

# Data Analysis of Document Tracker

---



**Prepared by:**  
**Akshay Garg (H00338776)**  
**Irfan Syed (H00389591)**

**Prepared on:**  
**7<sup>th</sup> December 2023**

**Course Name:**  
**Industrial Programming (F20SC)**

**Group Number:**  
**12**

# INTRODUCTION

This report is about coursework 2, where we built a Data Analysis of a Document Tracker using Python. The main purpose of this report is to demonstrate how we made this application, tested it, and see if it works as per the requirements mentioned in the coursework specifications. We want to check if our application does everything it was supposed to and is ready to be used as intended.

In this report, we will include easy-to-follow guides for both users and developers for the future. For users, we will explain how to use the application smoothly whereas for developers, we will talk about the structure of the application and how to maintain it, so it keeps working well. At the end of this report, we will share what we learned from doing this project. We believe that this project was not only about developing a useful tool but also about learning practical knowledge in data analytics and software development.

## REQUIREMENTS

The following table provides a detailed breakdown of the requirements as specified in our coursework specifications. For each requirement listed in the coursework specifications, we have included a corresponding status indicator. This status shows whether the requirement has been successfully implemented in our application or not. This structured approach ensures that we have addressed all requirements of the project as outlined in the coursework guidelines.

Sr. No.	Description	Required	Completed
1	The application should be designed to operate on Windows, Mac, or Linux.	Yes	Yes
2	The application should be developed using Python 3.	Yes	Yes
3	The application uses appropriate libraries for tasks such as input handling, data processing, and data visualization.	Yes	Yes
4	The application should take a string as input, which uniquely specifies a document and return a histogram of countries of the viewers.	Yes	Yes
5	Group the countries by continent and generate a histogram of the continents of the viewers.	Yes	Yes
6	The application should return and display a histogram of all browser identifiers of the viewers.	Yes	Yes
7	Process the input of the browser's string, so that only the main browser name is used to distinguish them and again display the result as a histogram.	Yes	Yes
8	Determine the total time spent reading documents for each user. The top 10 readers should be printed based on this analysis.	Yes	Yes
9	Implement a function that takes a document UUID and returns all visitor UUIDs of readers of that document.	Yes	Yes
10	Implement a function that takes a visitor UUID and returns all document UUIDs that have been read by this visitor.	Yes	Yes
11	Implement a function to implement the "also like" functionality, which takes as parameters the above document UUID and visitor UUID.	Yes	Yes

Sr. No.	Description	Required	Completed
12	Provide a document UUID and visitor UUID as input and produce a list of top 10 document UUIDs as a result.	Yes	Yes
13	Generate a graph that displays the relationship between the input document and all documents that have been found as “also like” documents.	Yes	Yes
14	To read the required data and to display the statistical data, develop a simple GUI based on tkinter that reads the user inputs.	Yes	Yes
15	The application shall provide a command-line interface to test its functionality in an automated way.	Yes	Yes
16	Implement a function to count unique visitors.	No	Yes
17	Develop a function to calculate number of unique documents.	No	Yes
18	Create a function to calculate the average reading time per document in minutes.	No	Yes
19	Construct a function to count the number of visits.	No	Yes
20	Develop a function that visualizes the distribution of visitors by country without any document UUID or visitor UUID.	No	Yes
21	Develop a function to return top readers with the number of documents read and average reading time of the reader.	No	Yes
22	Develop a function to return top documents with the number of readers for the document and average reading time of the document.	No	Yes
23	Develop a function to plot the countries of the users, which is independent of visitor id or document id.	No	Yes
24	Develop functions to return count of users on hourly basis, daily basis, monthly basis, and yearly basis.	No	Yes
25	Develop a GUI for the Admin Dashboard where the user can see all the global metrics, independent of document ID or the visitor ID.	No	Yes

# DESIGN CONSIDERATIONS

## Choice of Data Structures

In the script, the use of lists and dictionaries as primary data structures plays a key role in managing and manipulating large datasets effectively. Lists, being ordered and dynamic collections, are instrumental in preserving the sequence of data, which is important for time-sensitive analysis like plotting or tracking user interactions. Their inherent flexibility to accommodate varying data sizes and types makes them particularly suited for handling user-generated data that often varies in volume and structure. Moreover, the ability to iterate over lists efficiently is a significant advantage when processing large datasets, essential for operations such as aggregating values or applying functions across elements.

