

Análisis y predicción de la popularidad de canciones en base a sus características

Pablo Ariño y Álvaro Laguna

Introducción

El objetivo del proyecto es predecir si una canción va a ser popular o no (en una escala de 0 a 100) en base a sus atributos. Por tanto, se trata de un problema de regresión.

Investigando un poco cómo se ha obtenido el dataset, hemos averiguado que probablemente el dataset venga de haber utilizado la API de Spotify. En el siguiente enlace viene una descripción con cada una de las columnas:

<https://developer.spotify.com/documentation/web-api/reference/#/operations/get-audio-features>

Nota: todo el código utilizado en este proyecto se puede encontrar en el siguiente repositorio de Github:

https://github.com/Pabloo22/song_popularity_mcp.

Este repositorio se ha hecho público a las 21:14 del miércoles para no revelar información al resto de participantes durante el transcurso de la competición.

Análisis exploratorio de los datos

Resumen de las columnas

`song_id`. No presenta valores duplicados. Es un identificador, por lo que **no se utilizará para la predicción**.

`song_name`. Existen varias canciones con un nombre de la canción repetido. No obstante, tal y como están los datos no es posible realizar un one-hot de la variable, ya que el nombre de canción más repetido es 'Better' con 16 apariciones. Si hiciésemos un one-hot todas las variables entrarían dentro de la clase 'infrecuente'. Por otro lado, un mapeo por frecuencia podría ser útil pero dejaría todos los datos muy cercanos a cero. Sería por tanto necesario algún tipo de escalado. Además de esto, quizás sería posible realizar un análisis de sentimientos sobre el título. Por otro lado, quizás también se puede **extraer información adicional como si la canción es una colaboración con otro artista**.

`song_popularity`. Esta es nuestra variable objetivo. Está centrada en 60 y tiene un pico en el valor cero.

`song_duration_ms`. Parece seguir una distribución gaussiana.

`acousticness`. Se encuentra en el rango [0, 1]. La mayoría de valores se agrupan cercanos a cero. Es decir, la mayoría de canciones no son acústicas.

`danceability`. "*Danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable*".

Se puede apreciar que la mayoría de canciones (>50%) se encuentran en torno a 0.5 y 0.75.

`energy`. "*Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale. Perceptual features contributing to this attribute include dynamic range, perceived loudness, timbre, onset rate, and general entropy*".

Una característica que no tenemos del dataset, es el género de la canción. No obstante, es razonable pensar que la probabilidad de que una canción sea popular depende en gran medida de su género. Hay géneros más populares que otros y, por tanto, las canciones más populares suelen serlo de los géneros más populares también. Si bien esta característica no indica a qué género pertenece la canción, podría dar muchas pistas sobre ello, especialmente si se combina con otras características.

`instrumentalness`. "Predicts whether a track contains no vocals. "Ooh" and "aah" sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly "vocal". The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content. **Values above 0.5 are intended to represent instrumental tracks**, but confidence is higher as the value approaches 1.0".

Por lo que se indica en la descripción, **podría ser interesante crear una columna binaria que redondee el valor a 0 o a 1.**

`key`. "The key the track is in. **Integers map to pitches** using standard Pitch Class notation. E.g. 0 = C, 1 = C#/D♭, 2 = D, and so on. If no key was detected, the value is -1".

Se encuentra en el rango 0-11 en nuestro dataset aunque valores de -1 serían posibles. Esta variable parece seguir una distribución relativamente uniforme.

`liveness`. "Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live. **A value above 0.8 provides strong likelihood that the track is live**".

La mayoría de valores sin embargo (75%) se agrupa entre 0.09 y 0.22. Hay muy pocas canciones (< 5%) que superen ese valor de 0.8 por lo que realizar esa categorización binaria podría no ser de mucha utilidad. A modo de reflexión adicional, si el objetivo fuese predecir si una canción va a ser popular o no antes de que salga, esta variable probablemente no debería de utilizarse para predecir la popularidad. Esto es porque a la hora de grabar una canción por primera vez es muy complicado que se haya grabado en un concierto en vivo. No obstante, otro posible objetivo es predecir la popularidad de una versión de la canción en cuestión para saber si merece o no la pena subir a Spotify la canción grabada en

un concierto. Sería interesante comprobar si las canciones con un *liveness* de más de 0.8 se encuentra el nombre de la canción repetida.

`loudness`. "The overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing relative loudness of tracks. **Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude)**.** Values typically range between -60 and 0 db".

En la muestra que tenemos nosotros, **el rango se encuentra entre 0 y -38.7**.

`audio_mode`. "Mode indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived. Major is represented by 1 and minor is 0".

El 37% de los datos de entrenamiento son cero.

`speechiness`. "Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.33 most likely represent music and other non-speech-like tracks".

El 95% de los datos se encuentran por debajo de 0.33. Esto tiene sentido puesto que se trata de canciones.

`tempo`. "The overall **estimated tempo of a track in beats per minute (BPM)**. In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration".

El rango de esta variables en nuestro dataset va entre 0 y 214.686. La media se encuentra en 121 y es muy cercana a la mediana (120). La desviación típica es de 28.

`time_signature`. "An estimated time signature. The time signature (meter) is a notational convention to specify how many beats are in each bar (or measure). The time signature ranges from 3 to 7 indicating time signatures of "3/4", to "7/4" ".

El 94% de las canciones tienen un compás de 4/4. Por tanto, **esta columna tampoco aporta mucha información por lo que se puede eliminar.**

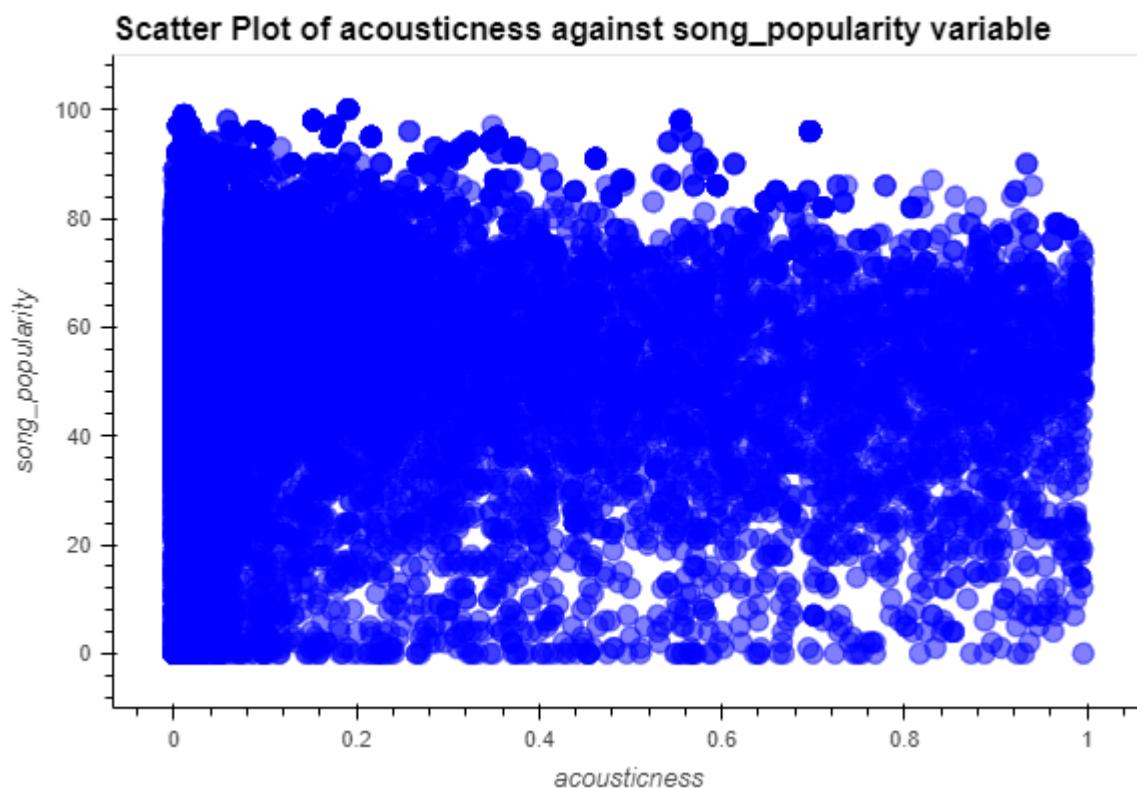
`audio_valance`: "A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. **Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry)"**.

Aunque la media se encuentra en torno a 0.5, encontramos una variación muy alta. Esta variable nos da información muy parecida a lo que en un principio queríamos conseguir con el análisis de sentimiento con respecto al título de la canción.

Análisis de las visualizaciones de las variables

Las visualizaciones que hemos podido obtener con la librería *autoviz* se encuentran disponibles en el repositorio. Además, en los reportes hechos con *pandas-profiling* también muestran visualizaciones.

Llama la atención que no se aprecia ninguna relación clara entre la popularidad de la canción y el resto de variables. A continuación se muestra un ejemplo:



Análisis canciones más y menos populares

A continuación se muestran las 10 canciones más populares:

| | song_name | song_popularity |
|-------|------------------------------------------------|------------------------|
| 2396 | Happier | 100 |
| 1964 | I Love It (& Lil Pump) | 99 |
| 3392 | Taki Taki (with Selena Gomez, Ozuna & Cardi B) | 98 |
| 7000 | Promises (with Sam Smith) | 98 |
| 2493 | Eastside (with Halsey & Khalid) | 98 |
| 11526 | In My Feelings | 98 |
| 4021 | Falling Down | 97 |
| 6264 | In My Mind | 97 |
| 8671 | SICKO MODE | 97 |
| 5017 | Lucid Dreams | 97 |

Y 10 de las menos populares:

| | song_name | song_popularity |
|-------|-------------------------------------|------------------------|
| 1537 | Habana Dreams (feat. Issac Delgado) | 0 |
| 13055 | The Watcher 2 | 0 |
| 6447 | Kamakaze | 0 |
| 2300 | nASSty | 0 |
| 7070 | Forever And Always | 0 |
| 9898 | Danky | 0 |
| 10501 | The Peanut Vendor | 0 |
| 9468 | Luv Them Girls | 0 |
| 10266 | When They Get Older | 0 |
| 8382 | Eras mi persona favorita | 0 |

Llama la atención el caso de “Eres mi persona favorita”, que pese a tener más de dos millones de reproducciones en YouTube, ha sido catalogada con una popularidad de cero. Esto también nos revela la presencia de canciones de otros idiomas además del inglés en el dataset.

Definimos la estrategia a seguir

Como hemos visto, estamos ante un problema de regresión, por tanto se pueden proponer varios métodos para la predicción. El modelo principal con el que vamos a tratar de resolver el problema es `ElasticNet`, ya que es el visto en la asignatura. Este modelo es un modelo de regresión lineal que fusiona tanto la regularización ridge como la lasso. No obstante, no nos permite ahorrarnos la utilización de estos últimos de forma directa por problemas numéricos al poner algunos de los hiperparámetros a cero. La ventaja de este método es que nos permite realizar predicciones rápidamente y su sencillez. Si los resultados no fueran del todo satisfactorios intentaremos utilizar otras técnicas y modelos más complejos basados en árboles como `RandomForestRegressor` o `XGBoost` puesto que suelen obtener mejores resultados en este tipo de datos.

Antes que nada faremos un tratamiento de los datos sencillo para tener un resultado base con el que podamos realizar comparaciones. En nuestra primera aproximación realizaremos el siguiente tratamiento:

Utilizaremos un `GridSearch` para ajustar los hiperparámetros de `ElasticNet`. El Pipeline estará compuesto por un `StandardScaler`, y el `ElasticNet`. No se realizará ni *feature engineering* ni *feature selection* en esta primera fase.

Una vez tengamos los hiperparámetros, entrenaremos nuestro modelo con todos los datos realizando validación cruzada y usaremos el R2 score medio para compararlo con otros modelos. A la hora de seleccionar el mejor modelo tendremos en cuenta tanto los datos de la validación cruzada como del test. Aunque se tendrá más en cuenta los resultados en test ya que con la validación cruzada hemos optimizado los hiperparámetros.

Una vez tengamos este modelo base, construiremos un modelo similar mediante un proceso algo más complejo. Además, optimizaremos los factores entre los que tenemos que decidir mediante un algoritmo genético. La segunda estrategia será la siguiente:

Feature Engineering. Esta fase es realmente importante y la que puede marcar una gran diferencia de cara a los resultados obtenidos. Para ello, tomando ventaja de que tenemos pocas características, vamos a crear de forma automática características. Además se buscará la creación de nuevas variables a través de la columna *song_name*.

Feature Selection. Una vez creadas las nuevas características deberemos elegir las más prometedoras. Puesto que tan solo disponemos de unas 13 mil filas, introducir demasiadas características puede llevarnos a cometer *overfitting*. A priori, creemos que lo ideal sería quedarnos con un número de columnas no superior a 50 (posteriormente haremos una reducción de dimensionalidad). Puesto que el número de características entre las que elegir puede ser potencialmente muy elevado, hemos decidido no incluir esta fase dentro del `Pipeline` de `sklearn` para acelerar el proceso de ajuste de hiperparámetros. Esta selección se va a realizar mediante dos etapas para comparar los resultados y poder tomar una mejor decisión. La primera etapa consiste en una selección de características previa al modelado. El segundo método, se realizará de manera manual dentro del `Pipeline`.

Diseño del `Pipeline`. Una vez tengamos las características iniciales con las cuales queremos entrenar nuestro modelo llega la hora de diseñar el `Pipeline` que más adelante tengamos que ajustar. Este `Pipeline` incluye:

- 1) **Preprocesado:** haremos uso del `StandardScaler` o el `MinMaxScaler` sobre todas las variables no binarias.
- 2) **Reducción de dimensionalidad:** utilizaremos PCA o Kernel PCA con diferentes núcleos. También se debe elegir el número de dimensiones a las que reducimos el problema. Incluso una posibilidad será no utilizar nada.
- 3) **Feature Selection.** Esta vez se utilizará `RFECV`, que elegirá las mejores características utilizando validación cruzada basándose en el atributo `.feature_importance_` de la clase `LinearRegression`. Todo esto de manera recursiva. Otra opción sería dejar al propio algoritmo genético que decida con qué características quedarse. Hemos decidido elegir la primera opción porque es más rápida a la hora de entrenar y creemos que puede dar los mejores resultados al ser más estable.
- 4) **Entrenar el modelo.** Como hemos mencionado anteriormente, el modelo a entrenar será `ElasticNet`. Tenemos que decidir entre el l1-ratio y el Alpha a utilizar.

Todas estas decisiones serán testeadas mediante el uso de un algoritmo genético en el pipeline. La librería que nos permite hacer esto es `sklearn-genetic-opt`.

Creamos el resultado base

Resultados

Fitting 10 folds for each of 4 candidates, totalling 40 fits

Score medio de la validación cruzada: 0.041993913922019045

Desviación estándar de la validación cruzada: 0.01237335242103942

Best params: {'elasticnet_alpha': 0.02600000000000002, 'elasticnet_l1_ratio': 0.0001}

Pipeline(steps=[('scaler', StandardScaler()), ('elasticnet', ElasticNet())])

Kaggle (test): 0.04473

Nota: antes que esta, hemos realizado 3 envíos más con resultados muy similares pero que no eran completamente reproducibles mientras ajustábamos el código.

Conclusiones

No hemos puesto como posibles hiperparámetros 0, debido a la inestabilidad numérica que genera. El modelo ha tratado de aproximarse mucho a una regresión lineal simple, por lo que en la futura prueba usaremos este modelo.

Limpieza de los datos

En esta fase también incluimos la selección de características y la creación de nuevas variables.

Eliminamos las filas duplicadas

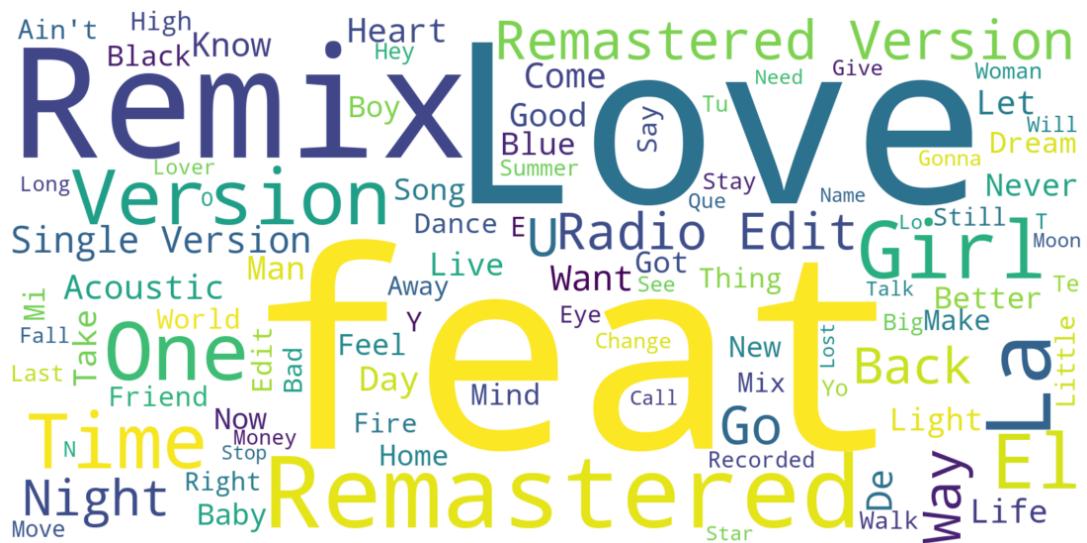
Al parecer, aunque todos los identificadores de canción son diferentes, el dataset tenía 2336 filas duplicadas si no consideramos la variable objetivo *song_popularity* y 2200 si la consideramos. Estas filas se han utilizado en el entrenamiento del *baseline* puesto que el análisis de esto fue posterior, pero a partir de este momento, se han eliminado las 2200. El resto se han mantenido puesto que la puntuación del target varía.

Análisis de la columna *song_name*

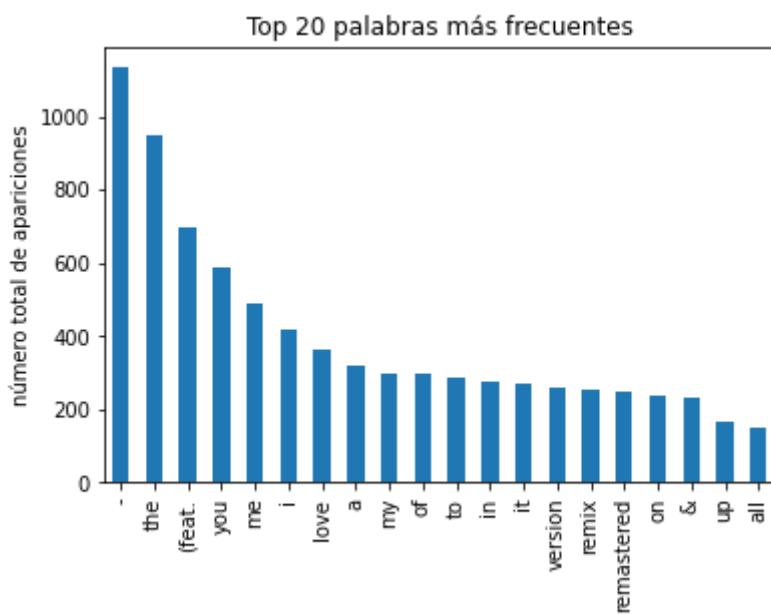
Una de las partes más importantes para obtener buenos resultados es tener unos datos que sean útiles. Con el objetivo de poder extraer información de la variable *song_name*, vamos a realizar un análisis en profundidad de esta característica.

Análisis por palabra

La idea con este análisis es ver cuáles son las palabras más comunes para poder crear características binarias que indiquen la presencia o no de una palabra o grupo de las. Para ello hemos creado una nube de palabras con la librería *wordcloud* para hacernos una idea inicial de qué palabras son más comunes. Esto es importante puesto que, para que la característica que creemos sea efectiva, debe de tener la máxima variación posible entre unos y ceros. Si casi todos los valores son cero, esta variable no aportará suficiente información.



A primera vista podemos ver que palabras como "feat" , "Remix" y "Love" son las más utilizadas. Veamos ahora de forma más numérica.



Más datos relevantes:

- Porcentaje de palabras que contienen "-": 10.77%
- Porcentaje de canciones que contienen "feat": 6.79%
- Porcentaje de palabras que son derivadas de otras (contienen "version", "remastered" o "remix"): 6.65%
- Porcentaje de palabras que contienen "love": 3.95%
- Porcentaje de palabras que contienen "(": 10.88%

Extracción de características a partir de *song_name*

Hemos decidido incluir únicamente aquellas palabras o conjuntos de ellas que se encuentren en al menos un 5% de las palabras. Por tanto, incluimos las siguientes variables binarias: *contains_hyphen*, *contains_feat*, *is_derived* (si es una canción creada a partir de otra) y *contains_parentheses*.

Además de esto, añadiremos columnas con la siguiente información: número de sílabas (*num_syllables*), número medio de caracteres por palabra, número de palabras que empiezan por mayúscula (*num_uppercase_words*) y por minúscula (*num_lowercase_words*). Además, realizaremos un análisis de sentimiento y una detección del idioma.

Análisis de las nuevas características

En esta sección vamos a incluir un pequeño resumen, para más información ver 'train_v2_report.html'. Los datos más llamativos son los siguientes:

- Se ha detectado un nombre de la canción en inglés en el 51.4% de las canciones. El resto de valores se encuentran muy repartidos. Por tanto, de cara a la predicción ***language* se codificará como binaria (1 si se ha detectado inglés, 0 si no).**
- Las columnas `num_uppercase_words` y `num_lowercase_words` no representan exactamente el número de palabras. No obstante, de cara a la predicción la información que se codifica es similar e incluso puede dar más información.

El número de palabras que empiezan por mayúscula se ha realizado contando el número de veces que un carácter en mayúsculas le sigue uno en minúsculas. En el caso de `num_lowercase_words`, restando el número de separaciones por un espacio y el 'número de palabras que empiezan por mayúscula'. Puesto que no son estimaciones exactas, se han generado 2018 filas con valores negativos en esta columna. No obstante, aunque no era el objetivo inicial de esta columna, hemos decidido mantenerlo así. El motivo tras esta decisión es que creemos que esto puede ayudar a identificar canciones con títulos más extraños, por lo que puede ser de utilidad.

- La columna `num_syllables` presenta un 12.1% de valores faltantes. Esto ha ocurrido en títulos de canciones que estaban en idiomas no soportados por la librería que contaba las sílabas. Estos valores se sustituirán por la media y se añadirá una columna bandera que indique si el valor era faltante.
- En cuanto al análisis de sentimiento, hemos utilizado una librería *textblob* que asigna un valor entre -1 y 1 en función de las palabras que encuentra (polaridad). Para ello, asigna a cada palabra un valor, el resultado total es la suma de todas ellas. La mayoría de canciones, un 78.2% han sido etiquetadas con una polaridad de 0. Es por ello que **vamos a redondear todas las palabras con un sentimiento total negativo a -1 y viceversa.**

Preprocesado de los datos

Además de las transformaciones mencionadas anteriormente, realizaremos el siguiente preprocesado:

- Aplicaremos una transformación *Box Cox* a las variables: *acousticness*, *instrumentalness*, *liveness*, *speechiness* y *loudness*. Esto hará que sigan distribuciones más gaussianas ya que presentan distribuciones algo asimétricas. Esto es especialmente útil cuando realicemos regresión lineal puesto que una de las suposiciones que el modelo hace sobre las variables es que siguen una distribución gaussiana.
- La columna *time_signature*, puesto que representa el compás de la canción, hemos decidido codificarla como una columna binaria que indique si es 4/4 o no, ya que el 94% de ellas lo son. La columna *key*, en cambio, sí que hemos decidido mantenerla así puesto que entendemos que puede haber más similitud entre canciones que usan notas parecidas, y además porque realizar un *one-hot* crearía 12 características nuevas. Esto se debe a que, al seguir una distribución relativamente uniforme, no es posible crear una característica donde agrupar a las infrecuentes.
- Una vez hechas estas transformaciones aplicaremos un *MinMaxScaler* a todas las columnas cuyo rango no esté en el rango [0, 1] o en valores negativos centrados en cero. Es decir, lo aplicaremos a: *key*, *tempo*, *num_syllables*, *avg_chars_per_word* y *num_uppercase_words*.
- La columna *num_lowercase_words*, la estandarizaremos para conservar los valores cero cercanos a cero.

- En *song_duration_ms*, observamos que hay canciones que son mucho más largas que otras por lo que será útil poner un tope a estos valores atípicos. Esto se limitará a 3 desviaciones típicas con respecto a la media ya que la distribución que sigue es gaussiana. Tras esto, se aplicará un *MinMaxScaler* (queremos seguir manteniendo los valores cero a cero). En *num_syllables* aplicamos encontramos la misma situación, por lo que también aplicamos esta estrategia
- En *average_chars_per_word*, cuando no hay palabras, hay valores faltantes, estos valores serán rellenados con cero.

Selección de características

Para algunos experimentos nos será útil realizar una selección de características previa al modelado. El objetivo de esto es mejorar la calidad del modelo, evitar *overfitting* y reducir el tiempo de cómputo.

Esta selección de características se realizará de manera automática mediante la librería *featurewiz*. La selección internamente se realiza en dos pasos:

1. Aplicamos el algoritmo SULOV:

"Find all the pairs of highly correlated variables exceeding a correlation threshold (say absolute(0.7)).

Then find their MIS score (Mutual Information Score) to the target variable. MIS is a non-parametric scoring method. So it's suitable for all kinds of variables and target.

Now take each pair of correlated variables, then knock off the one with the lower MIS score.

What's left is the ones with the highest Information scores and least correlation with each other."

2. XGBoost recursivo:

"Select all variables in the data set and the full data split into train and valid sets.

Find top X features (could be 10) on train using valid for early stopping (to prevent over-fitting)

Then take next set of vars and find top X

Do this 5 times. Combine all selected features and de-duplicate them."

