Universidad del Valle de Guatemala
Facultad de Ingeniería
Departamento de Ciencias de la Computación

Proyecto 2

Daniela Villamar 19086
Diego Crespo 19541
Rene Ventura
Andres Paiz 191142

Introducción

Un estudio hecho por Digital Music Alliance, en 2018, concluyó que el 54% de los usuarios de plataformas de streaming de música están reemplazando los álbumes por las playlists en sus hábitos. Pero los usuarios no solo escuchan las playlists, sino que también las crean. Los usuarios de Spotify han creado y compartido más de 4 billones de playlists. Los usuarios crean una playlist por diversas razones: algunas playlists agrupan la música de manera categórica. Ya sea por género, artista, año o incluso país, por estado de ánimo, tema u ocasión. Incluso algunas playlists están hechas para dar un mensaje específico, de amor, celebración, bienvenida, etc... Al comprender esta relación tan profunda entre la música y las personas, se puede hacer una relación congruente entre canciones y poder generar las playlist indicadas para los usuarios. Al aprender más sobre la naturaleza de las listas de reproducción, también podemos sugerir otras pistas que un oyente disfrutará en el contexto de una playlist. Esto puede facilitar la creación de playlist y, en última instancia, ayudar a las personas a encontrar más música que aman.

El problema proviene de la pregunta ¿Cómo podemos facilitar la creación de playlist a los usuario, mientras se mantiene la congruencia entre las canciones que contiene esta misma? La solución a este problema es generar un modelo el cual pueda entrenarse a base de diversas playlist las cuales tienen una congruencia entre las canciones o algunas que no la tengan. El modelo implementará algoritmos de nearest neighbor y correlación de Pearson. Nearest neighbor determina según los datos proporcionados un patrón de gustos y preferencias y utiliza los datos de un vecino cercano con características similares al inicial y partiendo de estos datos genera las recomendaciones. Por otra parte, el algoritmo de correlación de Pearson es un algoritmo de similaridad que recolecta los datos de preferencias de los usuarios y determina un peso de similitud para estimar la relación que existe entre dos usuarios y crear recomendaciones de contenido en base a dicha similaridad.

Objetivos

Se plantean los objetivos a cumplir para darle solución al problema planteado. Se enuncia al menos un objetivo general y 2 específicos. Los objetivos deben ser medibles y alcanzables durante la investigación Objetivo General Generar un sistema de recomendación de canciones para una playlist en base a las canciones que ya contiene la playlist y así facilitar dicha creación para el usuario.

Objetivos específicos

● Generar un sistema de recomendación utilizando el mejor de los algoritmos nearest neighbor y correlación de pearson.
● Por medio de métricas tales como R-precision y NDCG se podrá determinar la efectividad de cada algoritmo y poder llegar a una conclusión certera.
● Medir la congruencia entre canciones por medio de los atributos que ofrecen en la data (eg. artista, año, género).
● Crear una playlist de manera aleatoria ingresando solo el género que al usuario le gustaría escuchar.

Marco Teórico

Un sistema de recomendación es un algoritmo de inteligencia artificial o AI, generalmente asociado con el aprendizaje automático, que utiliza Big Data para sugerir o recomendar productos adicionales a los consumidores. Estos pueden basarse en varios criterios, incluidas compras anteriores, historial de búsqueda, información demográfica y otros factores. Los sistemas de recomendación son muy útiles ya que ayudan a los usuarios a descubrir productos y servicios que de otro modo no habrían encontrado por sí mismos.

Los sistemas de recomendación están capacitados para comprender las preferencias, las decisiones previas y las características de las personas y los productos utilizando los datos recopilados sobre sus interacciones. Estos incluyen impresiones, clics, me gusta y compras. Debido a su capacidad para predecir los intereses y deseos de los consumidores a un nivel altamente personalizado, los sistemas de recomendación son los favoritos entre los proveedores de contenido y productos. Pueden llevar a los consumidores a casi cualquier producto o servicio que les interese, desde libros hasta videos, clases de salud y ropa.

Si bien existe una gran cantidad de algoritmos y técnicas de recomendación, la mayoría se clasifican en estas amplias categorías: filtrado colaborativo, filtrado de contenido y filtrado de contexto.

Los algoritmos de filtrado colaborativo recomiendan elementos (esta es la parte de filtrado) en función de la información de preferencia de muchos usuarios (esta es la parte colaborativa). Este enfoque utiliza la similitud del comportamiento de preferencia del usuario, dadas las interacciones anteriores entre los usuarios y los elementos, los algoritmos de recomendación aprenden a predecir la interacción futura. Estos sistemas de recomendación construyen un modelo a partir del comportamiento anterior de un usuario, como artículos comprados anteriormente o calificaciones dadas a esos artículos y decisiones similares de otros usuarios. La idea es que si algunas personas han tomado decisiones y compras similares en el pasado, como una elección de película, entonces existe una alta probabilidad de que estén de acuerdo en futuras selecciones adicionales.

El filtrado de contenido, por el contrario, utiliza los atributos o características de un elemento (esta es la parte del contenido) para

recomendar otros elementos similares a las preferencias del usuario. Este enfoque se basa en la similitud de las características de los elementos y los usuarios, dada la información sobre un usuario y los elementos con los que ha interactuado (por ejemplo, la edad de un usuario, la categoría de la cocina de un restaurante, la reseña promedio de una película), modela la probabilidad de un nuevo Interacción.

El filtrado de contexto incluye la información contextual de los usuarios en el proceso de recomendación. Netflix habló en NVIDIA GTC sobre cómo hacer mejores recomendaciones al enmarcar una recomendación como una predicción de secuencia contextual. Este enfoque utiliza una secuencia de acciones contextuales del usuario, además del contexto actual, para predecir la probabilidad de la siguiente acción.

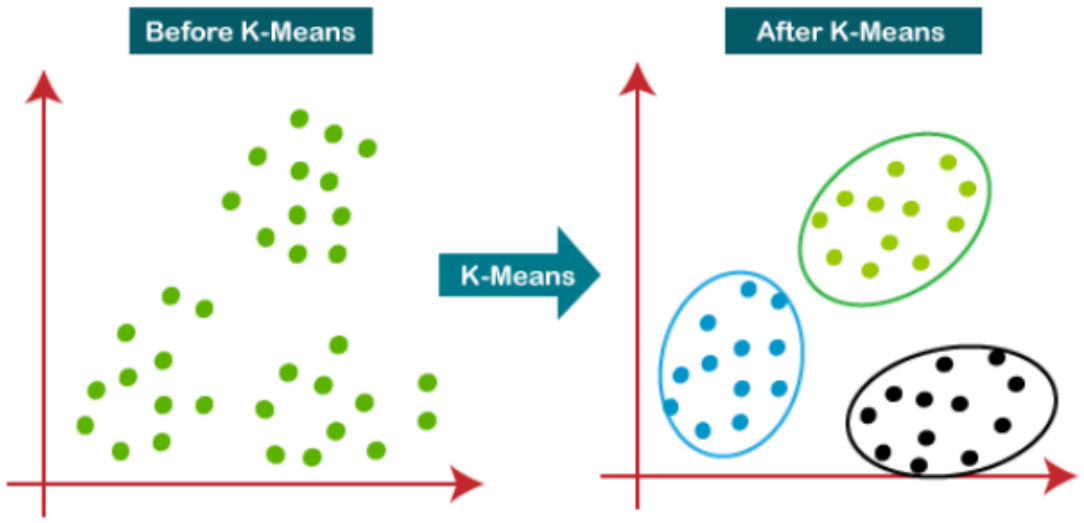
El preprocesamiento de datos es un paso en el proceso de extracción y análisis de datos que toma datos sin procesar y los transforma en un formato que puede ser entendido y analizado por computadoras y aprendizaje automático. Los datos sin procesar del mundo real en forma de texto, imágenes, video, etc., son desordenados. En este proyecto se llevo a cabo un analisis exploratorio de datos y luego de esto las operaciones de limpieza correspondientes para tener un dataset mas claro y facil con que trabajar.

Algoritmos a utilizar

1. **K-Means Clustering**

El agrupamiento de K-means es un tipo de aprendizaje no supervisado, que se usa cuando tiene datos sin etiquetar (es decir, datos sin categorías o grupos definidos). El objetivo de este algoritmo es encontrar grupos en los datos, con la cantidad de grupos representados por la variable K. El algoritmo funciona de forma iterativa para asignar cada punto de datos a uno de los grupos K en función de las características proporcionadas. Los puntos de datos se agrupan en función de la similitud de características. Los resultados del algoritmo de agrupamiento de K-medias son: Los centroides de los grupos K, que se pueden usar para etiquetar nuevos datos Etiquetas para los datos de entrenamiento (cada punto de datos se asigna a un solo grupo) En lugar de definir grupos antes de ver los datos, la agrupación le permite encontrar y analizar los grupos que se han formado orgánicamente. La sección "Elegir K" a continuación describe cómo se puede determinar el número de grupos. Cada centroide de un clúster es una colección de valores de características que definen los grupos

resultantes. El examen de los pesos de las características del centroide se puede utilizar para interpretar cualitativamente qué tipo de grupo representa cada grupo.

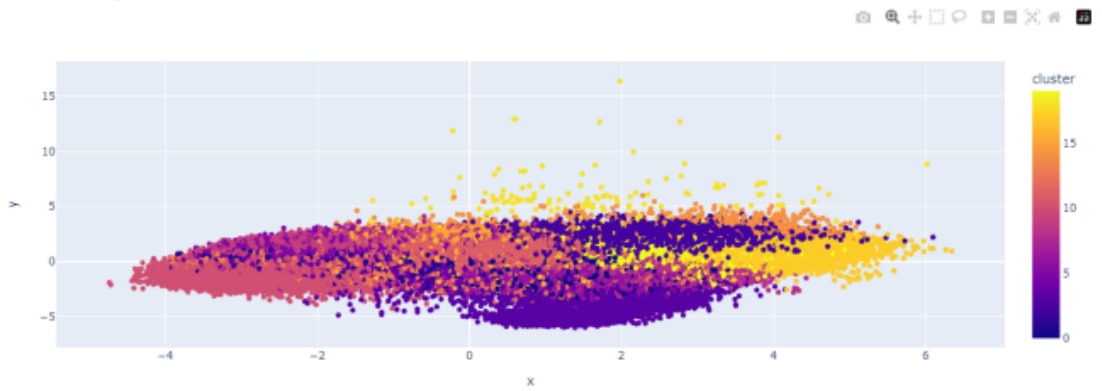


El algoritmo consta de tres pasos:

● Inicialización: una vez escogido el número de grupos, k, se establecen k centroides en el espacio de los datos.
● Asignación objetos a los centroides: cada objeto de los datos es asignado a su centroide más cercano.

Actualización centroides: se actualiza la posición del centroide de cada grupo tomando como nuevo centroide la posición del promedio de los objetos pertenecientes a dicho grupo. El algoritmo de agrupamiento de K-means se usa para encontrar grupos que no se han etiquetado explícitamente en los datos. Esto se puede usar para confirmar suposiciones comerciales sobre qué tipos de grupos existen o para identificar grupos desconocidos en conjuntos de datos complejos. Una vez que se ha ejecutado el algoritmo y se han definido los grupos, cualquier dato nuevo se puede asignar fácilmente al grupo correcto. Se consideró la utilización de K means ya que es un dataset muy grande y es una buena herramienta para iniciar nuestro proceso de recomendación. Igualmente es muy fácil de utilizar e implementar. Igualmente nos puede ayudar a identificar posibles datos o conjuntos de datos que pueden no estar identificados por los labels previamente establecidos en el dataset.

Clustering de Canciones:



Clustering de Géneros:

2. **Min Max Normalization**

Min-Max Normalization (conocida también por escalado de características) es un método que realiza una transformación lineal en los datos originales. Esta técnica obtiene todos los datos escalados en el rango (0,1). La fórmula que se utiliza para conseguir lo mencionado anteriormente es lo siguiente:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

La normalización min-max conserva las relaciones entre los valores de datos originales. El costo de tener este rango acotado es que terminaremos con desviaciones estándar más pequeñas, lo que puede suprimir el efecto de los valores atípicos.

