universidade de aveiro
theoria poiesis praxis

deti universidade de aveiro
departamento de electrónica,
telecomunicações e informática

Behind the scenes of the Recommendation system of the Goodreads platform

- Algorithm Development and Platform Impact

Diana Elisabete Siso Oliveira (98607)
Professional and Social Aspects of Computer Science
Informatics Engineering

19/07/2022

Index

1. Purpose of this document

This document was developed within the scope of the Professional and Social Aspects of Computer Science course unit, part of the Informatics Engineering course at the University of Aveiro, as a third assignment, whose focus is a technical analysis on a topic that the author considers relevant in relation to the course.

Summary / Abstract

Currently, the Goodreaders platform is the largest book recommendation site.

The Goodreads is a platform that allows a registered user to better manage the books they has read/is reading/will read in the "My Books" section, search for a particular book, author, references, genres, among other things, in the platform's database via the "Search" section and find new works that may interest you via the "**Recommendation**" section.

For this assignment, I decided to go deeper into the "**Recommendation**" section, analyzing the platform's recommendation algorithm and simulating a possible similar algorithm.

In addition, I will also address the impact that the platform currently causes on users.

2. Framework

The Goodreads platform was created in 2006 and was founded by engineer Otis Chandler and co-founder by journalist Elizabeth Khuri Chandler. The idea came about because Otis Chandler wanted a space where he could write reviews about the books he read and share lists of books with other people. On November 29, 2021 the Goodreads platform had 125 million members, 2.5 billion books and 80 million reviews.

The emergence of the Goodreads platform solved a problem that publishers dubbed the "discoverability problem"; this problem consisted in the difficulty of people, in the digital age, to find books that they might want to read.

The "Recommendation" section was added to the platform on September 15, 2011, a section that consists of recommending books through a machine learning algorithm that analyzes possible books that may correspond to the user's personal tastes, based on the books that the user has liked in the past and in books that people with similar tastes also liked. This mechanism only works after the user has classified 20 books on a scale from 0 to 5 stars.

An important aspect that distinguishes this recommendation engine from other recommendation engines, such as Amazon's, is that this engine does not include books that the user may have interacted with but not for personal use, such as books that the user has purchased as a gift.

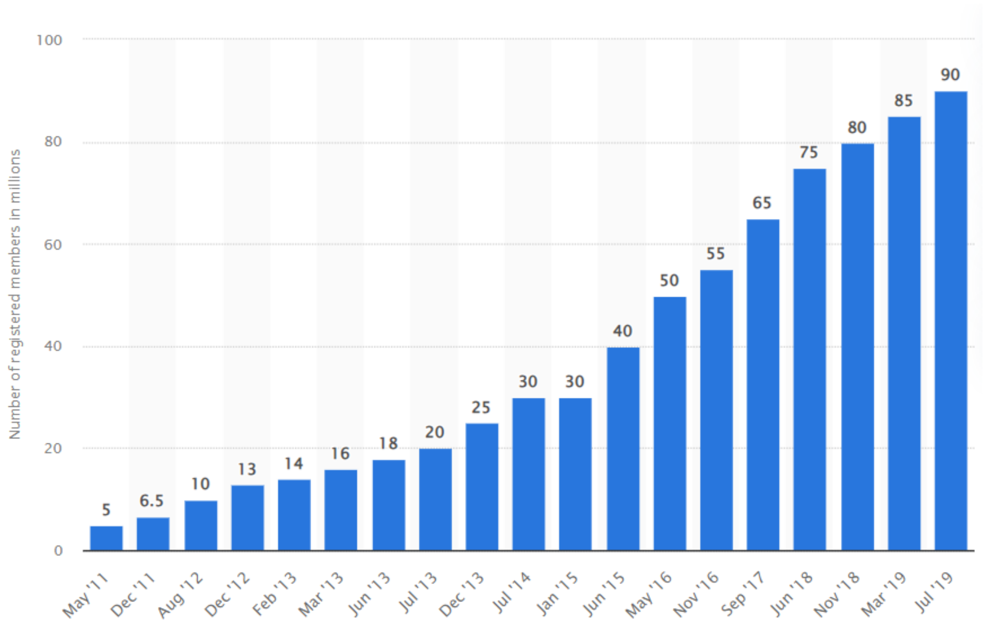
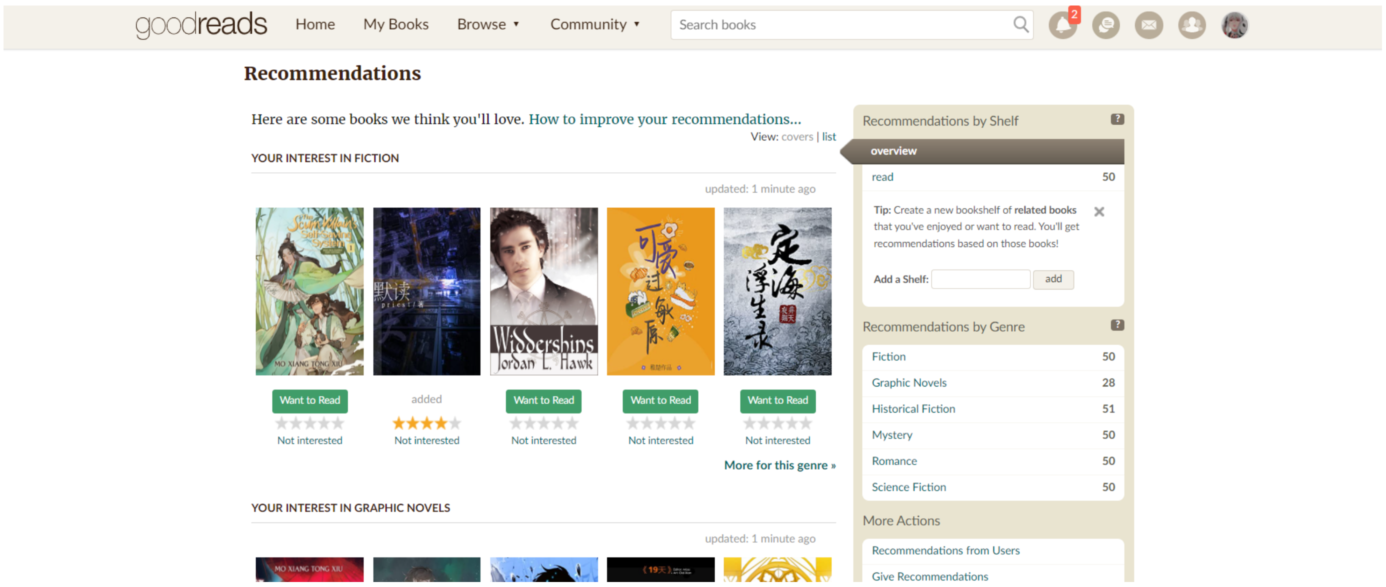


