Cooper Levine, Halle Michael, Isabelle Seelig
https://github.com/cooperlevine/CHI-SI-206-Final-Project.git

Original Goals

We initially planned to use Apple Music, Spotify, and SoundCloud APIs to collect data on the monthly listeners for a specific artist and number of albums for a specific artist (we planned on gathering length of each album, rating, number of tracks, number of plays per album, etc). After calculating the averages of the data from these three separate APIs, we planned on juxtaposing the calculated data for a specific artist relative to each individual streaming platform.

Achieved Goals

APIs/ websites used: iTunes, Spotify, YouTube

Throughout our project, we changed our objective to comply with what was readily available through the APIs. Apple Music and Soundcloud's APIs did not provide us with the data that we needed, like the popularity of each track and artist, in order to perform our desired calculations. In order to extract related data from different types of sources, we ended up using Spotify, iTunes and YouTube to collect information on the artist Drake. For the iTunes API, we measured the prevalence of collaborators in Drake's music by counting the number of times a supporting artist was featured on his songs or Drake was featured on another artist's song, collecting the ID of the specified song, the title of the songs, the features in the song, etc. For the Spotify API, we accumulated album IDs for all of Drake's albums and used the IDs to garner data on each individual track from all of his albums, gathering track IDs, titles, and popularity in one table and album IDs, titles, track count, and popularity in another table. Lastly, for the YouTube API, we extracted data on Drake's music tracks available on YouTube and found the popularity of his videos according to views, likes, dislikes, comments. Ultimately, by taking the totality of this data into account, we curated a multitude of visualizations that display the popularity of Drake's music across varying platforms including iTunes, Spotify, and Youtube as well as the popularity of featured artists that Drake collaborates with. These visualizations work to help new and old Drake fans find universally popular tracks/albums that they want to listen to or discover new artists that appear on Drake's music and create similar music.

Problems that we faced:

Problem #1: A major problem we had trouble fixing was that the iTunes API would not extract solely Drake's songs because it was grabbing data from every artist that includes the word Drake in their name (such as Drake Bell, Nick Drake, etc.), rather than just the 'Drake' we were looking for.

- Solution: In order to fix this issue, we added multiple parameters such as entity=song and attribute=allArtistTerm within the base url (shown below) in order to narrow down our

search results from the iTunes API to exclusively include songs and ensure that the artist term is exclusively limited to ‘Drake’ and nothing more.

```
15      base_url = f"https://itunes.apple.com/search?term={artist}&entity=song&attribute=allArtistTerm&limit=200"
```

Problem #2**:** We had a similar issue with the Vimeo API because the platform utilizes user-posted videos and not necessarily songs posted directly by the artist. It was difficult to differentiate between Drake, the artist we were looking to investigate, and other users named Drake who posted unusable tracks.

- Solution: To combat this issue with Vimeo, we found it was much easier to use the YouTube API in order to gather Drake’s tracks from a playlist that includes all of the available songs posted by Drake on YouTube. Once we fixed this error, we found that for the Youtube API, it was difficult to get album titles for Drake along with the song titles, views, likes, dislikes, etc. because the album titles were not included in the API results. So, instead, we utilized the Youtube API to calculate the data for each track found on Youtube as well as the Spotify API to pull out the album titles from Spotify for each of those tracks.

Problem #3: Another primary problem that we faced was ensuring that a maximum of 25 rows were added to the Spotify database during each execution of the code. Since the spotify database involves two tables, Albums and Tracks, it was difficult to organize how each album and all the accompanying tracks for that album would be entered into the database.

- Solution: The first solution to this problem came with the realization that we can enter one Drake album and all of its tracks into the database at a time by utilizing the Spotify API on two separate occasions, firstly to pull all of Drake’s album IDs from Spotify and secondly to gather all the tracks for one particular Album ID. After collecting all of the Album IDs from the API, if there are any album IDs in the database, our programming selects the albums IDs already in the database and locates the next available ID that is not yet in the database. Using that album ID, we find a maximum of 25 tracks within that album using a parameter known as ‘limit’ within the base url of the API call, and then we add the tracks to the Tracks table and album to the Albums table.

