SI 206 Final Report

By Julia Kaplan, Emma Tenner, & Yonit Robin

Github: https://github.com/tennere/SI206FinalProject

I. Project Goals

A. *Original Goals*

The original goal of this project was to create visualizations that would compare physical activity, music, and weather on a multitude of factors through collecting data from the Fitbit, Spotify, and Weatherbit APIs. However, within the first couple of days of beginning the project, we concluded that this project was no longer in all of our best interests and we completely switched the project idea. We determined that we were all most interested in the music element of our original idea and switched the project to center various factors relating to music.

B. *Revised Goals*

Once we switched our project, our goal was to find the association between various factors from the Billboard Hot 100 Chart, Spotify, and Deezer. To access songs on Spotify and Deezer, our goal was to web scrape the Hot 100 Chart to extract the top 100 songs. We wanted to extract more information from the Spotify and Deezer APIs and create visualizations comparing various factors from each music service.

C. *Achieved Goals*

Once we revised our project idea, we achieved every goal that we set out to complete. We successfully web scrapped the Billboard Hot 100 Chart. With the data from the chart, we extracted information from both of the APIs. Once our database successfully collected all of the data, we created three visualizations comparing factors from all three of the sources that we used.

II. Problems Faced

A. *Problem #1:* Using Spotipy, we couldn't directly access artist information, like number of followers on Spotify, number of likes on Spotify, etc., for each artist on the original Top 100 Artist Table.
   a. To fix this, we had to switch from web scraping the Top 100 Artists to web scraping the Top 100 Songs.

B. *Problem #2:* Once we switched to the Top 100 Songs, the next issue was accessing the Spotify IDs for each song which was required for every Spotipy function we wanted to use.
   a. We used a search() function and had to make sure to specifically access the ID from all of the information given.

C. *Problem #3:* We realized that for each song title, there were multiple IDs, so we had to figure out which was the correct one.
   a. We realized through trial and error that the one that worked was the one under the 'tracks' key.

D. *Problem #4:* We were going to use the YouTube API, but with every module we imported, we kept encountering an "exceeded quota" error.
    a. After trying many different ways of importing the YouTube API, we decided to switch the Deezer API, because it was simpler to navigate and did not have a quota

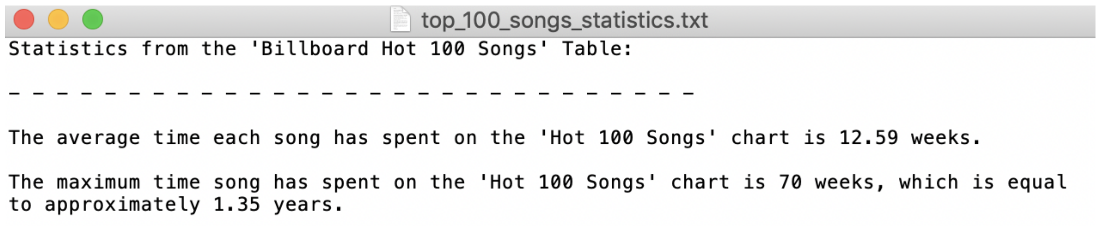
E. *Problem #5:* For whatever reason, one of our visuals was not displaying at all, despite it being the correct code and everything (even the GSIs / IAs couldn't figure it out).
    a. One of the GSIs suggested just literally copy and pasting everything into the other visual file and for whatever reason, it worked!

F. *Problem #6:* We realized that some of the artists did not have data on Deezer and that we would just be given data for a random artist whose name was similar.
    a. We decided to input their information as "N/A" in the Deezer table so that these outliers would not impact our data

G. *Problem #7:* Many of the artists of the top 100 Billboard songs consisted of multiple artists and used strings such as "Featuring", "Duet", or "&". This was a problem because Deezer does not include these details, and therefore, we were unable to fetch data for many of the artists.
    a. We decided to create if statements and strip those strings from the artists, and ultimately fetch data for only the first artist of the duo.

III. Calculation Files

A. *Billboard Hot 100 Calculations*

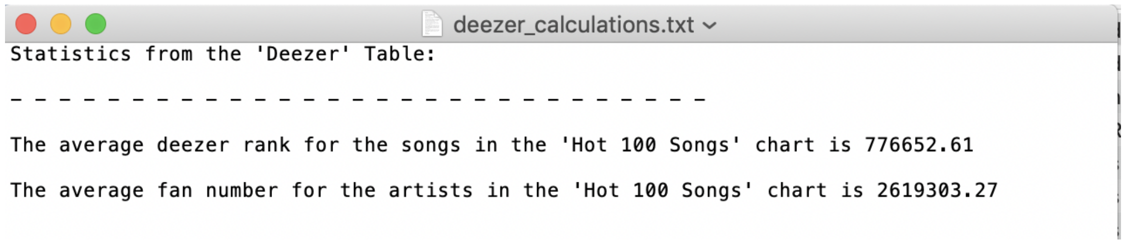


top_100_songs_statistics.txt

```
Statistics from the 'Billboard Hot 100 Songs' Table:

- - - - - - - - - - - - - - - - - - - - - - - - - - - - -

The average time each song has spent on the 'Hot 100 Songs' chart is 12.59 weeks.

The maximum time song has spent on the 'Hot 100 Songs' chart is 70 weeks, which is equal
to approximately 1.35 years.
```

a. Within the hot_100_songs.py file, Julia calculated the average time each song consecutively spent on the "Hot 100 Songs" chart. Julia also calculated the maximum time a song spent on the "Hot 100 Songs" Chart and also provided this statistic in years. The data was saved in a .txt file.