Resultados tras la selección de características

Variables seleccionadas:

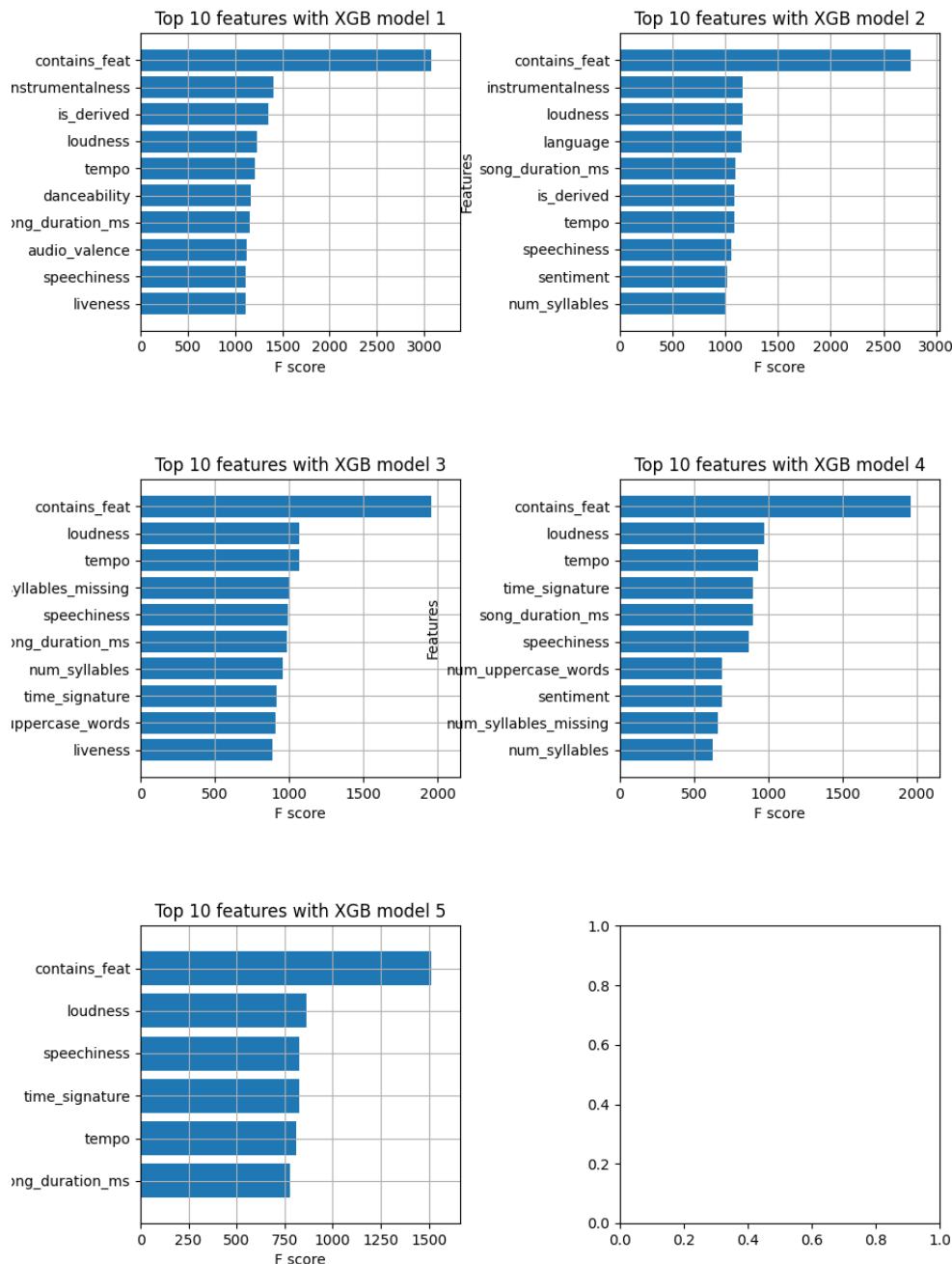
```
['contains_feat', 'instrumentalness', 'is_derived', 'loudness', 'tempo',
'danceability', 'song_duration_ms', 'audio_valence', 'speechiness', 'liveness',
```

'num_lowercase_words', 'language', 'sentiment', 'num_syllables_missing',
 'num_syllables', 'time_signature'] (17 en total)

Tras la primera fase se eliminaron las siguientes columnas por estar altamente correlacionadas con otras:

['contains_parentheses', 'energy']

Las gráficas obtenidas tras realizar XGBoost de manera recursiva:



Destaca la importancia de la variable *contains_feat*.

Primeros experimentos

Una vez completada la limpieza de los datos y el preprocessado inicial, comenzamos a modelar los diferentes *pipelines*. Estos se modelarán de manera independiente cada uno utilizando diferentes técnicas de optimización de hiperparámetros y selección de características. Los mejores hiperparámetros se obtendrán realizando el R2 score medio de 5 repeticiones de validaciones cruzadas de 10 *folds*. Aunque a costa de aumentar la complejidad computacional, hace que la confianza que tengamos en nuestros modelos sea más alta.

Los *pipelines* que den resultados relativamente aceptables serán combinados en un *Stacking Regressor*.

Experimento 1: Regresión Lineal

Uno de los objetivos tras este proyecto era obtener un mejor entendimiento de la regresión lineal. Es por ello que nos hemos propuesto como un reto el tratar de obtener unos resultados competitivos utilizando únicamente este modelo.

No obstante, por los resultados obtenidos en el modelo base y el análisis realizado, hemos podido ver que la relación entre las variables dependientes y la popularidad de la canción no es lineal. Es por esto, que necesitamos transformar las variables de entrada de alguna forma. Es por eso que probaremos a crear nuevas variables que sean producto de las características que tienen que ver con el nombre de la canción.

Una vez hecho esto, el pipeline se compondrá de los siguientes pasos:

- PCA. Nos quedaremos con el número de componentes que tenga una varianza explicada de al menos el 95%. Hemos decidido usar PCA en lugar de Kernel PCA, ya que, tras hacer una pequeña prueba, hemos visto que los resultados no son muy prometedores y que además tarda demasiado en entrenarse (+4 minutos) como para hacer un *grid search*.
- Seleccionaremos las K mejores características.
- Realizamos la predicción usando Regresión Lineal para obtener resultados más rápidamente que los resultados serán parecidos a utilizar regularización

La optimización del número de componentes, del K y de los hiperparámetros de regularización de los respectivos modelos se hará mediante una búsqueda en grid. Probaremos también en no utilizar las interacciones entre variables.

En este experimento no hacemos uso de la selección de características mencionada anteriormente ya que esta se basa en *XGBoost* principalmente y aquí estamos utilizando un algoritmo muy distinto.

Resultados

```
Score medio de la validación cruzada: 0.03306009473273754
Desviación estándar de la validación cruzada: 0.0015253907364974142
Best params: {'pca_n_components': 0.99, 'select_k_best_k': 13,
'select_k_best_score_func': <function f_regression at 0x000002749D1574C0>}
Pipeline(steps=[('pca', PCA(n_components=0.99)),
 ('select_k_best',
  SelectKBest(k=13,
               score_func=<function f_regression at 0x000002749D1574C0>),
  ('elastic_net', LinearRegression()))])
r2 score en train: 0.036020696107569194
mae en train: 15.996559706129057
```

USANDO LAS INTERACCIONES:

```
Número de columnas totales: 104
Fitting 5 folds for each of 48 candidates, totalling 240 fits
Score medio de la validación cruzada: 0.039375984544121924
Desviación estándar de la validación cruzada: 0.004355116295955313
Best params: {'pca_n_components': 0.99, 'select_k_best_k': 19,
'select_k_best_score_func': <function f_regression at 0x000002749D1574C0>}
Pipeline(steps=[('pca', PCA(n_components=0.99)),
 ('select_k_best',
  SelectKBest(k=19,
               score_func=<function f_regression at 0x000002749D1574C0>),
  ('elastic_net', LinearRegression()))])
r2 score en train: 0.04603747025374194
mae en train: 15.858934812596214
```

Los resultados en test son: 0.04543

Conclusiones

Si bien se obtiene una ligera mejora al añadir las interacciones, no es especialmente relevante, además los resultados son similares a los del modelo base. Por lo que este modelo no parece ser de utilidad. Esto concuerda con nuestra hipótesis de que la relación entre la variable objetivo y el resto de variables no es lineal.

Experimento 2: Random Forest Regressor

En este experimento se utilizarán las variables previamente seleccionadas y se tratará de optimizar los hiperparámetros de un *Random Forest Regressor*. Esta vez, la selección de hiperparámetros se hará mediante un algoritmo genético para acelerar el proceso y realizar una búsqueda informada. En concreto, haremos uso de la librería TPOT.

La espacio de búsqueda de hiperparámetros realizada ha sido el siguiente:

```
parameters = {  
    'n_estimators': [200],  
    'max_features': range(1, len(X_train.columns)),  
    'min_samples_split': [2, 3, 4, 5, 6, 7, 8, 9, 10],  
    'min_samples_leaf': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],  
}
```

Para construirlo hemos seguido los consejos de esta página:

<https://towardsdatascience.com/random-forest-hyperparameters-and-how-to-fine-tune-them-17aee785ee0d>

Resultados

En esta primera prueba se han realizado 4 generaciones con 8 individuos en cada una. Además, no se ha hecho uso de la selección de características vista anteriormente. La validación cruzada utilizada ha sido de 5 *folds*. Los resultados obtenidos son los siguientes.:

Generation 1 - Current best internal CV score: 0.057098516085050476

Generation 2 - Current best internal CV score: 0.06125504812803155

Generation 3 - Current best internal CV score: 0.06125504812803155

Generation 4 - Current best internal CV score: 0.06125504812803155

Best pipeline: RandomForestRegressor(input_matrix, max_features=6, min_samples_leaf=8, min_samples_split=8, n_estimators=200)

r2 score en train: 0.4166727766643924

mae en train: 12.275082622159541

R2 score en test: 0.20904

En este segundo enfoque, hemos utilizado los resultados de la selección de características anterior. Además, hemos probado con más muestras por generación (12) y más generaciones (8). Se ha realizado una parada prematura puesto que los resultados no mejoraban. La validación cruzada utilizada ha sido de 10 *folds*.

Generation 1 - Current best internal CV score: 0.0508634812180639

Generation 2 - Current best internal CV score: 0.05431985647656169

Generation 3 - Current best internal CV score: 0.05431985647656169

Generation 4 - Current best internal CV score: 0.05431985647656169

Generation 5 - Current best internal CV score: 0.05431985647656169

Generation 6 - Current best internal CV score: 0.05431985647656169

Generation 7 - Current best internal CV score: 0.05468911065458826

Generation 8 - Current best internal CV score: 0.05468911065458826

Best pipeline: RandomForestRegressor(CombineDFs(input_matrix, CombineDFs(input_matrix, input_matrix)), max_features=5, min_samples_leaf=10, min_samples_split=5, n_estimators=200)

r2 score en train: 0.34082172768948404

mae en train: 13.097924262685087

r2 score en test: 0.17333

Tras este experimento nos preguntamos cómo de útil han sido las nuevas características y el tratamiento realizado. Es por ello, que utilizamos la primera versión de los datos (ningún tipo de preprocesado ni de *feature engineering*). De hecho, ni siquiera hemos eliminado las filas duplicadas. Estos son los resultados obtenidos:

Fitting 10 folds for each of 5 candidates, totalling 50 fits

Score medio de la validación cruzada: 0.3633955807039154

Desviación estándar de la validación cruzada: 0.029637064966174777

Best params: {'max_features': 3}

RandomForestRegressor(max_features=3, n_estimators=200)

r2 score en train: 0.908409746405992

r2 score en test: 0.39341

Conclusiones

En cuanto a la primera prueba, por los resultados de la validación cruzada, este modelo parece ser mejor que la regresión lineal utilizada anteriormente. No obstante, llama la atención la diferencia entre los resultados obtenidos en cada uno de los conjuntos de datos. Se pasa de un 0.41 en entrenamiento a un 0.2 en test. Esto indica que se ha producido overfitting.

En el segundo experimento observamos más o menos lo mismo pero con peores resultados. Esto parece indicar que se han eliminado variables que contenían algo de información.

El tercer caso, en cambio, nos ha sorprendido. El hecho de que funcione tan bien sin preprocesado se explica por el hecho de que *Random Forest* no realiza ninguna suposición inicial sobre la distribución de las variables a diferencia de la regresión lineal. La conclusión que extraemos de aquí es que es posible que el preprocesado que habíamos hecho para la regresión lineal no sea el más adecuado para este algoritmo. Por otro lado, volvemos a observar una gran diferencia entre los resultados en el entrenamiento (0.9) y en los de test (0.39). Estos últimos se asemejan más a los de validación (0.36), puesto que además la desviación típica no es muy alta (0.02) podemos tener una confianza relativa en que los resultados del test privado se encontrarán también en torno a 0.36 pese al *overfitting*.

Análisis posterior de los datos de test

Tras estos resultados, y al recordar que en los datos de entrenamiento había filas repetidas. Nos ha surgido la duda de si en los datos de test hay filas exactamente iguales a las de los datos de entrenamiento.

Hemos analizado esto y la respuesta es que no, no existen filas que coincidan exactamente.

Realizamos un nuevo preprocessado

Este preprocessado está pensado para algoritmos que no realizan ninguna suposición sobre las distribuciones de las variables independientes. La idea es realizar un preprocessado que simplemente añada las características extraídas a partir del nombre de la canción y rellene los valores nulos generados. Es el mismo preprocessado que el visto anteriormente pero no se ha aplicado ningún tipo de escalado, ni de eliminación de outliers.

Este preprocessado también nos será útil si pretendemos usar alguna librería de *Machine Learning* automático.

Segunda tanda de experimentos

Experimento 3: Random Forest Regressor

En esta ocasión realizaremos un experimento similar al último. La única diferencia es que esta vez utilizaremos el preprocesado minimalista mencionado anteriormente.

Además probaremos a añadir únicamente la columna *contains_feat* puesto que anteriormente se vió que daba buenos resultados. También probaremos a entrenar con los datos originales pero eliminando las filas duplicadas.

Resultados

Resultados de aplicar Random Forest con los nuevos datos preprocesados:

Fitting 10 folds for each of 8 candidates, totalling 80 fits

Score medio de la validación cruzada: 0.04270250509981396

Desviación estándar de la validación cruzada: 0.01866821189782761

Best params: {'max_features': 4}

RandomForestRegressor(max_features=4, n_estimators=200)

r2 score en train: 0.8614859218897397

r2 score en test: 0.36958

Resultados de aplicar Random Forest a los datos originales y únicamente eliminando filas duplicadas y añadiendo la columna de si contiene feat o no.

Score medio de la validación cruzada: 0.04141314764132582

Desviación estándar de la validación cruzada: 0.023804282095744443

Best params: {'max_features': 7}

RandomForestRegressor(max_features=7, n_estimators=200)

r2 score en train: 0.8605094126676279

r2 score en test: 0.36947

Conclusiones

Se puede observar que, por un lado, el no cambiar la distribución de las variables mejora los resultados. Pero además, se puede apreciar que añadir nuevas variables genera un mayor *overfitting*. Esto se ve especialmente en los datos de la validación cruzada.

Experimento 4: AutoML

Resultados

Con el preprocesado minimalista y añadiendo las nuevas variables, sin selección de características:

```
Ensemble_Stacked rmse 19.705349 trained in 35.83 seconds
AutoML fit time: 3665.5 seconds
AutoML best model: Ensemble_Stacked
train r2 score: 0.20782638297748302
Test r2 score: 0.13333
Cross validation (10-fold) r2 score: 0.0725499
```

Columnas eliminadas:

```
[ "contains_parentheses", "num_uppercase_words", "random_feature",
  "avg_chars_per_word", "time_signature", "contains_hyphen",
  "num_syllables_missing", "key"]
```

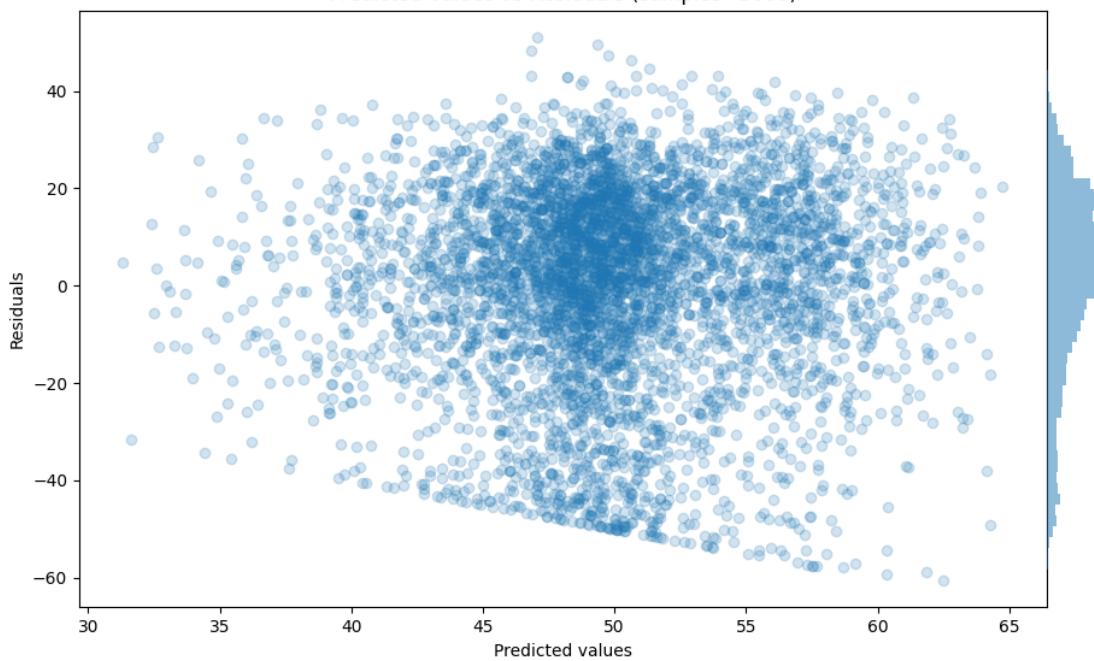
Columnas añadidas:

```
"instrumentalness_multiply_danceability",
"contains_feat_sum_instrumentalness",
"danceability_ratio_instrumentalness",
"is_derived_sum_instrumentalness",
"instrumentalness_ratio_danceability",
"instrumentalness_ratio_num_uppercase_words",
"audio_mode_sum_instrumentalness",
"energy_ratio_instrumentalness",
"time_signature_sum_instrumentalness",
"num_uppercase_words_ratio_instrumentalness"
```

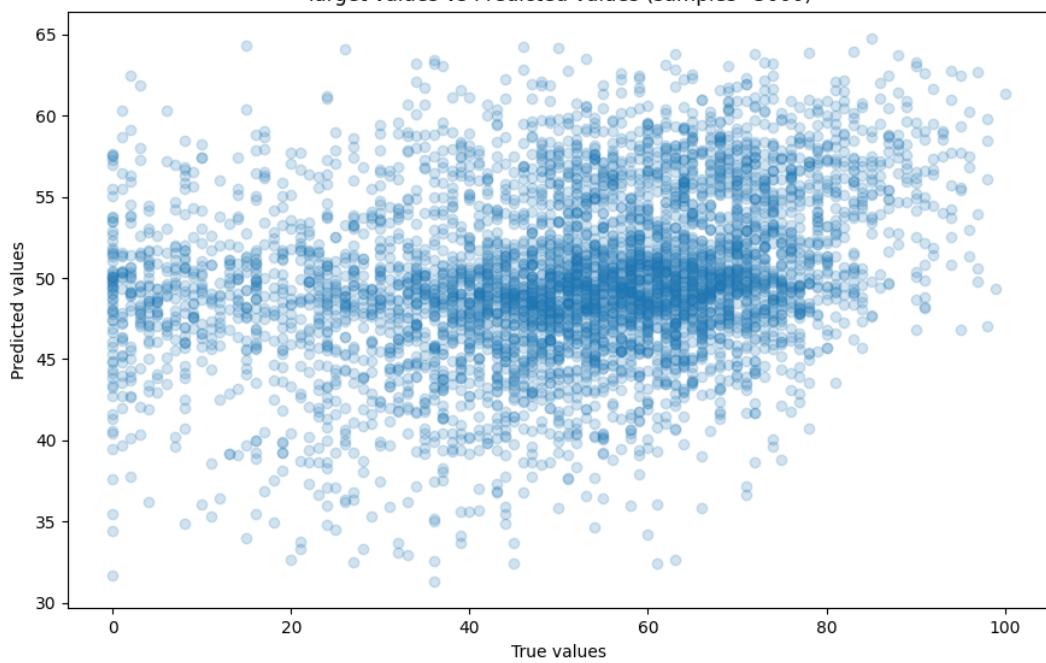
Estructura del modelo:

| Model | Weight |
|------------------------------------------|--------|
| 101_Xgboost_Stacked | 1 |
| 105_ExtraTrees_SelectedFeatures_Stacked | 36 |
| 107_NeuralNetwork_SelectedFeatures | 4 |
| 18_Xgboost_SelectedFeatures | 2 |
| 20_LightGBM | 2 |
| 31_CatBoost | 10 |
| 42_RandomForest_Stacked | 12 |
| 70_RandomForest_SelectedFeatures_Stacked | 19 |
| 7_Default_NeuralNetwork | 3 |
| 7_Default_NeuralNetwork_SelectedFeatures | 4 |
| 81_ExtraTrees_SelectedFeatures_Stacked | 37 |
| 85_NeuralNetwork_SelectedFeatures | 2 |
| Ensemble | 2 |

Predicted values vs Residuals (samples=5000)



Target values vs Predicted values (samples=5000)



Estas gráficas se han elaborado realizando una predicción sobre una muestra aleatoria del conjunto de entrenamiento con el mejor modelo obtenido.

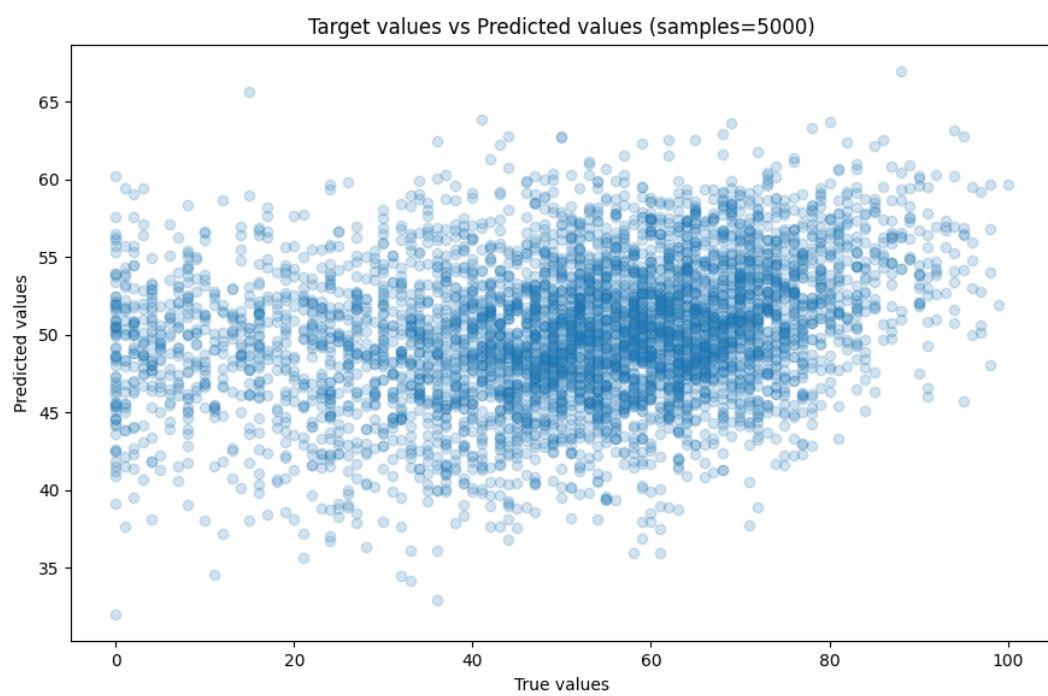
Con el preprocessado normal y añadiendo las nuevas variables, sin selección de características:

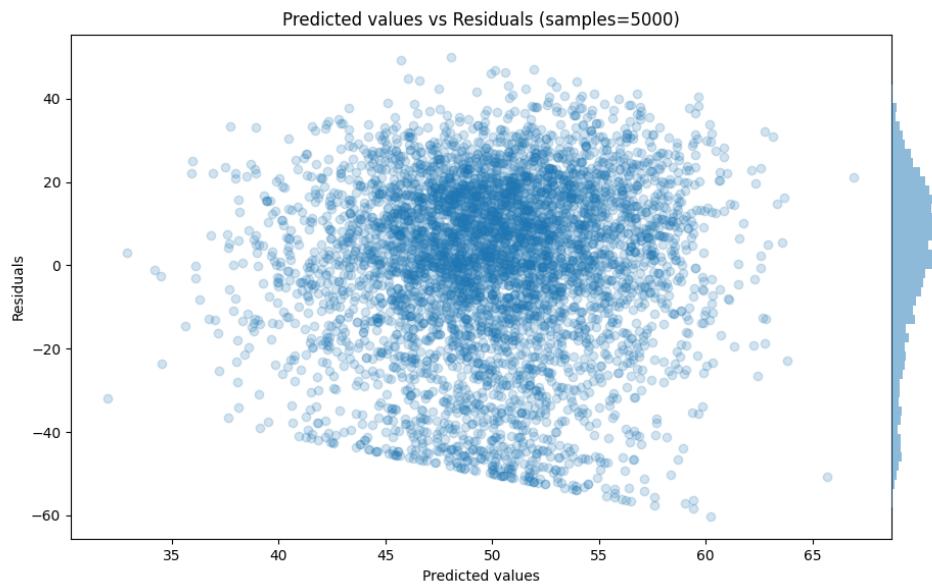
AutoML fit time: 4850.65 seconds

AutoML best model: Ensemble train

r2 score: 0.21294488656385435

Cross validation (10-fold) r2 score: 0.0668511





Estas gráficas se han elaborado realizando una predicción sobre una muestra aleatoria del conjunto de entrenamiento con el mejor modelo obtenido.

Únicamente eliminado filas duplicadas:

AutoML fit time: 3645.18 seconds

AutoML best model: Ensemble_Stacked

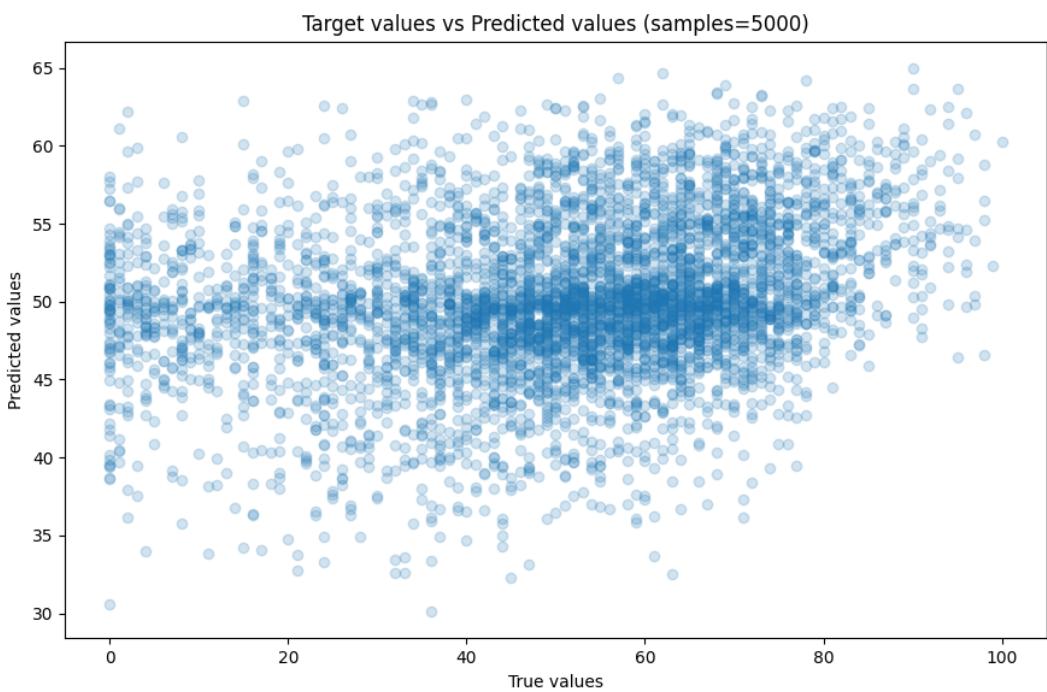
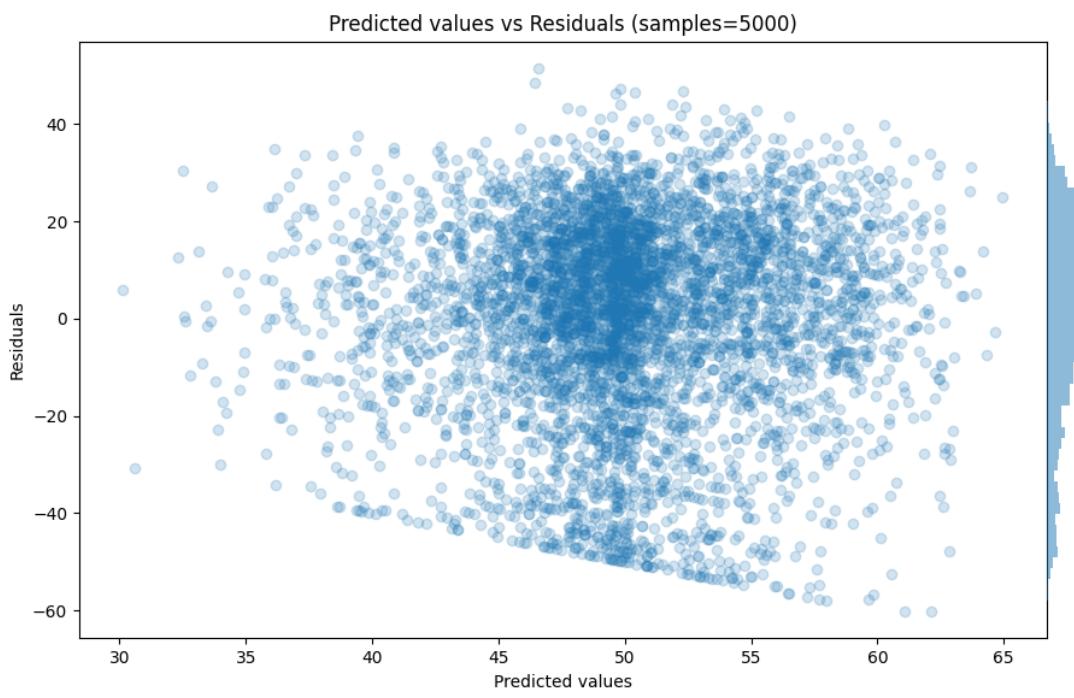
Ensemble structure

| Model | Weight |
|------------------------------------------|--------|
| 14_Xgboost | 4 |
| 20_LightGBM_KMeansFeatures | 4 |
| 26_LightGBM | 3 |
| 33_CatBoost | 5 |
| 38_RandomForest_SelectedFeatures_Stacked | 17 |
| 43_RandomForest | 1 |
| 47_ExtraTrees_SelectedFeatures_Stacked | 32 |
| 6_Default_CatBoost_KMeansFeatures | 1 |
| Ensemble | 8 |

train r2 score: 0.1873358835329907

test r2 score: 0.12376

Cross validation (10-fold) r2 score: 0.0696374



Estas gráficas se han elaborado realizando una predicción sobre una muestra aleatoria del conjunto de entrenamiento con el mejor modelo obtenido.

