UNIVERSIDAD AUTONOMA DE OCCIDENTE

MATERIA:

ETL

DOCENTE:

JAVIER ALEJANDO VERGARA ZORRILLA

ESTUDIANTE:

ANTONIO CARDENAS

CODIGO:

2230433

SEMESTRE:

5to

AÑO:

2025

INTRODUCCION

DESCRIPCIÓN GENERAL DEL PROYECTO:

Este proyecto consiste en la construcción de un ETL pipeline utilizando Apache Ariflow para automatizar la extracción, transformación y carga de un conjunto de datos provenientes de 3 diferentes recursos:

1. *Spotify Dataset (CSV).*
2. *Gammy Awards Dataset (PostgreSQL – Database)*
3. *Spotify API*

El objetivo es limpiar, procesar y combinar estos conjuntos de datos, almacenar los datos finales en PostgreSQL y generar visualizaciones interactivas con Tableau para analizar tendencias en la popularidad musical, patrones de premios y características de las canciones.

FLUJO DE TRABAJO DEL PROYECTO

El proceso ETL sigue la sgt estructura:

1. **Extracción:** Recolectar datos de tres fuentes (CSV de Spotify, base de datos de los Grammy, API de Spotify).
2. **Transformación:** Limpiar, normalizar e integrar los conjuntos de datos (manejo de valores nulos, duplicados y estandarización de formatos).
3. **Carga:** Almacenar los datos procesados en una base de datos PostgreSQL y exportar un archivo CSV a Google Drive**.**
4. **Visualización:** Generar insights mediante un dashboard en Power BI, analizando tendencias musicales, características de canciones ganadoras y el éxito de los artistas a lo largo del tiempo.

Para garantizar la automatización y eficiencia, se utiliza Apache Airflow para programar y gestionar el pipeline ETL dentro de un entorno Dockerizado.

Conjuntos de Datos Utilizados

1. **Dataset de Spotify (CSV)**

        - **Fuente:** Spotify Tracks Dataset
    - **Descripción: Contiene metadatos y características de audio de canciones, incluyendo:**
        - **track_name, artist_name, popularity, danceability, energy, tempo, entre otros.**

2. **Dataset de los Premios Grammy (Base de datos PostgreSQL)**
    - **Fuente: Dataset de los Grammy**
    - **Descripción: Incluye datos históricos de nominaciones y ganadores desde 1958 hasta 2019:**
        - **year, category, artist, winner (campo booleano).**
3. **API de Spotify for Developers**
    - **Fuente:** Spotify for Developers API
    - **Descripción:** Permite acceder a datos musicales en tiempo real y de forma detallada, incluyendo:
        - Información sobre canciones populares (top-tracks),
        - Artistas destacados,
        - Álbumes más reproducidos,
        - Popularidad, duración, fechas de lanzamiento y si el contenido es explícito.
    - **Uso:** Enriquecer el dataset agregando datos reales desde la plataforma Spotify, permitiendo comparaciones entre artistas populares, ganadores del Grammy y su comportamiento en plataformas de streaming. Además, estos datos se combinan con los datasets locales para generar un análisis más completo y actualizado.

TECNOLOGÍAS Y HERRAMIENTAS

Para manejar eficazmente la extracción, transformación, almacenamiento y visualización de datos, se usan las siguientes herramientas:

1. PIPELINE ETL Y ORQUESTACIÓN

- Apache Airflow: Gestiona y programa los workflows de ETL.
- Docker: Ejecuta Airflow en un entorno contenedor para facilitar el despliegue.

2. PROCESAMIENTO Y TRANSFORMACIÓN DE DATOS

- Python: Lenguaje principal para scripting y automatización.
- Polars: Librería de alto rendimiento para manipulación de datos.
- Pandas: Análisis y transformación de datos.
- Seaborn: Gráficas estadísticas atractivas.
- Matplotlib: Visualizaciones estáticas de datos.

3. BASE DE DATOS Y ALMACENAMIENTO

- **PostgreSQL:** Base de datos elegida para almacenar los datos procesados.
- **psycopg2:** Librería de conexión entre Python y PostgreSQL
- **Google Drive API:** Guarda el dataset final como CSV en Google Drive.

4. VISUALIZACIÓN DE DATOS

- Dashboard en Power BI.

