

## Analyzing Trends Between Top Spotify Songs and Billboard100 Hits

*Adriana Cieslak, Ksenia Dyakova, Megan Gebhart*

---

### Executive Summary

Project Team 7 completed an ETL project to create a dataset that allows for analysis of Spotify song popularity compared to the Billboard Top 100 charts. In addition to analyzing the Artist, Song, and number of weeks on the Top 100 chart, the data also looks at specific song attributes including danciness, liveliness, and duration.

**Extract** *your original data sources and how the data was formatted (CSV, JSON, pgAdmin 4, etc)*

The ETL project focused on two datasets. The Billboard data provides the foundational data while the Spotify data provides better details on the unique attributes of the song.

- [Billboard Charts CSV](#)
  - CSV file available on Kaggle
  - Contains the weekly updated HOT100 section in Billboards charts from 2015 to 2019
  - Available data: Song, Artist, Weeks On #1, Weeks On Chart, Peak Rank
- [Top Spotify songs from 2010-2019 - BY YEAR](#)
  - CSV file available on Kaggle
  - Contains Spotify's most popular songs in the world by year. The dataset includes 13 variables to analyze different song attributes
  - Available data: title, artist, top genre, song, bpm, nrgy, dnce, dB, live, val, dur, acous, spch, pop

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

There were multiple steps taken to clean and transform the data to prepare it for the load stage. The first stage was to remove all unnecessary data. The Billboard data covers 2015-2019 and the Spotify data covered 2010-2019. Since there was no Billboard data to correspond to the 2010-2014 Spotify data we removed it from the data set.

Next we took out the columns we were not going to be using for the analysis. From the Spotify data, we kept the following columns:

1. title
2. artist
3. dnce (Danceability - The higher the value, the easier it is to dance to this song.)
4. live (Liveness - The higher the value, the more likely the song is a live recording)
5. dur (Length - The duration of the song.)

We then updated the remaining columns to either match the Billboard tables, or be easier to understand:

- title > song
- dnce > dance
- live > live\_rec
- dur > duration

From the Billboard list we kept the following columns:

1. Song
2. Artist
3. Weeks On #1
4. Weeks On Chart

We then updated the title to remove spaces and match the case of the Spotify data:

- Song > song
- Artist > artist
- Weeks On #1 > weeks\_1
- Weeks On Chart > weeks\_chart

### **Load the final database, tables/collections, and why this was chosen**

The first step in loading the data was determining the schema within postgres:

```
CREATE TABLE spotify (  
  id SERIAL PRIMARY KEY,  
  song VARCHAR NOT NULL,  
  artist VARCHAR NOT NULL,  
  dance INT NOT NULL,  
  live_rec INT NOT NULL,  
  duration INT NOT NULL  
);
```

```
CREATE TABLE billboard (  
  id SERIAL PRIMARY KEY,  
  song VARCHAR NOT NULL,  
  artist VARCHAR NOT NULL,  
  weeks_1 INT NOT NULL,  
  weeks_chart INT NOT NULL  
);
```

