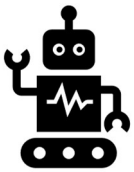
TRABAJO

APRENDIZAJE AUTOMÁTICO



Boyuan Chen (b.chen@alumnos.upm.es)

Álvaro Huisman(alvaro.huisman@alumnos.upm.es)

DATASET: AO5.csv

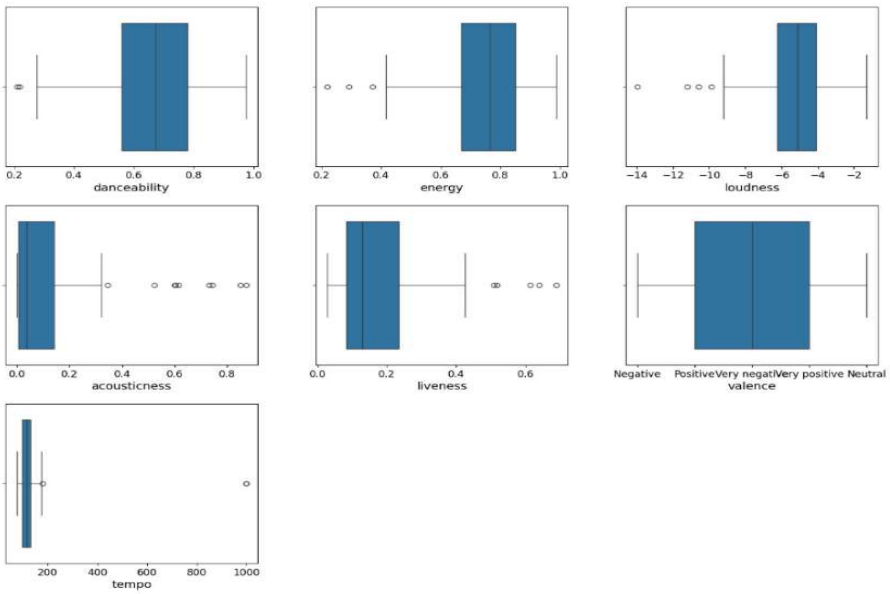
1.Introducción:

El objetivo de este proyecto es descubrir agrupaciones significativas en un conjunto de datos de canciones, utilizando varias técnicas de clustering como K-means, Clustering Jerárquico o DBSCAN. Este proceso nos permitirá identificar grupos de canciones que comparten características similares, y de este modo, obtener información relevante que puede ser útil para diferentes propósitos, como análisis de tendencias musicales o recomendaciones automáticas.

El conjunto de datos contiene varias características relacionadas con las canciones, que son fundamentales para el proceso de segmentación:

- **name y artists**: Nombre de la canción y artista.
- **danceability**: Mide cuán adecuada es para bailar (rango de 0.0 a 1.0).
- **energy**: Indica la intensidad y nivel de actividad de la canción.
- **loudness**: Volumen general en decibelios.
- **acousticness**: Probabilidad de que la canción sea acústica.
- **liveness**: Indica si fue grabada en vivo.
- **valence**: Refleja la positividad o negatividad emocional de la canción.
- **tempo**: Velocidad de la canción medida en beats por minuto (BPM).

Para un entendimiento más en detalle de las características haremos unos boxplots de cada característica individualmente



Podemos ver la existencia de outliers y las diferentes escalas que tienen los boxplots(Trataremos estos problemas más adelante)

2.Metodología:

2.1 Preprocesamiento

Antes de aplicar los algoritmos de clustering, el conjunto de datos requiere un preprocesamiento para manejar los valores faltantes y normalizar las características numéricas. Se realizaron los siguientes pasos:

Tratamiento de outliers

Como se observó en los boxplots, algunas características presentan datos atípicos. Inicialmente, pensamos en tratar estos outliers mediante escalado o utilizando algoritmos de clustering robustos. Incluso consideramos eliminar directamente las filas afectadas. Sin embargo, el tempo mostró valores atípicos demasiado extremos, lo que nos hizo sospechar de errores de transcripción.