B. *Deezer Calculations*



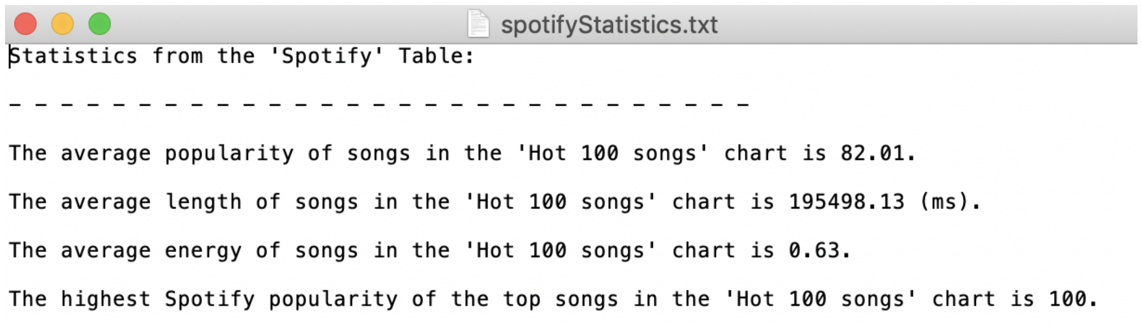
deezer_calculations.txt

```
Statistics from the 'Deezer' Table:

- - - - - - - - - - - - - - - - - - - - - - - - - - - - -

The average deezer rank for the songs in the 'Hot 100 Songs' chart is 776652.61

The average fan number for the artists in the 'Hot 100 Songs' chart is 2619303.27
```

a. Using the deezer.py file, Yonit calculated the average Deezer rank for songs found in the “Hot 100 Songs” chart. For reference, the value is out of 1,000,000, and the higher the value, the more popular the song is.

b. Secondly, Yonit calculated the average fan number for the artists that were featured in the “Hot 100 Songs” chart. The data was saved in a .txt file.

C. Spotify Calculations

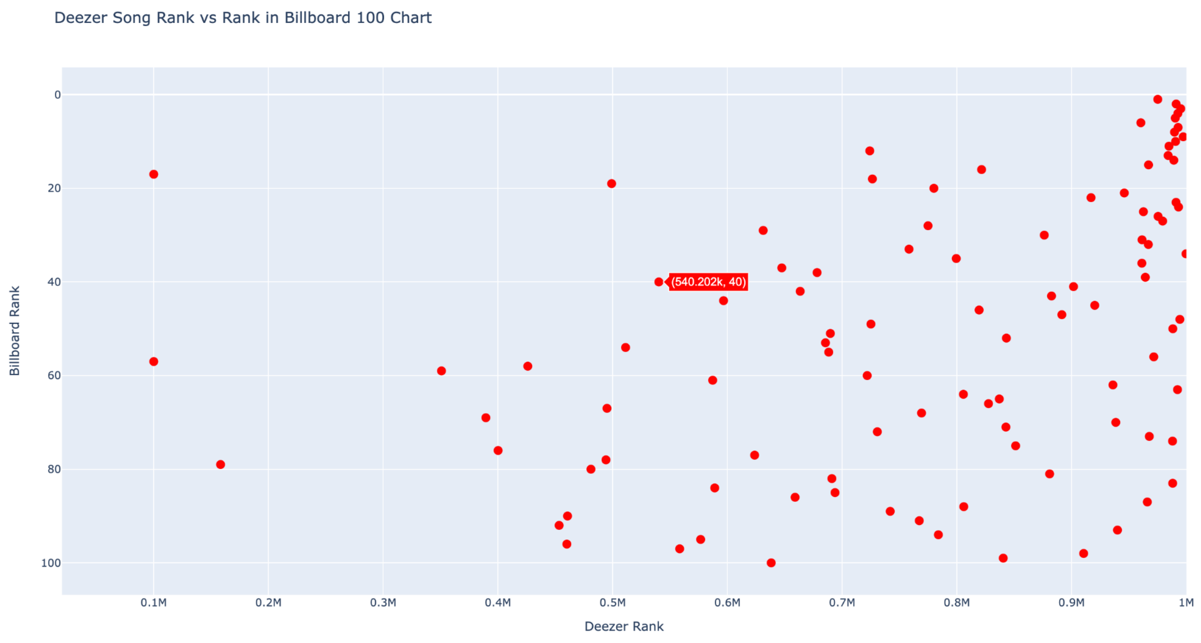


```
Statistics from the 'Spotify' Table:

- - - - - - - - - - - - - - - - - - - - - - - - - - - - -

The average popularity of songs in the 'Hot 100 songs' chart is 82.01.

The average length of songs in the 'Hot 100 songs' chart is 195498.13 (ms).

The average energy of songs in the 'Hot 100 songs' chart is 0.63.

The highest Spotify popularity of the top songs in the 'Hot 100 songs' chart is 100.
```

a. Using the spotifyInfo.py file, Emma calculated the average popularity of songs that were featured in the “Hot 100 Songs” chart. Emma also calculated the average length of each song (in ms), the average energy of the songs, and the highest popularity of the top songs (out of 100). The data was saved in a .txt file.

