March 9, 2020

… So I Can Sing Along to Katy Perry, that’s why!

For my ETL project, I chose to make a database of song lyrics. For my first data source, I chose a csv that I downloaded from Kaggle. This csv contained a list of all the songs that made it into the top ten songs on Spotify for the years 2010-2019.

The first step was loading the downloaded csv into the Jupyter notebook, then dropping NAN values and selecting only the title, artist, year and genre columns.

From this dataframe (songs\_df) I created a dataframe of artists with the value counts of how many top-ten Spotify songs they had in that time frame, perhaps showing the relative popularity of each artist in my list.

Knowing that the music industry is a constant ebb and flow of trends, I then grouped the artists’ value counts by year, showing the popularity of each artist by year (according to Spotify).

My main objective, however, was to produce a catalog of lyrics for this list of songs. From my songs\_df, I extracted the ‘artist’ and ‘title’ columns, then added an empty column ‘lyrics’. After experimenting with many APIs, I found some success with apiseeds, which has music lyrics that can be searched by artist and track title (among other data). After many tries, I discovered that there was one row in my lyrics\_df that triggered a JSONDecodeError. I could not control for this type of error using try and except, so I took out the row with the index 245 and saved it in the variable no\_245.

I ran an iterrows loop to fill in the ‘lyrics’ column using calls to apiseeds. This filled 337 of the 603 rows of lyrics\_df. I then extracted the rows from lyrics\_df that were still missing lyrics and called it empty\_df. I appended no\_245 to empty\_df, so it could go through the next API. Musixmatch’s API delivers JSONP data, which is a string, rather than a dictionary, so it was harder (for me) to isolate the text of the lyrics. I found a formula posted on <https://github.com/tohyongcheng/jsonp2json.git> that effectively converted JSONP string data into a JSON dictionary, so the response was searchable by index.

I applied this convert function within an iterrows loop that took the response from each call to musixmatch and converted it to a JSON format, allowing me to isolate the text of the lyrics by dictionary indexing, and inserting them in to the ‘lyrics’ column. Musixmatch’s API filled almost all remaining rows with lyrics, six songs still left

After eliminating empty rows from lyrics\_df, I concatenated empty\_df and lyrics\_df, to make all\_lyrics. The lyrics unfortunately retained ‘\n’ characters, which interfere with reading the lyrics as you sing along, so it was decided that the ‘\n’ characters must go. For this, I used re.sub in an iterrows loop to remove ‘\n’ from the ‘lyrics’ column.

Having the data I needed, I created a new database in Mongo called music\_db. I added four collections to theis database: songs, artists, lyrics, and words. I posted the data from songs\_df, artists\_df, and all\_lyrics into their respective Mongo collections. I used the find function to validate that the data had been posted to Mongo, and found that the lyrics appeared in Mongo with ‘\n’ between the lines of the songs, even though they had been removed from the all\_lyrics dataframe before the data was loaded into Mongo. Is this something I could have avoided by using SQL?

Last but not least, in order to analyze the language used in these music lyrics, I split the lyrics into a list called wordcount, which I then turned into a dataframe of words on which I could apply value counts to figure out the frequency with which each word was used, then created a dataframe of words and their frequency of use that I posted to the words collection in Mongo.