https://github.com/pwiaduck/si206_final_project

In addition to your API activity results, you will be creating a report for your overall project. The report must include:

1. The goals for your project including what APIs/websites you planned to work with and what data you planned to gather (10 points)

We worked with Ticketmaster, EventBrite, and SeatGeek. Our goal of the project was to find trends and calculate the percentage breakdowns of pricing, event types, and location. We gathered data from the API including region/location, pricing, event types, and event names.

2. The goals that were achieved including what APIs/websites you actually worked with and what data you did gather (10 points)

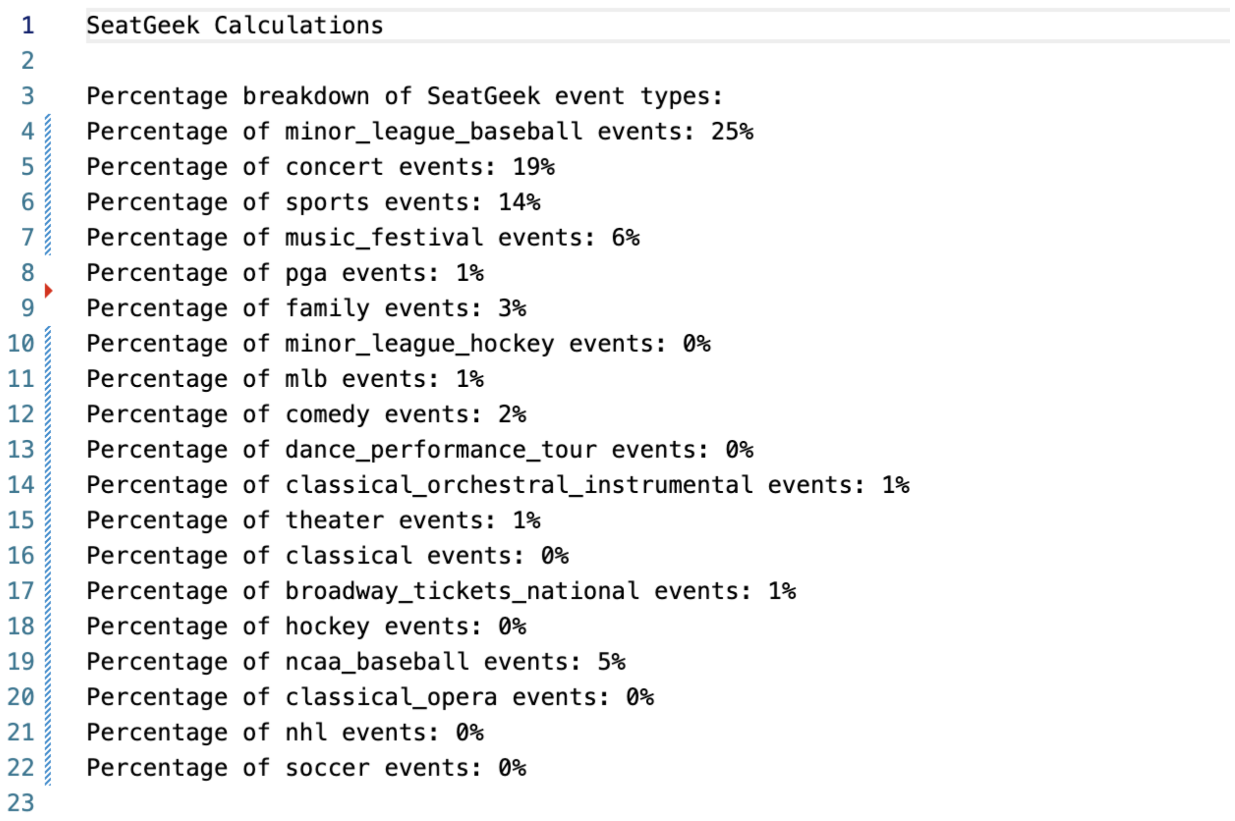
We were able to achieve the calculations of percentages and differences of pricing, location, and event types among the Ticketmaster, EventBrite, and SeatGeek APIs. We gathered data from the APIs regarding performer, venue, location, pricing, and event types.

