# Final Project Report

## Chase Burdin

## Introduction: what does your project do

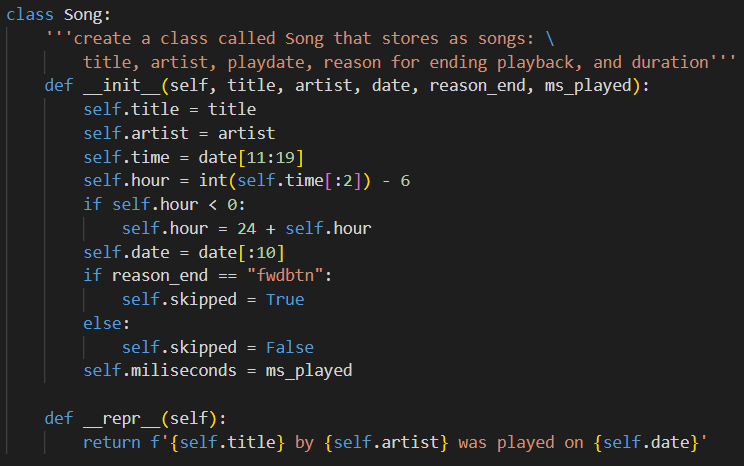
My project takes data from the Spotify json files, that include records for every song the user has every listened to since the creation of their account. It turns this data into a viable format for data analysis, then using various functions, it extracts key insights. This project makes use of many concepts learned in class, such as: functions, classes, dictionaries, lists, loops, and reading from a file.

## Project structure and components:

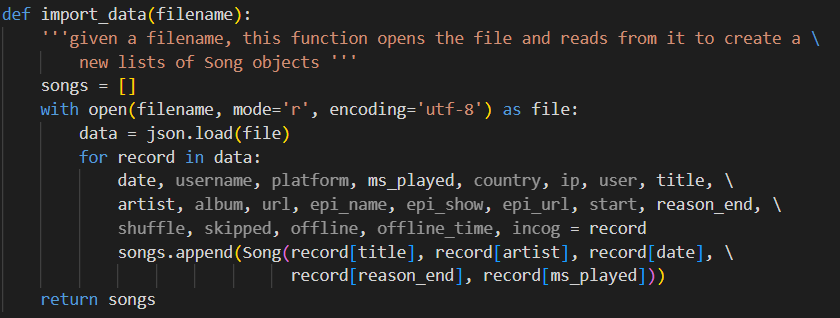
* File: MyData
  + Subfile: ReadMeFirst\_ExtendedStreamingHistory
    - Informs reader of what the shorthand for datapoints means and how to read the overall database
  + Subfile: Streaming\_History\_Audio\_2019-2021\_0
    - Includes streaming history in a json file for declared time range
  + Subfile: Streaming\_History\_Audio\_2021-2022\_1
    - Includes streaming history in a json file for declared time range
  + Subfile: Streaming\_History\_Audio\_2022\_2
    - Includes streaming history in a json file for declared time range
  + Subfile: Streaming\_History\_Audio\_2022-2023\_3
    - Includes streaming history in a json file for declared time range
  + Subfile: Streaming\_History\_Audio\_2023\_4
    - Includes streaming history in a json file for declared time range
  + Subfile: Spotify\_Analysis.py
    - A python files that reads the streaming history files and outputs an interactive data analysis in the python terminal as well as in a graphical visualization

## Describe the key program design:

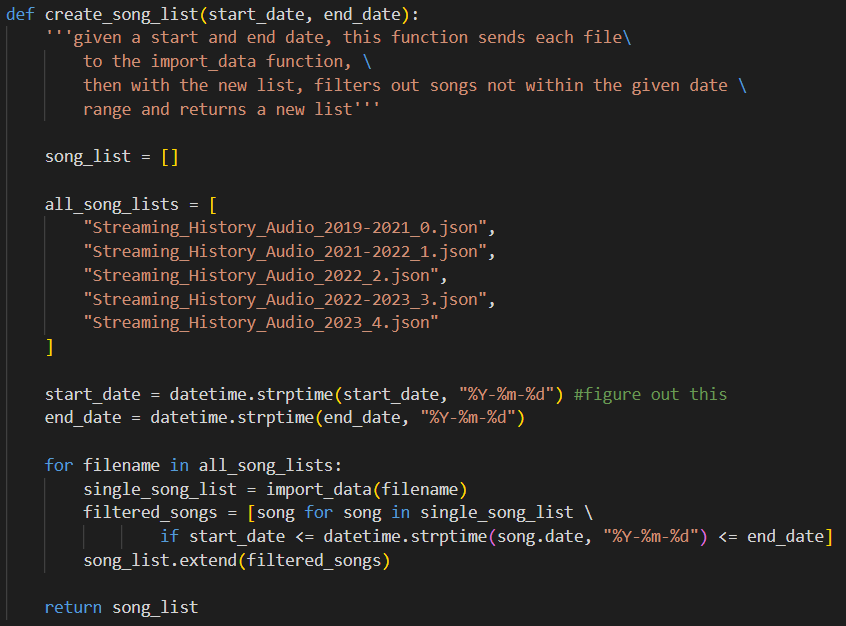
* Class: Song
  + Purpose: Creates a class called Song that stores as songs: title, artist, playdate, reason for ending playback, and duration



