F21SC

INDUSTRIAL PROGRAMMING

HERIOT WATT UNIVERSITY, DUBAI

COURSEWORK-2

DATA ANALYTICS

Done By:

Anugraha Antoo Kanjookaran - H00460688

Archana Tharammal - H00488368

CONTENT

INTRODUCTION

In this project, we developed a data analytics application to analyze user interactions with documents. The application features a graphical user interface (GUI) that allows us to load and process interaction data, generate various histograms (e.g., browser, country, and continent), and identify the top 10 readers based on time spent on documents. Additionally, we have implemented a feature to recommend documents that users who interact with a specific document also like. The system enables filtering by document UUID and visitor ID, ensuring flexibility in analysis. We have also accounted for error handling related to missing or invalid data inputs. Overall, our project simplifies the process of analyzing user behavior with document interactions to derive meaningful insights.

The assumptions taken are as follows:

Dataset Format and Content:

- The dataset is structured (JSON or gzipped JSON) and includes fields like document UUID, visitor UUID, browser, country, continent, and time spent.
- Each record is valid and well-formed.

Document Identification:

- Document UUID (subject_doc_id) uniquely identifies documents for analysis.
- Users input document UUIDs to generate specific insights.

Data Availability and Size:

- Data is valid, complete, and manageable for in-memory processing.

"Also Likes" Functionality:

- Assume shared visitor interactions establish relationships between documents.
- Works with or without a specific visitor UUID input.

Geolocation Data:

- Visitor countries are mapped to continents for generating continent-level histograms.

REQUIREMENTS CHECKLIST

- Implement in Python 3.
- **Views by Country/Continent**: Generate histograms of viewer countries and continents.
- **Views by Browser**: Create histograms of browser identifiers and simplified browser names.

- **Reader Profiles**: Identify and display top 10 readers based on time spent.
- **“Also Likes” Functionality**: Find related documents based on shared readers and return top 10 "liked" documents.
- **“Also Likes” Graph**: Generate a graph showing document relationships using Graphviz.
- **GUI**: Develop a Tkinter-based interface for input and data display.
- **CLI**: Implement a command-line interface for automated testing of tasks.

All the above requirements have been implemented.

Individual Contributions

- Anugraha : Also likes functionality, Also likes graph and GUI
- Archana : Views by Country/Continent, Views by browser, Reader profiles and CLI

DESIGN CONSIDERATIONS

In developing the document tracking application, we focused on ensuring that the program is both efficient and user-friendly. Below are the key design considerations we made.

File Structure

We organized the project in a way that supports scalability and maintainability. The file structure is as follows:

- display_histogram.py
- data_processing.py
- also_likes.py
- top_readers.py
- cw2.py
- gui.py
- main.py

GUI Design

For usability and accessibility, we implemented a Tkinter-based GUI [1]. The design focuses on clarity, ease of interaction, and providing essential functionalities at the user's fingertips. Here are the key design elements we considered:

- Input Fields: We provided input fields for the user to enter the document UUID, user UUID (optional), and file path.
- Buttons: Each task (view country/continent, browser histogram, etc.) is associated with a dedicated button. The buttons trigger specific data processing and visualization when clicked.
- Graphs: Results such as histograms and "also likes" graphs are displayed in a separate window, ensuring a clean layout.

- Error Handling: We added input validation to ensure the user provides valid document UUIDs and file paths.

We kept the interface simple to avoid overwhelming the user, with clear labels and responsive buttons that trigger relevant actions.

Advanced Coding

To handle the large dataset efficiently, we used advanced coding techniques like Dask and Python generators.

- Dask: To manage large files with millions of records, we integrated Dask to enable parallel processing of data. This approach helped us ensure the application remains responsive, even with massive input files [2].
- Generators: We utilized Python generators to process the data line by line. This approach reduces memory consumption, as it allows us to process the data lazily, processing only the necessary portions at a time [3] [3].
- Graphviz: It is an effective tool for creating visual graphs that allow complex relationships to be represented using both directed and undirected networks.

Usability and Accessibility Enhancements

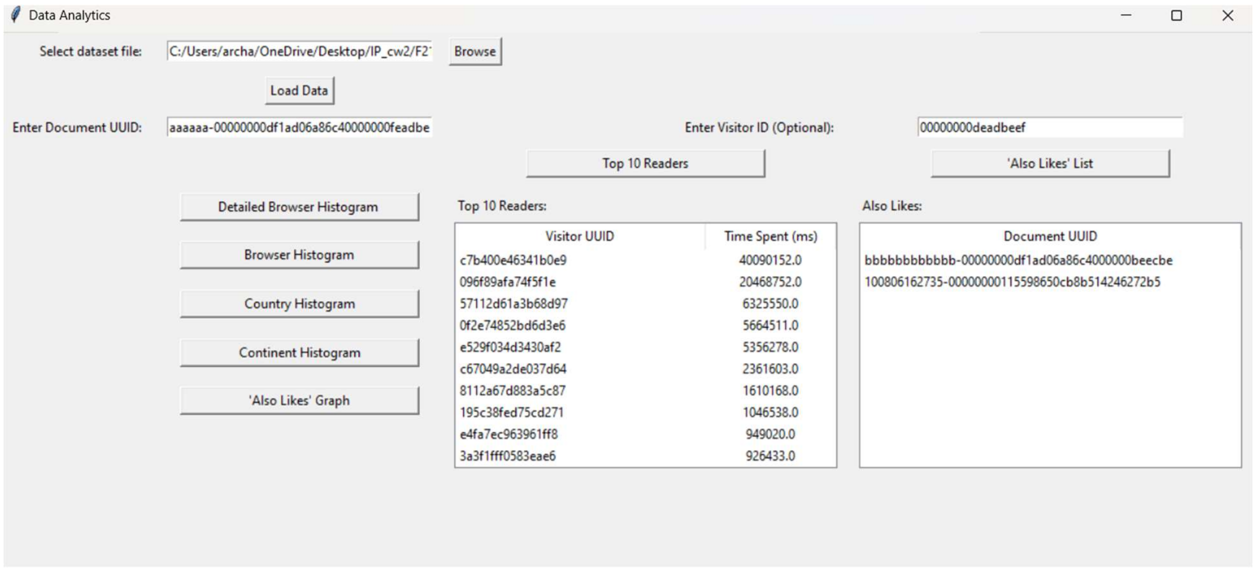
We have made several design decisions to enhance the application's usability and accessibility:

- Interactive GUI: The GUI offers a simple yet intuitive way for users to interact with the application. They can input their parameters, view histograms, and visualize "also likes" relationships without needing to touch the command line.
- Command-Line Interface (CLI): For users who prefer automated testing or working in headless environments, we implemented a CLI. This interface allows users to pass parameters like UUIDs and file paths to execute specific tasks directly.
- Error Handling: The application catches potential errors, such as invalid UUIDs or missing data files, and provides informative messages to the user.
- Efficient Data Handling: By leveraging Dask and Python generators, we ensured that the application could handle large datasets efficiently, making it suitable for practical use in real-world scenarios with millions of documents.
- Visualization and Graphs: By integrating matplotlib [4] and Graphviz, we provided clear, easily interpretable visualizations. The histograms and relationship graphs offer users a visual understanding of the document interactions, making the data analysis process much more accessible.

USER GUIDE

This application provides a GUI and command-line interface for analyzing and visualizing data, including tasks like generating histograms of document views, identifying top readers, and recommending "also likes" based on document interactions.

Using the GUI:



Select dataset file: C:/Users/archa/OneDrive/Desktop/IP_cw2/F2

Enter Document UUID: aaaaaa-00000000df1ad06a86c40000000feadbe

Enter Visitor ID (Optional): 00000000deadbeef

Top 10 Readers:

Visitor UUID	Time Spent (ms)
c7b400e46341b0e9	40090152.0
096f89afa74f5f1e	20468752.0
57112d61a3b68d97	6325550.0
0f2e74852bd6d3e6	5664511.0
e529f034d3430af2	5356278.0
c67049a2de037d64	2361603.0
8112a67d883a5c87	1610168.0
195c38fed75cd271	1046538.0
e4fa7ec963961ff8	949020.0
3a3f1fff0583eae6	926433.0

Also Likes:

Document UUID
bbbbbbbbbbbb-00000000df1ad06a86c4000000beecbe
100806162735-00000000115598650cb8b514246272b5

1.Load Data:

Open the application, and you'll be prompted to select a dataset file. You can choose a JSON or compressed .gz file. Click on the Browse button to select the file from your system.

After selecting the file, click Load Data to load it into the application. If the file is valid, a success message will appear; otherwise, an error message will prompt you to load a correct file.

2.Generate Histograms:

- Country Histogram: Enter the Document UUID in the respective input field and click on the Country Histogram button. This will display a histogram of the countries where the document was viewed.
- Continent Histogram: Similarly, enter Document UUID and click Continent Histogram. This generates a histogram of the continents based on the viewer's country.
- Browser Histograms: Click Browser Histogram or Detailed Browser Histogram to generate histograms showing the most popular browsers used by viewers, both simplified and detailed (including OS).

3.Top Readers Analysis:

Click Top 10 Readers to display a list of the top 10 readers based on the total reading time for each visitor in milliseconds.

4.Also Likes Functionality:

- Also Likes List: Enter a Document UUID and optionally a Visitor UUID, then click 'Also Likes' List. This will return the top 10 related documents liked by other readers of the given document.
- Also Likes Graph: After entering the Document UUID (and optionally a Visitor UUID), click 'Also Likes' Graph to generate a visual graph showing the relationships between documents and their readers.

Command-Line Usage:

The command-line interface allows testing the application's functionality via specific commands. Run the following command with the appropriate flags:

python cw2.py -t <task_id> -f <file_name> -d <doc_uuid> -v <visitor_uuid>

Example for task 6 :

```
python cw2.py -t 6 -f sample_400k_lines-1.json.gz -d 140310170010-0000000067dc80801f1df696ae52862b
```

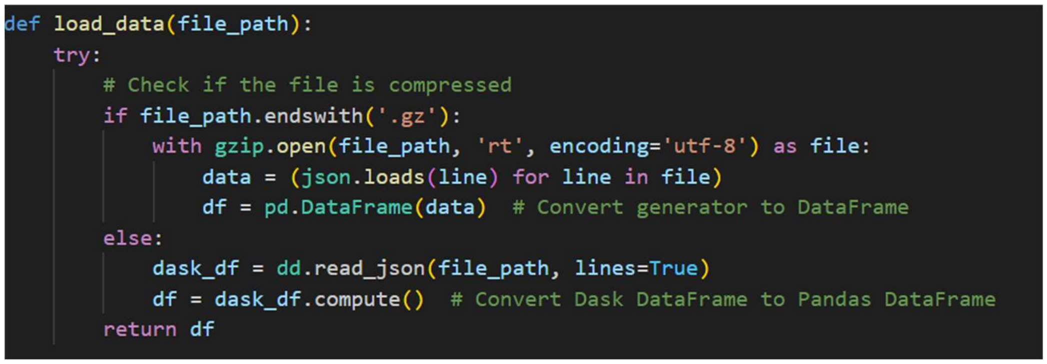
DEVELOPER GUIDE

The **primary goal** of the application is to provide a user-friendly and efficient tool for analyzing document tracking data, enabling insights into user interactions, document popularity, and reader behavior.

File Organization and Responsibilities

data_processing.py

- **Data Loading**:
    - Loads regular and gzipped JSON files into Pandas DataFrames.
    - Uses Dask for efficient handling of large datasets.
    - Includes error handling for file and decoding issues.



```
def load_data(file_path):
    try:
        # Check if the file is compressed
        if file_path.endswith('.gz'):
            with gzip.open(file_path, 'rt', encoding='utf-8') as file:
                data = (json.loads(line) for line in file)
                df = pd.DataFrame(data)  # Convert generator to DataFrame
        else:
            dask_df = dd.read_json(file_path, lines=True)
            df = dask_df.compute()  # Convert Dask DataFrame to Pandas DataFrame
        return df
```