ESTRUCTURA DEL PROYECTO:

```
WORKSHOP_02_ANTONIO/
│
├── client_secrets.json            # Credenciales OAuth para Google Drive
├── docker-compose.yml             # Configuración de servicios para Docker
├── Dockerfile                     # Imagen personalizada para Airflow
├── .env                           # Variables de entorno (credenciales, paths, etc.)
├── .gitignore                     # Exclusión de archivos en control de versiones
├── requirements.txt               # Dependencias del entorno Python
├── README.md                      # Documentación principal del proyecto
│
├── credentials/
│   └── Client_credentials.py      # Script para autenticación OAuth
│
├── dags/
```

```
│   ├── __pycache__/
│   └── etl_pipeline.py            # DAG de Airflow que orquesta el proceso ETL
│
├── data/
│   └── (puede estar vinculado a rutas simbólicas o data intermedia)
│
├── Data/
│   ├── final/
│   │   ├── .gitkeep
│   │   └── merged_music_data.csv    # Dataset final integrado
│   │
│   ├── processed/
│   │   ├── .gitkeep
│   │   ├── api_spotify_transformed.csv
│   │   ├── grammy_transformed.csv
│   │   ├── grammys_transformed.csv
│   │   └── spotify_transformed.csv
│   │
│   ├── raw/
│   │   ├── .gitkeep
│   │   ├── datos_api_spotify_expandido.csv
│   │   ├── grammy_awards.full.csv
│   │   ├── spotify_dataset.csv
│   │   ├── spotify_dataset_clean.csv
│   │   ├── the_grammy_awards.csv
│   │   └── the_grammy_awards_clean.csv
│
├── logs/                          # Logs de ejecución de Airflow
```

```
│   └── .gitkeep
│
├── notebooks/
│   ├── EDA_API.ipynb              # Análisis exploratorio del dataset de la API
│   ├── EDA_GRAMMY.ipynb           # Análisis exploratorio de los datos de los
Grammy
│   ├── EDA_SPOTIFY.ipynb          # Análisis exploratorio de los datos de Spotify
│   └── MERGE.ipynb                # Notebook para unión final de datasets
│   ├── merged_music_data.csv      # Versión intermedia del dataset combinado
│   └── merged_music_data_clean.csv  # Versión limpia del dataset combinado
│
├── plugins/                       # Plugins personalizados para Airflow (vacío o en uso futuro)
│
├── src/                           # Código fuente del proceso ETL
│   ├── extract/
│   │   ├── __pycache__/
│   │   ├── Data/                  # Posible carpeta para manejar rutas específicas
│   │   └── extract.py             # Funciones para extraer datos de cada fuente
│   │
│   ├── load/
│   │   ├── __pycache__/
│   │   ├── load_to_postgress.py   # Carga de datos a PostgreSQL
│   │   └── upload_to_drive.py     # Subida del dataset final a Google Drive
│   │
│   ├── merge_data/
│   │   ├── __pycache__/
│   │   └── merge.py               # Unión de los datasets transformados
│   │
│   └── transform/
```

│ └── transform.py

ANALISIS EXPLORATORIO DE DATOS

EDA_SPOTIFY:

Estructura General del Dataset

El dataset analizado contiene 114,000 filas y 21 columnas, lo que proporciona una base amplia y robusta para estudiar fenómenos musicales. La estructura de los datos es adecuada tanto para análisis estadísticos como para la aplicación de modelos de aprendizaje automático.

Tipos de Datos:

Las variables se agrupan en:

- Numéricas continuas (float64): como danceability, energy, tempo.
- Categóricas (object): como track_id, artists, track_name, track_genre.
- Discretas enteras (int64): como popularity, key, mode, time_signature.
- Booleanas (bool): como explicit.

Revisión de Calidad Inicial

- No se encontraron valores nulos ni duplicados, lo que indica un preprocesamiento limpio.
- La columna Unnamed: 0 es un índice generado automáticamente y puede eliminarse sin afectar el análisis.

Estadísticas Descriptivas y Distribuciones

El resumen estadístico revela comportamientos clave:

- popularity: distribución casi simétrica, pero ligeramente sesgada positivamente.
- duration_ms: media de 3.8 minutos; se detectó un valor extremo (5.2M ms) como probable outlier.
- danceability, energy, valence: comportamiento razonablemente normal.
- loudness: fuerte asimetría negativa.

- speechiness, instrumentalness, acousticness: alta asimetría positiva, sugieren valores extremos y presencia de música no vocal o hablada.
- tempo: valores normales, media en 122 BPM.
- time_signature: predomina el compás de 4/4, con algunas variaciones menores.

Las variables duration_ms, speechiness, instrumentalness y acousticness presentan valores extremos o alta asimetría, lo que sugiere la necesidad de transformación o normalización en modelos predictivos.

Artistas y Géneros Más Frecuentes

- Artistas con más canciones: predominan nombres icónicos como The Beatles, George Jones y Stevie Wonder, lo que puede influir en variables como duración y popularidad.
- Géneros más comunes: todos los géneros más frecuentes tienen exactamente 1000 canciones, evidenciando que el dataset fue balanceado intencionalmente por género, ideal para modelos de clasificación sin sesgo por clase.

Distribución de Variables Relevantes

Distribución de Popularidad

- Alta concentración en valores bajos (sesgo a la izquierda).
- Picos secundarios en rangos 20–40 y 50–60 sugieren niveles intermedios de reconocimiento.
- Puede impactar modelos predictivos y estrategias de marketing segmentado.

Distribución de Speechiness

- Valores típicos cercanos a 0.05 (canciones cantadas).
- Valores altos (≈1) representan contenido hablado como podcasts o rap explícito.
- Útil para clasificar géneros líricos y hablados.

Distribución de Acousticness

- Predominancia de valores cercanos a 0 (producción electrónica).
- Pequeño repunte hacia 1 indica presencia de música acústica pura.
- Representa un nicho de mercado específico.

Distribución de Instrumentalness

- Valores bajos indican predominancia de voz o canto.
- Pico cerca de 1.0 revela presencia de pistas completamente instrumentales.
- Relevante para tareas de clasificación o detección de contenido sin voz.

EDA_GRAMMYS

Estructura General del Dataset

- El dataset contiene 114,000 filas y 21 columnas, representando una amplia muestra para el análisis musical. Incluye variables como:
- Numéricas continuas (float64): danceability, energy, tempo.
- Categóricas (object): track_id, track_name, artist, track_genre.
- Discretas enteras (int64): popularity, key, mode, time_signature.
- Booleanas: explicit.
- Calidad Inicial
- No se encontraron valores nulos ni duplicados.
- Se identificó una columna Unnamed: 0 generada automáticamente que puede eliminarse si no se requiere.

Estadísticas y Análisis Numérico

- Popularity: distribución simétrica (media ≈ 33), ligera asimetría positiva.
- Duration: media ≈ 3.8 minutos, presencia de un outlier (~87 minutos).
- Speechiness, Instrumentalness, Acousticness: fuerte asimetría positiva, lo cual implica presencia de canciones habladas o instrumentales.
- Tempo: media 122 BPM, rango razonable.
- Time Signature: predominan compases 4/4, pero hay variedad.