3. The problems that you faced (10 points)

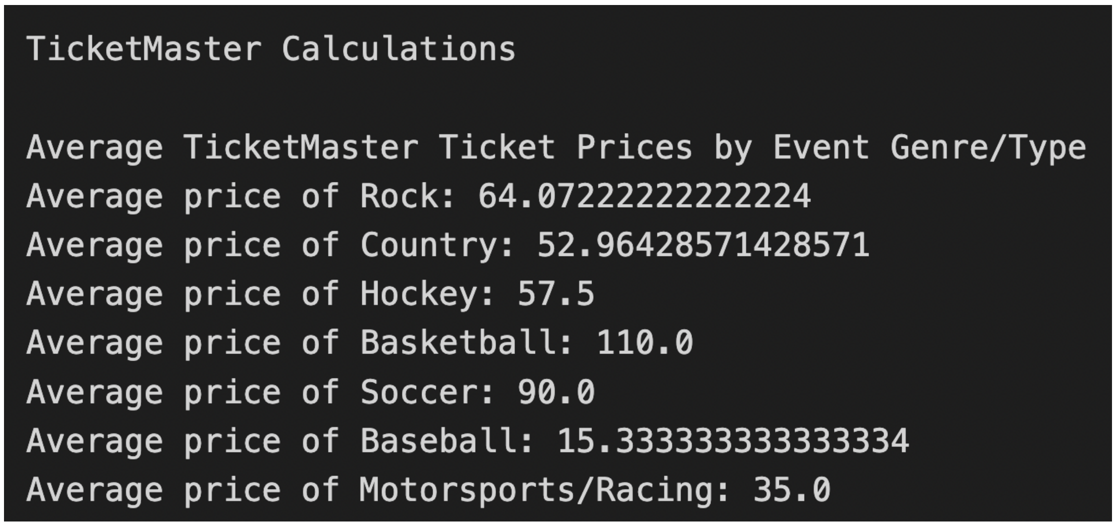
One of the main issues within the group was figuring out how to add the rows of data to the table 25 rows at a time. This was different for each member due to the differences of how the data was retrieved from each API.
Another issue faced was for the SeatGeek API. This API continuously adds data as more events are uploaded to the site, therefore, this took time to figure out and fix. The SeatGeek data changes day to day, therefore the calculations and data in the database slightly changes.

In the EventBrite API, they canceled access to multiple events through the use of one singular link, so all the information on events had to be accessed through individual urls that had each event id. In order to do this more efficiently and impartially, I had to use beautiful soup in order to scrape for the first 100 event ids of the most popular upcoming events. From there, I was able to access each url after gathering that information in order to access my api.

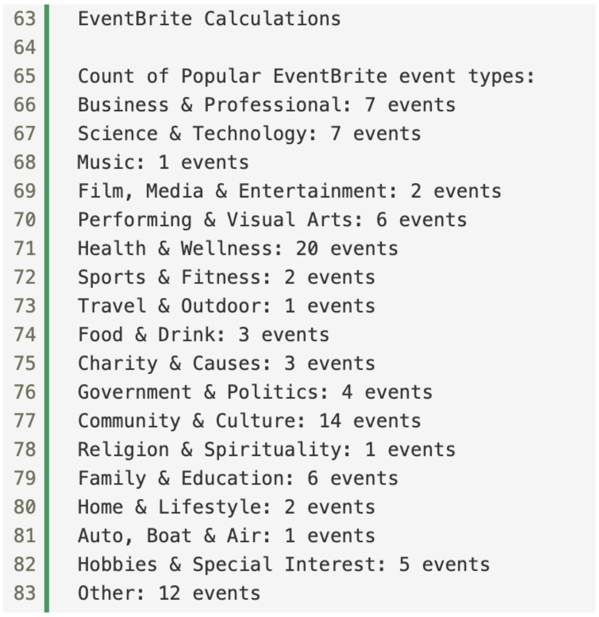
4. The calculations from the data in the database (i.e. a screenshot) (10 points)



```
SeatGeek Calculations

Percentage breakdown of SeatGeek event types:
Percentage of minor_league_baseball events: 25%
Percentage of concert events: 19%
Percentage of sports events: 14%
Percentage of music_festival events: 6%
Percentage of pga events: 1%
Percentage of family events: 3%
Percentage of minor_league_hockey events: 0%
Percentage of mlb events: 1%
Percentage of comedy events: 2%
Percentage of dance_performance_tour events: 0%
Percentage of classical_orchestral_instrumental events: 1%
Percentage of theater events: 1%
Percentage of classical events: 0%
Percentage of broadway_tickets_national events: 1%
Percentage of hockey events: 0%
Percentage of ncaa_baseball events: 5%
Percentage of classical_opera events: 0%
Percentage of nhl events: 0%
Percentage of soccer events: 0%
```

