

ETL Workshop 2

Airflow – Spotify Analysis

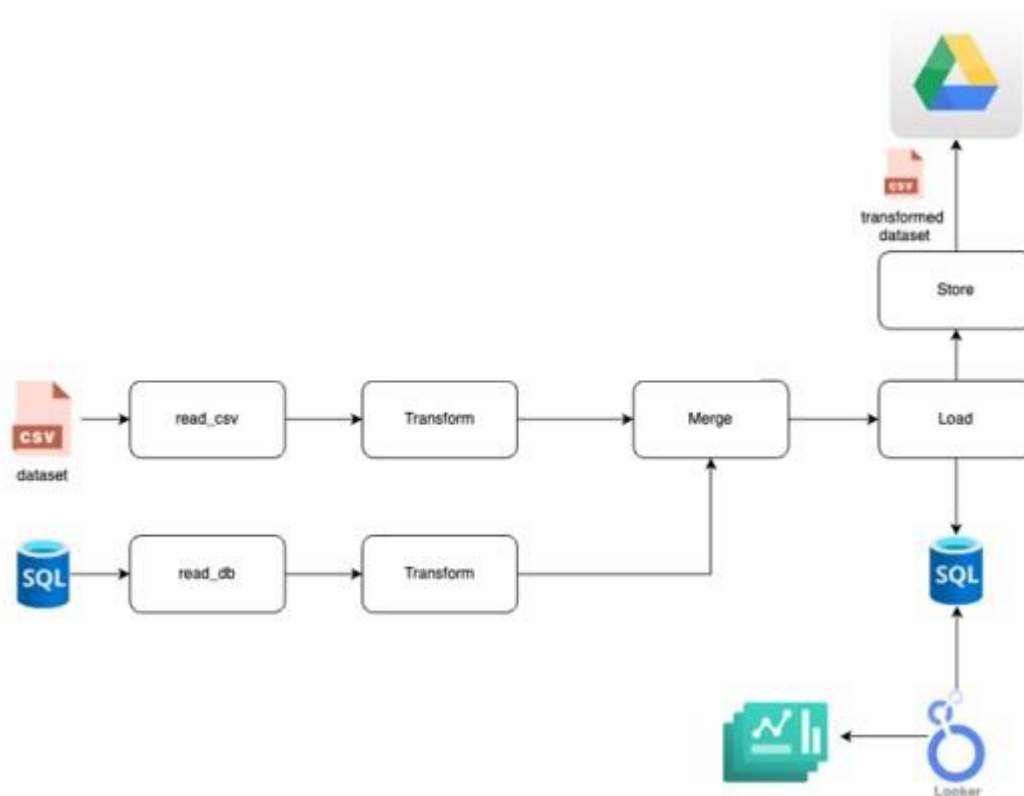
2221222

Joan Mateo Bermudez Collazos

2024

Universidad Autónoma de Occidente **Descripción del reto**

El segundo reto se trataba de generar un pipeline de datos que vienen de dos diferentes fuentes de datos (csv, sql) con datos sobre música un dataset sobre los Grammys nominados y el segundo sobre información de canciones sacado del api de Spotify, el objetivo es identificar y estudiar los datos que nos entregan para poder unir los datos de diferentes fuentes para generar así un análisis un poco mas valioso al poder complementar los datos, el reto de esto también es después de identificar y analizar como hacer el merge de los datos poder generar un proceso de etl con airflow donde podremos orquestar un proceso de etl donde al final después de tener los datos limpios y unidos los guardaremos tanto en una base de datos como en Google Drive mediante su API como un archivo csv



La idea es generar un flujo similar al anterior.

Desarrollo del reto

Planeación

Para el reto las tecnologías obligatorias son Python, Airflow, Drive-api, para poder cumplir con lo requerido decidí utilizar docker para la base de datos y wsl (Windows subsystem for linux) para correr el proyecto en general ya que Airflow corre nativamente en sistemas unix, utilizare power bi para la presentación del dashboard y analítica además de los notebooks de python la base de datos se conectara con los notebooks y el orquestador Airflow utilizando la red local de la maquina Ubuntu(wsl) pero cuando se necesite acceder a ella mediante la aplicación de visualización se hará a través de la red creada por wsl para acceder desde mi maquina local hacia la base de datos que corre en docker dentro de la maquina Linux, la red que se le asigno a tu maquina wsl se puede ver con el siguiente comando `ip route show | grep -i default | awk '{ print $3}'`.

instalar lo necesario

- 1) Instala y verifica el funcionamiento de wsl

Link Documentación: [Instalación de WSL | Microsoft Learn](#)

- 2) instalación de los programas para el funcionamiento

- a. instalar python3 una versión soportada por Airflow recomendando min 3.9

- b. instalar docker

Link Documentación: [Install Docker Engine on Ubuntu | Docker Docs](#)

- c. recomendable setear la variable de sistema para la ruta de funcionamiento de Airflow dentro del directorio del proyecto `echo export`

`AIRFLOW_HOME=$(pwd) >> ~/.bashrc`

- d. instalar Airflow con pip

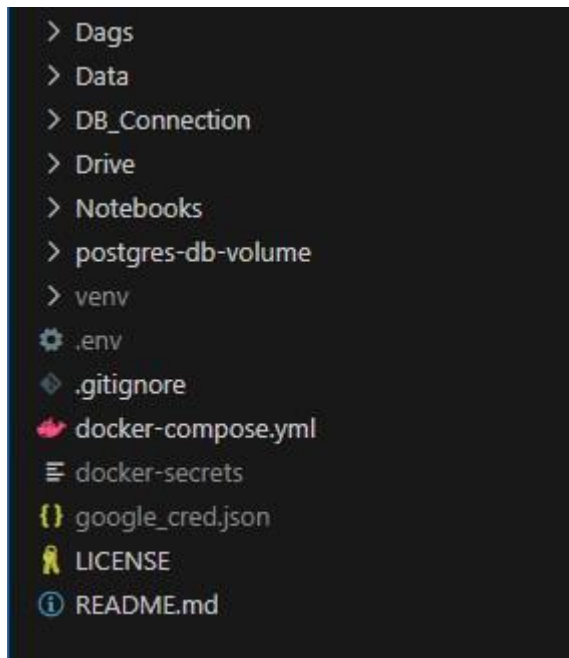
Link Documentación : [Installation of Airflow™ — Airflow Documentation \(apache.org\)](#)

