**Aishani Agrawal, Blair Bocklet, Alex Jacobs**
**Professor Barbara Ericson**
**SI 206**
**12 December 2022**
**Github: https://github.com/bbocklet/206Final**

# Final Project Report

1. Goals for the Project: Our goal for this assignment was to collect data from two APIs and one website using BeautifulSoup. We decided to focus on music streaming platforms for this assignment: one API is from iTunes, one API is from Last.fm, and the website is ACharts.co. Since we are all music lovers, we thought this would be a perfect theme for our project. We wanted to learn more information and make calculations related to our favorite music.

2. Goals We Achieved: We were able to achieve all of our goals of collecting data from the two APIs and the ACharts.co website. We were able to obtain data on song names, artist names, and release dates for the songs, as well as use Beautiful soup to get a list of the top 100 songs and their artists. Using this data, we could make calculations on which artists had the most songs in the top chart, who released the most music from 2012-2022, how many songs per album are represented by the API data, as well as artist representation via playcount and listener amount on the Last.fm platform.

3. Problems: Throughout the process, all three group members faced issues when trying to add our data to the database, figure out how to limit each run to 25 pieces of data, and figure out how to combine all of our data into one database after each working separately on our assigned tasks. On the iTunes API in particular, one problem we faced was the API only yielding 50 of the same results upon each run. The problem with the iTunes API functioning in that particular way is that we would be unable to make a data table with 100 elements without any repeats.

4. File that contains calculations
    For the iTunes API, the calculations are stored in blairData.csv
    For the Top Charts data, the calculations are stored in aishaniData.csv
    For the Last FM data, the calculations for the bar graph are stored in AlexData1.csv