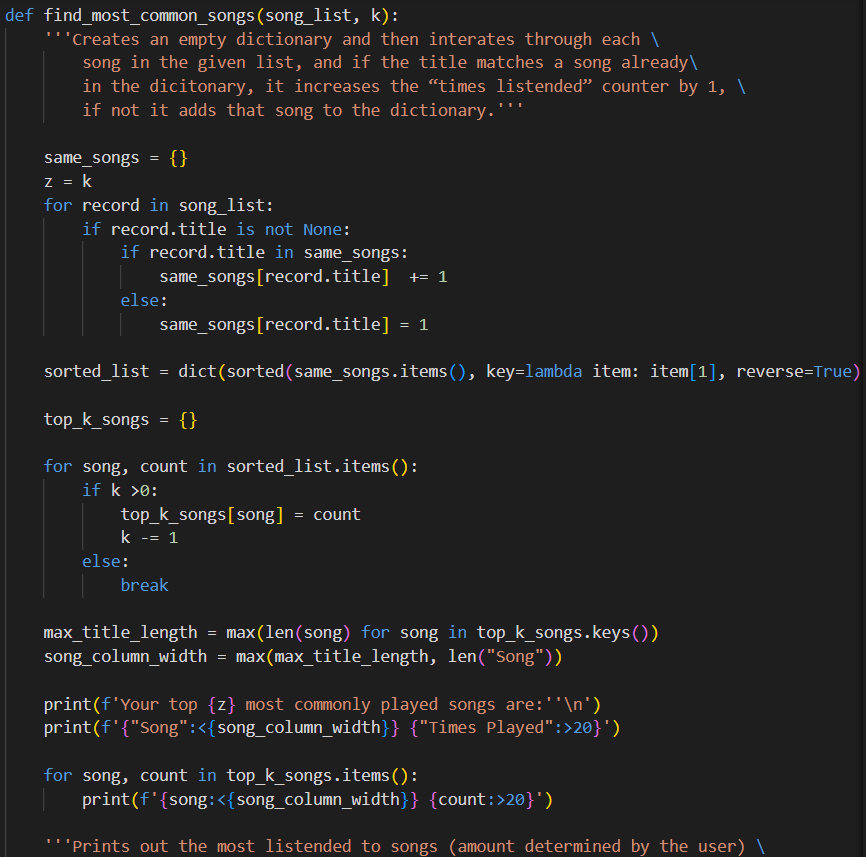
* Function: import\_data
  + Purpose: Given a filename, this function opens the file and reads from it to create a new lists of Song objects



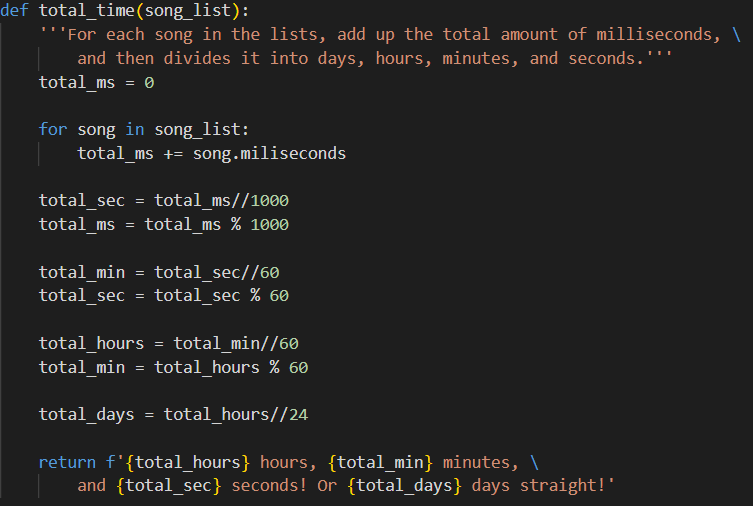
* Function: create\_song\_list
  + Purpose: Given a start and end date, this function sends each file to the import\_data function, then with the new list, filters out songs not within the given date range and returns a new list