Conclusión

Las variables duration_ms, speechiness, instrumentalness y acousticness presentan outliers o asimetrías fuertes. La mayoría de las variables están listas para modelado predictivo.

Distribución de Popularidad y Géneros

Artistas con Más Canciones

- Destacan The Beatles, George Jones, Stevie Wonder.
- Influencia probable en métricas como duración y popularidad.
- Géneros Más Frecuentes
- Todos los géneros top tienen exactamente 1,000 canciones, lo que indica una construcción balanceada del dataset.
- Popularidad

- Distribución sesgada a la izquierda.
- Picos adicionales en rangos 20–40 y 50–60 sugieren una distribución bimodal.

Análisis de Variables Críticas

Speechiness

- La mayoría de las canciones tienen valores bajos (≈ 0.05), indicando contenido cantado.
- Valores altos representan podcasts, spoken word o rap.

Acousticness

- Valores bajos predominan, aunque hay un pico menor cerca del 1.0, indicando canciones acústicas puras.

Instrumentalness

- La mayoría de canciones tienen valores < 0.1, es decir, con voz.
- Pico en 1.0 sugiere presencia de música puramente instrumental.

Dataset de Premios Grammy

Estructura y Calidad

- Contiene 4,818 registros y 10 columnas.
- Las variables year, winner, img están bien tipadas.
- Variables como artist, workers, img presentan valores nulos relevantes (hasta un 45%).

Conclusión

- El dataset está limpio, sin duplicados, pero requiere enriquecimiento externo para campos faltantes. Fechas published_at y updated_at deben transformarse a formato datetime.

Análisis de Frecuencia y Premios

Artistas más premiados

- Various Artists (66 premios).
- Solistas como Aretha Franklin, U2, Beyoncé, Stevie Wonder, Ray Charles.
- Nominados frecuentes
- Obras como Requiem o Bridge Over Troubled Water.
- Productores como Robert Woods y Steven Epstein.

- Categorías con más premios
- Song of the Year, Record of the Year, Album of the Year encabezan la lista.
- Destacan también categorías técnicas.

Evolución temporal

- Aumento sostenido de premios desde 1959.
- El año 2019 presenta un pico atípico que debe revisarse.

EDA_API

Información General

El dataset extraído desde la API de Spotify contiene 149 registros y 8 columnas, todos sin valores nulos. Las columnas incluidas son:

- **track_name:** Nombre de la canción.
- **artist_name:** Nombre del artista o banda.
- **album_name:** Álbum al que pertenece la canción.
- **release_date:** Fecha de lanzamiento del sencillo o álbum.
- **duration_ms:** Duración en milisegundos.
- **popularity:** Nivel de popularidad en Spotify (escala de 0 a 100).
- **explicit:** Indicador booleano de si la canción contiene lenguaje explícito.
- **spotify_url:** Enlace directo a la canción o artista en Spotify.

Aunque la estructura es limpia y completa, se detectaron 2 registros duplicados, los cuales deben eliminarse para evitar sesgos en los análisis.

Observaciones Generales

Este dataset es útil para:

- Analizar tendencias de popularidad y duración.
- Filtrar canciones con contenido explícito.
- Evaluar patrones por artista y año de lanzamiento**.**

Se trata de un conjunto de datos enfocado y curado, apto para análisis exploratorio.

Distribución de la Popularidad

- La variable popularity presenta una distribución ligeramente asimétrica hacia la izquierda, con un pico concentrado entre los valores 50 y 60. Esto indica que la mayoría de las canciones tienen una popularidad moderadamente alta, aunque hay una proporción considerable de canciones con popularidad por debajo de 40 y muy pocas con valores superiores a 75.
- Este patrón sugiere que el dataset incluye tanto artistas reconocidos como emergentes.

Duración de las Canciones

- La duración (duration_ms) se distribuye normalmente entre 3.5 y 4 minutos, consistente con estándares comerciales de la industria musical. La mayoría de las canciones se sitúan entre 3 y 5 minutos, con muy pocos casos extremos fuera de este rango. Esto valida que los datos corresponden a lanzamientos orientados al consumo masivo.

Contenido Explícito

- De las 149 canciones, solo una presenta lenguaje explícito. Esto indica que la gran mayoría de canciones en el conjunto están destinadas a todo público, posiblemente influenciado por géneros como pop, indie o acústico, donde el uso de lenguaje fuerte es menos común.

Artistas con Mayor Presencia

- El artista más recurrente es Jason Mraz, con más de 20 canciones. Le siguen Boyce Avenue y Andrew Belle, todos del ámbito acústico o alternativo. Esto evidencia una posible concentración del dataset en torno a ciertos artistas que tienen múltiples versiones o grabaciones disponibles en Spotify. Si no se controla este factor, podría sesgar ligeramente los análisis.

Años de Lanzamiento

- La mayoría de las canciones fueron lanzadas entre 2010 y 2023, con un pico claro en 2019. Esto refleja un sesgo hacia producciones recientes, lo cual es coherente con la tendencia de Spotify a promover contenidos nuevos. También se identifican algunos lanzamientos entre 2000 y 2010, pero en menor cantidad.

EXTRACCION API AND CSV SOURCES

EXTRACTING SPOTIFY DATA FROM LOCAL CSV

```python
def extract_spotify_csv(**kwargs):
    """
    Carga el dataset limpio de Spotify desde archivo local y lo guarda como copia base.
    """
    df = pl.read_csv(f"{RAW_DIR}/spotify_dataset_clean.csv")
    df.write_csv(f"{RAW_DIR}/spotify_dataset.csv")
    print(" Extraído: spotify_dataset_clean.csv")
```

- **Funcion:** extract_spotify_csv(**kwargs)

- **Proposito:** Carga un archivo limpio de datos de Spotify almacenado localmente y lo guarda como una copia base de trabajo.