5. Visualizations

Figure 1: Artists with 2 or More Songs in the Top Charts by Percent

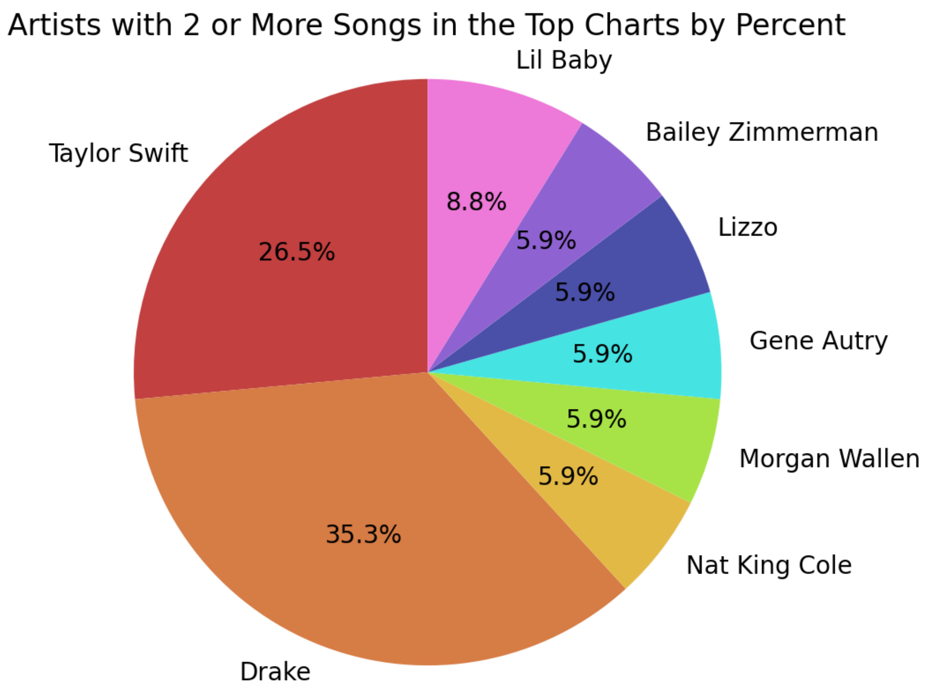


Figure 2: Artists Contribution to Music 2012-2022 by Percent

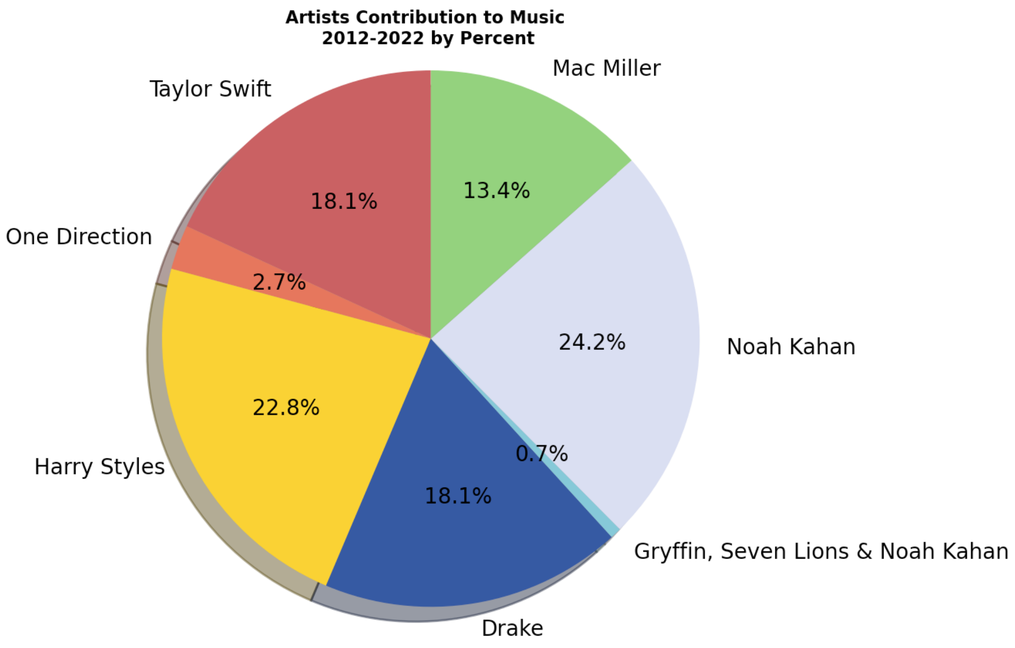