Dado que no conocemos en profundidad todos los detalles del conjunto de datos, decidimos acudir a una fuente confiable en la industria musical como Spotify. Utilizamos su API para verificar las características de las canciones con datos atípicos, y confirmamos que algunos de estos valores efectivamente correspondían a errores de transcripción. Este enfoque nos permitió asegurar que los datos corregidos provinieran de una "autoridad" en la música, lo que otorga mayor confianza en la validez de los ajustes realizados.

```
import spotipy
from spotipy.oauth2 import SpotifyOAuth

sp = spotipy.Spotify(auth_manager=SpotifyOAuth(client_id="",
                                               client_secret="",
                                               redirect_uri="http://localhost:8888/callback",
                                               scope="user-library-read"))

# Track ID of the song you're interested in
track_id = "4LwU4Vp6od3Sb08CsP99GC"  # The Next episode, tempo: 95.295
# track_id = '4bJygwUKrRgq1stlNXcgMg' #All the things she said, tempo: 179.92

# Fetch the audio features of the track
audio_features = sp.audio_features(track_id)

# Extract and print the tempo
print(audio_features)
```

```
'tempo': 95.295, 'type': 'audio_features', 'id': '4LwU4Vp6od3Sb08CsP99GC', 'uri': 'spotify:track:4LwU4Vp6od3Sb08CsP99GC'
```

Además, una vez que obtuvimos los datos reales, consideramos reemplazar los valores antiguos con los nuevos corregidos. También verificamos si otras características presentaban errores de transcripción similares, pero no encontramos más casos relevantes.

Tratamiento de NaN

Al examinar nuestros datos vimos que existían algunas columnas numéricas con NaN. De esta manera, Rellenamos los NaN con las medias de estas columnas (en un principio pensamos en usar las medianas, pero después del tratamiento de los outliers más pronunciados, pensamos que esto ya no era necesario).

Escalado

Como mencionamos anteriormente, en nuestro dataset existen características numéricas como 'bailabilidad', 'energía', 'loudness', etc., que se encuentran en escalas diferentes, esto resultaría en que las diferentes variables tendrían diferentes pesos. Para evitar este problema y asegurar que todas las características contribuyan equitativamente al proceso de clustering, hacemos un escalado de los datos. En nuestro caso hemos decidido usar MaxAbsScaler, ya que creemos que MaxAbsScaler es más adecuado que otros scalers ya que en nuestro conjunto de características los datos contienen valores positivos y negativos.

Tratamiento de columnas categóricas:

En este estudio, hemos decidido no utilizar las columnas categóricas como 'name' y 'artists', ya que (aunque se podría aplicar alguna transformación) creemos que no son directamente útiles para el análisis de nuestro clustering. En cuanto a la columna categórica ‘valence’ hemos optado por utilizar OrdinalEncoder ya que esta representa un atributo con un orden lógico ("Very negative", "Negative", "Positive"). Creemos que OrdinalEncoder es más adecuado que otros encoders como OneHotEncoder porque respeta esa jerarquía natural de orden, lo que permite que el algoritmo de clustering entienda mejor las relaciones entre las categorías. Además, evita la creación de múltiples columnas innecesarias como ocurriría con OneHotEncoder, haciendo el proceso más eficiente y sencillo.

2.2 PROCEDIMIENTO

Utilizamos primeramente el Ordinal Encoder para transformar nuestras instancias categóricas a numéricas (esta columna no tiene ningún valor vacío y por lo tanto no es afectado por un imputer), posteriormente aplicamos un **pipeline** en el que tratamos los valores NaN y escalamos:

```
# Definimos el pipeline solo con los pasos de preprocesamiento
pipe = make_pipeline(
    SimpleImputer(strategy='mean'),  # Imputación con la media
    MaxAbsScaler()  # Escalado MaxAbsScaler
)
```

```
X = df[['danceability', 'energy', 'loudness', 'acousticness', 'liveness', 'valence', 'tempo']]

# Cambiamos de variables categóricas a numéricas
ordinal_encoder = OrdinalEncoder(categories=[['Very negative', 'Negative', 'Neutral', 'Positive', 'Very positive']])
X.loc[:, 'valence'] = ordinal_encoder.fit_transform(X[['valence']])

# Aplicamos el pipeline de preprocesamiento
X = pipe.fit_transform(X)
```