- e. descargar la imagen de docker para la base de datos `postgres16.2-alpine3.19` con el comando `docker pull postgres16.2-alpine3.19`

Creación de archivos de contraseñas y el entorno virtual

- crear los archivos de contraseñas necesarios
 - a. .env
 - b. docker secrets
- creación del entorno virtual `python3 -m venv venv`
- creación del archivo .gitignore para evitar subir al repositorio los archivos de contraseñas y bases de datos

Generar la estructura de carpetas para el proyecto



Dags

- Carpeta la cual se define como casa de airflow

Data

- Carpeta donde leeremos y guardaremos los csvs

DB_Connection

- Carpeta donde creamos las funcionalidades para el manejo de la base de datos Drive
- Carpeta donde crearemos las funcionalidades para el manejo de la api de Google drive

Notebooks

- Carpeta donde revisaremos los datos y apartir de ese analisi definiremos las necesidades del dag

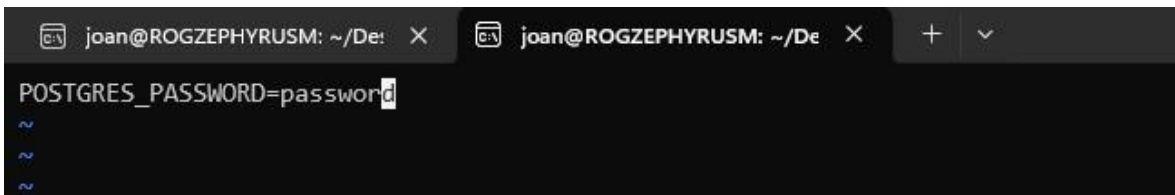
postgres-db-volume

- Carpeta que será utilizada como volumen por docker

Configuración de archivo docker compose

```
services:
  postgresdb:
    image: postgres:16.2-alpine3.19
    volumes:
      - postgres-db-volume:/var/lib/postgresql/data
    env_file:
      - ./docker-secrets
    ports:
      - 54321:5432
volumes:
  postgres-db-volume:
```

Definimos un servicio llamado postgresdb donde utilizamos la imagen ya antes descargada definimos puertos y volúmenes además de utilizar un archivo de variables de entorno llamado docker-secrets donde pasamos la variable de entorno obligatoria para correr el contenedor docker - Secrets



The screenshot shows a terminal window with two tabs. The active tab is titled 'joan@ROGZEPHYRUSM: ~/De'. The command 'POSTGRES_PASSWORD=password' has been entered and is followed by a cursor. Below the command, there are three tilde characters (~) on separate lines.

corremos el contenedor con el comando `sudo docker compose up`

ingresamos al contenedor y creamos la base de datos en este caso workshop_2

`sudo docker exec -it etl_workshop_2-postgresdb-1 /bin/bash` `psql -U postgres`

```
joan@ROGZEPHYRUSM:~$ sudo docker ps -a
[sudo] password for joan:
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                               NAMES
193375f01f57   postgres:16.2-alpine3.19           "docker-entrypoint.s..." 6 days ago    Up 12 minutes  0.0.0.0:54321->5432/tcp, :::54321->5432/tcp  etl_workshop_2-postgresdb-1
joan@ROGZEPHYRUSM:~$ sudo docker exec -it /bin/bash
"docker exec" requires at least 2 arguments.
See 'docker exec --help'.

Usage:  docker exec [OPTIONS] CONTAINER COMMAND [ARG...]

Execute a command in a running container
joan@ROGZEPHYRUSM:~$ sudo docker exec -it /bin/bash 193375f01f57
Error response from daemon: No such container: bin/bash
joan@ROGZEPHYRUSM:~$ sudo docker exec -it 193375f01f57 /bin/bash
193375f01f57:/# psql
psql: error: connection to server on socket "/var/run/postgresql/.s.PGSQL.5432" failed: FATAL:  role "root" does not exist
193375f01f57:/# psql -U postgres
psql (16.2)
type "help" for help.

postgres=# \l

```

Name	Owner	Encoding	Locale	Provider	Collate	Ctype	ICU Locale	ICU Rules	Access privileges
postgres	postgres	UTF8	libc		en_US.utf8	en_US.utf8			
template0	postgres	UTF8	libc		en_US.utf8	en_US.utf8			~c/postgres +
template1	postgres	UTF8	libc		en_US.utf8	en_US.utf8			~c/postgres +
workshop_2	postgres	UTF8	libc		en_US.utf8	en_US.utf8			postgres-CITc/postgres

```
(4 rows)

postgres=#
```

Vemos las tablas y eliminamos las que ya se habían trabajado

```
Schema | Name          | Type | Owner
-----+-----+-----+-----
public | grammys       | table | postgres
public | grammys_clean | table | postgres
public | merge         | table | postgres
(3 rows)

workshop_2=# ;
ERROR:  syntax error at or near "drop"
LINE 2: drop table grammys_clean
        ^

workshop_2=# drop table merge;
DROP TABLE
workshop_2=# drop table grammys_clean;
DROP TABLE
workshop_2=#
```

Ahora que ya tenemos los datos listos para leer podemos empezar el análisis exploratorio

EDA Grammys

```
import sys
sys.path.append("../DB_Connection")
import psycopg2
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime
import logging

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger("ml_logger")

def workers_separate(wstring:str)->list:
    """This function is for separate the job and the persons in the dataframe"""
    if type(wstring) != str:
        return wstring
    try:
        wlist = wstring.split(sep=",") #transform the string to list -> "name, role"
        wlist = [i if i[0]!=" " else i[1:] for i in wlist]
        wdict = {}
        for i in range(0, len(wlist)):
            wdict[i] = wlist[i].split(sep=",")
            if " " in wdict[i][0]:
                wdict[i] = wdict[i].split(sep=" " & ",")
            continue
        wdict[i] = wdict[i][1]
    except Exception as error:
        #look for strange thinks
        #logger.warning(f"({datetime.now()}) {error}") #this part is for the DAG
    return wdict
```

