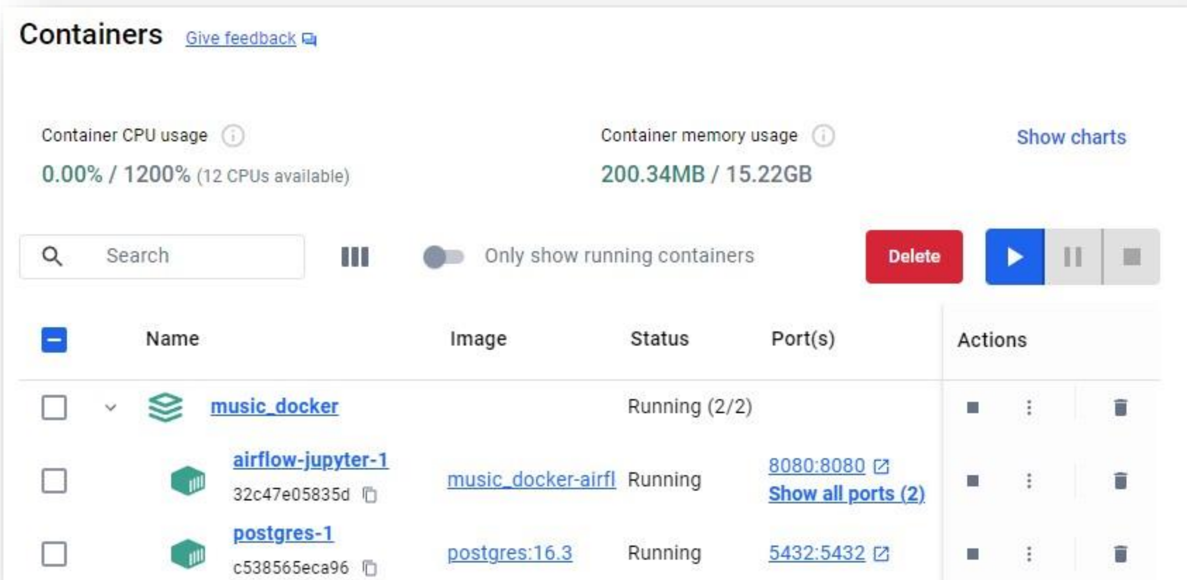
Integrated Music Systems

By Leesa Park, Jay Kim and Robert Dunphy

I. Build Docker Container Instructions

1. Unzip the music_docker.zip folder to a directory called 'music_docker'.
2. In a Terminal/Windows PowerShell go to the 'music_docker' directory.
3. Use the following command to create the Docker image and container:
   $ docker-compose up
4. When the installation completes, you should now see the container and image built in the Docker Desktop.



II. Pipeline Instructions

- **File placements/locations:**

  The following files must be in the following location ("\music_docker\shared"):

  - Final_API_Flask.py
  - Final_API_Flask.ipynb

  Create a 'data' folder inside the '\music_docker\shared' folder.
  The following files must be in the following location ("\music_docker\shared\data"):

  - artists.csv
  - mxmh_survey_results.csv

The following files must be in the following location
(“\music_docker\shared\airflow\dags”):

- lexi_airflow_automation.py

Create a ‘data’ folder inside the ‘\music_docker\shared\airflow\dags’ folder. The following files must be in the following location
(“\music_docker\shared\airflow\dags\data”):

- artists.csv
- mxmh_survey_results.csv

- **Automation and API:**

  Integrating Python for API interactions, Docker for environment consistency, and Apache Airflow for workflow automation, has allowed us to establish a highly efficient and automated data pipeline. This setup not only simplifies the process of data retrieval and storage, but also ensures that our database is consistently refreshed with the latest information. The combination of these tools enhances the reliability, scalability, and maintainability of our data integration system.

  - **Lexi_airflow_automation.py:**

    This script serves as the central component for our data handling process. It integrates data from three different data sources, transforms it as needed, and then executes SQL code to store the results in our designated “music” database. By consolidating these tasks into a single file, the process is streamlined and minimizes dependencies on multiple files, reducing the need for extensive user instructions. In order to setup this DAG, the “final_project_airflow_automation.py” python script needs to be run from within the docker Jupyter terminal.

