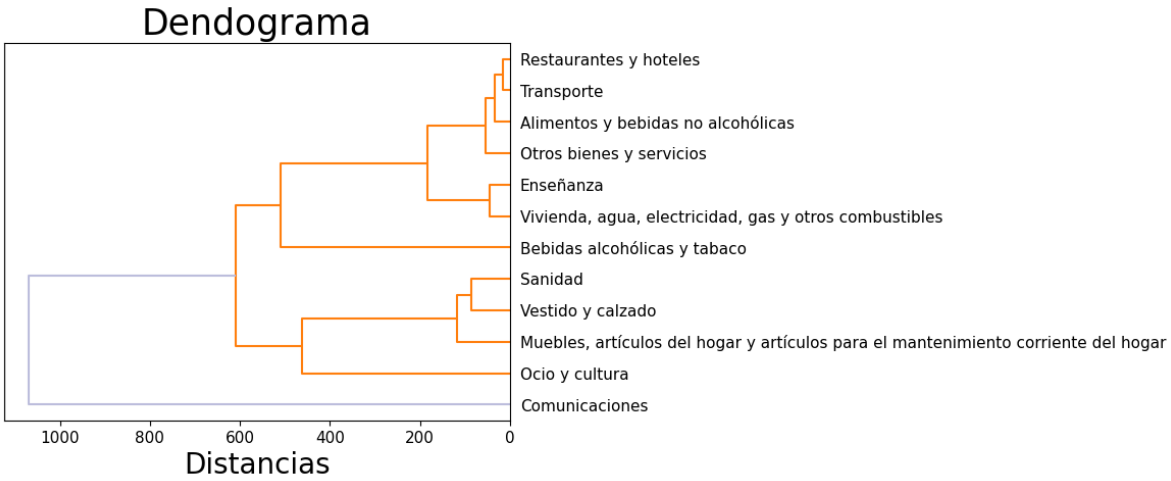
Jerárquico aglomerativo

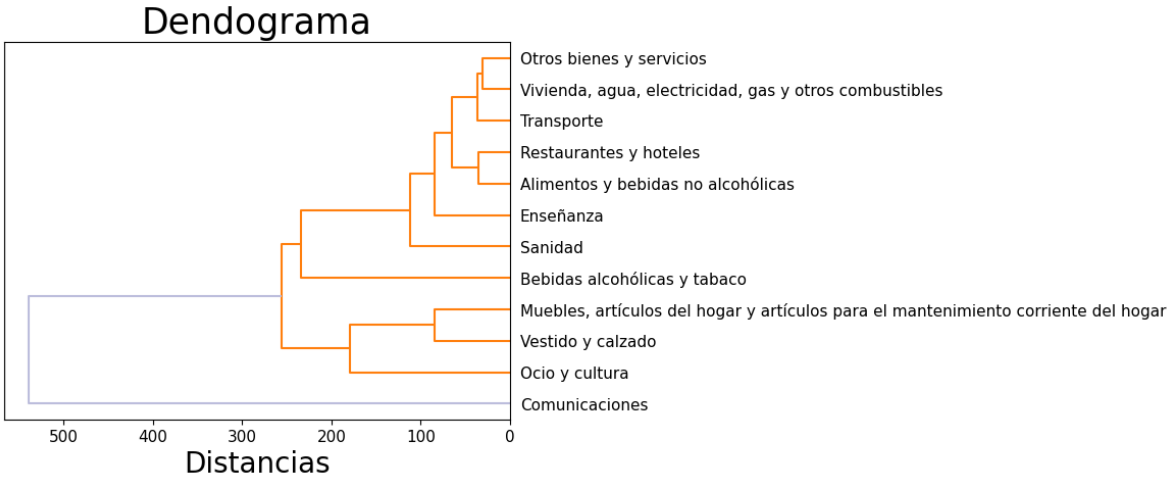
El método jerárquico aglomerativo se encuentra implementado en la librería *sklearn*.

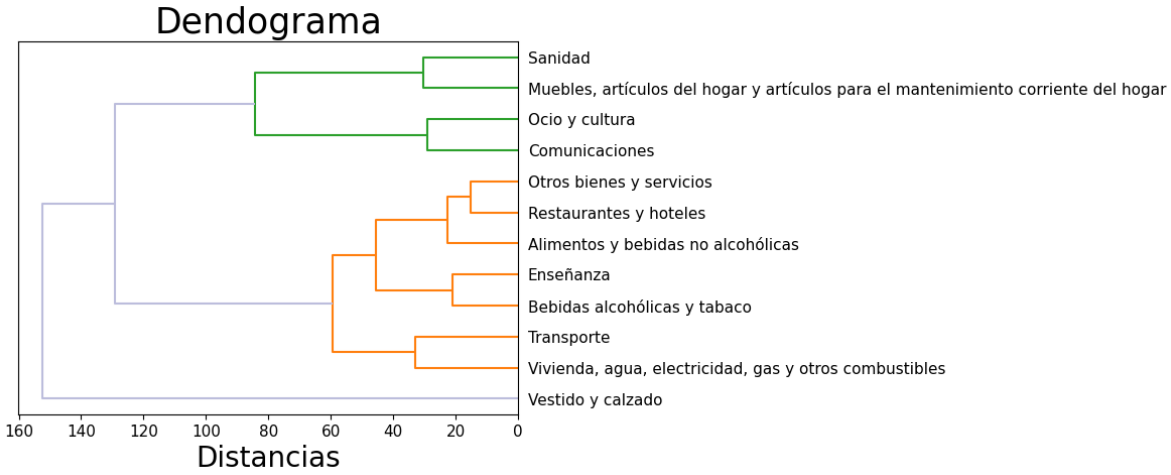
Previamente, para visualizar el dendograma de cada periodo, se emplean las las funciones *linkage*, para ajustar el modelo jerárquico a cada conjunto de datos de los diferentes periodos del estudio, y *dendrogram*, para representar el proceso de agrupación en conglomerados, llamado dendograma.

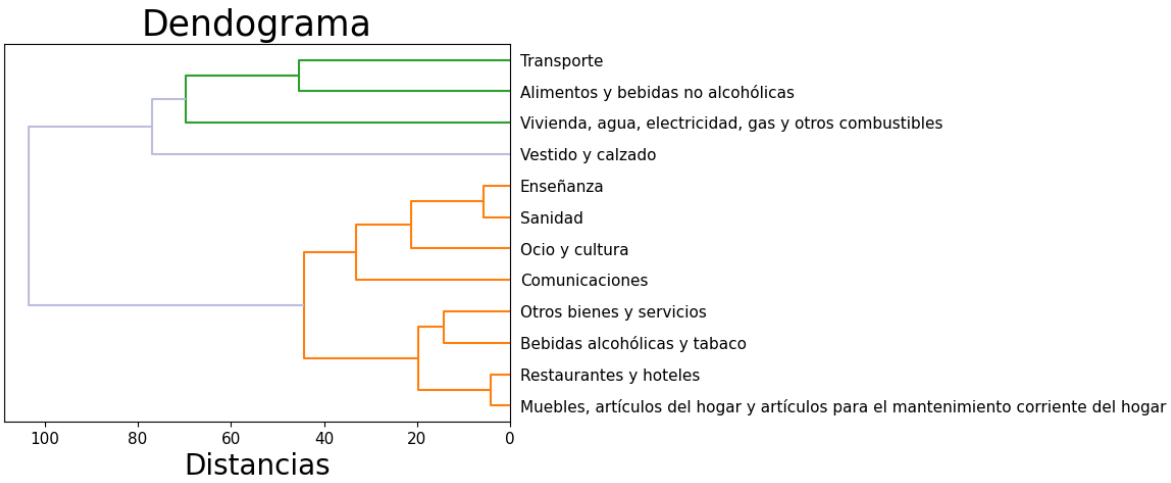
```
from scipy.cluster.hierarchy import dendrogram, linkage
```

```
for M in [
    matdist_P1,
    matdist_P2,
    matdist_P3,
    matdist_P4]:

    # Ajuste del modelo jurárquico sobre los datos
    model = linkage(M, method='average',
                    metric='euclidean', optimal_ordering=False)

    # Cálculo y representación del dendograma
    dendograma = dendrogram(model,labels = ECOICOP,
                        leaf_rotation = 0,
                        above_threshold_color='#bcbddc',
                        orientation='left')
    plt.title('Dendograma',fontsize=25)
    plt.xlabel('Distancias',fontsize=20)
    plt.ylabel('')
    plt.tick_params(labelsize = 11)
    plt.show()
```









Por unificar el criterio, y que los periodos sean comparables, se tomarán 4 clusters.

```
nclusters = 4
```

A continuación, una vez que se escoge el número de clusters en base a los dendogramas, se emplea la función *AgglomerativeClustering* de la librería sklearn para implementar el análisis. Es necesario

```
from sklearn.cluster import AgglomerativeClustering
```

Primero se define la técnica que se va a implementar.

```
clust_model1 = AgglomerativeClustering(n_clusters = nclusters,
                                       affinity = 'precomputed',
                                       linkage = 'average')
```

Después, se ajusta la técnica a la matriz de distancias.

```
clust_model1.fit(matdist_P1)
```

Ahora es posible consultar la agrupación que se obtiene como resultado.

```
clust_model1.labels_
```

```
array([2, 3, 0, 2, 0, 0, 2, 1, 0, 2, 2, 2])
```

Se construye un DataFrame en el que quede más clara toda la información de los clusters.

```
conglomerados1 = pd.DataFrame({'Conglomerado1': clust_model1.labels_,
                               'ECOICOP': ECOICOP})
```

Esto mismo se repite para cada periodo y después se unifican los resultados en un DataFrame.

```
# Periodo 2
clust_model2 = AgglomerativeClustering(n_clusters = nclusters,
                                       affinity = 'precomputed',
                                       linkage = 'average')
clust_model2.fit(matdist_P2)

conglomerados2 = pd.DataFrame({'Conglomerado2': clust_model2.labels_,
                               'ECOICOP': ECOICOP})

# Periodo 3
clust_model3 = AgglomerativeClustering(n_clusters = nclusters,
                                       affinity = 'precomputed',
                                       linkage = 'average')
clust_model3.fit(matdist_P3)

conglomerados3 = pd.DataFrame({'Conglomerado3': clust_model3.labels_,
                               'ECOICOP': ECOICOP})

# Periodo 4
clust_model4 = AgglomerativeClustering(n_clusters = nclusters,
                                       affinity = 'precomputed',
                                       linkage = 'average')
clust_model4.fit(matdist_P4)

conglomerados4 = pd.DataFrame({'Conglomerado4': clust_model4.labels_,
                               'ECOICOP': ECOICOP})

# Unificamos los resultados en un DataFrame que represente la evolución
conglomerados12 = conglomerados1.merge(conglomerados2,on = "ECOICOP")
conglomerados123 = conglomerados12.merge(conglomerados3,on = "ECOICOP")
conglomerados_evolucion = conglomerados123.merge(conglomerados4,
                                                 on = "ECOICOP")

# Pasamos el conjunto de datos a DataFrame
conglomerados_evolucion = pd.DataFrame({
    'Periodo1':[c for c in conglomerados_evolucion['Conglomerado1']],
    'Periodo2':[c for c in conglomerados_evolucion['Conglomerado2']],
    'Periodo3':[c for c in conglomerados_evolucion['Conglomerado3']],
    'Periodo4':[c for c in conglomerados_evolucion['Conglomerado4']]},
    index = conglomerados_evolucion['ECOICOP'])
conglomerados_evolucion.index.rename("Grupos ECOICOP", inplace = True)
conglomerados_evolucion['ECOICOP'] = [i for i in range(len(ECOICOP))]
conglomerados_evolucion
```