Se normaliza lla data por medio del algoritmo min max con las columnas a utilizar, para hacer un fit

```
feature_cols=['acousticness', 'danceability', 'duration_ms', 'energy',
              'instrumentalness', 'key', 'liveness', 'loudness', 'mode',
              'speechiness', 'tempo',  'valence',]

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
normalized_df =scaler.fit_transform(df[feature_cols])

print(normalized_df[:2])
```

✓ 0.3s

```
[[0.01024843 0.82482599 0.19073524 0.4263629  0.02243852 0.18181818
  0.15386234 0.74114059 1.         0.51444066 0.59603317 0.26243209]
 [0.19999772 0.72041763 0.3144808  0.35008137 0.00626025 0.09090909
  0.12439486 0.69216224 1.         0.07100517 0.6544742  0.57793565]]
```

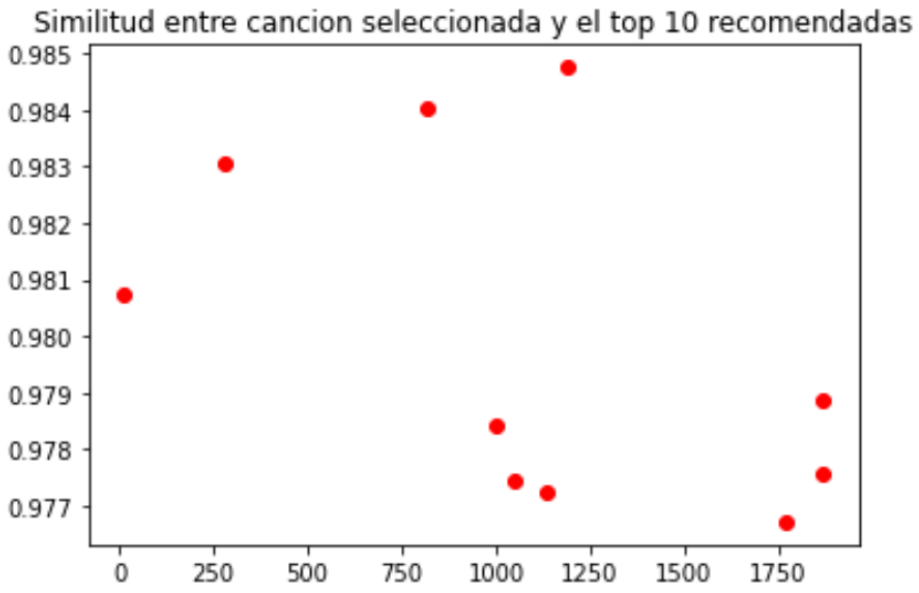
Una cosa importante a tener en cuenta cuando se usa MinMax Scaling es que está muy influenciado por los valores máximos y mínimos en nuestros datos, por lo que si nuestros datos contienen valores atípicos, estarán sesgados. También la escala de Min Max podría comprimir todos los valores internos en un rango estrecho.

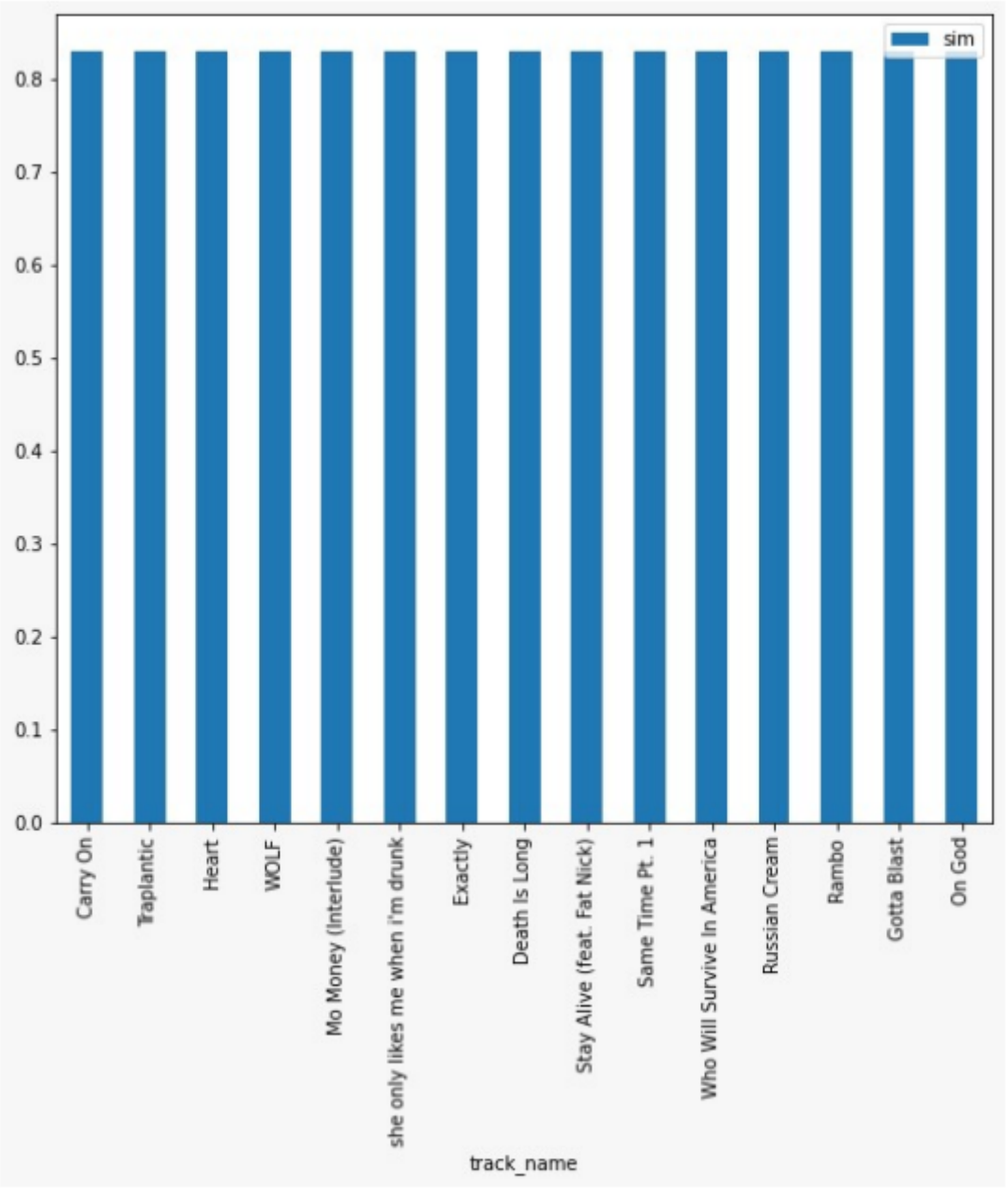
3. Cosine similarity

En términos de análisis de datos la similitud del coseno es una medida de similitud entre dos secuencias de números. Para definirlo, las sucesiones se ven como vectores en un espacio de producto interior, y la similitud de coseno se define como el coseno del ángulo entre ellos, es decir, el producto escalar de los vectores dividido por el producto de sus longitudes. De ello se deduce que la similitud del coseno no depende de las magnitudes de los vectores, sino sólo de su ángulo. La similitud del coseno siempre pertenece al intervalo $[-1, 1]$. Por ejemplo, en la recuperación de información y la minería de texto, a cada palabra se le asigna una coordenada diferente y un documento se representa mediante el vector del número de ocurrencias de cada palabra en el documento. La similitud del coseno luego brinda una medida útil de qué tan similares pueden ser dos documentos, en términos de su tema e independientemente de la longitud de los documentos. La técnica también se utiliza para medir la cohesión dentro de los clústeres en el campo de la minería de datos. Una ventaja de la similitud del coseno es su baja complejidad, especialmente para vectores dispersos, sólo se deben considerar las coordenadas distintas de cero.

$$\text{cosine similarity} = S_C(A, B) := \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

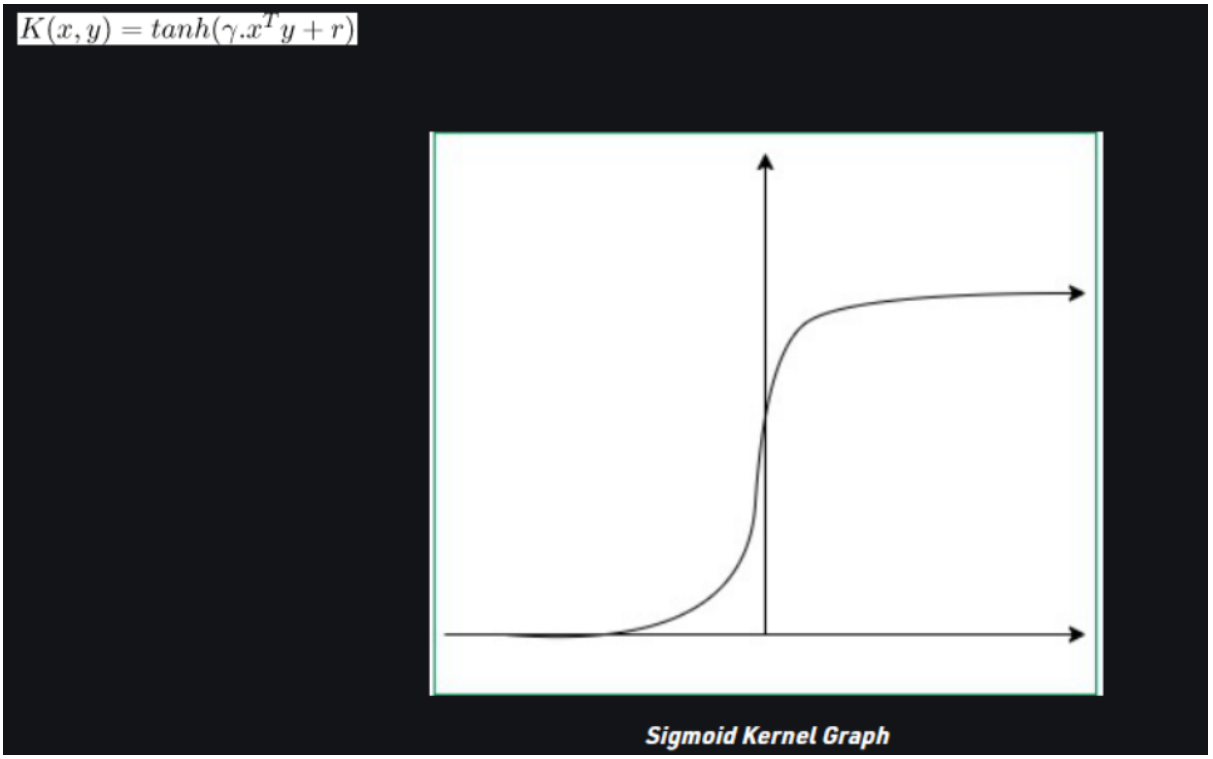
```
Similarity entre cancion escogida y recomendadas:  [(1187, 0.9847539845102656), (818, 0.9840309506844511), (281, 0.9830586579546059), (12, 0.9807487637970956), (1867, 0.9788710374641363), (1000, 0.9784134753225019), (1869,
0.9775609145362028), (1047, 0.9774532204955416), (1136, 0.9772488658365086), (1769, 0.9767138824482666)]
```



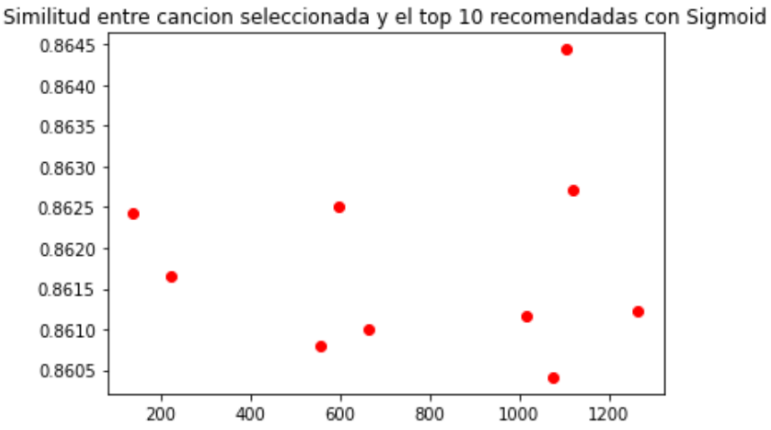


4. Sigmoid SVM Kernel Functions

El kernel sigmoide se aplica ampliamente en redes neuronales para tareas de clasificación. El clasificador SVM, que se aplica con el kernel sigmoide, tiene una excelente precisión de clasificación. Sin embargo, como el núcleo sigmoide tiene una estructura complicada, generalmente es difícil para los expertos humanos interpretar y comprender cómo el núcleo sigmoide toma su decisión de clasificación.



Esta función es equivalente a un modelo de perceptrón de dos capas de la red neuronal, que se utiliza como función de activación para neuronas artificiales.




```
sig_kernel = sigmoid_kernel(normalized_df)
a=generate_recommendation('Redbone',sig_kernel).values
```

```
Similarity entre cancion escogida y recomendadas:  [(1106, 0.8644406383322815), (1120, 0.8627221030363934), (598, 0.8625160346873847), (138, 0.8624361909857228), (222, 0.8616521369907595), (1266, 0.8612202479818735), (1015, 0.8611638718189624), (665, 0.8609989626256612), (557, 0.8608045723131829), (1075, 0.8604115445719285)]
```

Metodología

Descripción de los datos: Se describen los datos, tanto las variables y observaciones como las operaciones de limpieza que se le hicieron si fueron necesarias.