Figure 3: Taylor Swift Song Representation Per Album by iTunes API

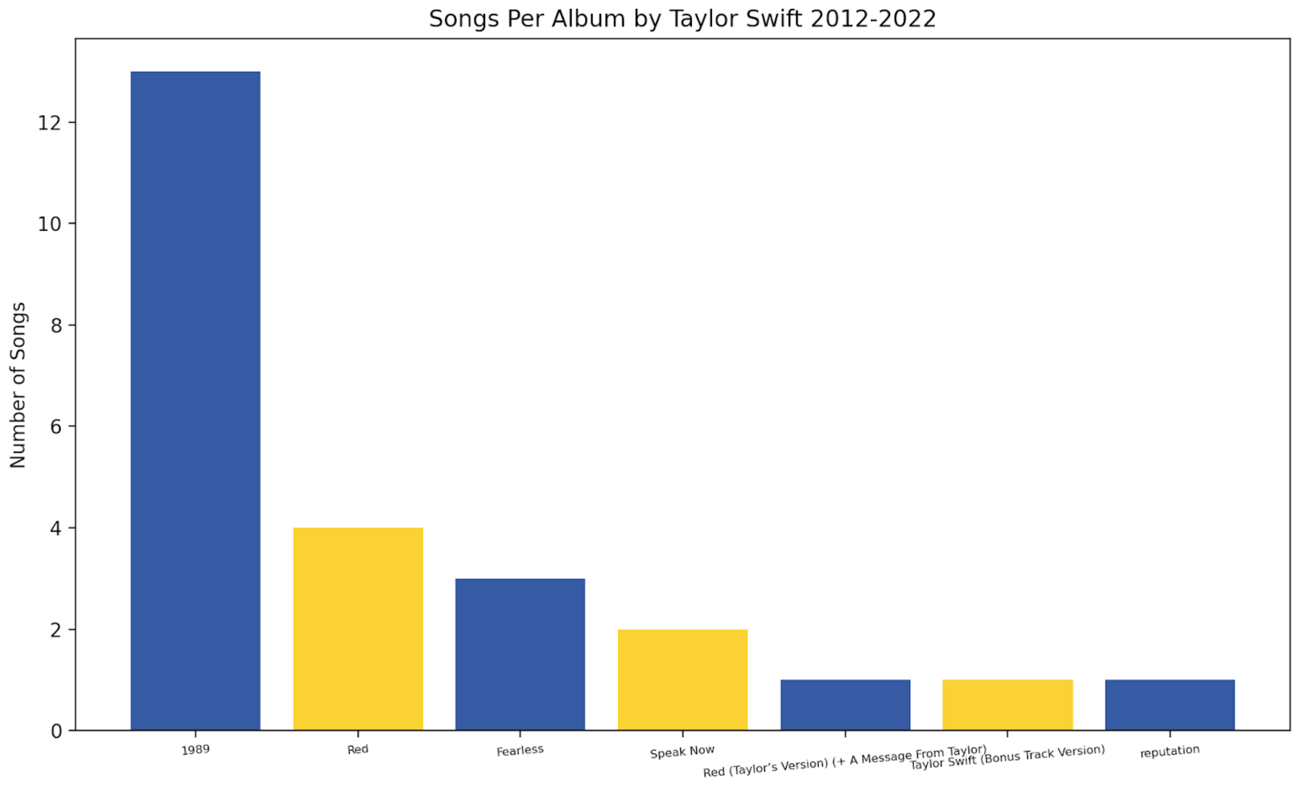


Figure 4: Comparison of Playcount vs. Listeners for #1 Songs by Artist via Last.fm API

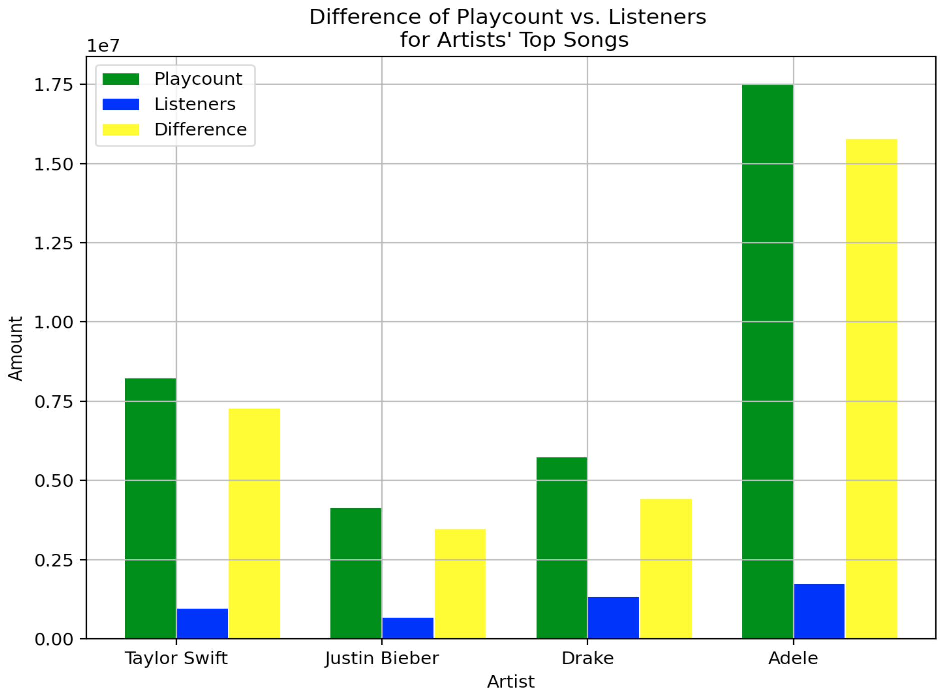
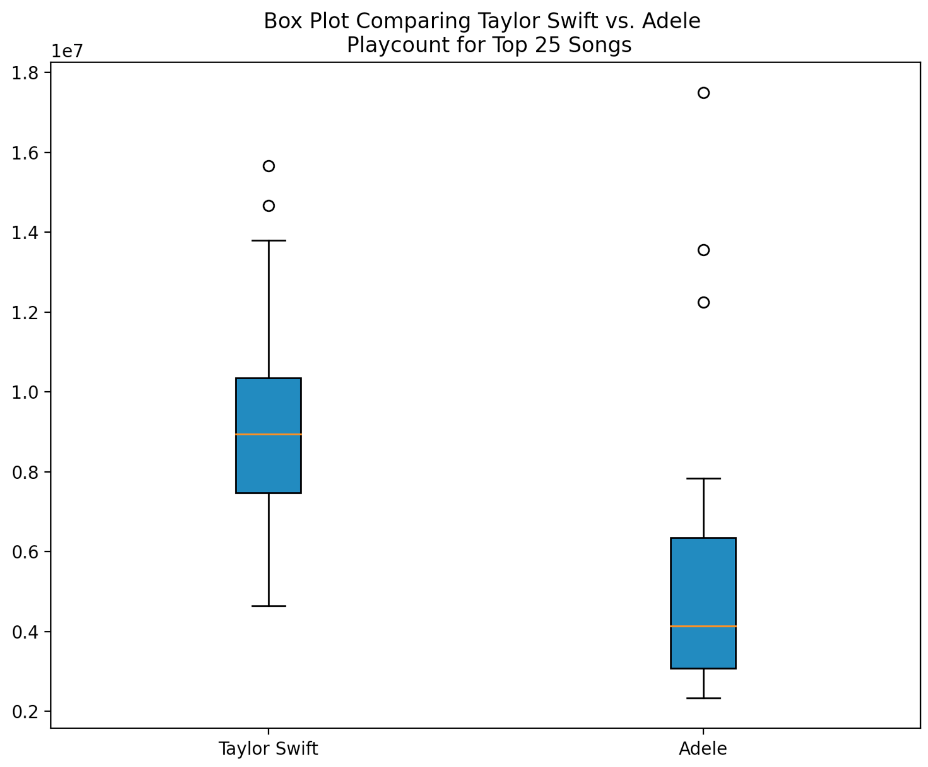


