# Proyecto final Gestión de datos
# Entrega Final

Damián Felipe Castillo González[a], Edwin Leonardo Silva Piracoca[b] & Daniel Alejandro Córdoba Pulido[c]

[a] *Estudiante Facultad de ingeniería, PontificiaUniversidad Javeriana, damian.castillo01@javeriana.edu.co*
[b] *Estudiante Facultad de ingeniería, Pontificia Universidad Javeriana, e.silva@javeriana.edu.co*
[c] *Estudiante Facultad de ingeniería, Pontificia Universidad Javeriana, daniel_cordobap@javeriana.edu.co*

Para esta entrega se tendrán en cuenta los siguientes criterios , otorgados para el desarrollo del entregable número 1 donde se realizó un análisis exploratorio de los datos, análisis de la calidad y limpieza de los mismos.
Para esto tenemos en cuenta los archivos de trabajo de nombres : artists mod.csv y tracks mod.csv , la información contenida en estos archivos nos permitirá relacionar la información con sus diferentes campos .

Iniciamos validando los tamaños de dichos archivos:

- *print(tracks_mod.shape).= 586672, 20)*
- *print(artists_mod.shape) = (1162095, 5)*

Con estos valores damos inicio a la configuración del dataset y su análisis, para esto comenzamos con su análisis exploratorio, encontrando la información más relevante dentro de estos datasets, de esta forma utilizamos el comando head()" del DataFrame "artists_mod" para mostrar las primeras 5 filas del DataFrame en la consola, este DataFrame tiene como tamaño el siguiente valor (1162095, 5) entregándonos tamaños de filas y columnas ,el dato obtenido de 1162085 corresponde a ID de los artistas , entregándonos la información de valores para la identificación de distintos artistas , se confirma con el comando .nunique() , que nos confirma que existen esta cantidad de artistas únicos dentro del DataSet , continuando con el EDA implementamos el comando "describe()" este se aplica a esta columna para calcular estadísticas descriptivas como el número de observaciones (count), la media (mean), la desviación estándar (std), los valores mínimo y máximo (min y max), y los percentiles 25, 50 y 75. Esto mismo lo aplicamos tanto para popularidad como para seguidores, los valores obtenidos en ambos casos se presentan a continuación:

- *summary_followers = artists_mod["followers"].describe()*
- *summary_popularity = artists_mod["popularity"].describe()*

| | |
|---|---|
| count | 1.162084e+06 |
| mean | 1.022070e+04 |
| std | 2.543995e+05 |
| min | 0.000000e+00 |
| 25% | 1.000000e+01 |
| 50% | 5.700000e+01 |
| 75% | 4.170000e+02 |
| max | 7.890023e+07 |

Name: followers, dtype: float64

| | |
|---|---|
| count | 1.162095e+06 |
| mean | 8.795961e+00 |
| std | 1.355777e+01 |
| min | 0.000000e+00 |
| 25% | 0.000000e+00 |
| 50% | 2.000000e+00 |
| 75% | 1.300000e+01 |
| max | 1.000000e+02 |

Name: popularity, dtype: float64

La función "**np.percentile()**" de la biblioteca NumPy se aplica a esta columna para calcular el percentil 99.2, que indica el valor por debajo del cual se encuentra el 99.2% de los datos, nos da de esta forma un total de 58, utilizamos la función "**.cut()**" que nos ayuda a segmentar y ordenar valores

de datos en contenedores. Esta función también es útil para pasar de una variable continua a una variable categórica:

- *artists_mod["Artist_popularity_bin"]=pd.cut(artists_mod["popularity"],5,labels=["low","low_med","medium","med_high", "High"])*

esta línea de código crea una nueva columna en el DataFrame "artists_mod" que clasifica los valores de la columna "popularity" en cinco categorías diferentes y los etiqueta con las etiquetas "low", "low_med", "medium", "med_high" y "High". Este resultado puede ser útil para analizar la distribución de los artistas en diferentes niveles de popularidad.

| | id | followers | genres | name | popularity | Artist_popularity_bin |
|---|---|---|---|---|---|---|
| 0 | 0DheY5irMjBUeLybbCUEZ2 | 0.0 | [] | Armid & Amir Zare Pashai feat. Sara Rouzbehani | 0 | low |
| 1 | 0DlhY15l3wsrnlfGio2bjU | 5.0 | [] | ปูนา ภาวิณี | 0 | low |
| 2 | 0DmRESX2JknGPQyO15yxg7 | 0.0 | [] | Sadaa | 0 | low |
| 3 | 0DmhnbHjm1qw6NCYPeZNgJ | 0.0 | [] | Tra'gruda | 0 | low |

- *grouped_single = artists_mod.groupby('Artist_popularity_bin').agg({'popularity': ['mean', 'min', 'max',"count"]})*

Esta porcion de codigo agrupa los datos en el DataFrame "artists_mod" por la columna "Artist_popularity_bin" y aplica cuatro funciones de agregación (media, mínimo, máximo y recuento) a la columna "popularity" del DataFrame, "grouped_single" es un nuevo DataFrame que muestra estas estadísticas descriptivas, se complementa con el código.