2.3 Algoritmos de Clustering

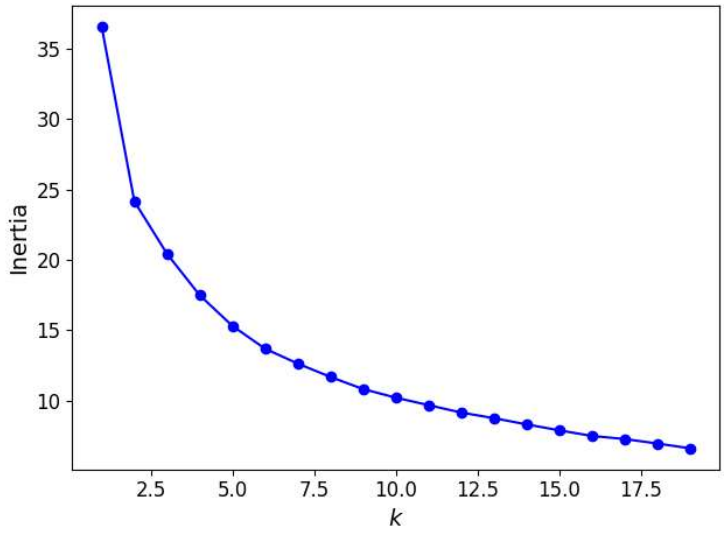
Una vez preprocesados los datos, procedemos al clustering, para ello analizamos una serie de algoritmos.

K-means:

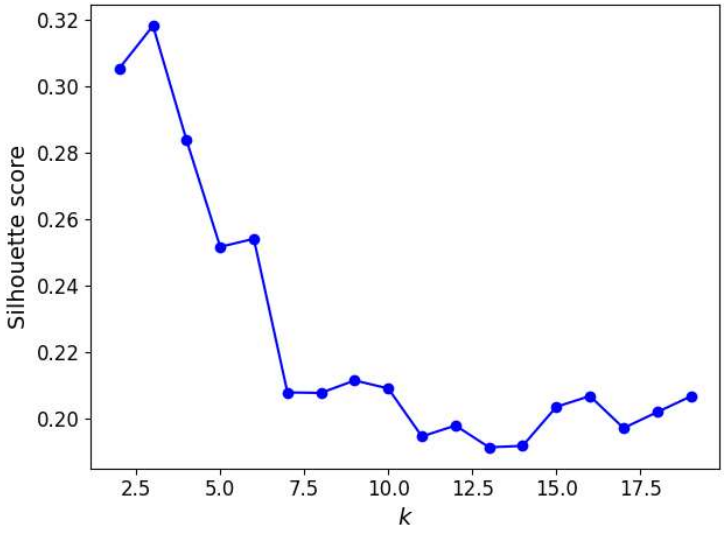
El algoritmo K-means funciona asignando cada punto de datos al cluster cuyo centroide está más cercano y luego recalcula los centroides de cada cluster. Este proceso se repite hasta que los centroides dejan de cambiar o hasta alcanzar un número máximo de iteraciones. El objetivo de K-means es minimizar la distancia dentro de los clusters y maximizar la diferencia entre los distintos clusters.

Uno de los aspectos más importantes en K-means es la elección del número de clusters. Esta elección afecta directamente la calidad del agrupamiento y los patrones que se pueden identificar en los datos.

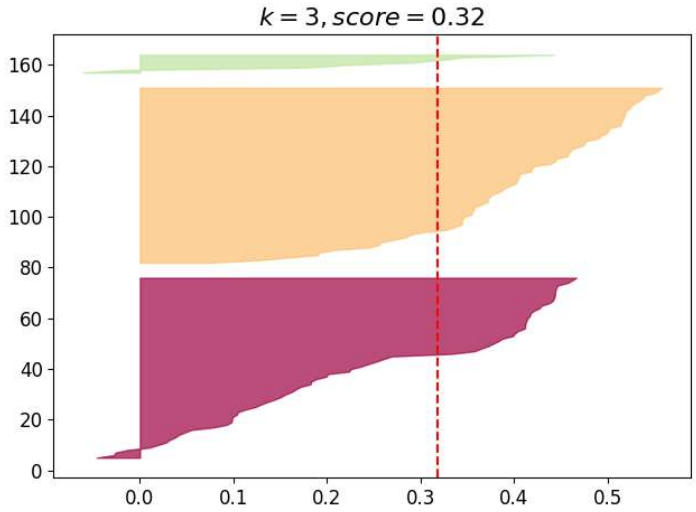
Para garantizar una elección óptima del número de clusters hemos llevado a cabo una exploración. Primero, hemos observado la **inertia**, que mide la calidad del clustering y consiste en la suma de las distancias cuadradas entre cada punto de datos y el centroide del cluster al que pertenece. Una vez graficada la inercia, aplicamos la regla que nos indica que el número óptimo de clusters en nuestro caso está entre K = 2 y K = 3.



Para verificar nuestra elección realizamos el **silhouette score** el cual evalúa la calidad de los clusters. Este score mide qué tan bien están agrupados los puntos dentro de un cluster y qué tan separados están de otros clusters, su valor varía entre -1 y 1. En nuestros datos para los distintos K obtuvimos unos resultados altos en K =2 y K = 3, optamos por K = 3 ya que pensamos que aumenta la calidad de los clusters sin perder la simplicidad (ya que solo aumenta en 1 el número de clusters) y además creemos que existen demasiadas variables en el conjunto como para limitarlo al análisis de 2 clusters.



