CSC111 Course Project Report: Book Recommendation System

Jayden Chiola-Nakai, Maria Ma, Kaiwen Zheng, Shaqeel Hazmi Bin Radzifuddin

April 3, 2023

Introduction

- **Background Information** Goodreads is an American social cataloging website founded by Otis and Elizabeth Chandler in 2006 and launched in 2007, aiming to provide a guide for consumers to find books they want to read in the era of the digital age. It is the world's largest website for users to not only discover books but also socialize with other users by establishing a network amongst all registered readers through group activities [1]. In particular, users of Goodreads can browse books that, for example, are newly released or on the featured lists, add books to their personal bookshelves, rate and review books, and get recommendations from Goodreads based on the previous reviews given by the users. In addition, users can engage in discussion boards on various topics in groups. They also have the option of adding other users as "friends," getting access to books on their bookshelves, as well as sharing reviews, posts, and book recommendations with each other [2].
- **Motivation** While Goodreads is a convenient platform that has provided helpful guidance to numerous users to explore books in variety since its launch in 2007, it also has several disadvantages with respect to its recommendation system, which may affect user experience to a certain degree depending on the specific user's preference. In order to obtain suggestions for future readings from Goodreads, the user needs to provide ratings to at least 20 books first so that the system learns the user's preference; otherwise, the user must manually explore books by narrowing down the genres in which they are interested. For newcomers to this website or reading in general, this feature can create inconvenience since it is both difficult and time-consuming to narrow the search amongst hundreds and thousands of books even in one genre. Therefore, **our goal is to build a book recommendation system for readers that can automatically make recommendations based on the preferences data the user provides as well as the books they have already saved, including the ones the user has already read and the ones they want to read**. Specifically, we propose a recommendation system that recommends books to new users and those who have already saved interesting books in the system. For the latter, the program will recommend books that, in terms of genre, are similar to those that already appear on their bookshelves.

Description of Datasets

The datasets we use are heavily based on the ones provided on UCSD Book Graph [3], [4]. We extracted 46,196 books in the original datasets for simulating our program. Specifically, we will use the following datasets for our simulation purpose:

1. `books.json.gz`

   This file contains detailed meta-data about 46,916 books on Goodreads. The attributes included in this file are as follows:

   - `isbn`: the ISBN of this book
   - `text_reviews_count`: the number of text reviews for this book
   - `series`: the series this book belongs to
   - `country_code`: the country code of this book
   - `language_code`: the language code of this book
   - `popular_shelves`: the popular user shelves on which this book appears
   - `asin`: the ASIN number (Amazon Standard Identification Number) of this book

- `is_ebook`: whether this book is an E-book
- `average_rating`: the average rating of this book
- `kindle_asin`: the Kindle ASIN of this book
- `similar_books`: a list of IDs of books that users who like the current book also like
- `description`: the description of this book
- `format`: the format of this book
- `link`: the link of this book on Goodreads
- `authors`: the authors of this book
- `publisher`: the publisher of this book
- `num_pages: the number of pages this book has`
- `publication_day`: the day on which this book was published
- `isbn13`: the ISBN13 of this book
- `publication_month`: the month in which this book was published
- `edition_information`: the edition of this book
- `publication_year`: the year in which this book was published
- `url`: the URL of this book on Goodreads
- `image_url`: the URL of the cover image of this book
- `book_id`: the ID of this book
- `ratings_count`: the number of ratings given to this book
- `work_id`: the ID of the work (the abstract version of this book regardless of editions)
- `title`: the title of this book
- `title_without_series`: the title of this book without including its series

The attributes we will include in our system are {`country_code, language_code, is_ebook, average_rating, similar_books, description, link, authors, publisher, num_pages, publication_year, book_id, ratings_count, title`}. Every other attribute will be removed for the purpose of our system.

A sample row in this JSON file with the attributes mentioned above removed is as follows:


```
{
"country_code": "US",
"language_code": "",
"is_ebook": "true",
"average_rating": "4.04",
"similar_books": ["519546", "1295074", "21407416"],
"description": "This is the final tale in the bestselling author L.J. Smith's romantic
horrortrilogy. Now, Kaitlyn Fairchild and her friends must put the powerful crystalto
the test--to stop an experiment that has turned one of them into a psychicvampire.
Kaitlyn must choose at last--Rob or Gabriel.",
"link": "https://www.goodreads.com/book/show/12182387-the-passion",
"authors": [{"author_id": "50873", "role": ""},
{"author_id": "232533", "role": "Ubersetzer"}],
"publisher": "",
"num_pages": "",
"publication_year": "",
"book_id": "12182387",
"ratings_count": "4",
"title": "The Passion (Dark Visions, #3)"
}
```


2. authors.json.gz

   This file contains the authors in books.json.gz.