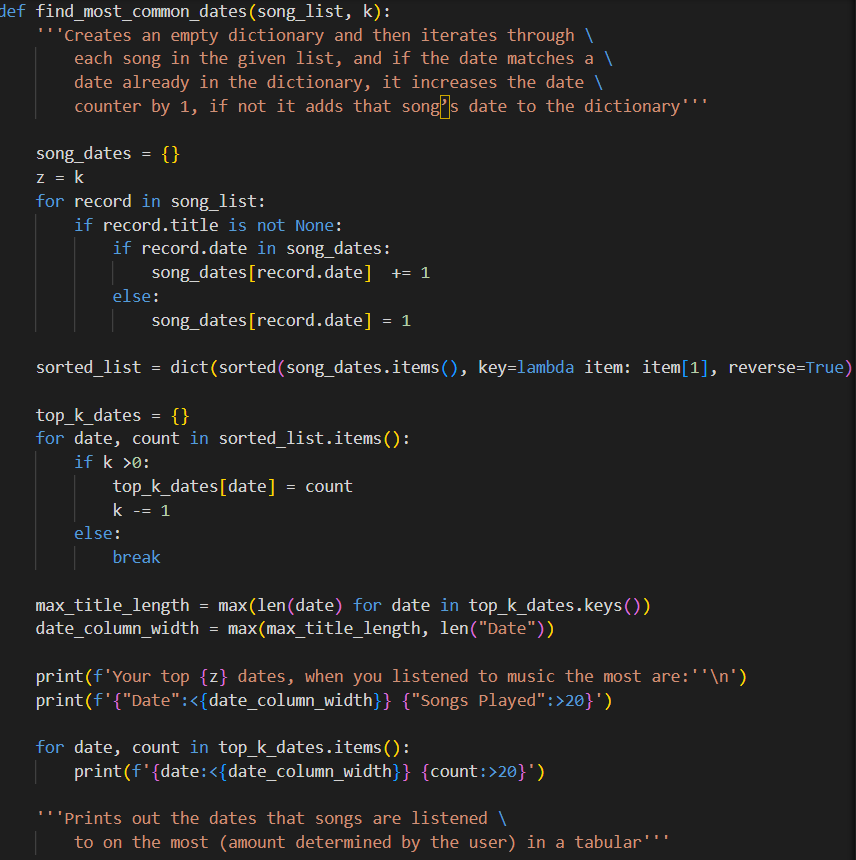
* Function: find\_most\_common\_songs
  + Purpose: Creates an empty dictionary and then iterates through each song in the given list, and if the title matches a song already in the dictionary, it increases the “times listened” counter by 1, if not it adds that song to the dictionary. Prints out the most listened to songs (amount determined by the user) in a tabular format.



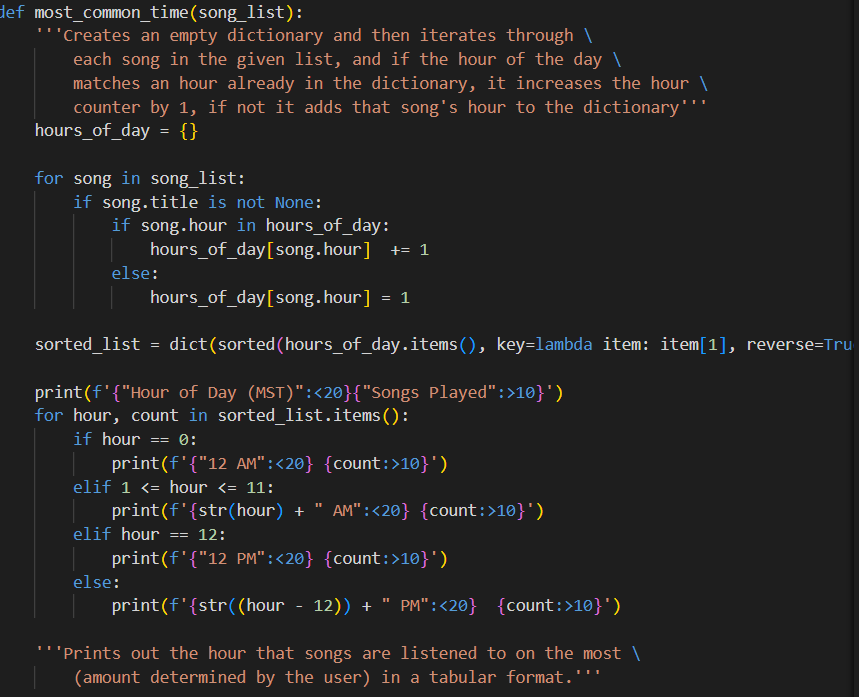
* Function: total\_time
  + Purpose: For each song in the lists, add up the total amount of milliseconds, and then divides it into days, hours, minutes, and seconds.