Teniendo en cuenta, la elección de K = 3, verificamos los clusters formados con su **silhouette score**:



A pesar de haber realizado el preprocesamiento adecuado y explorado el número de clusters, observamos que el silhouette score obtenido con K = 3 no es el ideal, con un valor de 0.32. Esto sugiere que podría haber una alta variabilidad en los datos, lo que dificulta una clara separación entre los clusters.

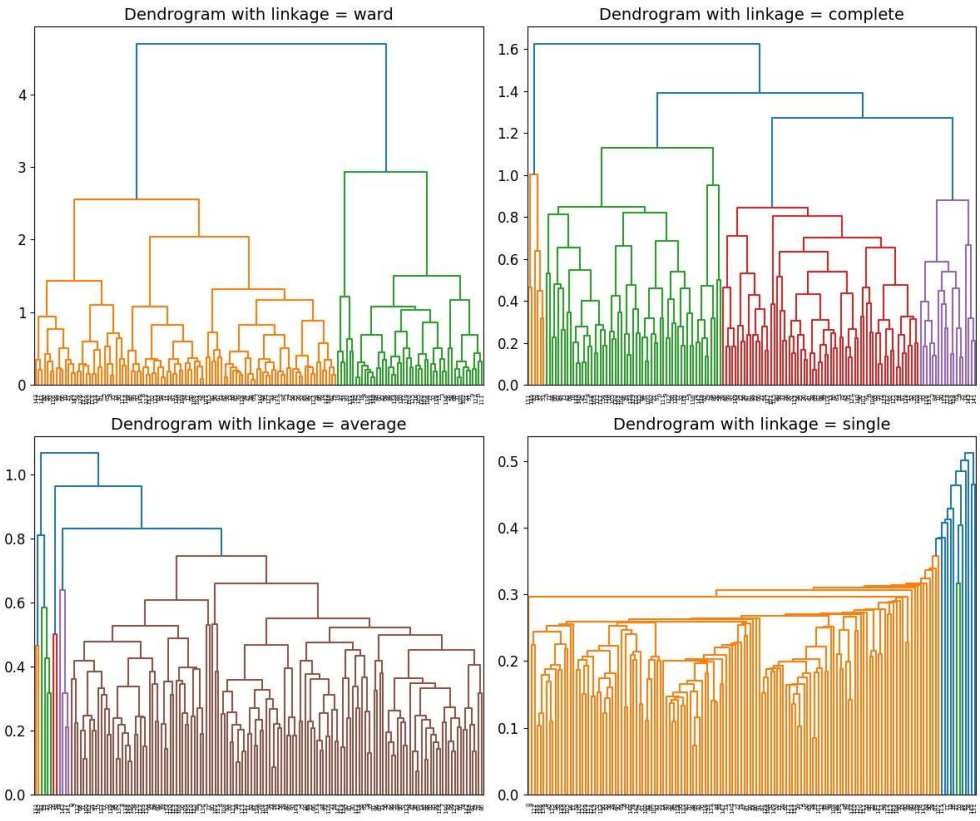
Hierarchical clustering:

El **clustering jerárquico aglomerativo** es un algoritmo el cuál comienza con todos los puntos como un cluster individual y va fusionando los clusters más cercanos segun distintas **métricas de distancia** (single linkage, ward method, complete linkage o average linkage), este proceso se repite hasta formar un único cluster. El resultado final es una estructura llamada **dendrograma** el cual muestra las fusiones de los clusters en cada paso, este dendrograma se puede cortar en distintos niveles para obtener finalmente un número de clusters óptimo. Las distintas métricas de distancia mencionadas anteriormente consisten en:

- **Single Linkage**: La distancia mínima entre puntos de diferentes clusters.
- **Complete Linkage**: La distancia máxima entre puntos de diferentes clusters.
- **Average Linkage**: El promedio de las distancias entre todos los puntos de dos clusters.
- **Ward's Method**: Minimiza el incremento en la varianza dentro de los clusters al fusionar clusters.

Para llevar a cabo la mejor decisión para inicializar el linkage y la métrica de afinidad hemos hecho una serie de dendrogramas.