Figure 01. - Number of registered members (in millions) since May, 2011 to Jul, 2019

3. Goodreads

Recommendation Page

The image belows shows an overview of the recommendation page.


goodreads Home My Books Browse ▾ Community ▾ Search books

Recommendations

Here are some books we think you'll love. How to improve your recommendations...

View: covers | list

YOUR INTEREST IN FICTION

updated: 1 minute ago

Want to Read / Not interested — added / Not interested — Want to Read / Not interested — Want to Read / Not interested — Want to Read / Not interested

More for this genre »

YOUR INTEREST IN GRAPHIC NOVELS

updated: 1 minute ago

Recommendations by Shelf

overview

read	50

Tip: Create a new bookshelf of related books that you've enjoyed or want to read. You'll get recommendations based on those books!

Add a Shelf: add

Recommendations by Genre

Fiction	50
Graphic Novels	28
Historical Fiction	51
Mystery	50
Romance	50
Science Fiction	50

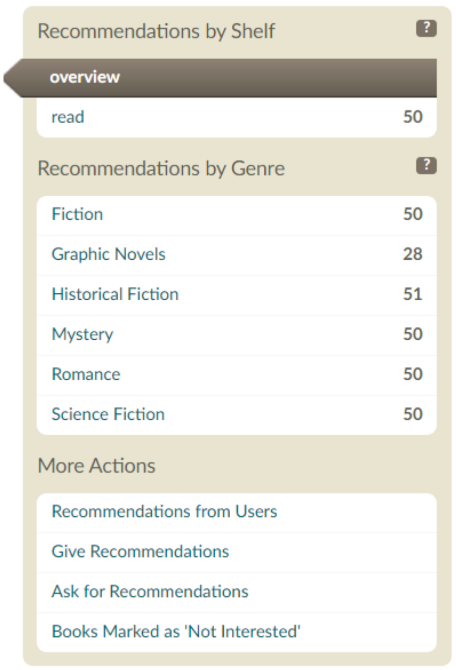
More Actions

Recommendations from Users
Give Recommendations

Figure 02. - Recommendations Page: Overview

The user may have **general** recommendations or recommendations **based on some shelf** that has been created. Shelves are groups of books that the user creates, these books having something in common that is used as a criterion for the shelf - for example, it is possible for the user to create a shelf where they only place the books of a certain author and another where they place only books that have a sad ending. In the case of my account on the platform, I created a shelf where I put all the books that are currently read; another shelf where I have the books I want to read and another where I have the books I'm in the process of reading. Note that the recommendation engine is based on **books already read** and therefore does not consider the other two shelves I created in addition to the "read" shelf.

The user can even find recommendations **filtered**, for example, by genre. The platform collects statistics on the genres that most interest the user and offers the option to recommend literary works related to those genres.


Recommendations by Shelf

overview

read	50

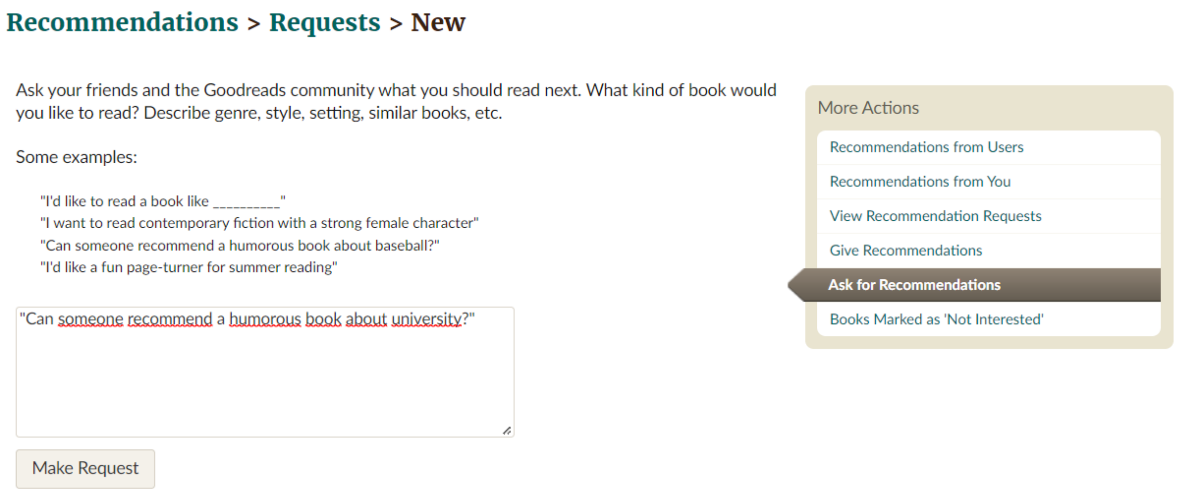
Recommendations by Genre

Fiction	50
Graphic Novels	28
Historical Fiction	51
Mystery	50
Romance	50
Science Fiction	50