Dictionaries provide an efficient means of data storage and retrieval through key-value pairs. This structure is particularly beneficial for quickly accessing specific data attributes, like user IDs or document IDs, which is a common requirement in data analysis. The unique keys in dictionaries ensure data integrity by preventing duplicate entries. Additionally, dictionaries support fast lookups, which is crucial when dealing with large datasets requiring frequent access to specific elements. The combination of these two data structures in the script offers a robust framework for organizing, accessing, and processing large volumes of data efficiently, making them ideal for complex data analysis tasks in various applications.

## Modular Design

Each major component of the dashboard (like cards, charts, and tables) is encapsulated in separate functions. This modular approach makes the code more maintainable and scalable. It allows easy modification or addition of new components without disrupting the overall structure.

## Interactive Data Visualization

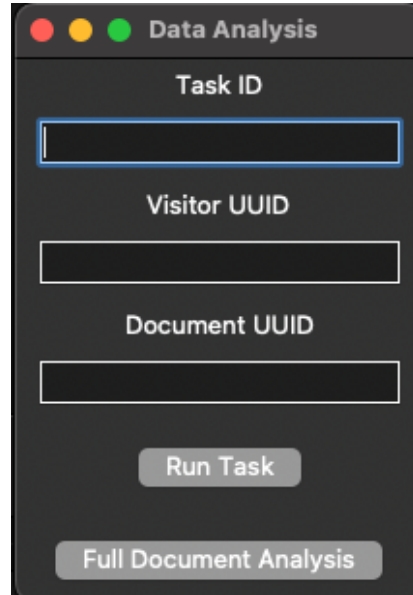
The use of matplotlib for plotting charts and mplcursors for adding interactive features enhances the user experience. Interactive charts enable users to engage more deeply with the data, such as hovering over a bar graph to see specific values. This interactivity not only makes the data more accessible but also more informative.

## Clean and Informative Cards

The `create_card` function produces card-like widgets that display key statistics in a clean and easily digestible format. These cards provide a quick overview of important metrics (like unique visitors or average reading time), which is essential for any dashboard. The use of different background colours for each card aids in distinguishing between different data points briefly.

# USER GUIDE

To run the application, the user can run a single executable file in which they can enter the task id, visitor id and document id. Or the user can run using the terminal for testing. We will be explaining the tasks below.



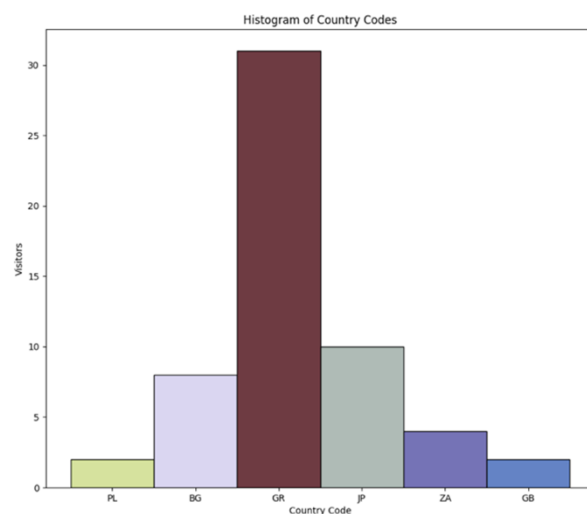
The screenshot shows a window titled "Data Analysis" with a dark background. It contains three input fields labeled "Task ID", "Visitor UUID", and "Document UUID". Below these fields are two buttons: "Run Task" and "Full Document Analysis".

## Task 2a

In this task, the user can plot countries for specific documents by entering the terminal input as:

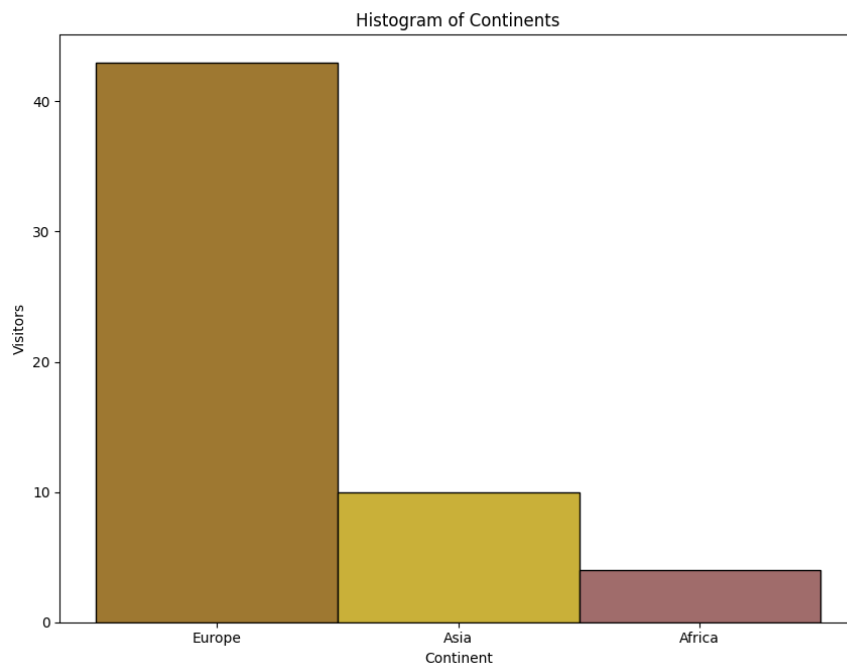
```
python3 app.py -d 131224090853-45a33eba6ddf71f348aef7557a86ca5f -t 2a -f  
issuu_cw2_train
```

Or they can use GUI to enter the task ID and the document ID. The graph plotted will be something like this:



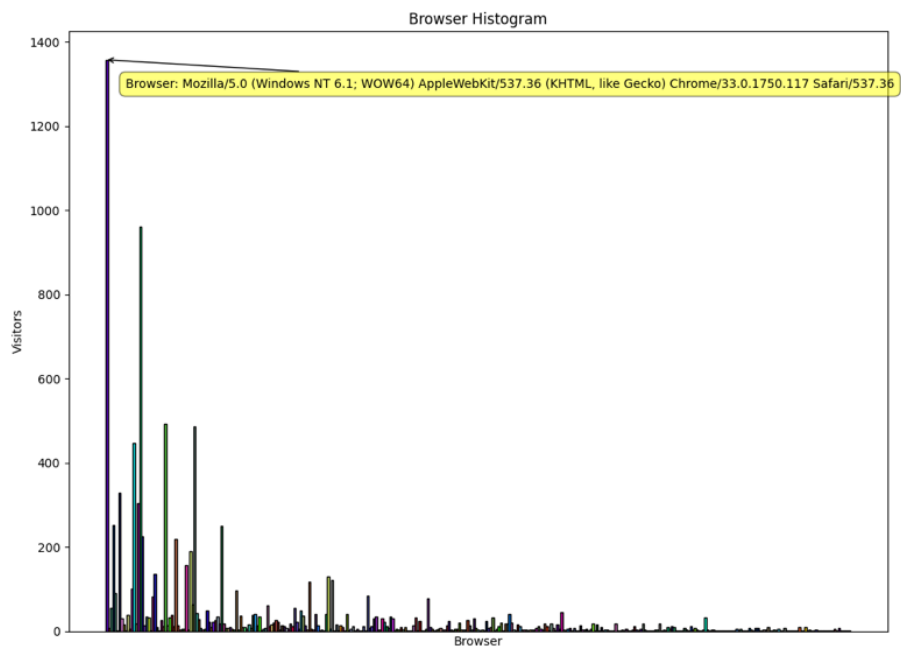
## Task 2b

```
python3 app.py -d 131224090853-45a33eba6ddf71f348aef7557a86ca5f -t 2b -f  
issuu_cw2_train
```



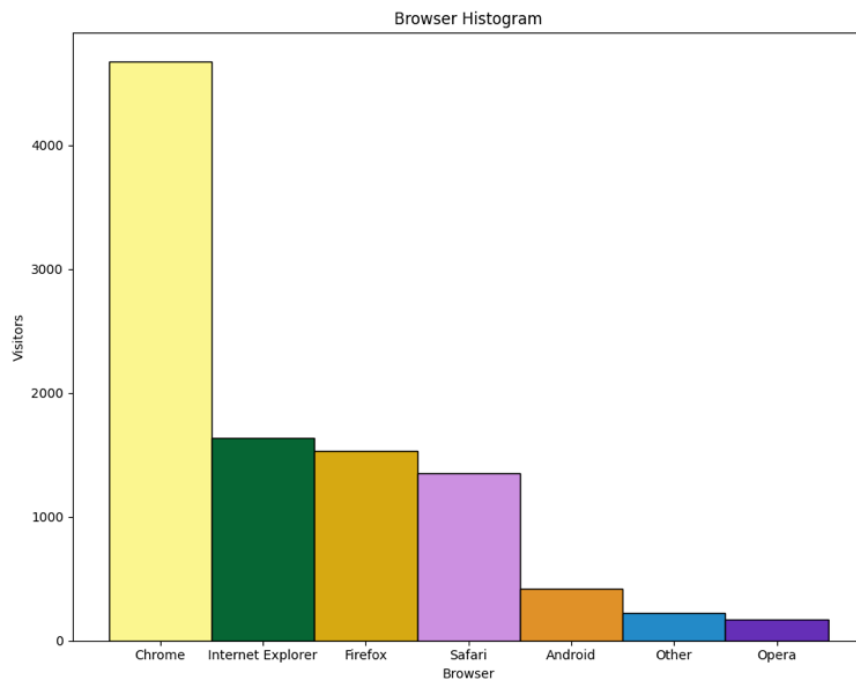
## Task 3a

```
python3 app.py -t 3a -f issuu_cw2_train
```



## Task 3b

```
python3 app.py -t 3b -f issuu_cw2_train
```



## Task 4

```
python3 app.py -t 4 -f issuu_cw2_train
```

Task 4 Results		
=====		
= Top 10 Readers =		
=====		
= User ID = Read Time =		
=====		
= e529f034d3430af2	= 5356 secs	=
= dd326898d5605e63	= 648 secs	=
= 458999cbf4307f34	= 576 secs	=
= 849bb060cb110347	= 486 secs	=
= df70cddb46fd5da	= 239 secs	=
= 14e1e343078d3d75	= 161 secs	=
= 2105dd9bc68afb9d	= 109 secs	=
= e1178362fc11d6ba	= 98 secs	=
= b9caded38e707eca	= 98 secs	=
= da0df8a63107e139	= 98 secs	=
=====		

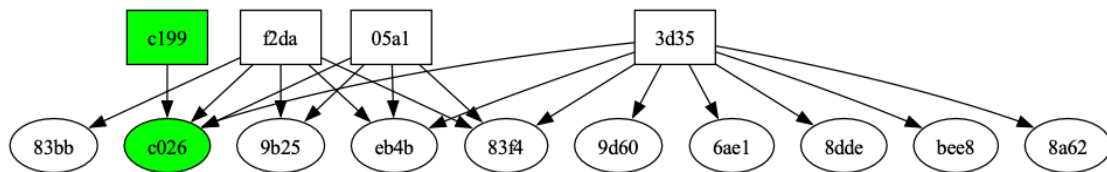
## Task 5d

```
python3 app.py -d 100713205147-2ee05a98f1794324952eea5ca678c026 -u  
489c02f3e258c199 -t 5d -f issuu_cw2_train
```

```
Task 5d Results  
=====   
= Top 10 Documents also Read by The Visitors =   
=====   
= Document ID =   
=====   
= 131202094202-a4ae3185bc84368f14bff266d276eb4b =   
= 100713205147-2ee05a98f1794324952eea5ca678c026 =   
= 140218233015-c848da298ed6d38b98e18a85731a83f4 =   
= 131218101426-7fe24377c762b8fe53d21b65fcfa9b25 =   
= 131105193559-dbac395e3cc43fc2b0077eaf789183bb =   
= 131121160738-0db9af9dbc996be9fce9d81638868dde =   
= 140227072831-649625805917e1f042bdb1f645d588ff =   
= 140105003611-c7cfe5f7cdeb7072cda8d885cc3be2ea =   
= 140227005600-2a71e9e5c5780a23f540112bd0038467 =   
= 130630171409-40897a57ebbe917e46a0735727ae8945 =   
=====
```

## Task 6

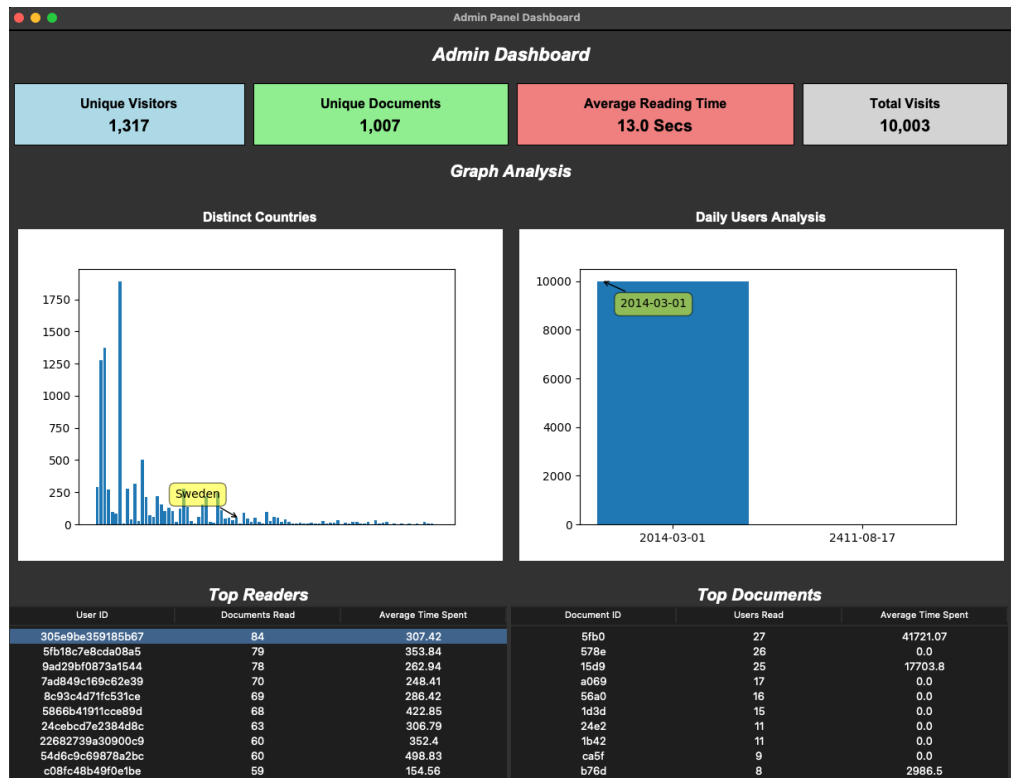
```
python3 app.py -d 100713205147-2ee05a98f1794324952eea5ca678c026 -u  
489c02f3e258c199 -t 6 -f issuu_cw2_train
```



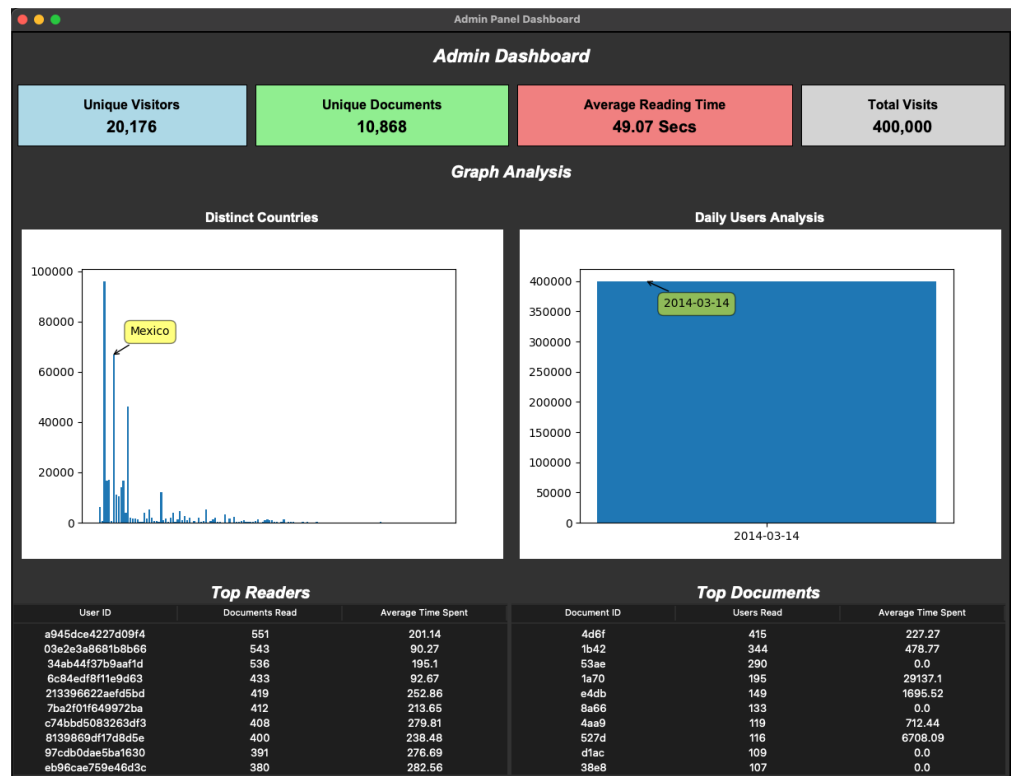


# Analysis Dashboard

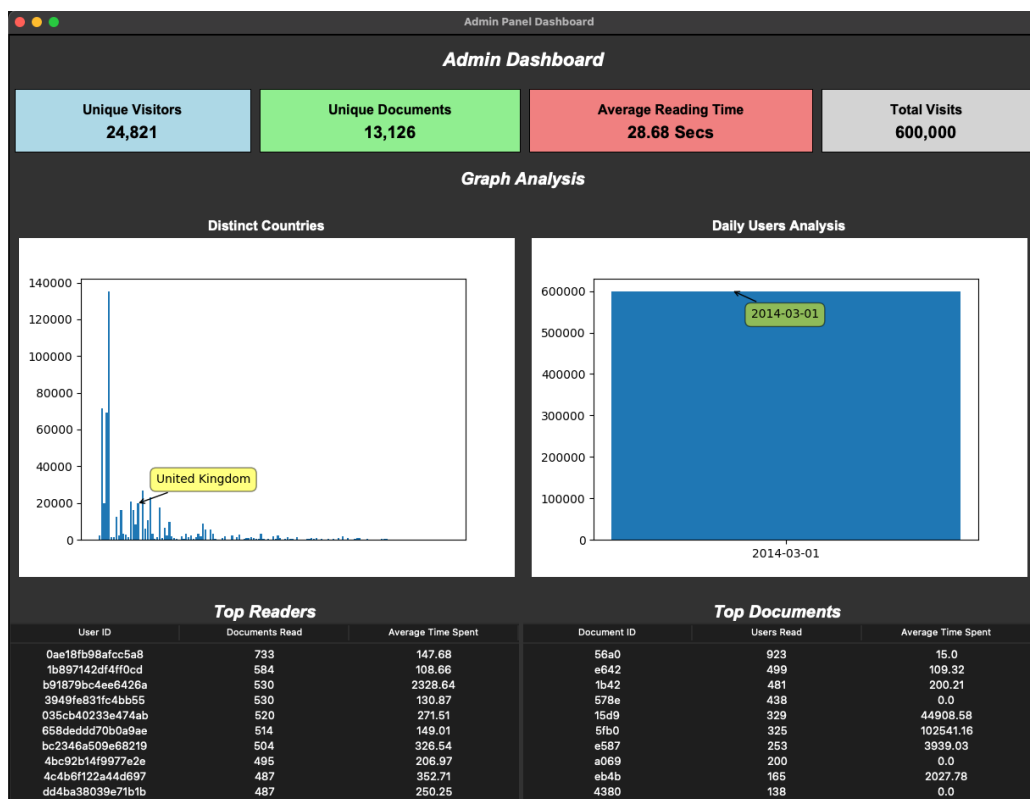
With 10k Records



With 400k Records



With 600k Records



# DEVELOPER GUIDE

## App.py

```
def handle_task_2a(data, doc_uuid):
    ...plot_countries(data, doc_uuid)

def handle_task_2b(data, doc_uuid):
    ...plot_continents(data, doc_uuid)

def handle_task_3a(data):
    ...# Get browser counts
    ...browser_counts = Counter(get_all_browsers(data))

    ...#plot
    ...create_histogram(browser_counts, 'Browser Histogram', 'Browser', hide_xticks=True)

def handle_task_3b(data):
    ...# Get browser counts
    ...browser_counts = get_browser(data)

    ...# Print results and create histogram
    ...create_histogram(browser_counts, 'Browser Histogram', 'Browser')

# Modify the handle_task_4 function in app.py
def handle_task_4(data):
    ...top_readers = user_reader_time(data)

    ...# Create a new Tkinter window for displaying the results
    ...result_window = tk.Tk()
    ...result_window.title("Task 4 Results")

    ...# Create a Text widget to display the results
    ...result_text = tk.Text(result_window)
    ...result_text.pack(padx=10, pady=10, fill=tk.BOTH, expand=True)

    ...# Insert the results into the Text widget with the desired format
    ...result_text.insert(tk.END, "=====\n")
    ...result_text.insert(tk.END, "=====Top 10 Readers=====\\n")
    ...result_text.insert(tk.END, "=====\n")
    ...result_text.insert(tk.END, "=====User ID=====Read Time=====\\n")
    ...result_text.insert(tk.END, "=====\n")
    ...
    ...for i in range(10):
    ...    user_id = top_readers[i][0]
    ...    read_time = top_readers[i][1]
    ...    result_text.insert(tk.END, f"---{user_id:16s}---{read_time:9d} secs---\\n")

    ...result_text.insert(tk.END, "=====\n")
```

## Importing Libraries and Setting Up Functions

This script starts by importing the important tools it needs. It uses `argparse` to get information from the user, like IDs and file names. `json` helps it work with data in JSON format, which is a common way to store and share information. The `Counter` from `collections` counts how many times something appears in the data. For making charts and graphs, it uses `matplotlib.pyplot` and `mplcursors`. The `user\_agents` tool is for understanding information about the browsers people use. `random` is for creating random numbers, and there are special functions from `helper\_func` for particular tasks.

## Functions for Different Tasks

The script has several functions, each for a different kind of job:

1. **task\_2a and task\_2b:** These look at where people are from. `task\_2a` picks out data for a certain document and counts how many visitors are from each country. `task\_2b` takes this a step further by looking at which continents these countries are in and making a graph of this.
2. **task\_3a and task\_3b:** These deal with what browsers people use. `task\_3a` counts the different browsers used. `task\_3b` focuses on the most common browsers like Chrome, Safari, and Firefox, and makes a graph showing their use.
3. **task\_4:** This one looks at how much time readers spend on the site. It finds and lists the readers who spend the most time.
4. **task\_5d:** This function finds out what other documents are liked by people who read a certain document, showing what else they might be interested in.
5. **task\_6:** For showing data visually, this function creates a dot plot showing interactions between users and documents.

## How the Script Works

The script is user-friendly and flexible. It uses `argparse` to take input from the user. It can read and handle data from a given JSON file and is set up to deal with any errors that might happen when opening the file.

The main part of the script decides which task to do based on the task ID given by the user. It then runs the right function for that task with the needed data. This setup makes the script useful for different kinds of data analysis related to how users interact with digital content.

In short, this script is a practical and adaptable tool for analyzing data. It's designed to help understand user behavior and preferences on digital platforms. With its range of functions and ability to handle different data analysis tasks, it's a great fit for studying user interactions.

## Also\_likes.py

This code creates a visual graph showing the relationship between documents and their visitors.

First, we create a function

Get documents by readers : to identify documents of a specific reader

Get readers by documents : function to find readers of a given document

The function also\_likes selects the top 10 related documents based on reader overlap.

The generate\_graph function then visualizes this data using a graph. In the graph, documents and readers are represented as nodes. The document in focus and the optional specific reader are highlighted in green. Edges connect readers to the documents they've read, illustrating the shared interest network among documents and readers.

```

# Get the list of all the readers who visited the document
def get_readers_by_document(data, doc_uuid):

    # Initialize the list to store the readers
    all_readers = []

    for record in data:
        # Check if the subject_doc_id is present in the record
        # if the subject_doc_id is equal to the doc_uuid
        # and if the visitor_uuid is present in the record
        if 'subject_doc_id' in record and record['subject_doc_id'] == doc_uuid and 'visitor_uuid' in record:
            all_readers.append(record['visitor_uuid'])

    # Return the list of readers
    return all_readers

# Get the list of all the documents read by the visitor
def get_document_by_readers(data, visitor_uuid):

    # Initialize the list to store the documents
    all_docs = []

    for record in data:
        # Check if the visitor_uuid is present in the record
        # if the subject_doc_id is present in the record
        # and if the visitor_uuid is equal to the visitor_uuid
        if 'visitor_uuid' in record and record['visitor_uuid'] == visitor_uuid and 'subject_doc_id' in record:
            all_docs.append(record['subject_doc_id'])

    return all_docs

# Get the top 10 documents that are read by the readers who also read the document
def also_likes(data, doc_uuid, visitor_uuid = None, sorting_function=None):

    # If the sorting function is not provided, then use the default sorting function
    if sorting_function is None:
        sorting_function = lambda x: x[1]['count']

    # Get the list of all unique readers who visited the document
    all_readers = set(get_readers_by_document(data, doc_uuid))

    # If the visitor_uuid is provided, then remove the visitor_uuid to the list of readers
    if visitor_uuid is not None:
        all_readers.remove(visitor_uuid)

    # Initialize the dictionary to store the documents

```

## Dashboard.py

### GUI Dashboard Overview

This Python script creates a Graphical User Interface (GUI) dashboard using 'tkinter', a standard GUI toolkit in Python, along with other libraries for data visualization and processing. The dashboard presents a visual and interactive overview of user interaction data, including charts, tables, and key statistics. It's designed to give a user-friendly view of data analytics.

### Different Function

The script uses the 'matplotlib' library for plotting charts and 'mplcursors' for interactive features in the charts. It also employs custom functions from 'card\_num' and 'graph\_data' modules for specific calculations and plotting tasks.

1. `create_card`: Generates a 'card' widget that displays key statistics like the number of unique visitors or documents. Each card contains a title and a value, and is styled with colors and fonts.
2. `create_matplotlib_chart`: Creates a chart frame to display data using matplotlib. It supports bar and line charts and includes options like hiding x-axis ticks.
3. `all_cards`: This function arranges multiple 'card' widgets at the top of the dashboard, showing key statistics like unique visitors and average reading time.

4. `all_charts`: It handles the visualization of data in graphical form. For instance, it can display the distribution of users across different countries or daily user activity using bar charts.
5. `create_table`: It generates a table with specified column names and content, displaying data in a structured format.
6. `place_tables`: This arranges two data tables side by side for comparative analysis or to present different data sets simultaneously.

```
# Function to create a card-like frame
def create_card(parent, title, value, bg_color):
    """# Create a frame for the card
    """
    frame = tk.Frame(parent, bg=bg_color, bd=1, relief="solid", padx=10, pady=10)
    """# Create a label for the title and value
    """
    tk.Label(frame, text=title, bg=bg_color, fg='black', font=("Arial", 16, 'bold')).pack()
    """# Create a label for the value
    """
    tk.Label(frame, text=value, bg=bg_color, fg='black', font=("Arial", 20, 'bold')).pack()
    """# Return the frame
    """
    return frame

# Function to create a chart frame and plot a Matplotlib chart
def create_matplotlib_chart(parent, title, x, y, type="bar", hide_x_ticks=False):
    """# Create a container frame for each chart and title
    """
    container = tk.Frame(parent)
    container.pack(side="left", fill="both", expand=True, padx=10, pady=10)

    """# Create a label for the chart's title
    """
    label = tk.Label(container, text=title, fg='white', font=("Arial", 16, "bold"), padx=5, pady=5)
    label.pack(fill="x")

    """# Create a figure for the plot
    """
    fig = Figure(figsize=(4, 4), dpi=100)
    plot = fig.add_subplot(111)

    """# Plot the data
    """
    if type == "bar":
        plot.bar(x, y)
        """# Use mplcursors to add interactive hover tooltips to the bars
        """
        cursor = mplcursors.cursor([plot, hover=True])
        """# Add the text to the tooltip
        """
        cursor.connect("add", lambda sel: sel.annotation.set_text(list(x)[sel.target.index]))
    elif type == "line":
        plot.plot(x, y)

    """# Hide the x ticks if required
    """
    if hide_x_ticks:
        plot.set_xticks([])
    else:
        plot.set_xticklabels(x)

    """# Create the canvas and add it to the Tkinter window
    """
    canvas = FigureCanvasTkAgg(fig, master=container)
    canvas.draw()
    canvas.get_tk_widget().pack(fill="both", expand=True)

# Function to create the top cards
def all_cards(parent, data):
```

## Full\_gui Function

Finally, the `full_gui` function brings all these elements together into a dashboard. It initializes the main window, sets up a canvas with a scrollbar for smooth navigation, and sequentially adds the cards, charts, and tables to the canvas. The function ensures that all elements are properly aligned and interactive, providing a comprehensive view of the data.

# TESTING

During the development of our application, we primarily relied on manual testing procedures to evaluate its functionality and performance. This involved systematically running the application with various inputs and scenarios to identify any issues, inconsistencies, or unexpected behaviors.

We conducted thorough testing by inputting normal, extreme, and exceptional data to ensure that the application responded correctly in different situations. This approach allowed us to gain a deep understanding of how the system operated and helped us identify and address any issues promptly.

While manual testing provided valuable insights and ensured that the application met our expectations, we acknowledge that automated unit tests, as mentioned earlier, would have offered a more systematic and efficient way to verify the correctness of individual functions. Nevertheless, our manual testing efforts were instrumental in ensuring the overall quality and reliability of the application for its intended use.

## LEARNING FROM CW1

In the following section, we have outlined the main issues that were pointed out to us in the feedback after CW1.

It was pointed out that the commenting was subpar so this time we have ensured to add plenty of meaningful comments. We were also told that the additional features in the last submission were too simple, which is why in this coursework we have added a complex dashboard for additional data viewing.

We received positive feedback for our reports in coursework 1 so we have made considerable effort to maintain the same level of delivery this time as well. The naming conventions used in this coursework are also a lot more refined than the last submission. We have used sensible names for our variables and file names. Last time, the GUI was quite simple and plain, so this time we have tried to incorporate a complex, modern and colored design using tkinter and matplotlib.

## CONCLUSIONS

In conclusion, we are very happy with several aspects of our project. Our code organization has improved significantly since our last coursework submission, with meaningful functions that serve their individual purposes. However, we've identified room for growth. There is a minor error in one part of our project regarding document IDs, which we'll address in future iterations. Additionally, our GUI interface could be better by using dropdown menus and predictive text. In this coursework, we have learned valuable lessons. The significance of error handling consistency, maintaining uniform error messages etc., has shown itself repeatedly in the coursework the importance of reliability and maintenance. Also, the importance of allocating proper time for compiling standalone executables has been highlighted, a lesson we'll carry forward into future projects. For us this project has been a valuable learning opportunity, teaching us the importance of code structure, error handling, and user interface design in python program development. These learnings will guide us in delivering better solutions in the future.