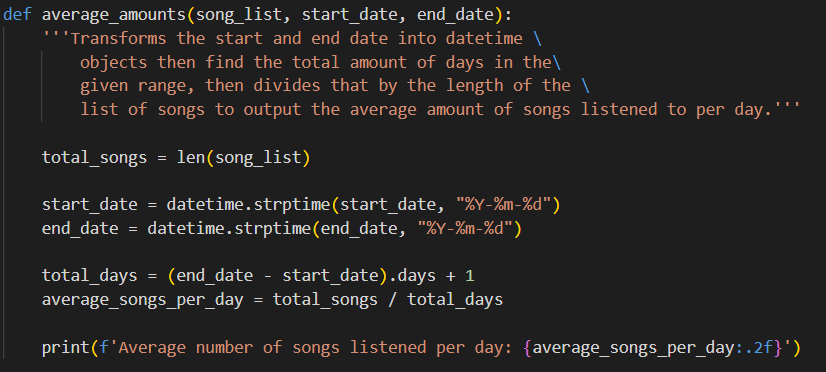
* Function: find\_most\_common\_dates
  + Purpose: Creates an empty dictionary and then iterates through each song in the given list, and if the date matches a date already in the dictionary, it increases the date counter by 1, if not it adds that song’s date to the dictionary. Prints out the dates that songs are listened to on the most (amount determined by the user) in a tabular format.



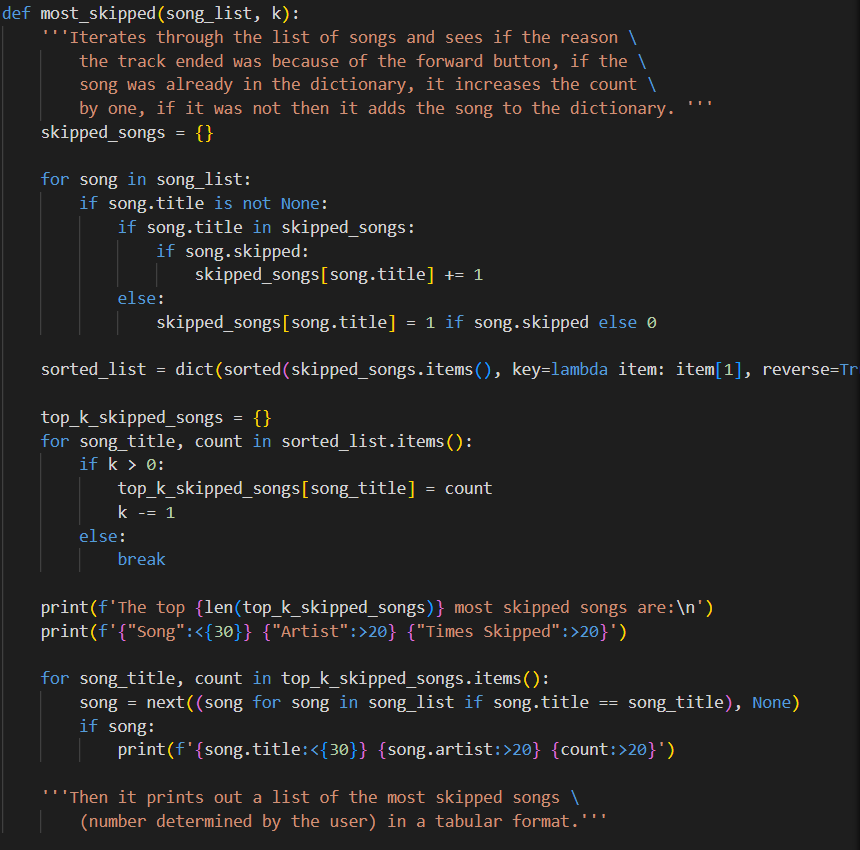
* Function: most\_common\_time
  + Purpose: Creates an empty dictionary and then iterates through each song in the given list, and if the hour of the day matches an hour already in the dictionary, it increases the hour counter by 1, if not it adds that song’s hour to the dictionary. Prints out the hour that songs are listened to on the most (amount determined by the user) in a tabular format.