  - **Final_API_flask.py:**

    To setup the flask API, the Final_API_Flask.py needs to be run from within the docker Jupyter terminal. Here’s the detailed instructions on each step in running the automation and web API:

    Here’s the detailed instructions on each step in running the automation and web API:

    1. To run the automation of the project:

       a. Open two new terminal/Windows PowerShell windows and use the following command in one:

          $ docker exec -it <first few characters from container ID> airflow webserver

       b. In the second window, use the following command:

$ docker exec -it <first few characters from container ID> airflow scheduler

c. In the Docker Desktop, click the port of the airflow-jupyter-1 container.

d. Enter these credentials --> User: admin & password: admin

e. Under the DAG section, go into the ‘final_music_dag_222’ and run the dag.

2. To run the web API:

   a. Inside the docker airflow-jupyter-1 terminal, run the API by using the following command:

      $ python3 “Final_API_Flask.py”

   b. Open and run the ‘Final_API_Flask.ipynb’ file through the docker airflow-jupyter-1 environment.

III. Project Documentation

- **Project Introduction**: The project aims to analyze the most popular songs from Spotify and learn the listeners music preferences and mental health conditions. To achieve this, we will collect datasets from three diverse sources and integrate them into a relational database. The datasets include songs playlist data from Spotify API, which provides 950 songs from the “Top Hits of the 2010s” playlist. The music artist data is sourced from Kaggle and contains 1.4 million musical artists. Additionally, music and mental health survey results data from Kaggle provides information on music taste and mental health responses of the listeners.
- **Datasets Overview**

  i. Song Playlist Dataset (Spotify API)

  - Description: This dataset is a playlist of songs from Spotify that we extracted using via API. While Spotify has over 100+ million songs, we will choose a smaller playlist called “Top Hits of the 2010s” as a dataset which consisted of 950 songs
  - Source: https://developer.spotify.com/documentation/web-api/reference/get-playlists-tracks
  - Sample 10 rows:

```
track_dataframe.head(10)
```

✓ 0.0s

	artist_name	track_name	track_id	popularity
0	Sabrina Carpenter	Please Please Please	5N3hjp1WNayUPZrA8kJmJP	98
1	Billie Eilish	BIRDS OF A FEATHER	6dOtVTDdiauQNBQEDOtlAB	98
2	Chappell Roan	Good Luck, Babe!	0WbMK4wrZ1wFSty9F7FCgu	94
3	Shaboozey	A Bar Song (Tipsy)	2FQrifJ1N335Ljm3TjTVVf	93
4	Benson Boone	Beautiful Things	6tNQ70jh4OwmPGpYy6R2o9	92
5	Hozier	Too Sweet	4ladxL6BUymXlh8RCJJu7T	84
6	Tommy Richman	MILLION DOLLAR BABY	7fzHQizxTqy8wTXwlrgPQQ	89
7	Kendrick Lamar	Not Like Us	6AI3ezQ4o3HUoP6Dhudph3	96
8	Post Malone	I Had Some Help (Feat. Morgan Wallen)	7221xIgOnuakPdLqT0F3nP	95
9	Sabrina Carpenter	Espresso	2qSkIjg1o9h3YT9RAgYN75	100

ii. Music Artists (Kaggle)

- Description: This dataset contains of 1.4 million musical artists from Kaggle. The data was original taken from the MusicBrainz database
- Source: https://www.kaggle.com/datasets/pieca111/music-artists-popularity
- Sample 10 rows:

	mbid	artist_mb	artist_lastfm	country_mb	country_lastfm	tags_mb	tags_lastfm	listeners_lastfm	scrobbles_lastfm	ambiguous_artist
0	cc197bad-dc9c-440d-a5b5-d52ba2e14234	Coldplay	Coldplay	United Kingdom	United Kingdom	rock; pop; alternative rock; british; uk; brit...	rock; alternative; britpop; alternative rock; ...	5381567.0	360111850.0	False
1	a74b1b7f-71a5-4011-9441-d0b5e4122711	Radiohead	Radiohead	United Kingdom	United Kingdom	rock; electronic; alternative rock; british; g...	alternative; alternative rock; rock; indie; el...	4732528.0	499548797.0	False
2	8bfac288-ccc5-448d-9573-c33ea2aa5c30	Red Hot Chili Peppers	Red Hot Chili Peppers	United States	United States	rock; alternative rock; 80s; 90s; rap; metal; ...	rock; alternative rock; alternative; Funk Rock...	4620835.0	293784041.0	False
3	73e5e69d-3554-40d8-8516-00cb38737a1c	Rihanna	Rihanna	United States	Barbados; United States	pop; dance; hip hop; reggae; contemporary r b;...	pop; rnb; female vocalists; dance; Hip-Hop; Ri...	4558193.0	199248986.0	False
4	b95ce3ff-3d05-4e87-9e01-c97b66af13d4	Eminem	Eminem	United States	United States	turkish; rap; american; hip-hop; hip hop; hiph...	rap; Hip-Hop; Eminem; hip hop; pop; american; ...	4517997.0	199507511.0	False
5	95e1ead9-4d31-4808-a7ac-32c3614c116b	The Killers	The Killers	United States	NaN	synthpop; alternative rock; american; new wave...	indie; rock; indie rock; alternative; alternat...	4428868.0	208722092.0	False
6	164f0d73-1234-4e2c-8743-d77bf2191051	Kanye West	Kanye West	United States	United States	synthpop; pop; american; hip-hop; hip hop; ele...	Hip-Hop; rap; hip hop; rnb; Kanye West; seen l...	4390502.0	238603850.0	False
7	5b11f4ce-a62d-471e-81fc-a69a8278c7da	Nirvana	Nirvana	United States	United States	rock; alternative rock; 90s; punk; american; e...	Grunge; rock; alternative; alternative rock; 9...	4272894.0	222303859.0	False
8	9c9f1380-2516-4fc9-a3e6-f9f61941d090	Muse	Muse	United Kingdom	United Kingdom	rock; electronic; synthpop; alternative rock; ...	alternative rock; rock; alternative; Progressi...	4089612.0	344838631.0	False
9	0383dadf-2a4e-4d10-a46a-e9e041da8eb3	Queen	Queen	United Kingdom	United Kingdom	rock; progressive rock; 70s; 80s; 90s; pop-roc...	classic rock; rock; 80s; hard rock; glam rock;...	4023379.0	191711573.0	False

iii. Music and Mental Health Survey results (Kaggle)

- Description: This dataset is based on a survey of music taste and mental health responses from Kaggle. Each row represents the survey results of one individual and includes details on music preferences, music listing frequency, music effects, severity of mental health conditions, etc.

- Source: https://www.kaggle.com/datasets/catherinerasgaitis/mxmh-survey-results
- Sample 10 rows:

	Timestamp	Age	Primary streaming service	Hours per day	While working	Instrumentalist	Composer	Fav genre	Exploratory	Foreign languages	...
0	8/27/2022 19:29:02	18.0	Spotify	3.0	Yes	Yes	Yes	Latin	Yes	Yes	...
1	8/27/2022 19:57:31	63.0	Pandora	1.5	Yes	No	No	Rock	Yes	No	...
2	8/27/2022 21:28:18	18.0	Spotify	4.0	No	No	No	Video game music	No	Yes	...
3	8/27/2022 21:40:40	61.0	YouTube Music	2.5	Yes	No	Yes	Jazz	Yes	Yes	...
4	8/27/2022 21:54:47	18.0	Spotify	4.0	Yes	No	No	R&B	Yes	No	...
5	8/27/2022 21:56:50	18.0	Spotify	5.0	Yes	Yes	Yes	Jazz	Yes	Yes	...
6	8/27/2022 22:00:29	18.0	YouTube Music	3.0	Yes	Yes	No	Video game music	Yes	Yes	...
7	8/27/2022 22:18:59	21.0	Spotify	1.0	Yes	No	No	K pop	Yes	Yes	...
8	8/27/2022 22:33:05	19.0	Spotify	6.0	Yes	No	No	Rock	No	No	...
9	8/27/2022 22:44:03	18.0	I do not use a streaming service.	1.0	Yes	No	No	R&B	Yes	Yes	...