More Actions

Recommendations from Users
Give Recommendations
Ask for Recommendations
Books Marked as 'Not Interested'

Figure 03. - Recommendation Filters



Recommendations > Requests > New

Ask your friends and the Goodreads community what you should read next. What kind of book would you like to read? Describe genre, style, setting, similar books, etc.

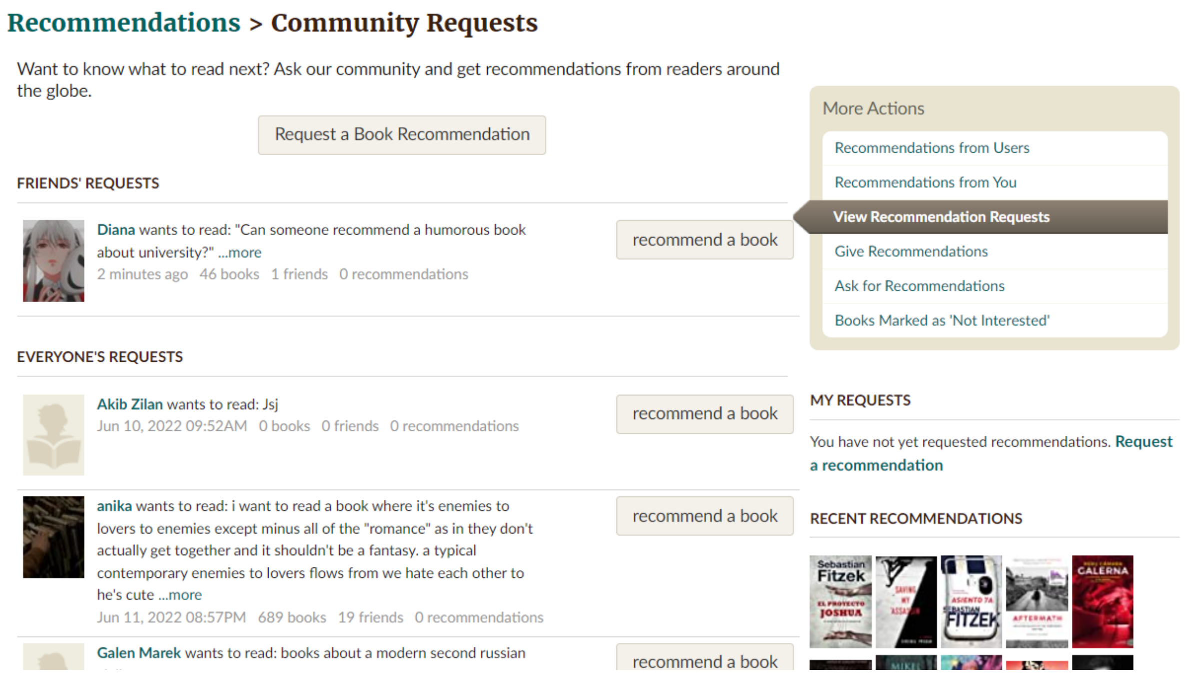
Some examples:

"I'd like to read a book like _________"
"I want to read contemporary fiction with a strong female character"
"Can someone recommend a humorous book about baseball?"
"I'd like a fun page-turner for summer reading"

"Can someone recommend a humorous book about university?"

Figure 04. - Ask for Recommendations Page

In addition to these forms of recommendation, the platform also allows the user to ask for a recommendation in the "**Ask for Recommendations**" section. This request is not only sent to the user's friends list or the people who follow them, this request is sent to the entire platform community. However, it appears as a priority in the "**View Recommendation Requests**" section of users who are friends with the person who requested the recommendation.



Recommendations > Community Requests

Want to know what to read next? Ask our community and get recommendations from readers around the globe.

FRIENDS' REQUESTS

Diana wants to read: "Can someone recommend a humorous book about university?" ...more
2 minutes ago 46 books 1 friends 0 recommendations

EVERYONE'S REQUESTS

Akib Zilan wants to read: Jsj
Jun 10, 2022 09:52AM 0 books 0 friends 0 recommendations

anika wants to read: i want to read a book where it's enemies to lovers to enemies except minus all of the "romance" as in they don't actually get together and it shouldn't be a fantasy. a typical contemporary enemies to lovers flows from we hate each other to he's cute ...more
Jun 11, 2022 08:57PM 689 books 19 friends 0 recommendations