To load the data into the newly created tables, we used the original Jupyter Notebook and sqlalchemy to create a connection to postgres:

```
#Create database connection
db_conn = "postgres:password@localhost:5432/etl_db"
engine = create_engine(f'postgresql://{db_conn}')

#Confirm tables
engine.table_names()

['billboard', 'spotify']

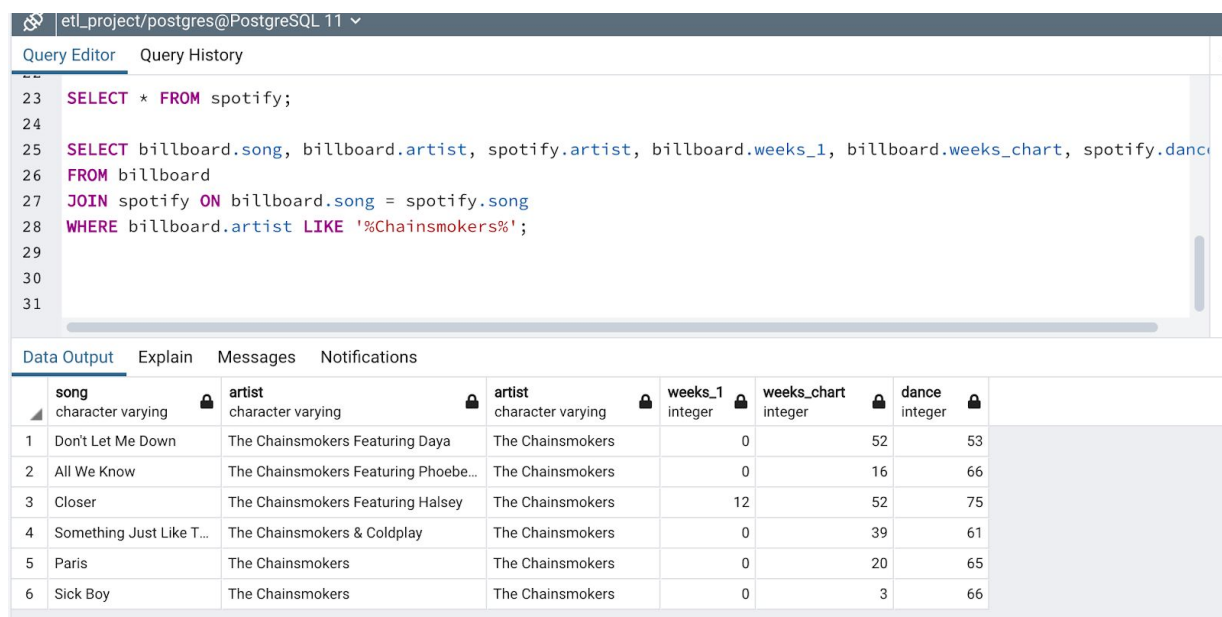
#Load DataFrames into database
billboard_df.to_sql(name='billboard', con=engine, if_exists='append', index=False)

songs_df.to_sql(name='spotify', con=engine, if_exists='append', index=False)

#Confirm data has been added by quering the spotify table
pd.read_sql_query('select * from spotify', con=engine).head()
```

Once the connection was verified and completed, our project team ran a series of tests to ensure the data was imported correctly and identify any potential issues. While running these tests, we discovered that any analysis using artist would require a match function in order to combine data from two different files without losing any of the information due to spelling errors or mismatches within the files themselves.

In this example, we are able to query an artist and get information about certain songs from both tables, although the artist's name may not always be consistently formatted within the data itself.



The screenshot shows a PostgreSQL query editor window titled 'etl\_project/postgres@PostgreSQL 11'. The 'Query Editor' tab is active, displaying a SQL query that joins the 'billboard' and 'spotify' tables. The query filters for artists whose names contain 'Chainsmokers'. Below the query editor, the 'Data Output' tab shows the results of the query as a table with 8 columns: song, artist, artist, weeks\_1, weeks\_chart, and dance. The results list 6 songs by The Chainsmokers.

	song	artist	artist	weeks_1	weeks_chart	dance
	character varying	character varying	character varying	integer	integer	integer
1	Don't Let Me Down	The Chainsmokers Featuring Daya	The Chainsmokers	0	52	53
2	All We Know	The Chainsmokers Featuring Phoebe...	The Chainsmokers	0	16	66
3	Closer	The Chainsmokers Featuring Halsey	The Chainsmokers	12	52	75
4	Something Just Like T...	The Chainsmokers & Coldplay	The Chainsmokers	0	39	61
5	Paris	The Chainsmokers	The Chainsmokers	0	20	65
6	Sick Boy	The Chainsmokers	The Chainsmokers	0	3	66

## Summary

After finalizing the ETL process for our two data types, it is now possible to run various types of analysis to uncover trends behind the most popular music across Spotify and Billboard's Hot 100. Because the data includes various attributes, years, and rankings the data can be aggregated and sliced in many different ways. This will allow those using the dataset to answer a multitude of questions--without having to sort through unnecessary data thanks to the cleanup process in the extract stage.