Grupos ECOICOP	Periodo1	Periodo2	Periodo3	Periodo4	ECOICOP
Alimentos y bebidas no alcohólicas	2	2	0	3	0
Bebidas alcohólicas y tabaco	3	3	0	0	1
Vestido y calzado	0	0	3	2	2
Vivienda, agua, electricidad, gas y otros combustibles	2	2	0	1	3
Muebles, artículos del hogar y artículos para el mantenimiento corriente del hogar	0	0	1	0	4
Sanidad	0	2	1	0	5
Transporte	2	2	0	1	6
Comunicaciones	1	1	2	0	7
Ocio y cultura	0	0	2	0	8
Enseñanza	2	2	0	0	9
Restaurantes y hoteles	2	2	0	0	10
Otros bienes y servicios	2	2	0	0	11

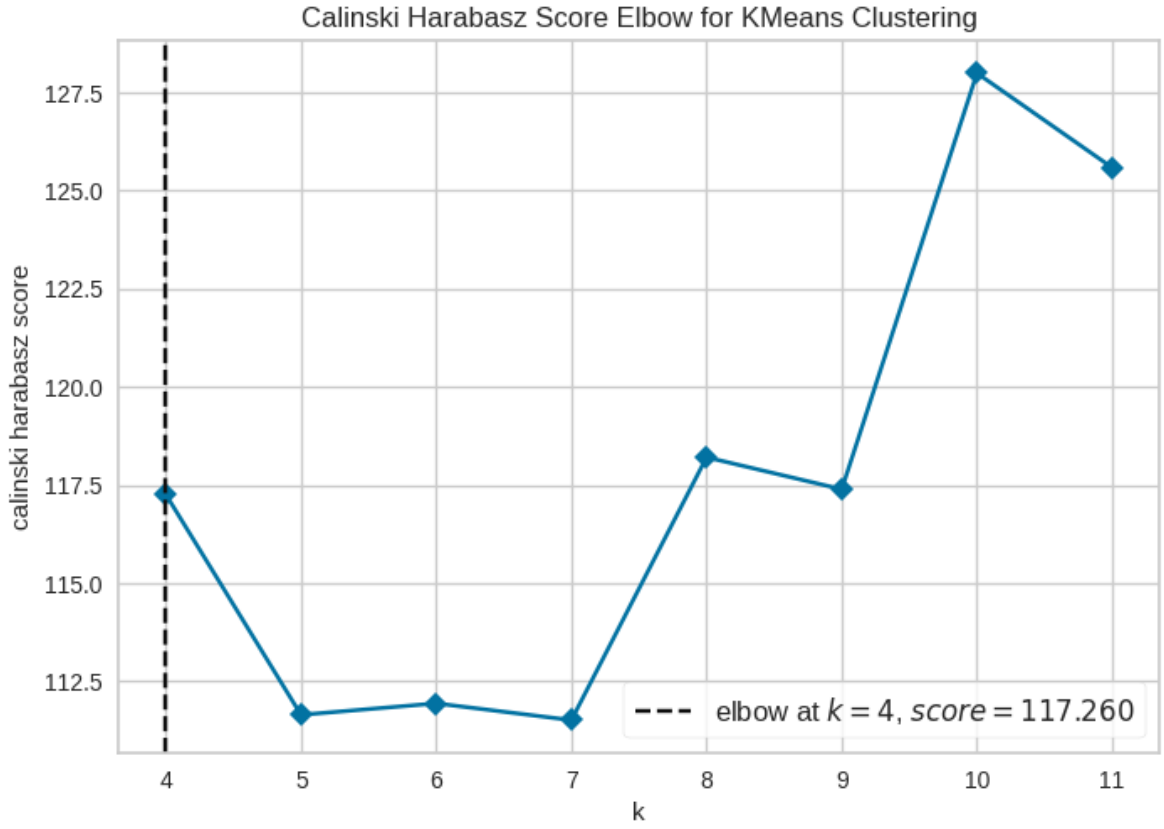
Selección del número de clusters K-Means

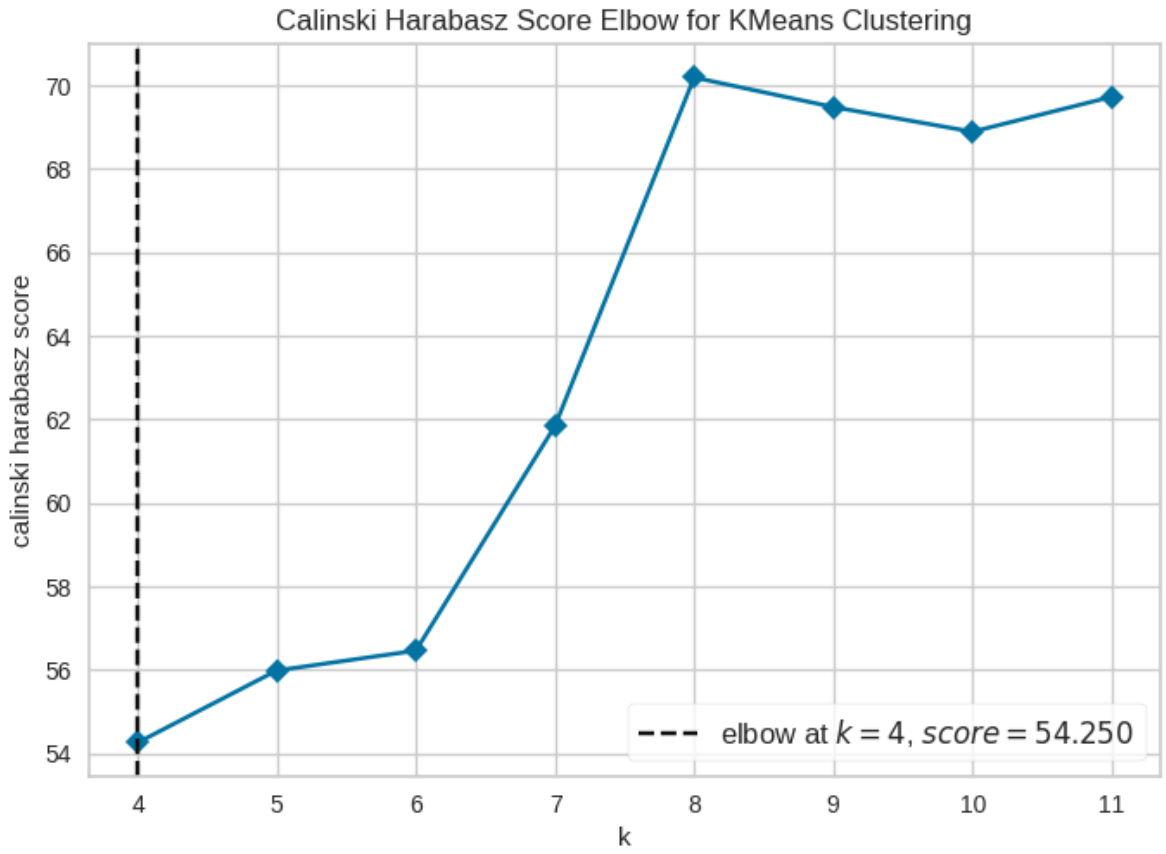
Para seleccionar el número óptimo de clusters, se usará una función ya implementada en la librería *yellowbrick*. También es necesario iniciar un modelo Kmeans, por lo que se importan ambas funciones.

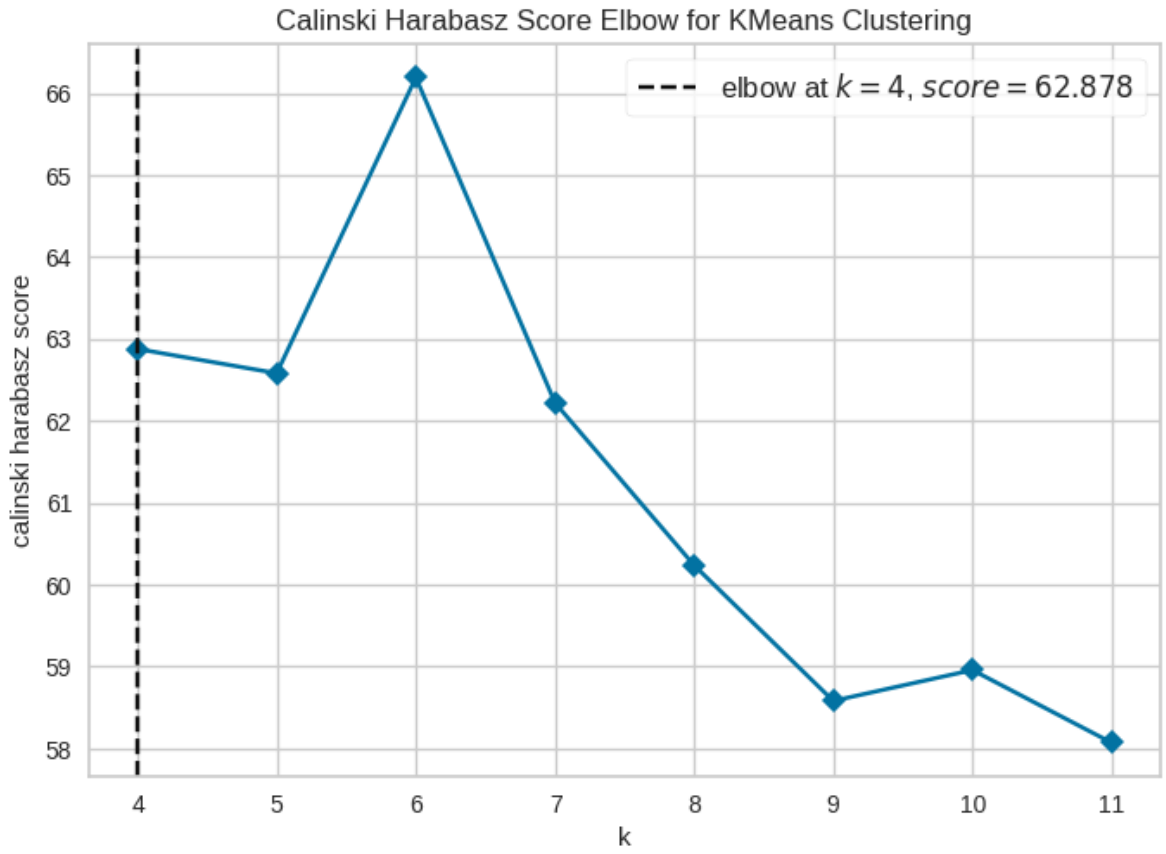
```
from yellowbrick.cluster import KElbowVisualizer
from sklearn.cluster import KMeans
```