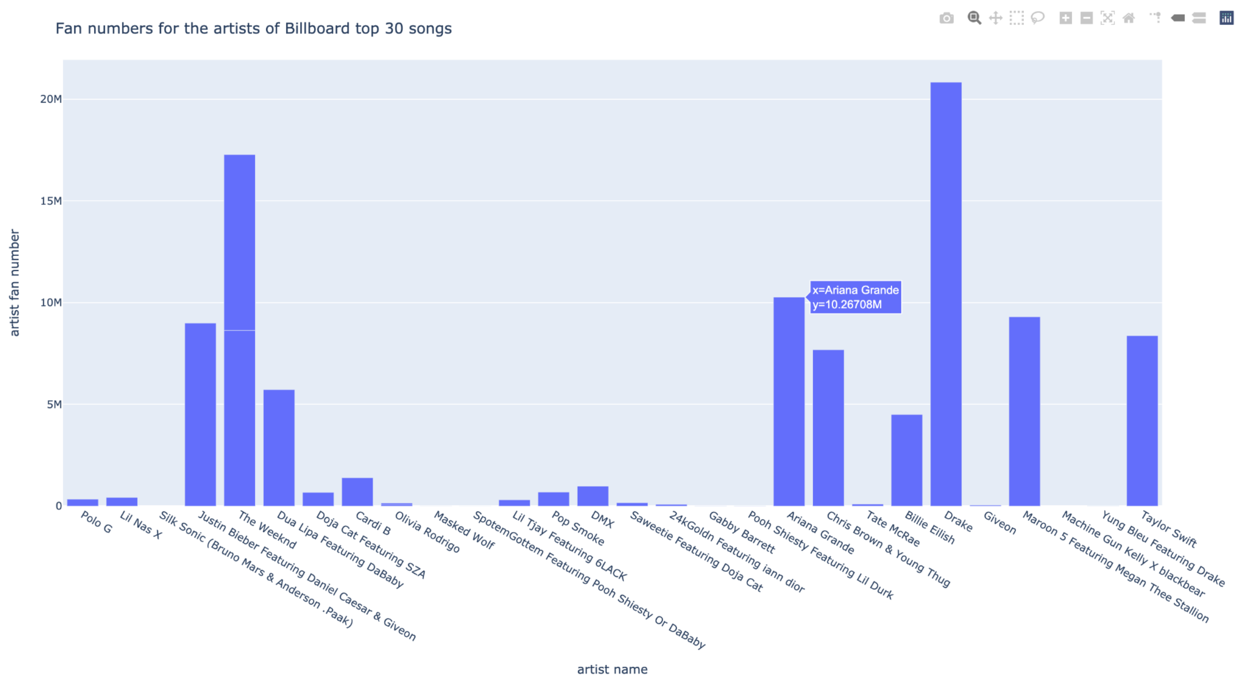
IV. Visualizations Created

A. Visualization #1:



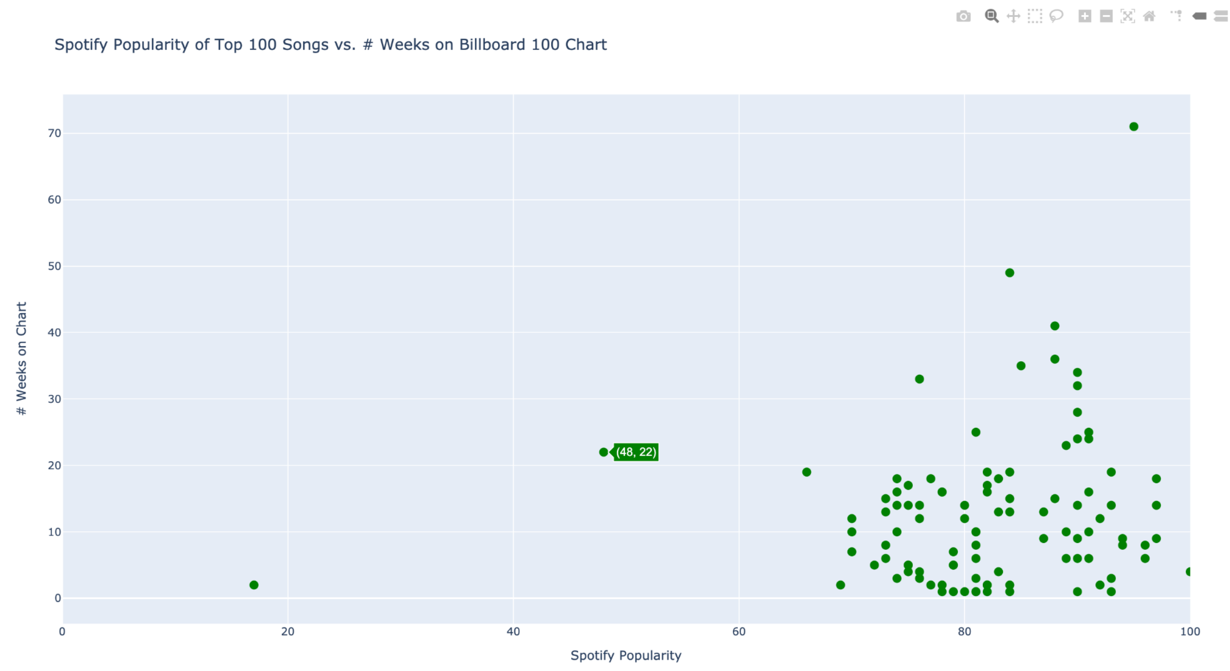
a.

B. Visualization #2:



a.

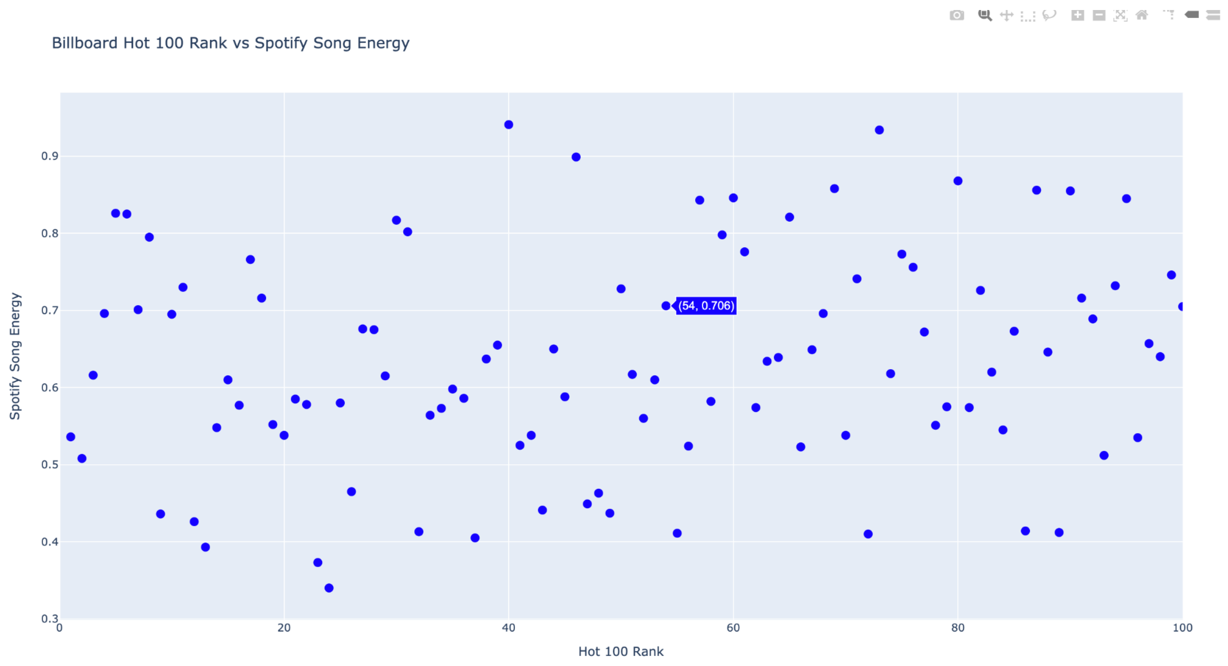
C. Visualization #3:



a.

