Data Analysis of Document Tracker

F20SC COURSEWORK 2

Harkins, Brodie 100% & Basri, adam 100%

2024

# 1. Introduction

## 1.1 Purpose

The purpose of this report is to document the development, design, and functionality of the Python-based document tracking application for coursework. The report details the implementation of key features, usability considerations, testing outcomes, and lessons learned during development. Some of these key features involve being able to analyse datasets of various sizes. For datasets of 3 million, the program should attempt to complete each task within a reasonable time frame.

## 1.2 Remit and Assumptions

The primary remit was to create a command-line and GUI-based application that analyses document views and interactions using JSON data

Assumptions include

* Users will provide valid input file paths and parameters during execution.
* JSON files contain well-formed event data.
* Developed using Python 3
* Designed to run on a Linux platform
* Tasks can be completed from a CLI (Command Line Interface)
* A Graphical User Interface (GUI) can be used to complete the same tasks launched from either the Command Line Interface or from the program itself
* Assume there are no incorrect datasets

# 2. Requirements Checklist

|  |  |
| --- | --- |
| **Requirements** | **Implemented (Yes/No)** |
| **Task 2A:** Views by Country | Yes |
| **Task 2B:** Views by Continent | Yes |
| **Task 3:** Views by Browser | Yes |
| **Task 4:** Reader Profiles | Yes |
| **Task 5:** 'Also Likes' functionality | Yes |
| **Task 6:** Visualisation of Graph | Yes |
| **Task 7:** GUI | Yes |
| **Task 8:** Command-line Interface | Yes |

# 3. Design Considerations

**Accessibility:**

* Provide clear error messages and prompts when the incorrect inputs are provided.
* Visualisations include well-labelled charts and graphs
* Make sure that the application flows smoothly without the user getting lost

**Usability:**

* There is a menu feature when using command-lines to select tasks intuitively.
* There is a GUI feature that offers file browsing, text editing, and graph generation in a user-friendly layout.

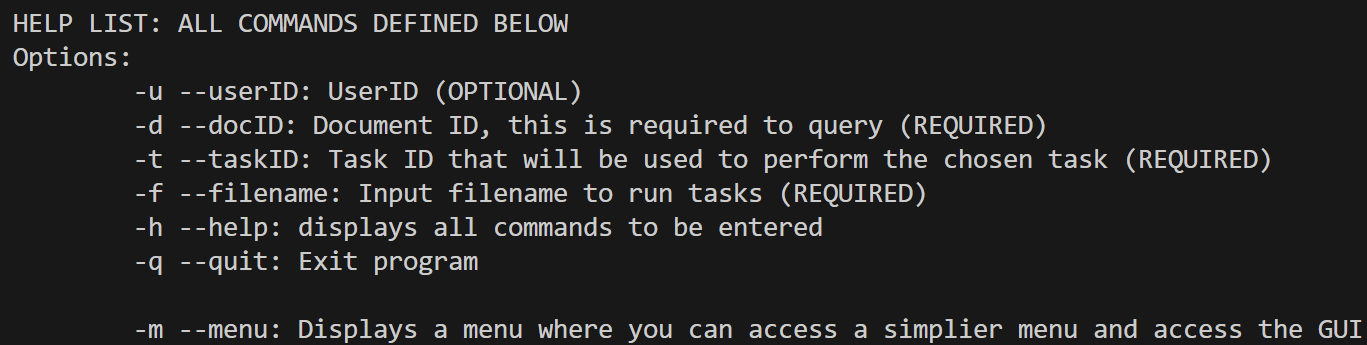
# 4. User Guide

**Command-line Application:**

1. Run the cw2-main.py file in a terminal or command prompt.

Command: **python3 ./cw2 --help**

This will display a list of options that you can input to complete a task

From the image above, the input options are:

* **-u userID**: This is optional but can be used to be searched for.
* **-d docID:** This will contain the document ID that will be used to search through JSON files
* **-t taskID**: This is the task which will be performed. This is required as each task has its own number which is required by the specification.
* **-f filename:** This contains the file location of our JSON files.
* **-m menu:** This will allow the user to access a menu where they can manually select a task and perform a task. GUI is also accessible from this area of the program.