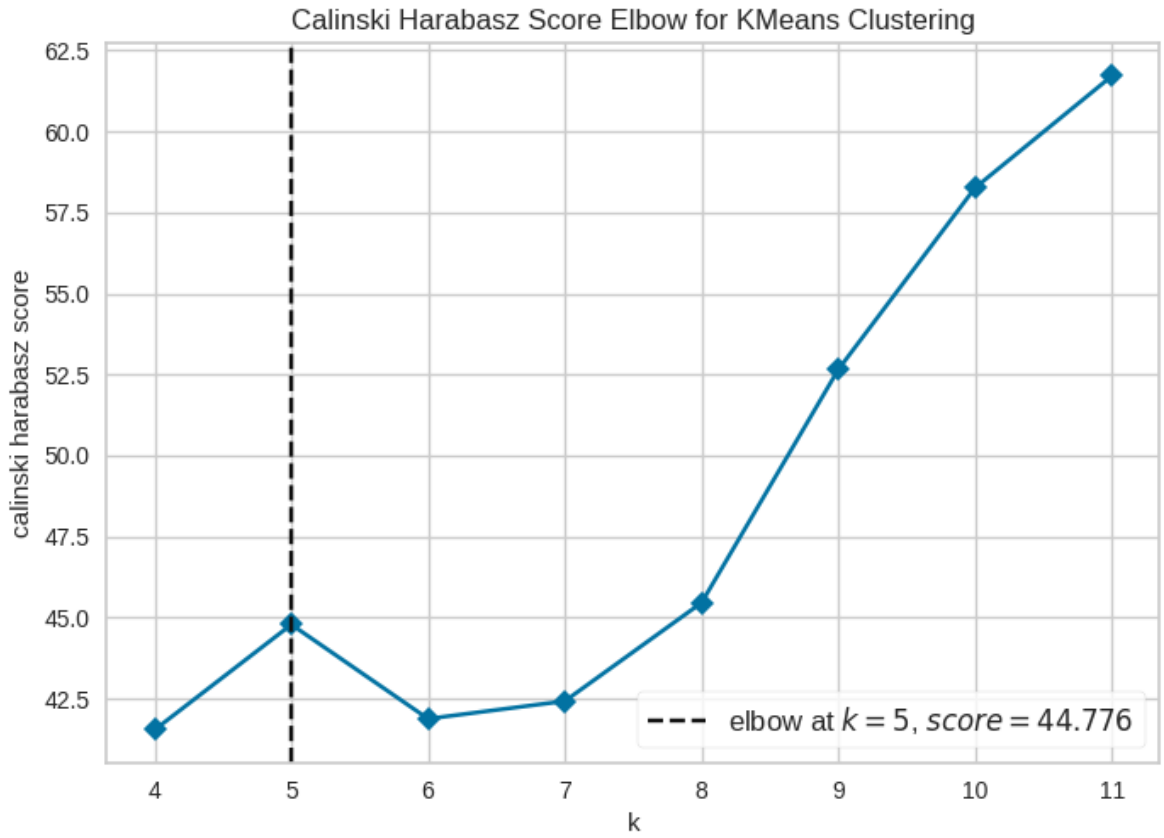
A continuación, para cada periodo se inicia un modelo, el K-Means, por ejemplo, y el visualizador que acaba de ser importado.

```
for D in [
    datos_P1,
    datos_P2,
    datos_P3,
    datos_P4]:

    # Se inicia el modelo y el visualizador para después
    # aplicarlo a los datos
    model = KMeans()
    visualizer = KElbowVisualizer(
        model, k=(4,12), metric='calinski_harabasz',
        timings=False, locate_elbow=True
    )

    visualizer.fit(D)  # Se ajustan los datos al visualizador
    visualizer.show()
```




Calinski Harabasz Score Elbow for KMeans Clustering
calinski harabasz score
70
68
66
64
62
60
58
56
54
4
5
6
7
8
9
10
11
k
elbow at $k = 4$, $score = 54.250$


Calinski Harabasz Score Elbow for KMeans Clustering
elbow at $k = 4$, $score = 62.878$
calinski harabasz score
66
65
64
63
62
61
60
59
58
4
5
6
7
8
9
10
11
k



Como en la mayoría se fija el número de conglomerados a 4, se toma este valor para todos los periodos, para unificar el criterio y poder tener resultados comparables.

K-means

```
from sklearn.cluster import KMeans
```

```
n = len(ECOICOP)

# Definición de los vectores de distancias
vecdist_P1 = np.zeros((n,1))
vecdist_P2 = np.zeros((n,1))
vecdist_P3 = np.zeros((n,1))
vecdist_P4 = np.zeros((n,1))

# Se rellenan los vectores con la primera
# fila de la matriz de distancias
for v,m in [
    [vecdist_P1,matdist_P1[0,:]],
     [vecdist_P2,matdist_P2[0,:]],
      [vecdist_P3,matdist_P3[0,:]],
       [vecdist_P4,matdist_P4[0,:]]]:
  for i in range(0,n):
    v[i,0] = m[i]

nclusters = 4 # según la mayoría de los gráficos anteriores
kmeans_P1 = KMeans(n_clusters=nclusters, random_state=0,
                   n_init="auto").fit(vecdist_P1)
kmeans_P2 = KMeans(n_clusters=nclusters, random_state=0,
                   n_init="auto").fit(vecdist_P2)
kmeans_P3 = KMeans(n_clusters=nclusters, random_state=0,
                   n_init="auto").fit(vecdist_P3)
kmeans_P4 = KMeans(n_clusters=nclusters, random_state=0,
                   n_init="auto").fit(vecdist_P4)
```

```
# Acceso a los conglomerados asignadas
kmeans_P1.labels_
```

```
array([1, 0, 0, 1, 0, 0, 1, 2, 3, 1, 1, 1], dtype=int32)
```

```
# Construir un conjunto de datos que contenga los clusters por periodo y
# los nombres de las observaciones

conglomerados1 = pd.DataFrame({'Conglomerado1': kmeans_P1.labels_,
                               'ECOICOP': ECOICOP}) # Periodo 1
conglomerados2 = pd.DataFrame({'Conglomerado2': kmeans_P2.labels_,
                               'ECOICOP': ECOICOP}) # Periodo 2
conglomerados3 = pd.DataFrame({'Conglomerado3': kmeans_P3.labels_,
                               'ECOICOP': ECOICOP}) # Periodo 3
conglomerados4 = pd.DataFrame({'Conglomerado4': kmeans_P4.labels_,
                               'ECOICOP': ECOICOP}) # Periodo 4

# Unificar los resultados en un DataFrame que represente la evolución
conglomerados12 = conglomerados1.merge(conglomerados2,on = "ECOICOP")
conglomerados123 = conglomerados12.merge(conglomerados3,on = "ECOICOP")
conglomerados_evolucion = conglomerados123.merge(conglomerados4,
                                                 on = "ECOICOP")

# Organizar el nuevo DataFrame
conglomerados_evolucion = pd.DataFrame({
    'Periodo1':[c for c in conglomerados_evolucion['Conglomerado1']],
    'Periodo2':[c for c in conglomerados_evolucion['Conglomerado2']],
    'Periodo3':[c for c in conglomerados_evolucion['Conglomerado3']],
    'Periodo4':[c for c in conglomerados_evolucion['Conglomerado4']]},
    index = conglomerados_evolucion['ECOICOP'])
conglomerados_evolucion.index.rename("Grupos ECOICOP", inplace = True)
conglomerados_evolucion['ECOICOP'] = [i for i in range(len(ECOICOP))]
conglomerados_evolucion
```

Grupos ECOICOP	Periodo1	Periodo2	Periodo3	Periodo4	ECOICOP
Alimentos y bebidas no alcohólicas	1	0	1	2	0
Bebidas alcohólicas y tabaco	0	3	3	1	1
Vestido y calzado	0	3	2	3	2
Vivienda, agua, electricidad, gas y otros combustibles	1	0	3	0	3
Muebles, artículos del hogar y artículos para el mantenimiento corriente del hogar	0	3	0	1	4
Sanidad	0	0	0	0	5
Transporte	1	0	3	1	6
Comunicaciones	2	2	2	3	7
Ocio y cultura	3	1	2	0	8
Enseñanza	1	0	3	0	9
Restaurantes y hoteles	1	0	1	1	10
Otros bienes y servicios	1	0	1	0	11

Representación de grafos

Para representar los resultados de forma amena y visual, se emplearán grafos y las imágenes elegidas para cada grupo ECOICOP.

Para ello, será necesario importar nuevas librerías como *networkx* y funciones de *matplotlib* como *OffsetImage* y *AnnotationBbox*.

```
import networkx as nx
from matplotlib.offsetbox import OffsetImage, AnnotationBbox
```

Se define el número de clusters escogidos.

```
nclusters = n_states
```

A continuación, se crea un diccionario con las imágenes que representan los grupos ECOICOP para que sea sencillo acceder a ellas.

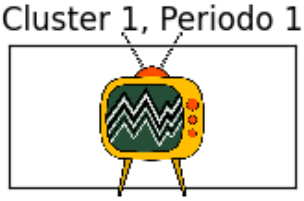
```
d = conglomerados_evolucion.to_dict('index')
ecoicop_grupos = [x for x,y in d.items()]
# Declaración de las imagenes que deben aparecer en los nodos
labeldict = {}
labeldict[ecoicop_grupos[0]] = ['apple.png',i1]
labeldict[ecoicop_grupos[1]] = ['beer.png',i2]
labeldict[ecoicop_grupos[2]] = ['chaqueta.png',i3]
labeldict[ecoicop_grupos[3]] = ['house.png',i4]
labeldict[ecoicop_grupos[4]] = ['closet.png',i5]
labeldict[ecoicop_grupos[5]] = ['medical.png',i6]
labeldict[ecoicop_grupos[6]] = ['car.png',i7]
labeldict[ecoicop_grupos[7]] = ['antena.png',i8]
labeldict[ecoicop_grupos[8]] = ['basketball.png',i9]
labeldict[ecoicop_grupos[9]] = ['books.png',i10]
labeldict[ecoicop_grupos[10]] = ['chef.png',i11]
labeldict[ecoicop_grupos[11]] = ['suma.png',i12]
```

Por último, se representan los clusters como grafos sin aristas. Para ello, se crea un bucle sobre los periodos y el número de clusters.

```
for p,per,fichero in [
    (1,'Periodo1','Periodo1.png'),
     (2,'Periodo2','Periodo2.png'),
      (3,'Periodo3','Periodo3.png'),
       (4,'Periodo4','Periodo4.png')]:
  d2 = {}
  for k in d.keys():
    # Se accede al cluster de la observación k en el periodo per
    d2[k] = [d[k][per]]
  df = pd.DataFrame(d2).transpose()
  for s in range(nclusters):
    cluster = df.loc[df.iloc[:,0] == s]
    # Se crea un grafo
    G = nx.Graph()

    # Se añaden los nodos
    c = 0
    for node in range(len(cluster[0])):
      name = 'image'+str(c)
      G.add_node(name, pos=(c,0))
      c += 1

    fig, ax = plt.subplots(figsize=(len(cluster[0])+1, 1))
    # Se añaden los nodos a la figura
    pos = nx.get_node_attributes(G, 'pos')
    nx.draw_networkx_nodes(G, pos, node_shape='s',
                           node_size=2000, node_color='white', ax=ax)
    nx.draw_networkx_edges(G, pos, width=2, edge_color='gray', ax=ax)

    # Se insertan las figuras en los nodos
    image_files = []
    for t in [im for im in cluster.index]:
      image_files.append(labeldict[t][0])
    for i, node in enumerate(G.nodes()):
        image = plt.imread(image_files[i])
        imagebox = OffsetImage(image, zoom=0.05)
        ab = AnnotationBbox(imagebox, pos[node], frameon=False)
        ax.add_artist(ab)

    # Se eliminan los ticks y las etiquetas
    ax.set_xticks([])
    ax.set_yticks([])
    ax.set_xticklabels([])
    ax.set_yticklabels([])

    ax.set_title('Cluster '+str(s)+', Periodo '+str(p))

    # Se guarda la imagen
    fich = 'cluster'+str(s)+'_'+fichero
    plt.savefig(fich, bbox_inches='tight')
```