   The attributes included in this file are as follows:

- `average_rating`: the average rating of this author
- `author_id`: the ID of this author
- `text_reviews_count`: the number of reviews given by this author to other books
- `name`: the name of this author
- `ratings_count`: the ratings assigned by this author to other books

The attributes we are going to include in our system are the following: {`author_id, name`}. Every other attribute will be removed for the purpose of our system.

A sample row in this JSON file with the attributes mentioned above removed is as follows:

```
{
"author_id":  "604031",
"name":  "Ronald J. Fields"
}
```

Computational Overview

- **Data Structures and Data Types Used** The main data structures that serve as the driving force for our recommendation system are the decision tree and graph. Specifically, the recommendation system is separated into the following two subsystems based on the user data:
  - Decision-tree-based recommendation subsystem (`RecommendationSystem`): this subsystem is used to make new recommendations to the user. We model this subsystem with a decision tree, where each tree node on each available path down the decision tree corresponds to a possible choice or range of an attribute of a book that is used to pair with the preferences data the user inputs to narrow down potential candidates for recommendation and the leaf node of each path stores a unique book that possesses all book attributes on its path.
  - Graph-based recommendation subsystem (`SimilarBookSystem`): this subsystem is used to make relevant recommendations based on the books generated by the decision-tree-based recommendation subsystem. We model this subsystem using a graph, wherein every vertex represents a book, and every edge connecting two vertices represents that these two books are similar based on Goodreads data.

  In order to lay foundations for implementing the data structures above and to structurally handle the datasets we use, we introduce several customized data types using Python class definition. Specifically, we define
  - `Book` class in `book.py`, which includes a collection of book attributes that we will use in our recommendation system,
  - `Library` class in `library.py`, which represents a repository of books,
  - `RecommendationSystem` class in `recommendation_system.py`, which simulates the decision-tree-based recommendation system, and
  - `SimilarBookSystem` class in `recommendation_system.py`, which simulates the graph-based recommendation system.
- **Computations**
  - Data wrangling: The book dataset we used, `books.json.gz`, is very large and contains much more information than what we need. Therefore, we removed some data that we believe aren't useful for our recommendation system. In fact, using `pandas.DataFrame.drop()`, book attributes such as `'isbn'`, `'text_reviews_count'`, `'series'`, `'asin'`, `'kindle_asin'`, `'format'`, `'isbn13'`, `'publication_day'`, `'publication_month'`, `'edition_information'`, `'url'`, `'work_id'`, and `'image_url'` were all ignored in our analysis and manipulation of the dataset. Also, to avoid any Python errors, we removed all entries with null values using `pandas.DataFrame.replace()` and `pandas.DataFrame.dropna()`. In addition, since we did not modify the original dataset and we also want to save time when retrieving data from that large file, we decided to use `pandas.read_pickle()` and `pandas.to_pickle()` to save the polished version of the data from `books.json.gz` into a `.pkl` file the first time the user runs the program and reading this file instead of the original one for all future uses.

- Decision tree and graph building: For the initialization of the decision-tree-based recommendation system, we first characterize each given book with the following list of attributes: $l =$ `[num_pages, country, language, title, authors, publisher, publication_year, book_id]`. Then, for each book in `book.json.gz`, we insert l for each book into an empty decision tree so that $\forall i \in \{1, 2, ..., \texttt{len(l) - 1}\}$, `l[i]` is the child of `l[i - 1]`. If `l[i]` has already appeared as a subtree of `l[i - 1]`, then insert the remaining list entries to the subtree coresponding to the subtree with the root `l[i]`. After this process, every leaf of this decision tree stores a unique ID for the corresponding book in `book.json.gz`. For the initialization of the graph-based recommendation system, we link each book with its similar books (which are stored in `book.json.gz`).
- Algorithms for making recommendations: We deploy different strategies to make recommendations based on the type of recommendation subsystems involved.