Figure 5: Boxplot Displaying Distribution of Playcount for Taylor Swift vs. Adele



6. Instructions for Running the Code:
    i. When running itunes.py, the user is fetching from an iTunes API that returns search results based on the provided term. This particular file makes five calls to the API per run yielding 25 results upon each file execution. Whilst four runs should, theoretically, yield the desired 100 results in the database, the user should run the code a total of five times to ensure a full dataset as repeat results are not inserted into the table. As the file runs, the data visualizations will appear in a popup window, so the user should simply exit out of the popups to run the code again. Between each run of the code, the output file, blairData.csv, will display the calculations of which artist released the most music from 2012-2022 for the dataset. On the fifth run of the code, the popups will display the final visualizations for the complete dataset.
    ii. When running top100.py, the user is using Beautiful Soup to scrape data from https://acharts.co/us_singles_top_100. This gets them the information for the top 100 songs and who they are by. The data is stored in a list of tuples with the song name and artist name. When the user runs

the code, data will be added to two tables in the database: one with the song names, song IDs, and artist IDs. The other table will have artist names and artist IDs. Each time the user runs the code, 25 items will be added to each table in the database. When the user runs the code, a pie chart visualization will pop up, but the user should close the popup and keep running the code until 100 items are added to the topChart table and 74 items are added to the artistID table. Another file called aishaniData.csv will also be created the first time the code is run. This file contains the calculations for which artists have 2 or more songs in the top charts by percent.

iii. When running the lastfm.py, the user is extracting data that displays the top 25 songs from 4 artists: Taylor Swift, Justin Bieber, Drake, and Adele. The information provided for all 4 artists includes song title, playcount, and listeners for the particular artist's song. One function then calls to the API to collect the top 25 songs and their information from the respective artist, storing them in a list that is being appended with each run of the code, adding to the datatable. To avoid duplicates, I created the variable "id" to be the primary key. As the user runs the code 4+ times, they will see that the datatable has reached 100 items, and prints out "100 items exist" if the code exceeds the limit. The code will also produce a csv file titled "AlexData1.csv" with calculations of percent difference between artists' playcount and listener amount for their number 1 ranked song, which is also shown by the first visualization that is presented. The second visualization is of a boxplot graph displaying the distribution of playcount for Taylor Swift and Adele's top 25 songs.