- *grouped_single["Ratio_Populatiry"]=[(964291/1162095)*100,(145893/1162095)*100,(45166/1162095)*100,(6404/1162095)*100,(341/1162095)*100].*

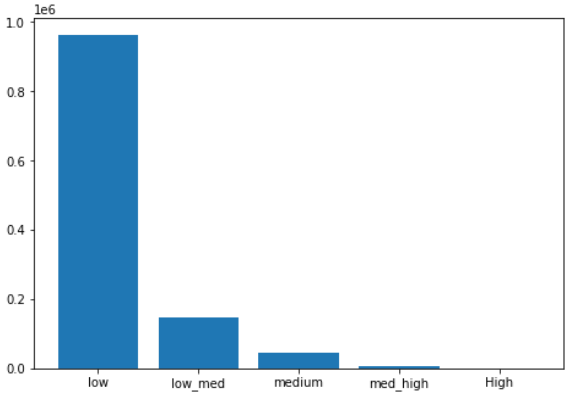
Calcula la proporción de artistas en cada categoría de popularidad en relación con el número total de artistas en el DataFrame. Estas proporciones se calculan #como un porcentaje y se asignan a una nueva columna "Ratio_Populatiry" en el DataFrame "grouped_single", seguido a esto confirmamos el comportamiento del nuevo dataframe:

| | popularity | | | | Ratio_Populatiry |
|---|---|---|---|---|---|
| | mean | min | max | count | |
| Artist_popularity_bin | | | | | |
| low | 3.484229 | 0 | 20 | 964291 | 82.978672 |
| low_med | 29.060106 | 21 | 40 | 145893 | 12.554309 |
| medium | 47.960634 | 41 | 60 | 45166 | 3.886601 |
| med_high | 66.685821 | 61 | 80 | 6404 | 0.551074 |
| High | 85.099707 | 81 | 100 | 341 | 0.029344 |

Para nuestro análisis utilizamos una de las herramientas de la librería MATPLOTLIB que nos proporciona resultados visuales

- *import matplotlib.pyplot as plt*

Se utiliza la biblioteca Matplotlib para crear un gráfico de barras que muestra el recuento de artistas en cada categoría de popularidad en el Data Frame "grouped_single" generado en las líneas anteriores, obteniendo como resultado:

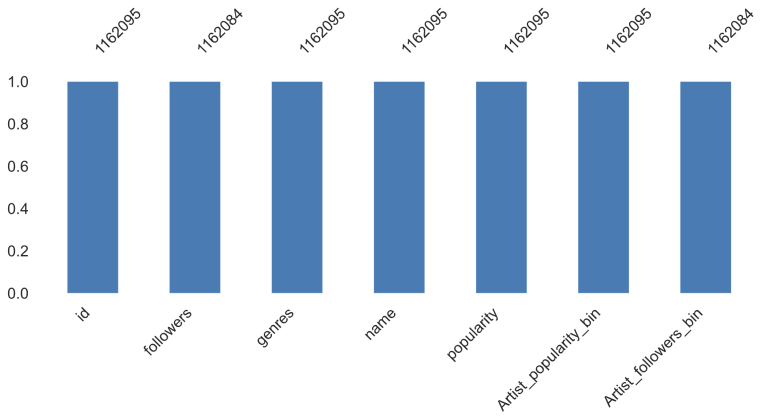


Continuamos con el tratamiento de los datos para calcular la **cantidad de usuarios por género** , la función "**.value counts()**", esta cuenta la frecuencia de cada valor en la columna "genres" del DataFrame "artists_mod" y devuelve una serie que contiene la cuenta de cada valor dada la cantidad de datos nulos, que se removerán, nos apoyamos con la función "**.LOC**" que permite acceder a un grupo de filas y columnas por etiqueta(s) o una matriz booleana, esta función filtra el DataFrame "artists_mod" para incluir solo las filas en las que la columna "genres" que no este vacía , posteriormente llamamos al nueva Data frame para observar los resultados. Incluimos la función **artists_mod["genres"].value_counts()** La diferencia con la línea de código anterior es que ahora se está trabajando en el DataFrame "**artists_mod_gr**" que previamente se había filtrado para eliminar las filas en las que la columna "genres" que estaba vacía, de esta forma, se obtiene la cuenta de géneros musicales solo para los artistas que tienen valores en la columna "genres".

- *Artist_genres["Quantity_Genres"]= pd.cut(Artist_genres["genres"],9, labels=["Nv_1","Nv_2","Nv_3","Nv_4","Nv_5","Nv_6","Nv_7","Nv_8","Nv_9"])*
-

En esta función se divide los valores de la columna "genres" del DataFrame "Artist_genres" en nueve intervalos iguales utilizando la función "**cut()**" de Pandas. Nuevamente con la función "cut()" clasificamos los valores en una columna en intervalos de tamaño fijo. Dicidimos "genres" en nueve intervalos iguales.Se crea una nueva columna "Quantity_Genres" en el DataFrame "Artist_genres" que contiene las etiquetas correspondientes a cada intervalo de la columna "genres".
En resumen, esta línea de código crea una nueva columna en el DataFrame "Artist_genres" que clasifica los géneros musicales según la cantidad de artistas que los interpretan en nueve niveles diferentes. Más información de este tratamiento se puede evidenciar en la documentación con la implementación de las funciones "**.sum()**" , "**groupby()**", para agrupar los datos del DataFrame "Artist_genres" por los niveles de "Quantity_Genres" y "**agg()**" es utilizada para aplicar diferentes funciones estadísticas.