Utilizando los datos originales sin eliminar las filas duplicadas:

AutoML fit time: 3623.04 seconds

AutoML best model: Ensemble_Stacked

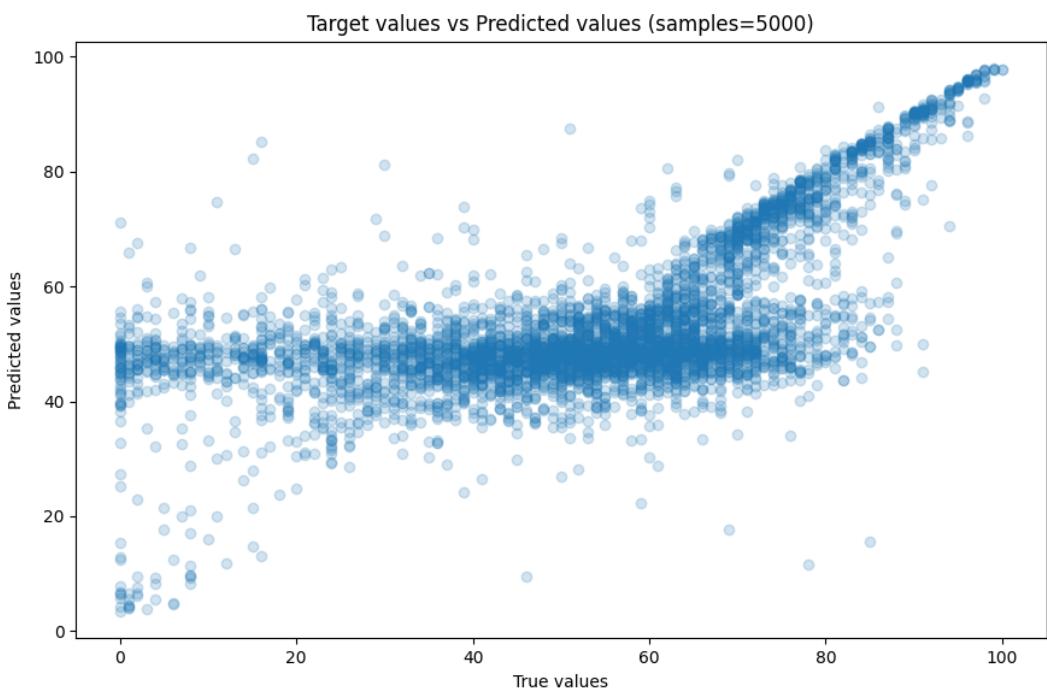
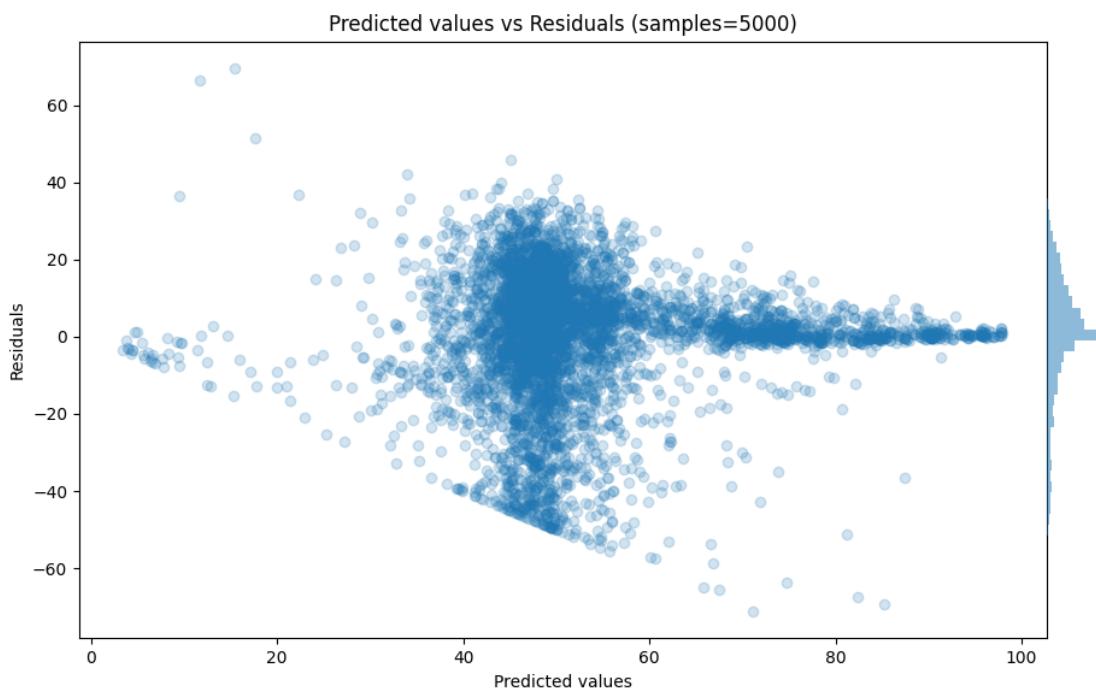
Ensemble structure

| Model | Weight |
|-----------------------------------|--------|
| 14_Xgboost_KMeansFeatures_Stacked | 2 |
| 39_RandomForest_Stacked | 19 |
| 48_ExtraTrees_Stacked | 14 |
| 75_Xgboost_Stacked | 11 |

train r2 score: 0.8853960958067962

test r2 score: 0.39854

Cross validation (10-fold) r2 score: 0.388393

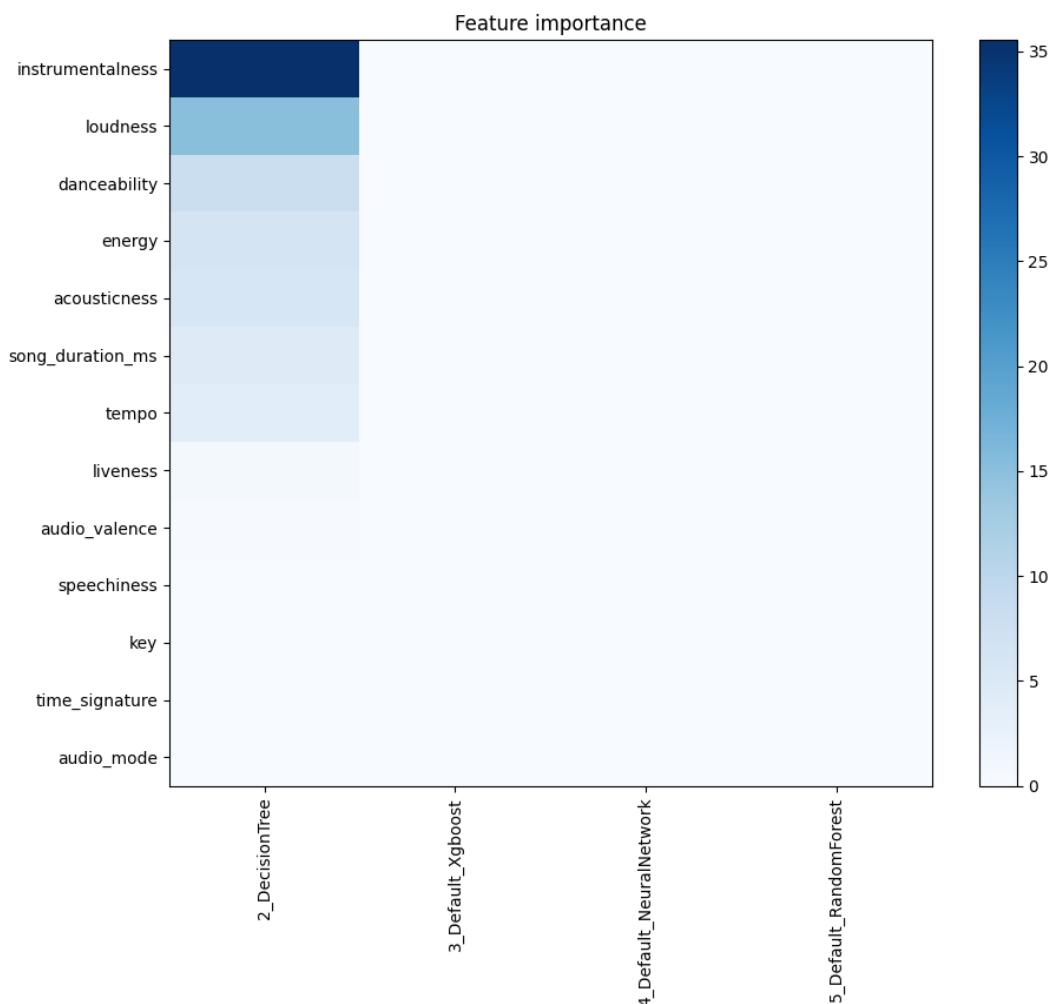


Estas gráficas se han elaborado realizando una predicción sobre una muestra aleatoria del conjunto de entrenamiento con el mejor modelo obtenido.

Además de esto, hemos realizado otro análisis con esta misma herramienta y con estos mismos datos para tratar de comprender más los datos. Para ello, hemos cambiado el modo de entrenamiento de "Compete" a "Explain". Estas es la información más relevante obtenida:

AutoML Leaderboard

| Best model | name | model_type | metric_type | metric_value | train_time |
|-----------------|-------------------------|----------------|-------------|--------------|------------|
| | 1_Baseline | Baseline | r2 | -0.000197932 | 0.61 |
| | 2_DecisionTree | Decision Tree | r2 | 0.0529606 | 13.63 |
| | 3_Default_Xgboost | Xgboost | r2 | 0.262295 | 13.82 |
| | 4_Default_NeuralNetwork | Neural Network | r2 | 0.0529571 | 1.5 |
| | 5_Default_RandomForest | Random Forest | r2 | 0.076466 | 4.41 |
| the best | Ensemble | Ensemble | r2 | 0.267628 | 0.21 |



Análisis de los resultados obtenidos: Descubriendo un problema de *Data Leakage*

Llama la atención como la distribución de los errores, en el experimento en el que añadimos las columnas duplicadas, cambia tanto en las canciones con mayor popularidad. En concreto, el modelo parece memorizar la popularidad de un porcentaje elevado de las canciones más populares. Esto nos ha llevado a indagar un poco más en el motivo de por qué esto ocurre.

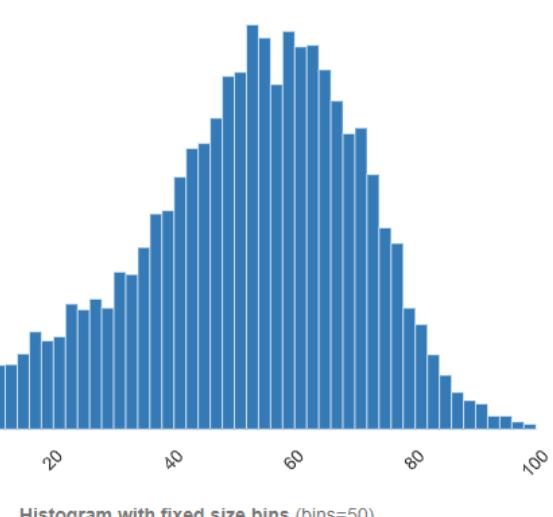
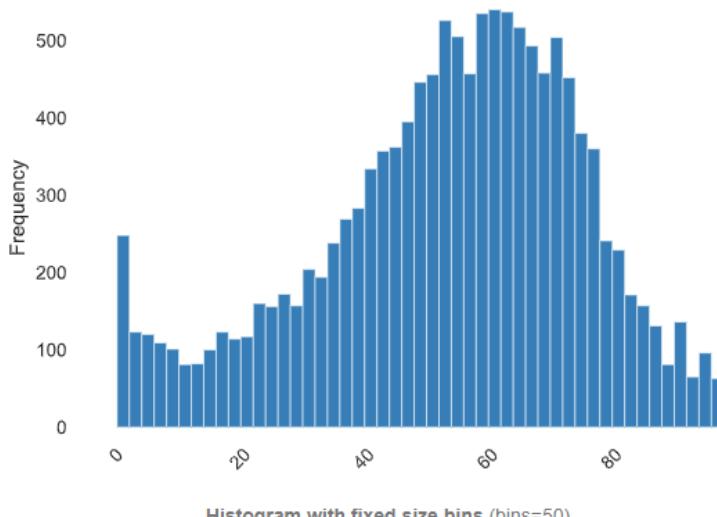
Lo primero que hemos comprobado es, si en efecto, las canciones duplicadas tienen de media una popularidad mayor. Echando un vistazo rápido a los reportes ya hechos hemos podido comprobar que esto es cierto. A la izquierda se muestran las estadísticas de *song_popularity* extraídas antes y después de eliminar duplicados y a la derecha después.

Quantile statistics

| | |
|---------------------------|-----|
| Minimum | 0 |
| 5-th percentile | 8 |
| Q1 | 40 |
| median | 56 |
| Q3 | 69 |
| 95-th percentile | 85 |
| Maximum | 100 |
| Range | 100 |
| Interquartile range (IQR) | 29 |

Quantile statistics

| | |
|---------------------------|-----|
| Minimum | 0 |
| 5-th percentile | 7 |
| Q1 | 38 |
| median | 53 |
| Q3 | 64 |
| 95-th percentile | 78 |
| Maximum | 100 |
| Range | 100 |
| Interquartile range (IQR) | 26 |



Por otro lado, la mejora en cuanto a los resultados en la validación cruzada eran esperables puesto que al haber datos duplicados, es muy probable que se estén validando los resultados con datos utilizados en el entrenamiento del *fold*. Por eso mismo, es sorprendente que los datos al evaluar en el conjunto de test sean incluso mejores que al validar.

Si bien ya habíamos comprobado que no hay filas del entrenamiento en los datos de test. Esto nos ha llevado a investigar un poco más en detalle. Tras este análisis hemos descubierto que:

- Existen 606 filas completamente duplicadas en el conjunto de test.
- Que hay 2143 filas en las que el nombre de la canción de los datos de test se encuentra en los datos de entrenamiento. Esto representa casi la mitad de los datos de test totales.

Especialmente esto último es un problema debido a que, si bien las filas no coinciden completamente se pueden comprobar que son muy parecidas.

Veamos un ejemplo con la canción *Seven Nation Army*:

En el conjunto de train encontramos esta fila:

```
[['Seven Nation Army' 232626 0.00597 0.745 0.464 0.41 4 0.239  
-7.5870000000000015 0 0.0819999999999999 123.893 4 0.297]]
```

En el conjunto de test encontramos estas filas:

```
[['Seven Nation Army' 231733 0.00817 0.737 0.463 0.447 0 0.255  
-7.827999999999985 1 0.0792 123.881 4 0.324]  
['Seven Nation Army' 172919 0.890999999999999 0.69 0.283 0.000386 2  
0.111 -7.309 1 0.0294 104.348 4 0.35]  
['Seven Nation Army' 231920 0.0066599999999999 0.745 0.466 0.35 4 0.272  
-7.62 0 0.0864 123.889 4 0.303]]
```

Se pueden comprobar que todas las filas son muy similares. Los resultados obtenidos en test, además, confirman que la popularidad asociada a cada fila será muy similar. Esto tiene sentido puesto que se trata de la misma canción pero con predicciones ligeramente distintas sobre sus propiedades.

Experimentos memorizando las canciones repetidas

Tras haber encontrado que se estaba evaluando en el conjunto de test con canciones presentes en el conjunto de entrenamiento y haber hecho pública esta información, hemos vuelto a realizar los experimentos previamente comentados.

A partir de este momento, para predecir las canciones del conjunto de test se utilizará la popularidad media de las canciones en el conjunto de entrenamiento que tengan ese mismo nombre. Para el resto de canciones se realizará una predicción con un modelo.

Resultados de actualizar las predicciones anteriores memorizando las canciones repetidas

Los r2-score en test obtenidos pasan a ser los siguientes:

- Baseline: 0.23
- (Experimento 1) Regresión lineal usando las interacciones: 0.28
- (Experimento 2) Random Forest sin selección de características: 0.296
- (Experimento 2) Random Forest con selección de características: 0.295
- (Experimento 2) Random Forest utilizando los datos originales sin eliminar las filas duplicadas: 0.295
- (Experimento 3) Random Forest con preprocessado minimalista + nuevas características: 0.297
- (Experimento 4) AutoML con preprocessado minimalista + nuevas variables: 0.302
- (Experimento 4) Utilizando los datos originales sin eliminar las filas duplicadas: 0.302

Llama la atención como utilizando este método de predicción en los algoritmos basados en árboles que fueron entrenados con los datos originales se empeoran los resultados. Esto quiere decir que la popularidad de la canción, que hemos “predicho” nosotros calculando la media por nombre de canción, en estas columnas, es menos exacta que el resultado obtenido por el propio modelo.

Tras pensarlo un poco, creemos que esto se debe a que hay canciones distintas con el mismo nombre. Para hacerlo de la manera adecuada, a la canción del conjunto de test, se le debe asignar la popularidad de la canción más parecida de las que se encuentran en el entrenamiento siempre y cuando cumpla un cierto mínimo (podría haber canciones que tengan el mismo nombre que otras del conjunto de entrenamiento pero que ninguna sea la misma).

Una posible forma de mejorar estos resultados sería volver a crear un nuevo dataset pero eliminando las filas que contiene un nombre de la canción que ya aparece en el conjunto de test usando el último método descrito. Esto se haría debido a que las predicciones para estas canciones se realizarían de forma manual. Por tanto, nos interesa predecir bien únicamente el resto de canciones. Aplicando este procedimiento, se conseguiría que el dataset de entrenamiento y el subconjunto de test en el que estamos interesados sigan una misma distribución. Esto es necesario puesto que es uno de los supuestos en los que se basan los modelos de aprendizaje automático: que el conjunto de entrenamiento y test siguen una misma distribución.

No hemos querido seguir avanzando en esta parte puesto que creemos que, aunque sí sería útil para ganar la competición, no tiene tanta relevancia en cuanto al aprendizaje de esta asignatura, puesto que se trata de buscar la mejor manera de explotar un error, no del uso de métodos clásicos para predecir.

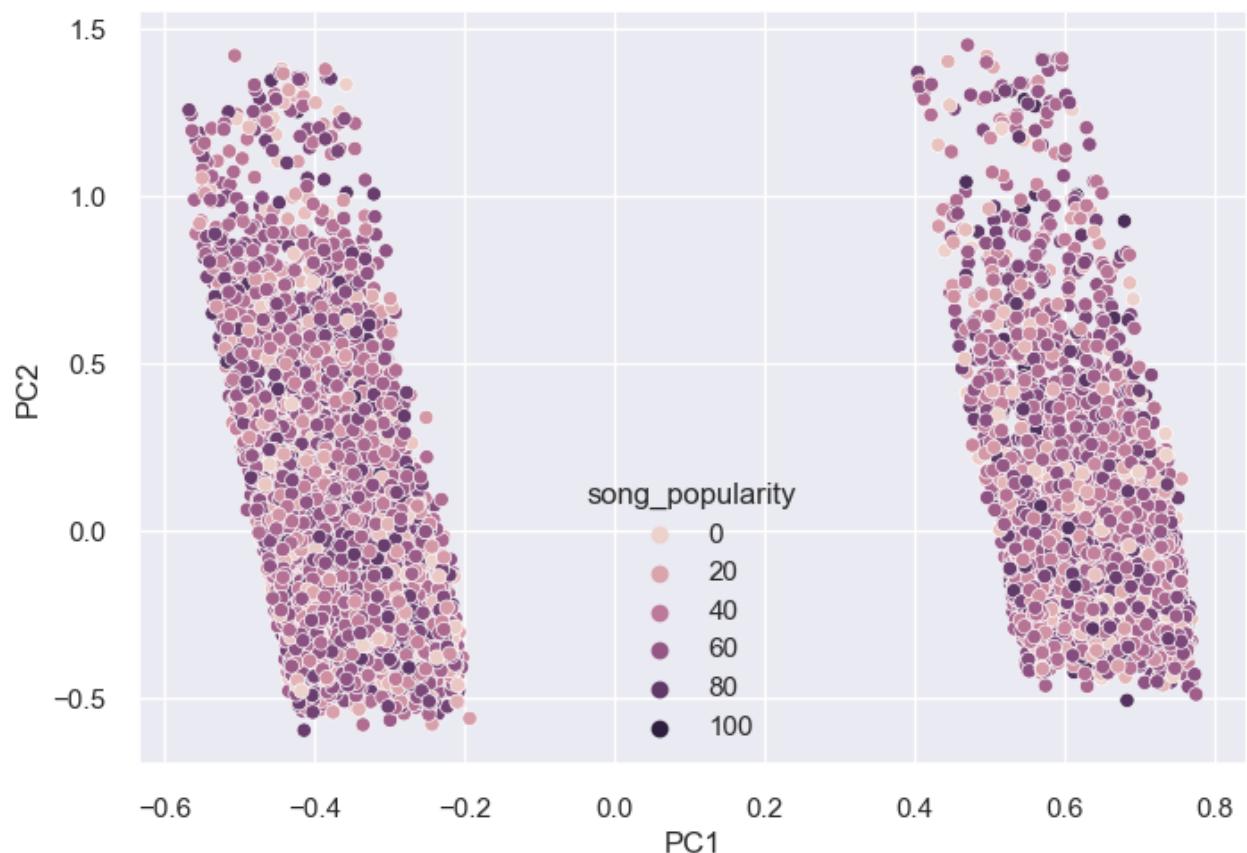
Visualización de los datos

Primeras visualizaciones

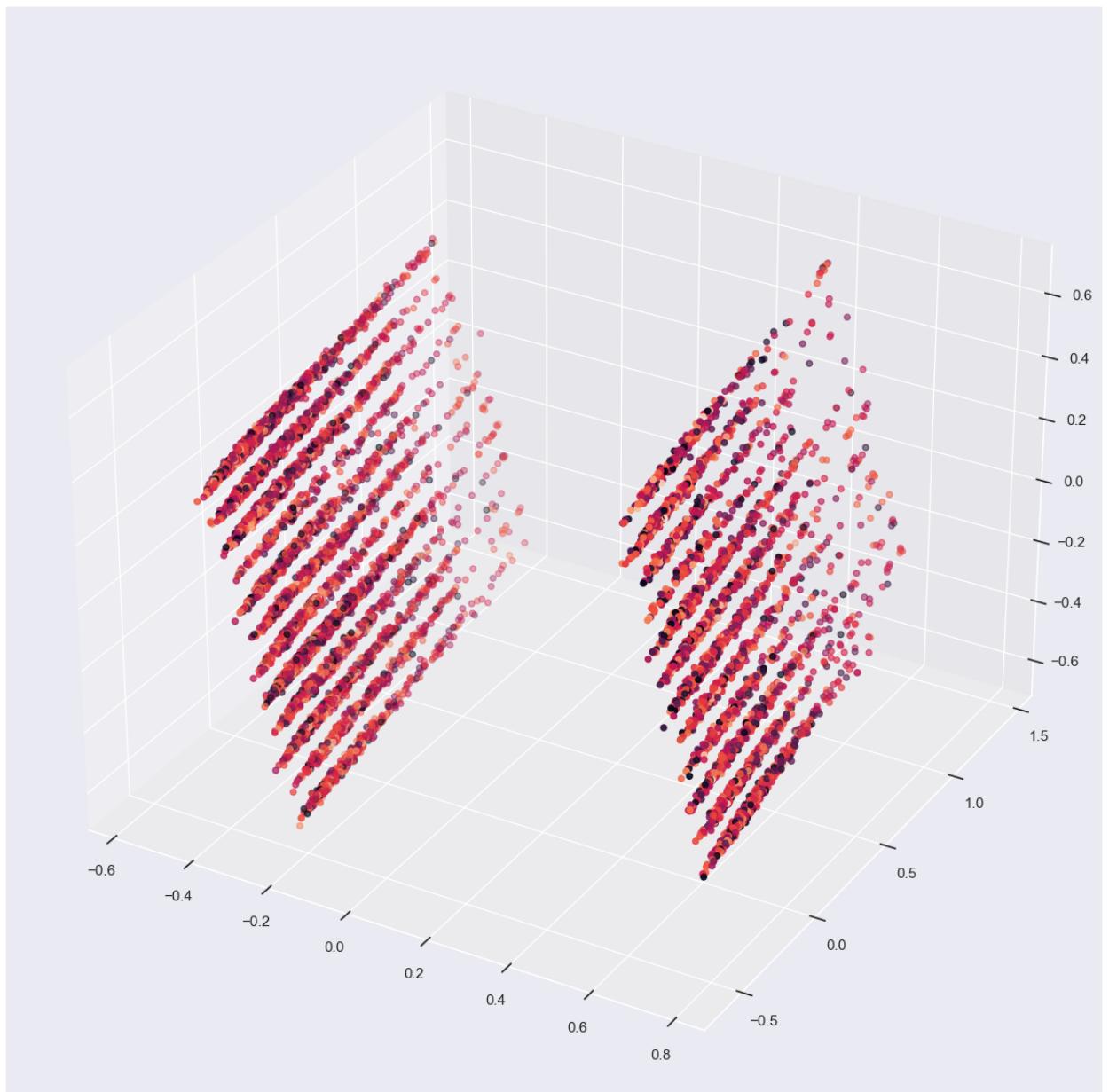
El objetivo de este apartado es, mediante la visualización de datos, intentar entender qué variables influyen en la popularidad de una canción. En particular, nuestro objetivo es observar si las canciones con una popularidad similar son similares, es decir, que en la visualización éstas salgan cerca.

Primero hemos probado con los datos con un simple preprocesado MinMax en todos los datos y descartando el nombre de la canción.