7. Function Documentation:

iTunes API

i. **setUpDatabase():** takes no input parameters and returns the creation of the music.db database.

ii. **create_taylorHarry_table(cur, conn)**: takes the input parameters of the cur and conn which are cursor and connection. These two parameters were created via the setUpDatabase() function. This function returns an empty table in the music.db file called taylorHarry. This table will eventually store data of song titles, artist and album ids, release year, and assign a song id.

iii. **create_artist_table(cur, conn)**: takes the input parameters of cur and conn. This function returns an empty table that will eventually store what artist id represents which artist in the taylorHarry table and also ensures that there are no duplicate strings in the taylorHarry table.

iv. **create_album_table(cur, conn)**: takes the input parameters of cur and conn. This function returns an empty table that will eventually store what album id represents which album in the taylorHarry table and ensures that there are no duplicate strings in the taylorHarry table.

v. **taylorData(cur, conn)**: takes the input parameters of cur and conn. This function calls the iTunes API and requests results for Taylor Swift. The function returns five results from the API call and stores these results in the three tables that were previously established.

vi. **harryData(cur, conn)**: takes the input parameters of cur and conn. This function calls the iTunes API and requests results for Harry Styles. The function returns five results from the API call and stores these results in the three tables that were previously established.

vii. **drakeData(cur, conn)**: takes the input parameters of cur and conn. This function calls the iTunes API and requests results for Drake. The function returns five results from the API call and stores these results in the three tables that were previously established.

viii. **noahData(cur, conn)**: takes the input parameters of cur and conn. This function calls the iTunes API and requests results for Noah Kaha. The function returns five results from the API call and stores these results in the three tables that were previously established.

ix. **macData(cur, conn):** takes the input parameters of cur and conn. This function calls the iTunes API and requests results for Mac Miller. The function returns five results from the API call and stores these results in the three tables that were previously established.

x. **join_calc_visual(cur, conn, show)**: takes the input parameters of cur, conn, and a boolean value named show. In this function, calculations for Artists contribution to music 2012-2022 by percent are returned and, if show=True, the function also displays a pie chart that displays the results from the calculations. This function also includes a join statement to connect the artist and dataHarry tables.

xi. **second_calc_visual(cur, conn)**: takes the input parameters of cur and conn. In this function, calculations for how many songs per Taylor Swift album are represented by the API data is returned in the display form of a bar chart. This function also includes a join statement to connect the album and dataHarry tables.

xii. **writeFile(data, file_name):** takes the input parameters of data – which is a function call to join_calc_visual with the show boolean value set to false – and a file called blairData.csv. This function writes the artist's contribution to music 2012-2022 by percent calculations to a csv file.

xiii. **main()**: defines cur and conn, along with calling all the functions in the file.

Top Chart data (top100.py)

xiv. **create_topchart_table(cur, conn)**: takes the input parameters of cur and conn which are cursor and connection. These two parameters are defined in the main() function. This function returns an empty table in the music.db file called topChart. This table will eventually store data of song titles and song and artist ids.

xv. **create_artist_id(cur, conn)**: takes the input parameters of cur and conn. This function returns an empty table in the music.db file called artistID. This table will eventually store data of artist names and artist ids.

xvi. **add_artist_data(cur, conn)**: takes the input parameters of cur and conn. The function takes data from top_list and data_dict which were defined at the beginning of the code. Top_list is a list of tuples with the song name and artist. Data_dict assigns a unique number to each artist. The function returns 25 results (artist name and ID) and stores them in the artistID table in the database.

xvii. **add_data(cur, conn)**: takes the input parameters of cur and conn. The function takes data from top_list and data_dict in order to return 25 results (song id, song name, and artist id) in the topChart table in the database.

xviii. **join_tables(cur, conn)**: takes the input parameters of cur and conn. This function contains a join statement that joins the topChart table and artistID table on the artist ids.

xix. **writeFile(percent_dict, file_name)**: takes the parameters percent_dict and file_name. Percent_dict is defined in the code above this function; it is a dictionary with artist names and the percent of songs they have in the top charts out of the total number of songs that artists with 2 or more songs in the chart have. File_name is defined as aishaniData.csv in the main() function. This function writes artists with 2 or more songs in the top charts by percent into a csv file.

xx. **main()**: defines cur and conn, along with calling all the functions in the file.