Importamos las librerías necesarias y ya podemos ver algunas de las funciones que estábamos probando

Esta primera función fue diseñada para poder separar los nombres de los que trabajaron con los artistas ya que venían en texto separados por comas y puntos y comas pero al hacerlo me di cuenta que no seguían una misma estructura y muchos no tenían que hicieron en el proyecto aveces aparecía varias veces el nombre o aparecía el parentesco con el artista por lo que decidí que esta columna no iba a dar un análisis consistente.

```
df_grammys["workers_test"] = df_grammys["workers"].apply(workers_separate)
#creates the column workers_test to analyze the data compliance
#the column meets the respective job and the names

df_grammys.head(5)
```

	year	title	published_at	updated_at	category	nominee	artist	workers	img	winner	workers_test
0	2019	62nd Annual GRAMMY Awards (2019)	2020-05-19T05:10:28-07:00	2020-05-19T05:10:28-07:00	Record Of The Year	Bad Guy	Billie Eilish	Finnneas O'Connell, producer: Rob Kinelski & Ft...	https://www.grammy.com/sites/com/files/styles/...	True	[producer: [Finnneas O'Connell], engineers...
1	2019	62nd Annual GRAMMY Awards (2019)	2020-05-19T05:10:28-07:00	2020-05-19T05:10:28-07:00	Record Of The Year	Hey, Ma	Bon Iver	BJ Burton, Brad Cook, Chris Messina & Justin V...	https://www.grammy.com/sites/com/files/styles/...	True	[Brad Cook: [BJ Burton], Zach Hanson & Ch...
2	2019	62nd Annual GRAMMY Awards (2019)	2020-05-19T05:10:28-07:00	2020-05-19T05:10:28-07:00	Record Of The Year	7 rings	Ariana Grande	Charles Anderson, Tommy Brown, Michael Foster ...	https://www.grammy.com/sites/com/files/styles/...	True	[Tommy Brown: [Charles Anderson], John Ha...
3	2019	62nd Annual GRAMMY Awards (2019)	2020-05-19T05:10:28-07:00	2020-05-19T05:10:28-07:00	Record Of The Year	Hard Place	H.E.R.	Rodney "Darkchild" Jenkins, producer: Joseph H...	https://www.grammy.com/sites/com/files/styles/...	True	[producer: [Rodney "Darkchild" Jenkins] ...
4	2019	62nd Annual GRAMMY Awards (2019)	2020-05-19T05:10:28-07:00	2020-05-19T05:10:28-07:00	Record Of The Year	Talk	Khalid	Disclosure & Denis Kosiak, producers: Ingmar C...	https://www.grammy.com/sites/com/files/styles/...	True	[producers: [Disclosure', Denis Kosiak] ...

After try to do the workers column transformation i discover that the data in this column dont have the consistency necesari to have a good analysys for job

```
df_grammys.drop(columns=["workers_test"], inplace=True) #Drop the column workers_test
df_grammys.head(5)
```

	year	title	published_at	updated_at	category	nominee	artist	workers	img	winner
0	2019	62nd Annual GRAMMY Awards (2019)	2020-05-19T05:10:28-07:00	2020-05-19T05:10:28-07:00	Record Of The Year	Bad Guy	Billie Eilish	Finnneas O'Connell, producer: Rob Kinelski & Ft...	https://www.grammy.com/sites/com/files/styles/...	True
1	2019	62nd Annual GRAMMY Awards (2019)	2020-05-19T05:10:28-07:00	2020-05-19T05:10:28-07:00	Record Of The Year	Hey, Ma	Bon Iver	BJ Burton, Brad Cook, Chris Messina & Justin V...	https://www.grammy.com/sites/com/files/styles/...	True
2	2019	62nd Annual GRAMMY Awards (2019)	2020-05-19T05:10:28-07:00	2020-05-19T05:10:28-07:00	Record Of The Year	7 rings	Ariana Grande	Charles Anderson, Tommy Brown, Michael Foster ...	https://www.grammy.com/sites/com/files/styles/...	True
3	2019	62nd Annual GRAMMY Awards (2019)	2020-05-19T05:10:28-07:00	2020-05-19T05:10:28-07:00	Record Of The Year	Hard Place	H.E.R.	Rodney "Darkchild" Jenkins, producer: Joseph H...	https://www.grammy.com/sites/com/files/styles/...	True
4	2019	62nd Annual GRAMMY Awards (2019)	2020-05-19T05:10:28-07:00	2020-05-19T05:10:28-07:00	Record Of The Year	Talk	Khalid	Disclosure & Denis Kosiak, producers: Ingmar C...	https://www.grammy.com/sites/com/files/styles/...	True

```
def datetime_transform(stdate:str)->datetime:
    date = datetime.strptime(stdate, "%Y-%m-%d %H:%M:%S%z")
    formatdate = date.strftime("%Y-%m-%d") #this format decision is for analysis requirement
    return formatdate
```