Calculations

Spotify API:

13 lines (13 sloc) | 371 Bytes

```
Album Title,Album Popularity,Average Track Popularity
Certified Lover Boy,96,79.57
Dark Lane Demo Tapes,80,66.36
Care Package,73,58.12
So Far Gone,43,23.61
Scorpion,85,66.08
More Life,83,65.64
Views,84,67.1
What A Time To Be Alive,45,30.0
If You're Reading This It's Too Late,77,63.35
Nothing Was The Same,72,59.23
Take Care,46,30.89
Thank Me Later,69,55.43
```

iTunes API:

35 lines (35 sloc) | 456 Bytes

```
Artist,Count
JAY-Z,2
Static Major,1
Kendrick Lamar,1
Birdman,1
Rick Ross,2
Lil Wayne,4
Hell Ya Fucking Right,1
The Weeknd,1
Rihanna,2
Nicki Minaj,2
Alicia Keys,1
The-Dream,1
Young Jeezy,1
T.I.,1
2 Chainz,2
Jhene Aiko,1
Majid Jordan,1
Detail,1
JAY Z,1
PARTYNEXTDOOR,4
Travi$ Scott,1
Pimp C,1
Future,2
Wizkid,1
Giggs,2
Black Coffee,1
Sampha,1
Quavo,1
Kanye West,1
Young Thug,1
The Throne,1
Trey Songz,1
Bun B,1
21 Savage,1
```

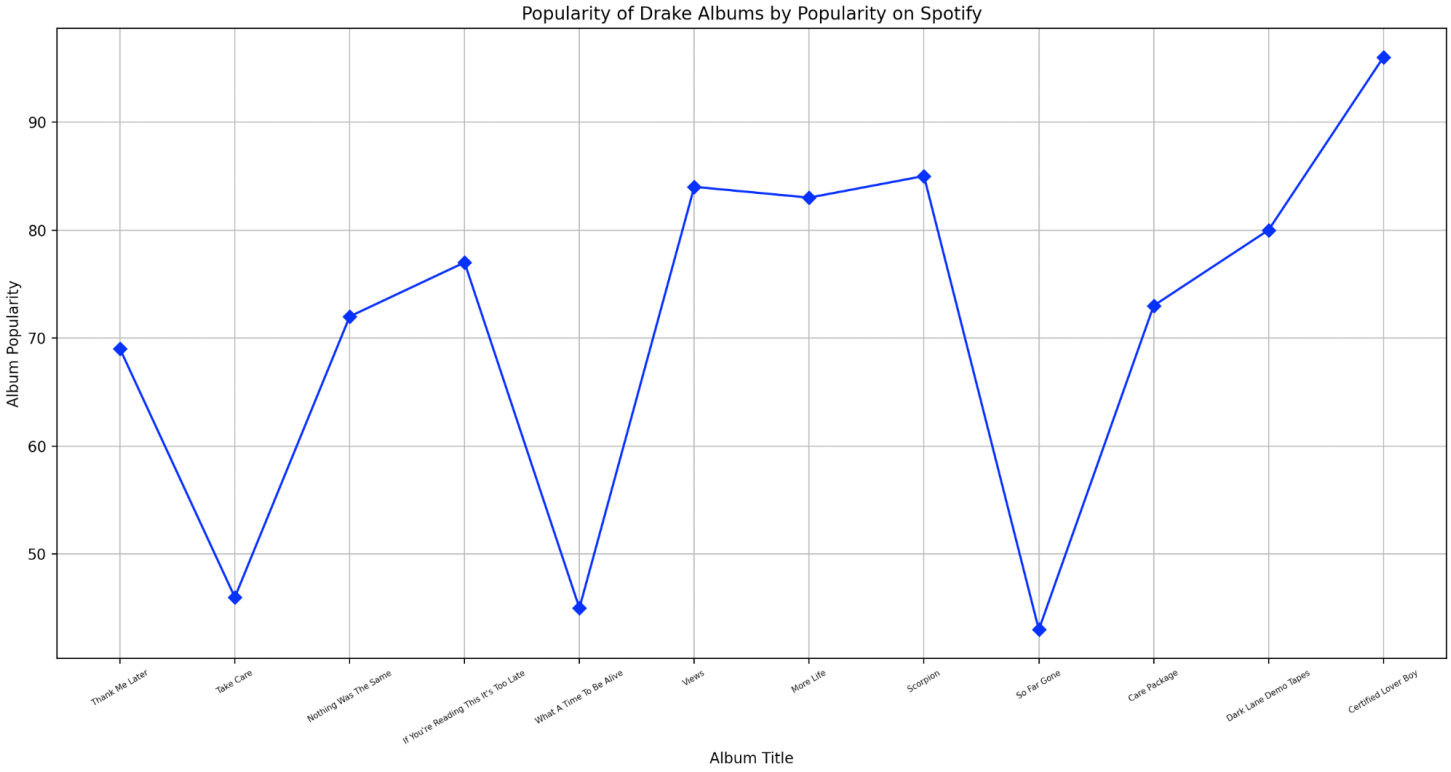
YouTube API:

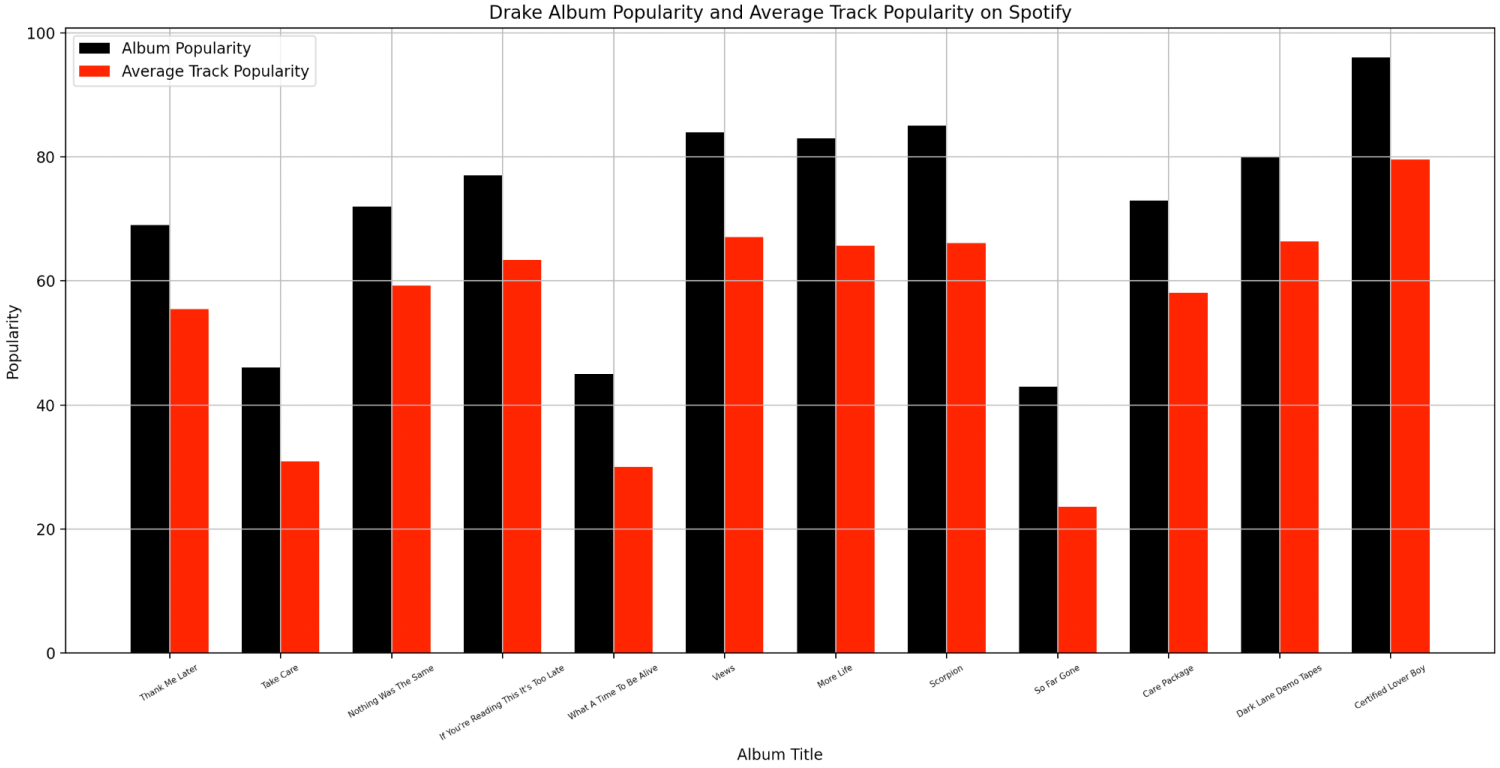
178 lines (178 sloc) | 3.59 KB

```
Track Title,Number of Likes and Comments
Champagne Poetry,240434
Papi’s Home,95912
Girls Want Girls,256998
In The Bible,201999
Love All,155248
Fair Trade,537199
Drake,1071836
TSU,123498
N 2 Deep,90350
Pipe Down,82107
Yebba’s Heartbreak,57054
No Friends In The Industry,134679
Knife Talk,356591
Race My Mind,53567
7am On Bridle Path,63458
Fountains,78389
Get Along Better,56992
You Only Live Twice,71043
IMY2,61317
Fucking Fans,33862
Nonstop,1045594
Elevate,53295
Emotionless,84405
God's Plan,14827055
```