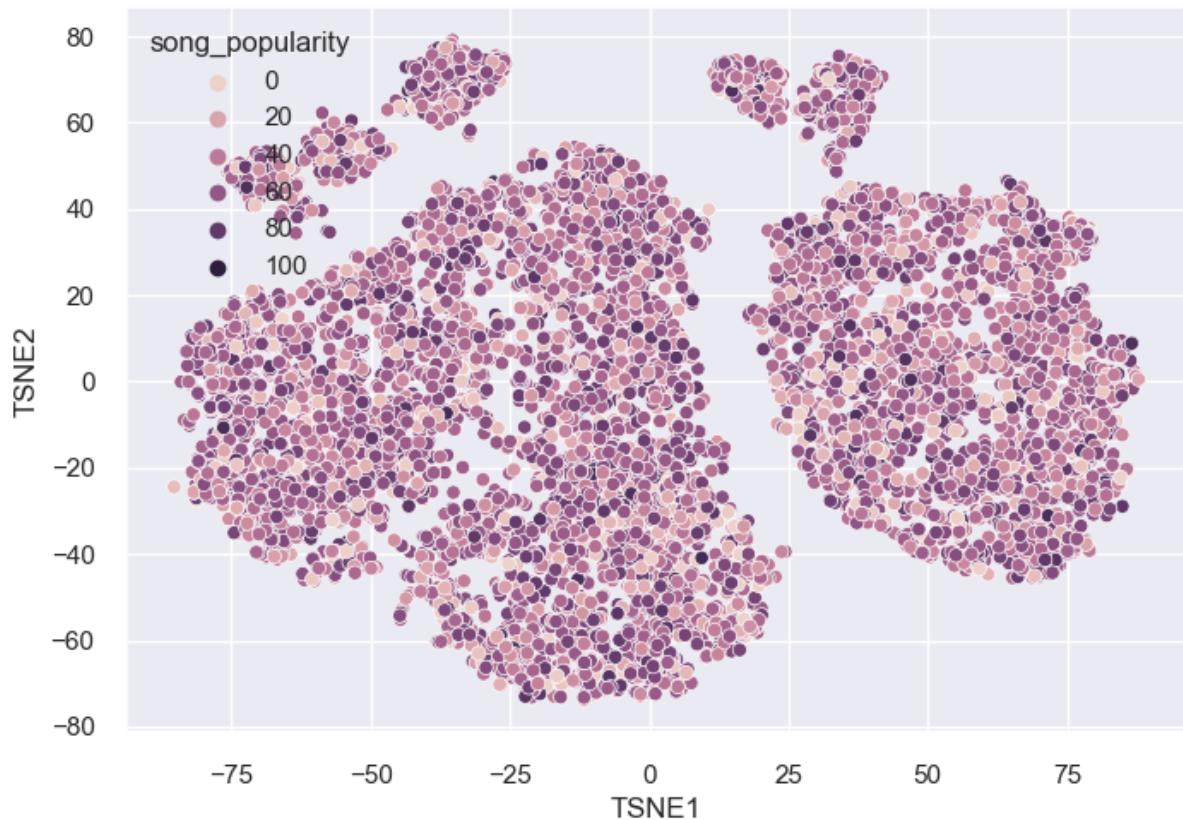
PCA:



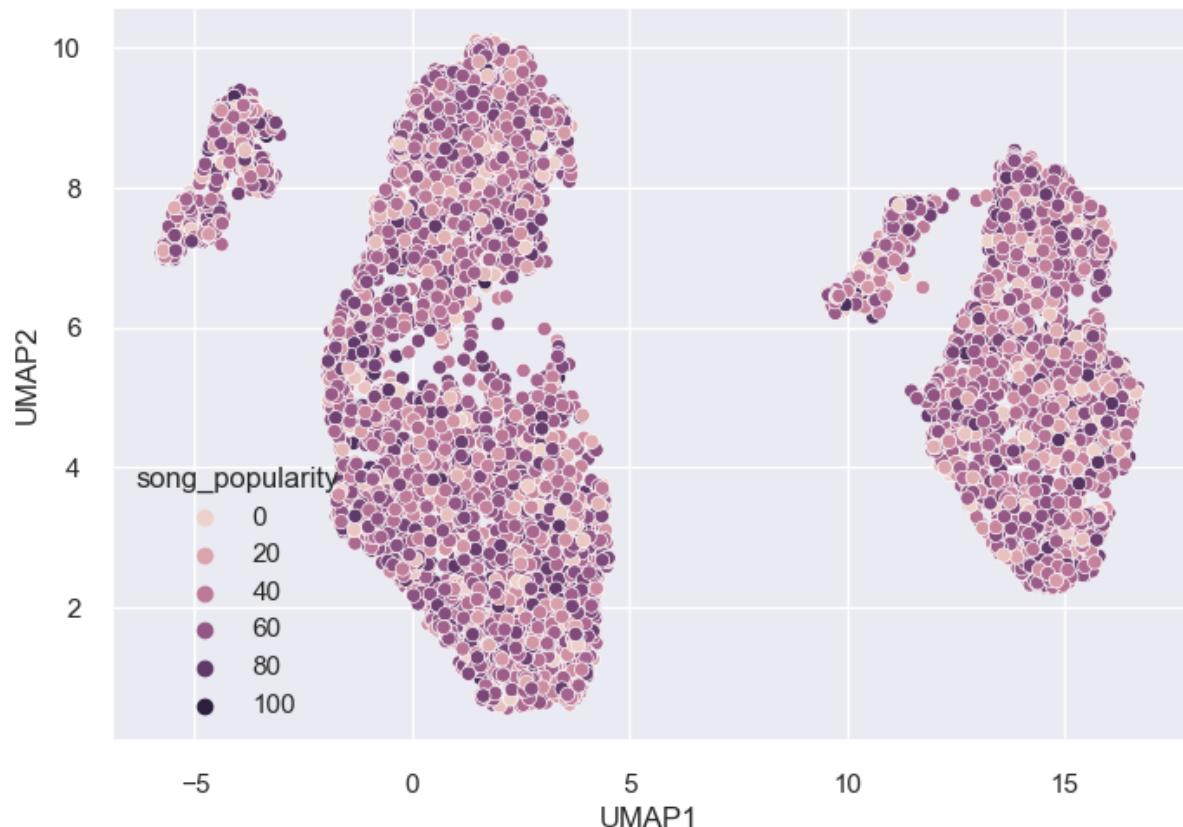
Si se ve en 3 dimensiones, se puede observar que son dos planos:



TSNE:



UMAP:



Conclusiones

Aunque sí que se pueden apreciar que existen distintos tipos de canción, no parece que las canciones más populares sean más similares. Al contrario, estas se encuentran repartidas de manera aparentemente uniforme.

Creando embeddings de las canciones usando word2vec

Por los resultados obtenidos anteriormente, hemos querido comprobar si los datos que tenemos son realmente útiles a la hora de predecir la popularidad de una canción. Es por esto que hemos querido buscar otras formas de representar las canciones que tenemos utilizando datos adicionales. En concreto, hemos encontrado este notebook de Kaggle: [Songs Recommendation - Using Word2Vec on Playlist | Kaggle](#). En él, se muestra como, utilizando un dataset de canciones de Spotify, mucho más grande que el nuestro, se ha creado *embeddings* de las canciones aplicando *word2vec*.

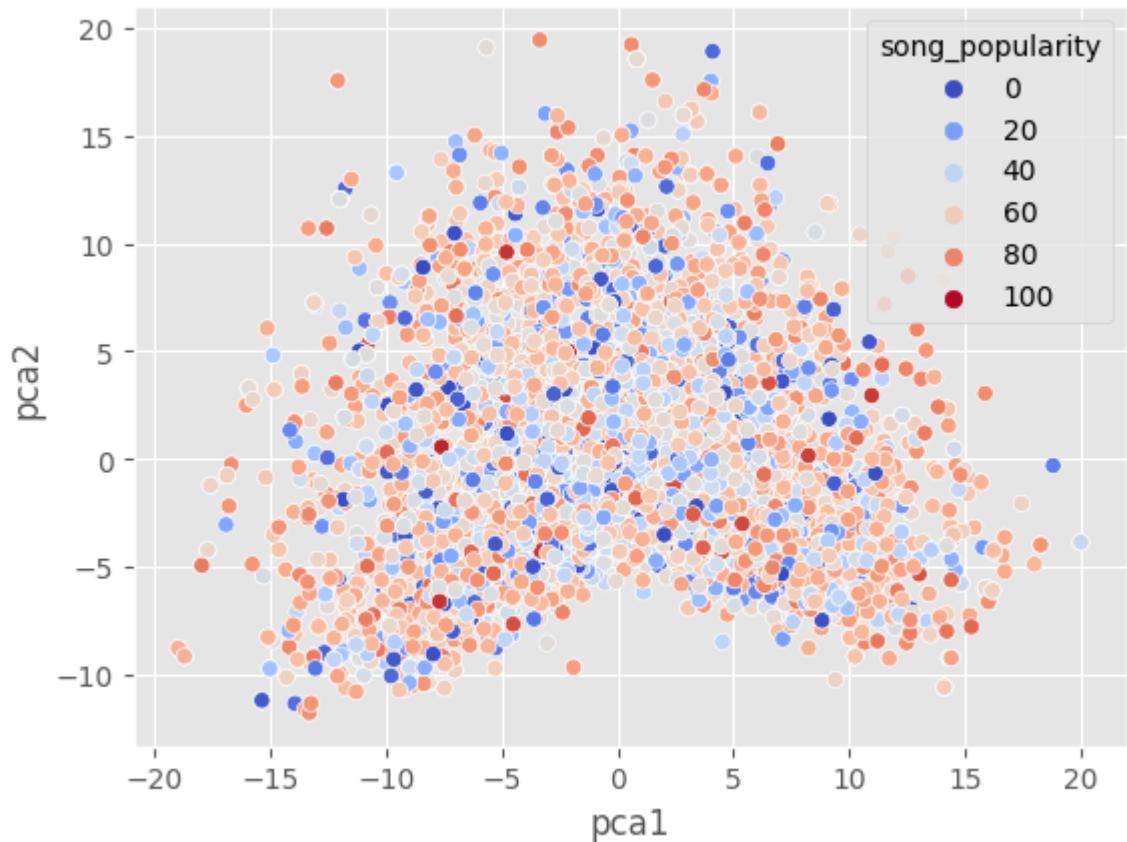
La técnica utilizada ha sido la siguiente:

- Utilizando el nuevo dataset, se crea una lista con las canciones escuchadas por cada usuario, cada una de ellas codificadas con un id.
- Con las listas de reproducción de cada usuario se han creado “frases” con las que posteriormente se ha entrenado el modelo word2vec. De esta forma, el modelo crea una representación de las canciones teniendo en cuenta que canciones se suelen escuchar juntas.

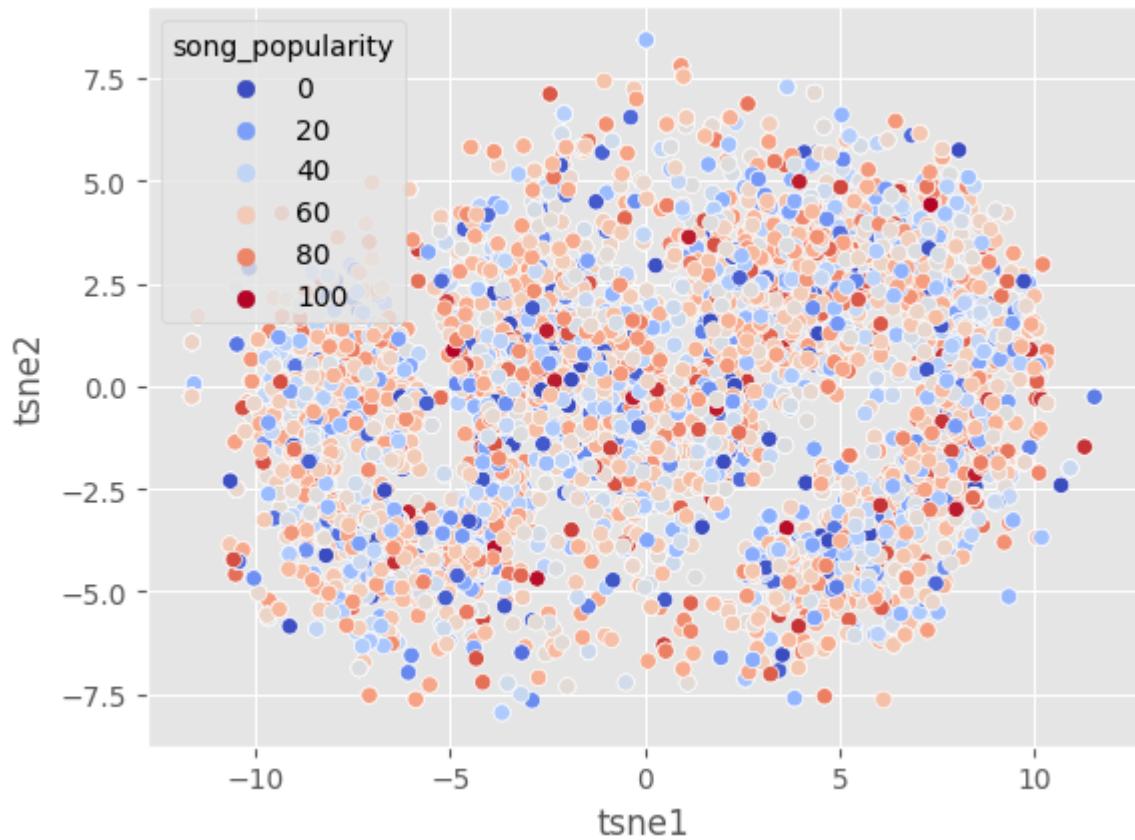
En el caso del notebook, el uso que se le daban a estos embeddings era para recomendar canciones similares. En nuestro caso, los hemos utilizado para realizar nuevas visualizaciones. Para ello, se han seleccionado las canciones de nuestro dataset que aparecen en dataset utilizado para obtener los embeddings. Hemos utilizado el nombre de la canción para determinar que dos canciones son la misma. Si bien esta suposición no es perfecta, es la única que podemos hacer puesto que en el otro dataset no tenemos más información. Además, como hemos podido comprobar, en la mayoría de casos es correcta.

En total, el 64.65% de las canciones de train aparecen en el nuevo dataset si seguimos el criterio anterior. Hay que tener en cuenta que cuánto más popular sea una canción, más probable es que aparezca, y que, por tanto, la nueva muestra está sesgada a incluir canciones más populares. No obstante, este porcentaje es suficientemente alto como para estar seguros de que hay presentes canciones con distintas popularidades y, por tanto, poder realizar un análisis.

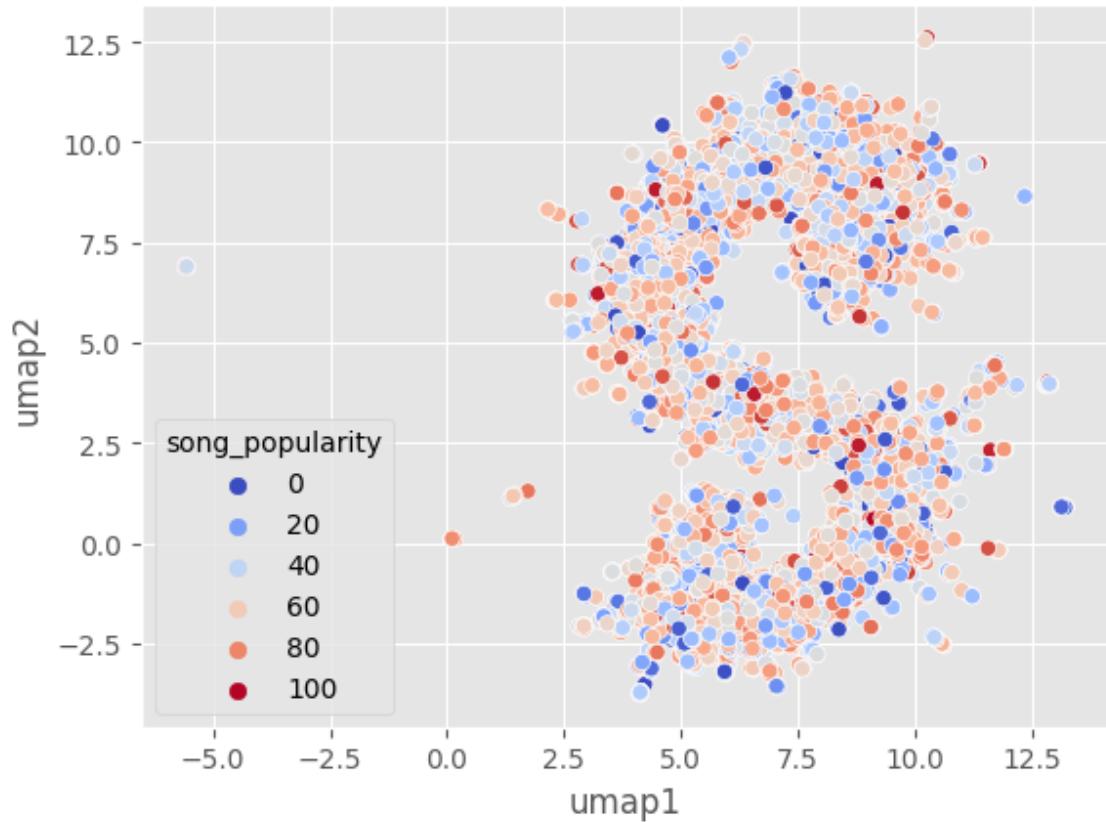
PCA:



TSNE:



UMAP:



Conclusiones

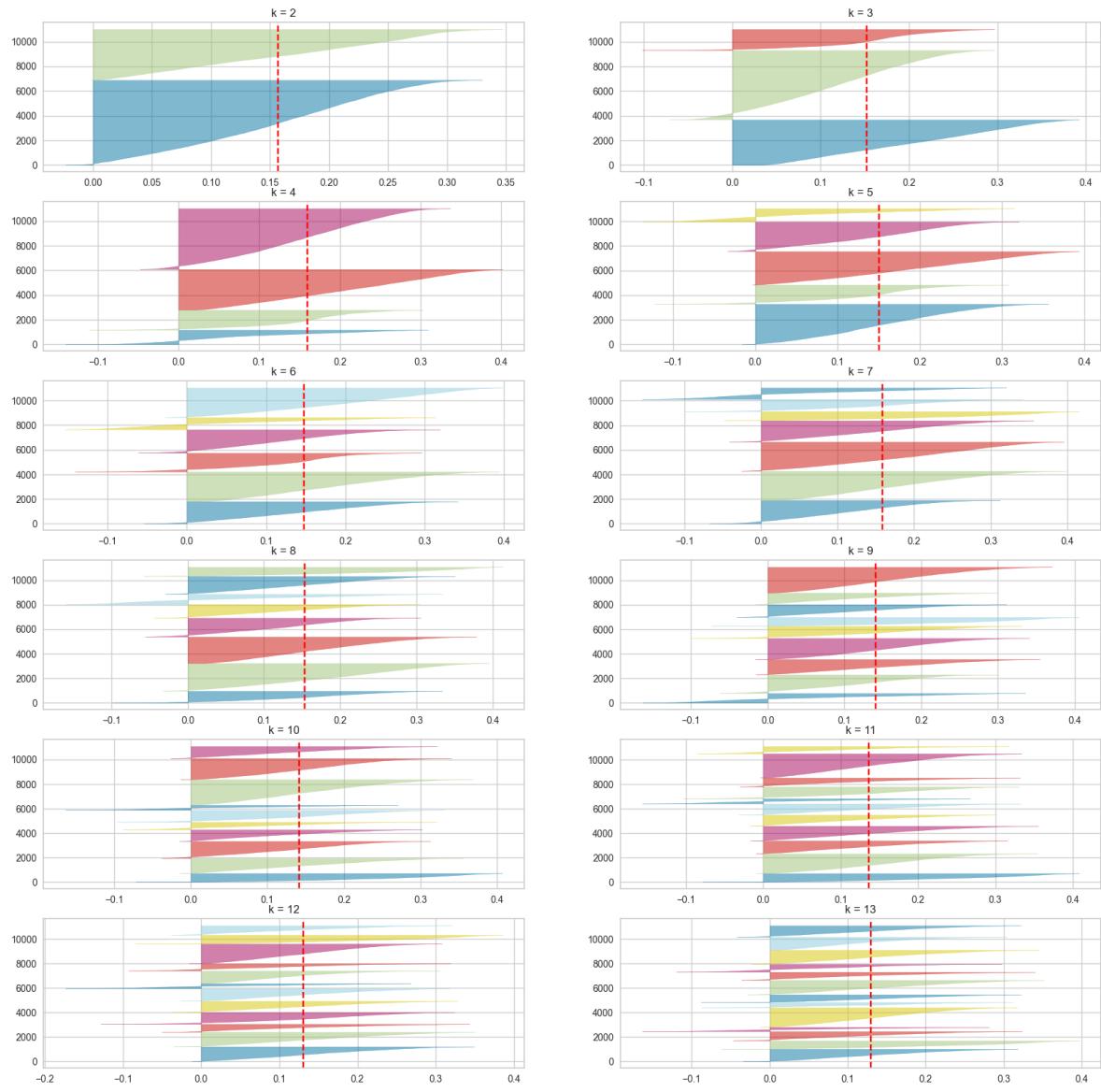
Usando esta nueva representación de las canciones llegamos a las mismas conclusiones que anteriormente. No es posible encontrar una agrupación clara de canciones muy populares. Esto nos lleva a pensar que, el hecho de que una canción sea muy popular, no depende tanto de sus características.

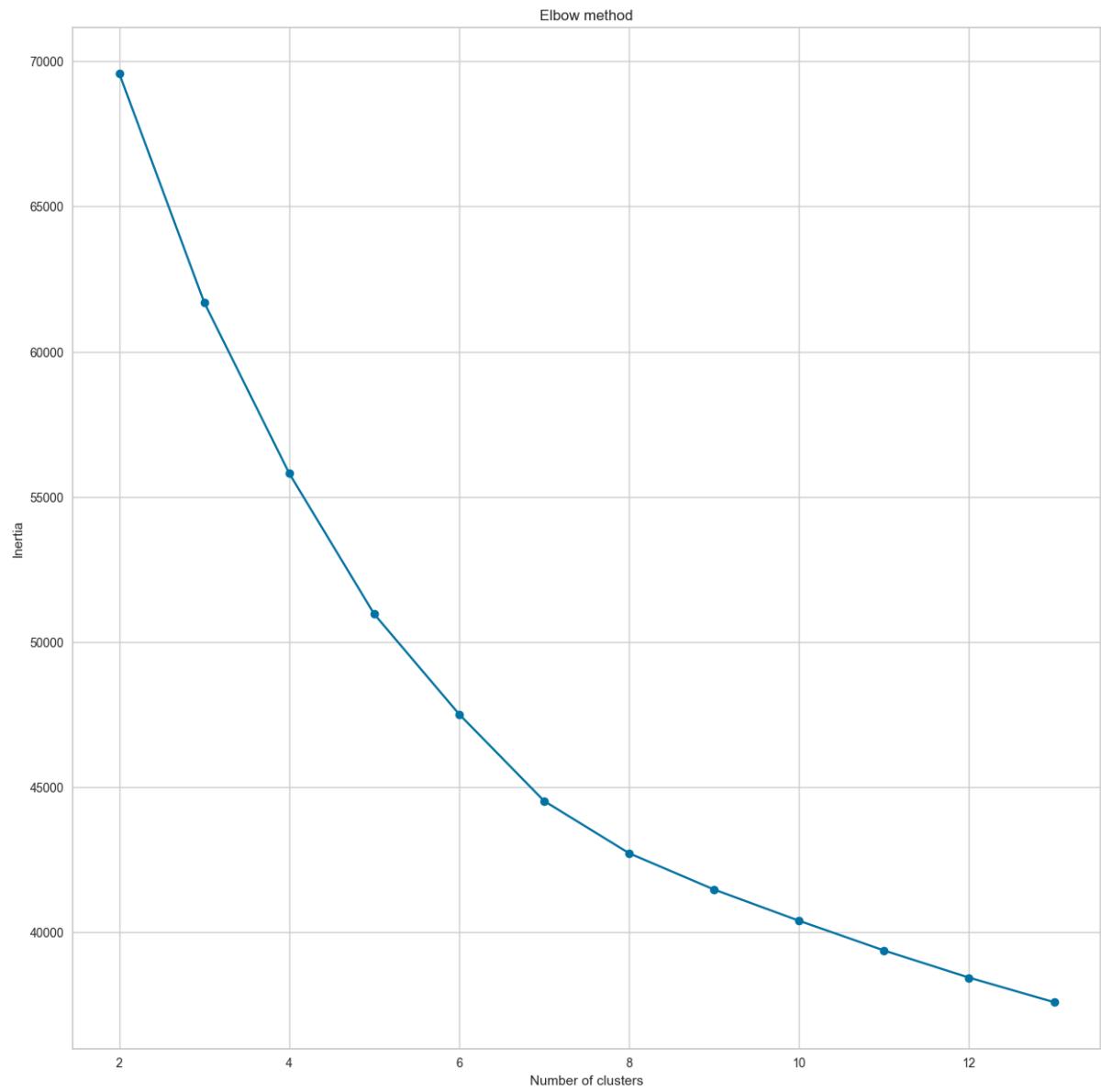
Visualizando clústeres de canciones

Todo parece indicar que las canciones más populares pueden ser de tipos muy diferentes. Esto tiene sentido ya que las canciones de nuestro dataset son muy variadas: hemos encontrado canciones en varios idiomas, por ejemplo. Es por esto que hemos decidido realizar un análisis agrupando por tipo de canción. Nuestro objetivo es realizar este mismo análisis pero para cada tipo de canción. Si, por ejemplo, un cluster consigue agrupar la mayoría de canciones de rap, quizás en ese clúster se pueda observar qué características deben tener dichas canciones para ser más populares.

Para realizar esta prueba, decidimos probar con el dataset original una vez más, pero con el *feature extraction* del título, y utilizando a su vez un algoritmo de clustering (*KMeans*) para verificar cómo de bien se está efectuando la reducción de dimensiones. Primero, probamos con un preprocessado distinto para cada columna, que incluye MinMaxScaler, StandardScaler, PowerTransformer. Este es el preprocessado que se hizo para tratar de predecir la popularidad aplicando regresión lineal. A continuación, se muestran los resultados obtenidos con el mismo.