Esta segunda función permitió separar la fecha de la columna published_at que venia en un formato mas extenso y en string a un formato tipo Date para sql y un análisis mas fácil

```
df_grammys["published_date"] = df_grammys["published_at"].apply(datetime_transform)

df_grammys.head(5)
```

	year	title	published_at	updated_at	category	nominee	artist	workers	img	winner	published_date
0	2019	62nd Annual GRAMMY Awards (2019)	2020-05-19T05:10:28-07:00	2020-05-19T05:10:28-07:00	Record Of The Year	Bad Guy	Billie Eilish	Finnese O'Connell, producer; Rob Kretzki & FL	https://www.grammy.com/sites/com/files/styles/_...	True	2020-05-19
1	2019	62nd Annual GRAMMY Awards (2019)	2020-05-19T05:10:28-07:00	2020-05-19T05:10:28-07:00	Record Of The Year	Hey, Ma	Bon Iver	BJ Burton, Brad Cook, Chris Messina & Justin V...	https://www.grammy.com/sites/com/files/styles/_...	True	2020-05-19
2	2019	62nd Annual GRAMMY Awards (2019)	2020-05-19T05:10:28-07:00	2020-05-19T05:10:28-07:00	Record Of The Year	7 rings	Ariana Grande	Charles Anderson, Tommy Brown, Michael Foster ...	https://www.grammy.com/sites/com/files/styles/_...	True	2020-05-19
3	2019	62nd Annual GRAMMY Awards (2019)	2020-05-19T05:10:28-07:00	2020-05-19T05:10:28-07:00	Record Of The Year	Hard Place	H.E.R.	Rodney "Darkchild" Jerkins, producer; Joseph H...	https://www.grammy.com/sites/com/files/styles/_...	True	2020-05-19
4	2019	62nd Annual GRAMMY Awards (2019)	2020-05-19T05:10:28-07:00	2020-05-19T05:10:28-07:00	Record Of The Year	Talk	Khalid	Disclosure & Denis Kosiak, producers; Ingmar C...	https://www.grammy.com/sites/com/files/styles/_...	True	2020-05-19

Identificamos valores faltantes y duplicados y los trate para mantener la consistencia en el análisis

```
for i in df_grammys.keys():
    print(f"{i} nulls : {4810 - df_grammys[i].notnull().value_counts()[0]}")

year nulls : 0
title nulls : 0
published_at nulls : 0
updated_at nulls : 0
category nulls : 0
nominee nulls : 0
artist nulls : 1840
workers nulls : 2190
img nulls : 1847
winner nulls : 0

We see that workers have a lot of nulls entries and artist column have a lot of nulls entries
```

```
print(f"Data entries : {df_grammys.shape[0]} | Data entries without duplicates : {df_grammys[['year','artist','category','nominee']].duplicated().value_counts()[0]}")

Data entries : 4810 | Data entries without duplicates : 4810

We can see that we do not have duplicates if we look at the categories that will be repeated if a disk were duplicated year of the grammys, artist of the song, category and finally the nominee
```

```
df_grammys.published_date.unique()

array(['2020-05-19', '2018-12-06', '2017-11-28', '2018-05-22'],
      dtype=object)

We observe some strange think althoug we have 4810 entries whit a lot of differents songs most of the published dates are the same day (2017-11-28)
```

Vemos que hay muy pocas fechas donde se hicieron los Grammys en nuestro dataset

```
#look for the songs in this date 2017-11-28
df_grammys[df_grammys.published_date == "2017-11-28"]["winner"].value_counts()

winner
True    4285
Name: count, dtype: int64
```

Vemos que la mayoría están en esta fecha "2017-11-28"

```
df_grammys.drop(columns=["winner", "updated_at", "published_at", "workers", "img"], inplace=True)
```

Por ultimo elimine las columnas que no se me hacían interesantes para el análisis en conjunto con el otro dataset como ganador que todo estaba en True fecha de actualización fecha de publicación que ya habíamos transformado en otra columna

Img que era un link a la imagen de la canción y workers que ya vimos que no era muy consistente

Por ultimo volvemos a guardar en la base de datos

```
[28]: df_grammys.to_sql("grammys_clean", Pysqlconnect.connection(), if_exists="replace", index_label="id")  
... 978
```

Aquí ya hemos identificado las tareas que asignaremos en el workflow para el tratamiento del dataset de Grammys

Spotify EDA

```
EDA spotify  
#preproces stage import libraries  
import pandas as pd  
import matplotlib.pyplot as plt  
import numpy as np  
from datetime import datetime  
Python
```

Empezamos igual importando los módulos necesarios para el trabajo

```
df_spotify["Unnamed: 0"].nunique()  
#we dont have duplicates but i will delete this column  
Python  
... 114800  
  
df_spotify.index  
Python  
... RangeIndex(start=0, stop=114800, step=1)  
  
df_spotify.drop(columns=["Unnamed: 0"], inplace=True)  
Python
```

Vemos una columna extraña y identificamos que se trata de una columna tipo index la eliminamos ya que tenemos la columna propia de pandas que también tiene un nombre mas descriptivo de lo que es.

Buscamos valores nulos pero encontramos que solo hay una fila con nulos en todo el dataset

```
#look for nulls values
df_spotify.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 114000 entries, 0 to 113999
Data columns (total 20 columns):
#   Column                Non-Null Count  Dtype
---  -
0   track_id              114000 non-null object
1   artists               113999 non-null object
2   album_name            113999 non-null object
3   track_name            113999 non-null object
4   popularity             114000 non-null int64
5   duration_ms           114000 non-null int64
6   explicit              114000 non-null bool
7   danceability           114000 non-null float64
8   energy                114000 non-null float64
9   key                   114000 non-null int64
10  loudness               114000 non-null float64
11  mode                  114000 non-null int64
12  speechiness            114000 non-null float64
13  acousticness           114000 non-null float64
14  instrumentalness       114000 non-null float64
15  liveness               114000 non-null float64
16  valence                114000 non-null float64
17  tempo                 114000 non-null float64
18  time_signature         114000 non-null int64
19  track_genre            114000 non-null object
dtypes: bool(1), float64(9), int64(5), object(5)
memory usage: 16.6+ MB
```

```
df_spotify[df_spotify["artists"].isnull()]
```

	track_id	artists	album_name	track_name	popularity	duration_ms	explicit	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo
65900	1NR4glb7nGxHP3D2f659	NaN	NaN	NaN	0	0	False	0.501	0.583	7	-9.46	0	0.0605	0.69	0.00396	0.0747	0.734	138.391

Empty markdown cell, double-click or press enter to edit.

```
#here we can see the register with null columns
```

```
df_spotify.drop_duplicates(inplace=True) #Remove duplicates entries
```