1. CLI COMMAND STRUCTURE

When entering commands using the Command Line Interface, use the following format:

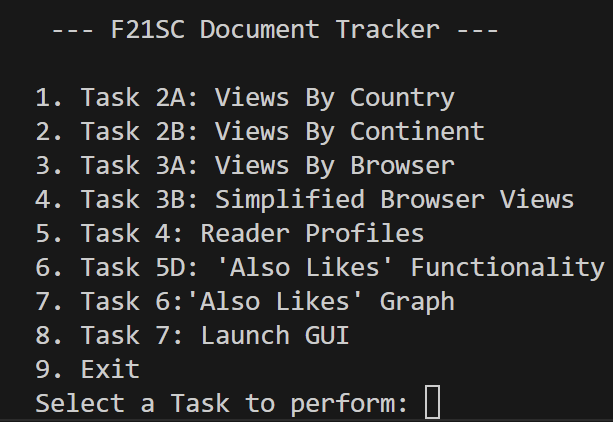
***Python3 ./cw2 -f filename –d documentID –u userID(OPTIONAL) –t taskID***

Using this structure, simply swap out the placeholder names for their respective value. So, filename will be replaced by the json\_file\_path etc. Note that userID is optional, this can be entered or left out of the command line.

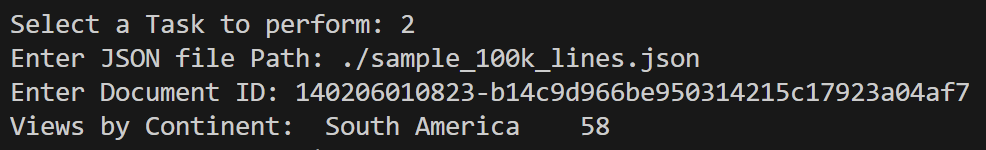
1. Follow the menu options to select a task.



By entering the command above, you will be taken out of the CLI and shown a menu displaying all available tasks.



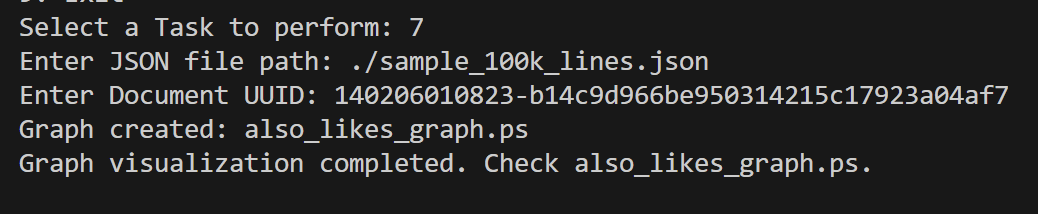
Upon loading the menu, you will be prompted to select a task. This is where the user can manually enter JSON files and document IDs to compute these tasks. As an example, Task 2B is selected and a json file is entered



In the image above, the user is prompted to enter the file path which will always be the path to where the JSON file is saved. If it’s in the same directory, “./json\_file\_path” will suffice.

1. Provide the required file paths and parameters when prompted.
2. Outputs are displayed or saved to specified locations.

When it comes to Tasks such as Task 6 where a .dot file is produced from the given input.



When task 6 is performed, entering the JSON file with the correct document UUID it will produce a .dot file alongside a PostScript file. This file can be located within the same directory. By reiterating this process, the files will simply be overwritten.

**GUI Application:**

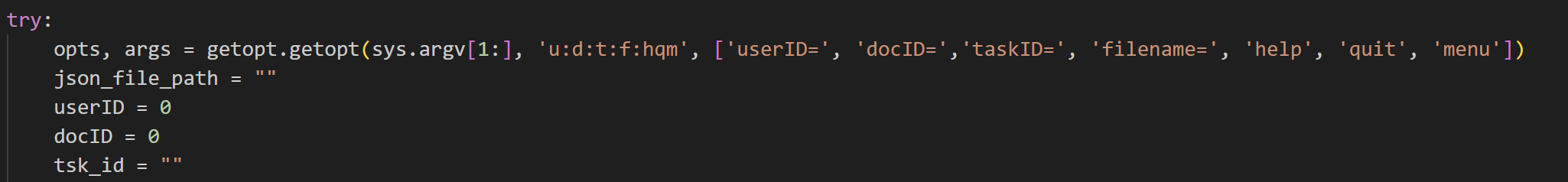
1. Launch the GUI by selecting Task 7 from the menu.
2. Use the “Open .dot File” button to load a .dot file for editing.
3. Modify the file in the text box and save changes using the "Generate Graph" button.
4. Specify an output path for the .ps file to visualize the graph.