- **Cómo funciona:**
  - Lee el archivo spotify_dataset_clean.csv usando la librería Polars.
  - Crea una copia como spotify_dataset.csv en la carpeta /data/raw.
  - Imprime un mensaje de éxito.

EXTRACTING GRAMMY AWARDS DATA FROM LOCAL CSV

```python
def extract_grammy_csv(**kwargs):
    """
    Carga el dataset limpio de los Grammy desde archivo local y lo guarda como copia base.
    """
    df = pl.read_csv(f"{RAW_DIR}/the_grammy_awards_clean.csv")
    df.write_csv(f"{RAW_DIR}/grammy_awards_full.csv")
    print(" Extraído: the_grammy_awards_clean.csv")


# ----------------------------------------
```

- **Funcion: extract_grammy_csv(**kwargs)**

- **Proposito:** Carga el dataset limpio de los premios Grammy desde un archivo local y genera una copia base para el flujo de trabajo.

- **Cómo funciona:**
  - Lee el archivo the_grammy_awards_clean.csv.

- Guarda una copia como grammy_awards_full.csv en la carpeta /data/raw.
- Imprime un mensaje indicando que se completó la extracción**.**

EXTRACTING SPOTIFY FROM PUBLIC API

```python
def extract_spotify_api(**kwargs):
    """
    Extrae los top tracks de múltiples artistas populares desde la API de Spotify.
    Guarda el resultado en un CSV en 'data/raw/datos_api_spotify_expandido.csv'.
    """
    load_dotenv()
    token = os.getenv("SPOTIFY_TOKEN")

    if not token:
        print(" No se encontró SPOTIFY_TOKEN en el .env")
        return

    headers = {
        "Authorization": token
    }

    artist_ids = {
        "Ed Sheeran": "6eUKZXaKkcviH0Ku9w2n3V",
        "Drake": "3TVXtAsR1Inumwj472S9r4",
        "The Weeknd": "1Xyo4u8uXC1ZmMpatF05PJ",
        "Ariana Grande": "66CXWjxzNUsdJxJ2JdwvnR",
        "Taylor Swift": "06HL4z0CvFAxyc27GXpf02",
        "Coldplay": "4gzpq5DPGxSnKTe4SA8HAU",
        "Lady Gaga": "1HY2Jd0NmPuamShAr6KMms",
        "U2": "51Blml2LZPmy7TTiAg47vQ",
        "Beyoncé": "6vWDO969PvNqNYHIOW5v0m",
        "Stevie Wonder": "7guDJrEfX3qb6FEbdPA5qi"
    }

    all_tracks = []

    for artist_name, artist_id in artist_ids.items():
        url = f"https://api.spotify.com/v1/artists/{artist_id}/top-tracks?market=US"
        response = requests.get(url, headers=headers)

        if response.status_code == 200:
            tracks = response.json().get("tracks", [])
            for track in tracks:
                all_tracks.append({
                    "track_name": track["name"],
                    "artist_name": track["artists"][0]["name"],
                    "album_name": track["album"]["name"],
                    "release_date": track["album"]["release_date"],
                    "duration_ms": track["duration_ms"],
                    "popularity": track["popularity"],
                    "explicit": track["explicit"],
                    "spotify_url": track["external_urls"]["spotify"]
```

- **Funcion:** extract_spotify_api(**kwargs)

- **Proposito:** Extrae las canciones más populares de varios artistas famosos directamente desde la API de Spotify.

- **Cómo funciona:**
  - Carga el token de autenticación desde un archivo .env.
  - Define un diccionario con los IDs de artistas populares como Beyoncé, Ed Sheeran, The Weeknd, entre otros.
  - Realiza una solicitud HTTP GET a la API de Spotify para obtener sus canciones más populares.
  - Extrae campos relevantes: track_name, artist_name, album_name, release_date, duration_ms, popularity, explicit, spotify_url.
  - Almacena los datos en un archivo CSV: /data/raw/datos_api_spotify_expandido.csv.
  - Imprime mensajes según el éxito o falla de la solicitud.

FASE DE TRANSFORMACIÓN

```python
def normalize_text(value: str) -> str:
    if isinstance(value, str):
        value = value.lower()
        value = re.sub(r"[^a-z0-9\s]", "", value)
        value = re.sub(r"\s+", " ", value)
        return value.strip()
    return value
```

Proposito: Estandariza cadenas de texto convirtiéndolas a minúsculas, eliminando caracteres especiales y normalizando espacios.

Como funciona:

- Verifica si el valor es tipo str.
- Convierte a minúsculas.
- Usa expresiones regulares para eliminar todo lo que no sea letra, número o espacio.
- Elimina espacios extra y recorta los bordes.

Escenario de uso: Se utiliza para normalizar nombres de canciones, artistas y categorías antes de fusionar los datos o realizar comparaciones.

TRANSFORM_SPOTIFY

```python
def transform_spotify():
    df = pl.read_csv(f"{DATA_RAW}/spotify_dataset_clean.csv")
    df = df.rename({col: col.strip().lower().replace(" ", "_") for col in df.columns})

    df = df.with_columns([
        pl.col("track_name").map_elements(normalize_text, return_dtype=pl.Utf8),
        pl.col("artists").map_elements(normalize_text, return_dtype=pl.Utf8)
    ])

    df = df.select([
        pl.col("track_name").alias("song_name"),
        pl.col("artists").alias("artist"),
        "track_genre", "popularity", "explicit",
        "tempo", "valence", "energy", "danceability",
        "acousticness", "duration_ms"
    ])

    df = df.with_columns([
        (pl.col("duration_ms") / 60000).alias("duration_minutes")
    ])

    df.write_csv(f"{DATA_PROCESSED}/spotify_transformed.csv")
    print("Spotify transformado.")
```

Proposito: Limpia y transforma el dataset de Spotify para facilitar su integración con otras fuentes.