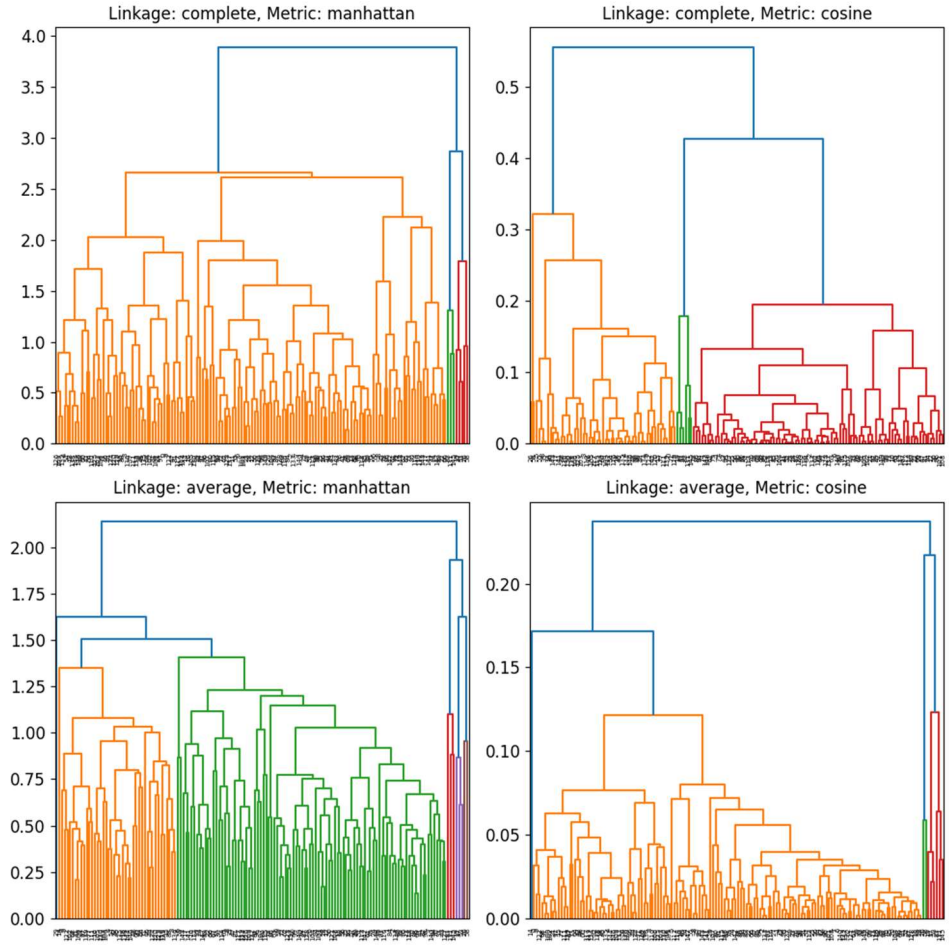
Primero hemos graficado unos dendrogramas con diferentes linkage y con la métrica de afinidad ‘euclidean’ (ya que Ward solo funciona con esta)



Si observamos los gráficos, podemos ver que tanto utilizando linkage ‘ward’ como ‘complete’ dan resultados buenos. Sin embargo, creemos que es más interesante el dendrograma con 4 clusters usando complete linkage. En este se observa una mayor flexibilidad en la agrupación, lo que permite identificar más claramente las diferencias en las características musicales a lo largo del eje y.

Además, podemos observar que tiene una varianza cercana a 1, lo que indica que las canciones dentro de un cluster tienen variabilidad controlada, es decir, las canciones son similares entre sí, pero no idénticas. Esto nos parece interesante, porque trabajamos con bastantes variables y no queremos forzar agrupamientos en solo 2 bloques grandes (pueden llegar a ser demasiado generales)

A continuación, vamos a explorar las diferentes métricas de afinidad. En este caso, solo analizaremos los métodos de enlace average y complete, ya que consideramos que el enlace simple puede generar clusters demasiado dispersos y poco compactos.