...	Frequency [R&B]	Frequency [Rap]	Frequency [Rock]	Frequency [Video game music]	Anxiety	Depression	Insomnia	OCD	Music effects
...	Sometimes	Very frequently	Never	Sometimes	3.0	0.0	1.0	0.0	NaN
...	Sometimes	Rarely	Very frequently	Rarely	7.0	2.0	2.0	1.0	NaN
...	Never	Rarely	Rarely	Very frequently	7.0	7.0	10.0	2.0	No effect
...	Sometimes	Never	Never	Never	9.0	7.0	3.0	3.0	Improve
...	Very frequently	Very frequently	Never	Rarely	7.0	2.0	5.0	9.0	Improve
...	Very frequently	Very frequently	Very frequently	Never	8.0	8.0	7.0	7.0	Improve
...	Rarely	Never	Never	Sometimes	4.0	8.0	6.0	0.0	Improve
...	Sometimes	Rarely	Never	Rarely	5.0	3.0	5.0	3.0	Improve
...	Never	Never	Very frequently	Never	2.0	0.0	0.0	0.0	Improve
...	Sometimes	Rarely	Sometimes	Sometimes	2.0	2.0	5.0	1.0	Improve

- **Pipeline Overview**

  The data pipeline for the project is designed to efficiently collect, integrate, and visualize data from multiple sources using a PostgreSQL database. It starts with data collection, where datasets are acquired from the Spotify API for the song playlists from 2010, music artists data from Kaggle for the artists names, genres, and country information, and music and mental healthy survey results data from Kaggle for the listeners music taste and mental health conditions. Data integration follows, combining these datasets based on common fields such as artists name and genre, and then loading the integrated data into a PostgreSQL database. In the data processing stage, the data is cleaned, transformed, and aggregated as needed.

- **Data Transformation**

  The data transformation process mostly consisted of data manipulation usings pandas before it is loaded into a PostgreSQL database. The initial data transformation was performed within the “lexi_airflow_automation.py” before the completion of automation with airflow. The following describes the data transformations on the three key datasets.

i. Song Playlist Dataset (Spotify API)

The Spotify data was extracted to a csv file using the Spotify API and associated spotipy python library within the “Spotify_API_FINAL.ipynb” Jupyter notebook and airflow automation scripts. The Spotify dataset comprised of 950 rows with 7 columns and was subsequently reduced to 500 rows after removal of duplicates.

In terms of normalization, a foreign key id for the artist (i.e., ‘artist_id’) was generated using a hash of the artist name. A generate_hash_id function in python takes text values and converts these to a numerical value based on the lower case characters of a text.

We dropped the artist_name column to reduce redundancy which was replaced with the artist_id, which was designed to be joined to the artist dataset.

We also added a year column that was based on the release_date column.

ii. Music Artists (Kaggle)

The artist data source was in csv format and this dataset contained 1,466,083 rows and 10 columns. We identified that there were 957,811 unique artist names. To reduce that size of the data, we removed artists that did not have songs in the Spotify playlist data set and also dropped all duplicate records. This resulted in having 225 rows unique artists comprising of the dataset.

An artist_id was created using the generate_hash_id function mentioned previously.

In terms of normalization, the tags column (i.e., genre) had repeating values of tags that were structured as a list within each rows of data. (i.e., one artist could have many tags such as indie, alternative, rock, etc.). As such, a separate table called artist_genre_mapping was created. The genre names will to genre_id mapping will be explained in the next section.

Further in terms of normalization, the country column had repeating values of country names that were structured as a list within each row of data. (i.e., one artist could belong to two countries). As such, a separate table called artist_countries_mapping was created to convert the list into rows within a table. Further, a separate country table was created of country_id and country name to reduce redundancy of country names within the artist_country_mapping table.

To reduce the number of columns, the final artist table comprised of artist_id, artist_name, listeners and plays.

iii. Music and Mental Health Survey results (Kaggle)

The artist data source was in csv format and this dataset contained 736 rows and 33 columns.   An individual id was assigned based on the index of

the rows in the dataframe. To filter out rows with missing data, we removed rows without an age or music effect.
A new column called music_effect_num was created to store a numerical representation of the music_effects column which comprised of “No effect”, “Improve”, or “Worsen”.
A genre id was created based on the individual’s favourite genre of music. This was created using generate_hash_id function mentioned previously.
In terms of normalization, the fav_genre column consisted of 16 genres and thus had repeating values. As such, a separate table called genre_results containing the genre_id and the associated genre text (i.e., classical, country, EDM, etc.) as created to reduce redundancy. In additional, summarized statistic were included in this table such as mean of age, mean of hours of listening, count of values, etc.