Como funciona:

- Lee el dataset limpio desde CSV.
- Renombra columnas a minúsculas y formato estándar con guiones bajos.
- Aplica normalize_text a track_name y artists.
- Selecciona columnas relevantes y renombra track_name a song_name.
- Agrega duration_minutes como conversión de duration_ms.
- Guarda el resultado transformado en spotify_transformed.csv.

Escenario de uso: Parte fundamental de la fase de transformación previa a la integración con datos de premios o rankings.

TRANSFORM_GRAMMYS

```python
def transform_grammys():
    df = pl.read_csv(f"{DATA_RAW}/the_grammy_awards_clean.csv")
    df = df.rename({col: col.strip().lower().replace(" ", "_") for col in df.columns})

    df = df.with_columns([
        pl.col("nominee").map_elements(normalize_text, return_dtype=pl.Utf8),
        pl.col("artist").map_elements(normalize_text, return_dtype=pl.Utf8),
        pl.col("category").map_elements(normalize_text, return_dtype=pl.Utf8)
    ])

    df = df.filter(
        (pl.col("category").str.contains("song") | pl.col("category").str.contains("record")) &
        ~(pl.col("category").str.contains("album") | pl.col("category").str.contains("artist"))
    )

    df = df.with_columns([
        pl.col("nominee").alias("song_name")
    ])

    df.write_csv(f"{DATA_PROCESSED}/grammys_transformed.csv")
    print("Grammys transformado.")
```

Proposito: Filtra y transforma los datos de los premios Grammy para centrarse en canciones y grabaciones**.**

- **Como funciona:**
    - Lee el CSV limpio de los Grammy.
    - Renombra columnas a un formato consistente.
    - Normaliza los campos nominee, artist y category.
    - Filtra las categorías que contienen "song" o "record", excluyendo "album" y "artist".
    - Renombra nominee como song_name.
    - Guarda el dataset transformado en grammys_transformed.csv.
- **Escenario de uso:** Prepara los datos de los Grammy para ser unidos con datasets de canciones individuales.

TRANSFORM_API

```python
def transform_api():
    df = pl.read_csv(f"{DATA_RAW}/datos_api_spotify_expandido.csv")
    df = df.rename({
        "track_name": "song_name",
        "artist_name": "artist"
    })

    df = df.with_columns([
        pl.col("song_name").map_elements(normalize_text, return_dtype=pl.Utf8),
        pl.col("artist").map_elements(normalize_text, return_dtype=pl.Utf8),
        (pl.col("duration_ms") / 60000).alias("duration_minutes")
    ])

    # Lista de columnas deseadas
    columnas_deseadas = [
        "song_name", "artist", "popularity", "explicit",
        "tempo", "valence", "energy", "danceability", "acousticness",
        "duration_minutes", "album_name", "release_date", "spotify_url"
    ]

    # Filtrar solo las que existen en el DataFrame
    columnas_finales = [col for col in columnas_deseadas if col in df.columns]

    df = df.select(columnas_finales)

    df.write_csv(f"{DATA_PROCESSED}/api_spotify_transformed.csv")
    print(" API Spotify transformado.")



if __name__ == "__main__":
    transform_spotify()
    transform_grammys()
    transform_api()
```

Proposito: Limpia y normaliza los datos obtenidos desde la API de Spotify para unificarlos con el resto de fuentes.

Como funciona:

- Lee el CSV extraído de la API de Spotify.

- Renombra track_name a song_name y artist_name a artist.
- Aplica normalize_text a song_name y artist.
- Calcula la duración en minutos (duration_minutes).
- Filtra solo las columnas relevantes que existan.
- Guarda el dataset transformado como api_spotify_transformed.csv.

Escenario de uso: Normaliza datos enriquecidos desde la API para integrarlos en el análisis junto a Spotify y Grammy.

FASE DEL MERGE

```python
import polars as pl
import re
import os

def merge_all_data(**kwargs):
    # Define rutas absolutas
    DATA_PROCESSED = "/opt/airflow/dags/data/processed"
    DATA_FINAL = "/opt/airflow/dags/data/final"

    os.makedirs(DATA_FINAL, exist_ok=True)

    # Normalizador
    def normalize_text(value: str) -> str:
        if isinstance(value, str):
            value = value.lower()
            value = re.sub(r"[^a-z0-9\s]", "", value)
            value = re.sub(r"\s+", " ", value)
            return value.strip()
        return value

    # Cargar datasets transformados
    df_spotify = pl.read_csv(f"{DATA_PROCESSED}/spotify_transformed.csv")
    df_grammy = pl.read_csv(f"{DATA_PROCESSED}/grammys_transformed.csv")
    df_api    = pl.read_csv(f"{DATA_PROCESSED}/api_spotify_transformed.csv")

    # Normalizar nombres de canciones y artistas
    df_spotify = df_spotify.with_columns([
        pl.col("song_name").map_elements(normalize_text, return_dtype=pl.Utf8),
        pl.col("artist").map_elements(normalize_text, return_dtype=pl.Utf8)
    ])
    df_grammy = df_grammy.with_columns([
        pl.col("song_name").map_elements(normalize_text, return_dtype=pl.Utf8),
        pl.col("artist").map_elements(normalize_text, return_dtype=pl.Utf8)
    ])
    df_api = df_api.with_columns([
        pl.col("song_name").map_elements(normalize_text, return_dtype=pl.Utf8),
        pl.col("artist").map_elements(normalize_text, return_dtype=pl.Utf8)
    ])

    # Unir datasets
    merged = df_spotify.join(df_grammy, on=["song_name", "artist"], how="left")
    merged = merged.join(df_api, on=["song_name", "artist"], how="left")

    # Guardar resultado
    output_path = f"{DATA_FINAL}/merged_music_data.csv"
    merged.write_csv(output_path)
    print(f"✅ Dataset combinado guardado en: {output_path}")
```

Proposito: Fusiona los datasets transformados de Spotify, Grammy y API para construir un único dataset unificado y limpio.