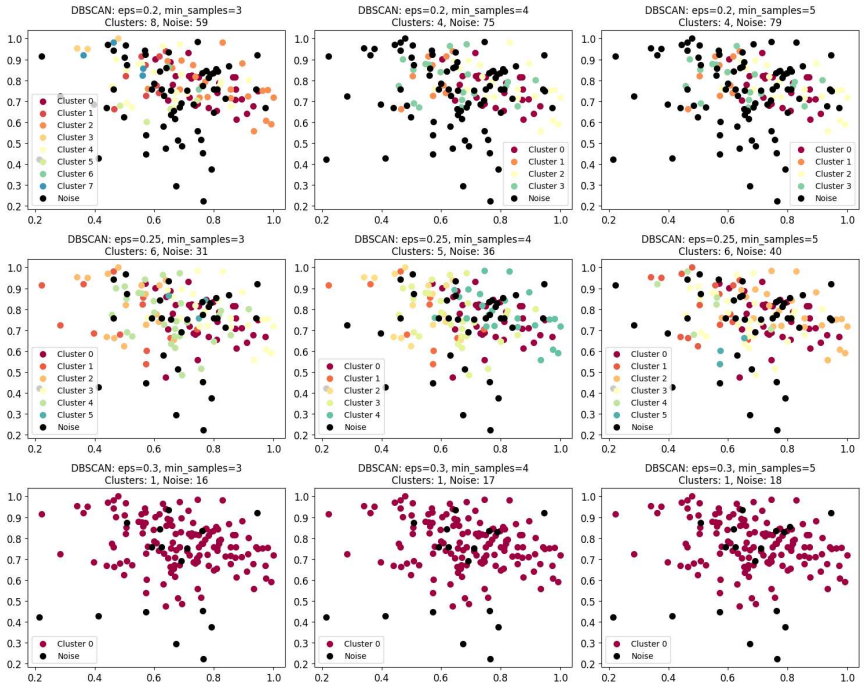


Tras ver estos gráficos, seguimos manteniendo que la mejor opción sería quedarnos con el enlace *Complete* con la métrica Euclidiana. La mayor diferencia con otras métricas es que la Euclidiana produce clusters más equilibrados, lo cual creemos que es importante en nuestro estudio, ya que no queremos clusters demasiado pequeños o sobrepoblados.

DBSCAN:

Es un algoritmo de clustering que agrupa puntos en función de la densidad de datos en el espacio, este algoritmo no requiere especificar un número de clusters, ya que funciona identificando regiones densas, conectando puntos cercanos, y separando áreas de baja densidad como el ruido. Es útil para descubrir clusters de formas arbitrarias y manejar bien los puntos atípicos.

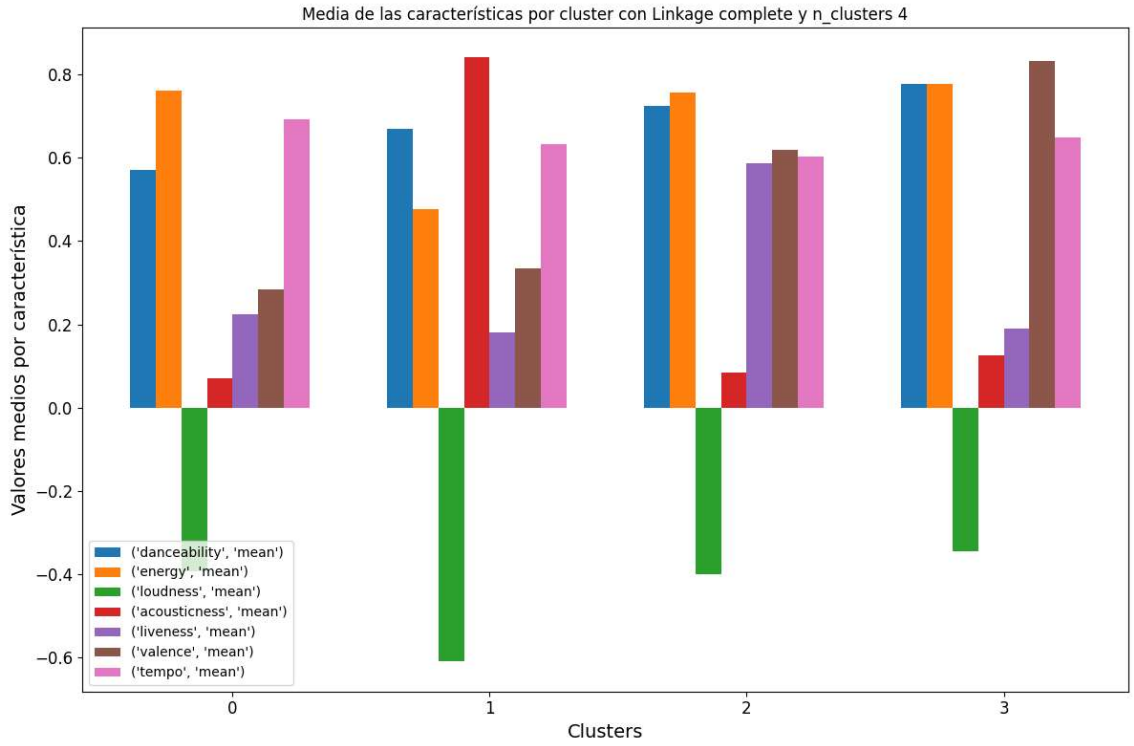
En nuestros datos hemos ejecutado un bucle que prueba distintas distancias “eps” y distintos “min_samples” pero no hemos obtenido ningún clustering bueno, creemos que esto es debido a que nuestros clusters tenían una densidad variable y estaban solapados y por lo tanto no pudimos identificar ninguna métrica buena. Esto lo pudimos verificar después con el silhouette score que daba unas puntuaciones extremadamente bajas a nuestros clusters realizado con DBSCAN.