D. Visualization #4:

a.



V. Instructions to Run the Code

A. Order of Files Execution

a. Repeat 4 times: (1) hot_100_songs.py, (2) deezer.py, (3) spotifyInfo.py

b. Finally, run visuals.py.

B. Project Download Instructions

a. Search https://github.com/tennere/SI206FinalProject

b. Click the green "Code" button

c. Copy the HTTPS web URL

d. Open terminal

e. Locate the file you want to store the project (use `CD`)

f. Type: `git clone https://github.com/tennere/SI206FinalProject.git`

g. Open Visual Studio Code and open the SI206FinalProject folder

C. hot_100_songs.py Execution

a. If you already have a "top_100_songs.db" file, delete it from the computer

b. If you already have a "top_100_songs_statistics.txt" file, also delete it from your computer

c. Run the file in Visual Studio Code

d. Look in the SI206FinalProject folder, there will be a "top_100_songs.db" file. Open the file.

e. Once you see that the first 25 songs have been added to the database, you can now run the deezer.py file.

f. * After the fourth run of the code, top_100_songs_statistics.txt file will be created.

D. *deezer.py Execution*
   a. Now, run the deezer.py file.
   b. The file will take a couple of minutes to run. You will know that the data has been added to the database when you see the “done filling 25 rows” message in the terminal.
   c. A deezer_calculations.txt file will be created. Every time that you run the code, it will be updated.
   d. Once you see this message, move to the spotifyInfo.py file.

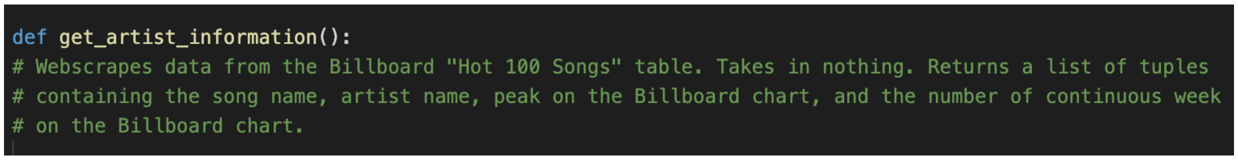
E. *spotifyInfo.py Execution*
   a. First, make sure that you have spotipy and pandas imported on your computer, using pip install.
   b. Next, run the file.
   c. A spotifyStatistics.txt file will be created. Every time that you run the code, it will be updated.
   d. This file will run much faster than the Deezer file. Open the “top_100_songs.db” file to see the spotipy data. Once you see the new 25 entries, move to the next step.

F. *visuals.py Execution*
   a. Once there are 100 entries in each of the three tables, it is time to run the visuals.py file.
   b. First, make sure that you have plotly and numpy downloaded to your terminal, using pip install
   c. Next, run the file.
   d. Your default internet browser will open and you will be presented with four different visualizations, each in their respective tab.

VI. Code Documentation

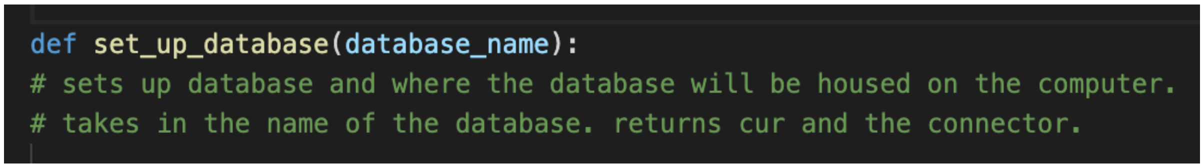
A. *hot_100_songs.py Documentation*

a.



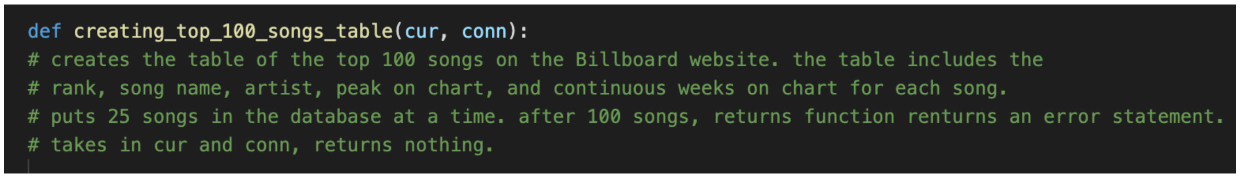
```
def get_artist_information():
# Webscrapes data from the Billboard "Hot 100 Songs" table. Takes in nothing. Returns a list of tuples
# containing the song name, artist name, peak on the Billboard chart, and the number of continuous week
# on the Billboard chart.
```

b.

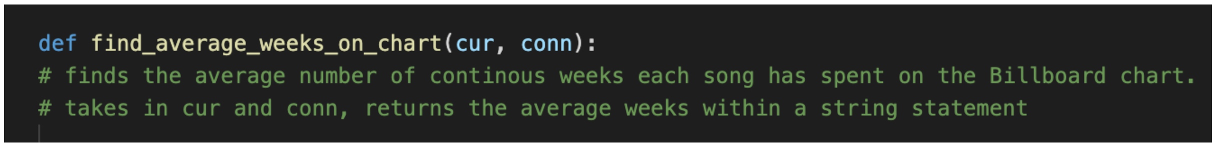


```
def set_up_database(database_name):
# sets up database and where the database will be housed on the computer.
# takes in the name of the database. returns cur and the connector.
```

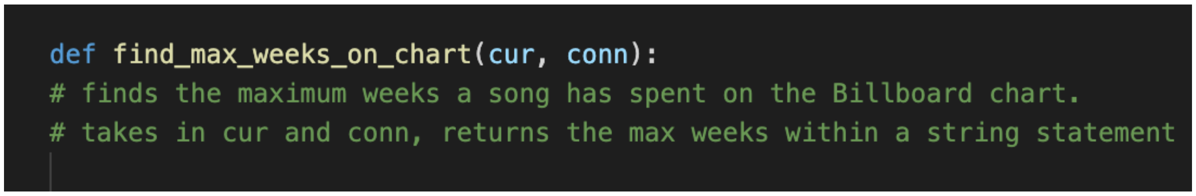
c.