  For the tree-based recommendation system, we first ask the user a sequence of pre-set questions to obtain the user's preference data. This sequence of questions consists of the following:

  * What range does the user want the number of pages of a book fall into?
  * What country of origin of the book does the user prefer (the user can select multiple countries here)?
  * What language of the book does the user prefer (the user can select multiple languages here)?
  * What is the title of the book? (Optional)
  * What is the author the user is looking for? (Optional)
  * What is the publisher the user is looking for? (Optional)
  * In what year was the book published? (Optional)

  Then, we gather the user's responses to each of these questions in the order they appear above. Ideally, if there exists a path in the decision tree on which every book attribute matches the corresponding user preferences, then the book attached to such a path will be returned for recommendation. However, if there is no perfect match, we then find the books that are most "relevant" to the user's preferences. To measure the degree of relevance, we define the *match score* between a given book and the user's preferences as the number of book attributes that match the corresponding user preferences. Since we set the first three book attributes to be mandatory for the user to select, we thus give them higher weight compared to the remaining criteria. In particular, for each of the first three book attributes, if it matches the corresponding user preference, we increment the total match score by 10, otherwise 0 is added. For the remaining book attributes, if the corresponding match is detected, we increment the total match score by 1, otherwise 0 is added. The higher the match score is, the more suited this particular book is for the user with respect to their preferences. This algorithm is illustrated in greater details in the docstring of the method `RecommendationSystem.recommend` in `recommendation_system.py`.

  For the graph-based recommendation system, we first get the collection of books that are read by the user and those saved for future reading. Then, for each book in this collection, we search for the similar books connected to that book in the graph. For each similar book we get, we recommend it to the user only when it makes appearance as a similar book to all books for at least a number of times, which is a pre-set positive integer called the *relevance factor (RF)*. Also, we recommend books in reversely sorted order with respect to their number of occurrences. This computational scheme is illustrated in detail in the docstring of the method `SimilarBookSystem.recommend` in `recommendation_system.py`.

- **User Interaction** This program will make use of a GUI (graphical user interface) to allow the user to interact with the book recommendation system. `PyQt6` will be the Python library used to implement the GUI. It will consist of two main parts: the first part, for first-time users, will recommend the user books based on the preferences they enter (done using the decision-tree-based subsystem); the second part, for existing users, will allow the user to browse those books as well as other similar books (retrieved using the graph-based subsystem), save and unsave them in the system, and provide them detailed book information.

- **Libraries Used** In our recommendation system, we will use several Python libraries that have yet to be covered in this course so far. In particular, we will use the following libraries:

  - `pandas`: `pandas` is a Python library that provides high-performance data analysis tools for Python programming language [5]. Since our recommendation system is heavily based on building up a repository of information about books and their interactions with the users using the datasets we obtained, improving the efficiency of data manipulations plays a central role in boosting the overall efficiency of our system. To obtain well-organized tabular data repositories, we will use the `pandas.core.frame.DataFrame`

data type to accomplish this goal. In particular, we will use `pandas.read_json` or `pandas.read_csv` by passing the file path of the relevant JSON/CSV file as a string into these methods to obtain a `pandas.core.frame.DataFrame` object that organizes the file into a tabular data sheet, wherein the attributes in the original dataset correspond to the columns of the resulting data sheet. We will use various methods such as `pandas.DataFrame.drop` (to remove the columns that correspond to the attributes to be removed) and `pandas.DataFrame.dropna` (to remove entries with null values).

- PyQt6: `PyQT6` is a set of Python bindings for Qt v6—which is a set of cross-platform C++ libraries—that allow the creation of a GUI (graphical user interface) [6]. It will be used to let the user interact with the book recommendation system, using widgets such as `QLineEdit` (for text entry), `QPushButton` (for button clicks), and `QListWidget` (to display list items) as well as functions such as connect (to add widget functionality), `setFont`/`setFixedSize`/`setStyleSheet` (to add graphical design), and `addWidget`/`setLayout` (to organize the arrangement of the widgets). `PyQt6` is useful because it makes the program not only limited to the CLI (command line interface) and increases UX (user experience).
- `json`, `gzip`, `numpy`: these Python libraries will be used to extract (in the `.json.gz` format) and analyze (e.g. using `numpy.nan` which represents an invalid entry) data.

