A MILLION ROWS OF MUSIC

ETL Project Report by: Ema, Kayti & Dinesh

Summary: For this project we aimed to take a large music data base in the form of a csv (gathered from https://www.kaggle.com/edumucelli/spotifys-worldwide-daily-song-ranking) which has the top 200 songs across the country captured for every day of 2017 and 2018. It captures the chart position of the song, track name, artist name, number of streams, the Spotify URL, date of the captured data, and the country code. There was some additional data that we wanted to add to this data base. This missing data includes the countries of corresponding country codes and the genre of each song.

Use-cases: The stored dataset can be used to analyze the trending musical numbers of 2017 and 2018 across the globe and identify the most popular genre, artist or the track. With this information we query to find which artist had the most songs on the top 200 chart list and the location they were based out of. We can find out which genre is the most popular based on region. This would be useful in evaluating how music culture spreads across countries and which countries have been dominating the top charts. After we have this information we can begin to start asking why a certain country might be more likely to produce music that would hit the top 200.

The following is a description of the data-sources to pull this information, how we transformed our sources to meet our needs, and how we chose to load out data into different databases. Hope you enjoy!

Extract: The Data-sources:

1.CSV from Kaggle https://www.kaggle.com/edumucelli/spotifys-worldwide-daily-song-ranking

2.Web Scraping https://en.wikipedia.org/wiki/ISO_3166-1

3.APIhttp://ws.audioscrobbler.com/2.0/method=track.getInfo&api_key=b848087a7bcf37ce7a1404dc164ed41d&artist=J%20Balvin&track=Safari&format=json

Transform: *what data cleaning or transformation was required.*

Our music csv was in pretty decent shape upon downloading, however there were a few things we needed to change to meet our needs. First, we needed the column named 'Region' to be renamed 'country_codes' and we wanted the country codes themselves to be in all-caps below is a picture of how we accomplished this:

```
In [9]:
#Clean up the data by renaming the columns
music_data_df = music_data_df.rename(columns = {'Region':'country_codes'})
music_data_df['country_codes'] = music_data_df['country_codes'].str.upper()
music_data_df.head()
```

Out[9]:

	Position	Track Name	Artist	Streams	URL	Date	country_codes
0	1	Reggaetón Lento (Bailemos)	CNCO	19272	https://open.spotify.com/track/3AEZUABDXNtecAO...	1/1/2017	EC
1	2	Chantaje	Shakira	19270	https://open.spotify.com/track/6mICuAdrwEjh6Y6...	1/1/2017	EC
2	3	Otra Vez (feat. J Balvin)	Zion & Lennox	15761	https://open.spotify.com/track/3QwBODjSEzelZyV...	1/1/2017	EC
3	4	Vente Pa' Ca	Ricky Martin	14954	https://open.spotify.com/track/7DM4BPaS7uofFul...	1/1/2017	EC
4	5	Safari	J Balvin	14269	https://open.spotify.com/track/6rQSrBHf7HlZjtc...	1/1/2017	EC

Once we retrieved our countries with the corresponding country codes, the result yielded more data than what we needed:

	COUNTRY	A2 (ISO)	A3 (UN)	NUM (UN)	DIALING CODE
0	COUNTRY	A2 (ISO)	A3 (UN)	NUM (UN)	DIALING CODE
1	Afghanistan	AF	AFG	4	93
2	Albania	AL	ALB	8	355
3	Algeria	DZ	DZA	12	213
4	American Samoa	AS	ASM	16	1-684

Thus we dropped the unnecessary columns, renamed the 'A2' to country_codes, and renamed 'COUNTRY' to 'Country' which returned the following:

```
#clean up the data by renaming
counrty_df = counrty_df.iloc[1:]
counrty_df = counrty_df.rename(columns = {"COUNTRY":"Country", "A2 (ISO)":"country_codes", "A3 (UN)":"A3 (UN)", "NUM (UN)":
                                          "NUM (UN)", "DIALING CODE":"DIALING CODE" })
#drop the columns we don't need
counrty_df= counrty_df.drop(columns=["A3 (UN)", "NUM (UN)", "DIALING CODE"])

counrty_df.head()
```

	Country	country_codes
1	Afghanistan	AF
2	Albania	AL
3	Algeria	DZ
4	American Samoa	AS
5	Andorra	AD

At this point we were ready to inner join our music CSV with the country codes:

```
# merge counrty_df to music_data_df with an inner join
music_df = music_data_df.merge(counrty_df, how="inner", on =["country_codes", "country_codes"])
music_df.head()
```

	Position	Track Name	Artist	Streams	URL	Date	country_codes	Country
0	1	Reggaetón Lento (Bailemos)	CNCO	19272	https://open.spotify.com/track/3AEZUABDXNtecAO...	1/1/2017	EC	Ecuador
1	2	Chantaje	Shakira	19270	https://open.spotify.com/track/6mICuAdrwEjh6Y6...	1/1/2017	EC	Ecuador
2	3	Otra Vez (feat. J Balvin)	Zion & Lennox	15761	https://open.spotify.com/track/3QwBODjSEzelZyV...	1/1/2017	EC	Ecuador
3	4	Vente Pa' Ca	Ricky Martin	14954	https://open.spotify.com/track/7DM4BPaS7uofFul...	1/1/2017	EC	Ecuador
4	5	Safari	J Balvin	14269	https://open.spotify.com/track/6rQSrBHf7HlZjtc...	1/1/2017	EC	Ecuador

At this point we were ready to gather our genre data. We used an API that needed the artist name and the track name to render the music genre. Because our music CSV was a gathering of the top 200 songs, taken every day for two years (2016 and 2017) this meant that there would be double the artists, given that the popularity of a song would trend over multiple days.

