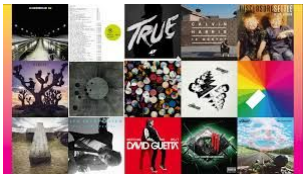
Due: *Thursday, Sept 3rd, 2020*
Team members: Justin, Neethu, Jennie Lee

Project #2: Using ETL to build a music database

Overview: Using numerical and categorical data collected from Spotify's Web API, we set out to transform music-related data into a SQLite database.

Our database was designed to make it easy for different companies to build Spotify playlists to enhance their customer experience across a range of contexts. For instance, this database includes the type of songs you might hear in a doctor's office waiting room to songs you'd expect to hear a DJ mixing from a hotel bar.

Specifically, our database is ideal for all kinds of contexts because the database makes it possible for business executives to search our full collection of songs by danceability, artist, decade and many other factors.

Building our Groovy Database using ETL

Extract: The original source for our data was a set of six CSVs posted on Kaggle that each included different configurations of information from Spotify's Web API. (The documentation is posted here for reference.) To extract the information from these six CSVs, we completed the following tasks:

1. Set up a new repository in GitHub
2. Cloned the new repository to a desktop folder
3. Downloaded the 6 CSVs from Kaggle and placed them in the local GitHub folder
4. Pushed files from the desktop folder to our GitHub repository using the command line
5. Opened Jupyter Lab from the desktop folder using the command line

Transform: Once in Jupyter Lab, we read each of the CSVs we'd extracted from Kaggle and entered the Transform phase of our project, executing the following tasks:

1. After analyzing our data, we decided to focus on cleaning the following three CSVs files because they included information most pertinent to our business goals stated above:
    a. data.csv
    b. Data_by_artists.csv
    c. Data_by_year.csv
2. Using the information from the 3 CSVs above we built six DateFrames in Pandas:
    a. Groovy_songs
    b. Groovy_years

    c. Groovy_artists
    d. Popular_artists
    e. Spa_music
    f. Decades
3. To ensure each DataFrame was user friendly and readable, we removed extraneous columns, cleaned row values, and converted certain metrics into a prettier format. The specifics of those three main tasks are included below:
    a. To keep each DataFrame unique and specific, **we dropped columns from the CSVs** that didn't relate to the song metrics we wanted to focus on and updated column headers to create consistency across each DataFrame.
    b. **We converted the 'duration_ms' column** in milliseconds into the following format 'minutes:seconds' (00:00) to express a song's length in a more readable format.
    c. **We extracted unnecessary brackets and single quotes** in each 'Artists' column to make it again easier for other companies to view the information stored in the DataFrame.
    d. **We rounded up all the metric columns** to a decimal point of 2 so the information appeared cleaner when viewing.
    e. For Decades in particular, **we grouped our Year index by decades** and took the average of each metric column for that decade so users could see at a glance what decades were known for certain song-related factors.

Load: Once we had transformed the information from our original data source into 6 new 'Groovy' DataFrames in Python, we moved into the final phase of our project, the loading phase.

We chose to load our DataFrames into SQLite because the information was uniform and from our market research we learned that the business executives we plan to market our Database to prefer searching information in SQLite because it's familiar to them. Listed below are the tasks we completed to load our DataFrames into SQLite.

1. In Python, we created an engine connecting our DataFrames to SQLite

```
engine = create_engine('sqlite:///Spotify.db', echo=True)
sqlite_connection = engine.connect()
```

2. Once in SQLite, we refreshed the client. This loaded the information from our Python DataFrames into SQLite as 6 individual tables that belonged within our Spotify Database/schema

```
engine = create_engine('sqlite:///Spotify.db', echo=True)
sqlite_connection = engine.connect()
```

The purpose of our Spotify DataBase: We wanted to bring business a new Spotify Database to make it easier to build 'groovy' playlists fit for various environments.