Operaciones de limpieza necesarias: La data entregada viene en formato JSON, este formato no es compatible con las librerías de análisis de datos en Python. Fue necesario realizar un programa que realice la conversión de todos los JSON's a CSV para poder utilizar las herramientas necesarias para analizar los datos.

Variables:

Info:

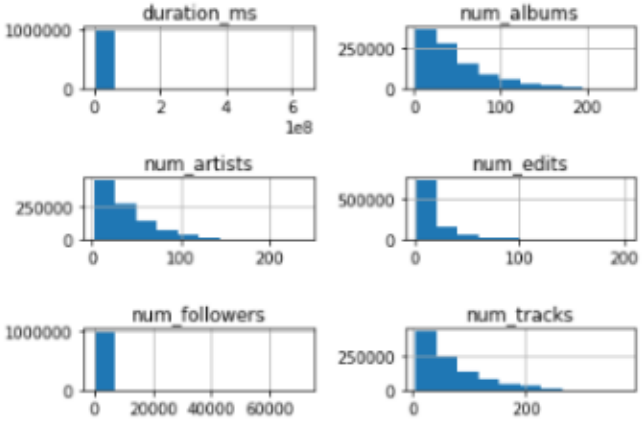
- slice: el rango de cortes que hay en este archivo en particular, como 0-999
- version: la versión actual del MPD (que debería ser v1)
- description: una descripción del MPD
- license: información de licencia del MPD
- generated_on: fecha de generación del corte. Playlist:
- pid: playlist id
- name: name of playlist
- description: string opcional, descripcion de playlist.
- modified_at: segundos - marca de tiempo (en segundos desde la época) cuando esta lista de reproducción se actualizó por última vez. Los tiempos se redondean a la medianoche GMT de la fecha en que se actualizó la lista de reproducción por última vez.
- num_artists:el número total de artistas únicos para las pistas en la lista de reproducción.
- num_albums: el número de álbumes únicos para las pistas en la lista de reproducción
- num_tracks: el número de pistas en la lista de reproducción
- num_followers:el número de seguidores que tenía esta lista de reproducción en el momento en que se creó el MPD.
- num_edits:el número de sesiones de edición separadas. Las pistas añadidas en una ventana de dos horas se consideran añadidas en una única sesión de edición.
- duration_ms: duración total de todas las canciones en milisegundos.
- collaborative: verdadero si es playlist colaborativa, falso si no lo es.

- tracks: una matriz de información sobre cada pista en la lista de reproducción. Cada elemento de la matriz es un diccionario con los siguientes campos:
    - track_name: nombre
    - track_uri: URI de spotify
    - album_name: nombre del álbum
    - album_uri: URI de spotify del álbum
    - artist_name: nombre del artista primario
    - artist_uri: URi de spotify del artista primario
    - duration_ms: duración en milisegundos
    - pos: posición de la canción en la playlist.

Número de:

➔ Listas de reproducción: 1000000
➔ Pistas: 66346428
➔ Pistas únicas: 2262292
➔ Álbumes únicos: 734684
➔ Títulos únicos: 92944

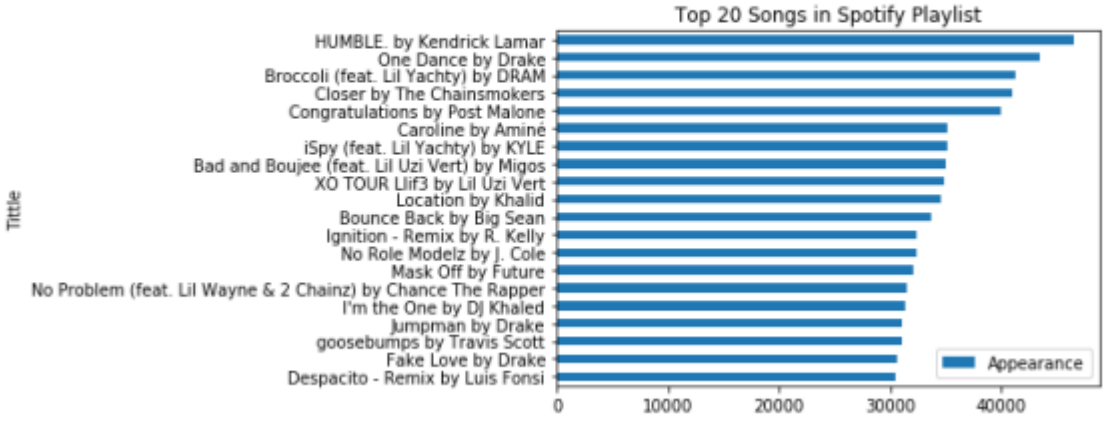
Distribución de: duración de la lista de reproducción, número de álbumes/lista de reproducción, número de artista/lista de reproducción, número de ediciones/lista de reproducción, número de seguidores/lista de reproducción, número de pistas/lista de reproducción.



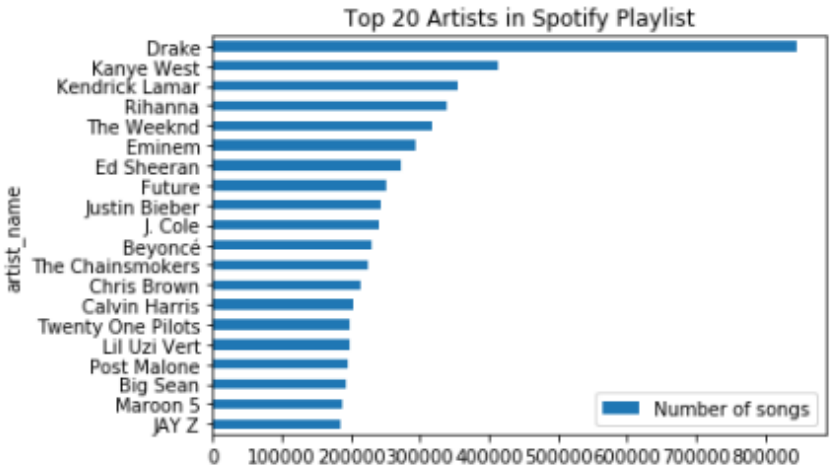
Como podemos ver, todas las distribuciones están sesgadas a la izquierda, lo que significa que si estamos buscando un valor promedio, debemos elegir "Mediana", no "Media".

- ➢ Mediana de duración de la lista de reproducción: 11422438.0
- ➢ Mediana del número de álbumes en cada lista de reproducción: 37.0
- ➢ Mediana del número de artistas en cada lista de reproducción: 29,0
- ➢ Mediana del número de ediciones en cada lista de reproducción: 29,0
- ➢ Mediana del número de seguidores en cada lista de reproducción: 1.0
- ➢ Mediana del número de pistas en cada lista de reproducción: 49,0

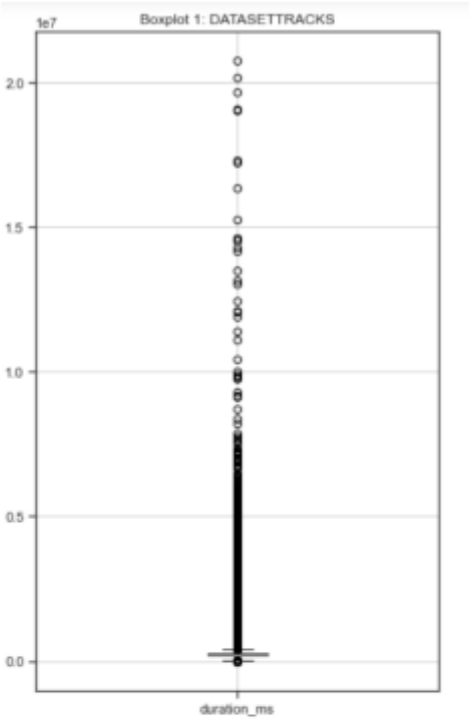
Las 20 mejores canciones en las listas de reproducción de Sporify

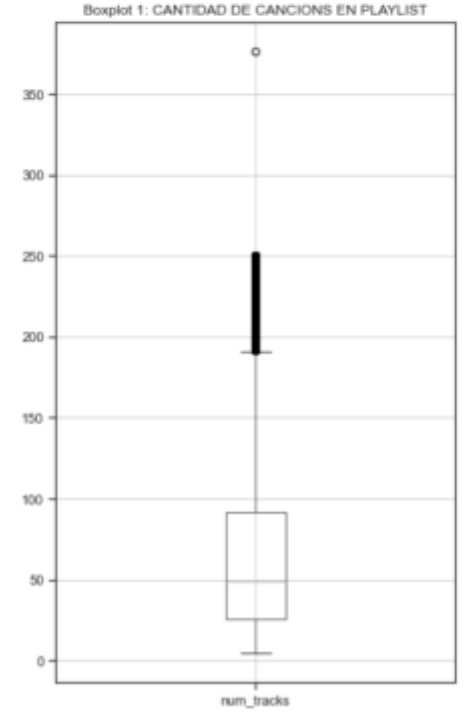


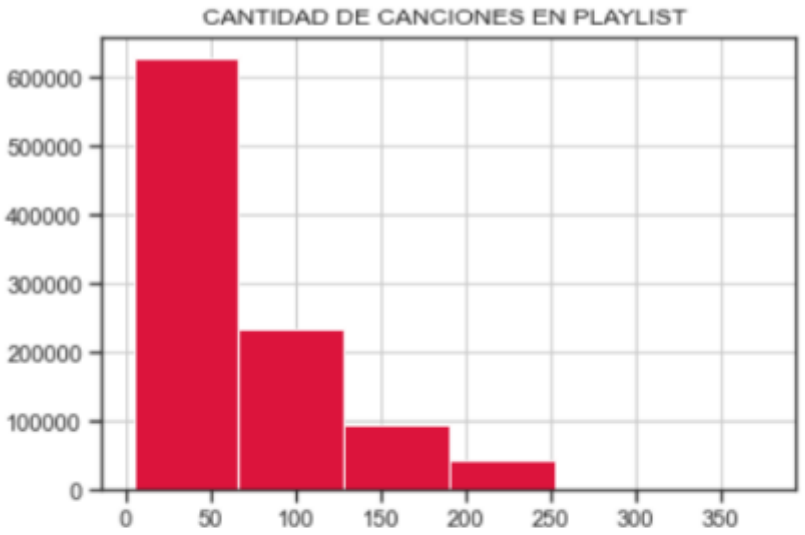
Los 20 mejores artistas en las listas de reproducción de Spotify



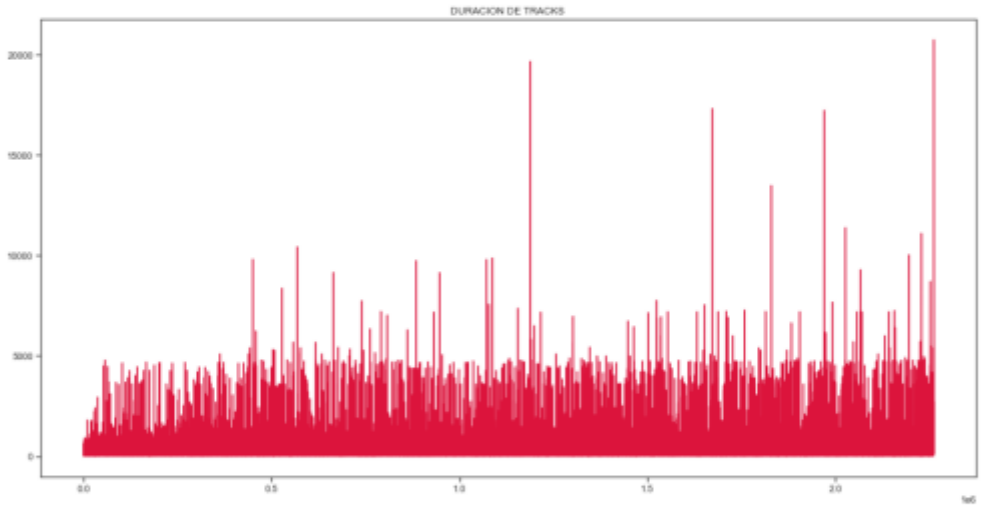
- Estudia las variables cuantitativas mediante técnicas de estadística descriptiva
- Hace gráficos exploratorios como histogramas, diagramas de cajas y bigotes, gráficos de dispersión que ayudan a explicar los datos


Boxplot 1: DATASETTRACKS
1e7
2.0
1.5
1.0
0.5
0.0
duration_ms


Boxplot 1: CANTIDAD DE CANCIONS EN PLAYLIST
350
300
250
200
150
100
50
0
num_tracks


CANTIDAD DE CANCIONES EN PLAYLIST
600000
500000
400000
300000
200000
100000
0
0
50
100
150
200
250
300
350


CANTIDAD DE ALMBUMS EN PLAYLIST
300000
250000
200000
150000
100000
50000
0
0
50
100
150
200
250



Media duracion de track en segundos

```
statistics.median(DATASETTRACKSDIVS)
```

```
2.2528000000000006
```

Inicialmente se puede observar que la distribución no es simétrica para ninguna de las dos boxplots. Podemos observar que la mayoría de playlist se encuentran en el rango de 0 a 50 canciones, lo mismo se puede decir de la cantidad de álbumes en la playlist. Esto nos puede dar una pista de cómo están distribuidas

las playlists de los usuarios en términos de álbumes y canciones. En el caso de los edits la mayoría se encuentra en el rango aproximado de 0 a 20. Para la duración de las canciones podemos observar que la mayoría duran aproximadamente entre 2.25 minutos y 1.2 minutos, aunque se debe de ajustar la gráfica para mostrar valores mayores a 2.25 minutos. Podemos observar que la media calculada muestra el valor mencionado previamente (2.25 minutos). Lo cual puede ayudar a la hora de la recomendación, tal vez evitar canciones demasiado cortas o largas es una buena idea. Igualmente al observar la cantidad de álbumes por playlist, tal vez se podría evitar recomendar canciones que sean del mismo álbum.