- **Database Schema**
  The database schema is designed to efficiently store and organize data collection from various sources, including Spotify API, Kaggle’s music artist data, and music and mental health survey results data from Kaggle within a PostgreSQL database. The schema features several interconnected tables to maintain normalization and avoid redundancy. The ‘music’ schema should follow the third normal form of database normalization. For example, within the spotify_songs table, there is no redundant information about the artists part from the artist_id. Also, within the artists table, information such as country and genres have been put into separate tables. The tables are listed below:

#	Tables	Description	Row Count	Original Source
1	spotify_songs	This comprises of songs from a playlist on Spotify	468	Spotify API
2	artists	This comprises of musical artist details	225	Musical Artists Kaggle
3	artist_genres_mapping	This was used to normalize the genre information in the artists dataset	745	Musical Artists Kaggle
4	artist_country_mapping	This was used to normalize the countries that are linked to each artist	278	Musical Artists Kaggle
5	country	This was used to normalize country names	33	Musical Artists Kaggle
6	individual_survey	This comprises of survey data on mental health and music	727	Mental Health & Music Kaggle

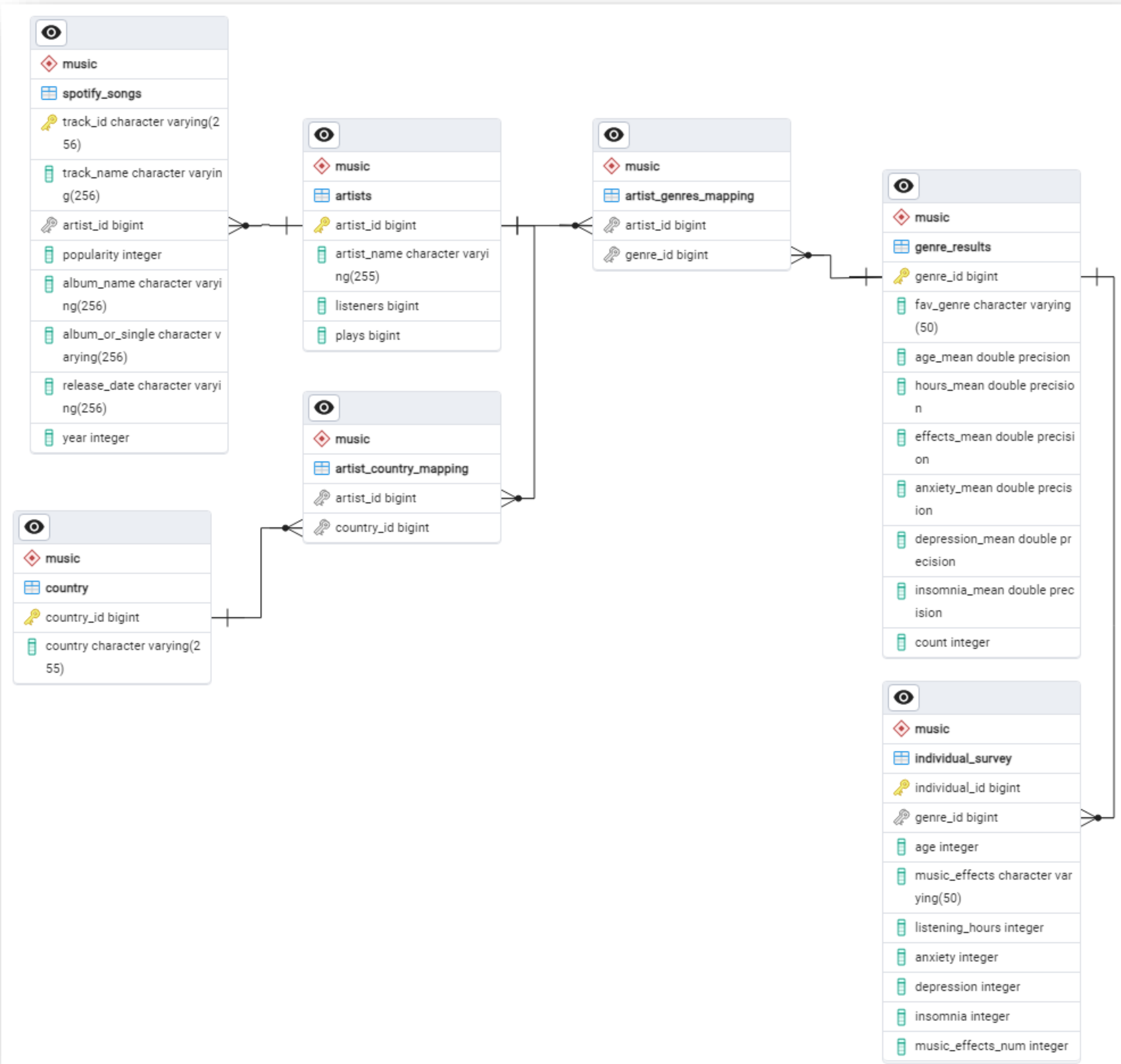
7	genre_results	This is a summary of the survey data on mental health and music by genre	16	Mental Health & Music Kaggle