# 5. Developer Guide

A screenshot of a computer program

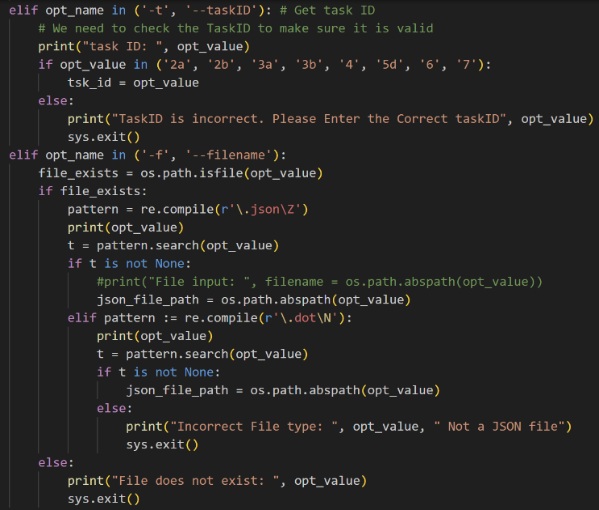
Description automatically generated

**cw2:** Contains the main menu and logic for task selection

When launching the application, the main method will begin the Command Line Interface where the user has a couple of options. 

The image above shows the creation of the input commands that can be utilized by the user. Creating an option to input commands such as userID, docID, taskID and filename. There are also a few extra options such as “help” which will display all available commands that can be used and “menu” which will take the user out of the Command Line Interface and display a menu where the user can manually enter json files and compute tasks.

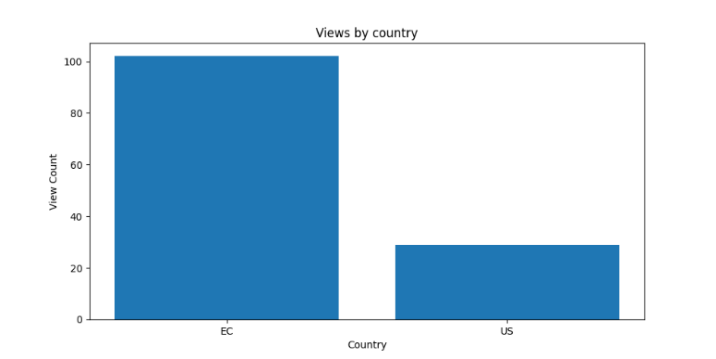
From the above image, we declare our variables and create our inputs. We must ensure that any values entered by the user are appropriate and do not violate any boundaries.

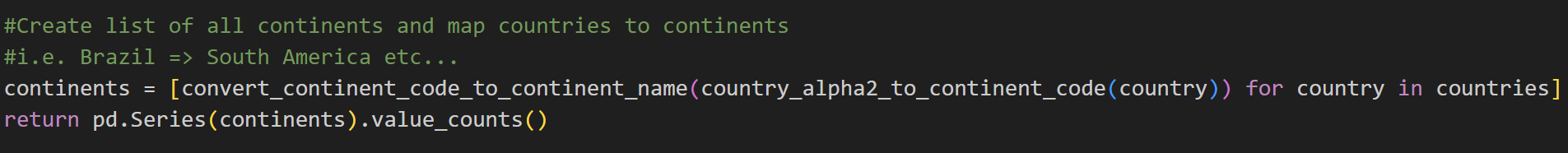


Handling files that are entered by the user are checked to ensure that the correct file type is being used, alongside checking that the taskID entered aligns with the tasks available. Any values entered that violate these conditions, an error message will display, and the user will be forced to try again.

**Task2.py:** Processes and visualises views by country and continent

Using a dictionary to hold all countries and their respective information relative to the country code. From this, we open the file given and loop through holding any new country and their count (How many times it appeared within the JSON file). Once this process has been complete, it will call the function using matplotlib to plot the bar chart itself. Using the doc ID (Using the doc ID: 140228030434- d5c063c15739f1060d8146d5f19160d1 & file: “./sample\_100k\_lines.json” it will produce this result below:



When it comes to getting continent count, there are a few ways to tackle this problem. Using pycountry which will allow mapping from country to continent without too much hassle. The way this function will work is we obtain data from the json file, once this has complete, it will then create a list of countries. Once complete, it will then create a list of continents and map the countries to continents. 

**Task3.py:** Views by browser and user agent

Task 3A (process\_views\_by\_useragent): Counts how often each unique user agent string appears in data. This can help in identifying numerous browsers being used to access the system.

Task 3B (process\_views\_by\_browser): Groups the user agent strings into simplified browser name i.e “Chrome”, “Safari” & “Firefox” etc) to analyze broader browser trends. Simplifying user agents allow us to remove additional information that isn’t relevant.

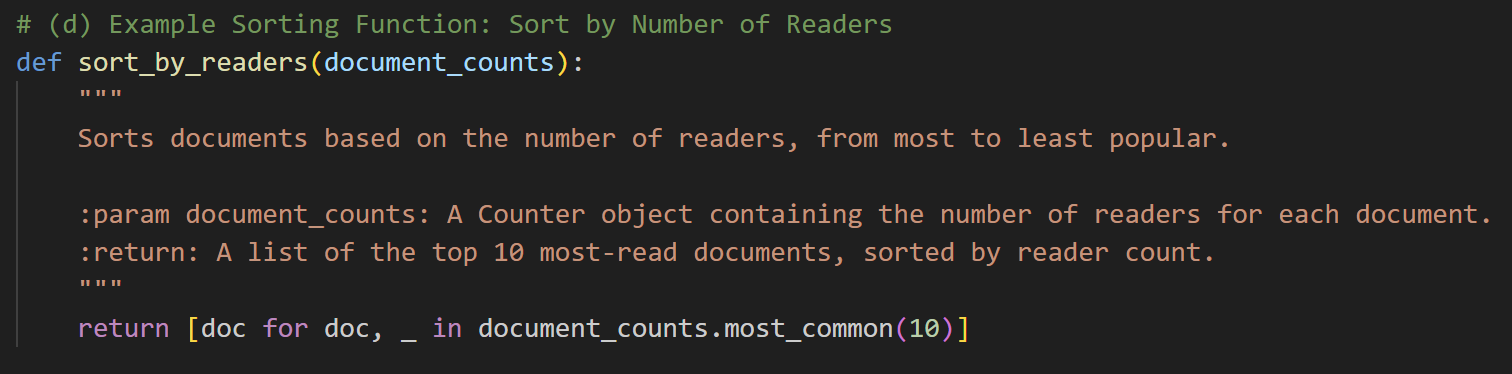
**Task4.py:** Identifies the top 10 reader

Task 4 focuses on identifying and returning the Top 10 readers that have the most engagement by analysing the total reading times. In this task we simply group together the “event\_readtime” with the visitor\_uuid resulting in gathering the top 10 readers.

**Task5.py:** “Also likes” functionality

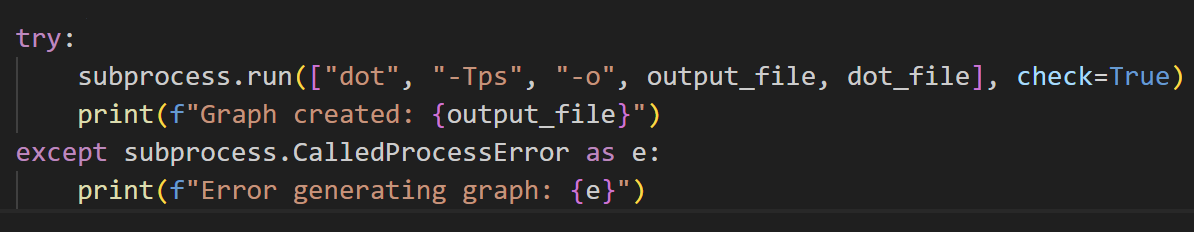
There are 3 functions used to achieve this task.

1. **Retrieve Visitors by document (get\_visitors\_by\_document)**: This function will return the users who have read the file given (subject\_doc\_id). Filtering bsed on documentID and ensuring the interaction type is “reader”.
2. **Retrieve Documents by Visitor(get\_documents\_by\_visitor)**: This function will retrieve all documents read by a specific visitor.
3. **“Also Likes” Recommendation System (also\_likes):** This function will suggest documents related the current document. It achieves this by Identifying visitors who read the specific document, compiling other documents read by those visitors and ranking the related documents using a providing sorting function.
4. **Sorting Function (sort\_by\_readers):** This acts as our sorting function as it will sort documents by the number of unique readers. It extracts the top 10 most-read documents.



**Task6.py:** “Also likes” graph using functionality from Task5.py

When generating the graph for this task, graphviz can be used to help aid its creation. Using Python’s subprocess module to run an external command (“dot” from Graphviz) to generate a graph. Code:



**Task7.py:** GUI

Libraries used:

* **json:** For parsing JSON input files
* **matplotlib:** For creating visualisation
* **defaultdict:** For efficient data aggregation
* **graphviz:** For generating .dot files in Task 6
* **pandas:** For reading CSV files in task 2
* **pycountry:** For mapping countries to continents

# 6. Testing

|  |  |  |
| --- | --- | --- |
| TASK | EXPECTED | OUTCOME |
| CLI inputs incorrect task value -> “-t 10” | Error message is displayed, and user is prompted to re-enter task value | Error message is displayed “Task Value is incorrect” |
| CLI – File not found | CLI outputs Error message | Error message is displayed, and user is prompted to re-enter filename |
| CLI – Incorrect file format | When entering a filename, ensure format is either .dot or .JSON else display error message | Error message is displayed, and user is prompted to re-enter their CLI statement |
| CLI – Empty file | Upon entering a CLI command, ensure that the property –f is satisfied and not null | Error message is displayed, and user is prompted to re-enter CLI command |
| CLI – Task ID = Null | Upon entering a CLI command, ensure that property –t is satisfied and not Null | Error message is displayed, and the user is prompted to re-enter CLI command |
| CLI – Enter –m to load menu | Upon entering the command –m, take the user to the menu where they can view and compute tasks | Menu is displayed showing the user all available tasks to compute |
| CLI – Enter “--help” | Upon entering “--help” display all available commands to be used when computing tasks or accessing the GUI or menu | A list is displayed showing all commands available and their use. |
| CLI - “--quit” | Application exits | Application exits without any bugs |
| CLI – Entering a command for Task 2 | Command entered: *“python3 ./cw2 -f./sample\_100k\_lines.json -d 140228030434-d5c063c15739f1060d8146d5f19160d1 -t 2a*” | Given the file input “-f”, task id “-t” and documentID “-d” we get the result of “***TASK 2A RESULT: {'EC': 102, 'US': 29}”*** Testing this input in the menu by manually entering the values results in the same output -> TRUE |
| Testing Task 2 from the menu | User is prompted to enter a json file path and a documentID to compute | Using the values above -> */sample\_100k\_lines.json & 140228030434-d5c063c15739f1060d8146d5f19160d1. We get a result of ==*  {'EC': 102, 'US': 29} |
| Test Task 2B | User is prompted to enter the JSON\_File\_Path and DocID.  DocID = *140228030434-d5c063c15739f1060d8146d5f19160d1140228030434-d5c063c15739f1060d8146d5f19160d1*  JSON\_FILE\_PATH = */sample\_100k\_lines.json* | We get a result of:  South America 102  North America 29  TRUE  An ambiguous warning does display but this does not effect the final result. Warning has been ignored for now |
| Test 3a | User is prompted to enter a json\_file\_path. | Using the same “./sample\_100k\_lines.json” we get a successful output of all browsers albeit a messy output but Task 3B will take care of this |
| Test 3B | User is prompted to enter a json file path | Using the same file:  Browser Count: {'Chrome': 49174, 'Opera': 1015, 'Mozilla': 52150, 'Other': 62}  A graph is also generated from this as “barChart.png” this is overridden each time a task is called that uses “plot\_chart()” |
| Test 4 | Should return the top 10 readers given a JSON\_file | Using “./sample\_100k\_lines.json”  Result:  Top 10 Readers:  1. User ID: c7b400e46341b0e9, Total Time: 40090152 seconds  2. User ID: 096f89afa74f5f1e, Total Time: 20468752 seconds  3. User ID: 57112d61a3b68d97, Total Time: 6325550 seconds  4. User ID: 0f2e74852bd6d3e6, Total Time: 5664511 seconds  5. User ID: e529f034d3430af2, Total Time: 5356278 seconds  6. User ID: c67049a2de037d64, Total Time: 2361603 seconds  7. User ID: 8112a67d883a5c87, Total Time: 1610168 seconds  8. User ID: 195c38fed75cd271, Total Time: 1046538 seconds  9. User ID: e4fa7ec963961ff8, Total Time: 949020 seconds  10. User ID: 3a3f1fff0583eae6, Total Time: 926433 seconds |
| Test Task 5A | User is prompted to enter a json\_file\_path and a docID | Using the docID: 140206010823-b14c9d966be950314215c17923a04af7  Result:   1. Document UUID: 120919141545-1d96ec7477c24feab8f963e195818329 2. Document UUID: 140217030525-02371963a4277fc84c1da3666c1c29d9 3. Document UUID: 111025114646-19d940a727ae48df91e742123420dca6 4. Document UUID: 100506192710-9a796e48b16e4ed3bc5be8ceedff6b56 5. Document UUID: 121218093303-02a195c1bf2b4bc9b88c8af4c89fe37a |
| Test TASK 6 | User is prompted to enter a json\_file\_path and a docID. Using the doc id from the last test case and the same json file. | Select a Task to perform: 7 Enter JSON file path: ./sample\_100k\_lines.json Enter Document UUID: 140206010823-b14c9d966be950314215c17923a04af7  RESULT:  Graph created: also\_likes\_graph.ps Graph visualization completed. Check also\_likes\_graph.ps. |
| Test Task 7 | Launch GUI | GUI is successfully launched |

# 7. Conclusion

## 7.1 Achievements

What we are most proud of in this coursework is the application itself successfully achieved all objective, including the visualising views by browser and continent, implementation of “also likes” functionality and generating graphs. Integrating GUI (Task 7) also further enhances user interaction and accessibility.

## 7.2 Areas for Improvement

While the application meets the following requirements, there are aspects that could be improved. First, the application itself relies heavily on external libraries, and optimising certain operations could reduce dependencies or improve performance. The other area that could have been improved is more extensive testing as it could identify edge cases and further ensure robustness. Due to an ambiguous error that takes place during Task 2B, that would be the focus in repairing this error, however, this doesn’t affect the overall functionality of the system, if time wasn’t an issue the structure of the code would be solved to avoid future ambiguous errors from occurring.

## 7.3 Scripting vs System Languages

Developing the application in Python highlighted the flexibility and speed of scripting languages for prototyping and data-driven tasks. Python’s vast libraries and concise syntax allowed for rapid development and integration of advanced functionalities. As an example, tasks such as visualising data and processing JSON files were made much easier due to the availability of libraries such as json and matplotlib.

By comparison, the earlier experience of developing a web browser in C# using Windows Form offered insights into system languages. C# provides strong type-checking, better performance for resource-intensive tasks, and a robust framework for building GUI-based applications. However, it takes more effort to implement functionalities in comparison to Python. While C# excels in creating high-performance applications, Python is better suited for iterative development and data-heavy applications dues to its ease of use and library support

# 8. Reference

W3Schools (n.d.) *Matplotlib Pyplot Tutorial*. Available at: <https://www.w3schools.com/python/matplotlib_pyplot.asp> (Accessed: 9 December 2024).

Python Software Foundation (n.d.) *Tkinter — Python interface to Tcl/Tk*. Available at: <https://docs.python.org/3/library/tkinter.html> (Accessed: 9 December 2024).

Python.org (n.d) C-style parser for command line options. Available at:

<https://docs.python.org/3/library/getopt.html>

(Accessed 10th December 2024).

Geeks for Geeks (n.d) Bar plot in Matplotlib. Available at:

<https://www.geeksforgeeks.org/bar-plot-in-matplotlib/>

(Accessed 10th December).

# 9. Contributions

|  |  |
| --- | --- |
| Adam | Brodie |
| 25% of Report | 75% of Report |
| Cw2 file | Cw2 file |
| Task 4 | Task 1 |
| Task 5 | Task 2 |
| Task 6 | Task 3 |
| Task 7 | Task 8 |
| Video |  |

To clarify

Adam: 100%

Brodie: 100%

Total: 200%