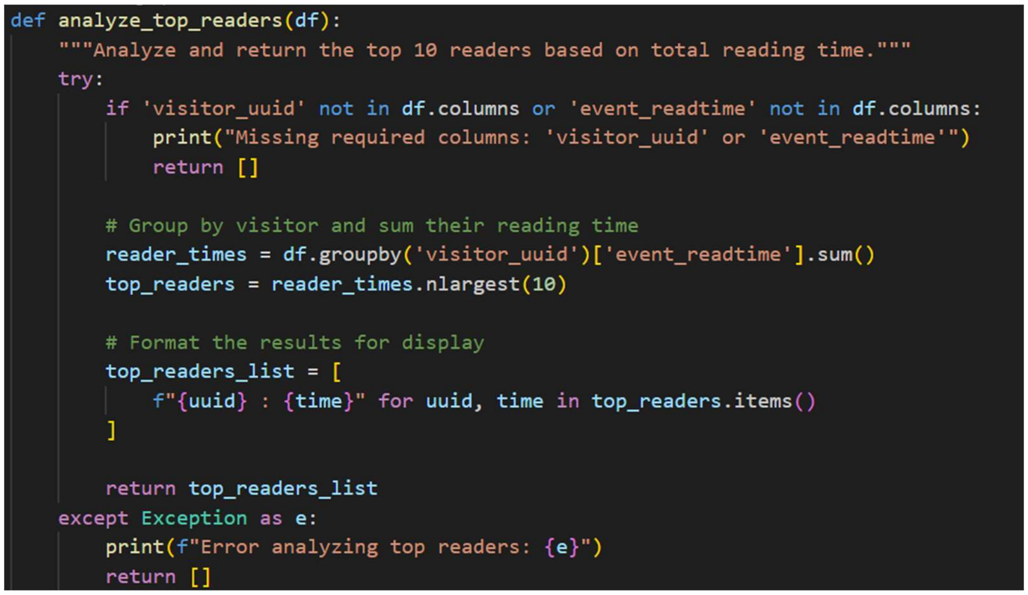
- **Browser Name Extraction**:
    - Identifies main browser names (e.g., Chrome, Firefox) using regex.

    - Returns "Unknown" if no match is found.
- **Detailed Browser Info Extraction**:
    - Extracts browser names with OS details from `useragent` strings.
    - Formats as <Browser Name> (<OS Details>).

display_histogram.py

- **Country Histogram**:
    - Filters data by subject_doc_id (document UUID) to count views per country.
    - Displays a bar chart showing the distribution of views by country.
- **Continent Histogram**:
    - Maps countries to continents using continent_mapping.
    - Filters data by subject_doc_id and displays a bar chart of views by continent.
- **Browser Histogram**:
    - Extracts main browser names using extract_browser_name.
    - Generates a bar chart showing the distribution of views by browser.
- **Detailed Browser Histogram**:
    - Extracts detailed browser names with OS info using extract_detailed_browser_name.
    - Displays a comprehensive bar chart of browser details, including OS.

top_readers.py:



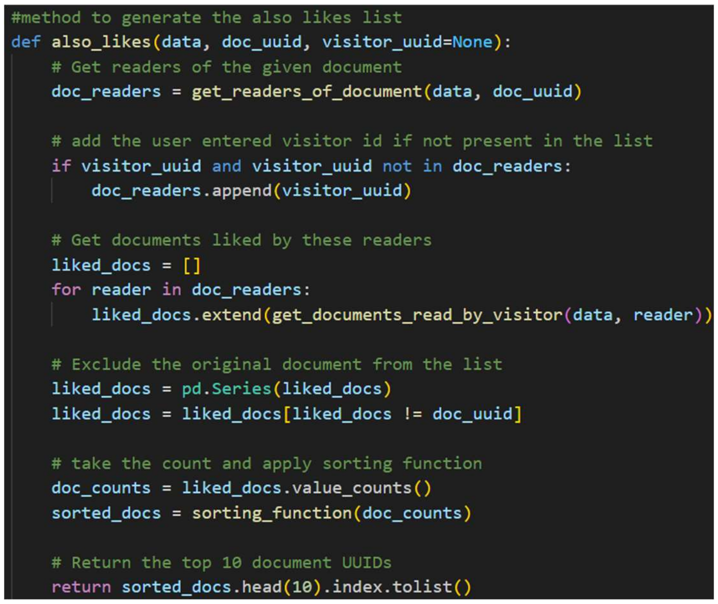
```python
def analyze_top_readers(df):
    """Analyze and return the top 10 readers based on total reading time."""
    try:
        if 'visitor_uuid' not in df.columns or 'event_readtime' not in df.columns:
            print("Missing required columns: 'visitor_uuid' or 'event_readtime'")
            return []

        # Group by visitor and sum their reading time
        reader_times = df.groupby('visitor_uuid')['event_readtime'].sum()
        top_readers = reader_times.nlargest(10)

        # Format the results for display
        top_readers_list = [
            f"{uuid} : {time}" for uuid, time in top_readers.items()
        ]

        return top_readers_list
    except Exception as e:
        print(f"Error analyzing top readers: {e}")
        return []
```

- The analyze_top_readers function identifies the top 10 readers by grouping data on 'visitor_uuid', summing their 'event_readtime', and formatting the results.

- It ensures proper validation, handles errors gracefully, and returns a list of the most engaged users.

also_likes.py

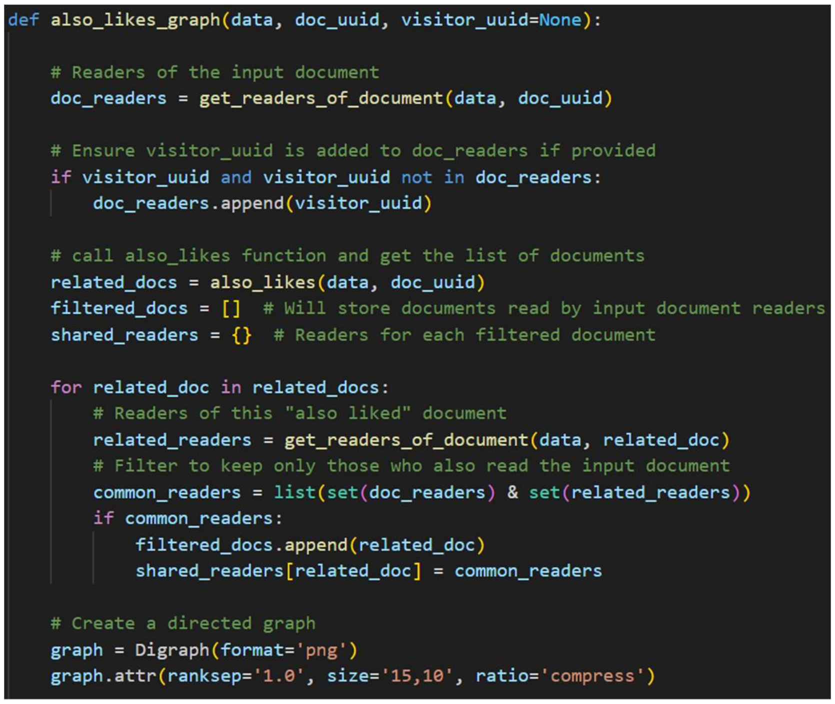
- **Reader Analysis**:
    - **get_readers_of_document**:
        - Fetches a list of unique visitor IDs who read the specified document.
    - **get_documents_read_by_visitor**:
        - Retrieves a list of documents read by a specific visitor.
- **Sorting**:
    - **sorting_function**:
        - Sorts document counts in descending order to prioritize popular recommendations.
- **"Also Likes" Feature**:
    - **also_likes**:
        - Finds documents commonly read by users who also read the specified document.
        - Uses a sorting function to prioritize the most liked documents.
        - Excludes the original document from recommendations and returns the top 10 related documents.



```python
#method to generate the also likes list
def also_likes(data, doc_uuid, visitor_uuid=None):
    # Get readers of the given document
    doc_readers = get_readers_of_document(data, doc_uuid)

    # add the user entered visitor id if not present in the list
    if visitor_uuid and visitor_uuid not in doc_readers:
        doc_readers.append(visitor_uuid)

    # Get documents liked by these readers
    liked_docs = []
    for reader in doc_readers:
        liked_docs.extend(get_documents_read_by_visitor(data, reader))

    # Exclude the original document from the list
    liked_docs = pd.Series(liked_docs)
    liked_docs = liked_docs[liked_docs != doc_uuid]

    # take the count and apply sorting function
    doc_counts = liked_docs.value_counts()
    sorted_docs = sorting_function(doc_counts)

    # Return the top 10 document UUIDs
    return sorted_docs.head(10).index.tolist()
```

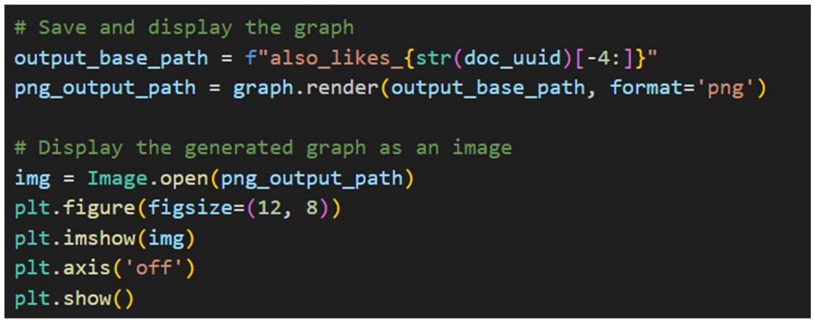
- **Graph Visualization**:
    - **also_likes_graph**:
        - Builds a directed graph using the **Graphviz** library.
        - Visualizes relationships between the input document, its readers, and the related documents.

- Highlights the input document and optionally a specified visitor.
- Ensures clear connections between documents and readers.



```python
def also_likes_graph(data, doc_uuid, visitor_uuid=None):

    # Readers of the input document
    doc_readers = get_readers_of_document(data, doc_uuid)

    # Ensure visitor_uuid is added to doc_readers if provided
    if visitor_uuid and visitor_uuid not in doc_readers:
        doc_readers.append(visitor_uuid)

    # call also_likes function and get the list of documents
    related_docs = also_likes(data, doc_uuid)
    filtered_docs = []  # Will store documents read by input document readers
    shared_readers = {}  # Readers for each filtered document

    for related_doc in related_docs:
        # Readers of this "also liked" document
        related_readers = get_readers_of_document(data, related_doc)
        # Filter to keep only those who also read the input document
        common_readers = list(set(doc_readers) & set(related_readers))
        if common_readers:
            filtered_docs.append(related_doc)
            shared_readers[related_doc] = common_readers

    # Create a directed graph
    graph = Digraph(format='png')
    graph.attr(ranksep='1.0', size='15,10', ratio='compress')
```

- **Graph Rendering and Display**:
  - Generates a PNG of the graph and displays it using Matplotlib and PIL.
  - Saves the graph for future reference and analysis.



```python
# Save and display the graph
output_base_path = f"also_likes_{str(doc_uuid)[-4:]}"
png_output_path = graph.render(output_base_path, format='png')

# Display the generated graph as an image
img = Image.open(png_output_path)
plt.figure(figsize=(12, 8))
plt.imshow(img)
plt.axis('off')
plt.show()
```

TESTING

2. Views by Country/Continent

Test Case 2.a: Generate a histogram of views by countries.

- Input: Load file and input the Document UUID.
- Expected & Observed Output: A histogram showing views by country for the specific document UUID. The X-axis displays countries, and the Y-axis shows the number of views.

- Edge Cases:
    - No input UUID (e.g., abcd): Error message displayed.

Test Case 2.b: Display a histogram of views by continent.

- Input: Load file and input the Document UUID.
- Expected & Observed Output: Matplotlib histogram showing viewer counts grouped by continent based on the document UUID.
- Edge Cases:
    - No input UUID (e.g., abcd): Error message displayed.

3. Views by Browser

Test Case 3.a: Display histogram of browser usage.

- Input: Input data with multiple browser identifiers.
- Expected & Observed Output: A histogram displaying the count of each browser's usage based on the visitor_useragent field. The histogram correctly visualizes the most frequently used browsers, with accurate counts.
- Edge Cases:
    - Missing or malformed useragent: Browsers classified as "Unknown."

Test Case 3.b: Display simplified browser names.

- Expected & Observed Output: A histogram showing the most popular browsers by name (e.g., "Chrome," "Firefox") after extracting the main browser name. The histogram groups all versions of the same browser under the main name, providing a simplified view.

4. Top 10 Reader Profiles

Test Case 4: Identify the top 10 readers.

- Expected Output: List of top 10 readers with total time spent.
- Observed Output: The function returns a list of the top 10 readers sorted by total reading time, formatted as UUID:time. The output correctly reflects the sum of reading times for each visitor.

5. “Also Likes” Functionality

Test Case 5: Generate "also likes" list.

- Input: Document UUID and visitor UUID optional.
- Expected & Observed Output: A list of top 10 document UUIDs that are "liked" by readers of the input document. The list is sorted based on the number of readers sharing both documents.
- Edge Cases:
    - Document with no readers: Returns empty list.
    - Multiple visitors with identical reading patterns: Sorted correctly.

6. "Also Likes" Graph

Test Case 6: Generate graph for "also likes."

- Expected & Observed Output: A graph showing the input document, related documents, and readers. Edges between nodes represent shared readers, with the input document and visitor (if provided) highlighted. The graph is displayed as an image and saved as pdf file.
- Edge Cases:
    - Visitor UUID omitted: Graph still generated.

7. GUI Functionality

Test Case 7: GUI operation for all tasks.

- Inputs: Various user inputs for document UUID, visitor UUID, and other parameters.
- Expected Output: Buttons trigger appropriate actions, and results display correctly.
- Observed Output: All functions operate as intended, with GUI interface and proper histograms outputs.

8. Command-line Usage

Tested command-line tasks include generating histograms (country, continent, browser, detailed browser), analysing top readers, and producing "also likes" lists and graphs.

REFLECTIONS

We found Python very useful for this project, especially with tools like pandas for analyzing data, matplotlib for creating graphs, and Dask for working with large datasets. The integration of graphviz helped us visualize complex relationships, such as the "also likes" feature, while tkinter made the GUI development smooth and accessible.

One limitation we faced was performance when dealing with very large datasets. While Dask helped handle this better, we plan to explore more optimization techniques like data partitioning and parallel processing to further improve efficiency. Another limitation was the GUI's simplicity. We aim to enhance its user experience and make it more interactive in future updates. Lastly, we plan to implement better error handling to improve the system's reliability.

Python was a great choice for this application. Its simplicity and flexibility made it easy for us to focus on solving the problem without worrying too much about syntax. The broad range of libraries available in Python (for data manipulation, visualization, and UI development) made it very suitable for this project. Python's ability to handle both the computational tasks and the front-end development made it an ideal fit for this application.

We can use this application to gain valuable insights from the data collected and apply it in today's world. By analyzing the data in various ways, such as identifying the most active readers, understanding the geographical distribution of views, and identifying common document preferences through the "also likes" feature, the application helps us discover patterns and trends.

FEEDBACK CW1

From CW1, we learned the importance of providing detailed testing and a comprehensive developer guide. The feedback highlighted the need for clearer and more concise testing explanations, ensuring it covers all edge cases. We also focused on expanding the developer guide, adding more details about the code and integrating advanced concepts for a better understanding. For CW2 in Python, we made sure to apply this feedback by improving the testing section, offering thorough explanations, and making the developer guide more detailed.

CONCLUSION

We are most proud of the way the application integrates data processing, visualization, and interactive features into one seamless tool. The ability to load and process large datasets efficiently, generate insightful visualizations (like histograms and "also likes" graphs), and offer a user-friendly interface for document-specific analysis demonstrates the power of combining various technologies. We're also pleased with how the application uses Dask for handling larger datasets and the use of Graphviz for generating intuitive visual graphs to represent user behavior patterns.

However, there are areas where we wish we could have done better. For example, although Dask helps with larger datasets, we could improve how the application handles even bigger ones. We also realize that the error handling and user feedback could be stronger to make the app more reliable. The user interface could be more polished, with more interactive features to make it easier for users to explore the data.

Looking back, we see that improving these aspects would make the app more powerful, user-friendly, and able to handle even more complex tasks in the future.

Reference

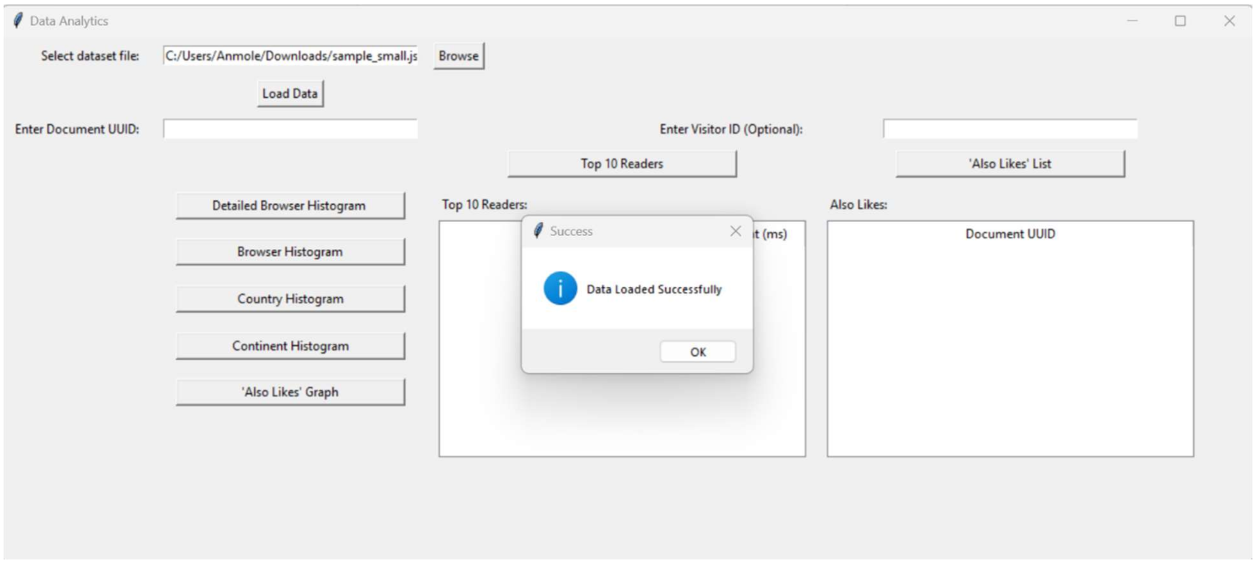
[1]. www.javatpoint.com. (2022). *Python Tkinter Tutorial - Javatpoint*. [online] Available at: https://www.javatpoint.com/python-tkinter

[2].GeeksforGeeks (2020). *Dask in Python*. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/introduction-to-dask-in-python/.

[3].GeeksforGeeks. (2016). *Generators in Python*. [online] Available at: https://www.geeksforgeeks.org/generators-in-python/.tkin

[4].W3Schools (n.d.). *Matplotlib Pyplot*. [online] www.w3schools.com. Available at: https://www.w3schools.com/python/matplotlib_pyplot.asp.

APPENDIX



Data Loaded Successfully

Figure1: GUI -Loading data successfully

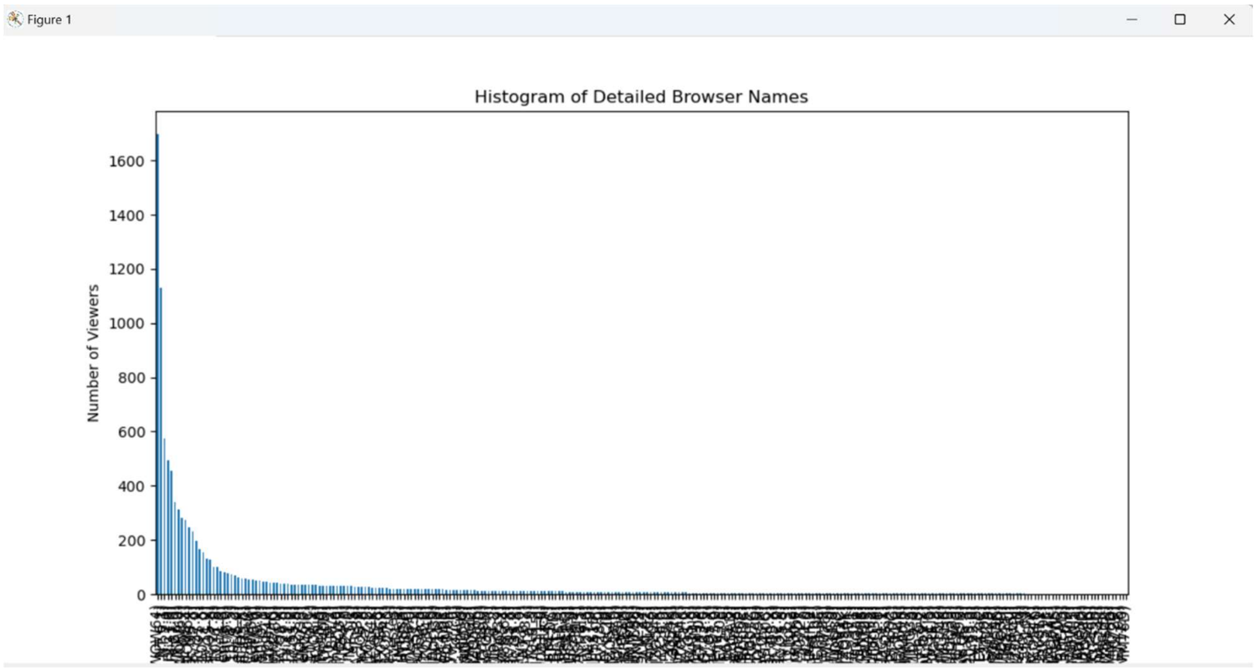


Figure2: Histogram of Detailed Browser Names

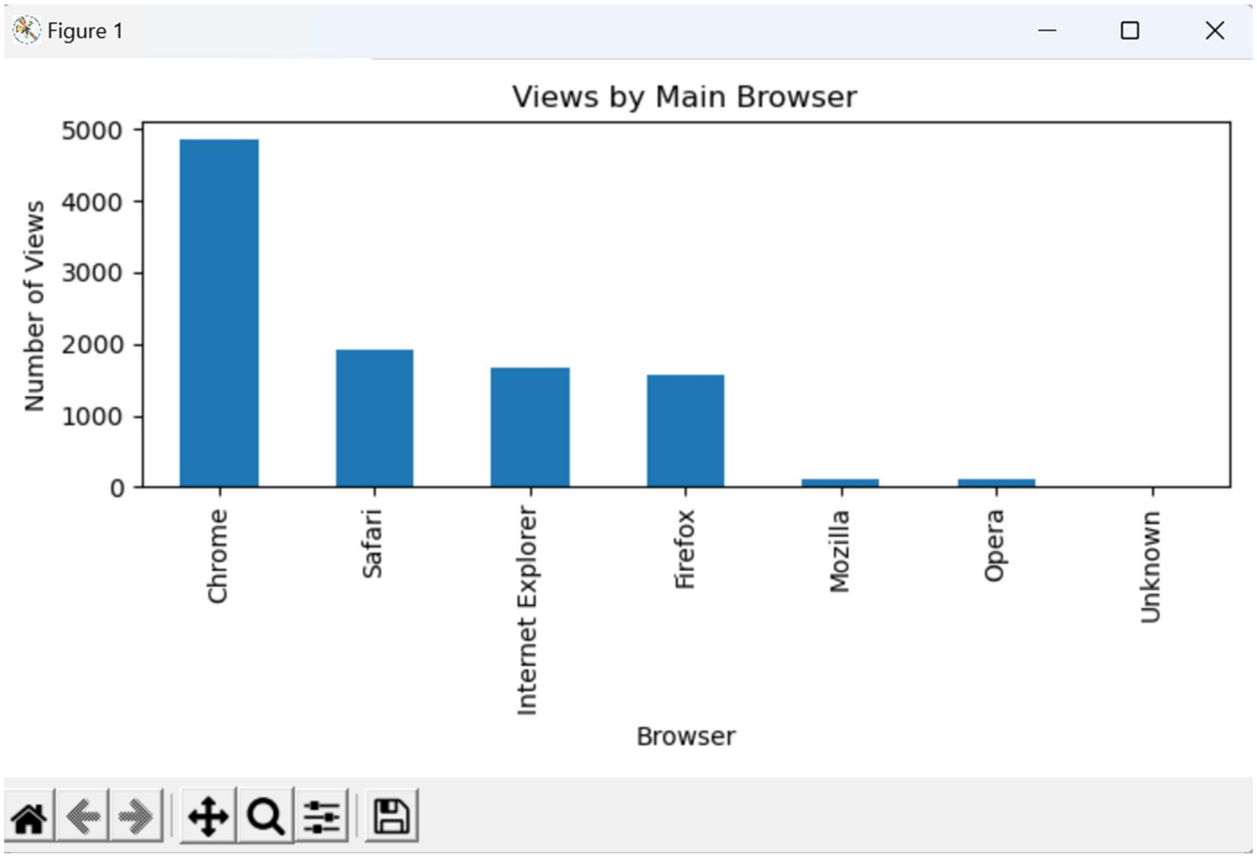
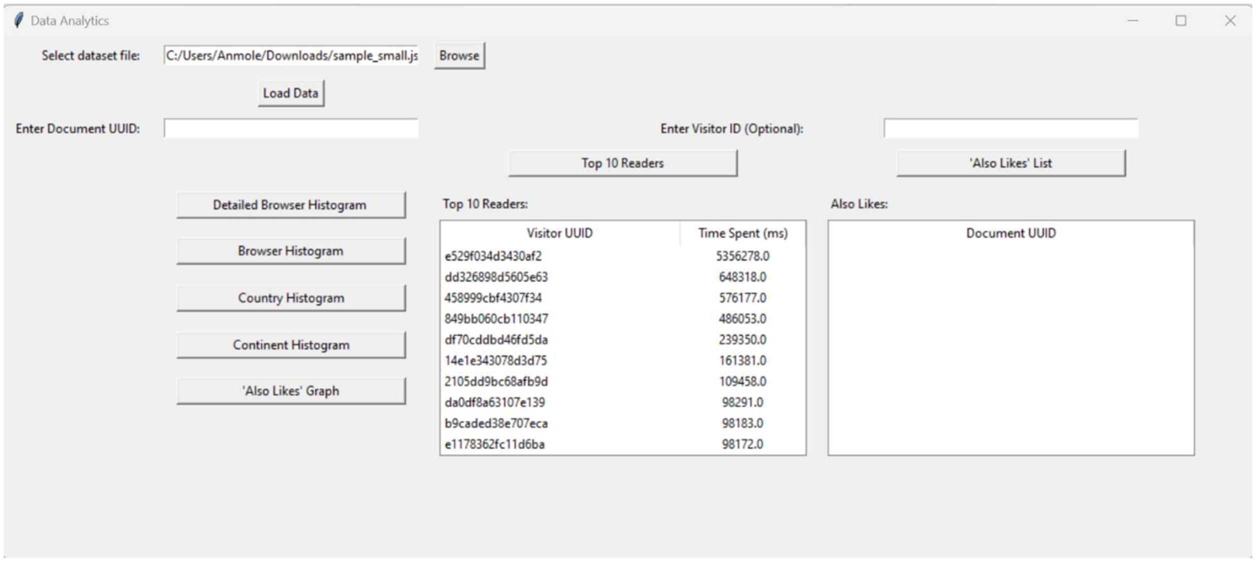


Figure3: histogram of Views by main browser



Top 10 Readers:

Visitor UUID	Time Spent (ms)
e529f034d3430af2	5356278.0
dd326898d5605e63	648318.0
458999cbf4307f34	576177.0
849bb060cb110347	486053.0
df70cddbd46fd5da	239350.0
14e1e343078d3d75	161381.0
2105dd9bc68afb9d	109458.0
da0df8a63107e139	98291.0
b9caded38e707eca	98183.0
e1178362fc11d6ba	98172.0

Figure4: Top 10 readers

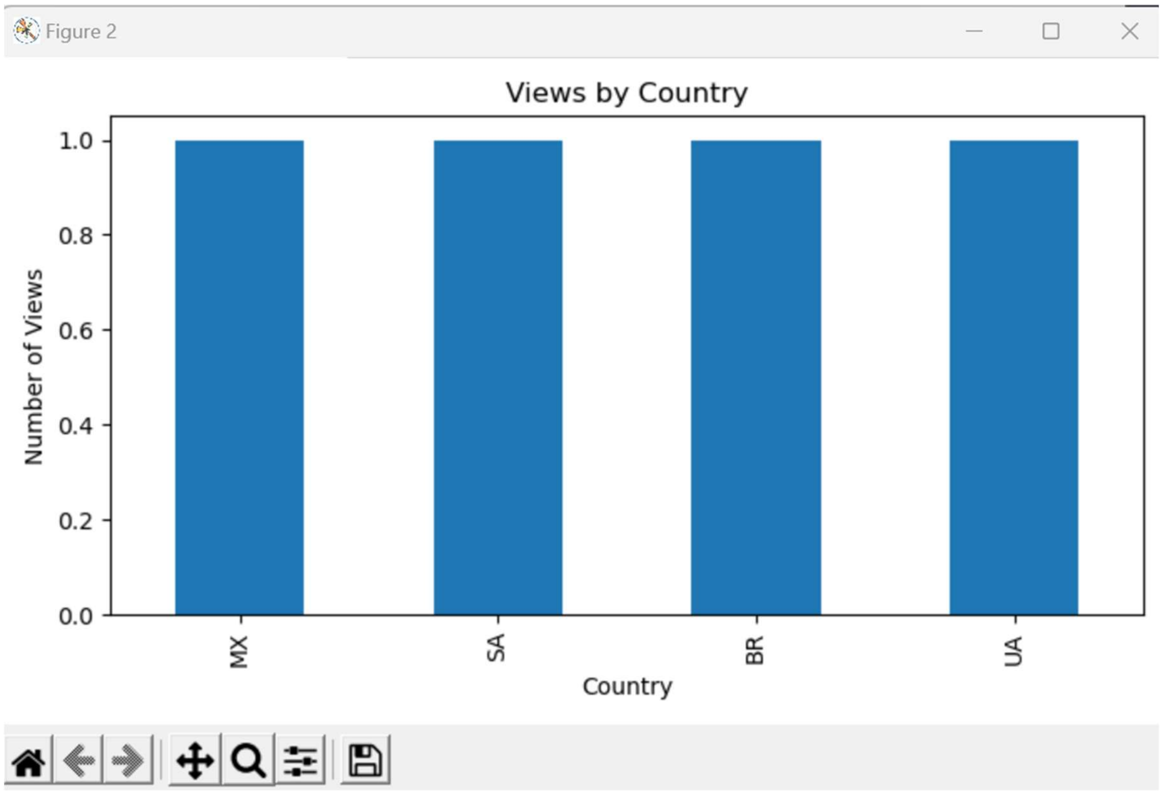
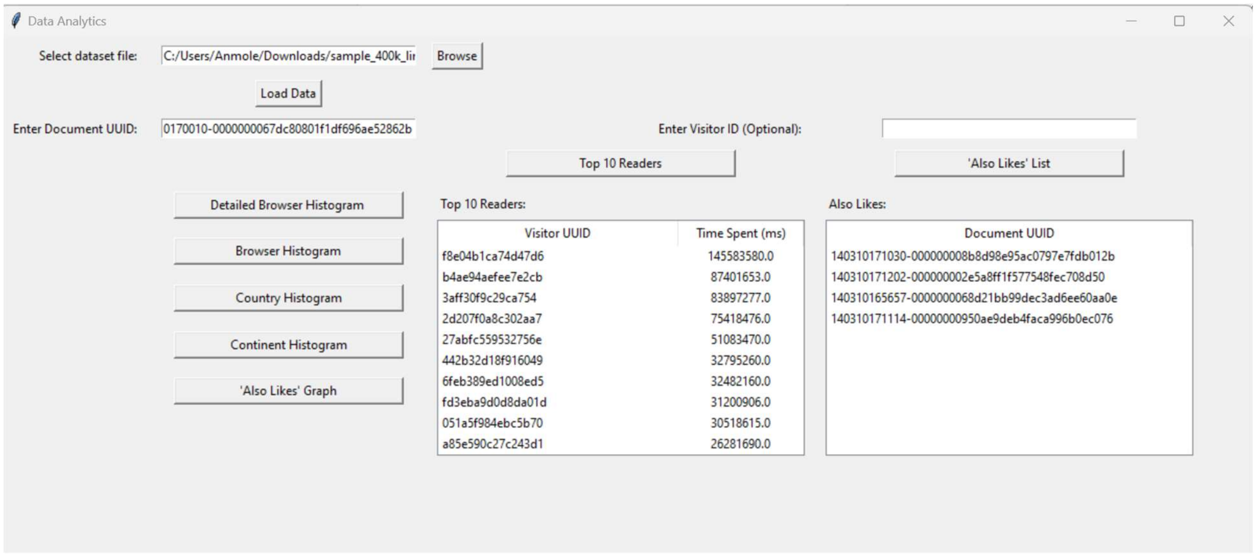


Figure5: histogram of Views by country



Select dataset file: C:/Users/Anmole/Downloads/sample_400k_lir

Enter Document UUID: 0170010-0000000067dc80801f1df696ae52862b

Enter Visitor ID (Optional):

Top 10 Readers:

Visitor UUID	Time Spent (ms)
f8e04b1ca74d47d6	145583580.0
b4ae94aefee7e2cb	87401653.0
3aff30f9c29ca754	83897277.0
2d207f0a8c302aa7	75418476.0
27abfc559532756e	51083470.0
442b32d18f916049	32795260.0
6feb389ed1008ed5	32482160.0
fd3eba9d0d8da01d	31200906.0
051a5f984ebc5b70	30518615.0
a85e590c27c243d1	26281690.0

Also Likes:

Document UUID
140310171030-000000008b8d98e95ac0797e7fdb012b
140310171202-000000002e5a8ff1f577548fec708d50
140310165657-0000000068d21bb99dec3ad6ee60aa0e
140310171114-0000000950ae9deb4faca996b0ec076

Figure6: Also likes list

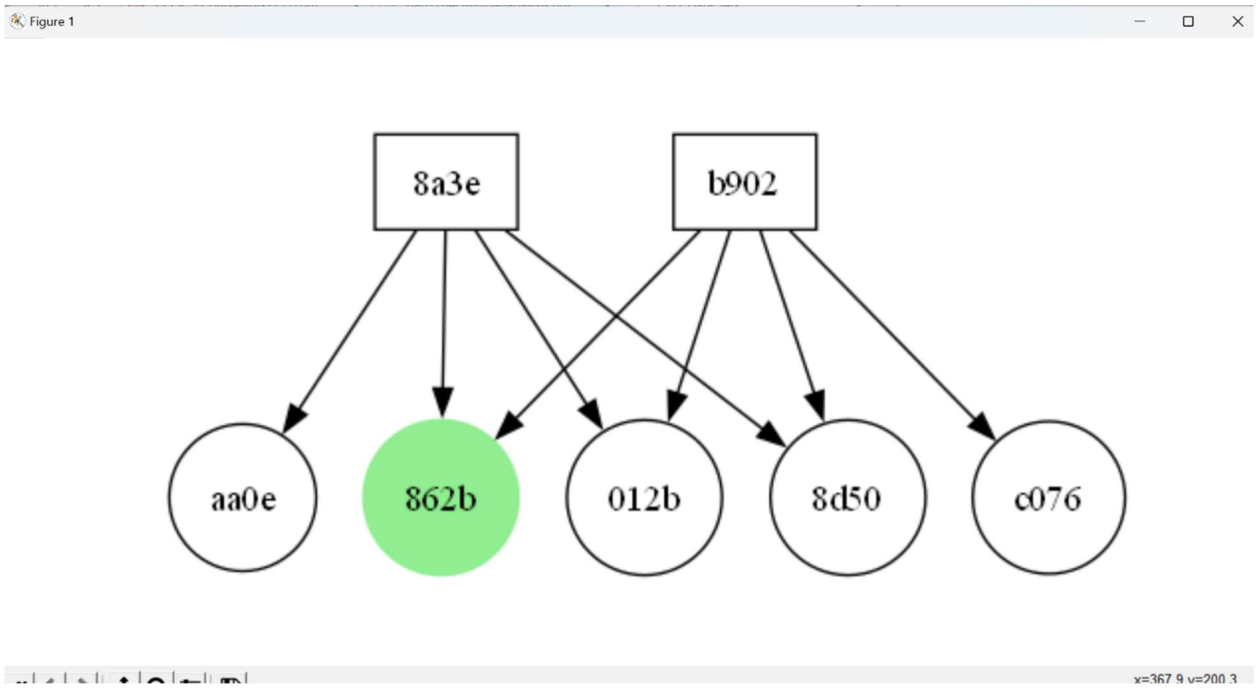


Figure7: Also likes graph