Resultados:

Tras analizar los diferentes tipos de clustering y observar los resultados que ofrecen, hemos optado por realizar un análisis de clustering jerárquico aglomerativo con un enlace completo y afinidad euclidiana, utilizando 4 clusters.

Para evaluar los resultados obtenidos con el clustering elegido, hemos realizado un análisis de los diferentes clusters formados. Para ello, hemos graficado las medias de cada característica en cada uno de los agrupamientos obtenidos.



Vamos a desglosar poco a poco los clusters.

En el Cluster 0, las canciones tienen una valencia baja, lo que indica un tono más melancólico o triste. Además, presentan un acousticness bajo, con pocos elementos acústicos, y una danceability moderada.

En el Cluster 1, predominan canciones con un acousticness elevado, lo que implica una mayor presencia de instrumentos acústicos. Tienen un loudness y una energy bajos, lo que las convierte en canciones suaves y tranquilas.

En el Cluster 2, las canciones muestran una valencia más alta, lo que sugiere un tono más positivo. Además, destacan por un liveness elevado, lo que podría indicar una sensación más cercana a actuaciones en vivo.

Finalmente, en el Cluster 3, se agrupan las canciones más alegres y energéticas. Son las que presentan el mayor loudness, así como altos niveles de danceability y energy, lo que las hace dinámicas y bailables. Sin embargo, tienen un liveness más bajo, lo que sugiere una producción más de estudio.

En general, la principal diferencia entre los clusters es la valencia. Esta característica parece ser central y arrastra consigo una serie de propiedades asociadas. Por ejemplo, en el Cluster 3 se encuentran las canciones más alegres, lo que también las hace más bailables y enérgicas. En cambio, el Cluster 1 agrupa las canciones más tristes, que suelen ser más acústicas y tranquilas.

Por supuesto, esto es una interpretación algo subjetiva y puede no darse siempre.

Para comprobar la veracidad de la tabla mostrada, vamos a hacer una pequeña investigación sobre los clusters formados. En este caso hemos analizado un artista (‘Linkin Park’) y miramos en qué cluster se han agrupado sus canciones

	name	artists	danceability	energy	loudness	acousticness	liveness	valence	tempo	cluster
4	In the End	Linkin Park	0.556	0.864	-5.870	0.00958	0.209	Negative	105.143	0
14	Numb	Linkin Park	0.496	0.863	-4.153	0.00460	0.639	Negative	110.018	2
45	What I've Done	Linkin Park	0.623	0.930	-5.285	0.01410	0.138	Negative	120.119	0

Aquí podemos ver que las canciones de “Linkin Park” se encuentran en dos clusters el 0 y el 2, en el cual observando las columnas vemos una notable diferencia en el valor del liveness que en la instancia del cluster 2 es más que el triple que de las otras instancias, al observar nuestras medias de características podemos efectivamente observar que este valor varía entre los dos clusters significativamente.

Extras

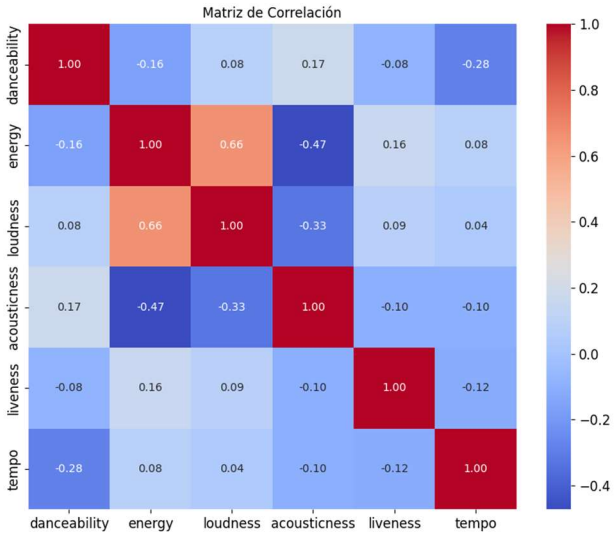
Hemos querido probar unas cosas para hacer más interesante el trabajo.