Para el apartado de FOLLOWERS utilizamos la función "**ProfileReport()**" éste genera un informe detallado con estadísticas y visualizaciones sobre el DataFrame, que incluyen información sobre variables categóricas, variables numéricas, correlaciones, valores perdidos y otros datos relevantes.
(para ver todos los resultados remitirse a la documentación)




**followers**

Real number (ℝ)

HIGH CORRELATION SKEWED ZEROS

| | | | |
|---|---|---|---|
| **Distinct** | 51998 | **Minimum** | 0 |
| **Distinct (%)** | 4.5% | **Maximum** | 78900234 |
| **Missing** | 11 | **Zeros** | 70924 |
| **Missing (%)** | < 0.1% | **Zeros (%)** | 6.1% |
| **Infinite** | 0 | **Negative** | 0 |
| **Infinite (%)** | 0.0% | **Negative (%)** | 0.0% |
| **Mean** | 10220.704 | **Memory size** | 8.9 MiB |

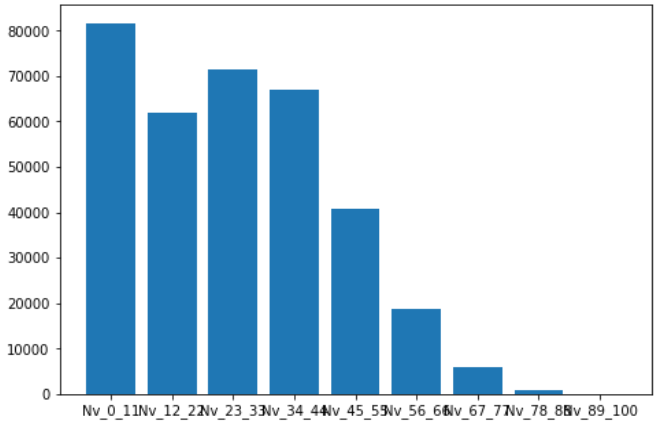
En la exploratoria por canciones el tratamiento se realiza de forma similar a la ya planteada , comando como "**head()**" del Data Frame "tracks_mod" para mostrar las primeras 6 filas del Data Frame en la consola, "**.shape**" que devuelve una tupla que indica el número de filas y columnas que tiene el Data Frame, "**tracks_mod.isna().sum()**" que devuelve la suma de valores nulos (NaN) por columna en un DataFrame llamado tracks_mod,es decir, para cada columna en tracks_mod, devuelve la cantidad de valores nulos presentes en ella,el resultado es un índice con los nombres de las columnas en "tracks_mod" y los valores son las sumas de los valores nulos, "**tracks_mod.popularity.max()**" devuelve el valor máximo de la columna 'popularity' en el Data Frame 'tracks_mod', En otras palabras, devuelve el valor más alto de popularidad entre todas las canciones en el conjunto de datos 'tracks_mod'.

- *tracks_mod["Popularity_Song_bins"]= pd.cut(tracks_mod["popularity"],9, labels=["Nv_0_11","Nv_12_22","Nv_23_33","Nv_34_44","Nv_45_55","Nv_56_66","Nv_67_77","Nv_78_88","Nv_89_100"])*

En esta línea se está creando una nueva columna en el Data Frame tracks_mod llamada "**Popularity_Song_bins**", que categoriza la popularidad de las canciones en 9 niveles diferentes, #especificados mediante los límites del rango de popularidad y etiquetas de las categorías. Los límites y etiquetas se pasan como argumentos a la función "**pd.cut()**".La popularidad de cada canción se toma de la columna "popularity" del Data Frame tracks_mod.

- *import matplotlib.pyplot as plt*

Este gráfico muestra el recuento de artistas en cada categoría de popularidad en el Data Frame "grouped_songs" generado en las líneas anteriores




La creación de la base de datos se realizó teniendo en cuenta los resultados obtenidos en la entrega número 1 , de forma que sabemos de antemano que los datos se encuentran normalizados , las 3 tablas escogidas fueron :

```
CREATE TABLE artistas(id VARCHAR(255),
CREATE TABLE generos(id SERIAL,name VARCHAR(255));
CREATE TABLE canciones(id VARCHAR(255)...
```

esto teniendo en cuenta los requerimientos del taller ,la información se puede ver de forma más detallada en los archivos adjuntos a este trabajo , allí se incluyen los datos de los

archivos de datos originales, Se importan los archivos "artists_mod" y "tracks_mod" con los cuales se crearán las tablas relacionadas y luego se visualizan los DataFrame creados con los cuales se alimentan las tablas relacionadas.

Para continuar se hace un proceso de conversión de los datosa SQL de la siguiente manera:

```
data1.to_sql('artistas', conn, if_exists= 'replace')
data2.to_sql('generos', conn, if_exists= 'replace')
data3.to_sql('canciones', conn, if_exists= 'replace')
```

Para luego recorrer las tablas e imprimir para poder verificar el guardado de la información.

```
sql1='''select * from artistas;'''
cursor.execute(sql1)
for i in cursor.fetchall():
   print(i)
 conn1.commit()
conn1.close()
```

los datos se pueden visualizar en la siguiente imagen que nos confirma el correcto funcionamiento y operación de la base de datos:

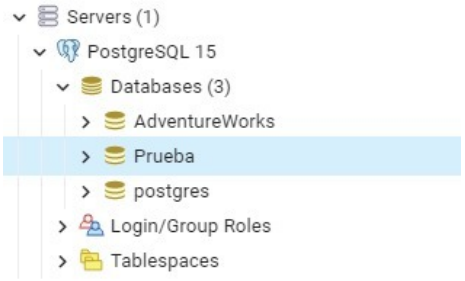



Figura 1 . creación de las Base de datos en Postgres.