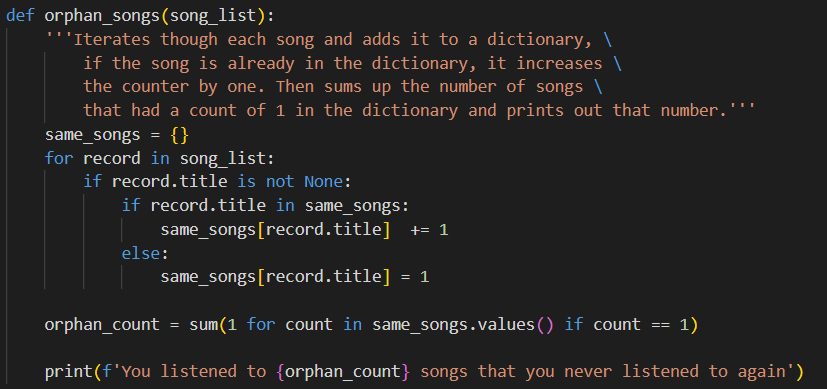
* Function: average\_amounts
  + Purpose: Transforms the start and end date into datetime objects then find the total amount of days in the given range, then divides that by the length of the list of songs to output the average amount of songs listened to per day.



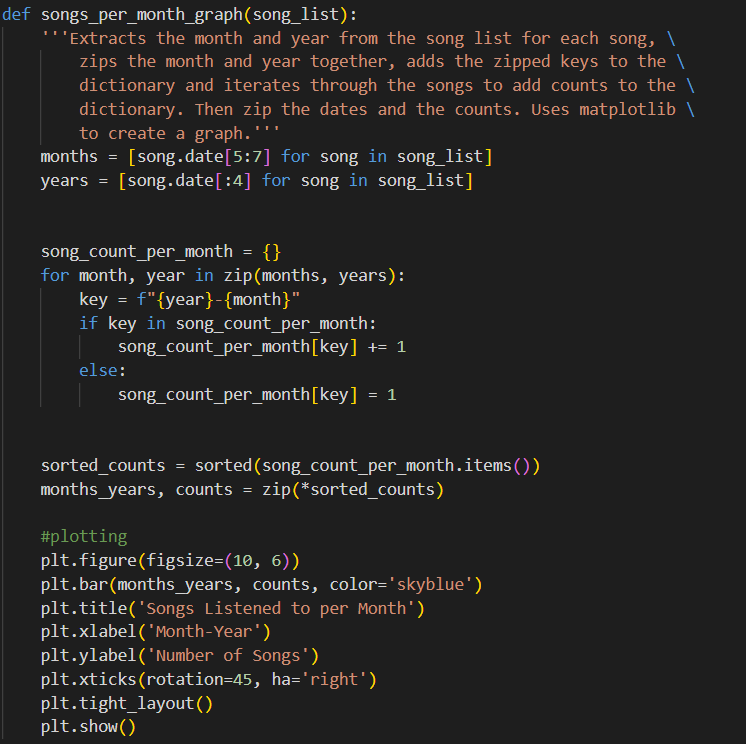
* Function: most\_skipped
  + Purpose: Iterates through the list of songs and sees if the reason the track ended was because of the forward button, if the song was already in the dictionary, it increases the count by one, if it was not then it adds the song to the dictionary. Then it prints out a list of the most skipped songs (number determined by the user) in a tabular format.