Resultados y Análisis de Resultados

Para poder hacer la aplicación con una interfaz amigable para el usuario se utilizaron los siguientes para tener 3 programas:

Diagrama del codo

El visualizador K-Elbow implementa el método de "codo" para seleccionar el número óptimo de grupos para el agrupamiento de K-means. K-means es un algoritmo simple de aprendizaje automático no supervisado que agrupa datos en un número específico (k) de grupos. Debido a que el usuario debe especificar de antemano qué k elegir, el algoritmo es sencillo: asigna todos los miembros a k clústeres incluso si ese no es el k correcto para el conjunto de datos.

El método del codo ejecuta la agrupación de k-medias en el conjunto de datos para un rango de valores de k (por ejemplo, de 1 a 10) y luego, para cada valor de k, calcula una puntuación promedio para todas las agrupaciones. De forma predeterminada, se calcula la puntuación de distorsión, la suma de las distancias al cuadrado desde cada punto hasta su centro asignado. También se pueden usar otras métricas, como la puntuación de silueta, el coeficiente de silueta medio para todas las muestras o la puntuación *calinski_harabasz*, que calcula la relación de dispersión entre y dentro de los grupos.

Cuando se trazan estas métricas generales para cada modelo, es posible determinar visualmente el mejor valor para k. Si el gráfico de líneas parece un brazo, entonces el "codo" (el punto de inflexión en la curva) es el mejor valor de k. El "brazo" puede estar hacia arriba o hacia abajo, pero si hay un fuerte punto de inflexión, es una buena indicación de que el modelo subyacente se ajusta mejor a ese punto.

Pyclustertend

pyclustertend es un paquete de python especializado en tendencia de clúster. La tendencia de agrupamiento consiste en evaluar si los algoritmos de agrupamiento son relevantes para un conjunto de datos.

Hopkins Test

La estadística de Hopkins se utiliza para evaluar la tendencia de agrupación de un conjunto de datos midiendo la probabilidad de que un conjunto de datos determinado se genere mediante una distribución uniforme de datos. En otras palabras, prueba la aleatoriedad espacial de los datos.

★Recomender

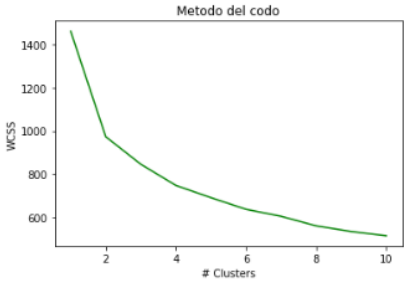
```
import os
import numpy as np
import pandas as pd

import seaborn as sns
import plotly.express as px
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
from sklearn.metrics import euclidean_distances
from scipy.spatial.distance import cdist



import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
import seaborn as sb
import sklearn.cluster as cluster
import sklearn.metrics as metrics
import sklearn.preprocessing
import scipy.cluster.hierarchy as sch
import pylab
import sklearn.mixture as mixture
import pyclustertend
import random
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.mixture import GaussianMixture


from sklearn.datasets import make_blobs
from sklearn.metrics import silhouette_samples, silhouette_score

import matplotlib.cm as cm

import warnings
warnings.filterwarnings("ignore")

df=pd.read_csv("data2.csv")
```

In [3]:
```
#INICIA EL CLUSTERING

Lets = df[[ 'acousticness', 'danceability', 'duration_ms', 'energy',
            'instrumentalness', 'key', 'liveness', 'loudness', 'mode',
            'speechiness', 'tempo',  'valence',]] #Declaramos nuestra lista de columnas a trabajar

CantDatos = np.array(Lets.sample(frac=0.1, random_state=123).reset_index(drop=True)) #Declaramos CantDatos
CantDatos_scaled = sklearn.preprocessing.scale(CantDatos) #Realizamos las modificaicones a CantDatos
pyclustertend.hopkins(CantDatos, len(CantDatos)) #Aplicamos HOPKINS para ver  la tendencia para los clusters
```

Out[3]: 0.16628231798039692

El valor de hopkins nos da un 0.16628231798039692 alejado de 0.5 validando nuestro clustering.

In [4]:
```
Lets_norm = (Lets - Lets.min()) / (Lets.max() - Lets.min())  #Realizamos una normalizacaion entre las variables a trabajar.
wcss = []

for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, max_iter=300)         #Utilizamos el metodo del codo y el algoritmo KMEANS
    kmeans.fit(Lets_norm)                                 #para determinar la mejor cantidad de clusters dependiendo los datos normalizados.
    wcss.append(kmeans.inertia_)

plt.plot(range(1, 11), wcss,color='green')
plt.title('Metodo del codo')
plt.xlabel('# Clusters')
plt.ylabel('WCSS')
plt.show()
```



In [2]:
```
df.head()
```

Out[2]:

	Unnamed: 0	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mode	speechiness	tempo	time_signature	valence	target	song_title
0	0	0.0102	0.833	204600	0.434	0.021900	2	0.1650	-8.795	1	0.4310	150.062	4.0	0.286	1	Mask Off
1	1	0.1990	0.743	326933	0.359	0.006110	1	0.1370	-10.401	1	0.0794	160.083	4.0	0.588	1	Redbone
2	2	0.0344	0.838	185707	0.412	0.000234	2	0.1590	-7.148	1	0.2890	75.044	4.0	0.173	1	Xanny Family
3	3	0.6040	0.494	199413	0.338	0.510000	5	0.0922	-15.236	1	0.0261	86.468	4.0	0.230	1	Master Of None
4	4	0.1800	0.678	392893	0.561	0.512000	5	0.4390	-11.648	0	0.0694	174.004	4.0	0.904	1	Parallel Lines

In [29]:
```
df.info()
```

```
RangeIndex: 2017 entries, 0 to 2016
Data columns (total 17 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Unnamed: 0        2017 non-null   int64
 1   acousticness      2017 non-null   float64
 2   danceability      2017 non-null   float64
 3   duration_ms       2017 non-null   int64
 4   energy            2017 non-null   float64
 5   instrumentalness  2017 non-null   float64
 6   key               2017 non-null   int64
 7   liveness          2017 non-null   float64
 8   loudness          2017 non-null   float64
 9   mode              2017 non-null   int64
 10  speechiness       2017 non-null   float64
 11  tempo             2017 non-null   float64
 12  time_signature    2017 non-null   float64
 13  valence           2017 non-null   float64
 14  target            2017 non-null   int64
 15  song_title        2017 non-null   object
 16  artist            2017 non-null   object
dtypes: float64(10), int64(5), object(2)
memory usage: 268.0+ KB
```

Se normaliza lla data por medio del algoritmo min max con las columnas a utilizar, para hacer un fit

In [30]:
```
feature_cols=['acousticness', 'danceability', 'duration_ms', 'energy',
              'instrumentalness', 'key', 'liveness', 'loudness', 'mode',
              'speechiness', 'tempo',  'valence',]

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
normalized_df =scaler.fit_transform(df[feature_cols])

print(normalized_df[:2])
```

```
[[0.01024843 0.82482599 0.19073524 0.4263629  0.02243852 0.18181818
  0.15386234 0.74114059 1.         0.51444066 0.59603317 0.26243209]
 [0.19999772 0.72041763 0.3144808  0.35008137 0.00626025 0.09090909
  0.12439486 0.69216224 1.         0.07100517 0.6544742  0.57793565]]
```

Se crea la funcion de similarity para generar recomendaciones, utilizando SVM y cosine similarity

In [65]:
```
indices = pd.Series(df.index, index=df['song_title']).drop_duplicates()

cosine = cosine_similarity(normalized_df)

def generate_recommendation(song_title, model_type=cosine ):


    index=indices[song_title]

    score=list(enumerate(model_type[indices[index]]))

    similarity_score = sorted(score,key = lambda x:x[1],reverse = True)

    similarity_score = similarity_score[1:11]
    top_songs_index = [i[0] for i in similarity_score]

    top_songs=df['song_title'].iloc[top_songs_index]
    print("Similarity entre cancion escogida y recomendadas: ",similarity_score)
    return top_songs
```

In [66]:
```
a=generate_recommendation('Redbone',cosine).values
```

```
Similarity entre cancion escogida y recomendadas:  [(1187, 0.9847539845102656), (818, 0.9840309506844511), (281, 0.9830586579546059), (12, 0.9807487637
970956), (1867, 0.9788710374641363), (1000, 0.9784134753225019), (1869, 0.9775609145362028), (1047, 0.9774532204955416), (1136, 0.9772488658365086), (1
769, 0.9767138824482666)]
```

In [67]:
```
print("Recomendadas:")
for i in range(0,len(a)):
    print("No.",i+1, " ",a[i])
```

```
Recomendadas:
No. 1   Lollipop
No. 2   My Main
No. 3   Zion Gate Dub
No. 4   Cemalim
No. 5   Adventurers
No. 6   No Security
No. 7   Memorial Day
No. 8   Real Thing
No. 9   Bad Liar
No. 10   U Got It Bad
```

In [63]:
```
sig_kernel = sigmoid_kernel(normalized_df)
a=generate_recommendation('Redbone',sig_kernel).values
```

```
Similarity entre cancion escogida y recomendadas:  [(1106, 0.8644406383322815), (1120, 0.8627221030363934), (598, 0.8625160346873847), (138, 0.86243619
09857228), (222, 0.8616521369907595), (1266, 0.8612202479818735), (1015, 0.8611638718189624), (665, 0.8609989626256612), (557, 0.8608045723131829), (10
75, 0.8604115445719285)]
```

In [64]:
```
print("Recomendadas:")
for i in range(0,len(a)):
    print("No.",i+1, " ",a[i])
```

```
Recomendadas:
No. 1   La Bicicleta
No. 2   Hula Hoop
No. 3   Piss Test (feat. Juicy J & Dany Brown)
No. 4   Sippin On Some Syrup
No. 5   Pass The Dutchie
No. 6   The Happy Song
No. 7   6 In The Morning
No. 8   Fashion Killa
No. 9   Blueberry (Pills & Cocaine) (feat. Danny Brown)
No. 10   No Such Thing as a Broken Heart
```

★Programa de recomendación de playlist usando K Means

LIBRERIAS A UTILIZAR

In [2]:
```
import os
import numpy as np
import pandas as pd

import seaborn as sns
import plotly.express as px
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
from sklearn.metrics import euclidean_distances
from scipy.spatial.distance import cdist



import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
import seaborn as sb
import sklearn.cluster as cluster
import sklearn.metrics as metrics
import sklearn.preprocessing
import scipy.cluster.hierarchy as sch
import pylab
import sklearn.mixture as mixture
import pyclustertend
import random
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.mixture import GaussianMixture


from sklearn.datasets import make_blobs
from sklearn.metrics import silhouette_samples, silhouette_score

import matplotlib.cm as cm

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

In [4]:
```
cd Recomendacion
```