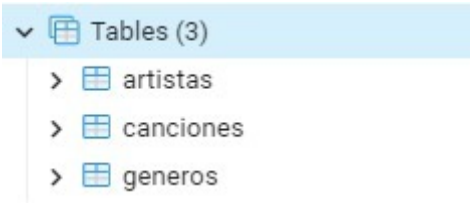



Figura 2 . creación de las tablas en Postgres

Realizamos una consulta sencilla para validar los datos obtenidos en nuestra base de datos, para cada una de las 3 tablas :

```
1  SELECT * FROM public.artistas
2
```

Data Output | Messages | Notifications

| | index bigint | id text | followers double precision | name text | popularity bigint |
|---|---|---|---|---|---|
| 1 | 0 | 0DheY5irMjBUeLybbCUEZ2 | 0 | Armid & Amir Zare Pashai feat. Sara Rouzbehani | 0 |
| 2 | 1 | 0DIhY15l3wsrnlfGio2bjU | 5 | ปูนา ภาวิณี | 0 |
| 3 | 2 | 0DmRESX2JknGPQyO15yx... | 0 | Sadaa | 0 |
| 4 | 3 | 0DmhnbHjm1qw6NCYPeZ... | 0 | Tra'gruda | 0 |
| 5 | 4 | 0Dn11fWM7vHQ3rinvWEI4E | 2 | Ioannis Panoutsopoulos | 0 |
| 6 | 5 | 0DotfDIYMGqkbzfBhcA5r6 | 7 | Astral Affect | 0 |

Figura 3 . Tabla de artistas

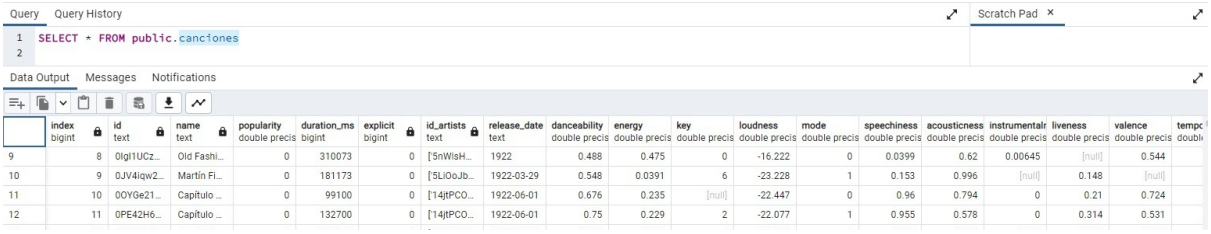



Figura 4 . Tabla de canciones


Figura 5 . Tabla de Géneros.

Teniendo en cuenta estos valores procedemos con la creación de la bodega de datos para posteriormente crear la ETL , La información detallada se encuentra en los anexos de este trabajo.

1) Una bodega de datos (p.e. BigQuery) que contenga los datos lo más preparados posible y sin ningún tipo de normalización para realizar el futuro entrenamiento del modelo de recomendación.

El archivo Json que nos arroja Bigquery con credenciales, deberá ser colocado en una carpeta dentro de nuestros archivos de drive y de allí se llamara invocándolo desde la ruta donde se encuentra.

```
credentials = service
account.Credentials.from_service_account_file("/content/drive/MyDrive/G
estion_de_Datos/Entrega_Proyecto/datos_clave.json",
scopes=["https://www.googleapis.com/auth/cloud-platform"])
client = bigquery.Client(credentials=credentials,
project=credentials.project_id)
```

El proceso de creación de la ETL  es el siguiente :

```
# Configuramos los campos que tendrá la tabla en BQ.
job_config_artist = bigquery.LoadJobConfig(
   schema=[
       # Supported data types:
https://cloud.google.com/bigquery/docs/reference/standard-sql/data-types
       bigquery.SchemaField("id", bigquery.enums.SqlTypeNames.STRING),
       bigquery.SchemaField("followers",
bigquery.enums.SqlTypeNames.FLOAT64),
       bigquery.SchemaField("genres", bigquery.enums.SqlTypeNames.STRING),
       bigquery.SchemaField("name", bigquery.enums.SqlTypeNames.STRING),
       bigquery.SchemaField("popularity",
bigquery.enums.SqlTypeNames.FLOAT64)
   ],
   # Drod and re-create table, if exist
   write_disposition="WRITE_TRUNCATE"
)
```

```
# Verificamos si el proceso de subida de la tabla a nuestro espacio fue
exitoso.
table = client.get_table(BQ_TABLE_NAME_artist)
print("Loaded {} rows and {} columns to {}".format(table.num_rows,
len(table.schema), BQ_TABLE_NAME_artist))
# Configuramos los campos que tendrá la tabla en BQ.

job_config_tracks = bigquery.LoadJobConfig(
   schema=[
       # Supported data types:
https://cloud.google.com/bigquery/docs/reference/standard-sql/data-types
       bigquery.SchemaField("id", bigquery.enums.SqlTypeNames.STRING),
       bigquery.SchemaField("name", bigquery.enums.SqlTypeNames.STRING),
       bigquery.SchemaField("popularity",
bigquery.enums.SqlTypeNames.FLOAT64),
       bigquery.SchemaField("duration_ms",
bigquery.enums.SqlTypeNames.FLOAT64),
       bigquery.SchemaField("explicit", bigquery.enums.SqlTypeNames.FLOAT64)
       ,bigquery.SchemaField("artists", bigquery.enums.SqlTypeNames.STRING)
```

```
        ,bigquery.SchemaField("id_artists",
bigquery.enums.SqlTypeNames.STRING)
        ,bigquery.SchemaField("release_date",
bigquery.enums.SqlTypeNames.STRING)
        ,bigquery.SchemaField("danceability",
bigquery.enums.SqlTypeNames.FLOAT64)
        ,bigquery.SchemaField("energy", bigquery.enums.SqlTypeNames.FLOAT64)
        ,bigquery.SchemaField("key", bigquery.enums.SqlTypeNames.FLOAT64)
        ,bigquery.SchemaField("loudness", bigquery.enums.SqlTypeNames.FLOAT64)
        ,bigquery.SchemaField("mode", bigquery.enums.SqlTypeNames.FLOAT64)
        ,bigquery.SchemaField("speechiness",
bigquery.enums.SqlTypeNames.FLOAT64)
        ,bigquery.SchemaField("acousticness",
bigquery.enums.SqlTypeNames.FLOAT64)
        ,bigquery.SchemaField("instrumentalness",
bigquery.enums.SqlTypeNames.FLOAT64)
        ,bigquery.SchemaField("liveness", bigquery.enums.SqlTypeNames.FLOAT64)
        ,bigquery.SchemaField("valence", bigquery.enums.SqlTypeNames.FLOAT64)
        ,bigquery.SchemaField("tempo", bigquery.enums.SqlTypeNames.FLOAT64)
        ,bigquery.SchemaField("time_signature",
bigquery.enums.SqlTypeNames.FLOAT64)
    ],
    # Drod and re-create table, if exist
    write_disposition="WRITE_TRUNCATE"
)
```