Cluster 0, Periodo 1

Cluster 1, Periodo 1

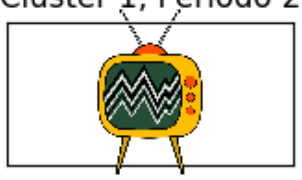


Cluster 2, Periodo 1

Cluster 3, Periodo 1

Cluster 0, Periodo 2

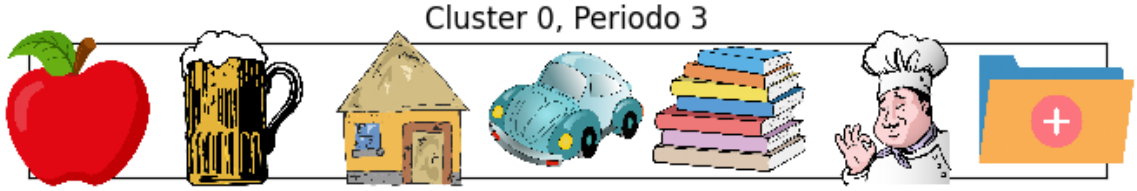
Cluster 1, Periodo 2



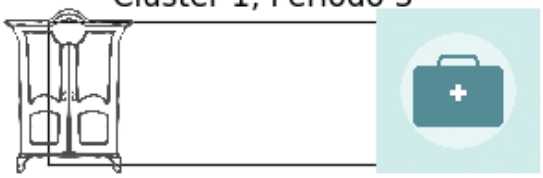
Cluster 2, Periodo 2

Cluster 3, Periodo 2

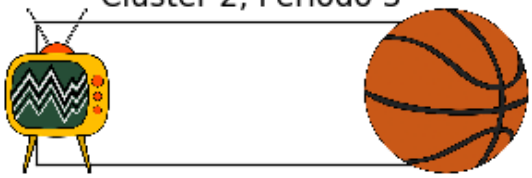
Cluster 0, Periodo 3



Cluster 1, Periodo 3

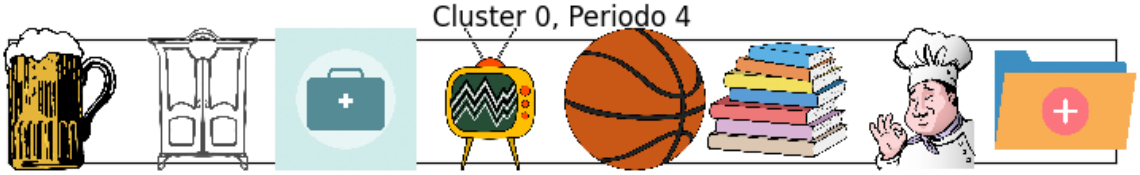


Cluster 2, Periodo 3

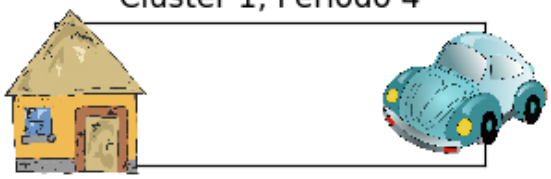


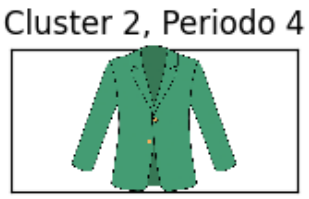
Cluster 3, Periodo 3

Cluster 0, Periodo 4



Cluster 1, Periodo 4



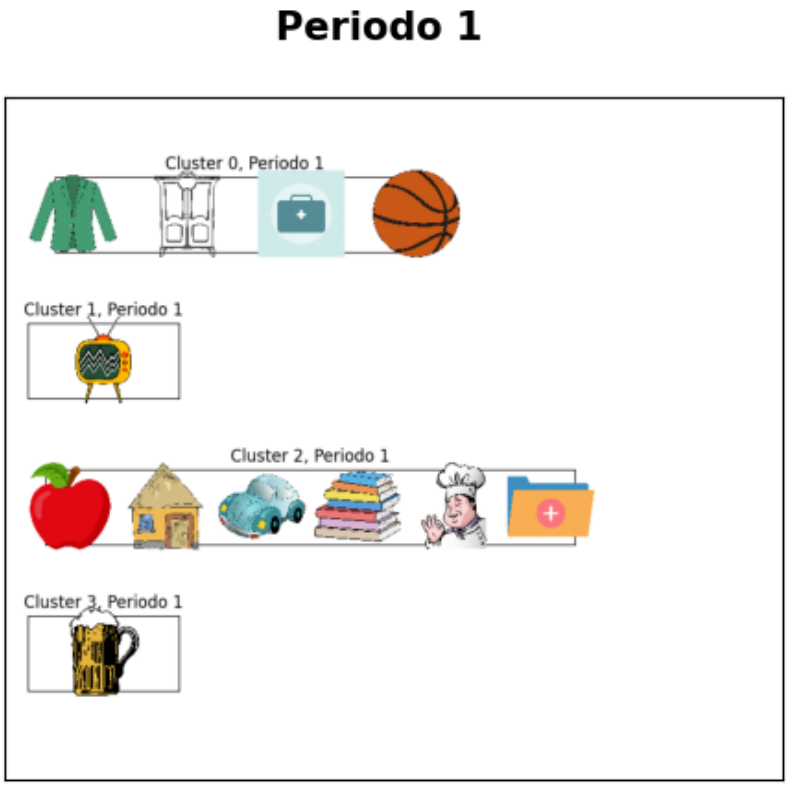


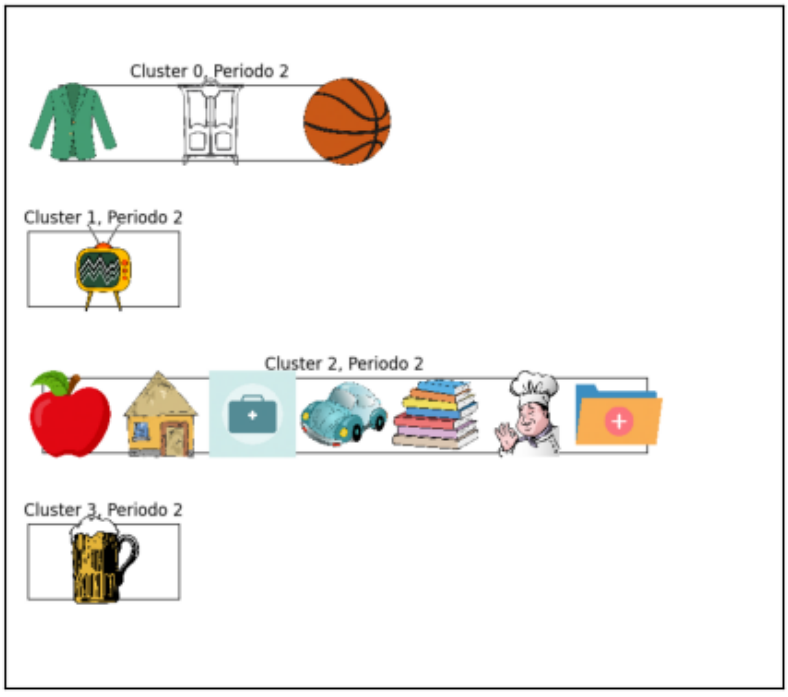
Se guardan todas las imágenes para hacer una composición de ellas después.

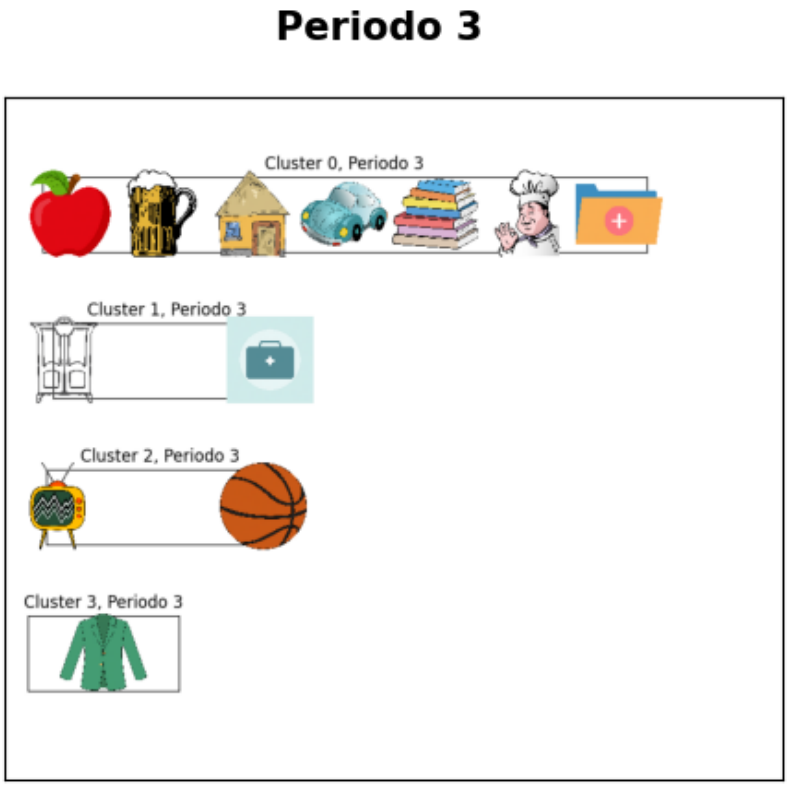
```
from PIL import Image
i01 = Image.open("/content/cluster0_Periodo1.png")
i02 = Image.open("/content/cluster0_Periodo2.png")
i03 = Image.open("/content/cluster0_Periodo3.png")
i04 = Image.open("/content/cluster0_Periodo4.png")

i11 = Image.open("/content/cluster1_Periodo1.png")
i12 = Image.open("/content/cluster1_Periodo2.png")
i13 = Image.open("/content/cluster1_Periodo3.png")
i14 = Image.open("/content/cluster1_Periodo4.png")

i21 = Image.open("/content/cluster2_Periodo1.png")
i22 = Image.open("/content/cluster2_Periodo2.png")
i23 = Image.open("/content/cluster2_Periodo3.png")
i24 = Image.open("/content/cluster2_Periodo4.png")

i31 = Image.open("/content/cluster3_Periodo1.png")
i32 = Image.open("/content/cluster3_Periodo2.png")
i33 = Image.open("/content/cluster3_Periodo3.png")
i34 = Image.open("/content/cluster3_Periodo4.png")
```

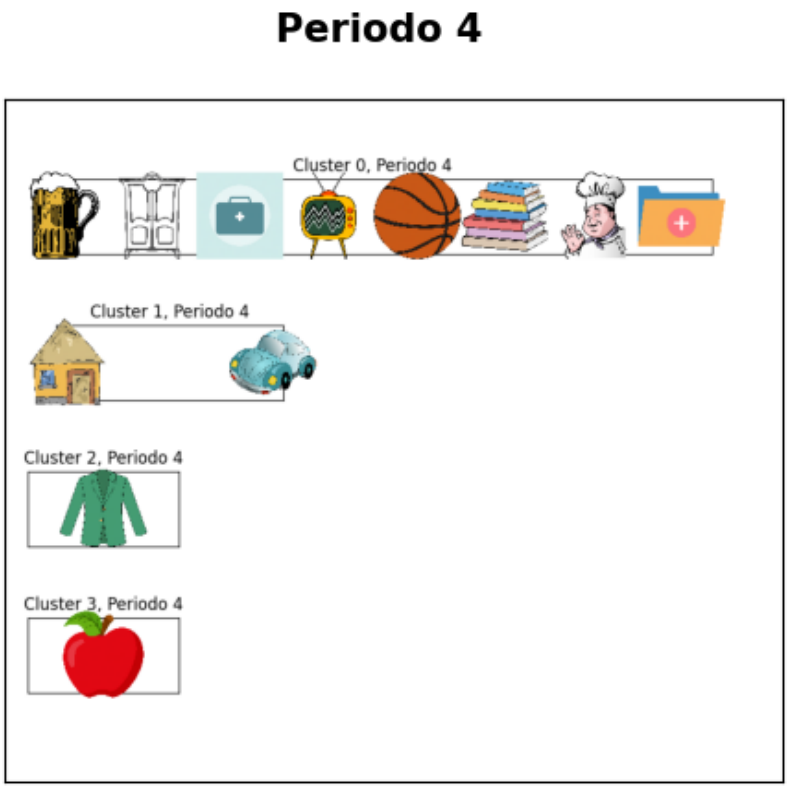
Las imágenes de cada cluster se recopilaran en una sóla, para cada periodo. Estas imágenes finales se guardarán para, posteriormente, unir todas en otra y poder comparar los clusters de cada periodo más fácilmente.

```
width = 800  # Ancho de la imagen final
height = 700  # Alto de la imagen final

count = 1
for ims in [
    [i01,i11,i21,i31],
     [i02,i12,i22,i32],
      [i03,i13,i23,i33],
       [i04,i14,i24,i34]]:
  fich = 'Periodo'+str(count)+'.png'
  # Crear imagen blanca de fondo
  imagen_final = Image.new('RGB', (width, height), (255, 255, 255))

  # Pegar imagen 1 en la posición (50, 10)
  imagen_final.paste(ims[0], (10, 50))
  # Pegar imagen 2 en la posición (200, 10)
  imagen_final.paste(ims[1], (10, 200))
  # Pegar imagen 3 en la posición (350, 10)
  imagen_final.paste(ims[2], (10, 350))
  # Pegar imagen 4 en la posición (500, 10)
  imagen_final.paste(ims[3], (10, 500))

  imagen_final.save('imagen_final.png')

  fig = plt.figure()
  fig.suptitle("Periodo "+str(count),size=15,
               weight='extra bold',stretch='extra-expanded')
  ax = fig.add_subplot(1,1,1)
  ax.imshow(imagen_final)
  ax.xaxis.set_ticks([])
  ax.yaxis.set_ticks([])
  plt.savefig(fich, bbox_inches='tight')
  count += 1

iP1 = Image.open("/content/Periodo1.png")
iP2 = Image.open("/content/Periodo2.png")
iP3 = Image.open("/content/Periodo3.png")
iP4 = Image.open("/content/Periodo4.png")
```