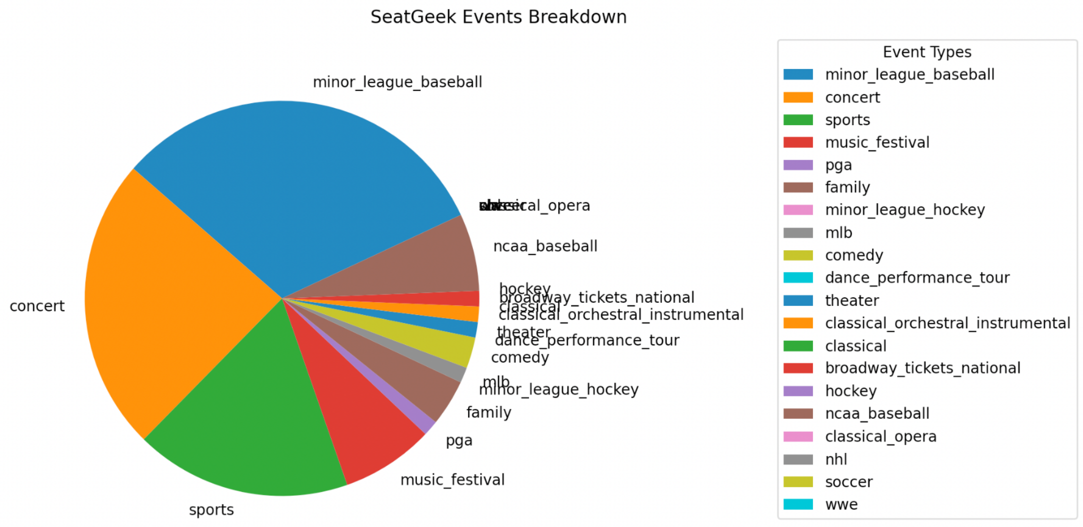


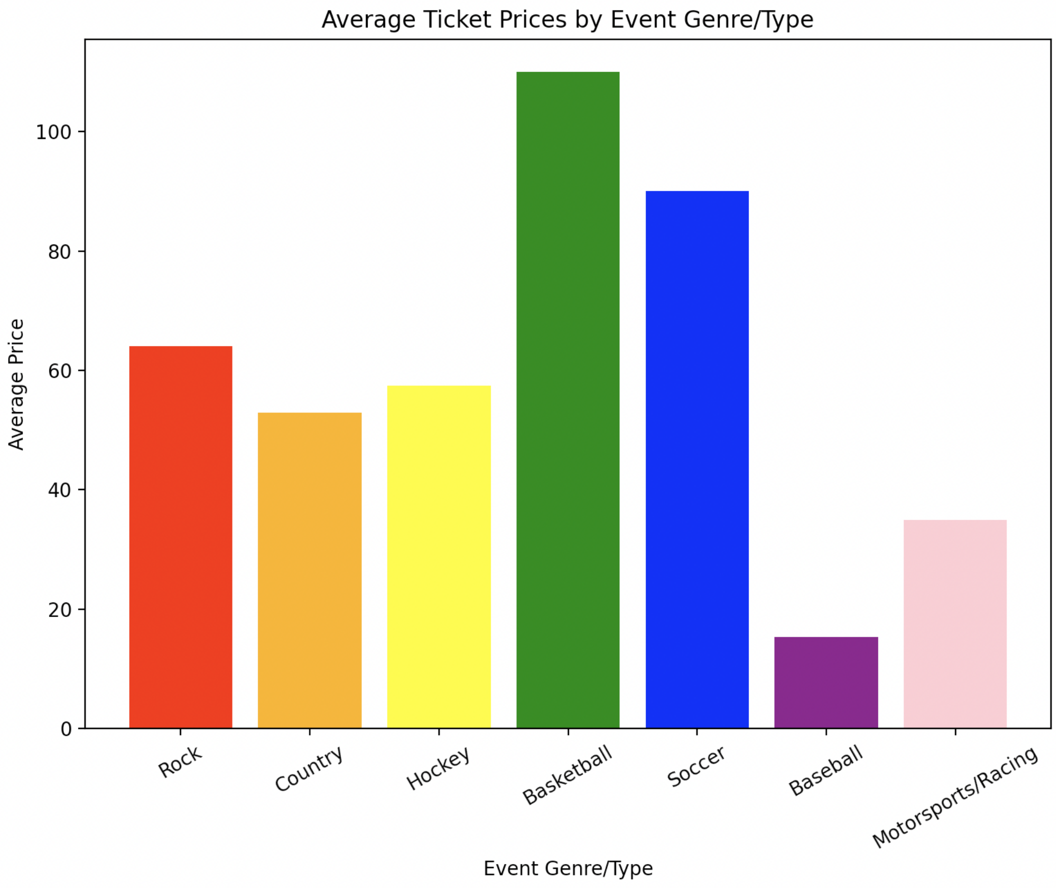
```
TicketMaster Calculations

Average TicketMaster Ticket Prices by Event Genre/Type
Average price of Rock: 64.07222222222224
Average price of Country: 52.96428571428571
Average price of Hockey: 57.5
Average price of Basketball: 110.0
Average price of Soccer: 90.0
Average price of Baseball: 15.333333333333334
Average price of Motorsports/Racing: 35.0
```



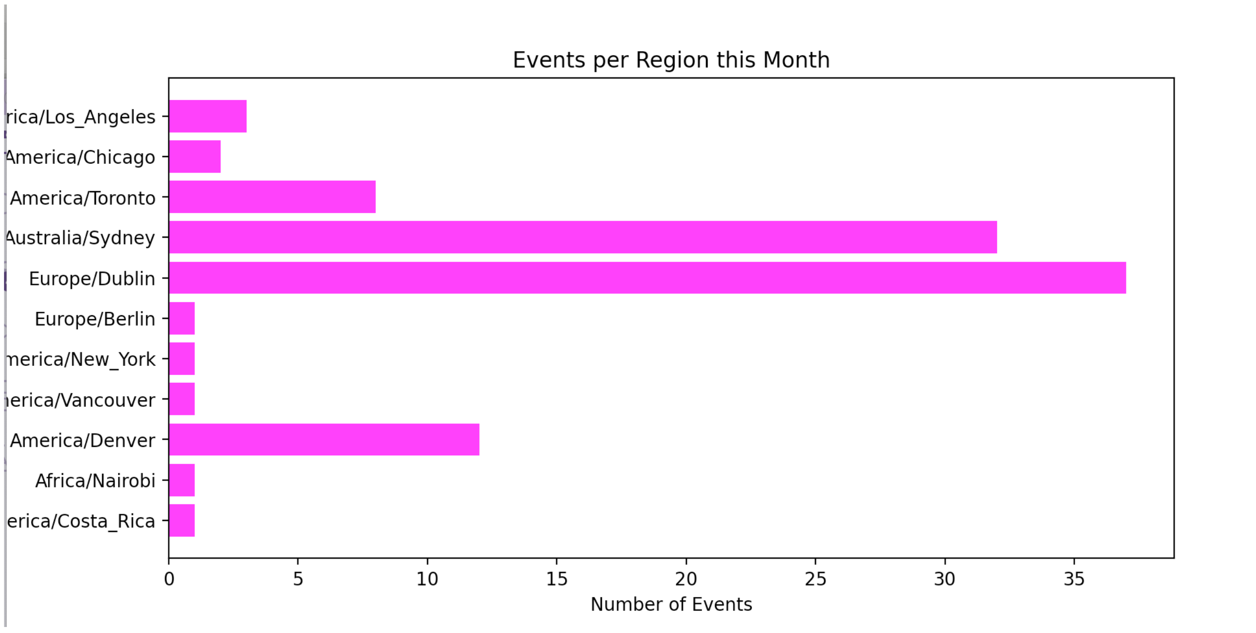
```
EventBrite Calculations

Count of Popular EventBrite event types:
Business & Professional: 7 events
Science & Technology: 7 events
Music: 1 events
Film, Media & Entertainment: 2 events
Performing & Visual Arts: 6 events
Health & Wellness: 20 events
Sports & Fitness: 2 events
Travel & Outdoor: 1 events
Food & Drink: 3 events
Charity & Causes: 3 events
Government & Politics: 4 events
Community & Culture: 14 events
Religion & Spirituality: 1 events
Family & Education: 6 events
Home & Lifestyle: 2 events
Auto, Boat & Air: 1 events
Hobbies & Special Interest: 5 events
Other: 12 events
```

