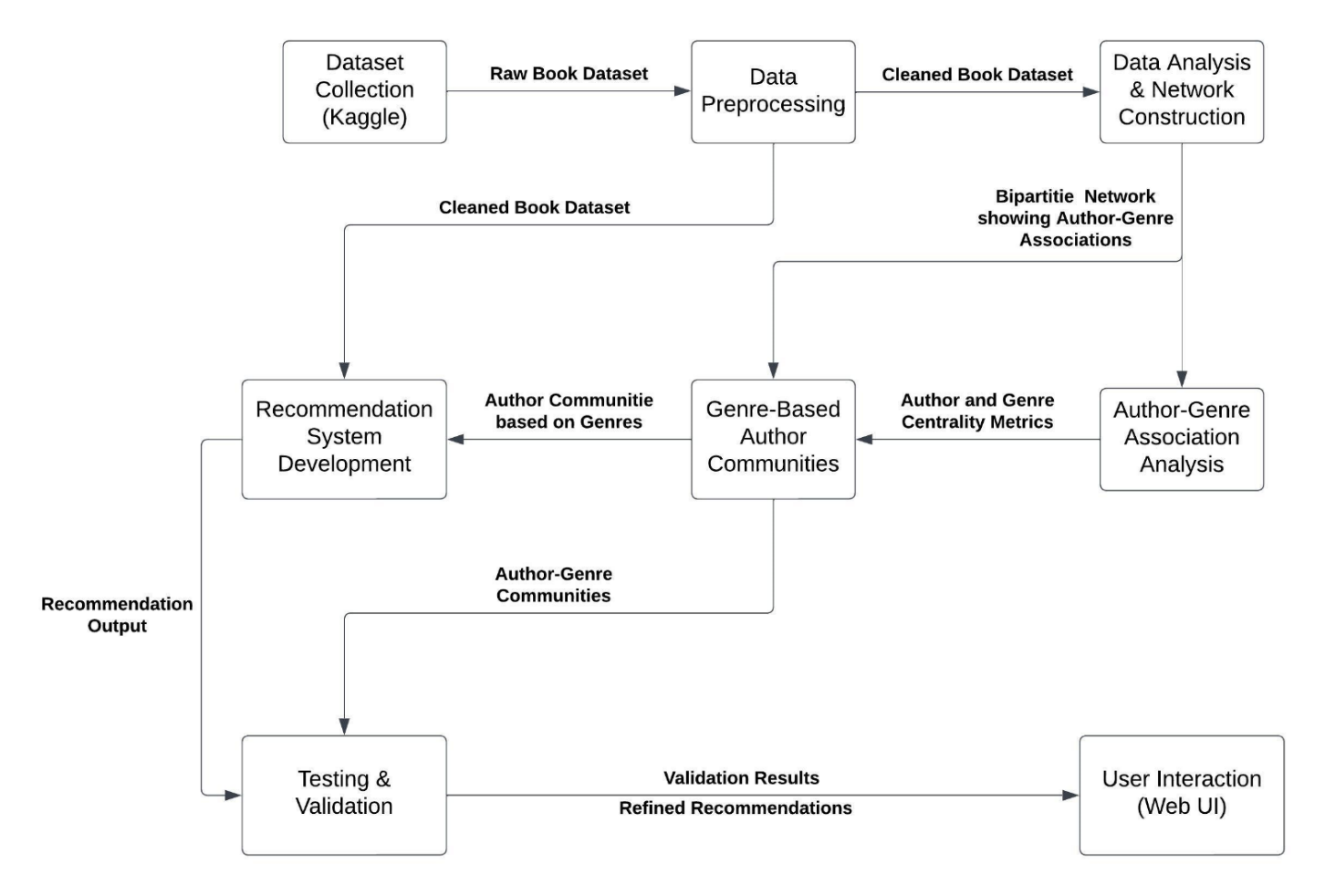
**Data Flow Diagram of the Genre-Centric Book Recommendation System**



**Implementation of Modules**

1. **Data Preprocessing Module**

This module cleans and preprocesses a dataset of book data, handling missing values, converting data types, filtering out non-English content, and processing genres before saving the processed data to a new CSV file.