Periodo 1
Cluster 0, Periodo 1
Cluster 1, Periodo 1
Cluster 2, Periodo 1
Cluster 3, Periodo 1


Periodo 2
Cluster 0, Periodo 2
Cluster 1, Periodo 2
Cluster 2, Periodo 2
Cluster 3, Periodo 2

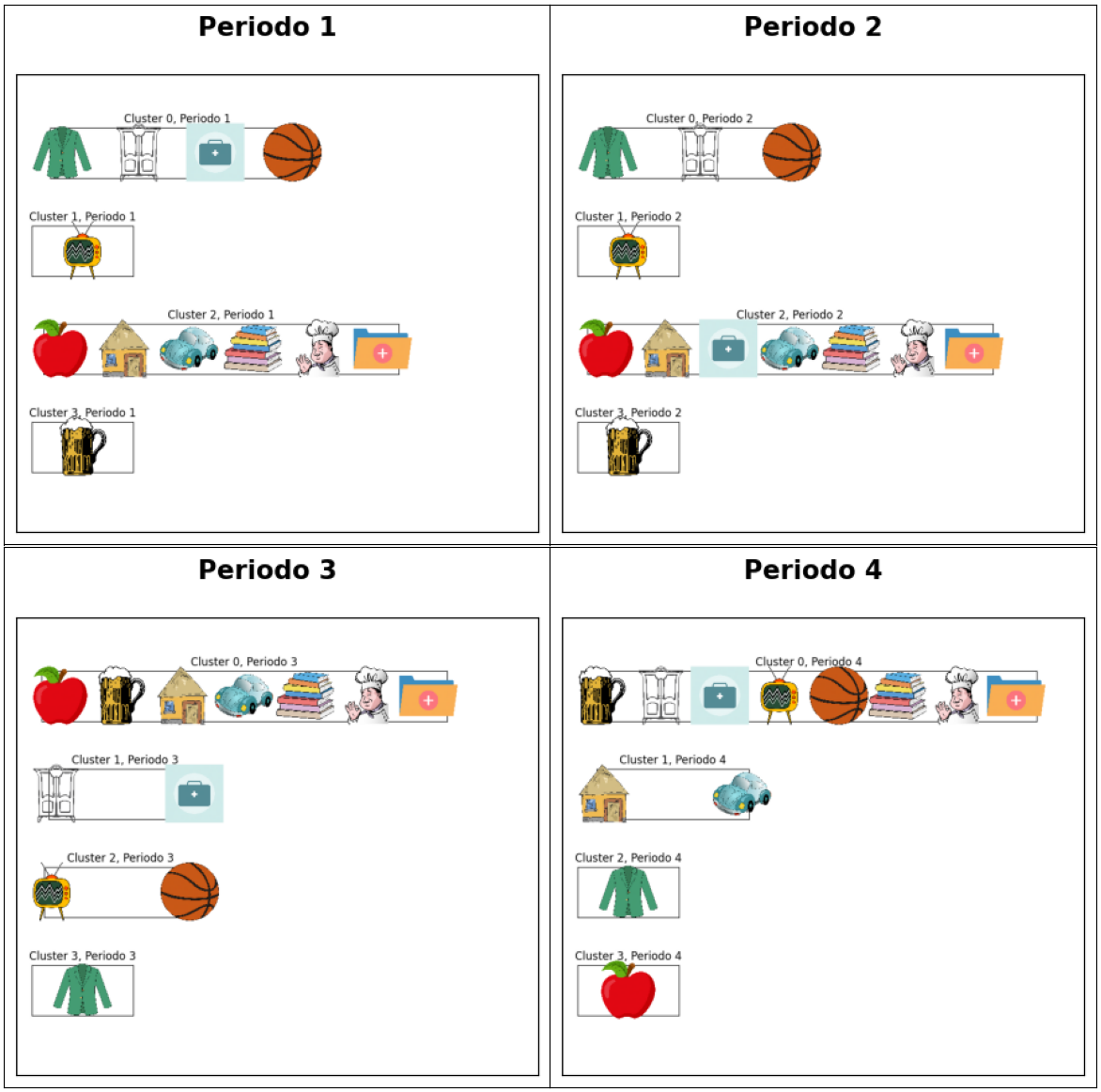




Se unifican las imágenes anteriores para observar la evolución entre conglomerados.

```
fig = plt.figure(figsize=(15,15))
fig.subplots_adjust(hspace=0, wspace=0)
fig.suptitle("Evolución conglomerados",fontsize=20,
             weight='extra bold',stretch='extra-expanded')

k = 1
for im in [iP1,iP2,iP3,iP4]:
    title = 'Periodo '+str(k)
    ax = fig.add_subplot(2,2,k)
    ax.xaxis.set_ticks([])
    ax.yaxis.set_ticks([])
    ax.imshow(im)
    k += 1

imagen = ax.get_figure()
ax.plot()
plt.savefig("Evolucion", bbox_inches='tight')
```

Evolución conglomerados



Estos resultados muestran que los conglomerados varían de un periodo a otro, lo cual avala la decisión de dividir las series en cuatro periodos para evitar el ruido que ello hubiera provocado. Si bien es cierto que la diferencia entre los dos primeros periodos únicamente es el cambio del grupo ECOICOP 'Sanidad'.

También resulta llamativo que 'Vestido y calzado' no esté siempre en un cluster aislado, debido a su gran estacionalidad. Pero es igualmente interesante, porque una de las razones por las que este análisis cobra sentido: estudiar agrupaciones no evidentes.

Otro dato a resaltar es que los grupos 'Vivienda, agua, gas y otros combustibles' y 'Transporte' parece que siempre van de la mano, lo que es lógico porque el gasto en transporte está ligado al combustible que consumen los vehículos. Igualmente ocurre con los grupos 'Enseñanza', 'Restaurantes y hoteles' y 'Otros bienes y servicios'. Esto último es difícil de predecir, ya que a priori para 'Restaurantes y hoteles' se esparía que fuera mucho más estacional.

Se observa también que 'Vestido y calzado' nunca comparte conglomerado con 'Enseñanza', lo cual tiene mucho sentido, dado que la 'Enseñanza' no depende de la estación, que es el factor que más peso tiene en 'Vestido y calzado'.

Bondad de los resultados

La puntuación Davies-Bouldin está implementada en la librería *sklearn*, por lo que será el que se usará en este ejemplo para evaluar la bondad del resultado obtenido.

```
from sklearn.metrics import davies_bouldin_score
```

Esta función necesita que los datos tengan las variables por filas, por lo que se pasan los conjuntos de datos traspuestas.

```
davies_bouldin_score(datos_P1.transpose(),
                     conglomerados_evolucion['Periodo1'])
```

```
0.3185558974655444
```

```
davies_bouldin_score(datos_P2.transpose(),
                     conglomerados_evolucion['Periodo2'])
```

```
0.44043915307706905
```

```
davies_bouldin_score(datos_P3.transpose(),
                     conglomerados_evolucion['Periodo3'])
```

```
0.5114479710989581
```

```
davies_bouldin_score(datos_P4.transpose(),
                     conglomerados_evolucion['Periodo4'])
```

```
0.37775160618684084
```

En definitiva, la mejor partición se obtiene para el periodo 1, porque por definición, cuanto más cercano a 0 es el índice, mayor es la homogeneidad dentro de los clusters y mayor distancia entre clusters distintos. Esto es: mejor es la clasificación.

Conclusiones

```
from IPython.display import Image
```

También es interesante ver las series temporales por conglomerado y periodo para entender mejor cuál ha sido el proceso de unión de las series.

```
# Se crea un array con los nombres los grupos ECOICOP.
ECOICOPdf = np.array(ECOICOP)
conglomerados1['Codigo'] = range(0,len(ECOICOP))
```

```
p = 1

for datos,cluster,name in [
    [datos_P1,conglomerados1,'Conglomerado1'],
     [datos_P2,conglomerados2,'Conglomerado2'],
      [datos_P3,conglomerados3,'Conglomerado3'],
       [datos_P4,conglomerados4,'Conglomerado4']]:

  k = 1
  fichero = 'ConglomeradosP'+str(p)

  fig = plt.figure(figsize=(30,17))
  fig.subplots_adjust(hspace=0.2, wspace=0.2)

  for c in range(n_states):
    title = 'Conglomerado '+str(c+1)
    idx = [i for i in range(cluster.shape[0]) if cluster[name][i] == c]
    ccaa0 = cluster[name][idx]
    ax = fig.add_subplot(2,2,k)
    x = range(0,datos.shape[0])
    ax.plot(x,datos[datos.columns[idx]])
    ax.set_ylim(40,130)
    ax.legend(ECOICOPdf[idx],fontsize=15)
    ax.grid(color='gray', linestyle='dashed', linewidth=1, alpha=0.4)
    ax.set_title(title,size='large',weight='extra bold',
                 stretch='extra-expanded',fontsize=25)
    plt.tick_params(labelsize = 12)

    k += 1

  p += 1

  imagen = ax.get_figure()
  ax.plot()
  plt.savefig(fichero, bbox_inches='tight')
```

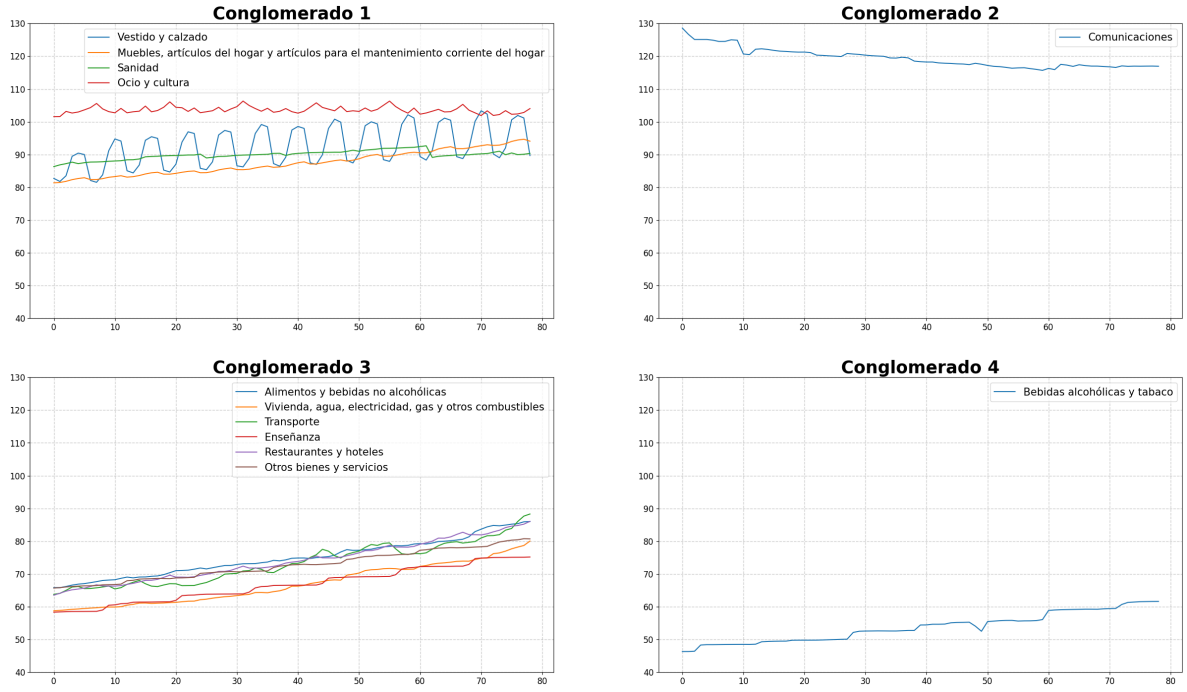
Los resultados del análisis revelan diversos hechos, aunque la primera observación es que, en general, son agrupaciones muy acertadas, al menos visualmente.