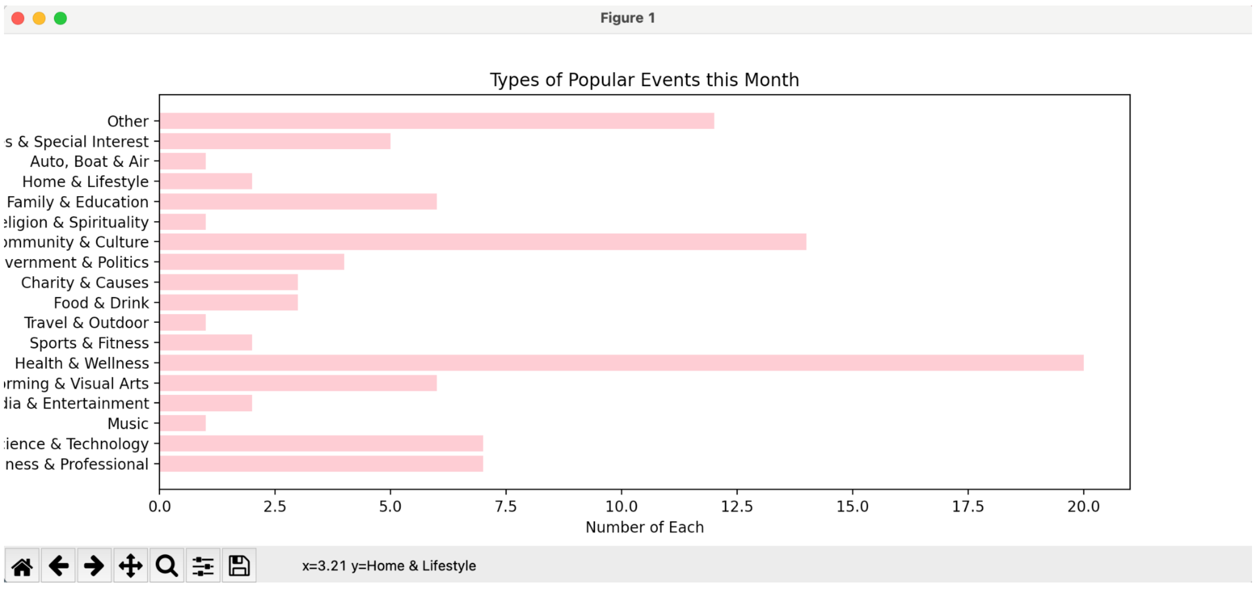
5. The visualization that you created (i.e. screenshot or image file) (10 points)





Average Ticket Prices by Event Genre/Type
Average Price
0
20
40
60
80
100
Rock
Country
Hockey
Basketball
Soccer
Baseball
Motorsports/Racing
Event Genre/Type





6. Instructions for running your code (10 points)

<u>EventBrite</u>

Call the main() function four times. Each time the code runs, the large table, Names, will increase increments of 25 with data up until 100 rows are filled. The chart visualizations will be displayed with each run.

