**Desafío Clase 44. TERCERA ENTREGA DEL PROYECTO FINAL**

**Top Hits Spotify from 2000-2019**

# Conformación del equipo de trabajo.

1. Sebastian Vadra
2. Santiago Pradini
3. Tania Garnica

# Presentación de la empresa, organización o problema específico.

### [Top Hits Spotify from 2000-2019](https://www.kaggle.com/code/thomaslazarus/visualizing-spotify-top-hits/data)

Spotify es un servicio de música, podcasts y videos digitales que le brinda acceso a millones de canciones y otro contenido de creadores de todo el mundo. Spotify está disponible en una gran variedad de dispositivos, como ordenadores, teléfonos, tabletas, altavoces, televisores o coches, y puedes pasar fácilmente de uno a otro.

El problema está en que, mientras más listas existan, menos relevancia tienen y esto dificulta identificar y medir el éxito de un artista frente a otro. Si un artista es Nº1 en YouTube pero Nº20 en la lista semanal de Spotify, ¿significa que es más o menos popular que alguien que ocupe el Nº8 en ambas listas? ¿Qué pasa con artistas que no aparecen en listas de popularidad pero que hacen llenos totales de sus conciertos en recintos más grandes que las estrellas del streaming? ¿Qué tan populares son?

Este conjunto de datos contiene estadísticas de audio de las 2000 mejores pistas en Spotify de 2000 a 2019. Los datos contienen alrededor de 18 columnas, cada una de las cuales describe la pista y sus cualidades.

**Problemas planteados:**

* El primer problema que buscamos abordar tiene que ver con la gran cantidad de datos dispares respecto a canciones y artistas, ya que existe una gran cantidad de ellos a lo largo del período tomado por el dataset que va desde el año 2000 al año 2019.  
  Para ello buscamos abordar estas cuestiones limitando la cantidad de datos que se toman, haciendo un **histplot** tanto del Target “Popularity” como de los valores de “duration\_ms” y “loudness”. Esta limitación fue desarrollada reduciendo el ruido mediante la eliminación de valores outliers que marcan un claro sesgo hacia la derecha en caso de popularity, hacia la izquierda en el caso de “duration\_ms” y hacia la derecha en “loudness”.
* El segundo problema se plantea con el género de las canciones. Existen no sólo géneros sino subgéneros como “pop”, “pop-rock”, “hip-hop, pop, rock”, etc.  
  Para esto lo que hicimos fue generar un **Barplot** para ver como se distribuyen todos estos géneros y tomamos la decisión de tomar sólo el primer nombre del género mediante la denominación **first\_genre** y vemos mediante un **countplot** como queda la distribución para luego eliminar los outliers de “**Folk/Acoustic”** y “**easy listening”**.

# Correlaciones:

1. Una vez que pudimos reducir el ruido para elevar la posibilidad de brindar mejor calidad de predicción, pasamos a analizar los métodos de correlación mediante **Spearman**.  
     
   En esta correlación podemos observar que las más fuertes se dan entre

* Explicit y Speechiness
* Explicit y Danceability
* Energy y Loudness
* Energy y Valence
* Valence y Danceability

# 

# **Regresión lineal**

1. Lo primero que hacemos es tomar las correlaciones de las que hablamos más arriba para elaborar un modelo de regresión lineal con lo cual spliteamos para el entrenamiento, aplicamos One Hot Encoder entrenando el first\_genre que generamos previamente con los keys.
2. Luego de testear y predecir vemos que el valor es:  
     
   **R2** = **0.14**

# **Random Forest**

1. Agarramos las variables Energy, eliminamos sus outliers
2. Hacemos lo mismo con Danceability y Acousticness.
3. Creamos nuestra variable Target en Popularity, que es la que cumple esa función en nuestro trabajo.
4. Entrenamos, hacemos el encoding de la totalidad del Dataset, y luego eliminamos algunas de las variables de Total\_train y luego realizamos las transformaciones numéricas Preprocessing y escalamos con MinMaxScaler.
5. Clasificamos importando el **Random Forest Regressor**, transformamos las categóricas estableciendo los estimadores y clasificamos las variables creadas con las escalas y entrenamientos seleccionados.
6. Luego de testear nos da el resultado de R2 algo más alto que con la regresión lineal pero no tanto. Su valor es:  
     
   **R2 = 0.16**
7. Finalmente ordenamos un nuevo DF por importancia de Feature dando como mayor valor Year y luego Energy.

# **Regresión logística**

1. Tomamos la regresión logística armando una función que nos permita crear un target por si Energía da más alto de 70 o menor.
2. Entrenamos el target seleccionado (Energy)
3. Sumamos esa transformación a nuestro entrenamiento mediante un **merge**.
4. Escalamos con MinMaxScaler e importamos **LogisticRegression**
5. Luego de Encodear y entrenar, establecemos la predicción que nos da:  
     
   **Accuracy = 0.67  
     
   f1-Score = 0.75 (-70 popularity) ; 0.54 (+70 popularity)  
     
   Precision = 0.70 (-70 popularity) ; 0.61 (+70 popularity)**

# **Clasificación**

1. Establecemos una función para cada uno de los segmentos de valores de nuestro Target Popularity.
2. Consultamos la proporción de nuestro target en el dataset que da que el mayor resultado es el de:  
     
   **Popularity + 70 = 0.397.**
3. Concluimos que no da un valor muy grande.

# **SVM**

Luego de definir nuestras variables x e y, spliteamos para ejecutar el training y el Test.  
  
Agrupamos las categóricas de nuestro modelo y transformamos con un *preprocessor.transform(X\_train)*. Eseos datos agrupados lo llevamos a una nueva columna que llamamos “*Columnas\_enc”*

Después creamos el conjunto encodeado para el train como X\_train\_enc para poder visualizar los conjuntos.

Importamos el algoritmo SVM desde la librería *sklearn* y entrenamos el classifier. Luego predecimos los resultados y comparamos la predicción con los resultados verdaderos .  
  
Testeamos y luego consultamos las métricas del modelo que nos dan:  
  
Un reloj digital en la pantalla

Descripción generada automáticamente con confianza media

# **AGRUPAMIENTO**

Comenzamos creando un nuevo target dividiendo la variable popularity en 3 labels. Luego, depende el rango de valores que toca dividimos esos labels en:  
  
**Popularidad baja**

**Popularidad media**

**Popularidad alta**

Con estos 3 laels reemplazamos en nuestro dataset los valores según esos rangos de valores, con lo cual nos permite, mediante un *value\_counts,* contabilizar qué cantidad hay de cada grado de popularidad, siendo el resultado el siguiente:  
  
alta 0.579247   
media 0.398518   
baja 0.022235

Esto nos indica que está bien distribuido pero que hay un muy bajo porcentaje para la popularidad baja.

Luego entrenamos , importamos el *KNeighbors* donde el resultado de nuestro train es muy alto (0.99786), mientras que el test da (0.57834), lo cual indica un muy buen resultado para el training pero no para el test.

Finalmente, por medio de un *cross validation* nuestro resultado nos da valores moderados:

Interfaz de usuario gráfica, Calendario

Descripción generada automáticamente con confianza media

Por eso es que buscamos, mediante una matriz de confusión mejorar el resultado pero no varió significativamente:

Imagen de la pantalla de un computador

Descripción generada automáticamente con confianza baja