1. Loading the Dataset:
   * The code begins by importing necessary libraries, such as pandas and numpy.
   * It loads a dataset from a CSV file ("My\_Book\_data.csv") into a Pandas DataFrame (df).
   * Initial exploratory data analysis is done using `head()`, `shape`, `size`, `info()`, and `columns` to understand the structure and content of the dataset.
2. Filtering Columns:
   * The code defines a list of columns to keep (`filter\_cols`) and removes the columns not in this list from the DataFrame.
   * Unimportant columns such as book\_description, book\_edition, book\_format, book\_isbn, image\_url are filtered out.
   * The resulting DataFrame (`df\_filtered`) is displayed, and its information is printed.
3. Handling Missing Values:
   * Rows with missing values are removed from `df\_filtered` using `dropna()`.
4. Processing 'book\_pages' Column:
   * The 'book\_pages' column, initially of object type, is converted to integers after removing the "pages" suffix from each element.
5. Handling Multiple Authors:
   * Some cells in the 'book\_authors' column have multiple authors separated by "|". The code defines a function (`removeMultipleAuthors`) to remove rows with multiple authors.
6. Language Filtering:
   * Two functions (`is\_english` and `removeOtherLangs`) are defined to filter out rows where 'book\_authors' or 'book\_title' are not in English.
   * Rows with non-English content are removed from the DataFrame.
7. Genres Processing:
   * The 'genres' column, initially containing pipe-separated strings, is converted to lists using the `changeDelimiter` function.
   * Duplicate genres within each row are removed, and the lists are sorted.
   * The processed lists are converted back to strings and assigned to the 'genres' column.
8. Saving Processed DataFrame:
   * The processed DataFrame is saved to a new CSV file ("Processed\_BookData.csv") using `to\_csv()`.
9. Final Output:
   * The code displays a subset of the processed DataFrame for examination (`df\_filtered[1:len(df\_filtered):2000]`).
10. **Network Construction Module**

This module focuses on creating and analyzing bipartite graphs for 1% and 10% samples of the processed book dataset. It calculates weights based on various metrics and visualizes the resulting graphs, providing insights into the relationships between authors and genres in the sampled data.

1. Loading the Processed Dataset:
   * The code begins by importing necessary libraries: pandas, networkx, matplotlib, random, and pickle.
   * It loads a previously processed dataset from a CSV file ("Processed\_BookData.csv") into a Pandas DataFrame (df).
2. Identifying Unique Genres:
   * The code extracts unique genres from the 'genres' column in the DataFrame, creating a set (`genreList`).
   * The total number of unique genres is printed.
3. Sampling 1% of the Dataset:
   * A 1% sample of the dataset (`df\_1\_sample`) is created for testing purposes using random sampling.
   * The genres column is transformed into a list of genres.
4. Creating a Bipartite Graph for 1% Sample:
   * A bipartite graph (`B\_1\_sample`) is created using NetworkX.
   * Nodes are added for authors and genres.
   * Edges are added with weights calculated based on specified metrics, including ratings, rating counts, review counts, and probabilities.
   * The resulting graph is saved using the `writeGraph` function.
5. Visualizing the 1% Sample Bipartite Graph:
   * The code visualizes the bipartite graph using matplotlib, with nodes representing authors and genres, and edges representing connections between them. Weights are displayed on the edges.
6. Sampling 10% of the Dataset:
   * A 10% sample of the dataset (`df\_10`) is created for further analysis using random sampling.
   * Similar to the 1% sample, the genres column is transformed into a list of genres.
7. Creating a Bipartite Graph for 10% Sample:
   * A new bipartite graph (`B\_10`) is created for the 10% sample.
   * Nodes, edges, and weights are added similar to the process for the 1% sample.
   * The resulting graph is saved using the `writeGraph` function.
8. Visualizing the 10% Sample Bipartite Graph:
   * The code visualizes the bipartite graph for the 10% sample using matplotlib, with nodes representing authors and genres, and edges representing connections between them. Weights are displayed on the edges.

**Author-Genre Edge Weight Computation**

Say there is an author A1 who has written books B111, B112, and B113 of genre G1.

The bipartite graph will therefore include an edge <A1,G1>.

In the dataset each book has a book rating, book rating count and book review count. In order to estimate the overall rating that people would give for the A1-G1 pair, I will compute the weighted average of the book ratings for B111, B112 and B113, where the weights will be the averages of the corresponding book rating counts and book review counts.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Book | Book rating | Book Rating Count | Book Review Count | Book weight |
| B111 | R111 | RC111 | ReC111 | BW111 = (RC111+ReC111)/2 |
| B112 | R112 | RC112 | ReC112 | BW112 = (RC112+ReC112)/2 |
| B113 | R113 | RC113 | ReC113 | BW113 = (RC113+ReC113)/2 |

Therefore, in general –

**where, nag is the number of books written by author *a* of genre *g*.**

The author A1 would have also written several other books B121, B131, and more, of other genres G2, G3, and so on.

Similarly, there will be several other books B211, B311, and more, of genre G1 which have been written by other authors A2, A3, and so on.

Hence, in order to compute the probability of seeing an A1G1 pair in the dataset, I will calculate a Joint Probability, which will be the product of 2 probabilities (assuming the 2 probabilities as being independent of each other) –

1. PAG(a,g): Probability of finding a book written by an author *a* which has the genre *g*.

Let the,

* number of books written by author A1 of genre G1 be C11, and
* total number of books written by author A1 be CA1.

Then,

Therefore, in general –

1. PGA(g,a): Probability of finding a book of genre G1 which is written by author A1.

Let the,

* number of books of genre G1 written by author A1 be C11, and
* total number of books of genre G1 be CG1.

Then,

Therefore, in general –

Thus, the Joint Probability of seeing a particular author-genre pair in the dataset is –

The weight of a particular author-genre pair will be directly proportional to the rating given to it and will also be directly proportional to the probability of its occurrence, so the overall weight will be a product of these two –

1. **Author Genre Association Module**

This module performs community detection and analysis on the bipartite graph of 1% sample of the dataset, representing relationships between authors and genres in the dataset. The resulting communities, link weights, and common genres are visualized and saved for further analysis.

* 1. Loading, visualizing and analyzing the 1% Sample Bipartite Graph:
     + The code reads a previously saved 1% sample bipartite graph from a pickle file ("1%\_Sample\_author\_genre\_bipartite.gpickle") using NetworkX.
     + The bipartite graph is visualized using matplotlib.
     + Nodes are categorized into genres and authors based on their bipartite attribute.
     + The unique genres and authors are printed along with their counts.
  2. Projecting Bipartite Graph onto Authors:
     + The bipartite graph is projected onto the set of authors to create a new graph (`sample\_author\_projection`).
     + Degree centrality is calculated for authors.
  3. Community Detection with Louvain Algorithm:
     + Louvain community detection algorithm is applied to identify communities within the projected graph.
     + Degree centrality and associated genres are printed for each author.
  4. Visualizing the Author Projection:
     + The projected graph is visualized using matplotlib, showing connections between authors.
  5. Weight Calculation for Author-Author Edges:
     + Weights are calculated for each author-author edge in the projected graph based on common genres and their weights in the original bipartite graph.
  6. Creating a Communities Network:
     + A new graph (`C`) is created to represent the communities, including edges with the computed weights.
     + The community network is visualized using matplotlib.
  7. Data Analysis for Communities:
     + Data is collected for each author-author edge in the community network, including community ID, link weight, and common genres.
     + The data is stored in a DataFrame (`communities\_df`) and printed.
  8. Community Filtering and Visualization:
     + A new community network (`C1`) is created by excluding inter-community edges.
     + The filtered community network is visualized, and the data is collected and stored in a new DataFrame (`communities\_df2`).
  9. Saving Community Data:
     + The final community data is saved to a CSV file ("Communities\_1%\_BookData.csv").

**Author-Author Edge Weight Computation**

Let there be authors A1 and A2 in the bipartite graph. A1 is connected to genres G1, G2 and G3, while A2 is connected to genres G1, G3, and G4. Then in the author-projected graph there will be an <A1,A2> edge because of genres G1 and G3.

Hence, the weight of this edge in the projected network is defined as such,

Therefore, in general –

**where,**

* + - **n is the number of common genres between authors x and y, and**
    - **W(x,Gk) is the weight of the edge between author x and genre Gk in the bipartite graph, similarly W(y,Gk) is the weight of the edge between author y and genre Gk in the bipartite graph.**

1. **Recommendation Module**

This module provides an interactive book recommendation system based on the user's input, leveraging community data from previously analyzed graph. It allows the user to interactively search for a book and receive recommendations for similar books. The recommendations are influenced by the community structure, suggesting books by authors within the same community as the selected book's author.

1. Loading the Dataset and Creating a Sample:
   * The code loads a processed book dataset from "Processed\_BookData.csv" using pandas.
   * A random 1% sample of the dataset (`df\_sample`) is created for testing.
2. Loading Communities Data:
   * The code loads a CSV file containing community data ("Communities\_1%\_BookData.csv") that was previously generated.
3. Interactive Book Search:
   * The user is presented with a list of books to choose from (output of `showBooks()` function).
   * The user is prompted to enter a book title and the number of book recommendations to get.
4. SearchBook Function:
   * The user's input (book title and number of recommendations) is taken.
   * The details of the selected book are displayed, including author, number of pages, rating, rating count, and genres.
   * The function `get\_all\_authors` retrieves other authors from the same community as the selected book's author.
   * The function `get\_n\_books` provides book recommendations based on the authors obtained from the community.
   * The recommendations are displayed, including the book title and author.
5. User Interaction:
   * The user can input the book title and the number of recommendations they want to receive.
6. Output:
   * The code outputs details about the selected book and a list of recommended books along with their authors.