Last.fm API:

**xxi.** **setUpDatabase(db_name):** returns the newly created music.db databases and takes  no input parameters.

**xxii.** **create_artist_table(cur, conn):** This function takes in the input parameters of cur and conn. It then creates an empty table called artist_table which stores artist ids and artist names into the empty table as later in the function.

xxiii. **create_compiled_table(cur, conn, data_list):** This function takes in the input parameters of cur and conn. The function creates the new empty table with columns ID as the primary key, artist ID, song title, rank, playcount and listeners. The function then loads 25 items into the table (or less) at a time via a for loop as the necessary data is extracted from data_list, which is the list called compiled_all_data (as collected in the following function) and added into the table titled compiled_table in the database.

xxiv. **get_data():** This function does not take any inputs. It pulls data from the API link for each specific artist, adding each tuple containing the artist, rank, song title, playcount and listener count into a list. The list, compiled_all_data, is what is used in the previous function, create_compiled_table(), and makes up the data that is being loaded into compiled_table.

xxv. **tables_visuals(cur, conn, data_list, show):** This function takes in the input parameters of cur and conn, as well as data_list, which is compiled_all_data, and "show," a boolean value, which is set to "True" when the function is being called. The function creates the first data visualization, which is a bar graph, displaying the results from the calculations that yield percent difference between playcount and listeners.

xxvi. **second_visualization(data, show):** This function takes in the input parameters of data, which is  combined_all_data, and the boolean value of "show," which is set to "True" when the function is called. This function creates the second visualization, which is in the form of a box plot. This visualization displays the distribution of playcount for the top 25 songs by Taylor Swift and Adele, comparing such side by side.

xxvii. **writeFile(data, file_name):** This function takes in data, which will then be represented in the csv output file, and a file name, which in this case is "AlexData1.csv." A list of percentages that represent the differences in artists' playcounts and listener counts from their number 1 songs are passed into the function. These numerical values and the values they represent, not yet put into a percentage, are then written into the file, which is created after this function is executed.

xxviii. **main():** This function does not take in any parameters. It defines cur and conn, and calls all of the functions in the file, lastfm.py, to then be executed accordingly.

8. Resources

| Date | Issue Description | Location of Resource | Result (did it solve the issue?) |
|---|---|---|---|
| December 3rd | My API did not yield enough results to make a table with 100 entries | Office hours | Yes. At office hours, it was recommended I add more API calls to ensure that I have backup data in the case of a repeat. |
| December 6th | When writing to my csv file, the data was formatted strangely | Office hours | Yes. Rather than use writer.writerow(), it was recommended I use f.write() which fixed the formatting of the csv file. |
| December 7th | I wasn't able to figure out exactly how to get my artist IDs to match up in my topChart table and artistID table.<br><br>Needed help with API data collection. | Meeting with IA (Antoinette Puca) | Yes. I created a dictionary to keep track of each unique artist and their corresponding id.<br><br>Helped decipher issue with my API link. |
| December 8th | In my code, a data visualization appeared, then the csv file was updated with new data, and then the correct data visualization appeared. This was because the function join_calc_visual is called twice, once in main and once as a | Office hours | Yes. To still allow for calling join_calc_visual twice, it was recommended I add a boolean parameter and use that as a condition for plt.show(). This allowed for the csv file to update and then show a |

| | | | |
|---|---|---|---|
| | parameter. | | visualization that matched the data. |
| December 8th | When I tried setting the limit to 25 to only add 25 items to the database at a time, I was running into issues. | Office hours | Yes. I added a try and except statement and set my start and end variables to the appropriate values. |
| December 9th | Although my add_data function and topChart table were working perfectly fine, the add_artist_data function and artistID table were not. The function would not add all 74 values to the table and would stop at 71. | Office hours | No. We tried multiple different methods and none of them seemed to work. Either that or they would decrease the amount of data that was added to the table. |
| December 11th | Trying to resolve the same issue as before. | Meeting with IA (Antoinette Puca) | No. We tried multiple different methods, such as adding a while loop, changing the for loop, etc, but just ended up reverting to my original code because that seemed like it was giving the best output. We figured out that the issue likely lies in the fact that repeating artists limit the amount of data that is added. |
| December 12th | Trying to resolve the same issue as before. | Meeting with GSI (Diana Tassew) | Yes. We were able to change the way I was |

| | | | |
|---|---|---|---|
| | | | assigning artist ids and add a select statement to match up the songs in my topChart table to the artists in my artistID table.<br><br>For Lastfm API, we were able to change the way I was loading the 25 items in so the issue would be resolved. |