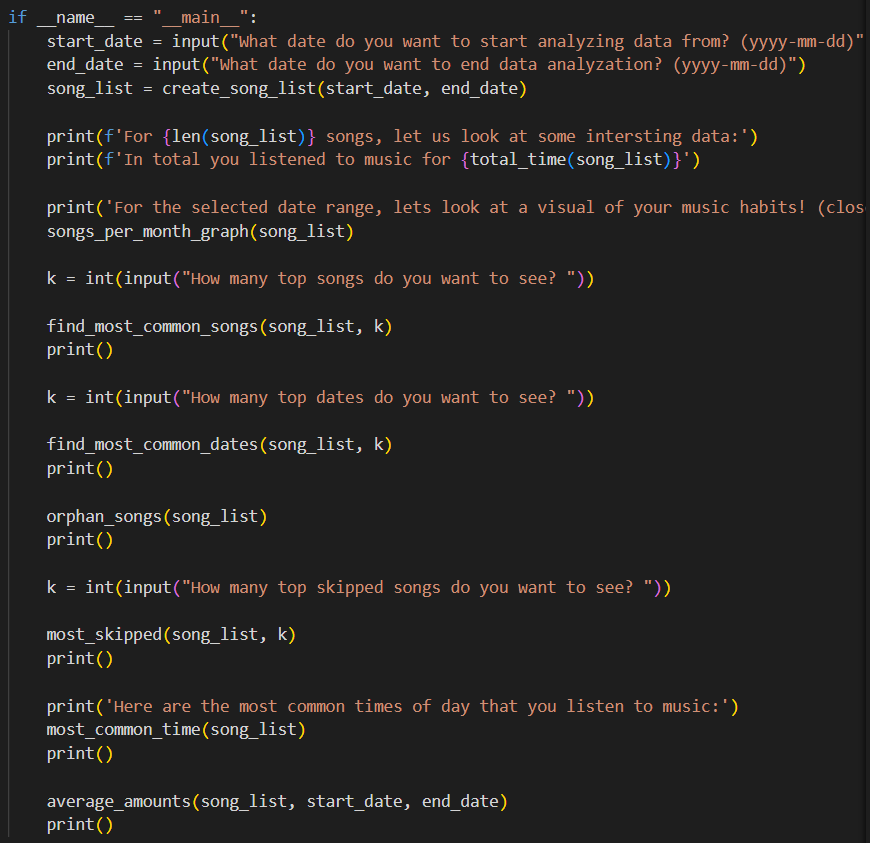
* Function: orphan\_songs
  + Purpose: Iterates though each song and adds it to a dictionary, if the song is already in the dictionary, it increases the counter by one. Then sums up the number of songs that had a count of 1 in the dictionary and prints out that number.



* Function: songs\_per\_month\_graph
  + Purpose: Extracts the month and year from the song list for each song, zips the month and year together, adds the zipped keys to the dictionary and iterates through the songs to add counts to the dictionary. Then zip the dates and the counts. Uses matplotlib to create a graph.



* Main:
  + Purpose: takes inputs from the user to go through each analysis in the project, outputs communicable print statements to guide the user.



* Final Terminal Lines: 340

## Known problems:

* Limitations:
  + Does not handle invalid inputs, assumes the user will enter correct date ranges and types of input
  + Does not have unit tests
  + Does not calculate actual trends in date such as linear regressions
* Bugs:
  + While in the final version the code runs just fine and there are no issues, in the writing of the code many bugs had to be fixed:
    - DateTime bugs: originally were not using the date time so many functions were unoptimized.
    - Reading Json Data: was not reading in correctly, had to add encoding='utf-8' to read properly
    - Table Titles: The length of the song titles were running into the other columns, so I made an adaptable title length value
    - Skipped Songs Value: No matter how I changed the skipped\_songs function it outputted songs that had been skipped zero times, to fix this I realized that the original data often had the skipped value as “null”, so I instead needed to use the “reason\_end” value to see if the song ended due to the next song being played prematurely.

## Potential future extensions:

Extension 1: Create a GUI interface with a slider for the date range and number of data points, so that the user doesn't have to redo the entire analysis every time they want to see different statistics.

Extension 2: Code more visualizations so that the data is easier to draw conclusions from.

### References: links to resources you have used such as tutorials, data, documentation, etc.

Json Reference: [Working With JSON Data in Python – Real Python](https://realpython.com/python-json/)

Visualizations Reference: [Matplotlib Tutorial - GeeksforGeeks](https://www.geeksforgeeks.org/matplotlib-tutorial/)

Datetime Reference: [datetime — Basic date and time types — Python 3.12.3 documentation](https://docs.python.org/3/library/datetime.html)

Lamda Reference: [How to Use Python Lambda Functions – Real Python](https://realpython.com/python-lambda/)

Zip Reference: [zip() in Python - GeeksforGeeks](https://www.geeksforgeeks.org/zip-in-python/)