A listing of the columns, primary keys and foreign keys are listed below:

Table Name	Column	Type	Key
artist_country_mapping	artist_id	bigint	Primary Keys
artist_country_mapping	country_id	bigint	Primary Keys
artist_genres_mapping	artist_id	bigint	Primary Keys
artist_genres_mapping	genre_id	bigint	Primary Keys
artists	artist_id	bigint	Primary Key
artists	artist_name	character varying	
artists	listeners	bigint	
artists	plays	bigint	
country	country	character varying	
country	country_id	bigint	Primary Key
genre_results	age_mean	double precision	
genre_results	anxiety_mean	double precision	
genre_results	count	integer	
genre_results	depression_mean	double precision	
genre_results	effects_mean	double precision	
genre_results	fav_genre	character varying	
genre_results	genre_id	bigint	Primary Key
genre_results	hours_mean	double precision	
genre_results	insomnia_mean	double precision	
individual_survey	age	integer	
individual_survey	anxiety	integer	
individual_survey	depression	integer	
individual_survey	genre_id	bigint	Foreign Key
individual_survey	individual_id	bigint	Primary Key
individual_survey	insomnia	integer	
individual_survey	listening_hours	integer	
individual_survey	music_effects	character varying	
individual_survey	music_effects_num	integer	
spotify_songs	album_name	character varying	
spotify_songs	album_or_single	character varying	
spotify_songs	artist_id	bigint	Foreign Key
spotify_songs	popularity	integer	
spotify_songs	release_date	character varying	

spotify_songs	track_id	character varying	Primary Key
spotify_songs	track_name	character varying	
spotify_songs	year	integer	

Below is the ERD diagram for the schema of our music datasets. In terms of constraints, for the most part all foreign keys must exist with the connected tables. For example, the spotify_songs table has an artist_id that must reside within the artists table. The same is true for the artists_genre_mapping and the artist_country_mapping, where the artist_id foreign keys must exist within the artists table.



- **Automation**

  We used Apache Airflow as our automation tool and used python to create the directed acyclic graph. Our DAG has 11 steps in it, which allows us to create our schema and tables, import all our data and process it, and insert the data into the tables.
  The first step in the automation process is the schema and table creation, in this function we check to see if the schema/tables are inside the database already and if not, create them. The order for table creations matters significantly here, as some of the tables require foreign keys that reference other tables in the schema.
  Our second step is our Spotify API pull, which allows us to get data directly from Spotify about songs in a specific playlist. For this project we used the "Top Hits of the 2010s" playlist, which contains a total of 500 songs, and save it as a csv file which will be pulled later in the automation. This is the step that also coerced us into automating our entire project because the information we gather from this API impacts the preprocessing for every other dataset and it is possible that the owners of these playlists in Spotify to add/remove songs.
  Our next two steps are our load and process data steps. Essentially, both functions perform the same type of work; they both import the data, whether it's from a csv file or a universal dictionary and manipulate it so it can be inserted into the tables correctly. We separated these two functions for debugging purposes as using the universal dictionary was tricky to get correct. Prior to making these automated steps, we figured out all the processing required in a Jupyter notebook, so we weren't as concerned about bugs in our "loading" function.
  Our last seven steps in the automation process involve inserting data into tables. Each function inserts data into a specific table and no function inserts data into multiple tables. Separating all the data inserts makes it significantly easier to debug in the event that the DAG breaks; we can read the logs for one of the automated functions to see exactly what broke our automation process.
  We've scheduled this DAG to run daily; this project does not require live data, and we've decided that once a day should be more than acceptable for this dataset. If we assume that the data gets new additional data added to the files in the future, DAG will run and update the database with new data.