ELIMINAMOS y eliminamos duplicados

```
df_spotify[df_spotify["artists"].isnull()]
```

track_id	artists	album_name	track_name	popularity	duration_ms	explicit	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	
65900	1k84glb7nGxHPi3D2iR59	NaN	NaN	NaN	0	0	False	0.501	0.583	7	-9.46	0	0.0605	0.69	0.00396	0.0747	0.734	138.391

```
df_spotify.dropna(inplace=True)
```

Después de eliminar duplicados seguimos viendo a mas profundidad y vemos que cuando filtramos por algunas columna con información sobre la canción que si es la misma muy probablemente sea la misma canción y nos encontramos con una falla en la consistencia de los datos y es que hay canciones duplicadas donde su única diferencia es su clasificación de genero esto probablemente por la forma en que se obtuvieron los datos originalmente que pude leer en kaggle fue por medio de una api utilizando descargas por genero

```
df_spotify[df_spotify["track_name"] == "Lucky"]
#search for duplicates inconsistent
```

track_id	artists	album_name	track_name	popularity	duration_ms	explicit	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence
8	0lktbUcnAGrvD03AWnz3Q8	Jason Mraz:Colbie Caillat	We Sing, We Dance, We Steal Things. Lucky	74	189613	False	0.625	0.414	0	-8.700	1	0.0369	0.2940	0.000000	0.151	0.669
18	2qLM6tutEC3uGJg45MMN6	Jason Mraz:Colbie Caillat	We Sing, We Dance, We Steal Things. Lucky	68	189613	False	0.625	0.414	0	-8.700	1	0.0369	0.2940	0.000000	0.151	0.669
17332	4BaEOlHGeP9gw5ToelY7a	Glee Cast	Glee: The Music, Volume 4 Lucky	49	187560	False	0.632	0.468	0	-8.089	1	0.0412	0.5410	0.000000	0.165	0.669
106085	2zIT6D1LJ5HghHajsq1XLc	Jubël:Noa Kirel	Happy Pop Vibes Lucky	0	142038	False	0.746	0.738	1	-6.350	0	0.0354	0.0899	0.000063	0.103	0.803

here i dont know wath dom, i need more information about the dataset i reed in kaggle repository and i think could be the section that say song its evaluated different for artist or for album

```
#now with the original df
df_spotify = df_spotify[df_spotify["track_id"].duplicated() == False]
```

```
df_spotify.shape
```

```
(89741, 20)
```

```
df_spotify.track_id.duplicated().value_counts()
```

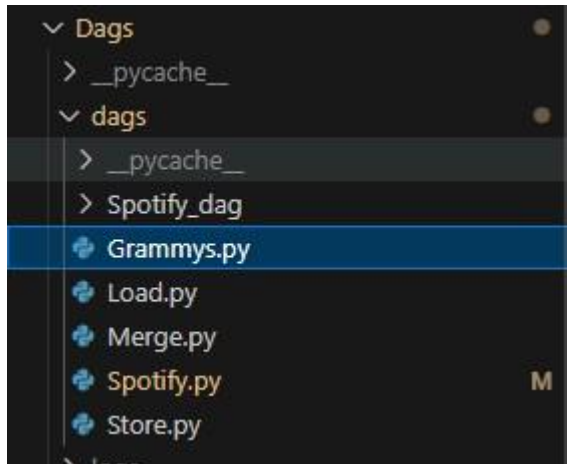
```
track_id
False      89741
Name: count, dtype: int64
```

now is not more duplicates

Creamos un dataframe excluyendo los duplicados esta vez por id de canción únicamente y ya nos quedamos sin canciones duplicadas

Definición del dag

Luego de haber identificado la limpieza necesaria para cada dataset crudo empezamos a definir las funciones que serán utilizadas por Airflow entonces empezamos a escribirlas para que puedan ser utilizadas dentro de Airflow y las escribiremos dentro de la carpeta dags esto no es necesario



```

import sys
sys.path.append("/home/joan/Desktop/etl_workshop_2/DB_Connection")
import Pysqlconnect
import pandas as pd
from datetime import datetime
import logging
import json

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger("[Grammys:logs]")

def Extract()->json:
    """This function load the data from postgres db to dataframe pandas"""
    try:
        df_grammys = pd.read_sql(sql="SELECT * FROM grammys;",
                                con=Pysqlconnect.connection()) #load the data using own module
        logger.log(level=20, msg=f"[{datetime.now()}] - Postgres Connected --- data-loaded --")
        return df_grammys.to_json(orient="records")

    except Exception as err:
        logger.error(f"[{datetime.now()}]: {err}")
        return None

def datetime_transform(stdate:str)->datetime:
    """This function is used for transform one string to datetime format sql"""
    try:
        date = datetime.strptime(stdate, '%Y-%m-%dT%H:%M:%S%z')
        formatdate = date.strftime('%Y-%m-%d') #this is the format to date to sql
        return formatdate
    except Exception as err:
        logger.error(f"[{datetime.now()}] - {err}")

def add_column_published_date(**kwargs:json)->json:
    try:
        logger.log(level=20, msg=f"[{datetime.now()}] - start 'add_column_published_date'")
        ti = kwargs["ti"]
        json_df = json.loads(ti.xcom_pull(task_ids="extract_grammys_task"))
        df = pd.json_normalize(data=json_df)
        df["published_date"] = df["published_at"].apply(datetime_transform)
        logger.log(level=20, msg=f"[{datetime.now()}] - finish task")
        return df.to_json(orient="records")
    except Exception as err:
        logger.error(msg=f"[{datetime.now()}] - {err}")
        err