```
C:\Users\diego\Desktop\PROYECTO2DS-main\Recomendacion
```

Leemos los datos a utilizar

In [5]:
```
data = pd.read_csv("data.csv")
genre_data = pd.read_csv('data_by_genres.csv')
year_data = pd.read_csv('data_by_year.csv')
```

In [6]:
```
data
```

Out[6]:

	valence	year	acousticness	artists	danceability	duration_ms	energy	explicit	id	instrumentalness	key	liveness	loudness	mode	
0	0.0594	1921	0.98200	['Sergei Rachmaninoff', 'James Levine', 'Berli...	0.279	831667	0.211	0	4BJqT0PrAfrxzMOxytFOIz	0.878000	10	0.6650	-20.096	1	C N Mi
1	0.9630	1921	0.73200	['Dennis Day']	0.819	180533	0.341	0	7xPhfUan2yNtyFG0cUWkt8	0.000000	7	0.1600	-12.441	1	L th
2	0.0394	1921	0.96100	['KHP Kridhamardawa Karaton Ngayogyakarta Hadi...	0.328	500062	0.166	0	1o6I8BglA6ylDMrIELygv1	0.913000	3	0.1010	-14.850	1	
3	0.1650	1921	0.96700	['Frank Parker']	0.275	210000	0.309	0	3ftBPsC5vPBKxYSee08FDH	0.000028	5	0.3810	-9.316	1	Da
4	0.2530	1921	0.95700	['Phil Regan']	0.418	166693	0.193	0	4d6HGyGT8e121BsdKmw9v6	0.000002	3	0.2290	-10.096	1	Wh
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
170648	0.6080	2020	0.08460	['Anuel AA', 'Daddy Yankee', 'KAROL G', 'Ozuna...	0.786	301714	0.808	0	0KkIkfsLEJbrcIhYsCL7L5	0.000289	7	0.0822	-3.702	1	
170649	0.7340	2020	0.20600	['Ashnikko']	0.717	150654	0.753	0	0OStKKAuXlxA0fMH54Qs6E	0.000000	7	0.1010	-6.020	1	Hall
170650	0.6370	2020	0.10100	['MAMAMOO']	0.634	211280	0.858	0	4BZXVFYCb76Q0Klojq4piV	0.000009	4	0.2580	-2.226	0	
170651	0.1950	2020	0.00998	['Eminem']	0.671	337147	0.623	1	5SIZJoLXp3WOI3J4C8IK0d	0.000008	2	0.6430	-7.161	1	D
170652	0.6420	2020	0.13200	['KEVVO', 'J Balvin']	0.856	189507	0.721	1	7HmnJHfs0BkFzX4x8j0hkl	0.004710	7	0.1820	-4.928	1	Azul

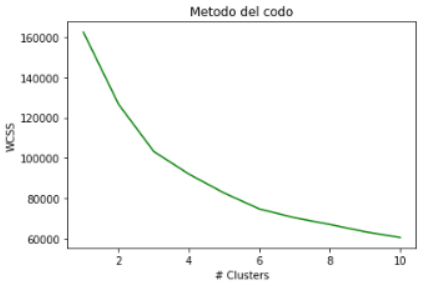
170653 rows × 19 columns

In [8]:
```
#INICIA EL CLUSTERING

Lets = data[['acousticness', 'danceability', 'energy', 'instrumentalness',
       'liveness', 'loudness', 'speechiness', 'tempo', 'valence','duration_ms','explicit','key','mode','year']] #Declaramos nuestra lista de columnas

CantDatos = np.array(Lets.sample(frac=0.1, random_state=123).reset_index(drop=True)) #Declaramos CantDatos
CantDatos_scaled = sklearn.preprocessing.scale(CantDatos) #Realizamos las modificaicones a CantDatos
pyclustertend.hopkins(CantDatos, len(CantDatos)) #Aplicamos HOPKINS para ver  la tendencia para los clusters
```

Out[8]:
```
0.0013610912102140697
```

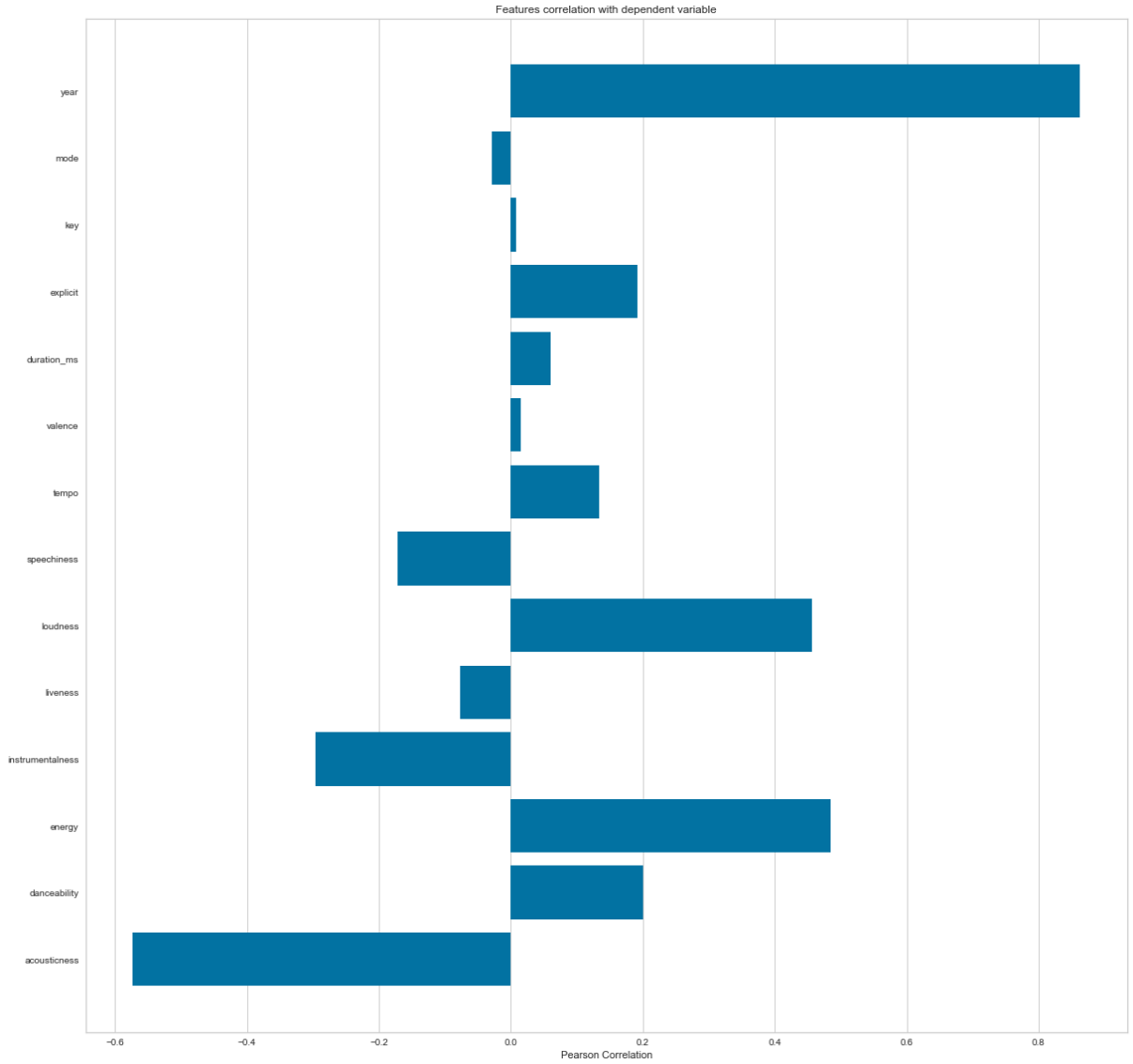
El valor de hopkins nos da un 0.0.001361 alejado de 0.5 validando nuestro clustering.

In [10]:
```
Lets_norm = (Lets - Lets.min()) / (Lets.max() - Lets.min())  #Realizamos una normalizacaion entre las variables a trabajar.
wcss = []

for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, max_iter=300)         #Utilizamos el metodo del codo y el algoritmo KMEANS
    kmeans.fit(Lets_norm)                                 #para determinar la mejor cantidad de clusters dependiendo los datos normalizados.
    wcss.append(kmeans.inertia_)

plt.plot(range(1, 11), wcss,color='green')
plt.title('Metodo del codo')
plt.xlabel('# Clusters')
plt.ylabel('WCSS')
plt.show()
```



Usando yellowbrick se crea una featureCorrelation lo cual crea una visualizacion de los coeficientes de correlacion de Pearson entre la variable dependiente (estamos usando la popularidad) y las features (se pueden ver dentro de feature_names) utilizando una escala de -1 a 1.

In [12]:
```
from yellowbrick.target import FeatureCorrelation

feature_names = ['acousticness', 'danceability', 'energy', 'instrumentalness',
       'liveness', 'loudness', 'speechiness', 'tempo', 'valence','duration_ms','explicit','key','mode','year']

X, y = data[feature_names], data['popularity']

# Create a list of the feature names
features = np.array(feature_names)

# Instantiate the visualizer
visualizer = FeatureCorrelation(labels=features)

plt.rcParams['figure.figsize']=(20,20)
visualizer.fit(X, y)     # Fit the data to the visualizer
visualizer.show()
```



Podemos observar como la acousticness es la feature que menos correlacion tiene con la popularidad y que el año es la feature que más correlacion tiene con la popularidad.

K-Means Clustering De Generos

Se utiliza el algoritmo de K-means para dividir los generos en el dataset en 10 clusters, basandonos en las características numéricas de audio de cada género.

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

cluster_pipeline = Pipeline([('scaler', StandardScaler()), ('kmeans', KMeans(n_clusters=10))])
X = genre_data.select_dtypes(np.number)
cluster_pipeline.fit(X)
genre_data['cluster'] = cluster_pipeline.predict(X)
```

```
# Visualizacion de clusters usando TSNE

from sklearn.manifold import TSNE

tsne_pipeline = Pipeline([('scaler', StandardScaler()), ('tsne', TSNE(n_components=2, verbose=1))])
genre_embedding = tsne_pipeline.fit_transform(X)
projection = pd.DataFrame(columns=['x', 'y'], data=genre_embedding)
projection['genres'] = genre_data['genres']
projection['cluster'] = genre_data['cluster']

fig = px.scatter(
    projection, x='x', y='y', color='cluster', hover_data=['x', 'y', 'genres'])
fig.show()
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 2973 samples in 0.008s...
[t-SNE] Computed neighbors for 2973 samples in 0.231s...
[t-SNE] Computed conditional probabilities for sample 1000 / 2973
[t-SNE] Computed conditional probabilities for sample 2000 / 2973
[t-SNE] Computed conditional probabilities for sample 2973 / 2973
[t-SNE] Mean sigma: 0.777516
[t-SNE] KL divergence after 250 iterations with early exaggeration: 76.110641
[t-SNE] KL divergence after 1000 iterations: 1.390078
```

K-Means Clustering de Canciones

Hacemos el clustering de canciones.

```
song_cluster_pipeline = Pipeline([('scaler', StandardScaler()),
                                  ('kmeans', KMeans(n_clusters=20,
                                   verbose=False))
                                 ], verbose=False)

X = data.select_dtypes(np.number)
number_cols = list(X.columns)
song_cluster_pipeline.fit(X)
song_cluster_labels = song_cluster_pipeline.predict(X)
data['cluster_label'] = song_cluster_labels
```

```
# Visualizing the Clusters with PCA

from sklearn.decomposition import PCA

pca_pipeline = Pipeline([('scaler', StandardScaler()), ('PCA', PCA(n_components=2))])
song_embedding = pca_pipeline.fit_transform(X)
projection = pd.DataFrame(columns=['x', 'y'], data=song_embedding)
projection['title'] = data['name']
projection['cluster'] = data['cluster_label']

fig = px.scatter(
    projection, x='x', y='y', color='cluster', hover_data=['x', 'y', 'title'])
fig.show()
```

Sistema de recomendaciones

- Según el análisis y las visualizaciones, se puede concluir que los géneros similares tienden a tener puntos de datos que se encuentran cerca unos de otros, mientras que los tipos de canciones similares también se agrupan.
- Los géneros similares sonarán de manera similar y provendrán de períodos de tiempo similares, mientras que lo mismo puede decirse de las canciones dentro de esos géneros. Podemos usar esta idea para construir un sistema de recomendación tomando los puntos de datos de las canciones que un usuario ha escuchado y recomendando canciones correspondientes a puntos de datos cercanos.
- Se utilizara spotipy para la siguiente seccion, por lo cual es necesario una client ID y un client secret para su cuenta en especifico.