Arquitectura general de la solución.

Estructura de la solución:

-Se llama la tabla Artist y tracks del espacio de Bigquery.

Después de eso se toma una muestra del 6 % del registro de las canciones pues ya que no se tiene el poder computacional de hacer una recomendación para todos los temas que aparecen en esa tabla.

```
tracks_sample = tracks.sample(frac=0.06
                              ,replace=False, random_state=13)
```

```
[ ] sql = """
        SELECT * FROM proyecto-gestion-datos.tablas_proyecto.artists_mod
    """
    artists = client.query(sql).to_dataframe()
    artists.head(1)
```

Una vez se calcula la muestra, se hace un left join con la tabla Artist para ver si se puede usar alguna característica de esa otra tabla.

Después de haber hecho el L.J., se demuestra que ninguna característica posible a usar de artist hace join con tracks ... Es por esta razón que el Algoritmo solo se va a desarrollar con información de la tabla tracks

```
DTF = tracks_sample.merge(artists.loc[:,["id","genres"]], on='id', how='left').reset_index()
DTF
```

| | index | id | name | popularity | duration_ms | explicit | artists | id_artists | rel |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0p8MBpeLLfadRVBgkrvrPl | Nothing's Going to Happen | 22.0 | 236600.0 | 0.0 | Tall Dwarfs | 4DKgySPRqtTAw0eUMmp3O9 | |
| 1 | 1 | 3BInmwpVjbcr3XyW6qGZ2Q | Cry For You - Radio Mix | 48.0 | 208347.0 | 0.0 | September | 6VX2R9L0O0d6qPvqGuIH7b | |
| 2 | 2 | 5nfw7xHh5nD2CVoLX0vQpX | Summer Beauty 1990 / ラテンでレッツ・ラブまたは1990サマー・ビューテ... | 23.0 | 282627.0 | 0.0 | "Flippers Guitar" | 5fkHPXDNaqz3GrYYhO8APB | |
| 3 | 3 | 61fmdWPvyQcv122r8TNasO | Hey Joey | 0.0 | 229135.0 | 1.0 | The Infinity crew | 0l8Jcz13eQsyAmgHIbxDSc | |

Dado la estructura de la data tracks, se va a usa un sistema de recomendación denominado "Item Recomendation Filter".

```
TRACKS_TRAIN = DTF.loc[:,['name', 'popularity', 'duration_ms', 'explicit', 'artists', 'danceability', 'energy', 'key',
       'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness',
       'liveness', 'valence', 'tempo', 'time_signature',"Song_popularity_bin"]]

one_hot_song_popularity = pd.get_dummies(TRACKS_TRAIN['Song_popularity_bin'])
TRACKS_TRAIN = TRACKS_TRAIN.drop('Song_popularity_bin',axis = 1)
TRACKS_TRAIN= TRACKS_TRAIN.join(one_hot_song_popularity)
TRACKS_TRAIN.tail(1)
```

El Algoritmo va a estar alimentado por todas las variables numéricas de esa tabla, como lo son , Nombre , Popularidad, Duración , si es explícito el contenido o no , artista , entre los demás que observamos en la figura anterior.

Este modelo es un cálculo de distancia de mahalanobis entre todas las canciones de esa muestra. Se decidió usar esta distancia ya que tiene en cuenta la correlación lineal entre estás variables.

Se programa el sistema de recomendación como una Clase, con los siguientes parámetros:

- **Df** : Tabla con data de entrenamiento y nombre de las canciones y artista asociado.
- **Metric** : Metodología con la cual se va a calcular la similaridad entre canciones.
- **Song** : Canción a Evaluar.
- **Top** : El top de canciones mas similares a la canción evaluada.

```
class Recomendation_engine:
  def __init__(self, df, metric="mahalanobis"):
    self.df = df
    self.metric = metric

  def similarity(self):

    df = self.df

    DT_TRAIN = df.loc[:,[ 'popularity', 'duration_ms', 'explicit', 'danceability', 'energy', 'key',
```

La recomendación que va a generar para cada canción, son las 5 canciones más cercanas en cuanto a distancia de mahalanobis con respecto a ella .

Para desplegar los resultados del modelo, se crea una tabla en bigquery que tiene dichos resultados para todas las canciones de la muestra, el despliegue por detrás es una consulta a esta tabla teniendo en cuenta que este proceso se le conoce como tipo  BATCH.