```

```

1  import pandas as pd
2  from datetime import datetime
3  import logging
4  import json
5
6  logging.basicConfig(level=logging.INFO)
7  logger = logging.getLogger("[Spotify:logs]")
8
9  def Extract()->json:
10     try:
11         logger.log(level=20, msg=f"[{datetime.now()}] - start Extract")
12         df = pd.read_csv("/home/joan/Desktop/etl_workshop_2/Data/spotify_dataset.csv")
13         logger.log(level=20, msg=f"[{datetime.now()}] - data extracted")
14         return df.to_json(orient="records")
15     except Exception as err:
16         logger.error(msg=f"[{datetime.now()}] - {err}")
17
18
19
20  def drop_fake_index_col(**kwargs)->json:
21     try:
22         logger.log(level=20, msg=f"[{datetime.now()}] - start drop_fake_index")
23         ti = kwargs["ti"]
24         json_df = json.loads(ti.xcom_pull(task_ids="extract_spotify_task"))
25         df = pd.json_normalize(data=json_df)
26         try:
27             df.drop(columns='Unnamed: 0', inplace=True)
28         except:
29             logger.log(level=20, msg=f"[{datetime.now()}] - finish drop_fake_index")
30             return df.to_json(orient="records")
31         logger.log(level=20, msg=f"[{datetime.now()}] - finish drop_fake_index")
32         return df.to_json(orient="records")
33     except Exception as err:
34         logger.error(f"[{datetime.now()}] - {err}")
35
36
37  def drop_duplicates(**kwargs)->json:
38     try:
39         logger.log(level=20, msg=f"[{datetime.now()}] - start drop duplicates")
40         ti = kwargs["ti"]
41         json_df = json.loads(ti.xcom_pull(task_ids="drop_fake_index_col_task"))
42         df = pd.json_normalize(data=json_df)
43         df.drop_duplicates(inplace=True)
44         df = df[df["track_id"].duplicated() == False]
45         logger.log(level=20, msg=f"[{datetime.now()}] - finish drop duplicates")
46         return df.to_json(orient="records")
47     except Exception as err:
48         logger.error(msg=f"[{datetime.now()}] - {err}")
49

```

Luego el archivo del dag donde se organizara todo el flujo


```

spotify.py Dags/dags/Spotify_dag/spotify.py/...
11 from store import store_data
12
13
14 default_args = {
15     'owner': 'airflow',
16     'depends_on_past': False,
17     'start_date': datetime(2024, 4, 1),
18     'email_on_failure': False,
19     'email_on_retry': False,
20     'retries': 3,
21     'retry_delay': timedelta(minutes=5),
22 }
23
24 dag = DAG(
25     'Spotify',
26     default_args=default_args,
27     description='Etl process to grammys analysis',
28     schedule_interval=timedelta(days=1),
29 )
30
31 def log_task_execution(task_name, **kwargs):
32     print(f"Executing task: {task_name}")
33
34 with dag:
35     extract_spotify_task = PythonOperator(
36         task_id='extract_spotify_task',
37         python_callable=ExtractSpotify,
38     )
39
40     drop_fake_index_col_task = PythonOperator(
41         task_id='drop_fake_index_col_task',
42         python_callable=drop_fake_index_col,
43     )
44
45     drop_duplicates_task = PythonOperator(
46         task_id='drop_duplicates_task',
47         python_callable=drop_duplicates,
48     )
49
50     extract_grammys_task = PythonOperator(
51         task_id='extract_grammys_task',
52         python_callable=ExtractGrammys,
53     )
54
55     add_column_published_date_task = PythonOperator(
56         task_id='add_column_published_date_task',
57         python_callable=add_column_published_date,
58     )
59
60     drop_columns_task = PythonOperator(

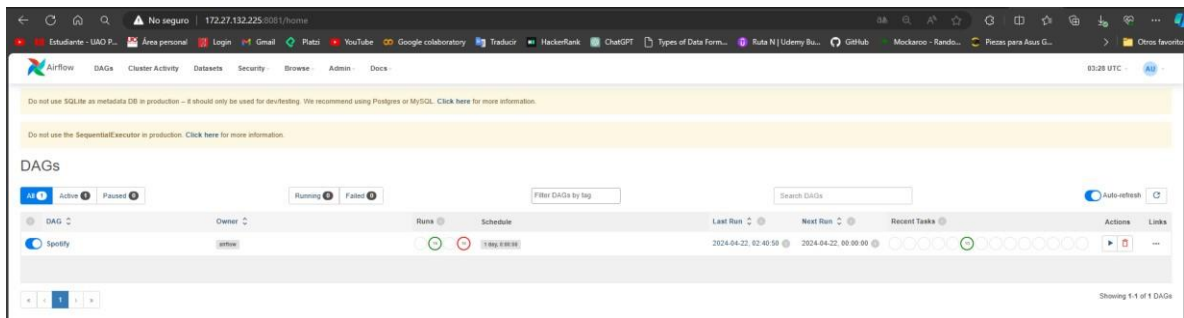
```

```
extract_spotify_task >> drop_fake_index_col_task >> drop_duplicates_task >> merge_dataset
extract_grammys_task >> add_column_published_date_task >> drop_columns_task >> drop_nulls_task >> merge_dataset
merge_dataset >> load_data >> Drive_upload
```