[ ]:
```
%pip install spotipy
%env SPOTIFY_CLIENT_ID=9c56eda59b544c55a63028477b5b7e18
%env SPOTIFY_CLIENT_SECRET=6fcc6b2dc613464a92c60ed4fc81cb52
```

PREVIAMENTE SE INSTALA SPOTYPY Y POSTERIORMENTE SE UTILIZA MI (Andres Paiz) CLIENT ID Y CLIENT SECRET. AGRADECERIA QUE USARAN LOS SUYOS PARA PRUEBAS O NOTACION GRACIAS ;)

24]:
```
import spotipy
from spotipy.oauth2 import SpotifyClientCredentials
from collections import defaultdict

sp = spotipy.Spotify(auth_manager=SpotifyClientCredentials(client_id=os.environ["SPOTIFY_CLIENT_ID"],
                                                           client_secret=os.environ["SPOTIFY_CLIENT_SECRET"]))

def find_song(name, year):
    song_data = defaultdict()
    results = sp.search(q= 'track: {} year: {}'.format(name,year), limit=1)
    if results['tracks']['items'] == []:
        return None

    results = results['tracks']['items'][0]
    track_id = results['id']
    audio_features = sp.audio_features(track_id)[0]

    song_data['name'] = [name]
    song_data['year'] = [year]
    song_data['explicit'] = [int(results['explicit'])]
    song_data['duration_ms'] = [results['duration_ms']]
    song_data['popularity'] = [results['popularity']]

    for key, value in audio_features.items():
        song_data[key] = value

    return pd.DataFrame(song_data)
```

```
from collections import defaultdict
from sklearn.metrics import euclidean_distances
from scipy.spatial.distance import cdist
import difflib

number_cols = ['valence', 'year', 'acousticness', 'danceability', 'duration_ms', 'energy', 'explicit',
 'instrumentalness', 'key', 'liveness', 'loudness', 'mode', 'popularity', 'speechiness', 'tempo']


def get_song_data(song, spotify_data):

    try:
        song_data = spotify_data[(spotify_data['name'] == song['name'])
                                 & (spotify_data['year'] == song['year'])].iloc[0]
        return song_data

    except IndexError:
        return find_song(song['name'], song['year'])


def get_mean_vector(song_list, spotify_data):

    song_vectors = []

    for song in song_list:
        song_data = get_song_data(song, spotify_data)
        if song_data is None:
            print('Warning: {} does not exist in Spotify or in database'.format(song['name']))
            continue
        song_vector = song_data[number_cols].values
        song_vectors.append(song_vector)

    song_matrix = np.array(list(song_vectors))
    return np.mean(song_matrix, axis=0)


def flatten_dict_list(dict_list):

    flattened_dict = defaultdict()
    for key in dict_list[0].keys():
        flattened_dict[key] = []

    for dictionary in dict_list:
        for key, value in dictionary.items():
            flattened_dict[key].append(value)

    return flattened_dict


def recommend_songs( song_list, spotify_data, n_songs=10):

    metadata_cols = ['name', 'year', 'artists']
    song_dict = flatten_dict_list(song_list)

    song_center = get_mean_vector(song_list, spotify_data)
    scaler = song_cluster_pipeline.steps[0][1]
    scaled_data = scaler.transform(spotify_data[number_cols])
    scaled_song_center = scaler.transform(song_center.reshape(1, -1))
    distances = cdist(scaled_song_center, scaled_data, 'cosine')
    index = list(np.argsort(distances)[:, :n_songs][0])

    rec_songs = spotify_data.iloc[index]
    rec_songs = rec_songs[~rec_songs['name'].isin(song_dict['name'])]
    return rec_songs[metadata_cols].to_dict(orient='records')
```

Le pasamos a la funcion de recomendacion de canciones una playlist de 5 canciones y como output nos da una lista de recomendaciones basandose en la playlist de input.

Favor utilizar el mismo formato de input :v

In [33]:
```
recommend_songs([{'name': 'Robbery', 'year':2019},
                 {'name': 'SAD!', 'year': 2018},
                 {'name': 'Life is Beautiful', 'year': 2018},
                 {'name': 'Lucid Dreams', 'year': 2018},
                 {'name': 'La mano de Dios (Homenaje a Diego Maradona)', 'year': 2011}],  data)
```

Out[33]:
```
[{'name': 'Congratulations',
  'year': 2016,
  'artists': "['Post Malone', 'Quavo']"},
 {'name': "I Think I'm OKAY (with YUNGBLUD & Travis Barker)",
  'year': 2019,
  'artists': "['Machine Gun Kelly', 'YUNGBLUD', 'Travis Barker']"},
 {'name': 'ROXANNE', 'year': 2019, 'artists': "['Arizona Zervas']"},
 {'name': 'Let Me Know (I Wonder Why Freestyle)',
  'year': 2019,
  'artists': "['Juice WRLD']"},
 {'name': 'Rider', 'year': 2019, 'artists': "['Juice WRLD']"},
 {'name': 'Play Date', 'year': 2015, 'artists': "['Melanie Martinez']"},
 {'name': 'Beamer Boy', 'year': 2017, 'artists': "['Lil Peep']"},
 {'name': 'Teardrops', 'year': 2020, 'artists': "['Bring Me The Horizon']"},
 {'name': "I Love You's", 'year': 2020, 'artists': "['Hailee Steinfeld']"},
 {'name': 'Parasite Eve', 'year': 2020, 'artists': "['Bring Me The Horizon']"}]
```

Aqui se mira el output de recomendacion para la playlist de input.

★Programa de recomendación de playlist usando Cosine similarity con min-max

]:
```
import spotipy
from spotipy.oauth2 import SpotifyClientCredentials
from spotipy.oauth2 import SpotifyOAuth
import spotipy.util as util

from skimage import io
import matplotlib.pyplot as plt
import pandas as pd
from datetime import datetime


from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics.pairwise import cosine_similarity



import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
import seaborn as sb
import sklearn.cluster as cluster
import sklearn.metrics as metrics
import sklearn.preprocessing
import scipy.cluster.hierarchy as sch
import pylab
import sklearn.mixture as mixture
import pyclustertend
import random
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.mixture import GaussianMixture


from sklearn.datasets import make_blobs
from sklearn.metrics import silhouette_samples, silhouette_score

import matplotlib.cm as cm
```

]:
```
cd Recomendacion
```

```
[WinError 2] The system cannot find the file specified: 'Recomendacion'
C:\Users\diego\Desktop\PROYECTO2DS-main\Recomendacion
```

]:
```
spotify_data = pd.read_csv("datam2.csv")
```

]:
```
spotify_features_df = spotify_data
genre_OHE = pd.get_dummies(spotify_features_df.genre)
key_OHE = pd.get_dummies(spotify_features_df.key)
```

Como podemos ver que las columnas numéricas tienen diferentes rangos, realizaremos una normalización máxima-mínima para cambiar los valores de las columnas numéricas en el conjunto de datos a una escala estándar.

Es el enfoque de normalización más común donde el valor mínimo en la columna de características se transforma en 0 y el valor máximo en la columna de características se transforma en 1.

[29]:
```
scaled_features = MinMaxScaler().fit_transform([
  spotify_features_df['acousticness'].values,
  spotify_features_df['danceability'].values,
  spotify_features_df['duration_ms'].values,
  spotify_features_df['energy'].values,
  spotify_features_df['instrumentalness'].values,
  spotify_features_df['liveness'].values,
  spotify_features_df['loudness'].values,
  spotify_features_df['speechiness'].values,
  spotify_features_df['tempo'].values,
  spotify_features_df['valence'].values,
  ])
```

[30]:
```
#ALmacenar los vectores de columna transformados en nuestro marco de datos
spotify_features_df[['acousticness','danceability','duration_ms','energy','instrumentalness','liveness','loudness','speechiness','tempo','valence']] =
```

Eliminamos las características que no se consideran para determinar la similitud y las características categóricas que ya están convertidas en vectores OHE.

[31]:
```
spotify_features_df = spotify_features_df.drop('genre',axis = 1)
spotify_features_df = spotify_features_df.drop('artist_name', axis = 1)
spotify_features_df = spotify_features_df.drop('track_name', axis = 1)
spotify_features_df = spotify_features_df.drop('popularity',axis = 1)
spotify_features_df = spotify_features_df.drop('key', axis = 1)
spotify_features_df = spotify_features_df.drop('mode', axis = 1)
spotify_features_df = spotify_features_df.drop('time_signature', axis = 1)
```

[32]:
```
spotify_features_df
```

[32]:

	track_id	acousticness	danceability	duration_ms	energy	instrumentalness	liveness	loudness	speechiness	tempo	valence
0	0BRjO6ga9RKCKjfDqeFgWV	0.000025	0.000022	1.0	0.000028	0.000018	0.000022	0.0	0.000019	0.001699	0.000027
1	0BjC1NfoEOOusryehmNudP	0.000042	0.000045	1.0	0.000046	0.000040	0.000042	0.0	0.000041	0.001307	0.000046
2	0CoSDzoNIKCRs124s9uTVy	0.000087	0.000085	1.0	0.000082	0.000082	0.000082	0.0	0.000082	0.000666	0.000084
3	0Gc6TVm52BwZD07Ki6tIvf	0.000084	0.000081	1.0	0.000082	0.000080	0.000081	0.0	0.000080	0.001207	0.000081
4	0lusIXpMROHdEPvSl1fTQK	0.000267	0.000260	1.0	0.000259	0.000257	0.000258	0.0	0.000256	0.001957	0.000261
...	...	...	...	...	...	...	...	...	...	...	...
232720	2XGLdVl7lGeq8ksM6Al7jT	0.000033	0.000035	1.0	0.000035	0.000034	0.000033	0.0	0.000033	0.000387	0.000036
232721	1qWZdk8l4UVPj9lK6HuuFM	0.000025	0.000027	1.0	0.000027	0.000025	0.000025	0.0	0.000025	0.000428	0.000028
232722	2ziWXUmQLrXTiYjCg2fZ2t	0.000055	0.000053	1.0	0.000052	0.000050	0.000050	0.0	0.000050	0.000554	0.000054
232723	6EFsue2YbIG4Qkq8Zr9Rir	0.000033	0.000035	1.0	0.000035	0.000032	0.000034	0.0	0.000033	0.000482	0.000034
232724	34XO9RwPMKjbvRry54QzWn	0.000021	0.000023	1.0	0.000022	0.000021	0.000021	0.0	0.000021	0.000373	0.000022

232725 rows × 11 columns

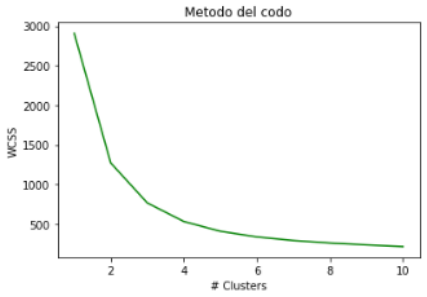
[33]:
```
Lets = spotify_features_df[[ 'acousticness', 'danceability', 'energy', 'energy', 'instrumentalness', 'liveness', 'loudness', 'speechiness', 'tempo', '
```

n [34]:
```
CantDatos = np.array(Lets.sample(frac=0.1, random_state=123).reset_index(drop=True)) #Declaramos CantDatos
CantDatos_scaled = sklearn.preprocessing.scale(CantDatos) #Realizamos las modificaicones a CantDatos
pyclustertend.hopkins(CantDatos, len(CantDatos)) #Aplicamos HOPKINS para ver  la tendencia para los clusters
```

ut[34]:
```
0.003954937094068303
```

El valor de hopkins nos da un 0.003954 alejado de 0.5 validando nuestro clustering.

n [35]:
```
Lets_norm = (Lets - Lets.min()) / (Lets.max() - Lets.min())  #Realizamos una normalizacaion entre las variables a trabajar.
wcss = []

for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, max_iter=300)         #Utilizamos el metodo del codo y el algoritmo KMEANS
    kmeans.fit(Lets_norm)                                 #para determinar la mejor cantidad de clusters dependiendo los datos normalizados.
    wcss.append(kmeans.inertia_)

plt.plot(range(1, 11), wcss,color='green')
plt.title('Metodo del codo')
plt.xlabel('# Clusters')
plt.ylabel('WCSS')
plt.show()
```



In [8]:
```
# Append a las columnas OHE de las características categóricas
spotify_features_df = spotify_features_df.join(genre_OHE)
spotify_features_df = spotify_features_df.join(key_OHE)
```

Al igual que en el modelo de k means utilizamos la api de spotify web

n [39]:
```
client_id = "9c56eda59b544c55a63028477b5b7e18"
client_secret= "6fcc6b2dc613464a92c60ed4fc81cb52"
```

ESTAS CLAVES SON DE LA CUENTA PERSONAL DEL AUTOR

Guardamos los detalles de la playlist en un diccionario de python

1]:
```
scope = 'user-library-read'
token = util.prompt_for_user_token(
    scope,
    client_id= client_id,
    client_secret=client_secret,
    redirect_uri='http://localhost:8881/callback'
  )
sp = spotipy.Spotify(auth=token)
playlist_dic = {}
playlist_cover_art = {}

for i in sp.current_user_playlists()['items']:
    playlist_dic[i['name']] = i['uri'].split(':')[2]
    playlist_cover_art[i['uri'].split(':')[2]] = i['images'][0]['url']

print(playlist_dic)
```

```
{'Daphne': '2h6g4MNyMwIyl5legG7q33', 'Lone': '0mYTzihgs31hj0yqomfNYN', 'Estudiar': '3fZDKXsEHPqa289vLqItdb', 'Mis pistas de Shazam': '0VDyTRhKn3Gr2zdnvbiFHl'}
```

El siguiente método crea una df para nuestra playlist utilizando el conjunto de datos de características de canciones de Spotify.