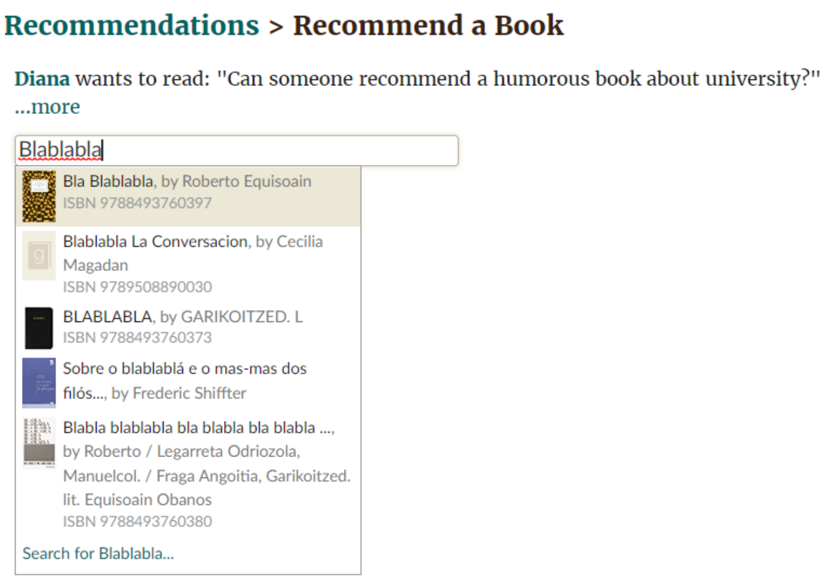
Galen Marek wants to read: books about a modern second russian

MY REQUESTS

You have not yet requested recommendations. Request a recommendation

Figure 05. - View Recommendation Requests Page

By clicking on the "recommend a book" button, it is possible to navigate to another section where the user can write directly in an input box the name of the book that they think best fits the recommendation request and, then, if it exists in the platform data, they can select it and submit it together with an optional text message.



Recommendations > Recommend a Book

Diana wants to read: "Can someone recommend a humorous book about university?"
...more

Blablabla

Bla Blablabla, by Roberto Equisoain
ISBN 9788493760397

Blablabla La Conversacion, by Cecilia Magadan
ISBN 9789508890030

BLABLABLA, by GARIKOITZED. L
ISBN 9788493760373

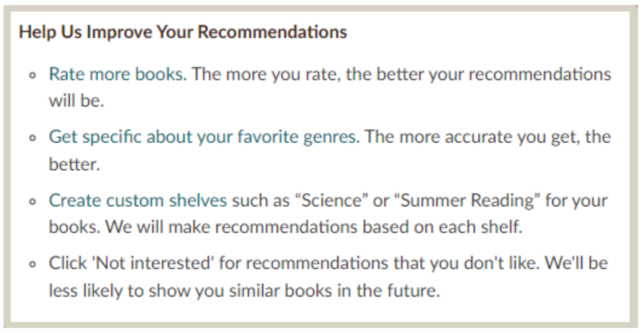
Sobre o blablablá e o mas-mas dos filós..., by Frederic Shiffter

Blabla blablabla bla blabla bla blabla ..., by Roberto / Legarreta Odriozola, Manuelcol. / Fraga Angoitia, Garikoitzed. lit. Equisoain Obanos
ISBN 9788493760380

Search for Blablabla...

Figure 06. - Recommend a Book Page

The user can also improve the results that the recommendation system offers them through several ways: evaluating more books; creating shelves with more specific criteria; rating recommendations they don't like as "Not interested" and getting specific about their favorite genres.



Help Us Improve Your Recommendations

- Rate more books. The more you rate, the better your recommendations will be.
- Get specific about your favorite genres. The more accurate you get, the better.
- Create custom shelves such as "Science" or "Summer Reading" for your books. We will make recommendations based on each shelf.
- Click 'Not interested' for recommendations that you don't like. We'll be less likely to show you similar books in the future.

Figure 07. – How to improve their recommendations Section

4. Recommender Models

When talking about recommender systems for a user, we are talking about systems that filter information in order to suggest content adapted to each user. In this topic, we will explore different approaches that can be taken when developing a recommender system.

Simple Recommender Model

When talking about the Simple Recommender Model, it refers to a model that, in this context, will use only one parameter for book recommendations: the **popularity** that a book has. These parameters are related according to the following proposition: the more popular and well acclaimed a book is, the more likely it is to please the public that has not yet read it.

As can be seen, the main disadvantage of this model is the fact that it does not take into account the user's tastes in order to customize the recommendations that are made.

When deciding to implement the algorithm based on this model, the only concern to have is to sort the books in ascending order of their ranking and popularity.

Content Based Filtering Recommender Model

When talking about the Content Based Filtering Recommender Model, it refers to a model that will use as parameters for recommendation the feedback provided by the user and their previous actions, that is, in this context, the books that the user liked. The proposition to follow now is: if the user liked a certain book, this one, because it is **similar**, is more likely to please him.

The limitation of this model is, as can be seen, in the fact that it does not fully capture the user's tastes; book recommendations are based on the most similar books of a given book, so users who have read that book will always get the same recommendations, regardless of their personal preferences.

When deciding to implement the algorithm based on this model, we need to take into account the similarity matrix between the books.

Collaborative Filtering Recommender Model

When talking about the Collaborative Filtering Recommender Model, it refers to a model that, in this context, will use as a parameter for recommending books that can please the user **other users** who have, to a certain extent, **similar tastes** to the user in question. The proposition to follow now is the following: if a user with similar tastes to mine liked this book that I haven't read yet, the probability that I will like it too is high.

The main limitation of this approach is when dealing with a user who is new to the system, that is, has just registered and does not yet have data to help with the analysis – this limitation is called a "**cold-start problem**". That's why the Goodreads recommendation system is only available after the user has evaluated at least 20 books.

For the development of this report, I decided to implement a recommendation algorithm based on the Collaborative Filtering Recommender Model. Although the code can be found in the "Appendix" section of this report or in my GitHub [09], I will explain in the "The Algorithm" section each important step that was done.

5. Work Environment

Before exploring the developed algorithm, it is important to assimilate some basic concepts in order to better understand what was accomplished.

Kaggle

To develop the algorithm, it was necessary to use some datasets. A dataset is a data structure that represents a set of information related to each other according to certain criteria. For example, an Excel file that contains the name of all students of a certain course at the University of Aveiro and the final grade they obtained in the APSEI curricular unit can be considered a dataset.

The datasets that will be used in the future (one referring to the books that exist on the Goodreads platform and another referring to the evaluations attributed by users of the platform) can be found on the Kaggle website [10], a website known for **having a huge variety of datasets** from different areas and themes. Although there are other options for sites that offer datasets, I chose Kaggle out of familiarity with the platform.

GitHub

As mentioned earlier, the developed code is on my GitHub. Briefly, GitHub [11] is a cloud storage site for data, that is, for storing data on the internet. Using GitHub is a very common practice among programmers, as it is a way to ensure that their work is **stored** elsewhere (**cloud**) other than the computers (local) and thus prevent that, if the computer is damaged, the work is lost. In addition, GitHub also allows several people to work on the same project at the same time (**collaboration**) and allows version control of that project (**control**). The information is then stored in repositories (basically a folder) that the user creates themself and that can be public or private. Another interesting fact is that, in the IT area, a programmer's GitHub is currently extremely important for their curriculum, since it contains all or most of the projects that the programmer has developed over time; it's like a portfolio.

Python

The algorithm code presented in the next section was developed in the python programming language [12]. I chose to use this language because python is one of the most basic programming languages; if someone wants to start delving into the world of programming, the most common thing is for that person to start by learning python. It is a language with a simple syntax and that is very close to the human language and, therefore, developing the algorithm code in python will make it **easier for the reader to understand it**, even if the person who is reading this report does not have any programming experience.

VS Code Editor

To write code, the programmer can choose to write it in various code editing applications, i.e. editors, even in Notepad if they want to. Most programmers, for writing code in basic languages like python, opt for the open source VS Code Editor [13], since this editor supports several programming languages and has useful tools and extensions for coding. For the sake of familiarity with the editor, I chose it to code the code found in the next section.

Folder Organization

During the analysis of the algorithm code, it will be observed that it was necessary to import the chosen datasets. In order to import files it is necessary to know where they are located and, for this reason, we proceeded to the following organization of the folders:

	code	*python file*
main folder	*datasets*	*books.csv, ratings.csv, etc...*
	report	*this report*

When we want to import files that are in the same folder, the directory is given as "./" + the name of the file. For example, if we want to import a file "name" that is in the code folder to the python file we use "./name".

When we want to import files that are located in other folders, the directory is given in another way: its initial symbol will be "../" instead of "./" and this symbol will be repeated the number of times equal to the distance between folders . Take for example the following folder organization:

		name_B file		*name_D file*	
A folder	*B folder*	*C folder*	*D folder*	*E folder*	*name file*

If in the file "name" we want to import something that is in folder D, the directory will be "../D/name_D", but if we want to import something that is in folder B the directory will be "../../../ B/name_B". The distance between folder E, which is where the file "name" is located, and folder D is 1, so we write "../" once; the distance between folder E and folder B is 3, so we write "../" three times.

As you can see, the directory in cases where the files we want to import are not in the same folder is given with "../" the number of times necessary + name of the folder where the file is + the name of the file.

Running the python file

To run a python file, we can use the VS Code Editor terminal or the command line of the computer itself. Since VS Code Editor has already been mentioned above, we will use the VS Code Editor terminal.

1. Open the python file in VS Code Editor.
2. In the program window header, click the triangular button on the upper right.
3. Done, the file is running.

We can also, instead of pressing the triangular button, click on View > Terminal. This action will open the VS Code Editor terminal, but it will probably not be open in the directory where the python file is. To go to the directory where the file is, type the command "cd " + directory where you want to go. If you don't know what the directory is, when opening a folder in Windows, the directory of the same is shown in the upper corner.

Once in the directory, type the command "python " + python filename + ".py" to run the python file, for example “python file.py”. If you have never programmed in python before, you may need to install python on your computer.

Therefore, after the explanations given, if someone wants to reproduce the code that was developed, they can do so using the code editor VS Code to code in Python language, making use of the datasets found on the Kaggle website and storing the code on GitHub for it to be stored elsewhere beyond the computer itself. All the links necessary to access the mentioned tools can be found at the end of this report.

6. The Algorithm

This section is dedicated to present and explain step by step the code written to develop the recommendation algorithm based on the Collaborative Filtering Recommender Model. In order to be able to present a more detailed explanation, it will be possible to observe a numbering of the lines of the code in **bold**; this expression (line 1:, line 2:, line 3:, ...) is **not** part of the code, it only serves to help reference the part of the code that is being addressed.

```
line 1: books = p.read_csv('../datasets/books.csv')
line 2: ratings = p.read_csv('../datasets/ratings.csv')
line 3: books_ratings = p.merge(books, ratings)
line 4: books_ratings['number_of_reviews'] =
books_ratings['user_id'].groupby(ratings['user_id']).transform('count')
line 5: books_ratings = books_ratings.loc[(books_ratings['number_of_reviews']
>= 20)]
line 6: books_ratings = books_ratings.filter(items=['user_id', 'title',
'rating'])
line 7: d = (books_ratings.groupby('user_id')[['title','rating']].apply(lambda
x: dict(x.values)).to_dict())
```

The first step in code development was to find datasets that fit the context; in this case, on the Kaggle website it was possible to find a dataset with the books existing on the Goodreads platform (`books.csv`) and a dataset with the ratings made by some users of the platform (`ratings.csv`). [04]

After the datasets have been downloaded and stored in the correct folder we proceed to import the datasets. The `books` variable will store information regarding the `books.csv` file (**line 1**) while the `ratings` variable will store information regarding the `ratings.csv` file (**line 2**). To simplify the content filtering process, I chose to combine the two variables into one, in order to have the information of both datasets in the same variable, and I assigned the name `books_ratings` to that variable (**line 3**).

Then, a new field was created in the `books_ratings` variable called `number_of_reviews` in order to have information on how many reviews were made by a given user. To do this, we track all ratings made by each user and count the total number of ratings made by them (**line 4**).

The `number_of_reviews` variable was not made without reason; as previously mentioned, when using the Collaborative Filtering Recommender Model we encountered a problem called "cold-start problem" and Goodreads gets around this problem by making its recommender system only available to users who have done at least 20 reviews. Therefore, the next step is to filter the information contained in the `books_ratings` variable so that we only use data from users whose `number_of_reviews` is greater than or equal to 20 (**line 5**).

After this filtering, we already have the `books_ratings` variable ready to be used, however I chose to do an additional step in the filtering in order to eliminate unnecessary fields. Therefore, the fields present in the variable become just the `user_id` (the user's id), the `title` (the book's title) and the `rating` (the rating that the user assigned to the book) (**line 6**).

The `book_ratings` variable is a structure called data frame; however, in python it is easier to work with structures called dictionaries. A dictionary is characterized by the information contained in it being organized in key-value pairs and, in this case, the variable `d` will be a dictionary that will have the `user_id` field as its key and another dictionary as its value.

This other dictionary has as a key the title of a book that the user has rated and as a key the respective rating (**line 7**).

To calculate the similarity between users I used the Euclidean Distance Model formula:

$$distance = \sqrt{\left[\left((x_2 - x_1)^2 + (y_2 - y_1)^2\right)\right]}$$

```
    line 8: def euclidean_distance(dictionary,user_1,user_2):
    line 9:     similar = 0
    line 10:    for x in dictionary[user_1]:
    line 11:        if x in dictionary[user_2]:
    line 12:            similar += pow(dictionary[user_1][x] - 
dictionary[user_2][x],2)
    line 13:    if similar == 0:
    line 14:        return 0
    line 15:    return 1/(1+similar)
```

The Euclidean Distance Model formula is calculated in the `euclidean_distance` function that takes as arguments the dictionary `d`, and two `user_id` keys (`user_1` and `user_2`) (**line 8**).

In this function we start by defining a variable called `similar` (**line 9**) and then, for each book associated with `user_1` (**line 10**), if that same book is also associated with `user_2` (**line 11**), that is, if both users read and rated the same book, we calculate the formula (**line 12**). The variable `x` present in lines 10, 11 and 12 represents the title of the book and, therefore, the expression `dictionary[user_1][x]` that appears in line 12 will represent the classification that this book obtained by user_1.

At the end of the similarity check between users, if the value of the `similar` variable is 0 (**line 13**), that is, there is no similarity between users, the function returns the value 0 (**line 14**). Otherwise, the function will return the similarity value on a scale from 0 to 1 (**line 15**). I chose to do this scaling because it is easier to perceive values that are between a predefined range than values that are between 0 and infinity.

The `euclidean_distance` function is used by two other functions: `get_users` and `get_recommendations`.

```
    line 16: def get_users(dictionary, user, n=10, similarity = 
euclidean_distance):
    line 17:    scores = []
    line 18:    for other_user in dictionary:
    line 19:        if other_user != user:
    line 20:            scores.append((similarity(dictionary,user,other_user),
other_user))
    line 21:    scores.sort()
    line 22:    scores.reverse()
    line 23:    return scores[0:n]
```

The `get_users` function was written to return a list with the greatest similarities found for a given user, receiving as parameters the dictionary `d`, the `user_id` of the user we want to analyze, the size of the list we want to get back and the function to calculate the `similarity`. By default, the last two arguments are previously defined as `n` with a value of 10 and `similarity` with the `euclidean_distance` function (**line 16**).

The list that will be returned is defined in the first line of the function with the name of `scores` (**line 17**). Then, we iterate the dictionary with the information of each user and, for each `other_user` found (**line 18**), if that `other_user` is not the `user` we are analyzing (**line 19**), we add to the `scores` list the value returned by the function application `euclidean_distance` from the two users, `other_user` and `user`, and the value of the `other_user` (**line 20**).

Finally, before returning the list, we sort the list in ascending order (**line 21**) using the `sort()` method, typical of the python language, and then we obtain the list in descending order (**line 22**) through the `reverse()` method, also typical of the python language. We end with the return of the first `n` elements of the `scores` list (**line 23**), which will be the `n` most similar users with the analyzed `user` and the respective similarity value.

An important thing to mention that will also be seen in the next function is that I decided to use the function name `euclidean_distance` as an argument of the function `get_users` instead of directly applying the name `euclidean_distance` on line 20. This is a good programming practice, since in the future this code can be improved and have other functions to calculate the similarity between users and, in this way, when making the call to the `get_users` function, the programmer can define which function will be used to calculate similarity and, if it does not define any, by default the `euclidean_distance` will be used.

```
    line 24: def get_recommendations(dictionary, user, n=10, similarity =
euclidean_distance):
    line 25:     total = {}
    line 26:     similar = {}
    line 27:     rankings = []
    line 28:     for other_user in dictionary:
    line 29:         if other_user == user:
    line 30:             continue
    line 31:         sim = similarity(dictionary, user, other_user)
    line 32:         if sim == 0:
    line 33:             continue
    line 34:         for x in dictionary[other_user]:
    line 35:             if x not in dictionary[user]:
    line 36:                 total.setdefault(x,0)
    line 37:                 similar.setdefault(x,0)
    line 38:                 total[x] += dictionary[other_user][x] * sim
    line 39:                 similar[x] += sim
    line 40:     for book, t in total.items():
    line 41:         rankings.append((t/similar[book]/5.0, book))
    line 42:     rankings.sort()
    line 43:     rankings.reverse()
    line 44:     return rankings[0:n]
```

The get_recommendations function was written to return a list with the books most likely to please a given user, receiving as arguments the dictionary `d`, the `user_id` of the user we want to analyze, the size of the list we want to get back and the function to calculate the `similarity`. By default, the last two arguments are previously defined as `n` with a value of 10 and `similarity` with the `euclidean_distance` function (**line 24**).

The list that will be returned by the function is defined at the beginning with the name of `rankings` (**line 27**) together with the `total` variable (**line 25**) and the `similar` variable (**line**

26). Then, we iterate the dictionary with the information of each user and, for each `other_user` found (**line 28**) we apply two checks: the first one is to verify if the `other_user` is not the same as the `user` (**line 29**) and, if it is, we analyze the next user (**line 30**); the second is to verify the similarity between the `other_user` and the `user`, storing it in the `sim` variable (**line 31**). If the value of the `sim` variable is 0 (**line 32**), it means that the users are not at all similar and therefore it is not worth continuing to apply the code for that `other_user`, so we move on to the next user (**line 33**).

In the next step, we iterate over the values associated with the `other_user` key, that is, we iterate over the books they read and rated (**line 34**). If the book we are analyzing is only associated with the `other_user` but not with the `user` (**line 35**) we assign the key-value pair, with key being `x`, that is, the title of the book, and value initially being 0 to the `total` dictionaries (**line 36**) and `similar` (**line 37**). Then we update the value of 0 in the `similar` variable to the previously calculated `sim` value (**line 39**) and in the `total` variable to the `sim` value to be multiplied by the rating that the `other_user` assigned to the analyzed book (**line 38**). This multiplication symbolizes the weight that the suggestion has, taking into account the similarity and classification of the book.

In the end, we will have a `total` dictionary with all the books suggested for the user and the weight of the suggestion and a `similar` dictionary with the sum of the similarities for each book. However, what we will add to the `rankings` list is, for all elements of the `total` dictionary (**line 40**), a tuple that contains the `book` and the probability that it will please the user (**line 41**).

Finally, before returning the list, we sort the list in ascending order (**line 42**) using the `sort()` method, typical of the python language, and then we obtain the list in descending order (**line 43**) through the `reverse()` method, also typical of the python language. We end with the return of the first `n` elements of the `rankings` list (**line 44**), which will be the `n` most likely books to please the user.

```
line 45:     print(get_users(d, 3))
line 46:     print(get_recommendations(d, 3))
```

Once the functions have been developed, the next step is to call them. As both functions, `get_users` and `get_recommendations` have 4 parameters, 2 of which already have a default value (the `n` parameter and the `similarity` parameter), let's just pass the dictionary `d` as parameters, a dictionary that contains all the users' information, and the `user_id`. I used `user_id` 3, but any `user_id` can be used as long as it is in the `d` dictionary, that is, as long as the `user_id` has evaluated more than 19 books.

As I want the result of what both functions return to be displayed, I use the `print()` method, a method specific to the python language, so that the returned data appears in the terminal (**line 45** and **line 46**).

7. The Impact

An activity that should be seen as a hobby, something to feel relaxed and have fun, can become a competition and, consequently, diminish the pleasure we feel when doing it. If reading becomes a compulsion just for the user to be able to observe the number of books on their "read books" shelf increase, then they stop really enjoying reading and the situation becomes just about numbers. And we don't just talk about the act of reading, but also, for example, writing or responding to reviews of a particular book, an action possible to do on the platform. "I have to have more reviews", "I have to have the best reviews", thoughts of this kind influence users of the platform, whether they are readers who are writing the reviews or authors who have published their book on the platform and expect to receive reviews.

Since we're talking about reviews, we can address the topic of: how fair is it to give a book a rating from 0 to 5? After all, art is art; can art be classified on a scale? How fair is it for a book to receive a 5-star rating from one user, but receive a 2-star rating from another? And what impact will it have on the author receiving a rating on a book they have worked hard to write?

The truth is that platforms that give their users the freedom to express themselves must be regularly monitored and set limits. We currently live in a time where we are faced with a lot of human intolerance, which gave rise to the famous "cancellation". For example, if an author has an attitude that is not very popular with the public, what is the easiest way to reach that author using platforms like Goodreads? Simple: rating their books with a 1-star rating. But since we decided to quantify a work of art, does this rating correspond to the quality of the book? In the midst of a cancellation, it probably won't match how good the book is, but rather will be the result of public outrage. So how reliable are the reviews found on the platform? Who can guarantee that a book that has a not very positive review is not a book worthy of a high rating, yet the author has an audience that doesn't have a good image of them?

We can use author Patrick S. Tomlinson as an example; for approximately a year and a half, this author was the victim of a group that created several accounts on the platform, these accounts with offensive and inappropriate names, to leave more than 1000 negative and defamatory reviews on several of his works.

In the particular case of the Goodreads platform, it is allowed to evaluate a book even before it is released, provided that the author has made the content available on the platform. The impact that this action has can be positive and serve to publicize the work and reach a wider audience; or it can be negative and if the author falls victim to a wave of negative reviews, even before the book is released, affect the release of the work. It has happened, for example, that users rate a book negatively just because it had a queer character, without even having finished reading the work.

One situation that happened was that author Beth Black received emails from an anonymous server threatening her that if she didn't pay for her book to be positively reviewed, then her book would receive a lot of negative reviews.

> "EITHER YOU TAKE CARE OF OUR NEEDS AND REQUIREMENTS WITH YOUR WALLET OR WE'LL RUIN YOUR AUTHOR CAREER. PAY US OR DISAPPEAR FROM GOODREADS FOR YOUR OWN GOOD."

Beth Black did not give in to the blackmail and, a few hours later, she found that her book's rating had indeed begun to decline. However, Beth Black wasn't the only author to be threatened, this is just one example of how the platform can become a weakness for some authors, especially novice authors who are still building their careers.

What I mean by this is that while the platform is good at recommending books and managing the books users read, it doesn't have the best policy when it comes to dealing with the bad intentions of some users and possible harassment that some authors may suffer. And even if the authors want to ignore the existence of the platform, since it can be harmful to them, the truth is that Goodreads, at this moment, has grown so much to the point of being the most influential platform in this market, making it impossible to ignore it as a strategy of marketing.

8. References

[01] **Summary/Framework**: Goodreads: what is? - https://en.wikipedia.org/wiki/Goodreads
[02] **Framework**: Stats on November 29, 2021 -
https://expandedramblings.com/index.php/goodreads-facts-and-statistics/
[03] **Framework**: Announcing the Recommendation System -
https://www.goodreads.com/blog/show/303-announcing-goodreads-personalized-recommendations
[04] **The Algorithm**: Datasets –
https://www.kaggle.com/code/omarzaghlol/goodreads-1-the-story-of-book/data
[05] **The Impact**: Is Goodreads a good thing? -
https://bookriot.com/is-goodreads-a-good-thing/
[06] **The Impact**: Reviews Bombing - https://time.com/6078993/goodreads-review-bombing/
[07] **Recommender Models**: Models -
https://www.kaggle.com/code/omarzaghlol/goodreads-2-book-recommender-system/notebook
[08] **The Algorithm**: Euclidean Distance -
https://mickzhang.com/goodreads-recommendation-using-collaborative-filtering
[09] **Recommender Models**: GitHub Repository -
https://github.com/DianaSiso/Recommendation-system
[10] **Work Environment**: Kaggle Site - https://www.kaggle.com
[11] **Work Environment**: GitHub Site - https://github.com
[12] **Work Environment**: Python Site - https://www.python.org
[13] **Work Environment**: VS Code Site - https://code.visualstudio.com

9. Appendix

```
import pandas as p
from collections import Counter

# returning distance based similarity between user_1 and user_2 using Euclidean
distance score
def euclidean_distance(dictionary,user_1,user_2):
    similar = 0
    for x in dictionary[user_1]:
        if x in dictionary[user_2]:
            print(x)
            print(dictionary[user_1][x])
            similar += pow(dictionary[user_1][x] - dictionary[user_2][x],2)
    if similar == 0:
        return 0
    return 1/(1+similar)

# getting best 10 users using Euclidean distance score
def get_users(dictionary, user, n=10, similarity = euclidean_distance):
    scores = []
    for other_user in dictionary:
        if other_user != user:
            scores.append((similarity(dictionary,user,other_user), other_user))
    scores.sort()
    scores.reverse()
    return scores[0:n]

# getting best 10 recommendations using Euclidean distance score
def get_recommendations(dictionary, user, n=10, similarity = euclidean_distance):
    total = {}
    similar = {}
    rankings = []
    for other_user in dictionary:
        if other_user == user:
            continue
        sim = similarity(dictionary, user, other_user)
        if sim == 0:
            continue
        for x in dictionary[other_user]:
            if x not in dictionary[user]:
                total.setdefault(x,0)
                similar.setdefault(x,0)
                total[x] += dictionary[other_user][x] * sim
                similar[x] += sim
    print(total)
    print(similar)
    for book, t in total.items():
        rankings.append((t/similar[book]/5.0, book))
    rankings.sort()
    rankings.reverse()
    return rankings[0:n]

# importing and merging data
books = p.read_csv('../datasets/books.csv')
ratings = p.read_csv('../datasets/ratings.csv')
books_ratings = p.merge(books, ratings)
```

```
# considering only users that reviewed at least 20 books
books_ratings['number_of_reviews'] =
books_ratings['user_id'].groupby(ratings['user_id']).transform('count')
books_ratings = books_ratings.loc[(books_ratings['number_of_reviews'] >= 20)]
books_ratings = books_ratings.filter(items=['user_id', 'title', 'rating'])

# analysing data
# checking how many times the book was rated
print(Counter(books_ratings['title']))

# checking how many times different users rated the same book
print(books_ratings.groupby('title')['user_id'].unique())

# filtering the unique user_id for their corresponding title
# books_ratings_2 = books_ratings.set_index(['user_id', 'title']).sort_index()

# converting dataframe to dict
d = (books_ratings.groupby('user_id')[['title','rating']].apply(lambda x:
dict(x.values)).to_dict())

# top 10 similar users to user id 3
print(get_users(d, 3))

# top recommendation for user id 3
print(get_recommendations(d, 3))
```