Resultados del periodo 1:

Los perfiles de las series de un mismo conglomerado son muy similares. Si bien es cierto que el tercer conglomerado y el primero del segundo periodo son agrupaciones de series visiblemente diferentes.

En el tercer conglomerado también se observa que la serie de 'Ocio y cultura' tiene un comportamiento muy diferente a las demás. Pero el dendograma de este periodo confirma que si se tomara un cluster más, aquí radicaría la división.

```
from PIL import Image
Image.open("/content/ConglomeradosP1.png")
```

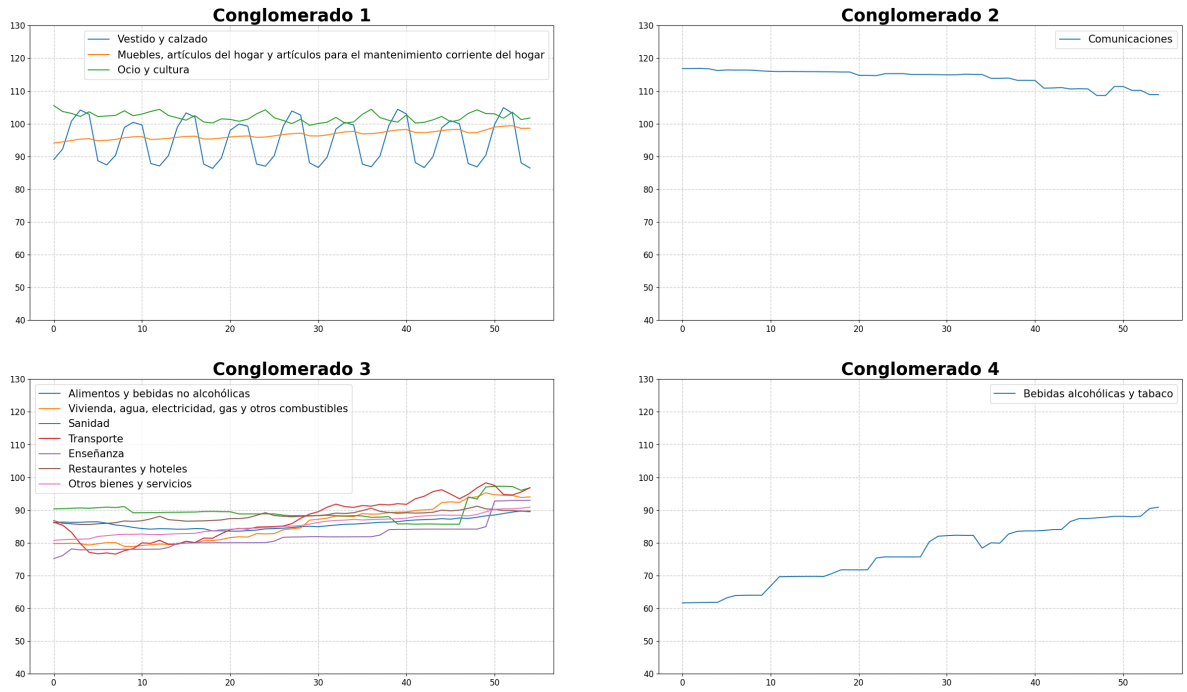


Resultados del periodo 2:

Los perfiles de las series de un mismo conglomerado son muy similares. Si bien es cierto que el primer conglomerado son agrupaciones de series visiblemente diferentes. La mayor cercanía puede deberse a la estacionalidad de las series.

Se observa que la única diferencia respecto del Periodo anterior es que la serie 'Sanidad' cambia de conglomerado, pero también puede deberse a que al final del periodo pierde estacionalidad.

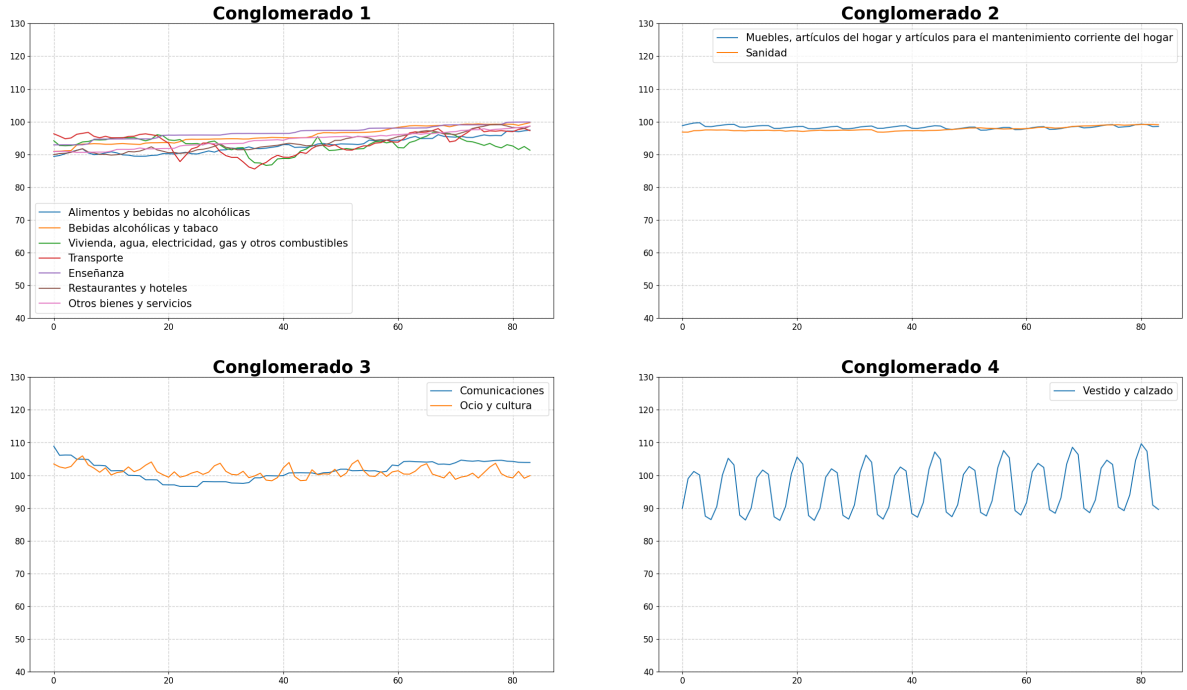
```
Image.open("/content/ConglomeradosP2.png")
```



Resultados del periodo 3:

En el conglomerado 1 hay dos series que parecen significantemente diferentes al resto: 'Vivienda, agua, electricidad, gas y otros combustibles' y 'Transporte'. Lo único que se puede prever es que al elegir cinco conglomerados en lugar de cuatro, probablemente esas dos series formarían el otro cluster. Efectivamente, volviendo al dendograma del tercer periodo, se observa que, de haber tomado cinco clusters, el primer conglomerado se dividiría en dos, uno de los cuales estaría formado por las series de 'Vivienda, agua, electricidad, gas y otros combustibles' y 'Transporte'.

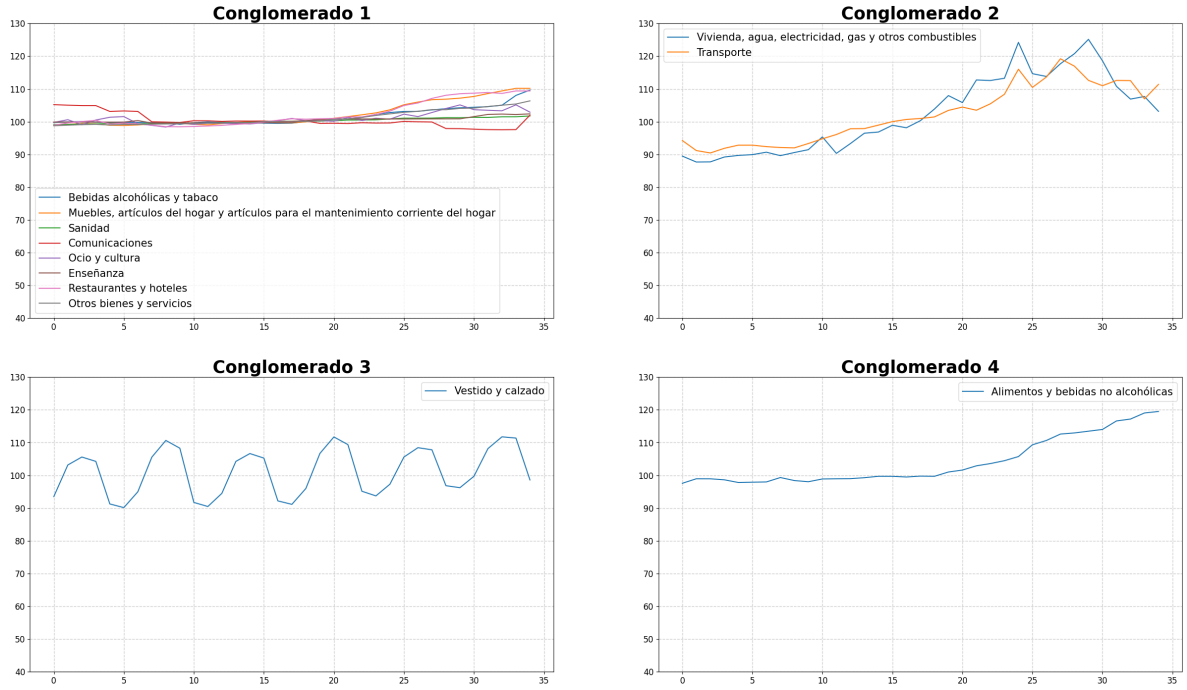
```
Image.open("/content/ConglomeradosP3.png")
```



Resultados del periodo 4:

En el conglomerado 1 ocurre lo mismo que en el periodo anterior: hay dos series que parecen significantemente diferentes al resto: 'Ocio y cultura' y 'Comunicaciones'. Lo único que se puede prever es que al elegir cinco conglomerados en lugar de cuatro, probablemente esas dos series formarían el otro cluster. Sin embargo, volviendo al dendograma del cuarto periodo, se observa que, de haber tomado cinco clusters, no hubiera habido la separación esperada.

```
Image.open("/content/ConglomeradosP4.png")
```



En conclusión, todos estos resultados deben mirarse bajo la óptica del dendograma, a través del cual se entiende de dónde vienen las agrupaciones y cuál seguiría siendo su evolución en el caso de tomar más conglomerados.

```
# Se restaura la configuración de advertencias
warnings.resetwarnings()
```

4.2.1. Más resultados

Para un mayor interés en todos los resultados posibles según las diferentes combinaciones de distancias y técnicas de clustering, se incluyen también, por orden de bondad de los resultados en base a la puntuación Davies-Bouldin, las tres mejores soluciones después de la ya mostrada.