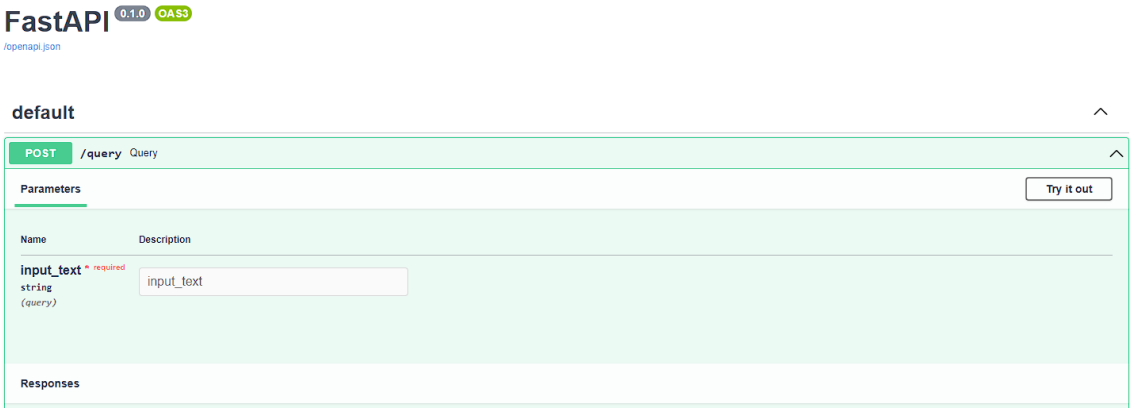
Dado que el objetivo es que un usuario pueda usar estos resultados desde cualquier lugar, se hace el montaje de dicha consulta en un servicio de Google cloud llamado App Engine.


Al realizar dicho proceso ,este servicio nos va a generar una URL.

URL : https://proyecto-gestion-datos.uc.r.appspot.com/docs#/default/query_query_post

Sin embargo, el front para poder hacer uso de dichos resultados es una propiedad que se invoca con el siguiente comando en la URL (/docs)

Para que el usuario pueda dar uso, debe darle click en el botón TRY OUT:

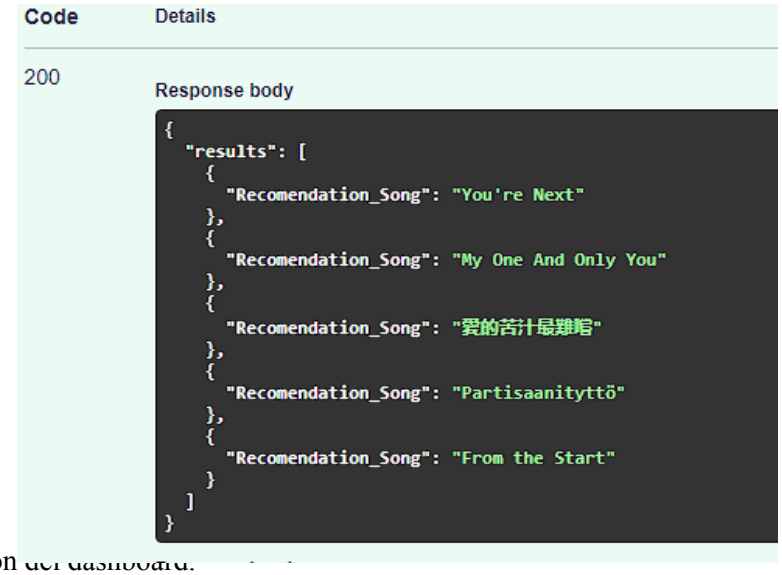



Y escribir el nombre de la canción en la caja de texto input_text

Aqui un posible listado para utilizar el programa:

- Lahaina Luna,
- One More Try
- Juice
- Monoi Mas Meiname
- Ring
- Yeke Yeke
- Verliefd

Una vez el usuario coloca el nombre de la canción se recibirá la recomendación en la caja llamada Response body como se muestra a continuación:




3. Descripción del dashboard.

A continuación, en la siguiente parte del trabajo crea un dashboard utilizando una herramienta de visualización de datos Python se crea el dashboard utilizando los siguientes KPIs (número total de canciones, artistas en la plataforma, cantidad de reproducciones por canción o artista y los

artistas o géneros más populares) y los datos identificados anteriormente. El dashboard que se generó es fácil de usar y navegar, con filtros y opciones de personalización para que el consumidor obtenga información específica que se necesita este fue generado en Jupyter Notebook en lenguaje Python3.

El primer paso a seguir es el llamado de las librerías que nos permitan generar un dashboard o gráficos interactivos de alta calidad, esto gracias a la librería "`plotly.graph_objs`" a continuación se hace el llamado de la librería "`psycopg2`" con la cual interactuamos con una base de datos PostgreSQL para manejar la base de datos desde Python y así poder tener todas las funciones de SQL y por último realizamos el llamado de la librería "`from sqlalchemy import create_engine`" que nos permite generar conexión con las bases de datos SQL alojadas en PgAdmin 4 mediante una URL de conexión.

```
import plotly.graph_objs as go
import psycopg2
from sqlalchemy import create_engine
```

En estas líneas de código, primero se establece una conexión a una base de datos PostgreSQL utilizando la biblioteca psycopg2. Se utiliza la función psycopg2.connect() para establecer la conexión y se proporcionan los detalles necesarios de la base de datos, como el nombre de la base de datos, el nombre de usuario, la contraseña, el host y el puerto.

```
# Conectarse a la base de datos
conn = psycopg2.connect(
                        database="Prueba",
                        user='postgres',
                        password='Leo1993+',
                        host='127.0.0.1',
                        port= '5432'
)

# Obtener los datos necesarios de la base de datos
cursor = conn.cursor()

# Total de canciones
cursor.execute("SELECT COUNT(*) FROM canciones")
total_canciones = cursor.fetchone()[0]

# Total de artistas
cursor.execute("SELECT COUNT(*) FROM artistas")
total_artistas = cursor.fetchone()[0]
```

Luego de generar conexión se realizan dos consultas en la base de datos utilizando el cursor. La primera consulta obtiene el recuento total de canciones en la tabla "canciones" y la segunda consulta obtiene el recuento total de artistas en la tabla "artistas". Estos recuentos se guardan en las variables total_canciones y total_artistas.