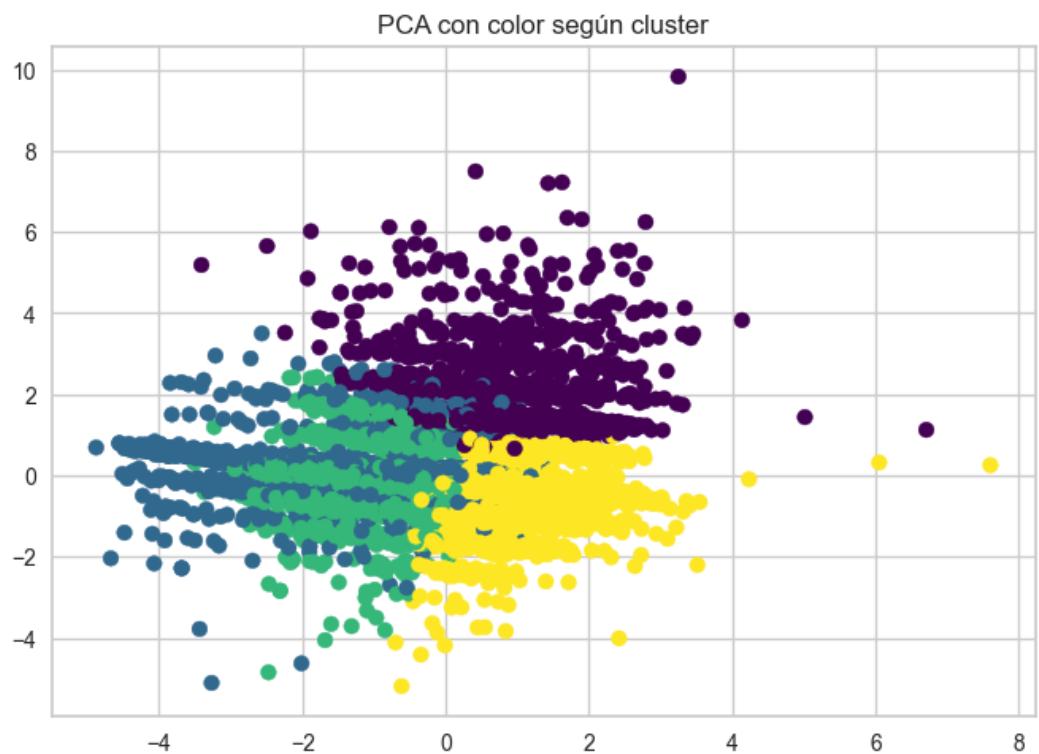
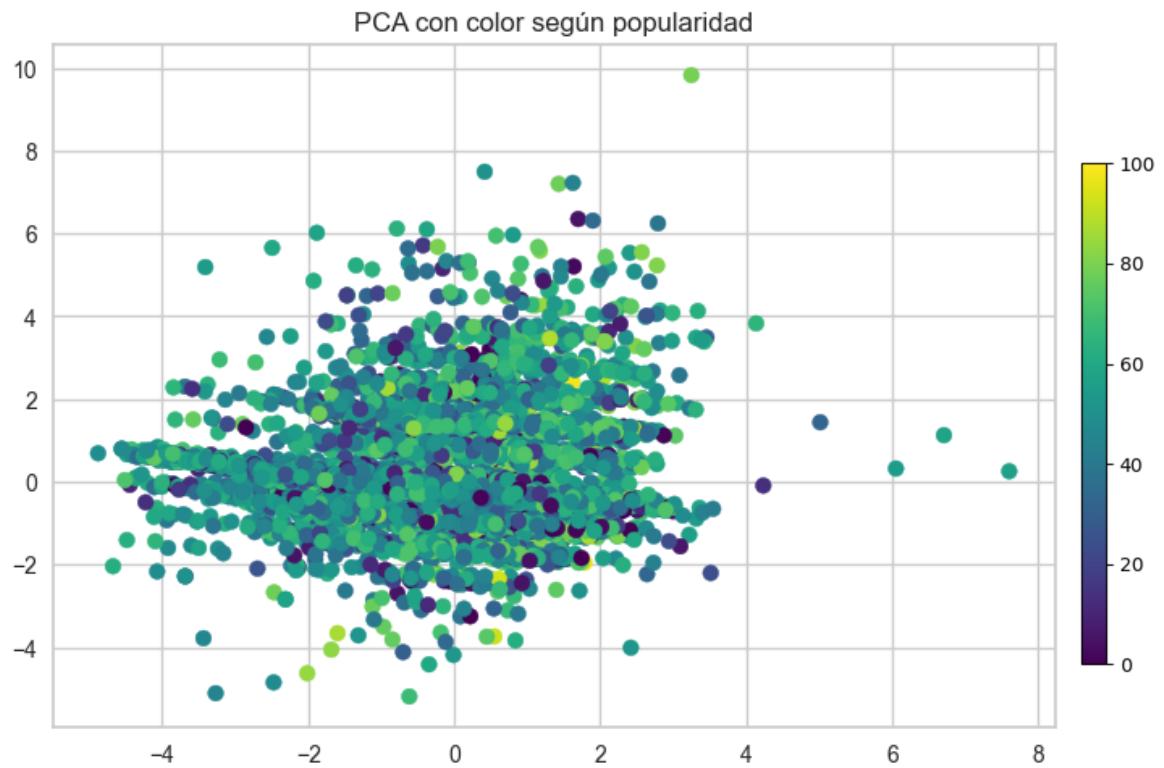
Primero, buscamos el mejor número de clusters usando silueta e inercia



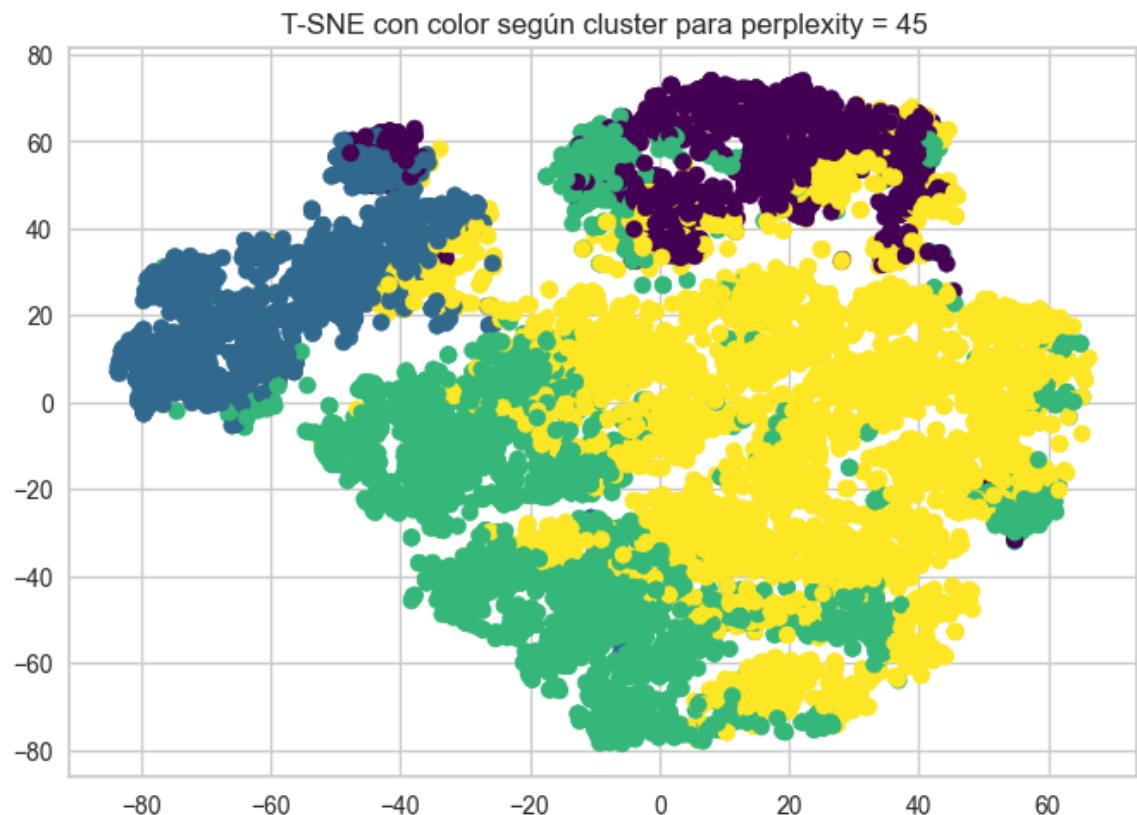
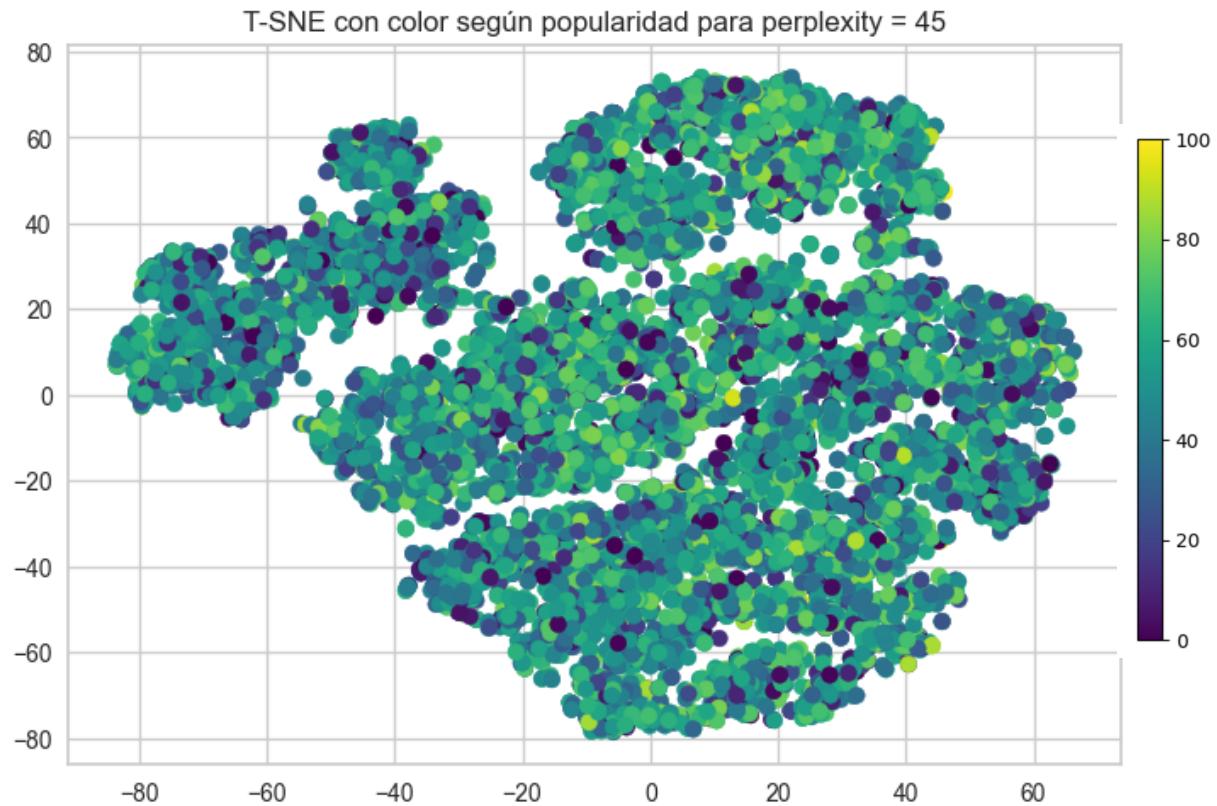


Aplicando el método del codo y basándonos en los resultados de la métrica silueta, hemos escogido $k=4$.

PCA:

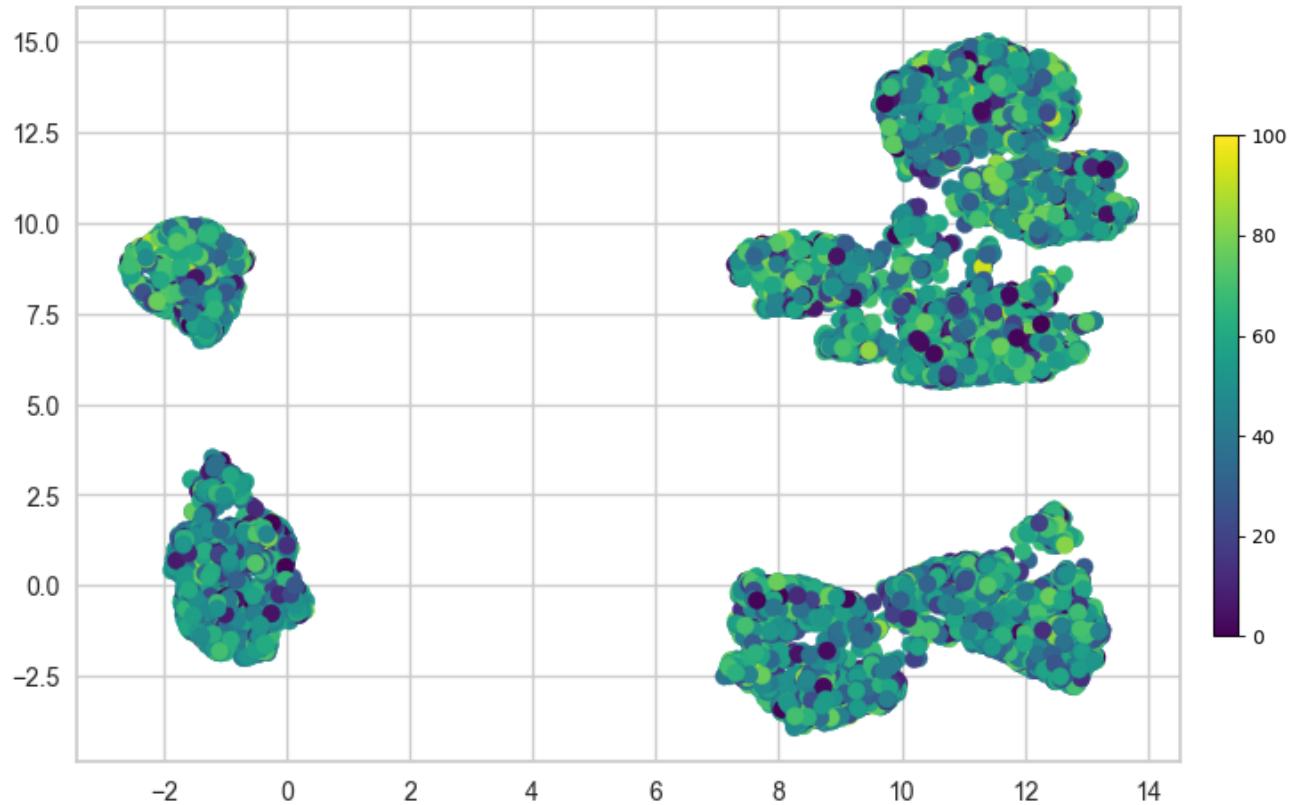


TSNE:

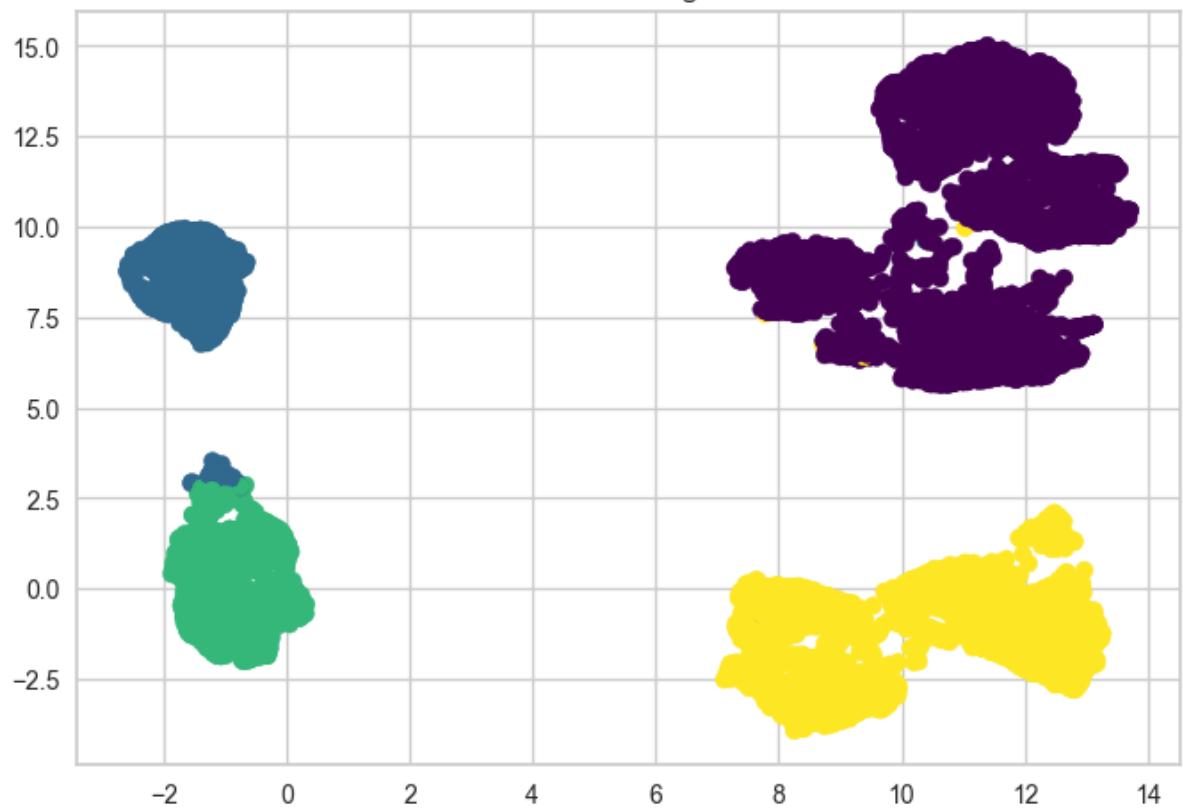


UMAP:

UMAP con color según popularidad

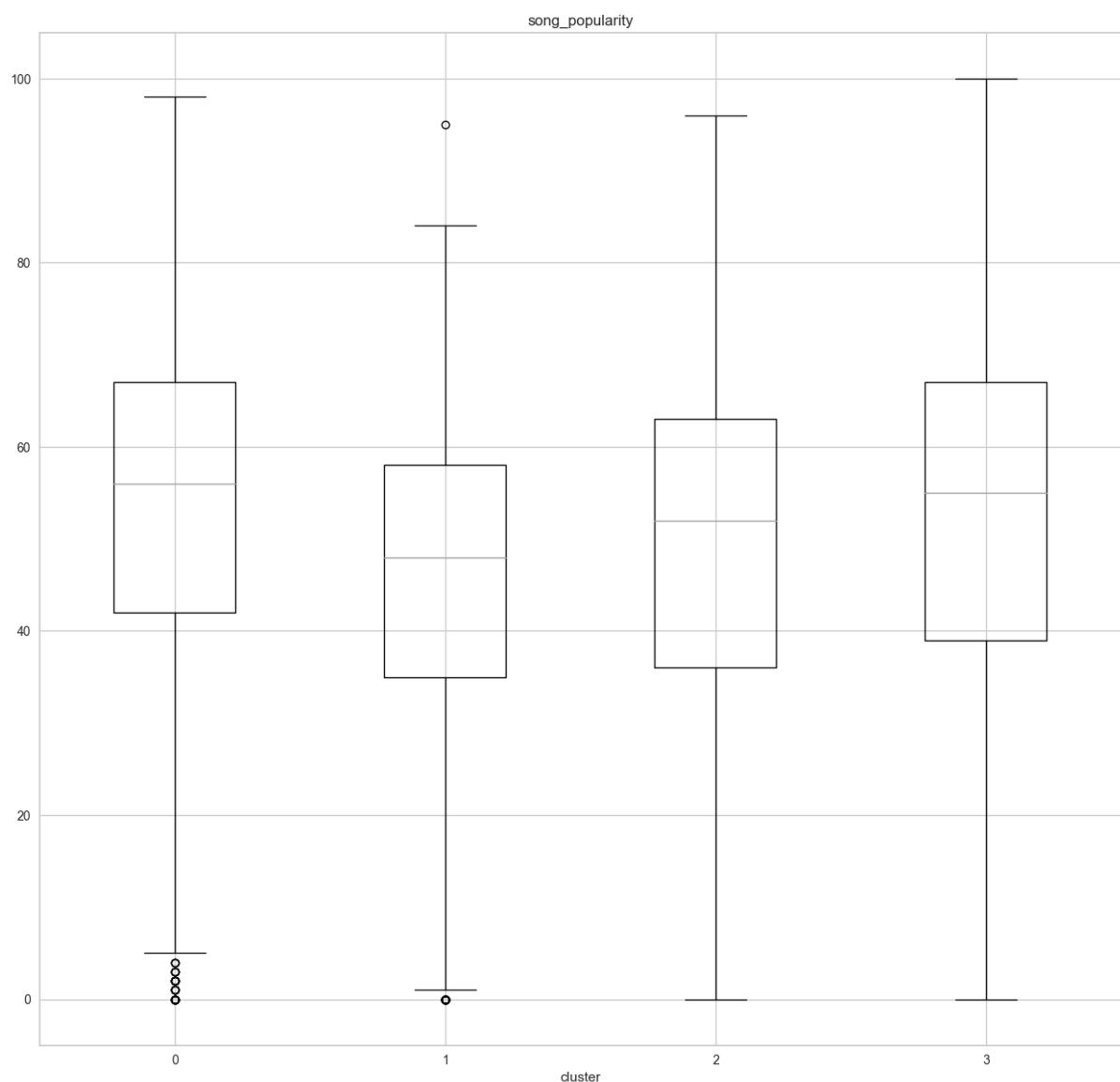


UMAP con color según cluster

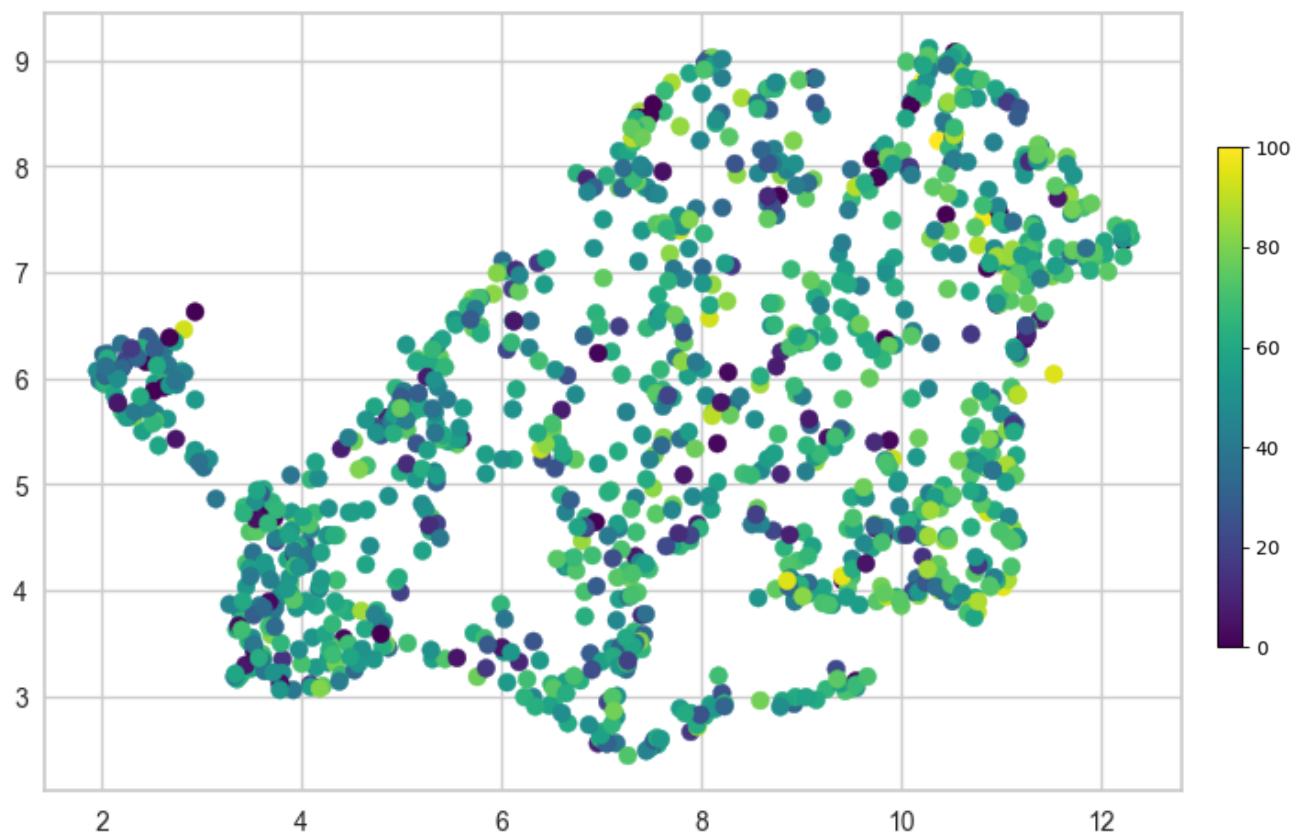


Distribución de *song_popularity* por clúster:

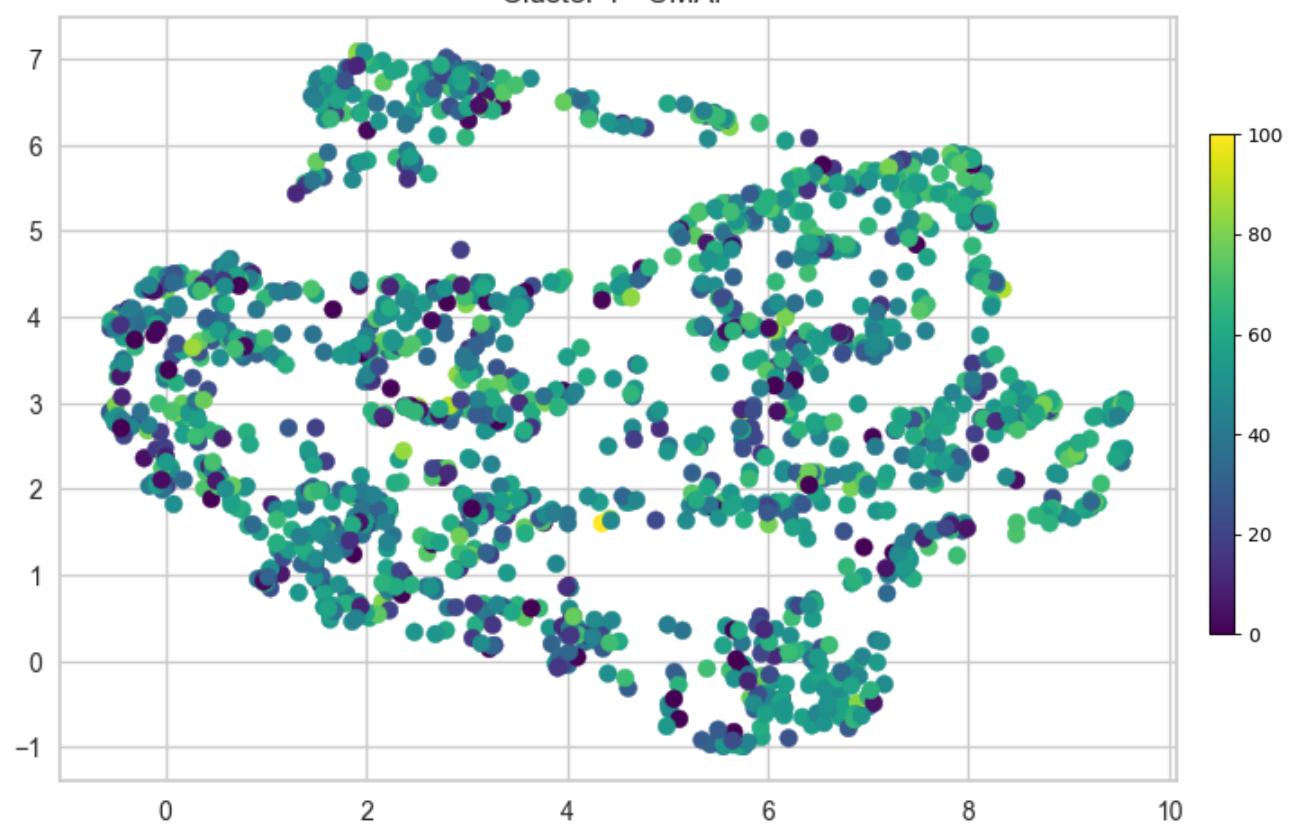
Boxplot grouped by cluster



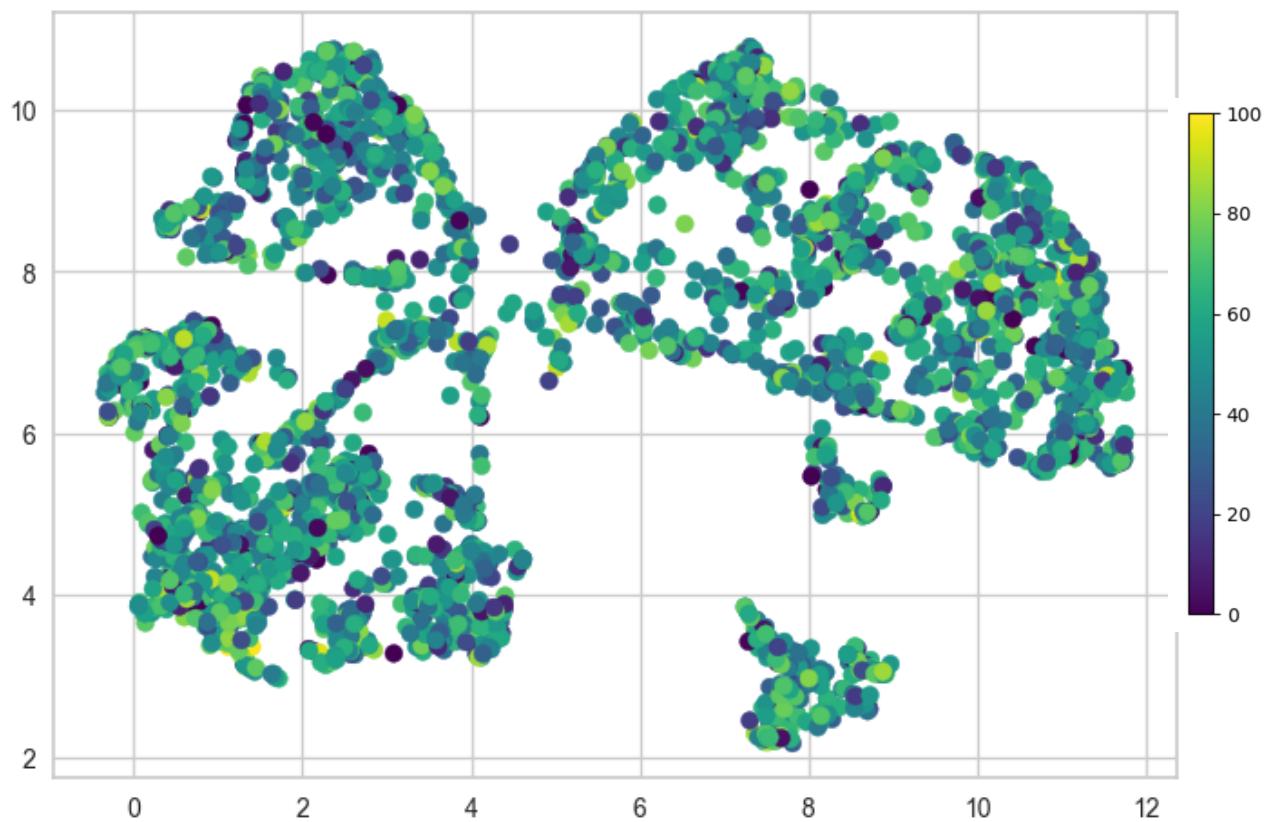
Cluster 0 - UMAP



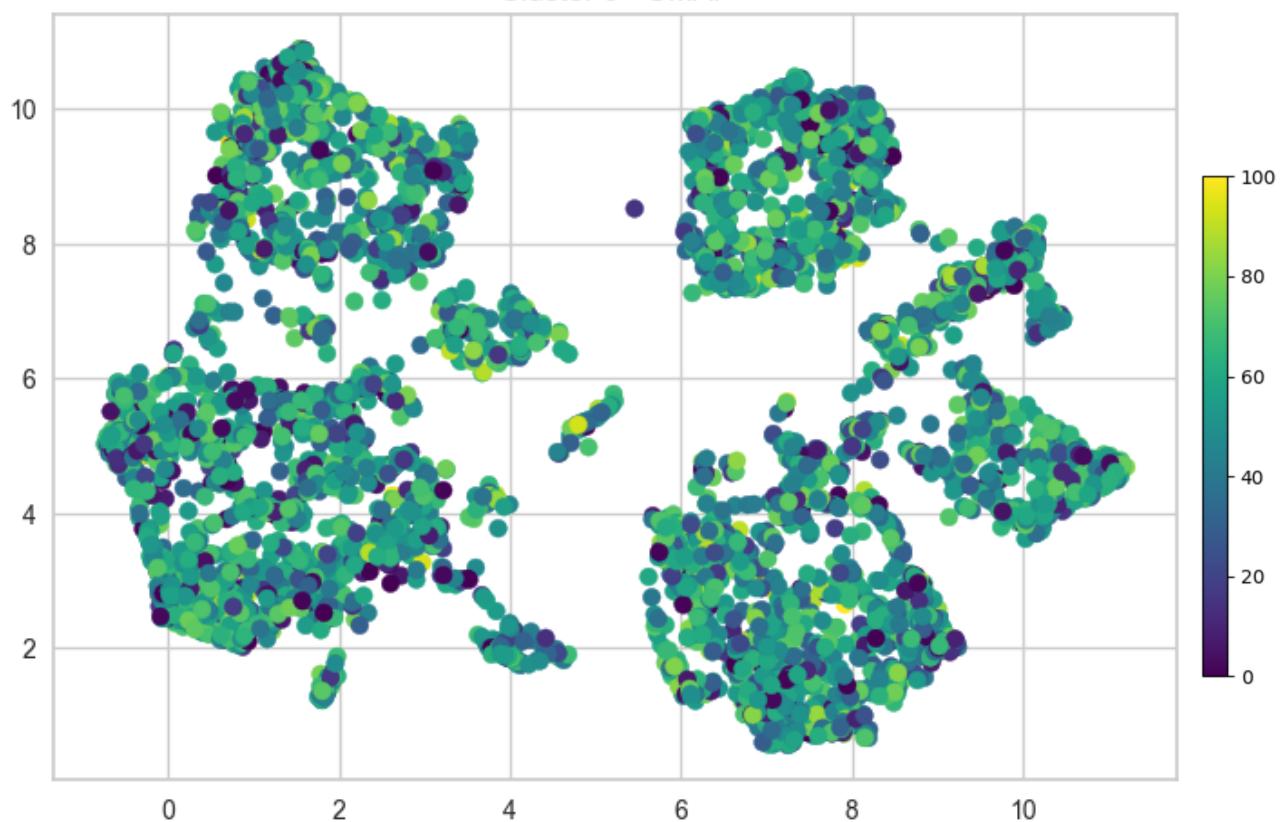
Cluster 1 - UMAP



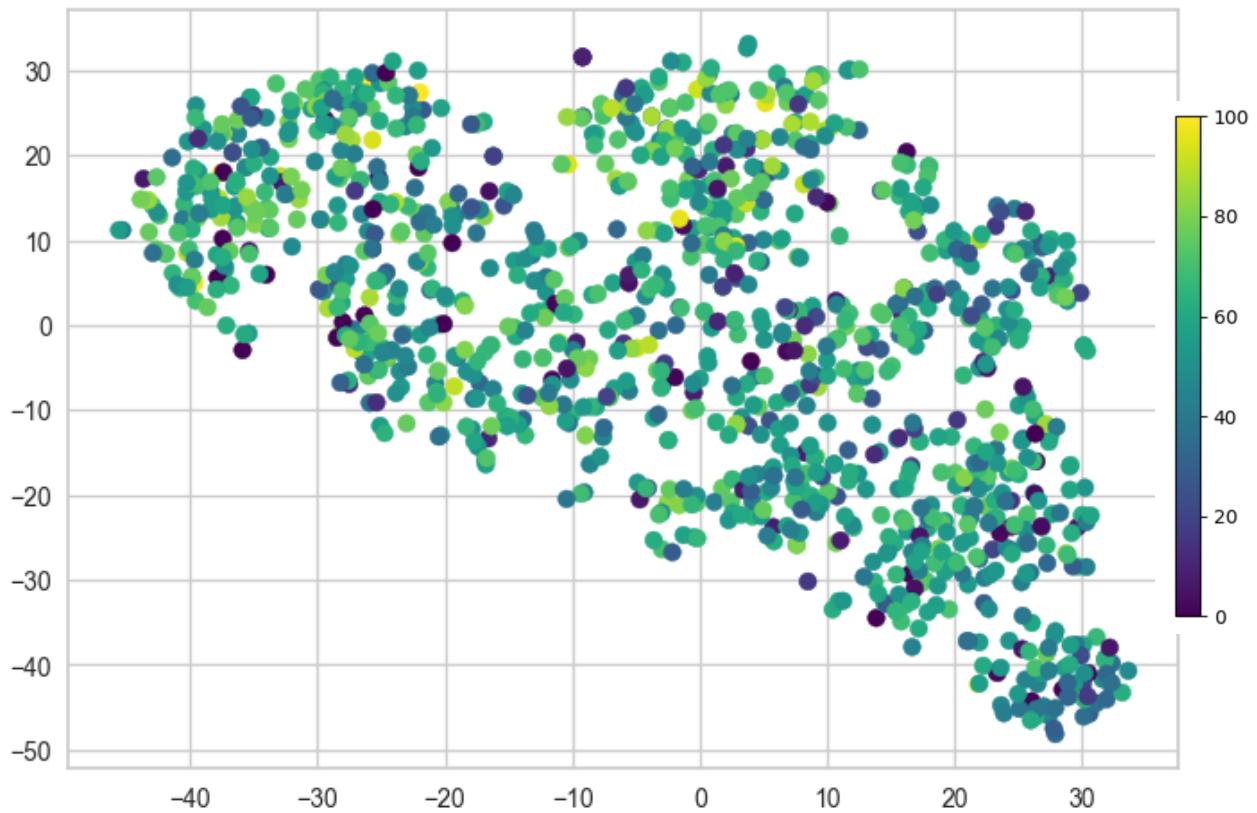
Cluster 2 - UMAP



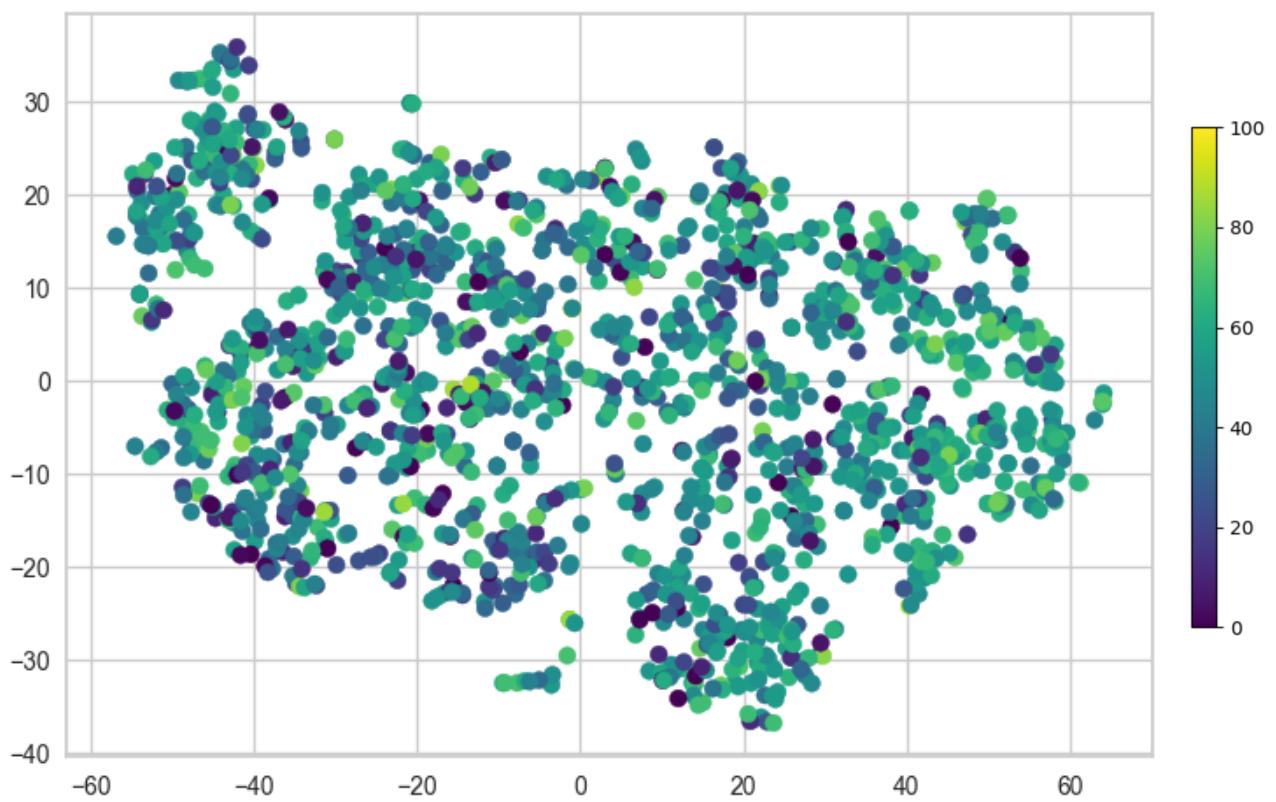
Cluster 3 - UMAP



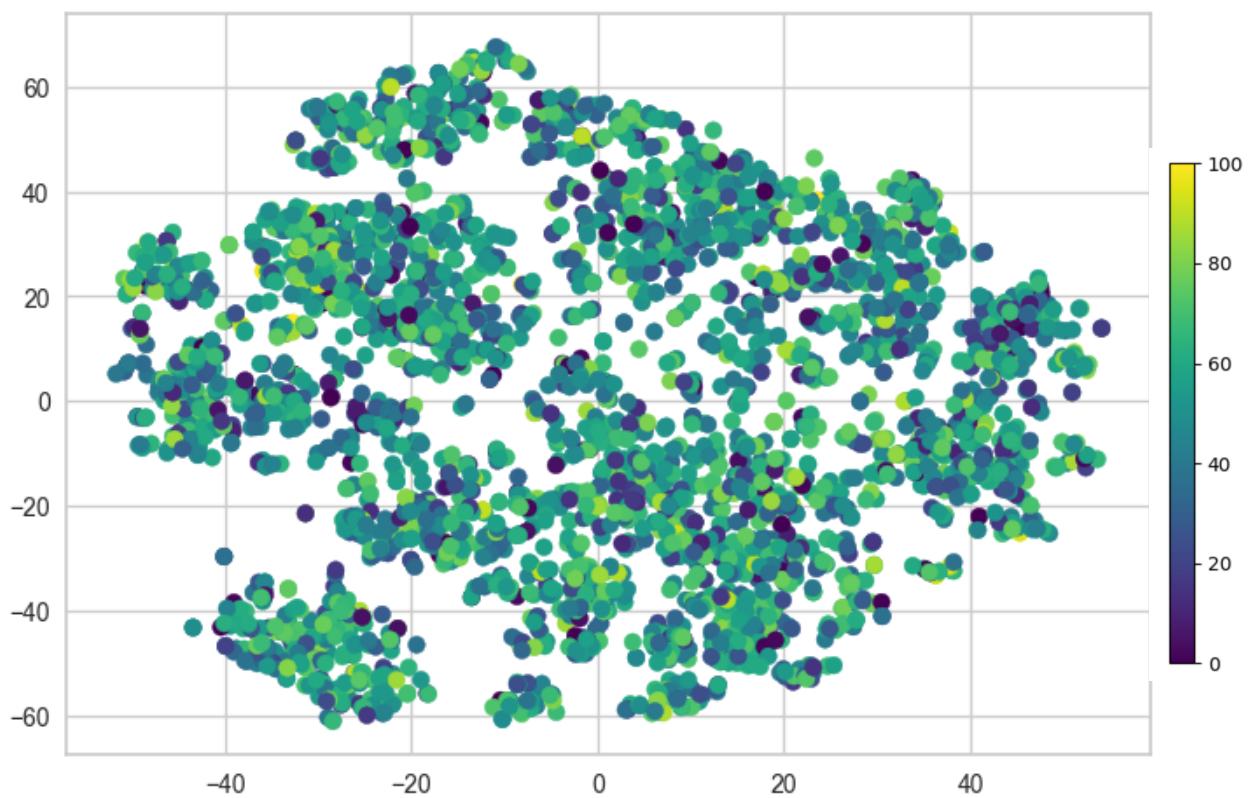
Cluster 0 - TSNE



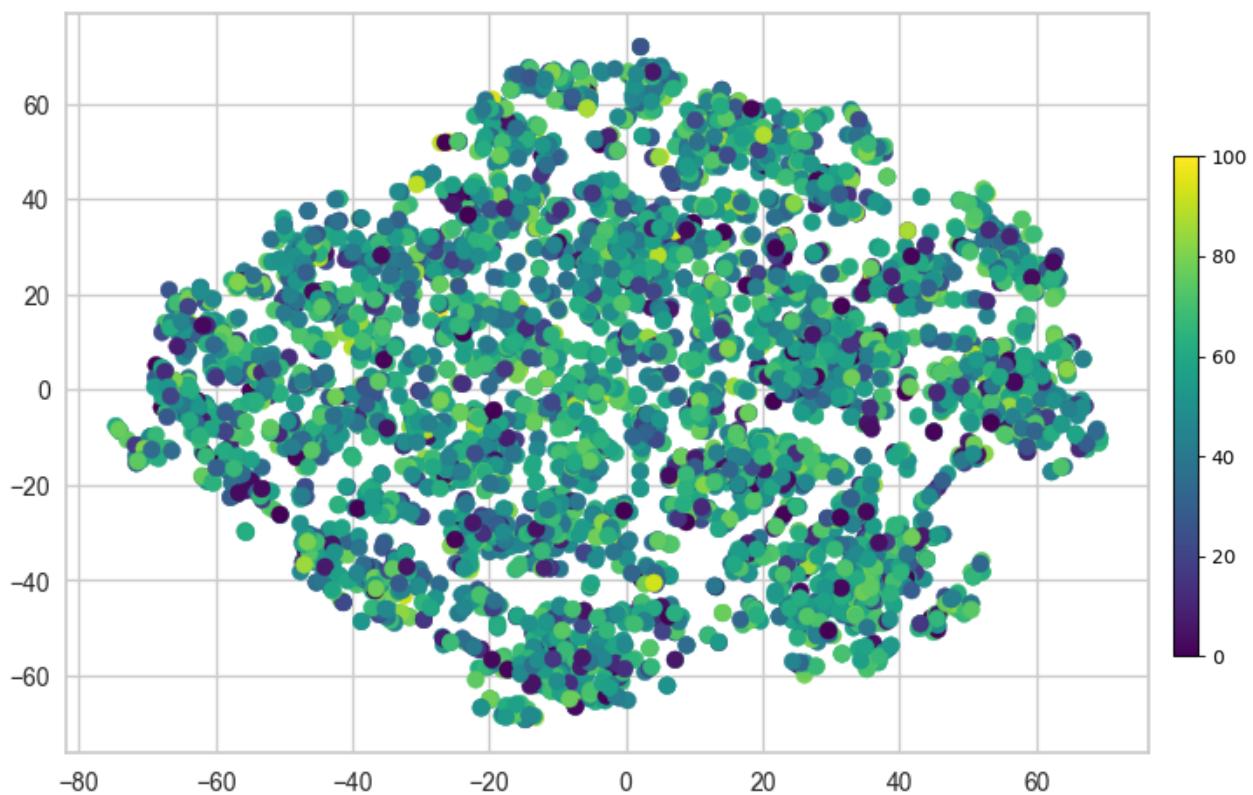
Cluster 1 - TSNE



Cluster 2 - TSNE



Cluster 3 - TSNE

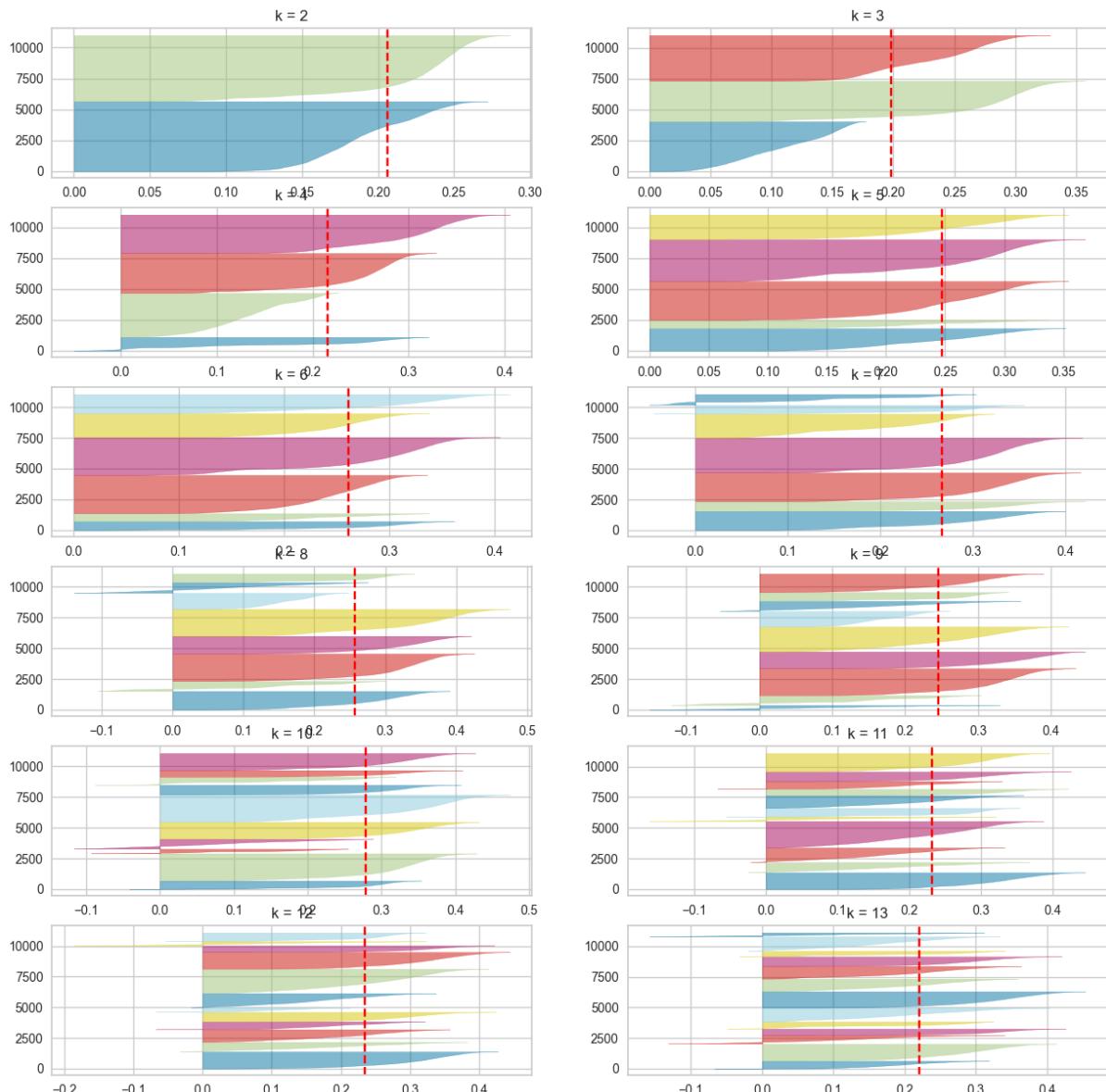


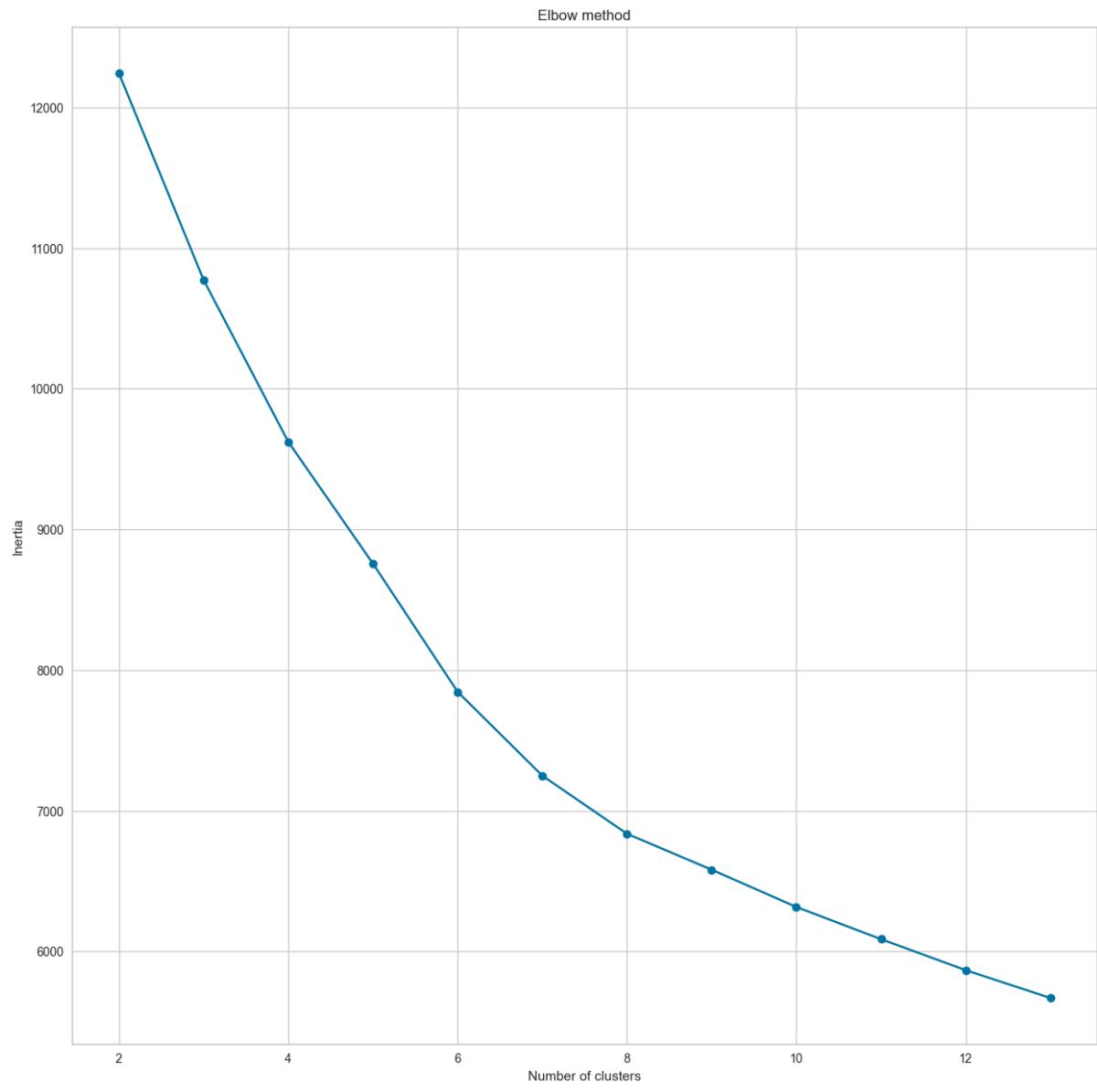
El color indica la popularidad de la canción en la escala mostrada a la derecha de la gráfica.

Aunque se aprecia una ligera diferencia en la distribución de *song_popularity* en el clúster 1, una vez más, no se aprecia una agrupación clara según la popularidad.

Tras esto, probamos con un preprocesado más sencillo, usando *MinMaxScaler*, ya que en el anterior preprocesado se modifica la distribución que siguen los datos, cambiando la visualización.