El segundo mejor resultado se obtiene con la distancia entre observaciones y el método jerárquico aglomerativo. Ver imagen 4.1.

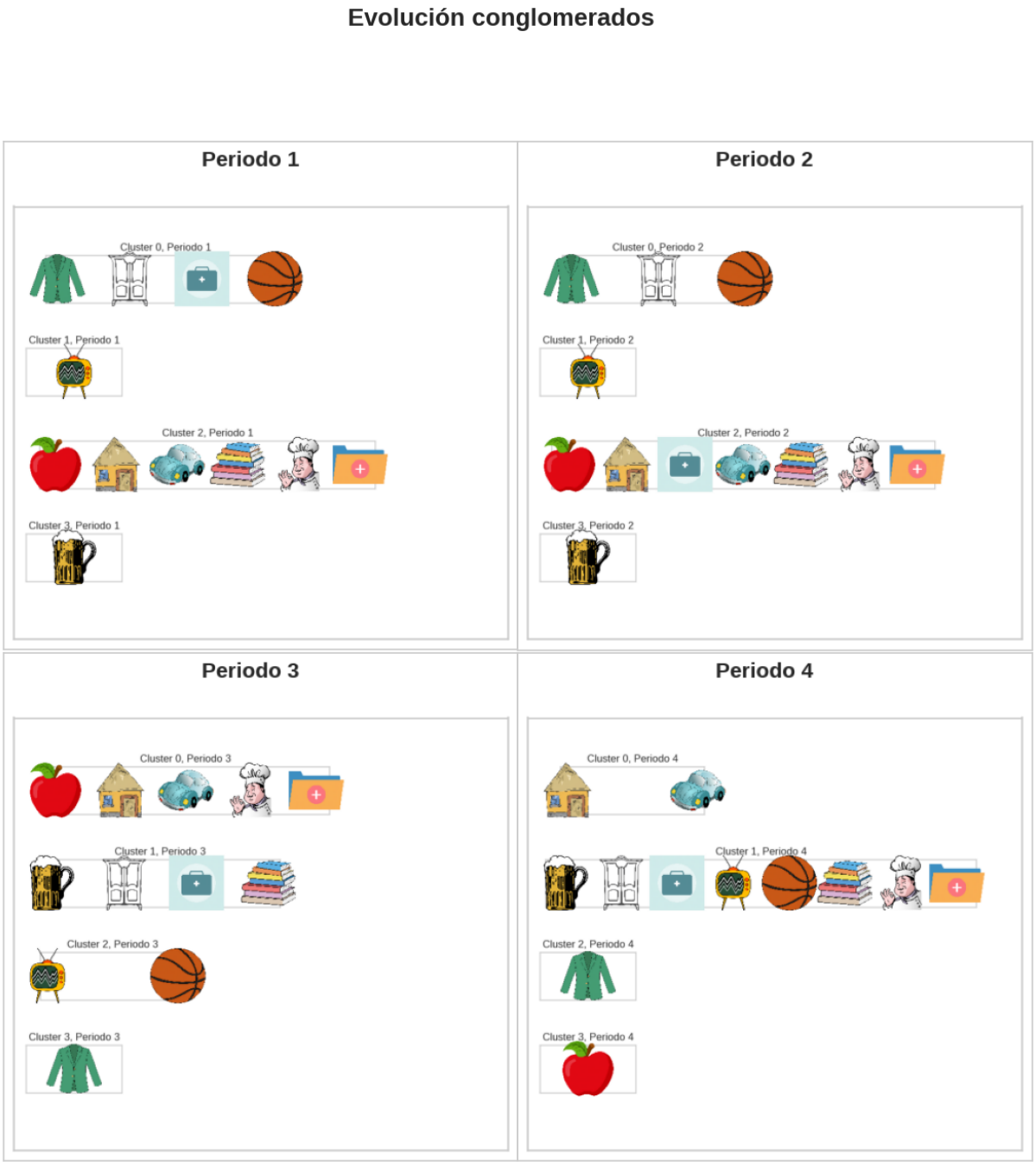


Figura 4.1: Resultados. Distancia Observaciones

El tercer mejor resultado se obtiene con la distancia basada en el error de predicción con modelos RLM y el método jerárquico aglomerativo. Ver imagen A.

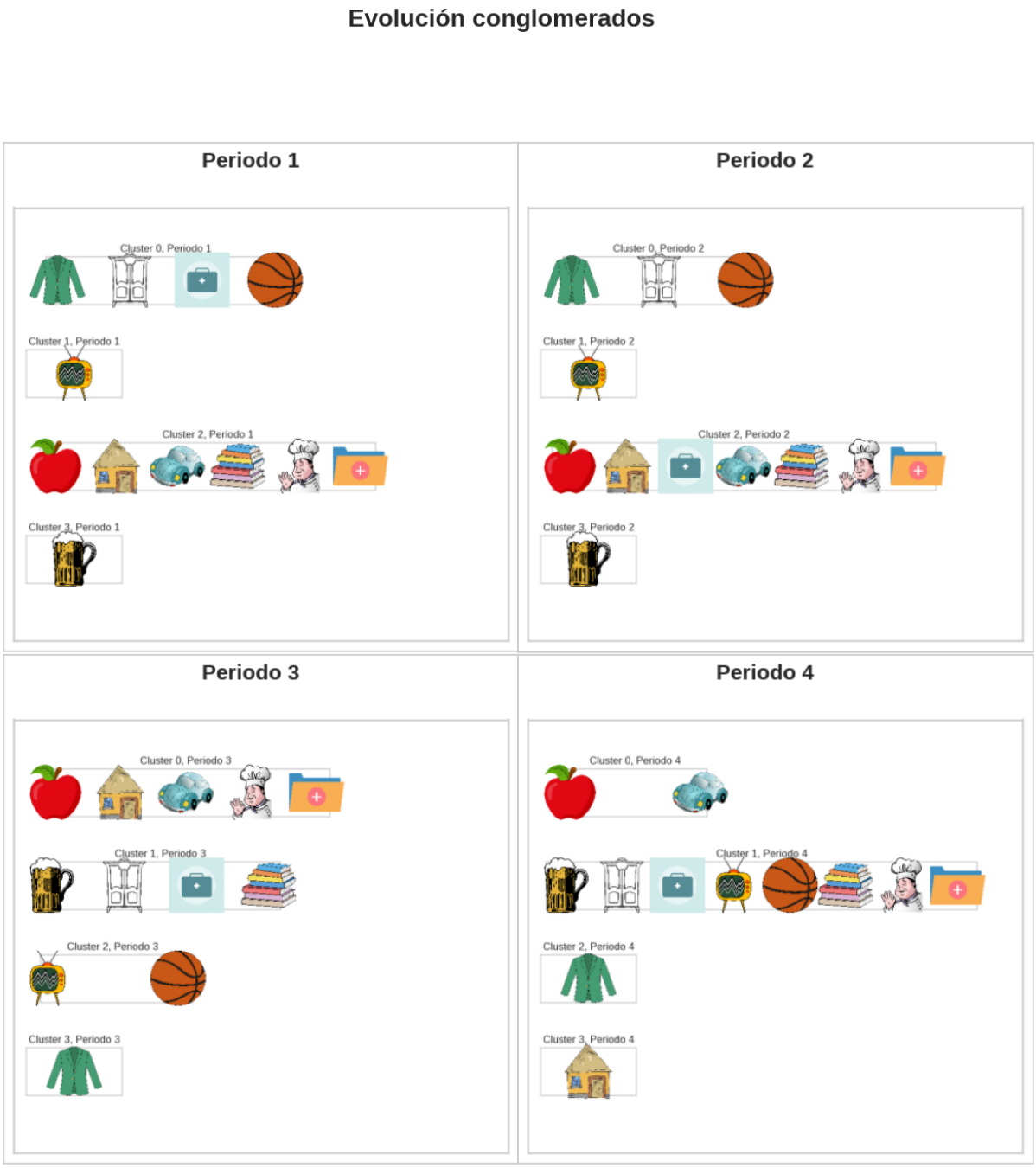


Figura 4.2: Resultados. Distancia Error RLM

Por último, el cuarto mejor resultado se obtiene con la distancia basada en el modelos de Markov. Ver imagen 4.3.

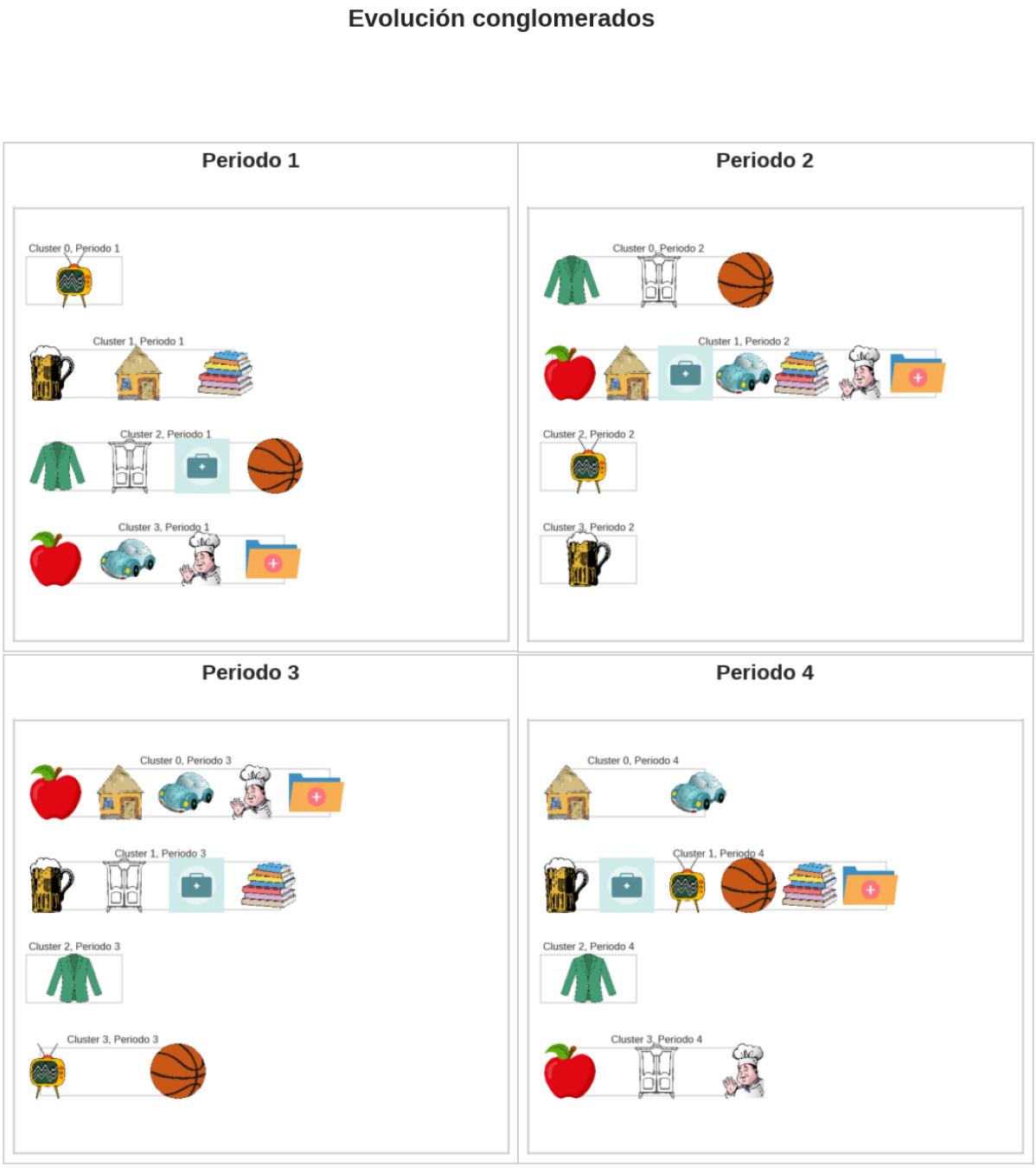


Figura 4.3: Resultados. Cadenas Markov

Bibliografía

[1] Alan J. Izenman (2008). *Modern Multivariate Statistical Techniques. Regression, Classification, and Manifold Learning.*