<u>SeatGeek</u>

Call the main() function 4 times. Data will be filled into database tables 25 rows at a time. Calculations will be made and written to a text file. A pie chart visualization will be created and displayed too.

<u>TicketMaster</u>

Call the main() function 4 times. After every run, 25 rows will be loaded into the ticketmaster datatable and however many duplicate genre strings that existed within the ticketmaster datatable will be loaded into the genretable datatable. At the end of the 4 runs, there will be 100 rows in the ticketmaster datatable and the genretable datatable will consist of 2 columns, one with assigned id numbers for duplicate strings and the other with the genre string names. A bar graph will be created with each run and progressively have more bars as more data is loaded into the datatable.

7. Documentation for each function that you wrote. This includes describing the input and output for each function (20 points)

<u>SeatGeek</u>

- create_db(): Takes a database as the input, and creates a database. Returns a cursor and connector variable.
- get_data(): Takes the client id, client secret, connector and cursor variable as input. The client id and client secret are registered API key and password given from the SeatGeek API. This function retrieves data from the API's data pages, and loads it in json format. The data is added into a list by page, then the function returns said list.
- find_unique_event_types(): Takes a list of json data as input. Iterates through the list and finds unique types of events and stores them in a list. A list of event types is returned without duplicates.
- create_type_table(): Takes a cursor, connector, and list of event types from find_unique_event_types() as input. Drops a table if it exists and creates a table named SeatGeek_types. Adds unique id integers and event types as strings into the table. Does not return anything.
- find_unique_cities(): Takes a list of json data as input. Iterates through the list and finds unique cities and stores them in a list. A list of cities is returned without duplicates.
- create_cities_table(): Takes a cursor, connector and list of cities from find_unique_cities() as input. Drops table if it exists and creates a table named SeatGeek_cities. Adds unique id integers and cities as strings into the table. Does not return anything.
- find_performers(): Takes a list of json data as input. Iterates through the data and returns a list with the names of the performers in order of iteration.
- find_types(): Takes a list of json data as input. Iterates through the data and returns a list of the event types in the order of iteration.
- find_cities(): Takes a list of json data as input. Iterates through the data and returns a list of the cities in the order of iteration.
- assign_type_vals(): Takes the returned lists from find_unique_event_types() and find_types() as input. Iterates through the second parameter and assigns an index it matches in the first list. Returns list of indexes.
- assign_city_vals(): Takes the returned lists from find_unique_cities() and find_cities() as input. Iterates through the second parameter and assigns an index it matches in the first list. Returns list of indexes.
- make_table(): Takes a cursor, connector, index, and returned lists from find_performers(), assign_type_vals(), and assign_city_vals() as input. Iterates through a

range of 25, and adds rows containing information indexed from each list. Adds 25 rows at a time. Does not return anything.
- main(): No input taken. Uses the functions listed above to create lists and tables with specific SeatGeek API data. Determines which set of 25 will be the index for make_table() with if/else statements. Finds the total number of rows from the database (100), and selects the number of rows for each event type. Finds percentages of each event type in the database, and writes it to a text file. Creates and displays a pie chart with the percentage calculations.

<u>EventBrite</u>

- make_events_table(index)
    - The index argument is pulled from the main function and facilitates how much data is inserted into the table "Names" during each run. The first part of the function loops through the different url endings, and appends each url to a list. That list of urls is then looped through and scraped using beautiful soup in order to find the event ids from the first 100 events on the "popular" page. Once the event ids are acquired they are appended to a different list. This list is then looped through, used as the event id part of the EventBrite API url. Through this data, the names, regions, and category ids of each event are gathered and appended into their respective lists. I then initialize the region ids, so there are no duplicate strings and append that to another list. Then, I initialize my cursor and connect for the database. I make a list with no duplicates from the region list, and create the region table of region ids and region names. Next, using the index made in the main() function, I insert values into my large table, "Names", by iterating through a range of i in range (index, index+25) so that it only does 25 per run. Then I commit these changes and nothing is returned.
- category_table(url)
    - EventBrite has an api url for data on the category ids and their corresponding names. The argument is that url which is called in the main function. I then loop through the category list from the data, which I insert into the Categories table. The changes are committed and nothing is returned in this function.
- calculations()
    - Takes in no arguments. I looped through a list of all of the category ids, and from there, used the COUNT function in order to determine how many of each category id there were in the Names table. I then used the Join function in order to pull the names of each category, joining Names and Categories. I did this so that I could create a dictionary with the names of each category, and the number of each category. This function returns that dictionary.
- extra_calculations()
    - Takes in no arguments. This function does the exact same process as the one above, but uses region ids and joins the Name table and Regions table this time. It returns a dictionary of each region and the amount of events in each region.
- extra_visualization(dict)