Instruction for Datasets Using and Program Running

- **Obtaining Datasets and Understanding Program Files**

(a) unzipped folder containing the datasets

(b) complete program files

Figure 1

  - Download the zip file named **"data"** from UTSend. This is where all of the datasets we use are kept.
  - Download all the files with the extension `.py` from Markus and put the *unzipped* version of **"data"** at the same level as other `.py` files in the *same* folder.
  - Open **"data"**. You should see that there are two files with the extension `.json.gz`: `books.json.gz` and `authors.json.gz` (Figure 1a). You do *not* need to decompress the files above as our system directly handles the zip files.
  - In the same level as **"data"** folder, there are seven files with the file extension `.py`, which are the main functioning program files for our recommendation system as well as the GUI. There is also a folder called **"images"**, which contains the images we use when writing the project report. To run latex code, you need to first create a folder in which **"images"** should be at the same level as other files (Figure 1b).

- **Installing Python Libraries** Please refer to `requirements.txt` for further details.

- **Running the Program**

  - Open the file `main.py` and activate the `main` block at the bottom of this file. One should see an interactive message asking whether the user wants the system to recommend new books (Figure 2). If **"Yes"** is selected, then the user will be prompted into the next few pages where the user's preference data is collected by a sequence of questions. If **"No"** is selected, then the system will automatically return a collection of books based on the books saved by the user (none if no books were saved).

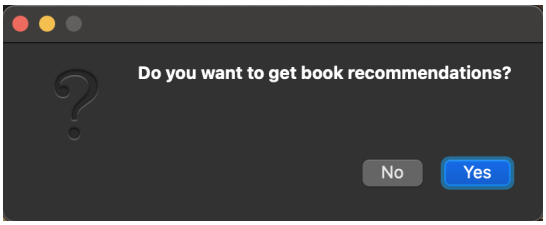

Figure 2: message box

– To get newly recommended books:
  * Press **"Yes"** in the message box in Figure 2 to advance into the front page of the recommendation system with the title **"Young Adult Book Recommendation System"** (Figure 3) that will asks the user a series of questions to customize the recommended books for the user.

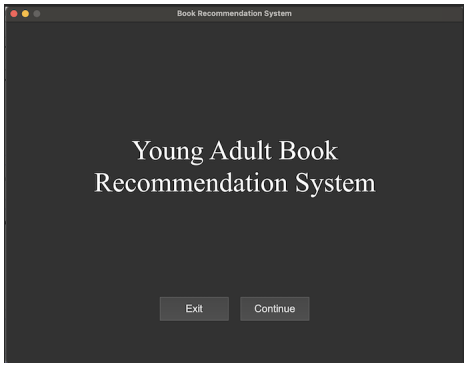

Figure 3: front page of the recommendation system

  * Click on **"Continue"** to proceed to the second page of the interface. There are two sliding bars with the names **"Minumum Number of Pages"** and **"Maximum Number of Pages"**, respectively (Figure 4), which ask the user to select the minimum and maximum number of pages the user is looking for. In order to proceed to the next page, the user is required to make corresponding choices so that $\textit{Maximum Number of Pages} - \textit{Minimum Number of Pages} > 5$. Otherwise, the choices will be deemed too restricted, as indicated in the message box in Figure 5.

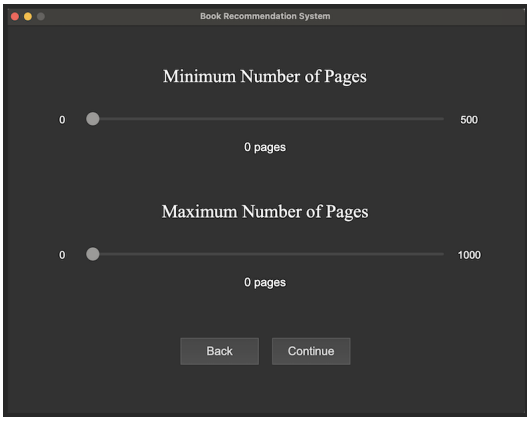

Figure 4: page where user inputs min/max number of pages

  * Click on **"Contitnue"** to proceed to the next page where one is asked to select the **country(ies) of origin** and the **language(s)** of the book the user is looking for (Figure 6). At least one selection must be made in the respective question in order to proceed to the next page in the interface, otherwise, the system will display a message to the user asking them to choose at least one option for each question (Figure 7).