Como funciona:

- Define las rutas absolutas para acceder a los archivos procesados y guardar el resultado final.
- Crea la carpeta data/final si no existe.
- Vuelve a aplicar la función normalize_text para asegurar que los campos song_name y artist estén uniformemente formateados en todos los datasets.

- Realiza dos uniones (join) secuenciales:
    - Primero: Spotify ↔ Grammy (por song_name y artist)
    - Segundo: el resultado anterior ↔ API de Spotify (por los mismos campos)
- Escribe el dataset resultante en un archivo llamado merged_music_data.csv.

Escenario de uso: Esta función es la etapa final del pipeline de transformación, y su salida es utilizada para análisis posteriores en Power BI u otros entornos de visualización.

FASE DE CARGA:

```
def load_and_store_final_dataset():
    try:
        # Conectar a PostgreSQL
        conn = psycopg2.connect(
            dbname=os.getenv("DB_NAME"),
            user=os.getenv("DB_USER"),
            password=os.getenv("DB_PASSWORD"),
            host=os.getenv("DB_HOST"),
            port=os.getenv("DB_PORT")
        )
        cur = conn.cursor()

        # Eliminar la tabla si ya existe
        cur.execute("DROP TABLE IF EXISTS public.final_tracks;")
        conn.commit()

        # Crear tabla con todas las columnas del CSV
        create_table_query = """
        CREATE TABLE IF NOT EXISTS public.final_tracks (
            song_name TEXT,
            artist TEXT,
            track_genre TEXT,
            popularity INTEGER,
            explicit BOOLEAN,
            tempo FLOAT,
            valence FLOAT,
            energy FLOAT,
            danceability FLOAT,
            acousticness FLOAT,
            duration_ms INTEGER,
            duration_minutes FLOAT,
            year INTEGER,
            title TEXT,
            category TEXT,
            nominee TEXT,
            workers TEXT,
            img TEXT,
            winner TEXT,
            published_date TEXT,
            published_time TEXT,
            updated_date TEXT,
            updated_time TEXT,
            popularity_right INTEGER,
            explicit_right BOOLEAN
```

```
            updated_time TEXT,
            popularity_right INTEGER,
            explicit_right BOOLEAN,
            duration_minutes_right FLOAT,
            album_name TEXT,
            release_date TEXT,
            spotify_url TEXT
        );
        """
        cur.execute(create_table_query)
        conn.commit()

        # Ruta absoluta al CSV final
        final_path = "/opt/airflow/data/final/merged_music_data.csv"


        # Cargar datos desde el CSV
        copy_query = """
        COPY public.final_tracks (
            song_name, artist, track_genre, popularity, explicit, tempo, valence, energy,
            danceability, acousticness, duration_ms, duration_minutes, year, title,
            category, nominee, workers, img, winner, published_date, published_time,
            updated_date, updated_time, popularity_right, explicit_right,
            duration_minutes_right, album_name, release_date, spotify_url
        )
        FROM STDIN WITH CSV HEADER DELIMITER ',' NULL ''
        """
        with open(final_path, 'r', encoding='utf-8') as f:
            cur.copy_expert(copy_query, f)

        conn.commit()
        print("✅ Datos cargados en PostgreSQL.")

    except psycopg2.Error as e:
        print("❌ Error al conectar o cargar datos en PostgreSQL:", e)

    finally:
        if 'cur' in locals():
            cur.close()
        if 'conn' in locals():
            conn.close()

if __name__ == "__main__":
    load_and_store_final_dataset()
```

- **Proposito:** Crea una tabla en PostgreSQL y carga en ella el dataset final resultante del proceso ETL.
- **Como funciona:**
    - Carga las variables de conexión desde un archivo .env.
    - Establece conexión con la base de datos PostgreSQL.
    - Elimina la tabla final_tracks si ya existe para evitar conflictos.
    - Crea una nueva tabla public.final_tracks con todas las columnas del dataset final.

- Usa el comando COPY FROM para cargar los datos directamente desde el archivo CSV merged_music_data.csv.
- Maneja posibles errores y cierra la conexión de forma segura.

IMPLEMENTACIÓN DEL PIPELINE EN AIRFLOW

Esta sección describe cómo se configuró y ejecutó el pipeline ETL utilizando **Apache Airflow** en un entorno contenerizado mediante **Docker Compose (versión 3.8)**.

La arquitectura incluye tres servicios principales:

- Una base de datos **PostgreSQL** que almacena tanto los metadatos de Airflow como los datos del proyecto,
- Un contenedor para el **webserver de Airflow**,
- Y un contenedor para el **scheduler**, que coordina la ejecución de tareas.

El pipeline ejecuta tareas de **extracción, transformación, combinación y carga de datos** a partir de las fuentes Spotify, Grammy y API, todo orquestado a través del DAG.

SERVICIOS DEFINIDOS:

POSTGRES

- **Imagen base:** postgres:14.
- **Variables de entorno:**
  - POSTGRES_USER=airflow
  - POSTGRES_PASSWORD=airflow
  - POSTGRES_DB=airflow
- Monta un volumen persistente postgres_db para mantener la base de datos.
- Expone el puerto 5432 internamente, accesible desde el host por el 5433.