- **API**

  For the API, we used flask as the basis for the API development. We developed two GET requests that will pull information from the PostgreSQL database containing the music schema. The script for these GET requests are in the Final_API_Flask.py file.

The first GET request (i.e., get_pg_data) pulls data from the music.genre_results table. A sample usage of this GET request can be found in the Final_API_Flask.ipynb file as per the screenshot below.

```python
### Sending a GET request for postgres data

# Set the API endpoint URL
url = 'http://localhost:8001/api/get_pg_data'

# Send the GET request to the API endpoint
response = requests.get(url)

# Print the response status code and content
print('Response Status Code:', response.status_code)

print(json.loads(response.content))
```


```
Response Status Code: 200
[{'age_mean': 26.22641509433962, 'anxiety_mean': 4.886792452830188, 'count': 53, 'depression_mean': 4.075471698113207
5, 'effects_mean': 0.7169811320754716, 'fav_genre': 'Classical', 'hours_mean': 2.8820754716981134, 'id': 4285884477,
'insomnia_mean': 3.792452830188679}, {'age_mean': 25.36, 'anxiety_mean': 5.4, 'count': 25, 'depression_mean': 4.32, 'e
ffects_mean': 0.8, 'fav_genre': 'Country', 'hours_mean': 3.42, 'id': 957831062, 'insomnia_mean': 2.72}, {'age_mean': 2
2.083333333333332, 'anxiety_mean': 5.361111111111111, 'count': 36, 'depression_mean': 5.111111111111111, 'effects_mea
n': 0.8333333333333334, 'fav_genre': 'EDM', 'hours_mean': 4.666666666666667, 'id': 100270, 'insomnia_mean': 3.94444444
44444446}, {'age_mean': 25.96551724137931, 'anxiety_mean': 6.689655172413793, 'count': 29, 'depression_mean': 5.137931
```


The second GET request (i.e., get_alltablesjoined_pg_data) pulls data from a SQL query that joins most of the key tables together. A sample usage of this GET request can be found in the Final_API_Flask.ipynb file as per the screenshot below:

```python
### Sending a GET request for postgres data

# Set the API endpoint URL
url = 'http://localhost:8001/api/get_alltablesjoined'

# Send the GET request to the API endpoint
response = requests.get(url)

# Print the response status code and content
print('Response Status Code:', response.status_code)

print(json.loads(response.content))
```


```
Response Status Code: 200
[{'album_name': 'Night Visions', 'artist_name': 'Imagine Dragons', 'country': 'United States', 'fav_genre': 'EDM', 'tr
ack_name': 'Radioactive'}, {'album_name': 'Night Visions', 'artist_name': 'Imagine Dragons', 'country': 'United State
s', 'fav_genre': 'Pop', 'track_name': 'Radioactive'}, {'album_name': 'Night Visions', 'artist_name': 'Imagine Dragon
s', 'country': 'United States', 'fav_genre': 'Folk', 'track_name': 'Radioactive'}, {'album_name': 'Night Visions', 'ar
tist_name': 'Imagine Dragons', 'country': 'United States', 'fav_genre': 'Rock', 'track_name': 'Radioactive'}, {'album_
name': 'Sorry For Party Rocking', 'artist_name': 'LMFAO', 'country': 'United States', 'fav_genre': 'EDM', 'track_nam
e': 'Party Rock Anthem'}, {'album_name': 'Sorry For Party Rocking', 'artist_name': 'LMFAO', 'country': 'United State
s', 'fav_genre': 'Pop', 'track_name': 'Party Rock Anthem'}, {'album_name': 'Sorry For Party Rocking', 'artist_name':
'LMFAO', 'country': 'United States', 'fav_genre': 'Rap', 'track_name': 'Party Rock Anthem'}, {'album_name': 'Sorry For
Party Rocking', 'artist_name': 'LMFAO', 'country': 'United States', 'fav_genre': 'Rock', 'track_name': 'Party Rock Ant
hem'}, {'album_name': 'Sorry For Party Rocking', 'artist_name': 'LMFAO', 'country': 'United States', 'fav_genre': 'Hip
hop', 'track_name': 'Party Rock Anthem'}, {'album_name': '21', 'artist_name': 'Adele', 'country': 'United Kingdom', 'f
av_genre': 'R&B', 'track_name': 'Rolling in the Deep'}]
```