2]:
```
def generate_playlist_df(playlist_name, playlist_dic, spotify_data):

    playlist = pd.DataFrame()

    for i, j in enumerate(sp.playlist(playlist_dic[playlist_name])['tracks']['items']):
        playlist.loc[i, 'artist'] = j['track']['artists'][0]['name']
        playlist.loc[i, 'track_name'] = j['track']['name']
        playlist.loc[i, 'track_id'] = j['track']['id']
        playlist.loc[i, 'url'] = j['track']['album']['images'][1]['url']
        playlist.loc[i, 'date_added'] = j['added_at']

    playlist['date_added'] = pd.to_datetime(playlist['date_added'])

    playlist = playlist[playlist['track_id'].isin(spotify_data['track_id'].values)].sort_values('date_added',ascending = False)

    return playlist
playlist_df = generate_playlist_df('Lone', playlist_dic, spotify_data)
```

3]:
```
playlist_df.head()
```

3]:

	artist	track_name	track_id	url	date_added
17	Peter Manos	In My Head	1tT55K6VEyO6XFDxK4IDQe	https://i.scdn.co/image/ab67616d00001e023c0c1f...	2020-09-18 13:57:12+00:00
15	XXXTENTACION	changes	7AFASza1mXqntmGtbxXprO	https://i.scdn.co/image/ab67616d00001e02806c16...	2020-09-18 13:53:57+00:00
14	Alec Benjamin	Let Me Down Slowly	2qxmye6gAegTMjLKEBoR3d	https://i.scdn.co/image/ab67616d00001e02459d67...	2020-09-18 13:52:49+00:00
13	Lil Yung Pharaoh	I Hate You, But I Love You	4upK9OgMepvBFEGD8CDE9A	https://i.scdn.co/image/ab67616d00001e02186fb7...	2020-09-18 13:52:35+00:00
12	Ivan B	Back to You	3hX2VFUZi2W5sIqkP7hAvw	https://i.scdn.co/image/ab67616d00001e02a6d9a0...	2020-09-18 13:51:10+00:00

```
from skimage import io
import matplotlib.pyplot as plt

def visualize_cover_art(playlist_df):
    temp = playlist_df['url'].values
    plt.figure(figsize=(15,int(0.625 * len(temp))) , facecolor='#8cfc03')
    columns = 5

    for i, url in enumerate(temp):
        plt.subplot(len(temp) / columns + 1, columns, i + 1)

        image = io.imread(url)
        plt.imshow(image)
        plt.xticks([])
        plt.yticks([])
        s=''
        plt.xlabel(s.join(playlist_df['track_name'].values[i].split(' ')[:4]), fontsize = 10, fontweight='bold')
        plt.tight_layout(h_pad=0.8, w_pad=0)
        plt.subplots_adjust(wspace=None, hspace=None)

    plt.show()
```

Para realizar la cosine similarity entre nuestra playlist y las canciones que no están presentes en nuestra playlist, se resume nuestra playlist en un vector. Este vector representará nuestra playlist en el espacio de caracteristicas, y podremos encontrar canciones similares a las canciones en nuestra playlist.

El siguiente metodo devuelve nuestra playlist como un solo vector y todas las canciones que no están presentes en nuestra lista de reproducción en una df.

```
def generate_playlist_vector(spotify_features, playlist_df, weight_factor):

    spotify_features_playlist = spotify_features[spotify_features['track_id'].isin(playlist_df['track_id'].values)]
    spotify_features_playlist = spotify_features_playlist.merge(playlist_df[['track_id','date_added']], on = 'track_id', how = 'inner')

    spotify_features_nonplaylist = spotify_features[~spotify_features['track_id'].isin(playlist_df['track_id'].values)]

    playlist_feature_set = spotify_features_playlist.sort_values('date_added',ascending=False)


    most_recent_date = playlist_feature_set.iloc[0,-1]

    for ix, row in playlist_feature_set.iterrows():
        playlist_feature_set.loc[ix,'days_from_recent'] = int((most_recent_date.to_pydatetime() - row.iloc[-1].to_pydatetime()).days)


    playlist_feature_set['weight'] = playlist_feature_set['days_from_recent'].apply(lambda x: weight_factor ** (-x))

    playlist_feature_set_weighted = playlist_feature_set.copy()

    playlist_feature_set_weighted.update(playlist_feature_set_weighted.iloc[:,:-3].mul(playlist_feature_set_weighted.weight.astype(int),0))

    playlist_feature_set_weighted_final = playlist_feature_set_weighted.iloc[:, :-3]



    return playlist_feature_set_weighted_final.sum(axis = 0), spotify_features_nonplaylist
```

```
playlist_vector, nonplaylist_df = generate_playlist_vector(spotify_features_df, playlist_df, 1.2)
print(playlist_vector.shape)
print(nonplaylist_df.head())
```

```
(50,)
                track_id  acousticness  danceability  duration_ms    energy  \
0  0BRjO6ga9RKCKjfDqeFgWV      0.000025      0.000022          1.0  0.000028
1  0BjC1NfoEOOusryehmNudP      0.000042      0.000045          1.0  0.000046
2  0CoSDzoNIKCRs124s9uTVy      0.000087      0.000085          1.0  0.000082
3  0Gc6TVm52BwZD07Ki6tIvf      0.000084      0.000081          1.0  0.000082
4  0IuslXpMROHdEPvSl1fTQK      0.000267      0.000260          1.0  0.000259

   instrumentalness  liveness  loudness  speechiness     tempo  ...  B  C  C#  \
0          0.000018  0.000022       0.0     0.000019  0.001699  ...  0  0   1
1          0.000040  0.000042       0.0     0.000041  0.001307  ...  0  0   0
2          0.000082  0.000082       0.0     0.000082  0.000666  ...  0  1   0
3          0.000080  0.000081       0.0     0.000080  0.001207  ...  0  0   1
4          0.000257  0.000258       0.0     0.000256  0.001957  ...  0  0   0

   D  D#  E  F  F#  G  G#
0  0   0  0  0   0  0   0
1  0   0  0  0   1  0   0
2  0   0  0  0   0  0   0
3  0   0  0  0   0  0   0
4  0   0  0  1   0  0   0

[5 rows x 50 columns]
```

Se utiliza cosine similarity como una métrica de similitud para determinar las canciones que son muy similares a nuestra playlist.

Realizaremos la cosine similarity entre nuestro vector de playlist y las canciones que no están presentes en playlist.

Luego, realizaremos la cosine similarity utilizando una biblioteca basada en Python, Scikit, y almacenaremos los valores de cosine similarity en una columna separada.

A continuación, invertiremos la ordenación del df en función de la columna de cosine similarity. Finalmente, generaremos las 15 mejores recomendaciones de canciones como nuestra playlist recomendada.

```
def generate_recommendation(spotify_data, playlist_vector, nonplaylist_df):

    non_playlist = spotify_data[spotify_data['track_id'].isin(nonplaylist_df['track_id'].values)]
    non_playlist['sim'] = cosine_similarity(nonplaylist_df.drop(['track_id'], axis = 1).values, playlist_vector.drop(labels = 'track_id').values.resha
    non_playlist_top15 = non_playlist.sort_values('sim',ascending = False).head(15)
    non_playlist_top15['url'] = non_playlist_top15['track_id'].apply(lambda x: sp.track(x)['album']['images'][1]['url'])

    return  non_playlist_top15
```

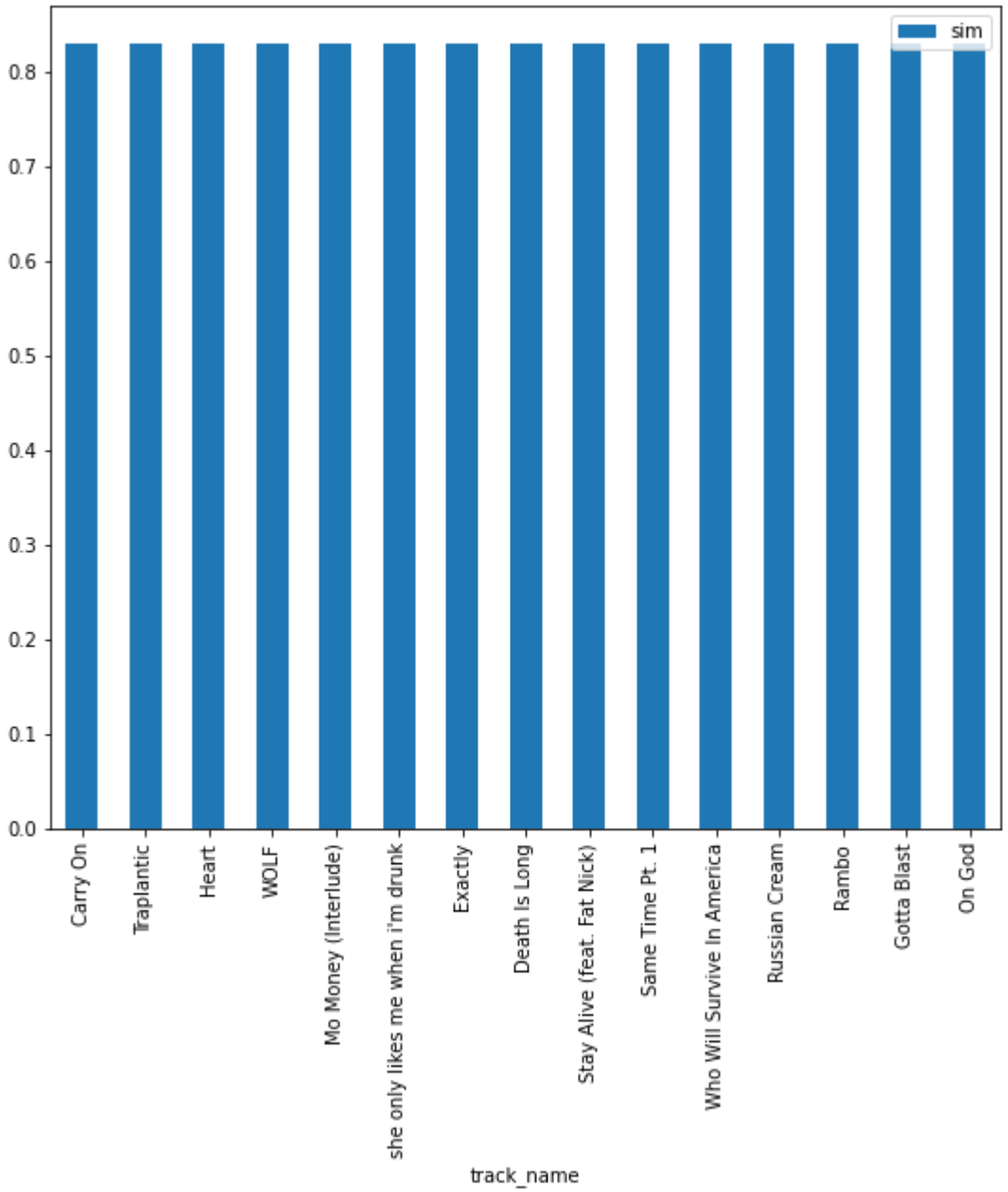
```
top15 = generate_recommendation(spotify_data, playlist_vector, nonplaylist_df)
top15.head(15)
```