    - Takes in the dictionary from the extra_calculations() function. Creates a horizontal bar graph with count on the x axis and region name on the y axis. The data is displayed in magenta. Nothing is returned.
- write_calculations(dict)
    - Takes in the dictionary returned from calculations(). It then opens up our calculations text file, and appends the data to the end of the text file so that there are no overwrites. Each category is on a new line in the file. Returns nothing
- create_visual(dict)
    - Takes in the dictionary from the calculations() function. Creates a horizontal bar graph with count on the x axis and category name on the y axis. The data is displayed in pink. Nothing is returned.
- main()
    - The driver function. The index is initialized here and determined based on how many rows are counted on the Names table, which is how it inserts data in 25 increments at a time. Calls all of the functions above.

<u>TicketMaster</u>

- make_tables(index)
    - This function takes in an index as an argument and is in charge of making sure that only 25 rows of data are inserted into the ticketmaster datatable at a time. The function accesses the api and then iterates through the data to append the data points to 3 separate lists, each list later becoming a column header in the ticketmaster datatable. The function then assigns each genre/event type to the integer value that the genretable datatable later gives the duplicated genre strings. The function then uses the index to go through and add 25 rows at a time into the ticketmaster datatable.
- main()
    - Main takes in 0 arguments, but is in charge of sending the data to the two datatables 25 rows at a time. Within main, there is an if, elif, else statement to increase the index by 25 after every run and ensure that there are only 25 rows loaded into the ticketmaster data table at a time.
- After the main, the data is fetched from the ticketmaster datatable. An empty dictionary is initalized as the fetched data is looped through to create a dictionary with genres as the keys and prices as the values. An average prices dictionary is then created that calculates the average price of events for each genre.
- After completing the calculation, the calculation is written into the Calculations.txt file.
- Lastly, matplotlib is used create a colorful bar graph display of the average price for each genre/event type.

8. You must also clearly document all resources you used. The documentation should be of the following form (20 points)

Date	Issue Description	Location of Resource	Result (did it solve the issue?)

4/16	Could not obtain multiple pages of data. Incorrect API call with wrong pagination parameters.	https://stackoverflow.com/questions/41551419/how-do-i-format-syntax-on-an-api-call-to-page-through-the-results	Yes Can access more than one page of data. Correct API call.
4/19	Creating pie chart with matplotlib	https://www.w3schools.com/python/matplotlib_pie_charts.asp https://www.geeksforgeeks.org/plot-a-pie-chart-in-python-using-matplotlib/	Yes
4/20	Relocating legend to not block pie chart	https://www.statology.org/matplotlib-legend-position/#:~:text=To%20change%20the%20position%20of%20a%20legend%20in%20Matplotlib%2C%20you,legend()%20function.&text=The%20default%20location%20is%20%E2%80%9Cbest,avoids%20covering%20any%20data%20points.	Yes
4/20	How to iterate over multiple lists at once	https://www.geeksforgeeks.org/python-iterate-multiple-lists-simultaneously/	Yes, used zip()
4/16	Could not understand what data I could access from API	https://www.eventbrite.com/platform/api	Yes, figured out I needed event id
4/16	“”	https://idratherbewriting.com/learnapidoc/docapis_eventbrite_example.html	“”
4/17	Having trouble getting my main function to be the driver function	https://www.guru99.com/learn-python-main-function-with-examples-understand-main.html	Yes

4/17	Remove list duplicates so I could create table for ids and region names	https://www.geeksforgeeks.org/python-ways-to-remove-duplicates-from-list/	Yes `[*set(region_list)]`