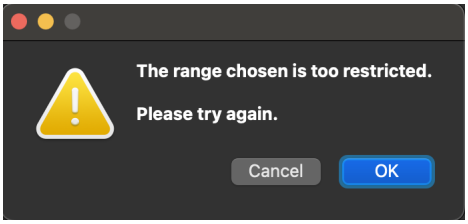

Figure 5: message user will get if inputs are too restricted

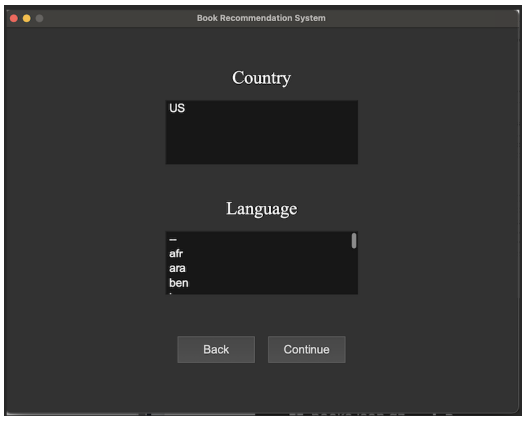

Figure 6: page where user is asked to input country(ies) and language(s) of the book

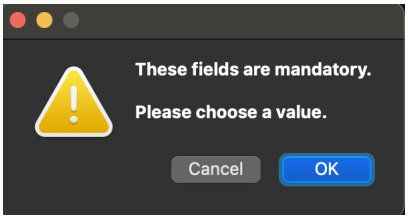

Figure 7: error message if user doesn't select any option on this page

* Click on **"Continue"** to reach the final question page in this interface where five optional questions **(the title of the book, the author of the book, the publisher of the book, the year in which the book the user is looking for was published, and whether the user accepts books with E-book version)** will be presented. The user needs not to feel obliged to complete all questions as all of them are optional. Questions that are left blank are not taken into consideration for recommendation (Figure 8).

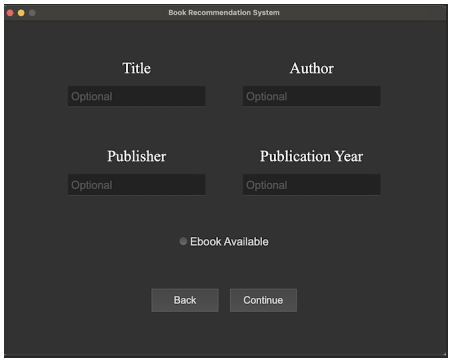

Figure 8: page that contains optional questions

* Press **"Continue"** to reach the page where recommended books are displayed (Figure 9). The left

side of this page contains books that match the user's preference the most and books similar to the recommended ones. The right side of the page displays information about a particular book. To see the book description, the user needs to click on the book they want and then press **"Search"** to see the detailed book description. To save a book into the system, the user needs to click on a book, press **"Search"**, and press **"Save"**. To un-save a book, the user needs to press on **"Search"** after selecting the book they want to remove from the system and press **"Unsave"**. Depending on the button the user selects, the system will display a corresponding message indicating that the book the user chooses has been saved/un-saved successfuly (Figures 10 and 11).

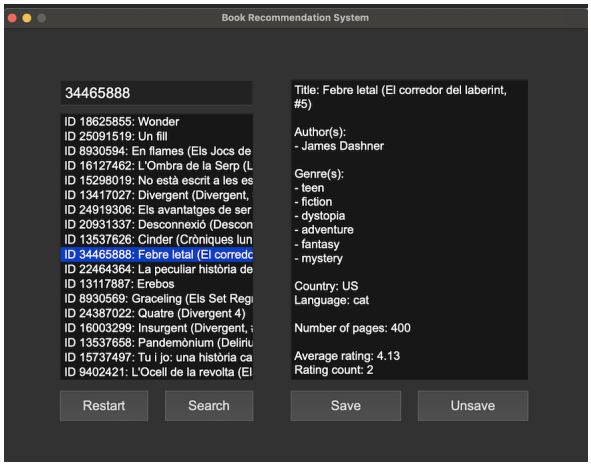

Figure 9: page of recommended books

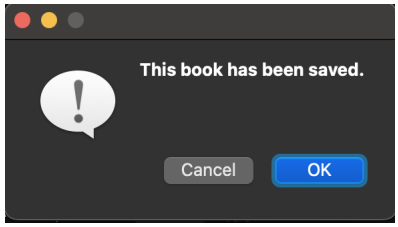

Figure 10: message indicating the book has been saved

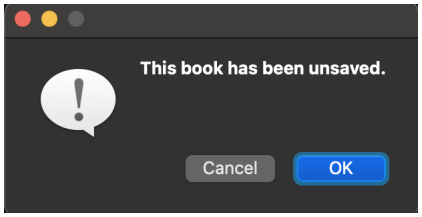

Figure 11: message indicating the book has been un-saved

– To get books similar to the ones saved into our system:
  * Press **"No"** .
  * The user will see a collection of books being displayed onto the screen if they have previously saved books.
  * The user can follow the instructions above to display the detailed book information, save a book into the system, and remove a book from the system.