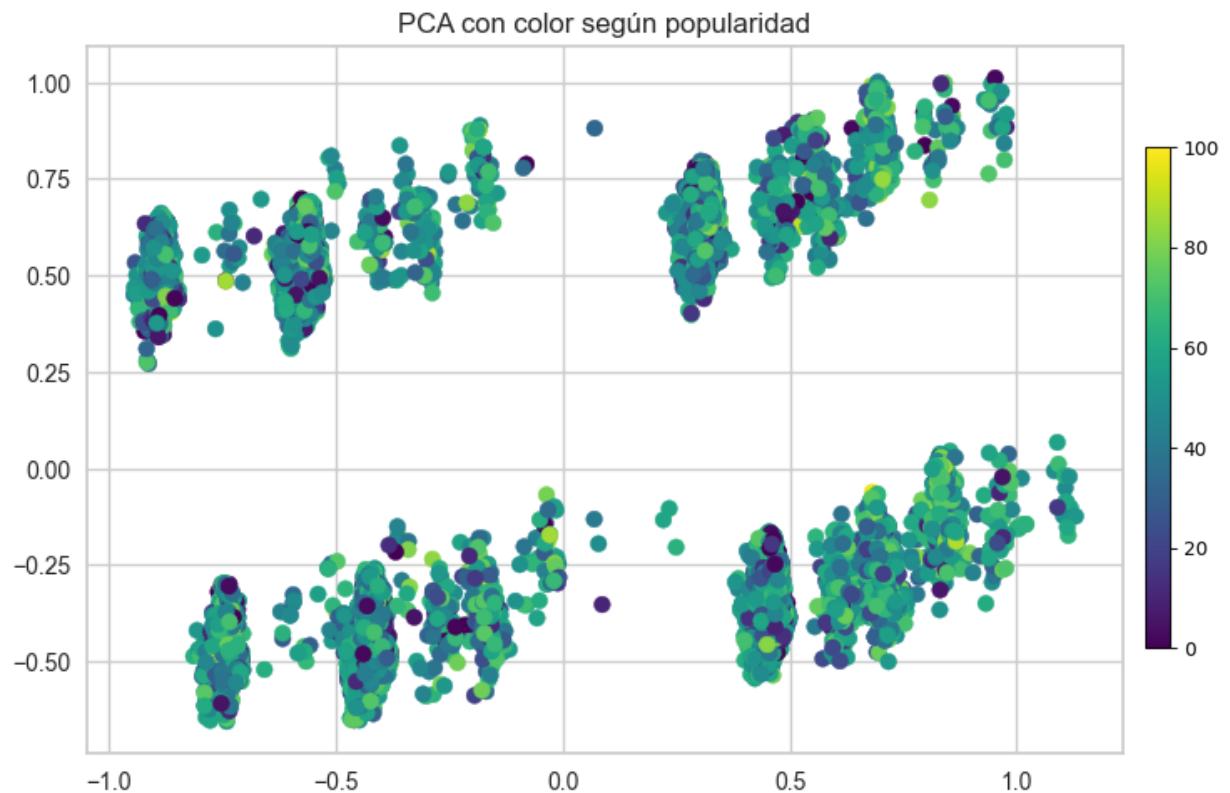
Como hemos hecho anteriormente, escogemos el número de clústers ideal:



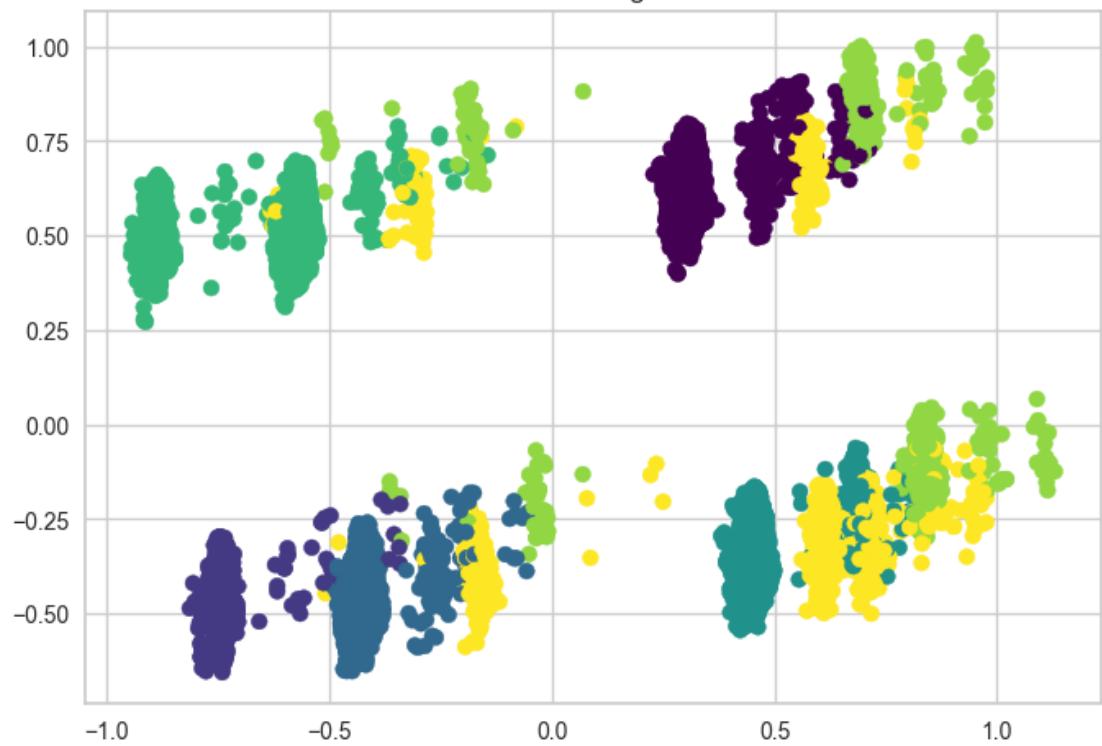


En este caso, hemos escogido $k=7$.

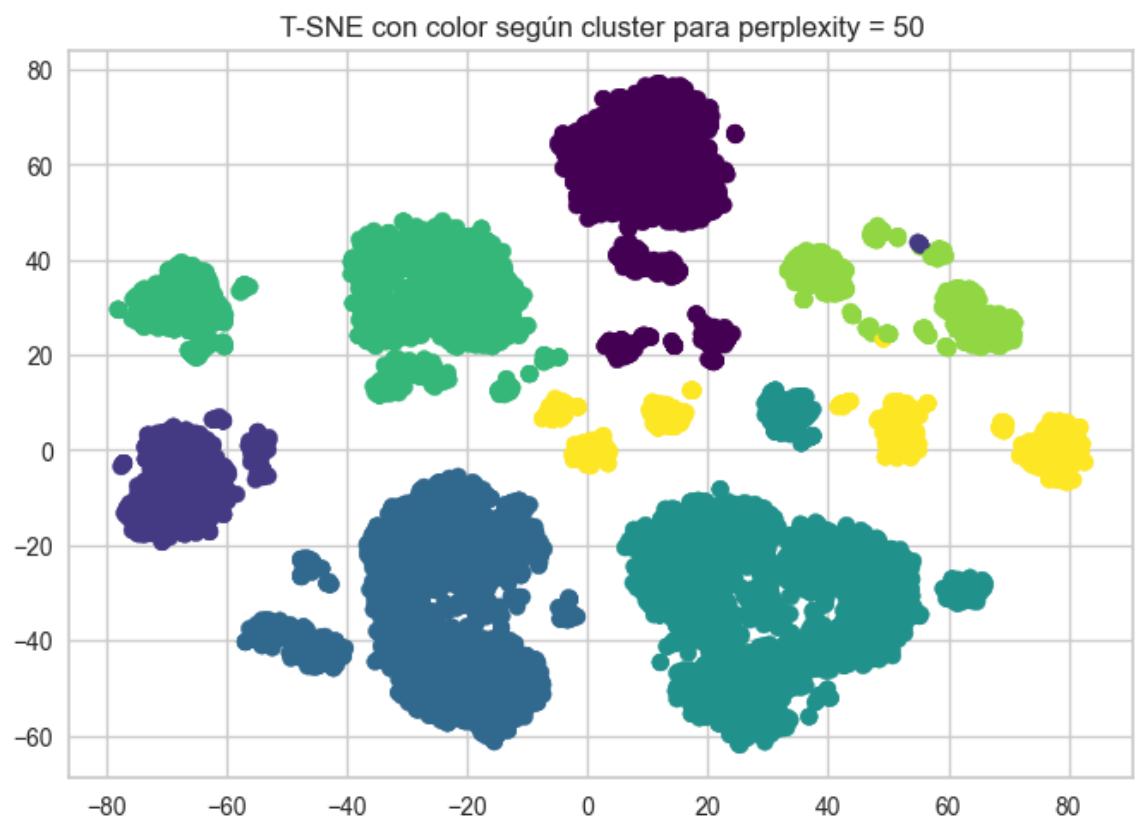
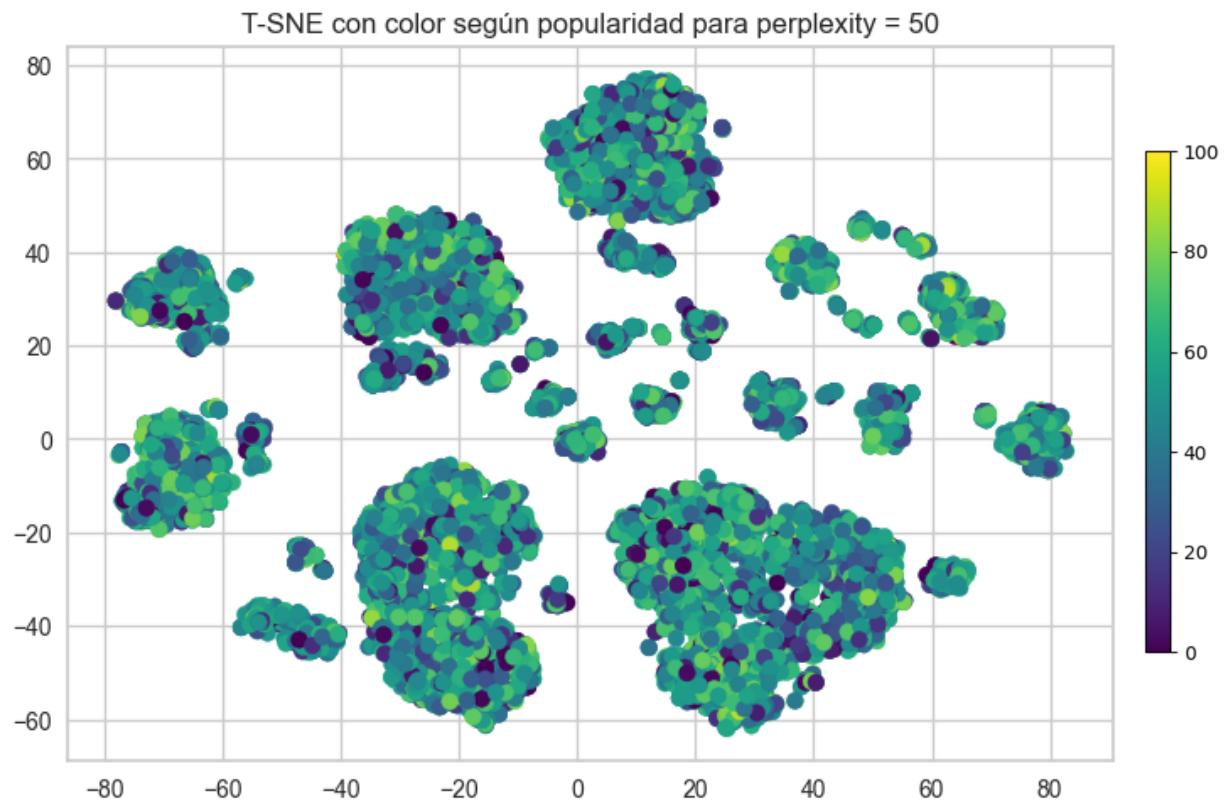
PCA:



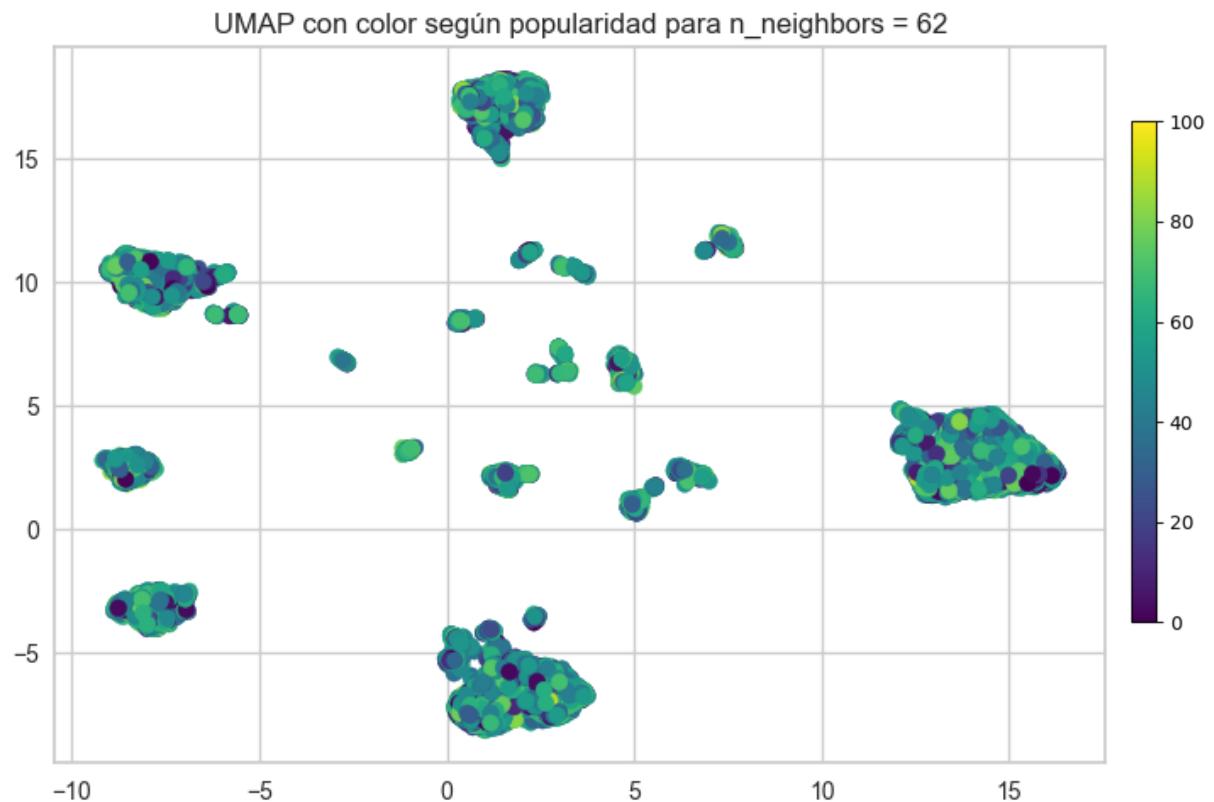
PCA con color según cluster



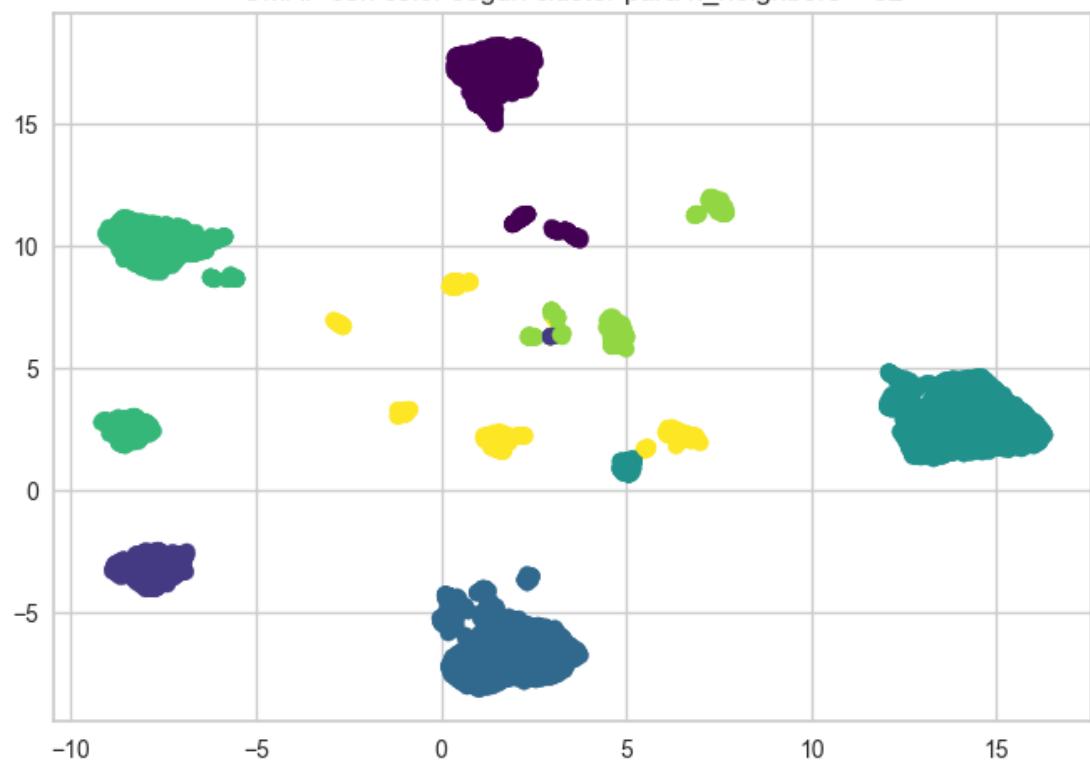
T-SNE:



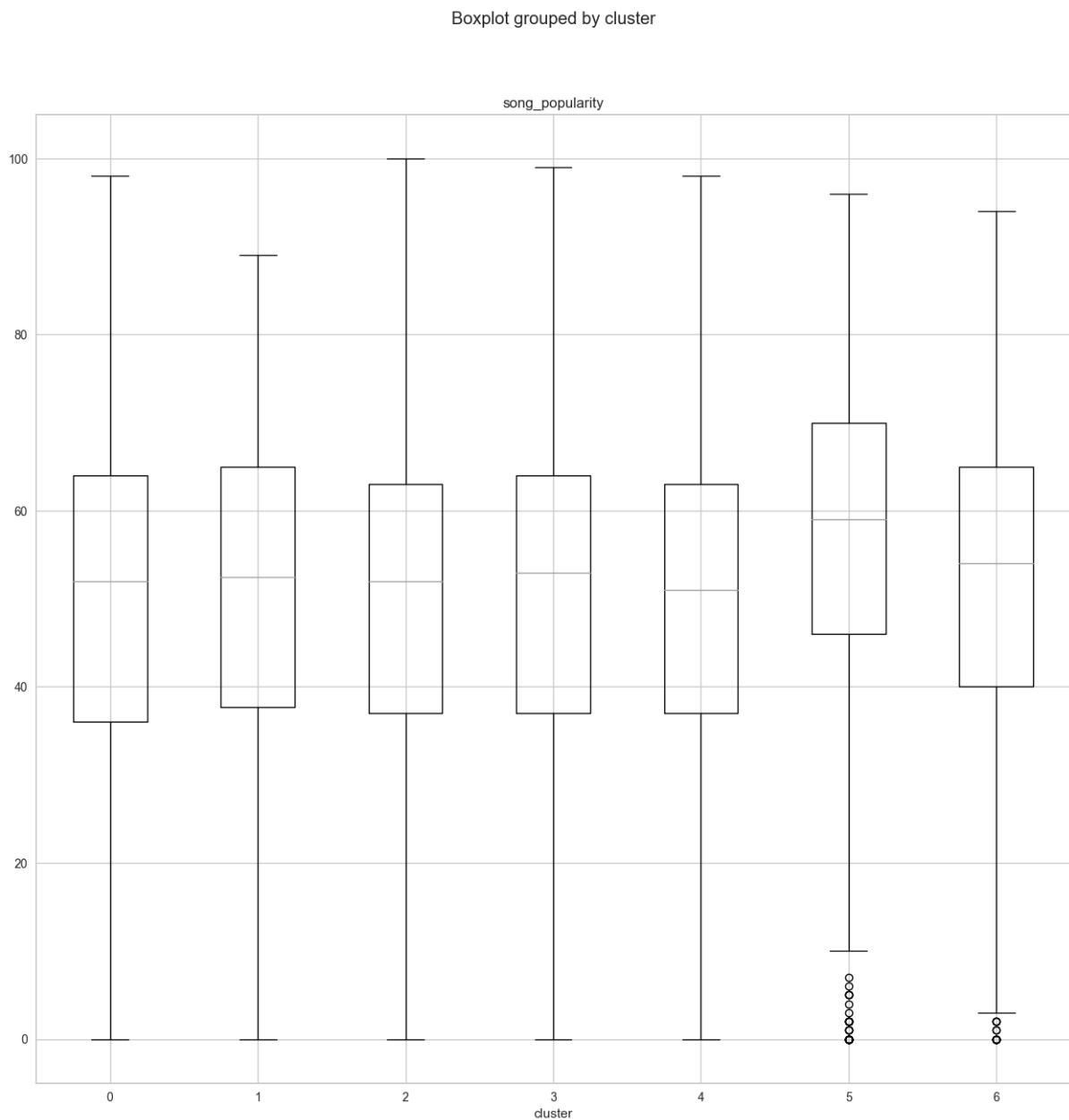
UMAP:



UMAP con color según cluster para n_neighbors = 62

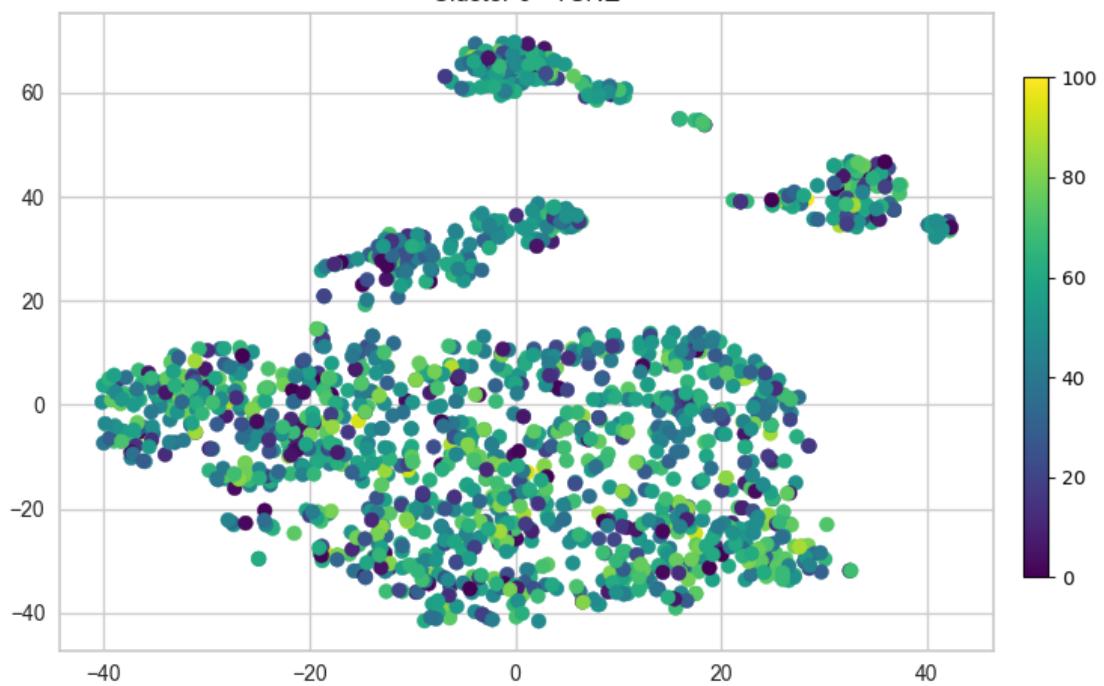


Distribución de *song_popularity* por clúster:

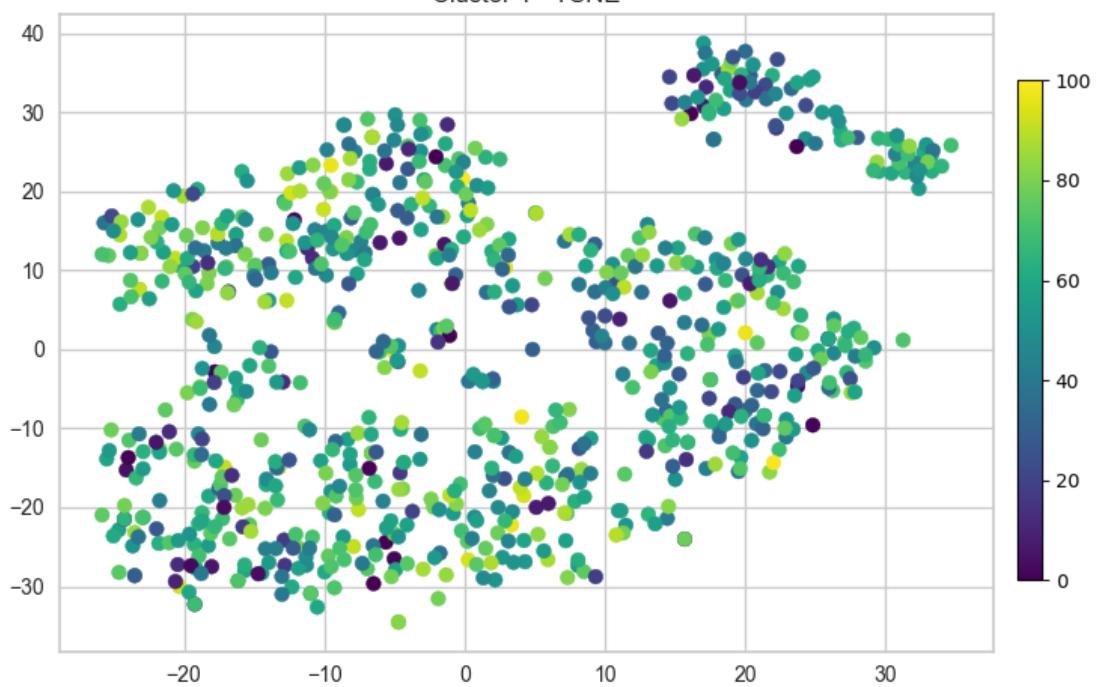


En este caso, no se aprecia ninguna diferencia significativa en la distribución de *song_popularity*.