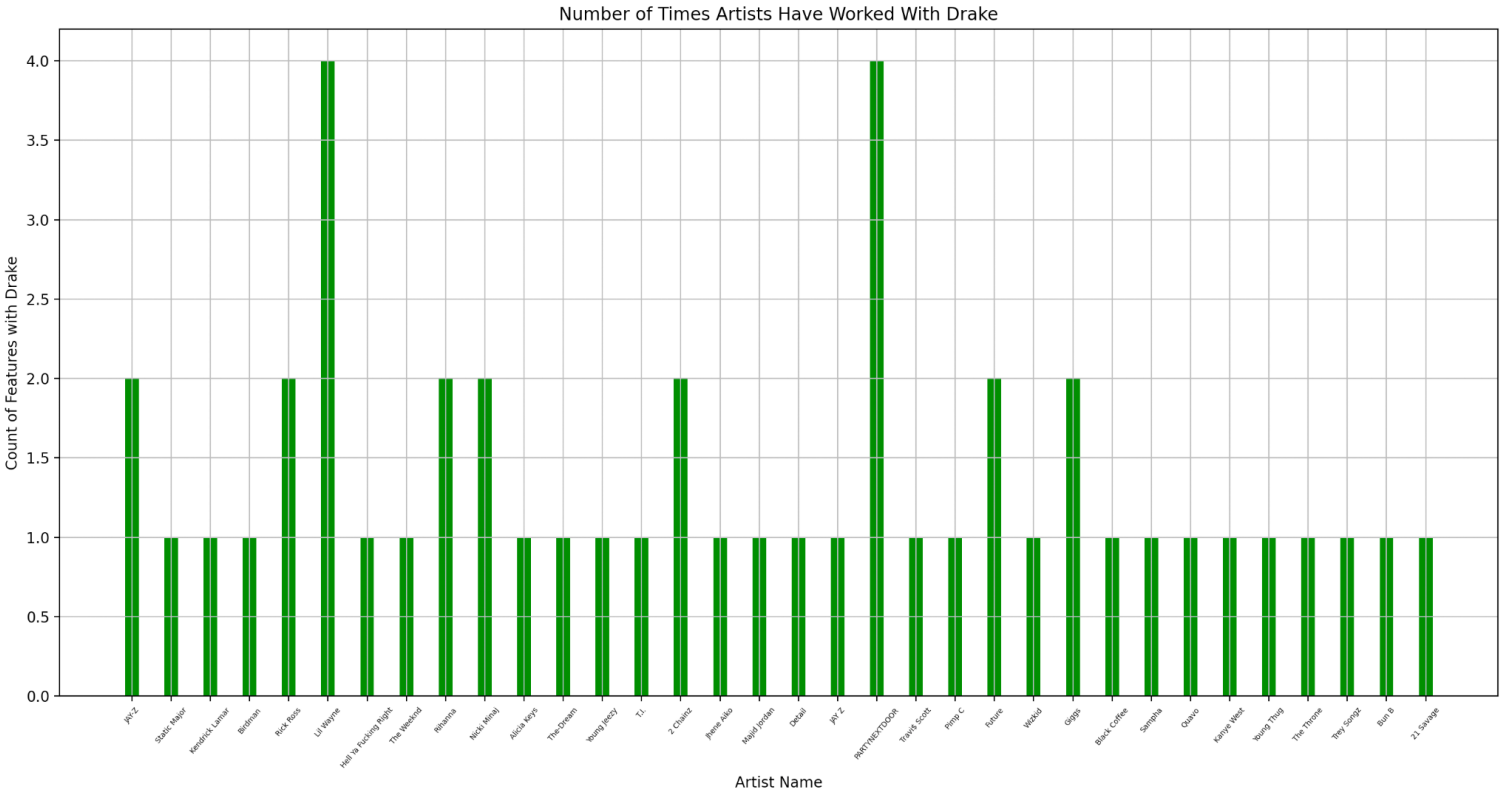
Visualizations

Spotify API:

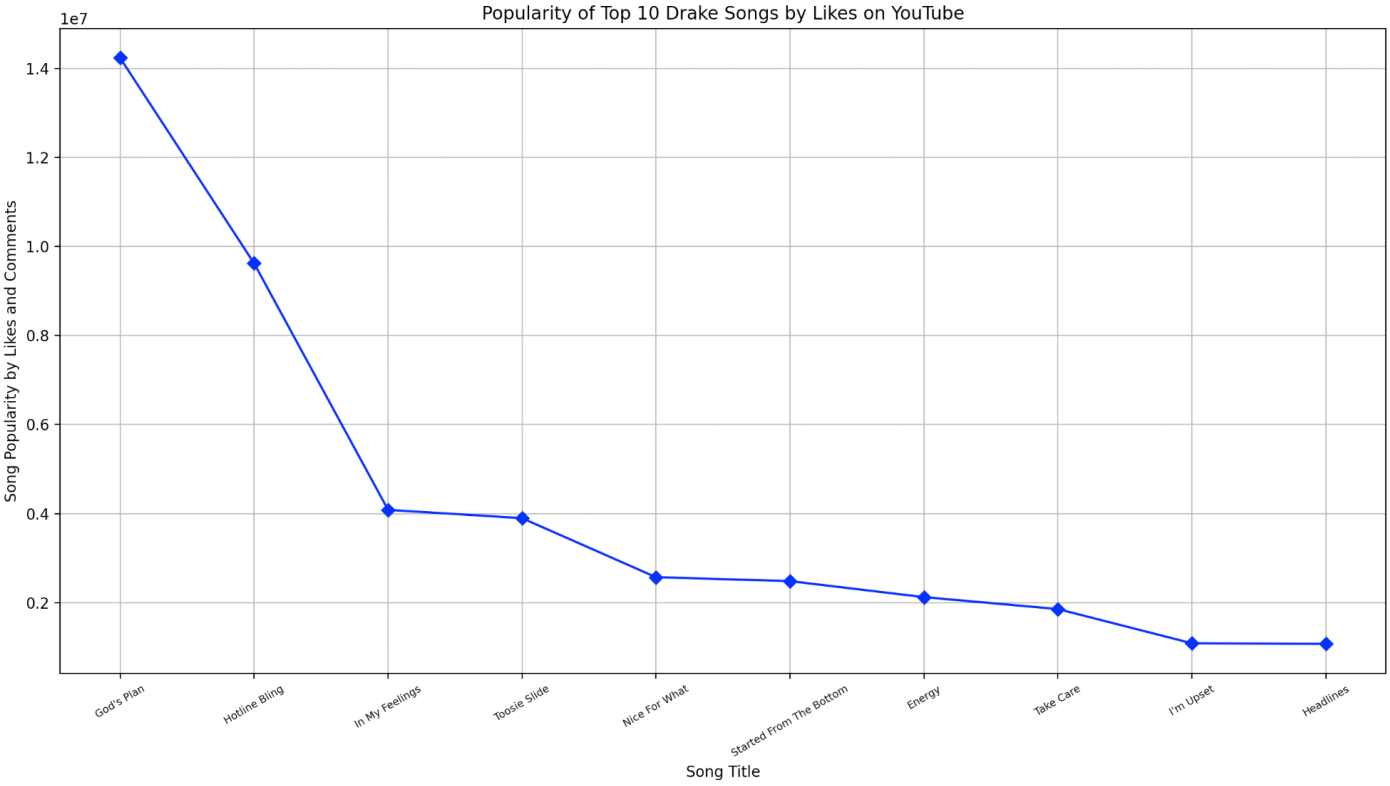




iTunes API:



YouTube API:



Instructions for running code

SpotifyAPI.py:

- First, run the program. If the program returns “Spotify token has expired. Generate new token to continue” follow the instructions at the top of the file in order to generate a new token to use the Spotify API. After generating a new token, run the code several times until the program returns “You have reached the maximum number of Drake albums.” This means you have collected all available data for all of Drake’s albums and tracks from Spotify, and that data can be found in the database ‘Drake.db’. If you want to start building the table from the beginning, uncomment Section 1 in the main function (lines 195 and 196) and run the program again. Then, make sure to re-comment lines 195 and 196 again after running the first execution (after grabbing the first Drake album). If you want to see visualizations for the data in the database, uncomment Section 2 in the main function (lines 237 and 238)

iTunesMusicAPI2.py:

- First, run the program. Then open up the application SQLite on your computer and click on “Open Database”. Find the file that says “Drake.db” and open it in SQLite. Run the code at least 4 times in order to generate at least 100 items into the database. Uncomment the visualization in the main function in order to display it after you are satisfied with the amount of items in the database.

youtubeAPI.py:

- First, run the program. If the program has an error with “tracks” that means you must generate a new Spotify token. Follow the same instructions to generate a new token. After generating a new token, run the code. The data can be found in the database ‘Drake.db’ - 25 items will be added to the table at one time. If you want to see the visualization, uncomment the def makeLinePlot under the main function.

Code Documentation

SpotifyAPI.py:

- `def get_album_and_track_popularity(token, album_id):`
    - This function takes in the spotify token and a Drake album_id that has yet to be added to the database. It then accesses the Spotify API using the album_id and collects data regarding that album, including the album title, album popularity, and the track IDs for each track in that album. The function limits the maximum amount of track IDs collected to 25. Then, the function accesses the Spotify API again, but using each track’s unique track ID, gathering data for each track in the

album such as the track title and track popularity. It finally returns a list of two dictionaries: the first dictionary containing a key of the album title with a value of the album's popularity and the second dictionary containing keys of the track titles from the album, each with a value of a tuple containing the track's popularity and ID.