[2] Amélie Guilbaut & Asel Belotti. (2018). *Chaînes de Markov cachées. Université de Lille.*

[3] Andrés M. Alonso. (2019). *Time Series Clustering.*

[4] Aouragh, Salima. (2006). *Detection d'objets mobiles par les modèles de Markov cachés (MMCS). Masters thèsis, Université Mohamed Khider - Biskra.*

[5] Box, G.E.P., Jenkins, G.M., Reinsel, G.C. & Ljung, G.M. (2016). *Time Series Analysis. Forecasting and control.*

[6] Brijnesh J. Jain & David Schultz (2018). *Optimal Warping Paths are unique for almost every Pair of Time Series.*

[7] Cryer, J.D. & Chan K.S. (2008). *Time Series Analysis With Applications in R.*

[8] Dimitrios Kotsakos, Goce Trajcevski, Dimitrios Gunopulos, & Charu C. Aggarwal. (2014). *Data Clustering: Algorithms and Applications. Capítulo 15.*

[9] González Velasco, Miguel & Del Puerto García, Inés María (2012). *Series Temporales. Manuales Uex. Universidad de Extremadura*

[10] Javier Rivera Pinto (2014). *Clustering de series de tiempo con datos categóricos.*

[11] Kirchgässner, Wolters, J., & Hassler, U. (2013). *Introduction to Modern Time Series Analysis. (2nd ed. 2013.). Springer Berlin Heidelberg.*

[12] Jackeline Martínez Ponce. (2009). *Formación y Análisis de Conglomerados de Series de Tiempo. Caso Índice Nacional de Precios al Consumidor (Enero 1989-Febrero 2008).*

[13] Jorge Caiado, Elizabeth Ann Maharaj, & Pierpaolo D'Urso. (). *Handbook of Cluster Analysis. Capítulo 12.*

[14] Juan de Burgos Román (2010). *Álgebra básica (conjuntos y estructuras algebraicas): definiciones, teoremas y resultados .*

[15] Luis O. Manuel (2011). *Espacios métricos. Matemática Aplicada II. Licenciatura en Física, Universidad Nacional de Rosario*

[16] Nizar Bouguila & Wentao Fan & Manar Amayri (2022). *Hidden Markov Models and Applications.*

[17] Pablo Montero, & José A. Vilar. (2014). *TSclust: An R Package for Time Series Clustering.*

[18] Peter J. Brockwell & Richard A. Davis (1991). *Time Series: Theory and Methods.*

[19] Pierre Brémaud (2009). *Initiation aux Probabilités et aux chaînes de Markov.*

[20] Sarda-Espinosa A. (2023). *dtwclust: Time Series Clustering Along with Optimizations for the Dynamic Time Warping Distance. R package version 5.5.12*

[21] Seabold, Skipper & Josef Perktold (2010). *Statsmodels: Econometrics and statistical modeling with python. Proceedings of the 9th Python in Science Conference.*

[22] Shumway, R.H. & Stoffer, D.S. (2017). *Time Series Analysis and Its Applications. With R Examples.*

[23] T. Caliński & J Harabasz (1974). *A dendrite method for cluster analysis.*

[24] Weber, & Skillings, J. H. (2000). *A first course in the design of experiments a linear models approach. CRC Press.*

Apéndice A

Anexo

En el siguiente anexo se incluyen las hojas del Excel utilizadas para el ejemplo práctico de software disponible y el caso práctico 4.

Fecha	y1	y2	y3	y4	y5	y6	y7	y8
01/09/2020	0,659087	1,046467	-0,0732	-0,77019	-0,35616	1,356159	-0,22955	0,787612
01/10/2020	1,361943	1,116432	-0,0587	0,687942	0,006143	0,993857	-0,08458	-0,22278
01/11/2020	1,909412	0,410758	0,237154	0,199685	-0,81508	1,81508	0,576937	-0,6519
01/12/2020	2,418062	0,587727	1,042446	-0,13109	-0,00165	1,001646	-0,90528	0,24583
01/01/2021	2,251217	0,662077	0,491974	-0,71565	0,172409	0,827591	0,222277	0,204014
01/02/2021	2,264311	0,688024	0,900688	0,575536	-0,16141	1,161408	1,255112	-0,34972
01/03/2021	1,675574	0,95738	1,010365	-0,03344	0,564947	0,435053	-1,24035	-0,04189
01/04/2021	0,746829	0,483284	0,641708	-0,38317	0,106064	0,893936	0,440837	-0,40894
01/05/2021	1,43255	1,485273	0,969661	0,12486	-0,14041	1,140409	0,223363	-0,3892
01/06/2021	0,725398	1,159085	0,306312	-0,08333	0,599894	0,400106	-1,23262	0,622888
01/07/2021	1,356047	1,760488	-0,33794	0,773345	-0,14949	1,149494	-0,41455	0,320561
01/08/2021	-0,01522	1,385452	-0,95102	-0,506	-0,1517	1,151703	0,973109	-0,27074
01/09/2021	-0,06845	0,909661	0,014953	0,101677	-0,17662	1,17662	0,445115	-0,11965
01/10/2021	0,028749	0,871664	-0,89247	-0,29007	0,336758	0,663242	-1,63457	0,884001
01/11/2021	0,125946	0,810682	-0,9531	-0,0688	-0,32926	1,329264	0,921184	-0,61945
01/12/2021	0,806242	0,11117	0,219397	0,231802	0,440485	0,559515	-0,08141	0,394518
01/01/2022	0,067168	-0,14995	0,484185	-0,16678	-0,03918	1,039179	1,098927	0,027861
01/02/2022	0,661586	0,623391	0,464531	-0,15288	0,491493	0,508507	-0,41671	-0,6631
01/03/2022	0,488783	0,019304	-0,26978	-0,31824	0,060739	0,939261	-0,40921	0,034405

Figura A.1: Hoja Datos. Ejemplo software disponible

Fecha	Periodo	IPC_01	IPC_02	IPC_03	IPC_04	IPC_05	IPC_06	IPC_07	IPC_08	IPC_09	IPC_10	IPC_11	IPC_12
2002M01	1	65,859	46,340	82,717	58,763	81,364	86,313	63,805	128,593	101,553	58,352	63,523	65,705
2002M02	1	65,836	46,348	81,782	58,888	81,475	86,844	64,089	126,667	101,580	58,430	64,123	65,834
2002M03	1	66,185	46,436	83,519	59,041	81,819	87,210	65,007	125,098	103,157	58,496	64,756	66,045
2002M04	1	66,614	48,321	89,456	59,223	82,336	87,665	66,018	125,098	102,658	58,556	65,182	66,185
2002M05	1	66,902	48,446	90,427	59,324	82,686	87,212	66,191	125,098	102,968	58,563	65,416	66,299
2002M06	1	67,075	48,447	89,997	59,470	82,920	87,501	65,545	124,891	103,616	58,571	65,696	66,381
2002M07	1	67,354	48,475	82,087	59,607	82,358	87,700	65,585	124,500	104,308	58,587	66,184	66,410
2002M08	1	67,672	48,506	81,520	59,696	82,370	87,726	65,784	124,516	105,582	58,603	66,701	66,471
2002M09	1	68,028	48,515	83,801	59,863	82,643	87,790	66,044	125,004	103,921	59,018	66,362	66,693
2002M10	1	68,153	48,523	91,177	59,927	83,055	87,934	66,341	124,883	103,085	60,469	66,305	66,744
2002M11	1	68,252	48,530	94,748	59,916	83,269	88,028	65,445	120,675	102,725	60,587	66,308	66,790
2002M12	1	68,682	48,516	94,117	60,022	83,529	88,103	65,833	120,482	104,070	60,934	66,484	66,909
2003M01	1	69,057	48,580	85,042	60,487	83,100	88,385	66,841	122,128	102,730	61,016	66,834	67,953
2003M02	1	68,826	49,324	84,384	60,749	83,252	88,403	67,473	122,260	103,045	61,389	67,144	68,076
2003M03	1	69,071	49,428	86,807	61,123	83,588	88,641	68,040	122,056	103,215	61,430	67,575	68,442
2003M04	1	69,100	49,479	94,346	61,147	84,066	89,297	67,033	121,771	104,778	61,435	68,056	68,514
2003M05	1	69,292	49,511	95,431	61,021	84,423	89,395	66,299	121,545	103,031	61,449	68,080	68,576
2003M06	1	69,412	49,519	94,928	61,098	84,588	89,479	66,167	121,446	103,429	61,483	68,363	68,667
2003M07	1	69,801	49,785	85,249	61,157	84,007	89,602	66,650	121,326	104,420	61,507	68,974	68,613
2003M08	1	70,404	49,806	84,666	61,283	83,999	89,649	67,050	121,246	106,062	61,518	69,627	68,580
2003M09	1	71,018	49,808	87,041	61,359	84,255	89,700	67,008	121,260	104,401	61,959	69,048	68,800
2003M10	1	71,060	49,808	93,803	61,532	84,580	89,771	66,457	121,099	104,278	63,400	69,046	68,820
2003M11	1	71,162	49,819	96,925	61,702	84,853	89,856	66,485	120,319	103,147	63,535	68,991	68,894
2003M12	1	71,467	49,868	96,427	61,729	84,979	89,859	66,502	120,239	104,197	63,576	69,178	68,978
2004M01	1	71,827	49,937	85,789	62,157	84,443	90,115	66,987	120,119	102,754	63,759	69,572	70,199
2004M02	1	71,545	50,002	85,369	62,321	84,465	88,951	67,407	120,024	103,050	63,835	69,907	70,319
2004M03	1	71,906	50,070	87,782	62,625	84,793	89,142	68,140	119,892	103,358	63,850	70,273	70,442
2004M04	1	72,258	50,115	95,973	62,861	85,308	89,402	68,827	120,834	104,412	63,873	70,784	70,593
2004M05	1	72,582	52,177	97,355	63,073	85,639	89,444	69,959	120,666	103,032	63,885	70,796	70,671

Figura A.2: Hoja Datos. Caso Práctico

Variable	Descripción	Periodo	Unidades	Tipo de variable	Operación estadística	Observaciones	Enlace página web INE
Fecha	Fecha del dato		añoMmes	Fecha			
Periodo	Periodo en el que lo clasificamos		{1,2,3,4}	Cualitativa			
IPC_01	Alimentos y bebidas no alcohólicas	Enero 2002 - enero 2023	Índice	Cuantitativa continua	30183 Indicadores de Actividad del Sector Servicios	El Índice de Precios de Consumo (IPC) que se publica mensualmente, tiene como objetivo medir la evolución del nivel de precios de los bienes y servicios de consumo adquiridos por los hogares residentes en España.	https://ine.es/jaxiT3/Tabla.htm?t=50902&L=0
IPC_02	Bebidas alcohólicas y tabaco						
IPC_03	Vestido y calzado						
IPC_04	Vivienda, agua, electricidad, gas y otros combustibles						
IPC_05	Muebles, artículos del hogar y artículos para el mantenimiento corriente del hogar						
IPC_06	Sanidad						
IPC_07	Transporte						
IPC_08	Comunicaciones						
IPC_09	Ocio y cultura						
IPC_10	Enseñanza						
IPC_11	Restaurantes y hoteles						
IPC_12	Otros bienes y servicios						

Figura A.3: Hoja Metadatos. Caso Práctico