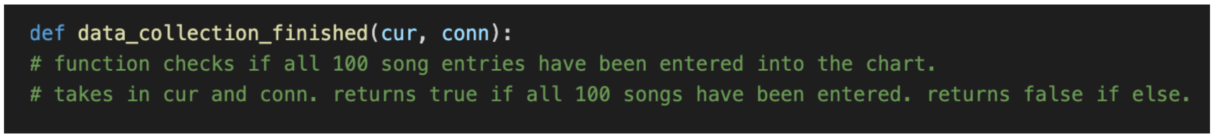
```
def creating_top_100_songs_table(cur, conn):
# creates the table of the top 100 songs on the Billboard website. the table includes the
# rank, song name, artist, peak on chart, and continuous weeks on chart for each song.
# puts 25 songs in the database at a time. after 100 songs, returns function renturns an error statement.
# takes in cur and conn, returns nothing.
```

d.


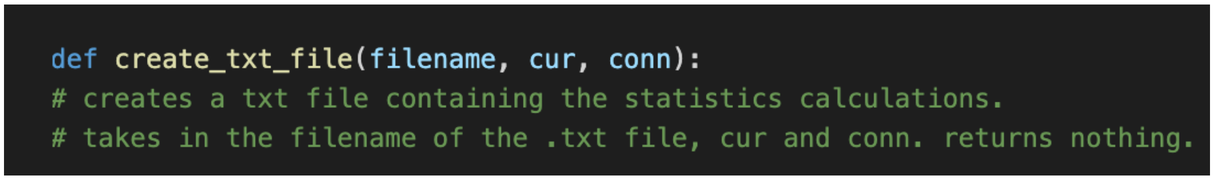
```python
def find_average_weeks_on_chart(cur, conn):
# finds the average number of continous weeks each song has spent on the Billboard chart.
# takes in cur and conn, returns the average weeks within a string statement
```

e.


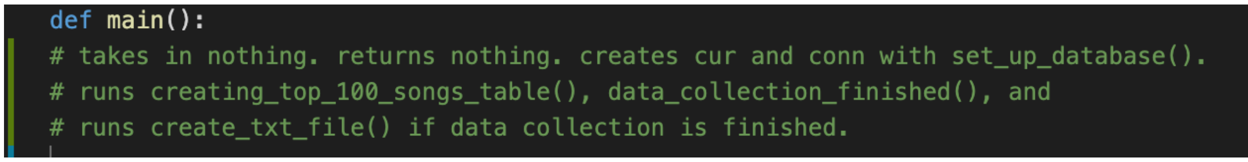
```python
def find_max_weeks_on_chart(cur, conn):
# finds the maximum weeks a song has spent on the Billboard chart.
# takes in cur and conn, returns the max weeks within a string statement
```



```python
def data_collection_finished(cur, conn):
# function checks if all 100 song entries have been entered into the chart.
# takes in cur and conn. returns true if all 100 songs have been entered. returns false if else.
```

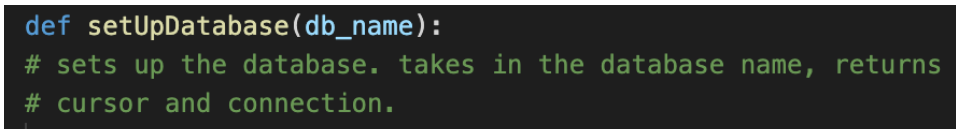
f.


```python
def create_txt_file(filename, cur, conn):
# creates a txt file containing the statistics calculations.
# takes in the filename of the .txt file, cur and conn. returns nothing.
```

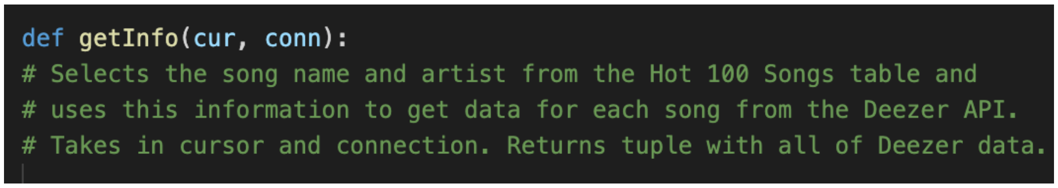
g.


```python
def main():
# takes in nothing. returns nothing. creates cur and conn with set_up_database().
# runs creating_top_100_songs_table(), data_collection_finished(), and
# runs create_txt_file() if data collection is finished.
```

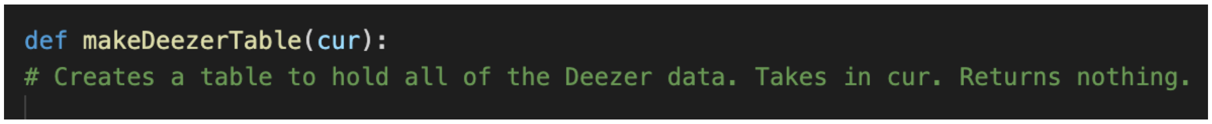
B. *deezer.py Documentation*

a.


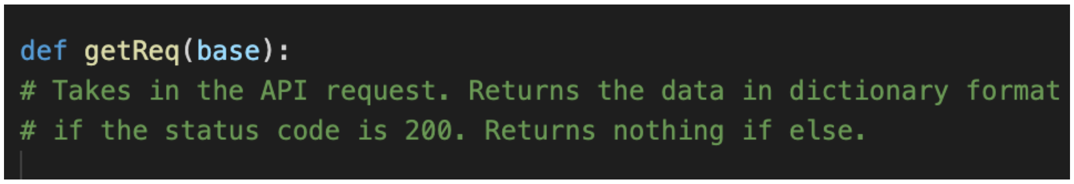
```python
def setUpDatabase(db_name):
# sets up the database. takes in the database name, returns
# cursor and connection.
```

b.