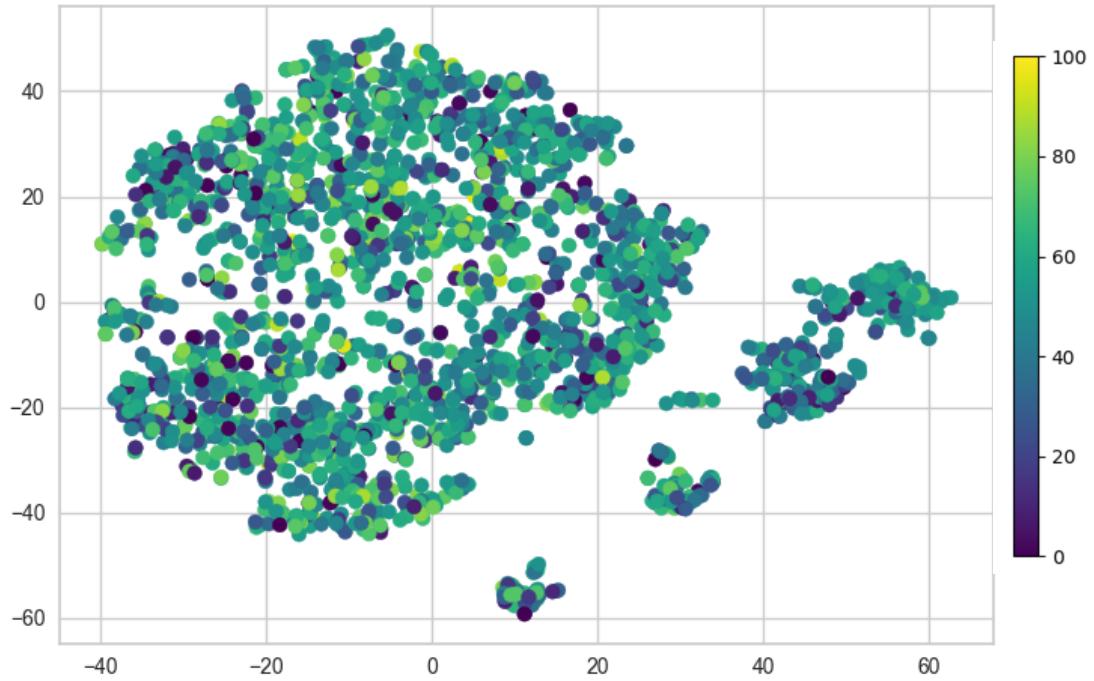
Cluster 0 - TSNE



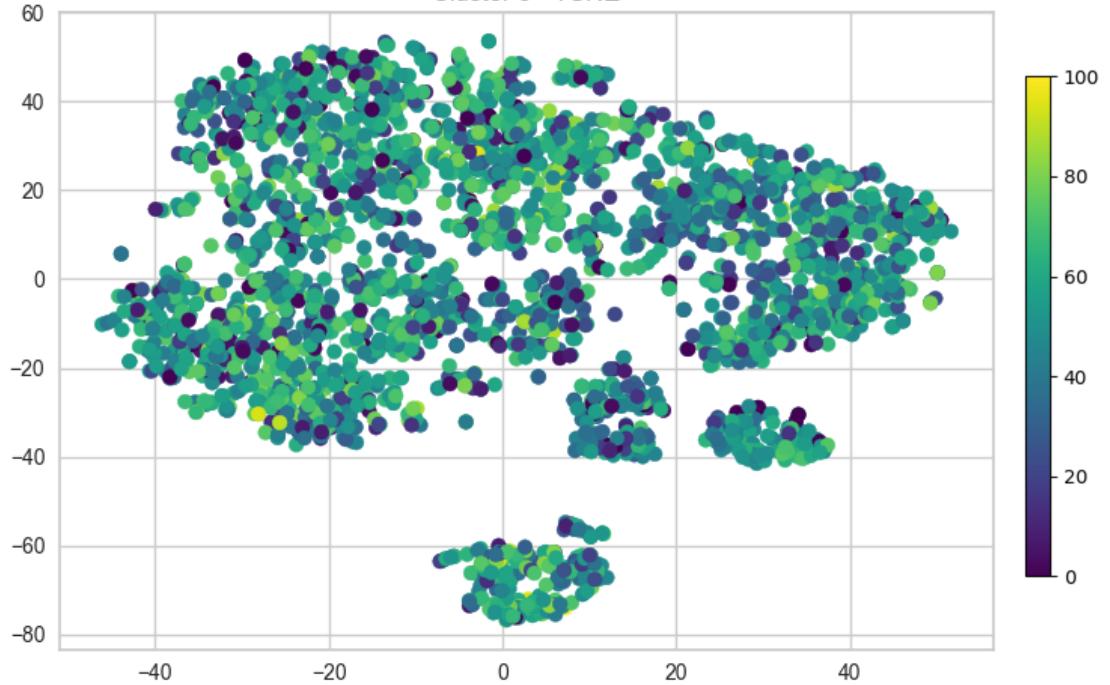
Cluster 1 - TSNE



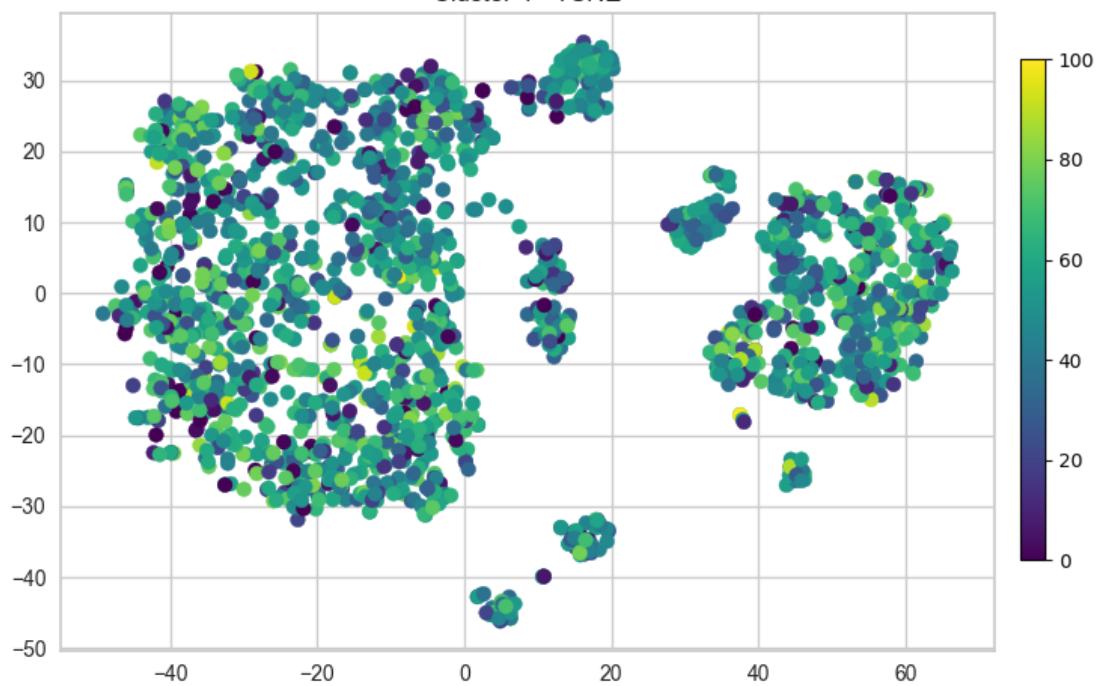
Cluster 2 - TSNE



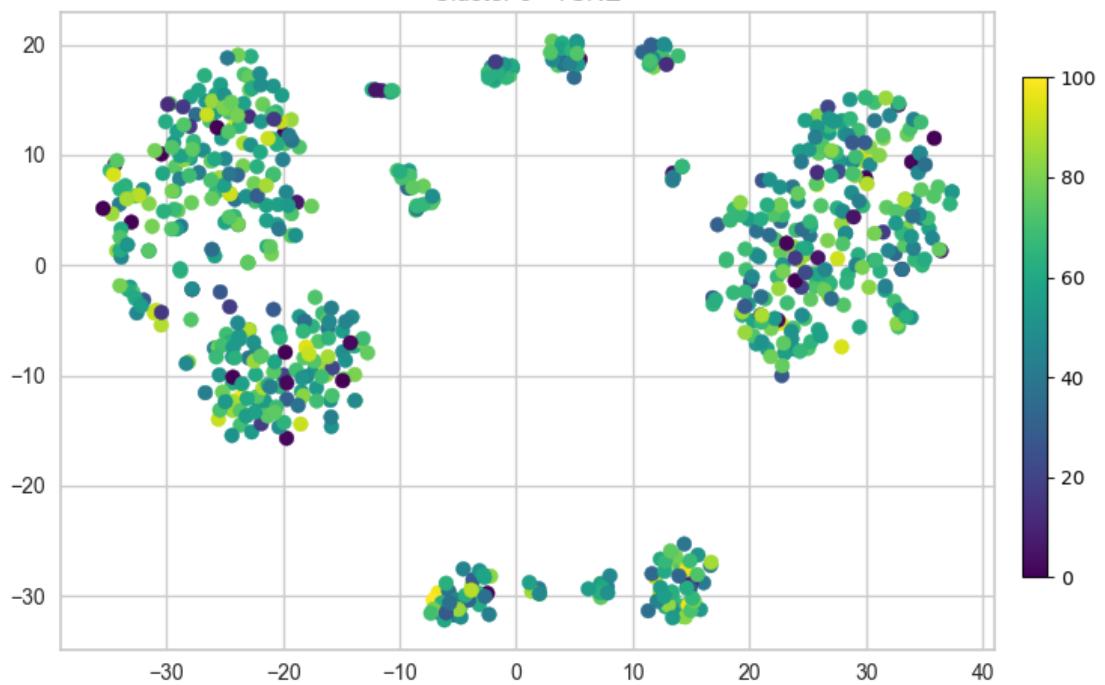
Cluster 3 - TSNE



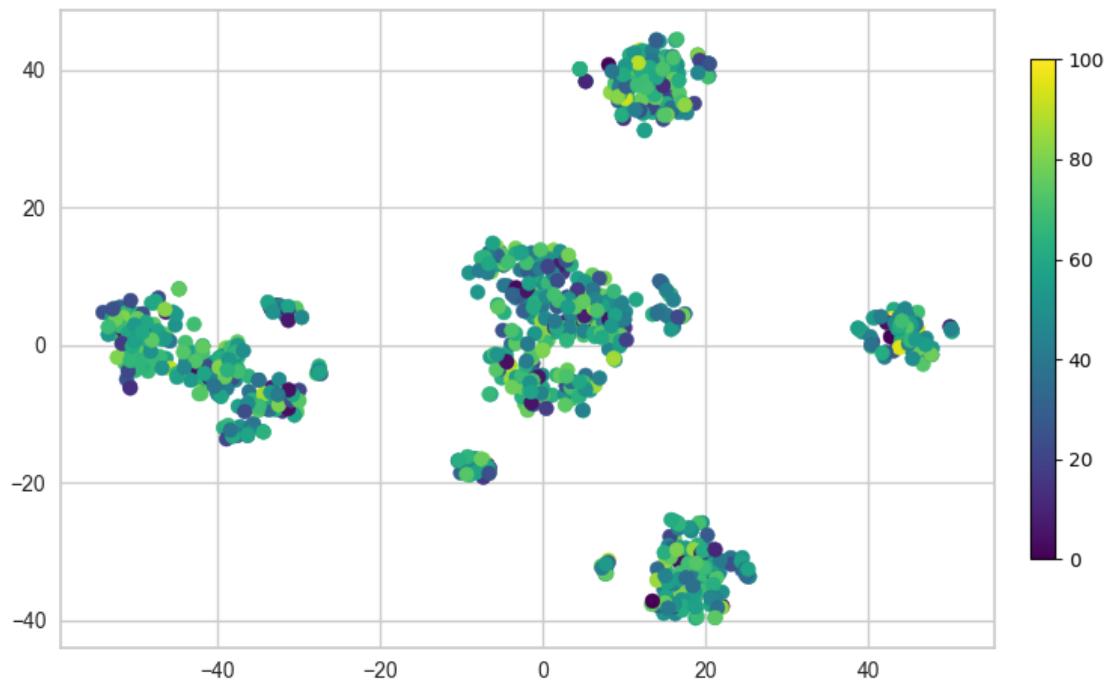
Cluster 4 - TSNE



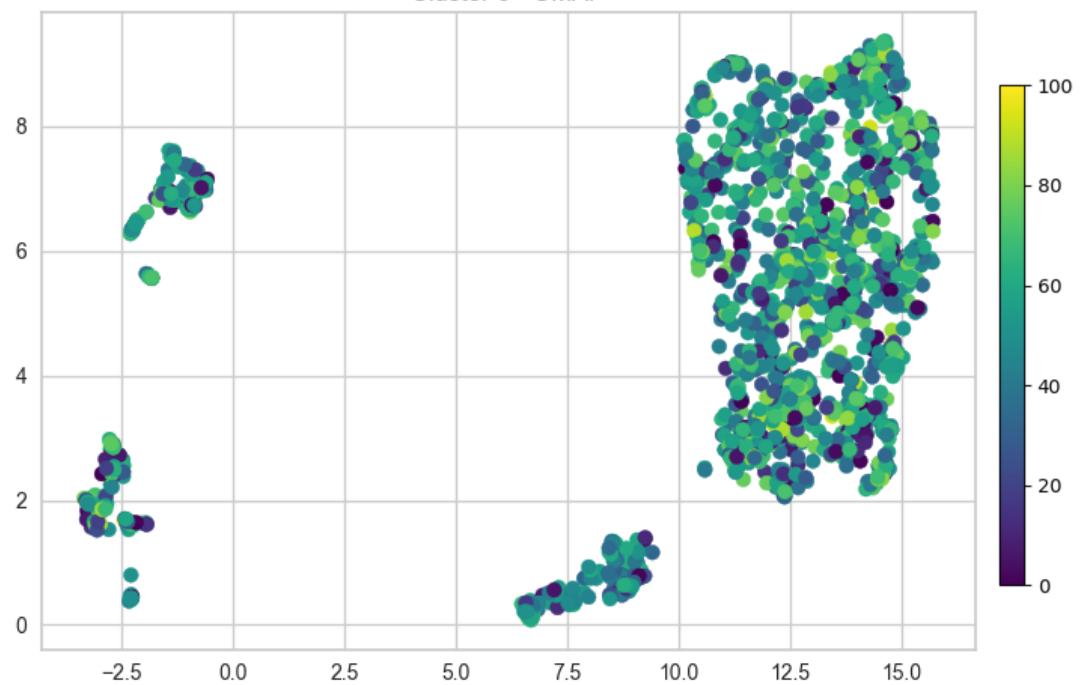
Cluster 5 - TSNE



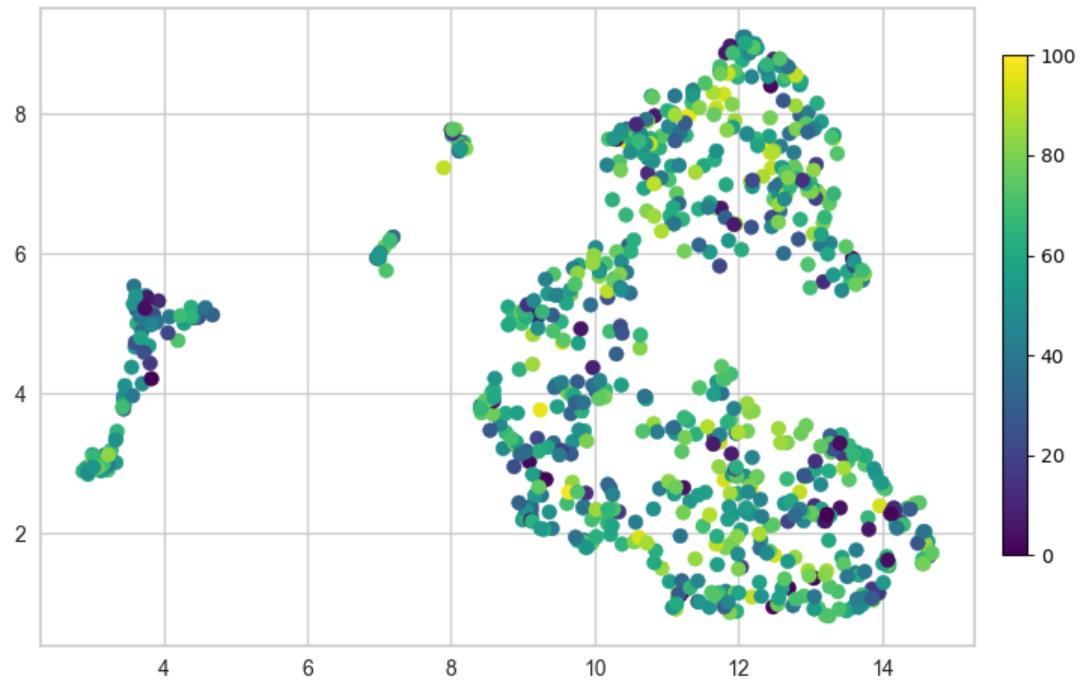
Cluster 6 - TSNE



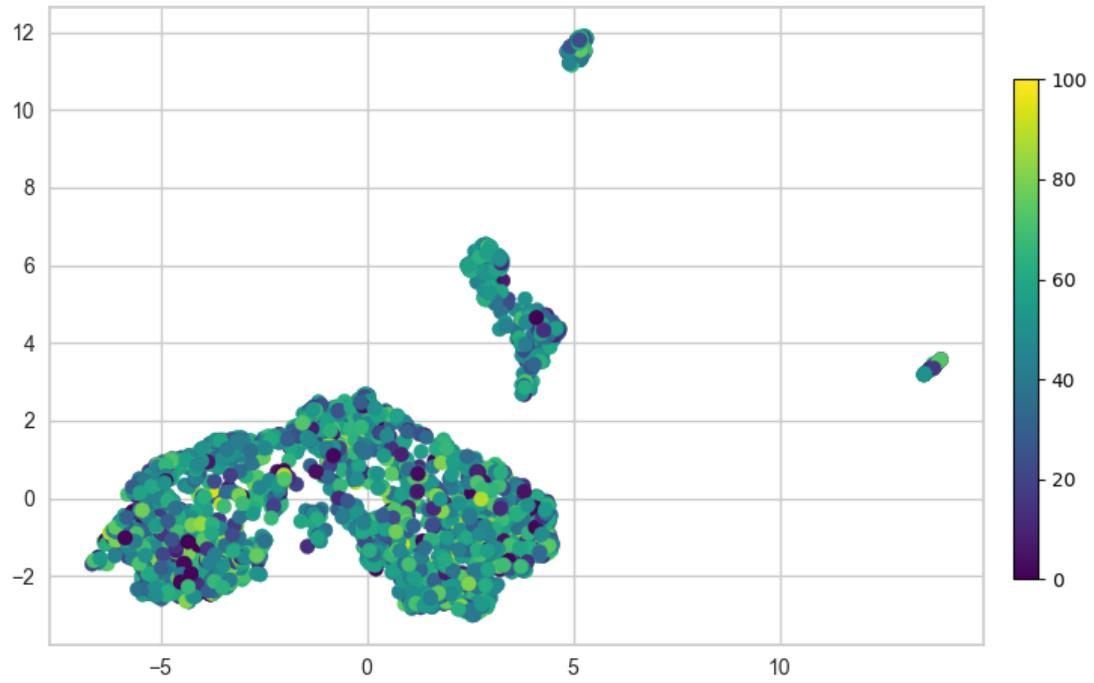
Cluster 0 - UMAP



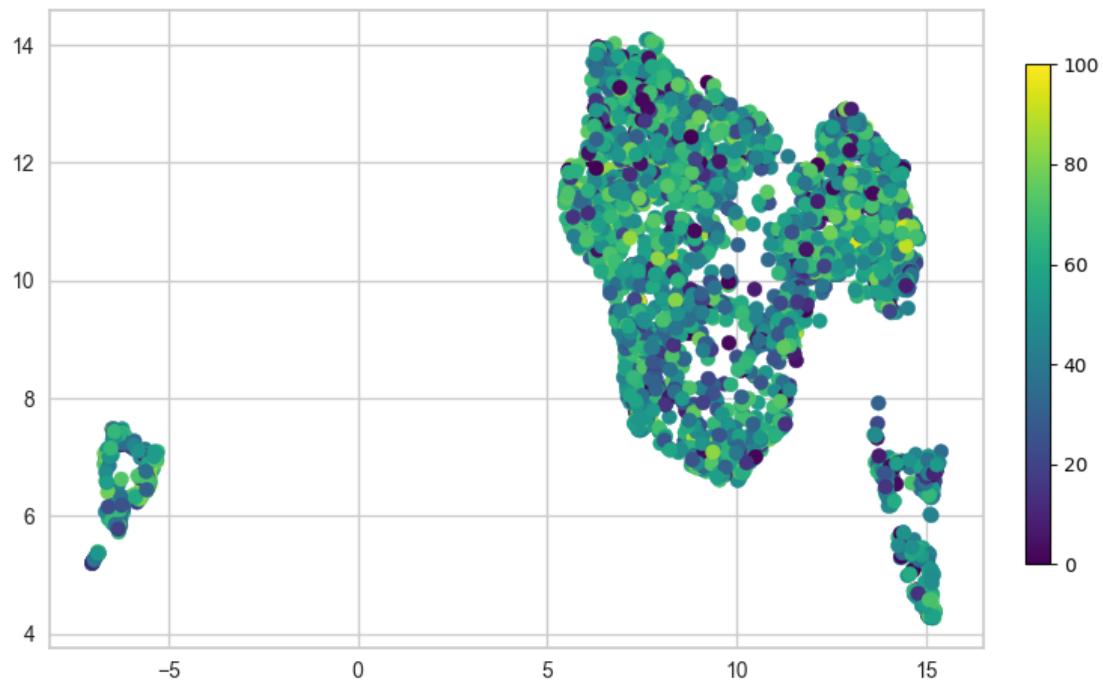
Cluster 1 - UMAP



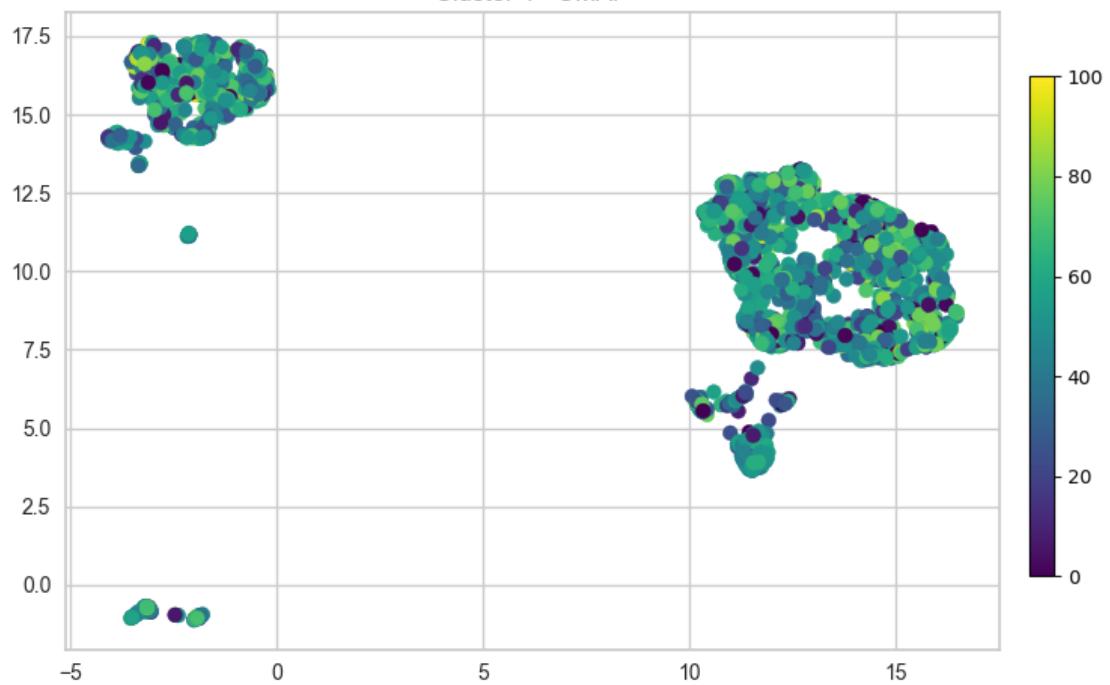
Cluster 2 - UMAP



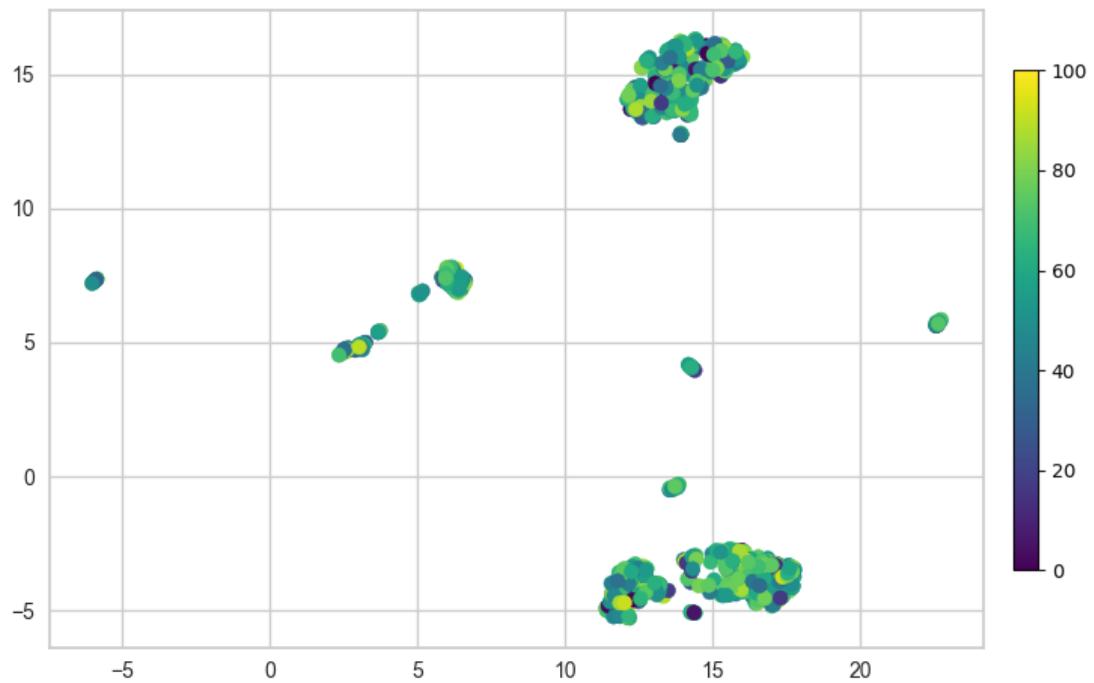
Cluster 3 - UMAP



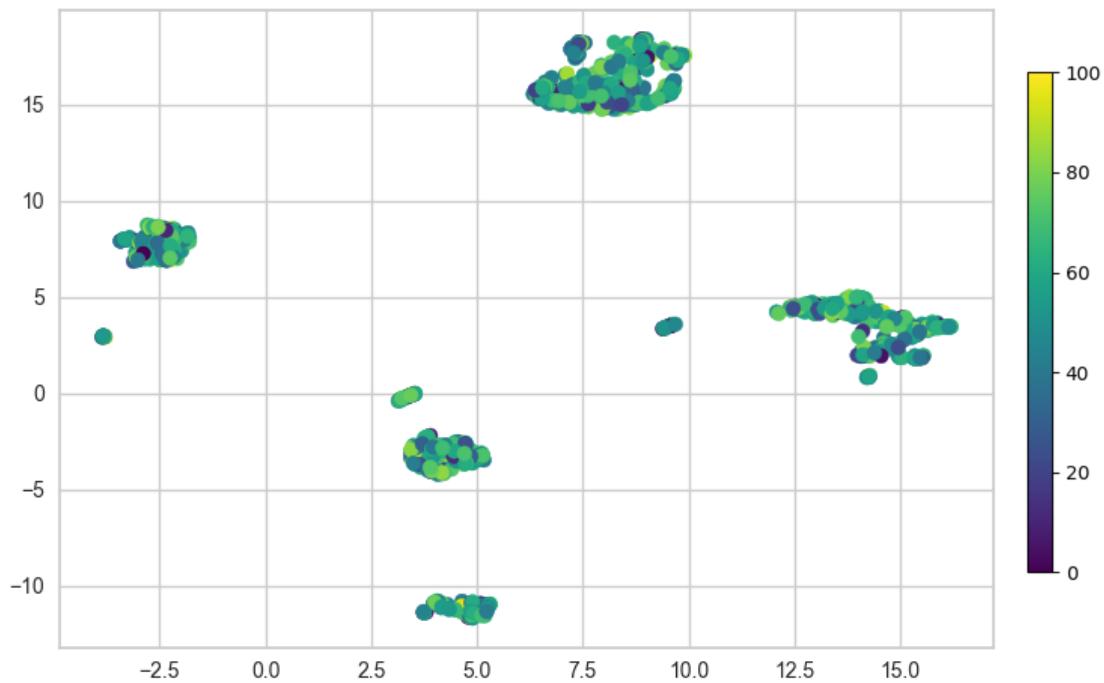
Cluster 4 - UMAP



Cluster 5 - UMAP



Cluster 6 - UMAP



Conclusiones

Tras hacer este análisis por clúster se puede comprobar que se sigue sin ver una agrupación clara de canciones populares.

Conclusión final

Nuestra conclusión final, tanto por la parte de predicción, como por la de visualización, es que, si bien se puede mejorar ligeramente la estimación a priori de la popularidad de una canción analizando sus características, no es posible saber con certeza si dicha canción va a ser realmente popular o no con los datos utilizados. Existen canciones muy populares de distintos tipos, y canciones muy similares a otras que son populares que no lo son tanto.