En pocas palabras, estos pasos nos permiten conectarnos a una base de datos PostgreSQL y obtener información específica de las tablas en esa base de datos.

Él pasó a seguir, de luego de generar las consultas en las bases de datos es seleccionar la columna que se va a trabajar, por lo cual se trabaja con el “id_artists” de la tabla "canciones" y se cuenta el número de veces que cada artista aparece en la tabla. Luego se agrupa y se ordenan los resultados por la cantidad de reproducciones en orden descendente, el cual se limita a 10 artistas más populares.

De igual manera se trabaja con los géneros más populares, se extrae la información de la columna “genres” y sé visualización los 10 géneros más populares.

```
# Artistas más populares
cursor.execute("""
    SELECT c.id_artists, COUNT(*) AS reproducciones
    FROM canciones c
    GROUP BY c.id_artists
    ORDER BY reproducciones DESC
    LIMIT 10
""")
artistas_populares = cursor.fetchall()

# Géneros más populares
cursor.execute("""

    SELECT d.genres, COUNT(*) AS reproducciones
    FROM generos d
    GROUP BY d.genres
    ORDER BY reproducciones DESC
    LIMIT 10

""")
generos_populares = cursor.fetchall()
```

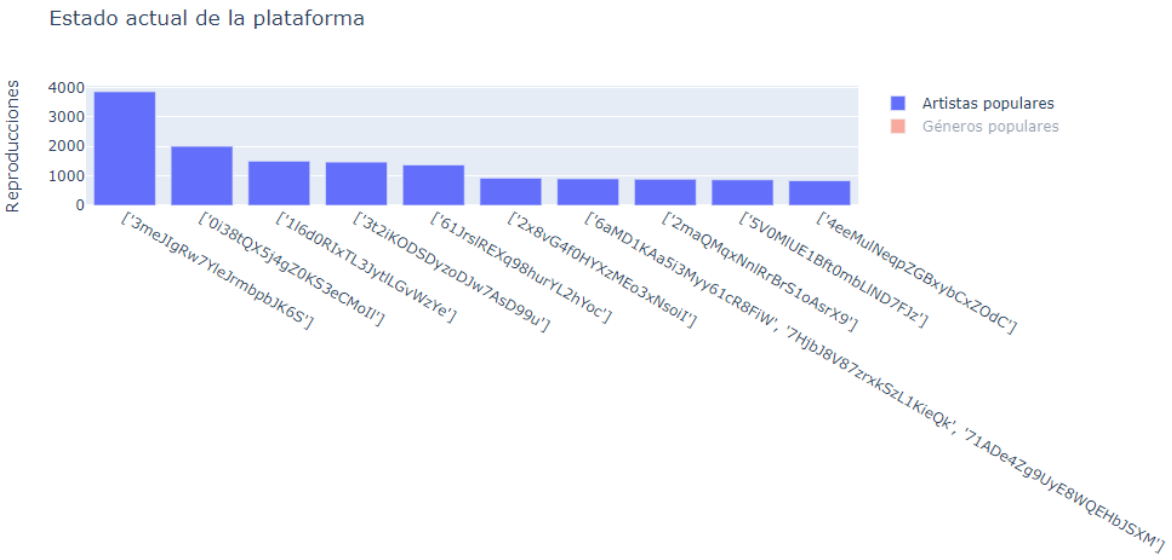
Y por último en el siguiente código se crean dos gráficos. Se utiliza el tipo de gráfico de barras (go.Bar()) para representar los artistas y géneros más populares. Se proporcionan las etiquetas para el eje x y las alturas de las barras para el eje y utilizando las listas generadas a partir de los resultados de las consultas a la base de datos (artistas_populares y generos_populares).

Luego se crea un objeto dashboard utilizando go.Figure() que contiene los gráficos de los artistas y géneros más populares y se establece el título del gráfico y las etiquetas.