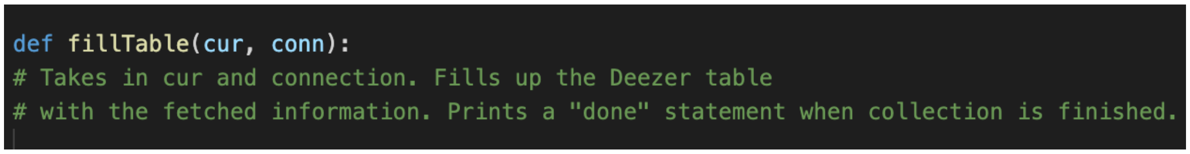
```python
def getInfo(cur, conn):
# Selects the song name and artist from the Hot 100 Songs table and
# uses this information to get data for each song from the Deezer API.
# Takes in cursor and connection. Returns tuple with all of Deezer data.
```

c.


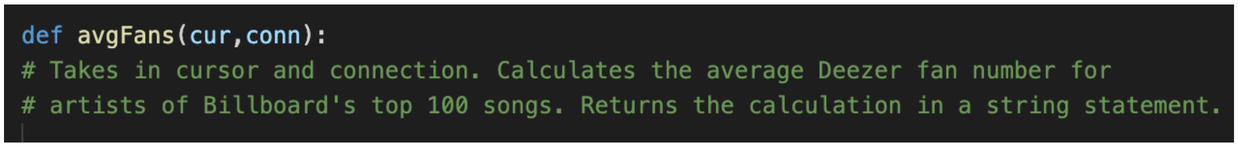
```python
def makeDeezerTable(cur):
# Creates a table to hold all of the Deezer data. Takes in cur. Returns nothing.
```

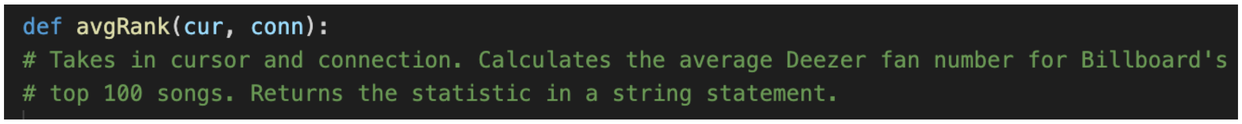
d.


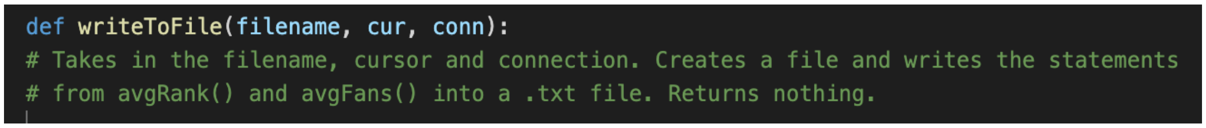
```python
def getReq(base):
# Takes in the API request. Returns the data in dictionary format
# if the status code is 200. Returns nothing if else.
```

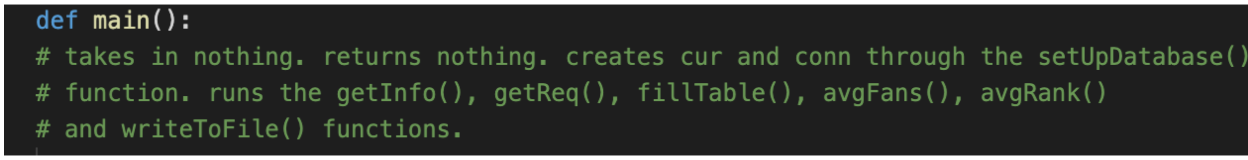
e.


```python
def fillTable(cur, conn):
# Takes in cur and connection. Fills up the Deezer table
# with the fetched information. Prints a "done" statement when collection is finished.
```

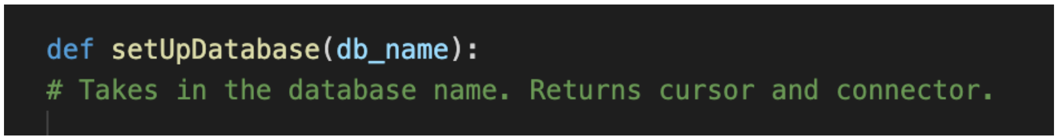
f.

```
def avgFans(cur,conn):
# Takes in cursor and connection. Calculates the average Deezer fan number for
# artists of Billboard's top 100 songs. Returns the calculation in a string statement.
```

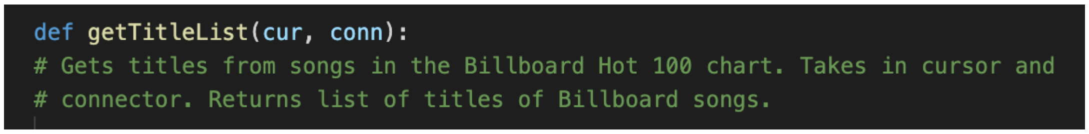
g.

```
def avgRank(cur, conn):
# Takes in cursor and connection. Calculates the average Deezer fan number for Billboard's
# top 100 songs. Returns the statistic in a string statement.
```

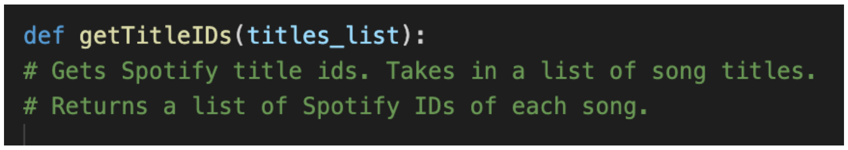
h.

```
def writeToFile(filename, cur, conn):
# Takes in the filename, cursor and connection. Creates a file and writes the statements
# from avgRank() and avgFans() into a .txt file. Returns nothing.
```

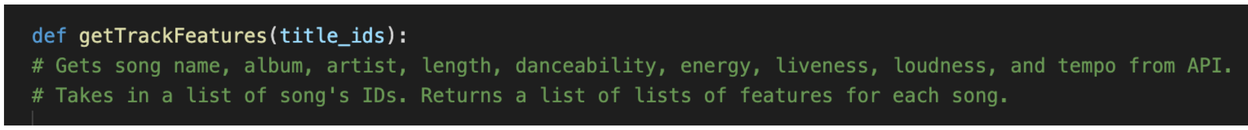
i.

```
def main():
# takes in nothing. returns nothing. creates cur and conn through the setUpDatabase()
# function. runs the getInfo(), getReq(), fillTable(), avgFans(), avgRank()
# and writeToFile() functions.
```