[68]:

	genre	artist_name	track_name	track_id	popularity	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	n
87125	Rap	XXXTENTACION	Carry On	2yZax79pOrYuylFVW2cZY2	80	0.000109	0.000109	1.0	0.000105	0.000103	D#	0.000105	0.0	M
117800	Rap	Shoreline Mafia	Traplantic	6ymdTUcJygnxQcRDXDRG5F	57	0.000083	0.000090	1.0	0.000087	0.000083	C#	0.000084	0.0	M
122157	Rap	Lil Tracy	Heart	3FwjVGj09AuW9cplShwmFI	55	0.000101	0.000105	1.0	0.000103	0.000101	C#	0.000101	0.0	M
119423	Rap	Tyler, The Creator	WOLF	2JZpS2sOdJy37o8pn1GuPW	53	0.000108	0.000109	1.0	0.000108	0.000105	C#	0.000109	0.0	M
117802	Rap	J. Cole	Mo Money (Interlude)	3Iz0A1nuTjiLvXQaBxjWzn	57	0.000108	0.000106	1.0	0.000108	0.000098	C#	0.000100	0.0	M
118464	Rap	frumhere	she only likes me when i'm drunk	3VDags0na6nY1gBjCt2e8v	60	0.000124	0.000125	1.0	0.000124	0.000127	C#	0.000121	0.0	M
120299	Rap	Moneybagg Yo	Exactly	08LjGHJ3xsccMyCJdztsns	52	0.000071	0.000077	1.0	0.000076	0.000071	C#	0.000072	0.0	M
120712	Rap	Ski Mask The Slump God	Death Is Long	5uTdUCENzIlVXTfse6MLGX	53	0.000096	0.000100	1.0	0.000098	0.000093	C#	0.000096	0.0	M
120737	Rap	BEXEY	Stay Alive (feat. Fat Nick)	0NtYGyNUHAvdjI4nlYGRMw	58	0.000074	0.000078	1.0	0.000079	0.000073	C#	0.000074	0.0	M
122058	Rap	Big Sean	Same Time Pt. 1	7yoe9vbbzakNAXwb1UaWLS	52	0.000087	0.000085	1.0	0.000085	0.000079	C#	0.000083	0.0	M
118402	Rap	Kanye West	Who Will Survive In America	4e5IPJxSGVWPsaxQzdRseN	55	0.000065	0.000069	1.0	0.000071	0.000062	C#	0.000071	0.0	M
118962	Rap	Roy Woods	Russian Cream	3bJS7RCjB6aNp5J219AmN3	59	0.000065	0.000069	1.0	0.000068	0.000064	C#	0.000065	0.0	M
119869	Rap	Ski Mask The Slump God	Rambo	6xyYSp2HCpeEzG5G4qir76	55	0.000061	0.000068	1.0	0.000066	0.000061	C#	0.000062	0.0	M
115613	Rap	Tay-K	Gotta Blast	7de1qG5xUfnKLNqncRksuk	62	0.000060	0.000065	1.0	0.000064	0.000060	C#	0.000062	0.0	M
119126	Rap	Gucci Mane	On God	6kBCCK9E4CnzcSkpXhwcWb	56	0.000061	0.000066	1.0	0.000065	0.000061	C#	0.000061	0.0	M

[72]:

```
top15.plot(x="track_name", y=["sim"], kind="bar", figsize=(9, 8))
plt.show()
```


sim
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Carry On
Traplantic
Heart
WOLF
Mo Money (Interlude)
she only likes me when i'm drunk
Exactly
Death Is Long
Stay Alive (feat. Fat Nick)
Same Time Pt. 1
Who Will Survive In America
Russian Cream
Rambo
Gotta Blast
On God
track_name

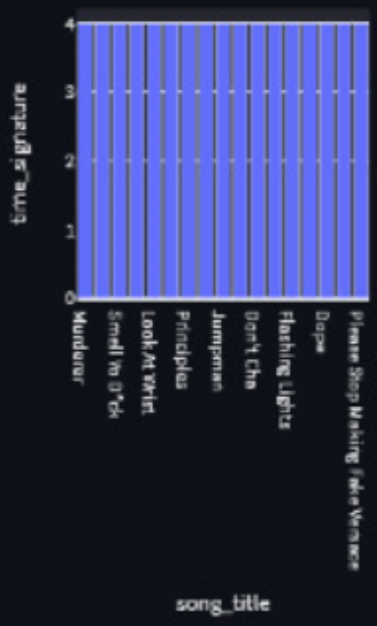
APLICACIÓN FINAL

Recomendacion para: Lollipop

Comparacion de tus Recomendaciones

Escoge una feature

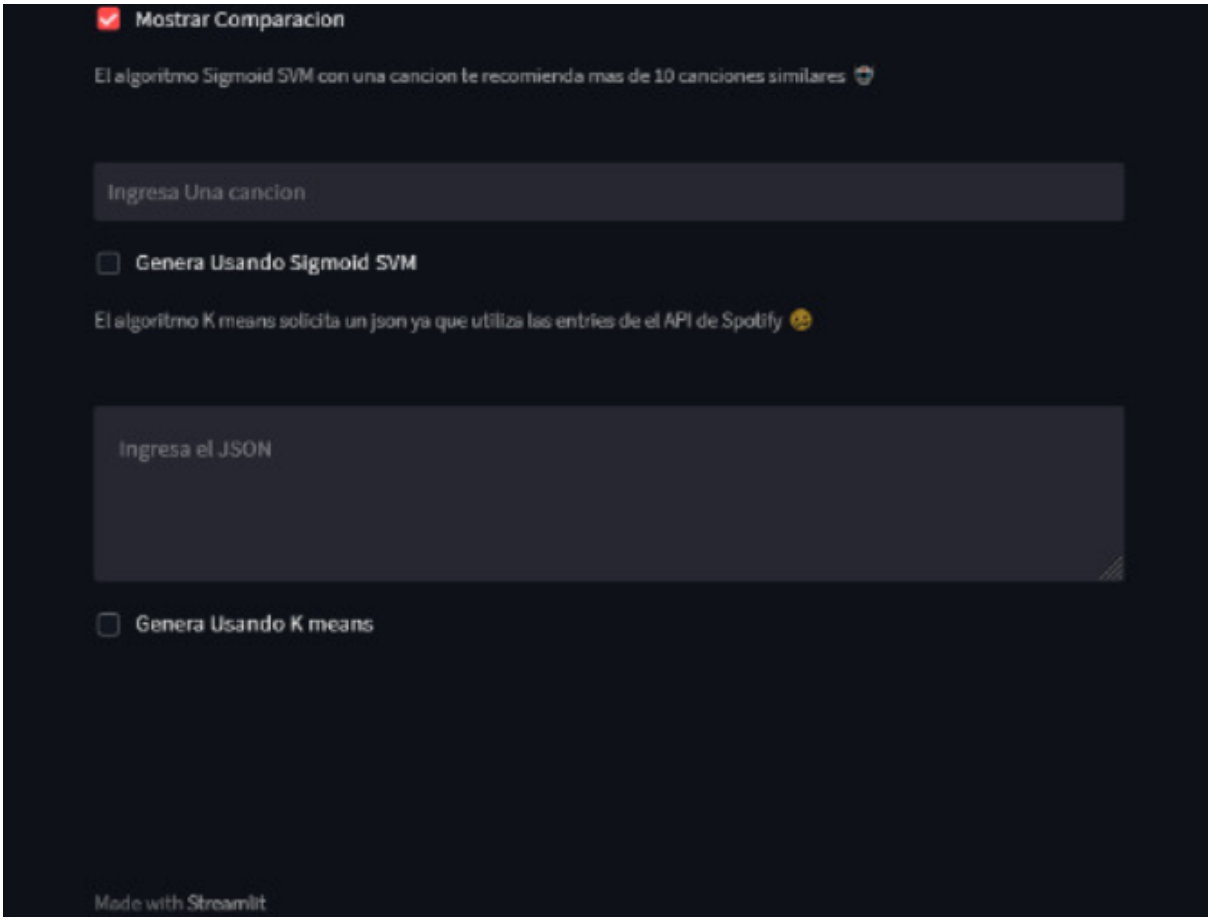
time_signature



Tus recomendaciones 🔥

	song_title
140	Sittin' Sidewayz - feat. Big Pokey
37	Jumpman
216	Hannah Montana
1928	Don't Cha
141	Gold and a Pager
471	Flashing Lights
117	Hotline Bling
732	Dope
586	Traffic
29	Please Stop Making Fake Versace

Mostrar Comparacion


Mostrar Comparacion
El algoritmo Sigmoid SVM con una cancion le recomienda mas de 10 canciones similares
Ingresa Una cancion
Genera Usando Sigmoid SVM
El algoritmo K means solicita un json ya que utiliza las entries de el API de Spotify
Ingresa el JSON
Genera Usando K means
Made with Streamlit

Conclusiones

El algoritmo de agrupamiento de K-means esta garantizado para converger a un resultado. El resultado puede ser un óptimo local (es decir, no necesariamente el mejor resultado posible), lo que significa que evaluar más de una ejecución del algoritmo con centroides iniciales aleatorios puede dar un mejor resultado. Para el clustering de canciones y generos nos ayuda a tener un modelo visual del conjunto de datos, mas que nada por la cantidad de datos que se obtuvo en el dat set. Definitivamente es un modelo que debemos de incluir en el proyecto para poder no solo tener conclusiones reales y confiables si no terminar de entender el dataset. La idea principal detrás de la normalización/estandarización es siempre la misma. Las variables que se miden en diferentes escalas no contribuyen por igual a la función de ajuste del modelo y aprendizaje del modelo y pueden terminar creando un sesgo. Por lo tanto, para lidiar con este problema potencial, la normalización de funciones, como Min-Max Scaling, generalmente se usa antes del ajuste del modelo. Esto puede ser muy útil para algunos modelos ML como los perceptrones multicapa (MLP), donde la propagación hacia atrás puede ser más estable e incluso más rápida cuando las características de entrada tienen una escala mínima-máxima (o en general escala) en comparación con el uso del original. La similitud del coseno es una medida que cuantifica la similitud entre dos o más vectores. La similitud del coseno es el coseno del ángulo entre vectores. Los vectores suelen ser distintos de cero y están dentro de un espacio de producto interno. Este tambien se describe matemáticamente como la división entre el producto escalar de los vectores y el producto de las normas euclidianas o la magnitud de cada vector. La similitud del coseno es una técnica de medición de similitud de uso común que se puede encontrar en bibliotecas y herramientas ampliamente utilizadas como Matlab, SciKit-Learn, TensorFlow, etc.

Los algoritmos SVM utilizan un conjunto de funciones matemáticas que se definen como el kernel. La función del kernel es tomar datos como entrada y transformarlos en la forma requerida. Diferentes algoritmos SVM usan diferentes tipos de funciones del núcleo. Estas funciones pueden ser de diferentes tipos. Por ejemplo, lineal, no lineal, polinomial, función de base radial (RBF) y sigmoide. Introduzca las funciones de Kernel para datos de secuencia, gráficos, texto, imágenes y vectores. El tipo de función kernel más utilizado es RBF. Porque tiene una respuesta localizada y finita a lo largo de todo el eje x. Las funciones del núcleo devuelven el producto interno entre dos puntos en un espacio de características adecuado. Definiendo así una noción de similitud, con poco coste computacional incluso en espacios de muy alta dimensión. En base a estos 4 modelos vamos a trabajar para poder obtener visualizaciones estáticas reales que puedan ser faciles de comprender para cualquier usuario. Lo unico dificil de este reto sera el manejo del dataset ya que si son demasiados los datos que hay que manejar. Por esta misma razón, consideramos que estos 4 modelos podran brindarnos estadisticas confiables.

La generación de clusters no tiene fundamento, por ende, debemos utilizar un método estadístico para obtener la cantidad de clusters más óptima. Analizando nuestros resultados concluimos que el método del codo o el método de la silueta serían óptimos para obtener esta cantidad. Por otra parte, se utilizaron datasets no del mismo tamaño , se debe obtener una muestra del dataset de spotify que cese con el dataset Kagle en términos de tamaño. Los algoritmos utilizados brindaron un sistema de recomendación bastante acertado, el algoritmo de cosine similarity fue el que más precisión tuvo ya que el valor fue muy cercano a 1 , indicando alta precisión. Todos los algoritmos van a ser utilizados en el progreso del proyecto, Cosine similarity fue el mas preciso en esta iteración , sin embargo, cuando se haga la modificación a los clústeres puede que alguno de los otros algoritmos utilizados sea más preciso.

Referencias

Han, Jiawei, Micheline Kamber, and Jian Pei. 2012. Data Mining: Concepts and Techniques. 3rd ed. Boston: Morgan Kaufmann. https://doi.org/10.1016/B978-0-12-381479-1.00016-2.

IBM Cloud Education. (2020). Natural Language Processing (NLP). IBM Cloud Learn Hub. Recuperado de: https://www.ibm.com/cloud/learn/natural-language-processin

(2022). Retrieved 25 September 2022, from https://www.hindawi.com/journals/cin/2022/7157075/

Music Recommendation System using Spotify Dataset. (2022). Retrieved 26 September 2022, from https://www.kaggle.com/code/vatsalmavani/music-recommendation-system-using-spotify-dataset/notebook

Music Recommender System Based on Genre using Convolutional Recurrent Neural Networks . (2022). Retrieved 26 September 2022, from https://www.sciencedirect.com/science/article/pii/S1877050919310646