Luego podemos correr el Airflow

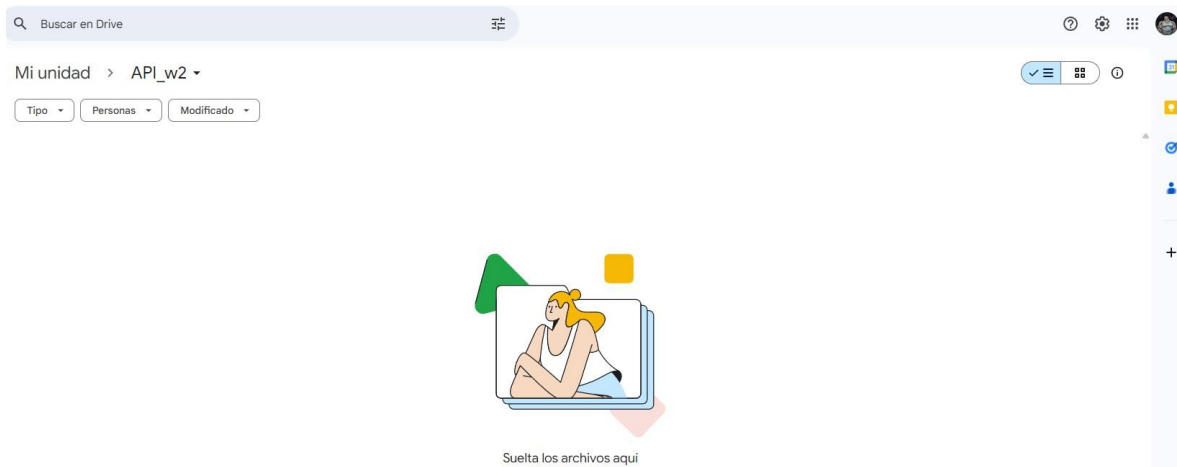
```
(venv) [joan@G21P490000 ~/Desktop/etl_workshop_2] $ airflow standalone
standalone | Starting Airflow Standalone
standalone | Checking database is initialized
INFO [alembic.runtime.migration] Context impl SQLiteImpl.
INFO [alembic.runtime.migration] Will assume non-transactional DDL.
ERROR [airflow.models.dagbag] Failed to import: /home/joan/Desktop/etl_workshop_2/dags/spotify_dag/spotify.py
Traceback (most recent call last):
  File "/home/joan/Desktop/etl_workshop_2/venv/lib/python3.8/site-packages/airflow/models/dagbag.py", line 346, in parse
    loader.exec_module(new_module)
  File "<frozen importlib._bootstrap_external>", line 848, in exec_module
  File "<frozen importlib._bootstrap>", line 215, in _call_with_frames_removed
  File "/home/joan/Desktop/etl_workshop_2/dags/dags/spotify.py", line 11, in <module>
    from Store import Store_data
  File "/home/joan/Desktop/etl_workshop_2/dags/dags/Store.py", line 5, in <module>
    sys.append({"config": "Apach"/"hive"})
AttributeError: module 'sys' has no attribute 'append'
INFO [alembic.runtime.migration] Context impl SQLiteImpl.
INFO [alembic.runtime.migration] Will assume non-transactional DDL.
WARNI [airflow.models.crypto] empty cryptography key - values will not be stored encrypted.
standalone | Database ready
/home/joan/Desktop/etl_workshop_2/venv/lib/python3.8/site-packages/flask_limiter/extension.py:337 UserWarning: Using the in-memory storage for tracking rate limits as no storage was explicitly specified. This is not recommended for prod
uction use. See: https://flask-limiter.readthedocs.io/en/latest/configuring-a-storage-backend/ for documentation about configuring the storage backend.
triggerer |
triggerer |
triggerer |
triggerer |
triggerer | [2024-04-21 21:32:37 -0500] [129882] [INFO] Starting gunicorn 21.2.0
triggerer | [2024-04-21 21:32:37 -0500] [129882] [INFO] listening at: http://[::]:8794 (129882)
triggerer | [2024-04-21 21:32:37 -0500] [129882] [INFO] Using worker: sync
triggerer | [2024-04-21 21:32:37 -0500] [129883] [INFO] Booting worker with pid: 129883
scheduler |
scheduler |
scheduler |
scheduler |
scheduler | [2024-04-21 21:32:37.047-0500] [task_context_trigger.py:63] INFO - Task context logging is enabled
scheduler | [2024-04-21 21:32:37.050-0500] [triggerer_scheduler.py:235] INFO - Loaded executor: SequentialExecutor
scheduler | [2024-04-21 21:32:37 -0500] [129885] [INFO] Starting gunicorn 21.2.0
scheduler | [2024-04-21 21:32:37 -0500] [129885] [INFO] listening at: http://[::]:8793 (129885)
scheduler | [2024-04-21 21:32:37 -0500] [129885] [INFO] Using worker: sync
scheduler | [2024-04-21 21:32:37 -0500] [129886] [INFO] Booting worker with pid: 129886
triggerer | [2024-04-21 21:32:37.670-0500] [triggerer_scheduler.py:174] INFO - Setting up TriggererHandlerWrapper with handler <fileTaskHandler (NOTSET)>
triggerer | [2024-04-21 21:32:37.672-0500] [triggerer_scheduler.py:226] INFO - Setting up logging queue listener with handlers [dedirectStdHandler <stdout> (NOTSET)], <TriggererHandlerWrapper (NOTSET)>]
scheduler | [2024-04-21 21:32:37.673-0500] [scheduler_scheduler.py:796] INFO - Starting the scheduler
scheduler | [2024-04-21 21:32:37.673-0500] [scheduler_scheduler.py:803] INFO - Processing each file at most -1 times
scheduler | [2024-04-21 21:32:37.673-0500] [manager.py:170] INFO - Launched DagFileProcessorManager with pid: 129888
scheduler | [2024-04-21 21:32:37.679-0500] [scheduler_scheduler.py:1595] INFO - Adopting or resetting orphaned tasks for active dag runs
scheduler | [2024-04-21 21:32:37.680-0500] [settings.py:60] INFO - Configured default timezone UTC
scheduler | [2024-04-21 21:32:37.682-0500] [triggerer_scheduler.py:331] INFO - Starting the triggerer
triggerer | [2024-04-21 21:32:37 -0500] [129890] [INFO] Booting worker with pid: 129890
scheduler | [2024-04-21 21:32:37.695-0500] [manager.py:393] WARNING - Because we cannot use more than 1 thread (parsing_processes = 2) when using sqlite. So we set parallelism to 1.
scheduler | [2024-04-21 21:32:37 -0500] [129899] [INFO] Booting worker with pid: 129899
/home/joan/Desktop/etl_workshop_2/venv/lib/python3.8/site-packages/flask_limiter/extension.py:337 UserWarning: Using the in-memory storage for tracking rate limits as no storage was explicitly specified. This is not recomme
nded for production use. See: https://flask-limiter.readthedocs.io/en/latest/configuring-a-storage-backend/ for documentation about configuring the storage backend.
scheduler | [2024-04-21 21:32:39 -0500] [129881] [INFO] Starting gunicorn 21.2.0
scheduler | [2024-04-21 21:32:39 -0500] [129881] [INFO] listening at: http://0.0.0.0:8081 (129881)
```