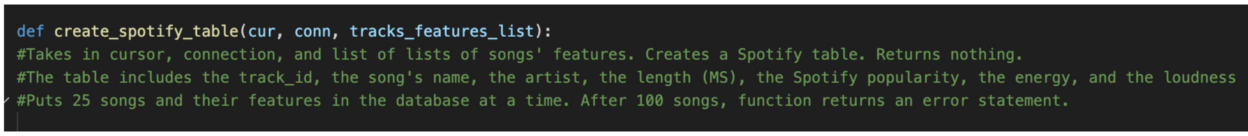
C. *spotifyInfo.py Documentation*

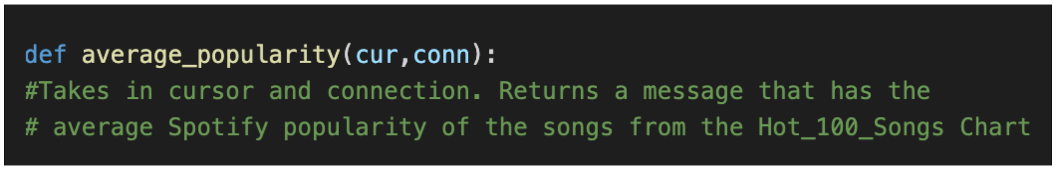
a.

```
def setUpDatabase(db_name):
# Takes in the database name. Returns cursor and connector.
```

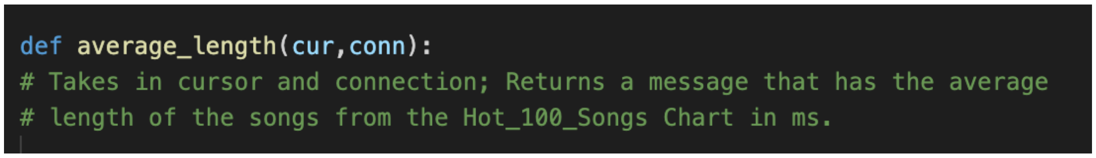
b.

```
def getTitleList(cur, conn):
# Gets titles from songs in the Billboard Hot 100 chart. Takes in cursor and
# connector. Returns list of titles of Billboard songs.
```

c.

```
def getTitleIDs(titles_list):
# Gets Spotify title ids. Takes in a list of song titles.
# Returns a list of Spotify IDs of each song.
```

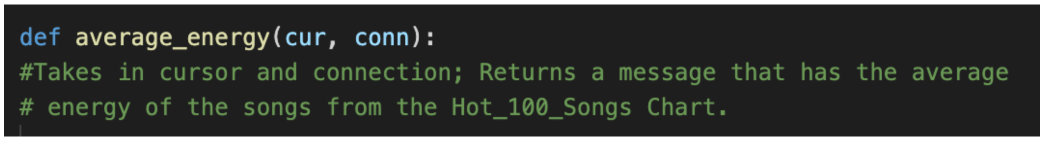
d.

```
def getTrackFeatures(title_ids):
# Gets song name, album, artist, length, danceability, energy, liveness, loudness, and tempo from API.
# Takes in a list of song's IDs. Returns a list of lists of features for each song.
```

e.

```
def create_spotify_table(cur, conn, tracks_features_list):
#Takes in cursor, connection, and list of lists of songs' features. Creates a Spotify table. Returns nothing.
#The table includes the track_id, the song's name, the artist, the length (MS), the Spotify popularity, the energy, and the loudness
#Puts 25 songs and their features in the database at a time. After 100 songs, function returns an error statement.
```

f.

```
def average_popularity(cur,conn):
#Takes in cursor and connection. Returns a message that has the
# average Spotify popularity of the songs from the Hot_100_Songs Chart
```

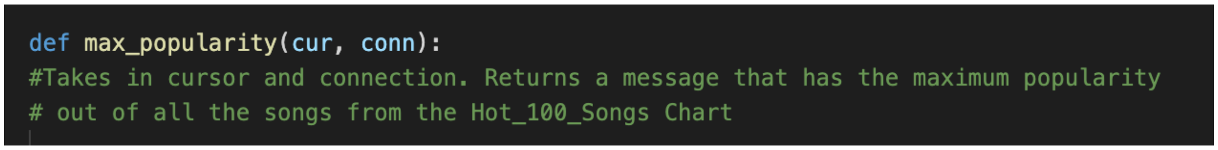
g.

```
def average_length(cur,conn):
# Takes in cursor and connection; Returns a message that has the average
# length of the songs from the Hot_100_Songs Chart in ms.
```

h.



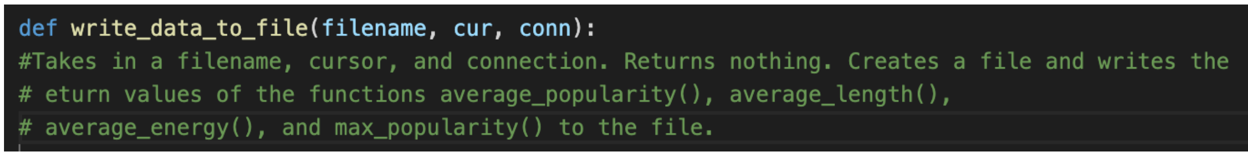
```
def average_energy(cur, conn):
#Takes in cursor and connection; Returns a message that has the average
# energy of the songs from the Hot_100_Songs Chart.
```

i.