- `def setUpDatabase(db_name):`
    - This function takes the database name 'Drake.db' as a parameter, sets up the database, and returns cur and conn.
- `def setUpAlbumsTable(popularity_data, album_id, cur,  conn):`
    - This function sets up the Albums table for the database set up above ('Drake.db'). It takes in popularity_data, which is the data returned by the get_album_and_track_popularity function, album_id which is the Spotify ID of the album yet to be added to the database, cur, and conn as parameters. Using the popularity_data, this function gathers the album ID, album title, album popularity, the track count, and calculates the average track popularity for that album. It then inputs all of those values into the Albums table in the database for a particular album.
- `def setUpTracksTable(popularity_data, cur, conn):`
    - This function sets up the Tracks table for the database set up above ('Drake.db'). It takes in popularity_data, cur, and conn as parameters. Using the popularity_data, the function gathers the track IDs, the track titles, track popularities, and album ID. It then inputs all those values into the Tracks table in the database for each track in the album. Since, the amount of tracks collected is limited to a maximum of 25 tracks in the get_album_and_track_popularity function, a maximum of 25 items are always entered into the database.
- `def getAverageTrackPopularityCalculation(cur, file):`
    - This function accepts parameters of cur as well as the file that the calculation will be stored in ('AverageTrackPopularity.txt'). It then uses a JOIN to select the titles of albums, track count of albums, the popularity of all the tracks in the album, and the album popularity from Tracks join Albums on the basis that the album IDs are equal in both databases. Using this data, the function calculates the average track popularity for an album, adding up the sum of all the track popularities for tracks in an album and dividing it by the track count for that album. It then writes a csv file containing columns of Album Title, Album Popularity, Average Track Popularity for all the Drake albums in the database.
- `def createLinePlot(cur):`
    - Taking in cur as a parameter, this function selects the album titles and album popularity from the Album Table and creates a line plot in order to depict the

current popularity of Drake albums from his earliest to most recent album according to the popularity indicator on Spotify.

- `def createBarPlot(cur):`
    - Taking in cur as a parameter, this function selects the album title, album popularity, and average track popularity from Albums and creates a bar plot with two different sets of data, album popularity and average track popularity, in order to depict the album popularity vs. the average popularity of the tracks in that album for each Drake album from earliest to most recent according to the popularity indicator on Spotify.
- `def main():`
    - This function first gathers all of Drake's album IDs from the Spotify API into a list. It then creates Albums.db and selects all the album IDs from the Albums table in the database if there is any data in the database. Comparing both sets of albums IDs, the function determines which album ID has yet to be added to the database. If all album IDs have been added to the database, it prints "You have reached the maximum number of Drake albums." Otherwise, it runs the functions above for the specific album ID that has not yet been added to the database. After collecting the data, one can then uncomment the visualization functions in order to see the graphs.

iTunesMusicAPI2.py:

- `def getAlbumfeatures(artist):`
    - This function takes in an artist as a parameter, in this case 'Drake'. It then accesses the iTunes API using the search term 'Drake' and returns a json response containing 200 of Drake's songs.
- `def cleanName(name):`
    - This function accepts the parameter name, or the artist(s) name(s) returned from a json response for a specific song, and returns a list of the cleaner version of the name(s) without being separated by commas (',') or ampersands ('&').
- `def getArtists(song):`
    - getArtsits accepts song as a parameter, or the json formatted dictionary containing data about a song collected from the iTunes API, and collects all the artists that

either wrote the song or appeared on that song as a feature. It works to return a set of the artists that appear on the song using the cleanName function to clean up the names of artists attached to unnecessary characters. The *set* module allows us to store multiple items (like the names of the artists) into the singular variable *ret*.

- `def getData(json):`
    - This function accepts the json formatted dictionary returned by the getAlbumfeatures function and parses through the data containing information on Drake's song. For each song in the json response, the function counts the number of times an artist collaborated with Drake on a song and collects data such as the song title, the song's iTunes ID, and the artist of the song using the getArtists function. It returns a dictionary of two dictionaries, with the first dictionary – artists – containing keys of artist names and values of the artist count and the second dictionary – ids – containing keys of the song name and values of a list containing the song ID and artist of the song.
- `def setUpDatabase(db_name):`
    - This function takes the database name 'Drake.db' as a parameter, sets up the database, and returns cur and conn.
- `def setUpFeatures(data, id_list, cur, conn):`
    - This function takes in parameters of data, or the data returned from getData, id_list, or a list of song IDs already within the database, cur, and conn. Using the data the function collects lists of the artist names, the track names, the track ids and the track features. It then checks the id_list to see what IDs are already in the database and chooses the 25 next available IDs to input into the table known as 'Features'. The function inputs the track ID, title, artist, and features into the table.
- `def getFeatureCount(json):`
    - This function first takes the items from the database and puts them into a list. Then it appends the items from the "features" column in the database to the list called new_ftrs. Next, it creates a dictionary and writes the CSV file with the count of the number of times an artist is featured on Drake's song or Drake is featured in another artist's song. If the column reads "N", the count does not go up. However, if the a
- `def main():`