WEBSERVER

- Construido desde el Dockerfile del proyecto.
- Usa el LocalExecutor como motor de ejecución.
- Conexión a la base de datos mediante SQL_ALCHEMY_CONN.
- Volúmenes montados:
    - dags, logs, plugins, src, data, client_secrets.json y requirements.txt.
- Expone el puerto 8080 como 9090 en el host.

Comando de inicio:

1. Inicializa la base de datos (airflow db init),
2. Crea el usuario administrador,
3. Inicia el scheduler en segundo plano,
4. Ejecuta automáticamente el DAG etl_music_pipeline,
5. Finalmente lanza el servidor web (airflow webserver).

SCHEDULER

- También se construye desde el Dockerfile.
- Ejecuta directamente el comando airflow scheduler.
- Depende tanto del webserver como del postgres.
- Monta los mismos volúmenes que el webserver.

```yaml
version: '3.8'

Run All Services
services:
  Run Service
  postgres:
    image: postgres:14
    environment:
      - POSTGRES_USER=airflow
      - POSTGRES_PASSWORD=airflow
      - POSTGRES_DB=airflow
    volumes:
      - postgres_db:/var/lib/postgresql/data
    ports:
      - "5433:5432"

  Run Service
  webserver:
    build: .
    restart: always
    depends_on:
      - postgres
    environment:
      - AIRFLOW__CORE__EXECUTOR=LocalExecutor
      - AIRFLOW__DATABASE__SQL_ALCHEMY_CONN=postgresql+psycopg2://airflow:airflow@postgres/airflow
      - AIRFLOW__WEBSERVER__EXPOSE_CONFIG=True
    volumes:
      - ./dags:/opt/airflow/dags
      - ./logs:/opt/airflow/logs
      - ./plugins:/opt/airflow/plugins
      - ./src:/opt/airflow/src
      - ./data:/opt/airflow/data
      - ./client_secrets.json:/opt/airflow/client_secrets.json
      - ./requirements.txt:/requirements.txt
    ports:
      - "9090:8080"
    command: >
      bash -c "
       airflow db init &&
       airflow users create --username admin --password admin --firstname Admin --lastname User --role Admin --email admin@example.com
       airflow scheduler & sleep 10 &&
       airflow dags trigger etl_music_pipeline &&
       airflow webserver
      "
```

```yaml
  Run Service
  scheduler:
    build: .
    restart: always
    depends_on:
      - webserver
      - postgres
    environment:
      - AIRFLOW__CORE__EXECUTOR=LocalExecutor
      - AIRFLOW__DATABASE__SQL_ALCHEMY_CONN=postgresql+psycopg2://airflow:airflow@postgres/airflow
    volumes:
      - ./dags:/opt/airflow/dags
      - ./logs:/opt/airflow/logs
      - ./plugins:/opt/airflow/plugins
      - ./src:/opt/airflow/src
      - ./data:/opt/airflow/data
      - ./client_secrets.json:/opt/airflow/client_secrets.json
      - ./requirements.txt:/requirements.txt
    command: scheduler

volumes:
  postgres_db:
```

IMPLEMENTACION DEL DAG (AIRFLOW)

El DAG (Grafo Acíclico Dirigido) define la estructura y el orden de ejecución del pipeline ETL dentro de **Apache Airflow**. Este archivo se encuentra en la carpeta dags/ del proyecto, lo que permite que Airflow lo detecte automáticamente y pueda ejecutarlo desde la interfaz web o CLI.

ASPECTOS CLAVE DEL DAG

1. Importación de módulos

- Se importan componentes esenciales de Airflow como DAG y PythonOperator.
- Se modifica el sys.path para incluir el directorio /opt/airflow/src, lo que permite acceder a los módulos de extracción, transformación, combinación y carga definidos en la carpeta src/.
- **Configuración por defecto:**

```
default_args = {
    'owner': 'airflow',
    'start_date': datetime(2024, 1, 1),
    'retries': 1,
}
```

  - Define al usuario responsable (owner) del DAG.
  - Establece una fecha de inicio fija y la cantidad de reintentos si alguna tarea falla.

DEFINICIÓN DEL DAG:

```
with DAG(
    dag_id='etl_music_pipeline',
    default_args=default_args,
    description='Pipeline ETL para integrar Spotify, Grammy y API',
    schedule_interval=None,  # Puedes cambiarlo a diario si gustas
    catchup=False,
) as dag:
```

  - El identificador del DAG es etl_music_pipeline.

- El schedule_interval está configurado como None, lo que implica que la ejecución es manual (aunque puede cambiarse a @daily para automatización).
- catchup=False evita la ejecución retroactiva de tareas desde la start_date.

DEFINICIÓN DE TAREAS

Cada tarea se implementa como un PythonOperator, que ejecuta una función definida en los scripts de ETL.

Tareas de extracción:

- extract_spotify: Carga datos de Spotify desde CSV local.
- extract_grammy: Carga datos limpios de los premios Grammy.
- extract_api: Obtiene canciones populares desde la API de Spotify.

Tareas de transformación:

- transform_spotify: Limpia y normaliza datos de Spotify.
- transform_grammys: Filtra y normaliza los datos de Grammy.
- transform_api: Procesa los datos obtenidos desde la API.

Tarea de combinación:

- merge_data: Une los tres datasets transformados en un solo conjunto de datos final.