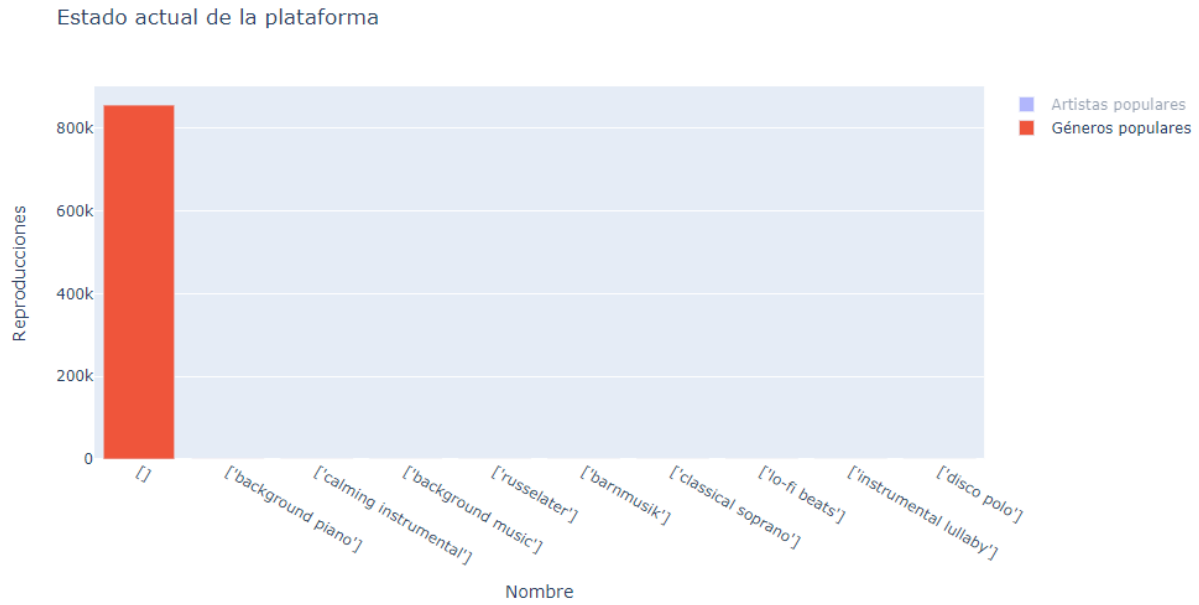
Finalmente, se muestra el gráfico en la linea “dashboard.show()” que abre una nueva pestaña en el navegador con el gráfico.

```python
# Crear las gráficas
grafica_artistas = go.Bar(
    x=[row[0] for row in artistas_populares],
    y=[row[1] for row in artistas_populares],
    name="Artistas populares"
)

grafica_generos = go.Bar(
    x=[row[0] for row in generos_populares],
    y=[row[1] for row in generos_populares],
    name="Géneros populares"
)

# Crear el dashboard
dashboard = go.Figure(
    data=[grafica_artistas, grafica_generos],
    layout=go.Layout(
        title="Estado actual de la plataforma",
        xaxis=dict(title="Nombre"),
        yaxis=dict(title="Reproducciones")
    )
)

# Mostrar el dashboard
dashboard.show()
```

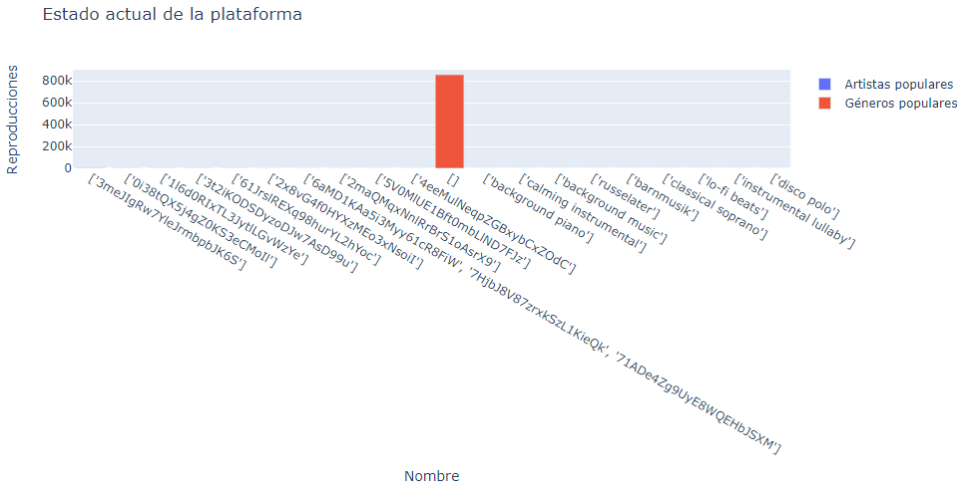
A continuación se muestra el resultado del codigo ejecutado para la visualización del dashboard, en él podemos observar como se encuentra la base de datos actualmente con respecto a los artistas más populares, en el eje y observamos el numero de reproducciones y en el eje x id del artista mas popular de izquierda a derecha.



Ahora visualizaremos los géneros más populares que encontramos en la base de datos actual, podemos observar que como primer item tenemos que la mayoría de canciones no se encuentran relacionadas con un tema en específico, por lo tanto, nos arroja estos resultados, pero se sigue observando que tenemos 9 géneros adicionales en los cuales si tenemos un orden de izquierda a derecho y se puede deducir su popularidad.




Este dashboard es interactivo y podemos acercarnos, alejarnos y extraer información de los artistas y géneros más populares actuales en la base de datos que se está trabajando actualmente.



Conclusiones

1. La limpieza y preprocesamiento de los datos son esenciales para garantizar que los resultados sean precisos y confiables. Es importante verificar la calidad y coherencia de los datos antes de utilizarlos en un análisis.
2. La elección adecuada de las herramientas y técnicas de análisis de datos es fundamental para obtener información valiosa. Dependiendo de los objetivos del análisis, se pueden utilizar diferentes métodos estadísticos, de visualización, de aprendizaje automático, entre otros.
3. Los resultados del análisis deben ser presentados de manera clara y accesible para que los usuarios puedan comprender fácilmente la información y tomar decisiones informadas en base a ella.
4. El análisis de datos puede ser útil para identificar patrones y tendencias, así como para descubrir nuevas oportunidades o problemas en un conjunto de datos. Sin embargo, también tiene limitaciones y es importante tener en cuenta el contexto en el que se recopilaron los datos y las limitaciones del análisis en sí mismo.
5. Ahora por último al realizar un agrupamiento de la popularidad de canciones, se pueden extraer valores como las canciones que menos suenan  y por otro las más exitosas, gracias a la función describe() nos permite observar la media , desviación estándar, los valores mínimos, maximos, quartiles y conteos de cada cancion tanto en exitosos como en no exitosas.

Anexos

- Código Implementado:

    - https://drive.google.com/file/d/1fi_iwNstciChPThh-FiP3R1gH_3RK-Uw/view?usp=sharing

    - https://colab.research.google.com/drive/10_zHfqOoUF2u26YaR9qN1E0CTpNe2M0O#scrollTo=x6Rt5sRD-KHY