Con **Airflow standalone**



Verificamos en la web que este el dag que creamos en este caso se llama Spotify

Luego para verificar que funciona corremos y vemos la base de datos y el Drive

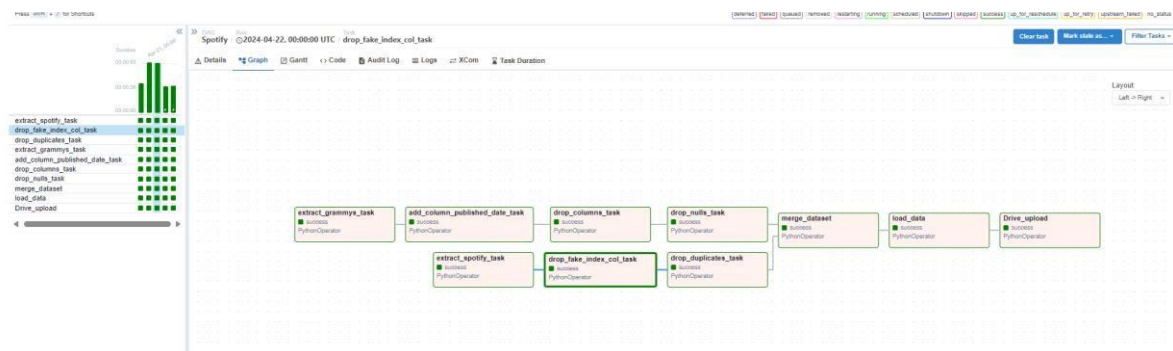


Vemos el drive vacio

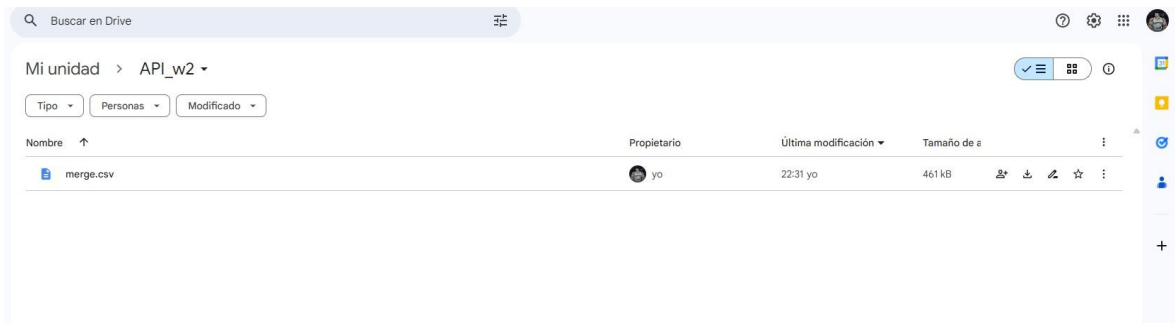
```
workshop_2=# drop table merge;
DROP TABLE
workshop_2=# drop table grammys_clean;
DROP TABLE
workshop_2=# \d
               List of relations
 Schema | Name      | Type  | Owner
-----+-----+-----+-----
 public | grammys   | table | postgres
(1 row)

workshop_2=#
```

Y postgres sin las tablas



Después de correr



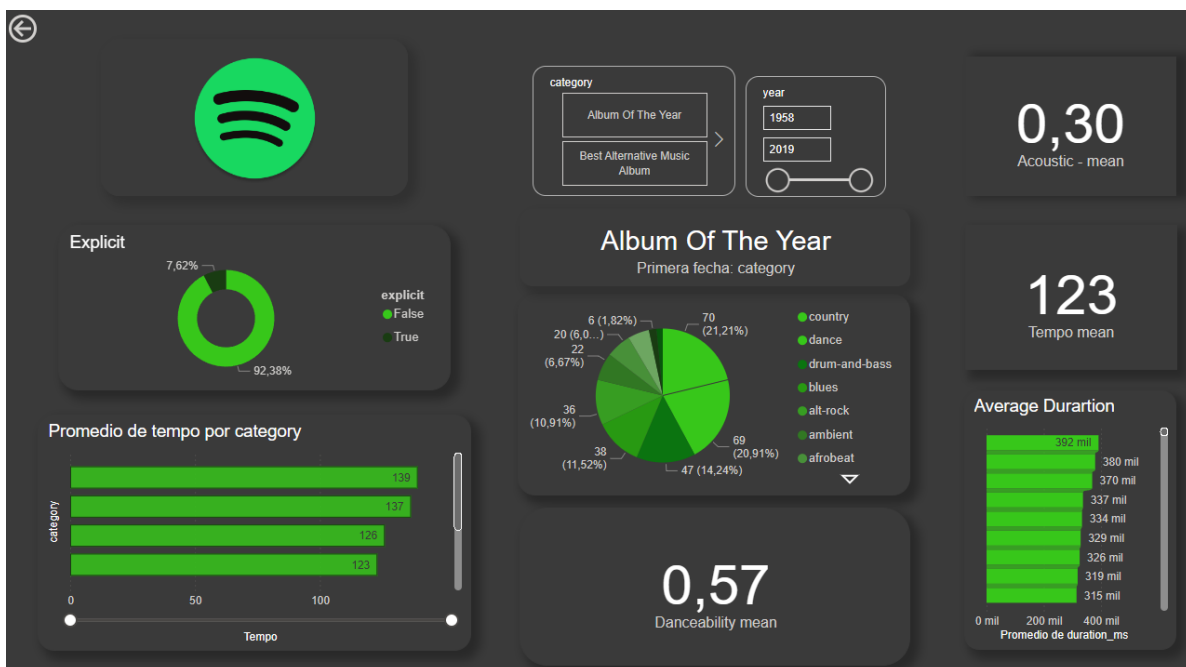
```
Schema | Name | Type | Owner
-----+-----+-----+-----
public | grammys | table | postgres
(1 row)

workshop_2=# \d
               List of relations
Schema | Name | Type | Owner
-----+-----+-----+-----
public | grammys | table | postgres
public | merge | table | postgres
(2 rows)

workshop_2=#
```

Los datos han sido cargados por lo que podemos decir que el pipeline tuvo éxito

Por ultimo podemos ver el tablero en power bi



Al final mi flujo termino siendo algo así