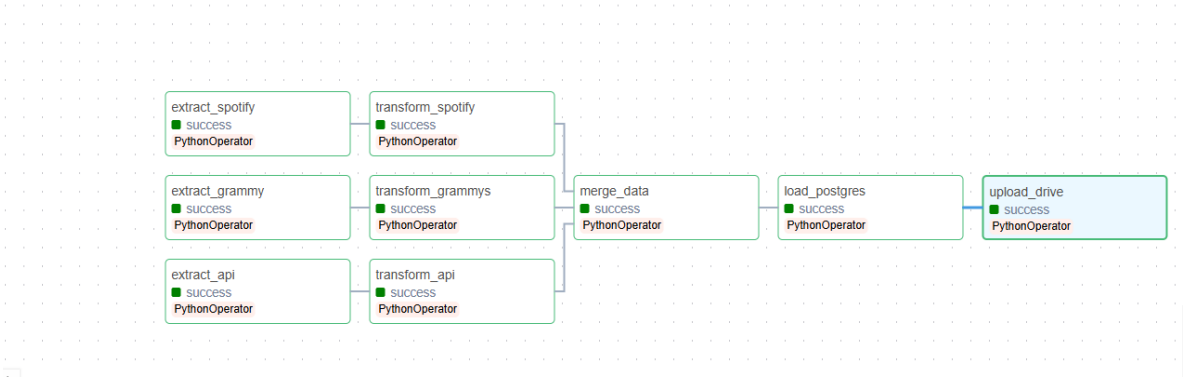
Tareas de carga:

- load_postgres: Carga el dataset combinado en una tabla PostgreSQL (final_tracks).
- upload_drive: Sube el archivo final a Google Drive para respaldo o uso externo.

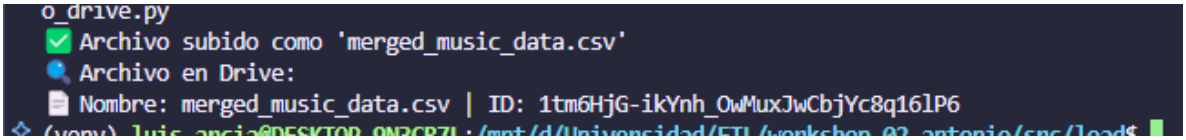
DEPENDENCIAS ENTRE TAREAS:

```
t1 >> t4
t2 >> t5
t3 >> t6
[t4, t5, t6] >> t7 >> t8 >> t9
```

- Primero se ejecutan las tres tareas de extracción en paralelo.
- Una vez finalizadas, se ejecutan las tres tareas de transformación también en paralelo.
- Luego se combinan los datos con merge_data.
- Finalmente, se cargan en PostgreSQL (load_postgres) y se suben a Google Drive (upload_drive).



CARGA A GOOGLE DRIVE:



ANALISIS MERGE

El dataset resultante de la fusión entre los datos de Spotify (CSV), los premios Grammy y la API de Spotify contiene 113,999 registros y 29 columnas. Esta integración representa un compendio enriquecido de información musical que incluye características técnicas, metadatos, premios y popularidad.

Variables Presentes

- **Variables numéricas continuas:** tempo, valence, energy, danceability, acousticness, duration_minutes, popularity, popularity_right, duration_minutes_right.

- **Variables categóricas:** song_name, artist, track_genre, album_name, category, nominee, workers, img.
- **Fechas:** published_date, updated_date, published_time, updated_time.
- **Boleanas:** explicit, explicit_right, winner.

Estadísticas Descriptivas

- La variable popularity presenta una distribución sesgada hacia la izquierda, con un pico muy alto en el rango de 0 a 10. También se identifican dos picos secundarios alrededor de los valores 20–40 y 50–60. Esto indica la coexistencia de canciones poco populares y canciones moderadamente reconocidas, con pocas canciones altamente virales.
- La variable duration_minutes se encuentra centrada entre los 3.5 y 4 minutos, en línea con los estándares comerciales. Existen algunos valores extremos que superan los 10 minutos, pero representan una proporción muy baja.
- No se identifican valores nulos en las columnas clave tras la integración, lo cual valida el proceso de limpieza y merge realizado en etapas previas.

Observaciones

- La fusión exitosa de múltiples fuentes permitió enriquecer el dataset sin perder calidad ni coherencia estructural.
- Los valores *_right (como popularity_right, explicit_right, duration_minutes_right) provienen de la API de Spotify y fueron integrados correctamente como respaldo o comparación.
- La distribución de los valores es adecuada para análisis predictivos y segmentación, especialmente al combinar variables de rendimiento (como popularidad) con aspectos líricos, técnicos y de premiación.

Conclusión

Este dataset final ofrece una visión global y multidimensional del panorama musical, combinando datos técnicos de audio, metadatos de canciones, reconocimientos musicales y métricas de consumo. Su estructura robusta lo hace apto tanto para visualización interactiva como para modelamiento predictivo avanzado.

CONCLUSION GENERAL

Este proyecto logró implementar con éxito un pipeline ETL completo e integrado para analizar la industria musical desde una perspectiva amplia, combinando tres fuentes clave: datos de Spotify, registros históricos de los Premios Grammy y una consulta

personalizada a la API de Spotify. La solución permitió extraer, transformar y unificar más de 100 mil registros musicales, los cuales fueron almacenados en una base de datos PostgreSQL y visualizados posteriormente en un dashboard interactivo construido con Dash y Plotly.

Desde el análisis exploratorio, se identificaron tendencias clave en la popularidad, duración y características técnicas de las canciones, así como patrones históricos en la entrega de premios y la trayectoria de los artistas más influyentes. Se detectaron sesgos relevantes, como la baja popularidad de la mayoría de canciones o la concentración de lanzamientos recientes, lo cual permitió hacer recomendaciones sobre el uso de los datos en modelos predictivos y estrategias de marketing.

Gracias a la orquestación automática con Apache Airflow y al despliegue modular con Docker Compose, el proyecto garantiza escalabilidad, mantenibilidad y reproducibilidad. Además, la carga automática en Google Drive facilita la distribución de los resultados procesados.