PCA

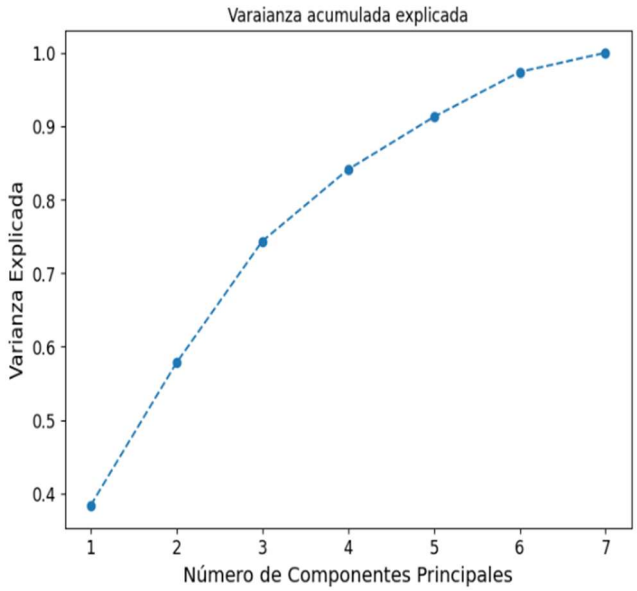
Uno de los principales problemas para clasificar en clusters es que no contamos con suficientes datos para el número de variables que tenemos, lo que complica el análisis. Para abordar esto, decidimos estudiar la posibilidad de reducir la dimensionalidad usando PCA (Análisis de Componentes Principales).

PCA es una técnica que transforma las variables originales en un conjunto más pequeño de componentes principales que capturan la mayor parte de la varianza, facilitando el análisis sin perder demasiada información.

Primero de todo para comprobar si tiene sentido hacer PCA hacemos una tabla de correlación entre variables:



Vemos que no hay demasiada correlación generalmente hablando, pero sí existe algo de correlación entre energy y loudness. Por tanto, vamos a probar a ver si PCA es capaz de crear pocas variables que expliquen una gran cantidad de la variabilidad de los datos originales.



Podemos observar que se necesitan entre 5/7 componentes principales para explicar el 90% de la variabilidad de las variables originales. Por lo tanto, realizar PCA en este caso no parece muy eficiente, ya que no reduciría significativamente la dimensionalidad. Además, un inconveniente de aplicar PCA es que los resultados tienden a ser más difíciles de interpretar, ya que las variables originales pierden parte de su explicabilidad.

Predicción

Una vez que tenemos los datos agrupados en clusters (cada dato tiene un label asociado), podemos hacer un entrenamiento supervisado de los datos para poder clasificar futuras canciones.

Como ejemplo, vamos a poner la canción 'Happy' de Pharrell Williams. Para ello, volvemos a usar la Spotify api, mencionada en el principio del documento, para sacar las características de esta canción.

```
cluster_predicho = knn.predict(nueva_cancion_escalada)
print(f"La nueva canción pertenece al cluster: {cluster_predicho[0]}")
```

```
La nueva canción pertenece al cluster: 3
```

Conclusión

Podemos concluir, que a pesar de haber recibido un dataset con mucha variabilidad en los datos y muchos datos desconocidos, hemos podido tratarlo de forma adecuada evaluando distintas métricas de cluster como K-Means, Clustering Jerárquico y DBSCAN y obteniendo así unos resultados relativamente coherentes teniendo en cuenta la alta dimensionalidad de las variables. Además, hemos experimentado distintos enfoques en cuanto al preprocesado. También hemos hecho uso de distintas métricas de evaluación para los distintos clusters, optimizando así los parámetros que influyen en la creación de dichos clusters. Finalmente, hemos sabido interpretar los resultados obteniendo así unas conclusiones razonables y hemos optado por hacer una pequeña indagación para sacar provecho al trabajo realizado (predicción).

Posible mejora:

> Una mejora posible sería incluir la columna 'artists' en el clustering. Esto permitiría generar agrupaciones no solo en función de las características musicales, sino también de los patrones relacionados con los artistas. Al usar OneHotEncoder para 'artists', podríamos ofrecer recomendaciones más personalizadas, sugiriendo canciones de artistas similares o basadas en las preferencias de los usuarios, combinando tanto el estilo musical(determinado por las variables del estudio) como el artista.