- The main function runs the various functions referenced above, first the getAlbumfeatures function, then the getData function, the setUpDatabase function, the setUpFeatures function, and the getCount function. It also creates an id_list of all the Drake track IDs that have already been entered into the database.

youtubeAPI.py:

- `def setUpDatabase(db_name):`
  - This function takes the database name ‘Drake.db’ as a parameter, sets up the database, and returns cur and conn.
- `def addLikes(cur, file):`
  - This function adds up all of the likes and comments and creates a csv file that displays the combined total in one column.
- `def get_album(name):`
  - This function gets the album of a song after searching for the name on Spotify. It used the data from the Spotify API to extract the album name. If the artist is Drake, it returns the album name.
- `def extractName(name):`
  - This function uses conditional statements to get and print the song title from the youtube video separated from unnecessary characters such as ‘Drake-’ or ‘feat.’ It used the split() function to break apart and locate the title.
- `def getItems():`
  - This function gets all of the data from each video in the playlist that is being used. It requests 50 tracks/videos at one time, since 50 is the maximum number of videos that can be requested, and adds the video data to a list. Then, while there are more videos in the playlist to be requested, the function requests the remaining videos and adds their video data to the list. It then returns the list.
- `def getVideoData(items):`
  - This function extracts the wanted information from each video. It takes in the parameter items, or the list returned from getItems, and gets the title and id and then requests video statistics for the specific video. The function takes into account whether the video is private, and allows for the code to continue if so.

The function then extracts the views, likes, dislikes, and number of comments, and then appends them to the data array. It then returns the data.

- `def setUpYouTubeTable(data, track_ids, cur, conn):`
    - This function sets up the Youtube table for the database set up above (‘Drake.db’). It takes in the parameter of data returned in the getVideoData function, track_ids which represent the track/video IDs that have already been entered into the database, as well as cur and conn. It ensures that only 25 tracks/videos from the youtube playlist are accepted into the database at a time and enters 25 new tracks into the Youtube table with columns such as track ID, title, album, views, likes, dislikes, and comments.
- `def makeLinePlot(cur):`
    - Taking in cur as a parameter, this function selects the song titles and likes from the Tracks Table and creates a line plot in order to depict the current number of likes from his top 10 songs.

- `def main():`
    - This function first gathers all of the information from a playlist containing all of Drake’s audio videos in the Youtube API through the getItems function. It then gathers all the information about the videos/tracks from the playlist through the getVideoData function, sets up the database through the setUpDatabase function, and then puts all the information into a table.

Resources

Date	Issue Description	Location of Resource	Result
12/2	Confusion with using the Spotify API and how we were supposed to structure the parameters in order for the authorized token to work	https://stackoverflow.com/questions/60425737/spotify-api-error-400-only-valid-bearer-authentication-supported	Used the parameters headers = {"Authorization": "Bearer " + token} in the requests.get method in order for the authorized token to successfully work
12/3	When calculating average track popularity for an album, we did not know how to format a float to only return to the hundredth decimal place. Otherwise, the float was way too long.	https://stackoverflow.com/questions/455612/limiting-floats-to-two-decimal-points	Using float(format(popularity_sum/count, '.2f'))), we were able to limit a float to solely two decimal places
12/10	When the data was being returned, the code was returning songs from a playlist called “Drake’s Playlist” that contained songs not made by Drake. This was because we were using the wrong playlist ID.	https://developers.google.com/youtube/v3/docs/playlistItems/list	In order to solve the issue, we decided to make our own YouTube playlist containing all of Drake’s available audio videos on YouTube.
12/13	When we each tried to push our changes to GitHub we accidentally got an error because of a merge conflict	https://stackoverflow.com/questions/24357108/updates-were-rejected-because-the-remote-contains-work-that-you-do-not-have-loca	We ultimately had to create a new GitHub repo to store all of our files and learned to utilize git pull before pushing files to GitHub.