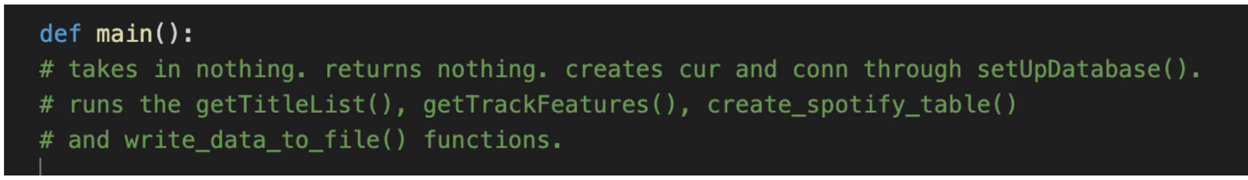
```
def max_popularity(cur, conn):
#Takes in cursor and connection. Returns a message that has the maximum popularity
# out of all the songs from the Hot_100_Songs Chart
```

j.



```
def write_data_to_file(filename, cur, conn):
#Takes in a filename, cursor, and connection. Returns nothing. Creates a file and writes the
# eturn values of the functions average_popularity(), average_length(),
# average_energy(), and max_popularity() to the file.
```

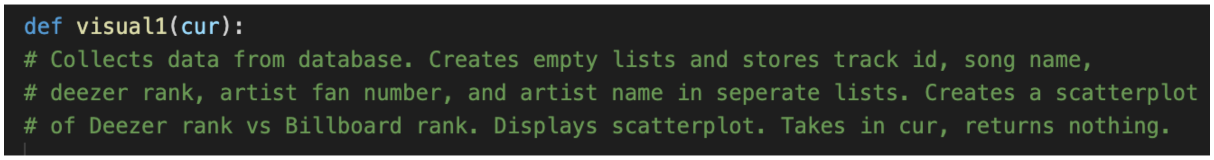
k.



```
def main():
# takes in nothing. returns nothing. creates cur and conn through setUpDatabase().
# runs the getTitleList(), getTrackFeatures(), create_spotify_table()
# and write_data_to_file() functions.
```

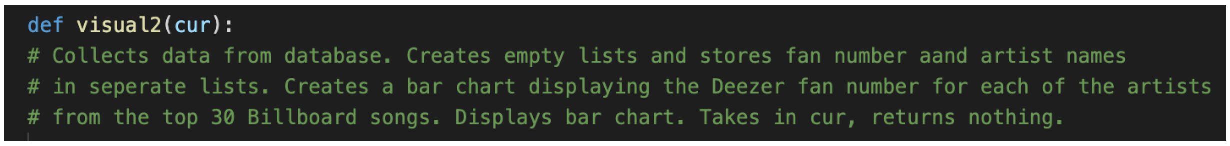
D. visuals.py Documentation

a.



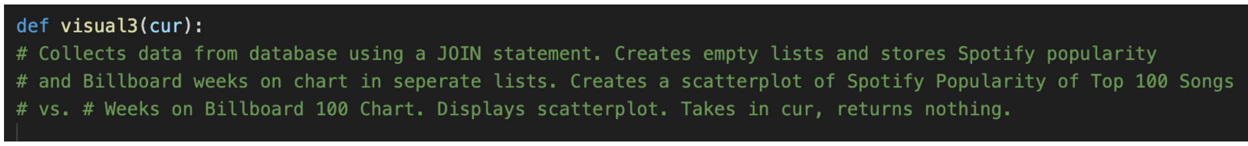
```
def visual1(cur):
# Collects data from database. Creates empty lists and stores track id, song name,
# deezer rank, artist fan number, and artist name in seperate lists. Creates a scatterplot
# of Deezer rank vs Billboard rank. Displays scatterplot. Takes in cur, returns nothing.
```

b.



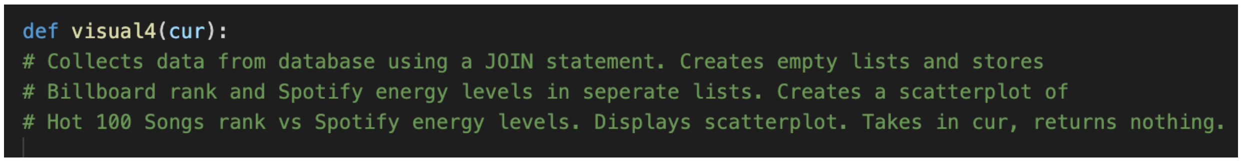
```
def visual2(cur):
# Collects data from database. Creates empty lists and stores fan number aand artist names
# in seperate lists. Creates a bar chart displaying the Deezer fan number for each of the artists
# from the top 30 Billboard songs. Displays bar chart. Takes in cur, returns nothing.
```

c.



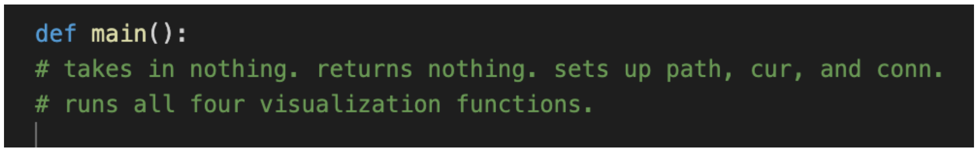
```
def visual3(cur):
# Collects data from database using a JOIN statement. Creates empty lists and stores Spotify popularity
# and Billboard weeks on chart in seperate lists. Creates a scatterplot of Spotify Popularity of Top 100 Songs
# vs. # Weeks on Billboard 100 Chart. Displays scatterplot. Takes in cur, returns nothing.
```

d.



```
def visual4(cur):
# Collects data from database using a JOIN statement. Creates empty lists and stores
# Billboard rank and Spotify energy levels in seperate lists. Creates a scatterplot of
# Hot 100 Songs rank vs Spotify energy levels. Displays scatterplot. Takes in cur, returns nothing.
```

e.



```
def main():
# takes in nothing. returns nothing. sets up path, cur, and conn.
# runs all four visualization functions.
```

V. Documentation of Resources

Date	*Issue Description*	*Location of Resource*	*Result (Was Issue Solved?)*

4/15	Couldn’t find the Spotify song IDs	https://github.com/plamere/spotipy/blob/master/examples/search.py	Yes, I figured out how to properly use the search() function
4/15	We didn’t know how to use the google developers console to acquire an API key for YouTube	https://www.youtube.com/watch?v=th5_9woFJmk&t=1223s	Yes, even though we didn’t use YouTube, this video walked us through how to create a project and get an API key for the YouTube Data API
4/16	Couldn’t narrow down the correct ID for each song because the search() function returned a huge dictionary of things that no JSON formatter would accept	https://jsonformatter.curiousconcept.com/#	Yes, after lots of searching Lucy found a formatter that finally worked!
4/18	Didn’t know how to access the song features that I wanted.	https://betterprogramming.pub/how-to-extract-any-artists-data-using-spotify-s-api-python-and-spotipy-4c079401bc37	Yes, I figured out I needed to add sp.track() and call the audio_features() function on that.