Discussion

Overall, based on the preference data provided by the user as well as the books saved by the user, our system successfully generate recommended books that cater to the specific needs of a user, whether it is for searching new books or for finding similar books based on the ones the user saves. When providing more detailed responses to the system, we observed that the system successfully narrowed the range of recommendation down to a number of specific books; with relatively fewer number of responses, we noticed that although there were a large number of books that matched the preference data we fed into the system, the system randomly selected 60 books that best matched our preference data. The system displayed a collection of books to the user in reversely sorted order with respect to the average rating of a book as well as the number of ratings this books received, which maximized the user's opportunity to explore as many high-quality books as possible. We also successfully implemented the feature that the system automatically recommended similar books to the user based on the books already saved by them with accuracy. Also, we felt accomplished in that our system incorporated a user-friendly interactive interface, which helped the user navigate through the system without much difficulty. Therefore, our general for this project has been achieved. Despite all the merits above, there are still limitations to our recommendation upon which further improvements are to made, which are elaborated as follows.

- **Deficiency in Datasets** The dataset of books we generated is heavily based on the one in which the targeted audience is mostly young adult, which keeps our system from generating books with a wider range of genres that cater to the taste of audience in general.
- **Absence of Genres in Recommendation Algorithms** Since the datasets we found for books and book genres have no overlap at all, we could not take the genre of a book into consideration when generating recommendations. Although the original dataset for books included an attribute called `popular_shelves`, which reflected the genre of a book to some extent, we noticed the categorizations in `popular_shelves` are inaccurate for the purpose of recommendation. For example, some names included in `popular_shelves` did not reflect the genre to which the book belonged to; rather, it served as a way to categorize books based on the user's criteria (e.g. whether or not the book had been read). With the absence of the genre of books, our recommendation algorithms may fail to further optimize the recommended results.

In response to the potential shortcomings above, we anticipate the following improvements:

- **Cross-Referencing Datasets in Greater Detail** Despite the inaccuracy of the attribute `popular_shelves` mentioned above, we noticed that there were some categories in `popular_shelves` that reflected the content of a book. Therefore, we developed, but need to perfect, an algorithm to determine the genre of a book by incorporating keyword matching algorithms on shelf names in `popular_shelves` and the genres obtained with web scraping.
- **Including Books with Diversity** The complete dataset of books contains approximately 2.36M books. In order to include more diverse books, we may pre-process the complete book dataset so that there are approximately equal number of books for each category, upon which the recommendation system in run.

References

[1] "About goodreads," *Goodreads*. [Online]. Available: https://www.goodreads.com/about/us. [Accessed: 02-Apr-2023].

[2] "Goodreads," *Wikipedia*, 10-Mar-2023. [Online]. Available: https://en.wikipedia.org/wiki/Goodreads. [Accessed: 02-Apr-2023].

[3] M. Wan and J. McAuley, "Item recommendation on monotonic behavior chains," *Proceedings of the 12th ACM Conference on Recommender Systems*, 2018.

[4] M. Wan, R. Misra, N. Nakashole, and J. McAuley, "Fine-grained spoiler detection from large-scale review corpora," *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

[5] "Pandas documentation," *pandas documentation - pandas 1.5.3 documentation*. [Online]. Available: https://pandas.pydata.org/docs/. [Accessed: 02-Apr-2023].

[6] "PYQT6," PyPI. [Online]. Available: https://pypi.org/project/PyQt6/. [Accessed: 02-Apr-2023].

[7] "The future is written with Qt," Qt documentation. [Online]. Available: https://doc.qt.io/. [Accessed: 02-Apr-2023].

[8] M. Fitzpatrick, "PyQt6 tutorial 2023, create python guis with Qt," Python GUIs, 16-Mar-2023. [Online]. Available: https://www.pythonguis.com/pyqt6-tutorial/. [Accessed: 02-Apr-2023].