We wanted to limit our API calls so we needed to drop all artist and song duplicates. Which resulted in the following:

```
#Collect all the unique track names and unique artists
tracks_df = music_data_df[['Track Name', 'Artist']].groupby(['Track Name', 'Artist']).nunique()
tracks_df = tracks_df[["Track Name","Artist"]].copy()
#Clean up the df so it only includes the columns we want
tracks_df.rename(columns={"Track Name":"track_count","Artist":"art_count"}, inplace=True)
tracks_df = tracks_df.reset_index()
tracks_df.rename(columns={"Track Name":"Track"}, inplace=True)
tracks_df_unique = tracks_df[["Track","Artist"]].copy()
tracks_df_unique.head()
```

	Track	Artist
0	"All That Is or Ever Was or Ever Will Be"	Alan Silvestri
1	"Read All About It, Pt. III"	Emeli Sandé
2	#99	JVG
3	#Askip	Black M
4	#Biziz - feat. Lil Bege	Reynmen

When getting our response, we decided to exclude null items to avoid a messy retrieval:

```
#remove all the info that is N/A
response = [x for x in response if x is not None ]
len(response)
```

After we had the genre dataset ready we were all set to join it to our music csv:

```
# merge the music_data_df with the additional info data frame
music_df_final = music_df.merge(additional_info, how="inner", on =["Artist","Track Name"])
#music_df.head()
```

```
#check out what our data frame looks like
music_df_final.head()
```

	Position	Track Name	Artist	Streams	URL	Date	country_codes	Country	album	Genre
0	40	Don't Let Me Down	The Chainsmokers	4379	https://open.spotify.com/track/0QsvXIfqM0zZoer...	1/1/2017	EC	Ecuador	The Chainsmokers-Japan Special Edition	daya
1	32	Don't Let Me Down	The Chainsmokers	4879	https://open.spotify.com/track/0QsvXIfqM0zZoer...	1/2/2017	EC	Ecuador	The Chainsmokers-Japan Special Edition	daya
2	29	Don't Let Me Down	The Chainsmokers	5602	https://open.spotify.com/track/0QsvXIfqM0zZoer...	1/3/2017	EC	Ecuador	The Chainsmokers-Japan Special Edition	daya
3	32	Don't Let Me Down	The Chainsmokers	5623	https://open.spotify.com/track/0QsvXIfqM0zZoer...	1/4/2017	EC	Ecuador	The Chainsmokers-Japan Special Edition	daya
4	35	Don't Let Me Down	The Chainsmokers	5106	https://open.spotify.com/track/0QsvXIfqM0zZoer...	1/5/2017	EC	Ecuador	The Chainsmokers-Japan Special Edition	daya

When we had our final music csv complete with genres and countries we we ready to load them into our databases.

Load:

MySQL: Since CSVs are already structured tables it would make sense that the SQL database would be the natural go-to for loading. However, when attempted to read in our csv files into MySQL we had trouble using the UTF-8 encoding type because some of the artist data was inputted with Latin characters. To remedy this we used a Latin encoding type.

We decided to used sqlite because of its accessibility. We wanted our dataframe to be accessible to users who did not have MySQL and MongoDB.

In [35]:
```
#Declare data base path
from sqlalchemy import create_engine
database_path = "music.sqlite"
```

In [36]:
```
#Create an engine connection
engine = create_engine(f"sqlite:///{database_path}")
conn = engine.connect()
```

In [32]:
```
#convert the results into a csv file
music_df_final.to_csv("music_data_final.csv")
```

In [37]:
```
#import the results into sql
music_df_final.to_sql('music_data', con=conn, if_exists='replace')
```

In [38]:
```
df_get_Data_from_sql = pd.read_sql("SELECT * FROM MUSIC_DATA" , conn)
```

In [39]:
```
df_get_Data_from_sql.head()
```

Out[39]:

	index	Position	Track Name	Artist	Streams	URL	Date	country_codes	C
0	0	40	Don't Let Me Down	The Chainsmokers	4379	https://open.spotify.com/track/0QsvXIfqM0zZoer...	1/1/2017	EC	E
1	1	32	Don't Let Me Down	The Chainsmokers	4879	https://open.spotify.com/track/0QsvXIfqM0zZoer...	1/2/2017	EC	E
2	2	29	Don't Let Me Down	The Chainsmokers	5602	https://open.spotify.com/track/0QsvXIfqM0zZoer...	1/3/2017	EC	E
3	3	32	Don't Let Me Down	The Chainsmokers	5623	https://open.spotify.com/track/0QsvXIfqM0zZoer...	1/4/2017	EC	E
4	4	35	Don't Let Me Down	The Chainsmokers	5106	https://open.spotify.com/track/0QsvXIfqM0zZoer...	1/5/2017	EC	E

MongoDB proved to be a bit challenging to load because we needed to transpose it into JSON and then load it up jsonified. The chunk of code that proved to be the winning ticket was the following:

```
for index , row in music_df_track.iterrows():
    mongoDoc={}
    mongoDoc["Track"] = row["Track Name"]
    mongoDoc["Artist"] = row["Artist"]
    mongoDoc["Album"]  = row["album"]
    mongoDoc["Genre"]  = row["Genre"]
    mongoDoc["Statistics"] = json.loads( music_df.loc[music_df["Track Name"].\
    ==row["Track Name"]][["Date","Position","Streams","Country"]].T.to_json())
    music_data.insert_one(mongoDoc)
```

After we got the data into MongoDB it proved to be a very beautiful platform to organize and showcase the data: