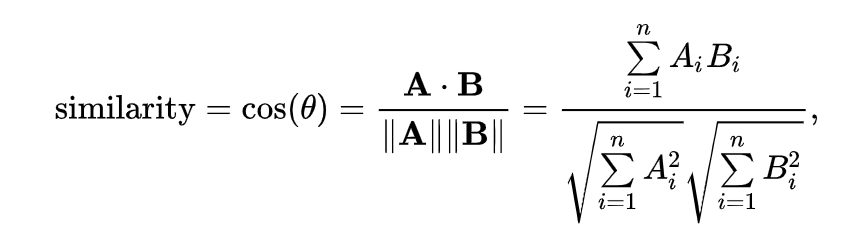
Part 3:

1. Algorithms and Data Structures **(if applicable)(shaz, alan, seymour, vedanta)**
   1. Algorithms **(Alan, Vedanta)**  
      Describe the algorithms that implement mathematical models from your Report #1. Does your system use any other complex algorithms? For example, when computing a motion trajectory for an animate figure in a game, you may use some numerical or computer-graphics algorithms. Or, when assessing stock market movements, you may be using statistical algorithms.   
      If NO, skip to the next item;   
      If YES, describe your algorithms. For example, for the animate figure example above, will the path coordinates be precomputed and stored in a look-up table or will they be computed using a spline interpolation algorithm.



For the recommending algorithm, we used a common similarity algorithm called the cosine similarity algorithm. It essentially maps all the genre and author tags contained in our database on a vector coordinate system. To see how two or more books may be similar to each other it measures the angle between the two vectors and then uses the equation above to measure how close they are two each other. If the books are highly similar the value will be close to 1, but the farther away the book or vector is the score will get closer to 0. To find the set of books that are similar to a set of books inputs, we find the matrix of similarity scores for each input book and then add up the all the scores. We then only have to sort it and give back the top results that are the most similar. This seems like a lot of steps, but using the pandas library in python a lot of the brute math is done for us using the built in methods.   
It is a good idea to use activity diagrams to describe the algorithm design.

* 1. Data Structures **(Shaz, Seymour)**  
     Does your system use any complex data structures, such as arrays, linked lists, hash tables, or trees?
     1. For our data structure we are using Lists from python. The reason that we are using lists is because lists provide us with the ability to add elements as we go. This feature is very useful due to the flexibility it allows since part of our algorithm searches the books entered, and collects all the tags as they go through the books. The flexibility plays a part in the role of the structure because the user can input an infinite amount of books. Secondly, with the use of list we are able to combine multiple lists together in order to create a resultant pseudo hashtable. For example, if we want to see how many times a certain tag was found; we could have one list for the tags generated from the data of each book, then a separate list for the count of the occurence of each tag. After the data from the user ceases to grow any further, we can simply combine the two lists into one list that has (Tag,Count) as their elements. This makes it much easier for us to visualize the data and go through it. The main idea of the approach was to emphasize on flexibility due to the large amounts of data we have, and how vulnerable these data is to change.  
        If NO, skip to the next item;   
        If YES, what criteria you used in deciding what data structure to use, e.g., performance vs. flexibility?

1. User Interface Design and Implementation **(akshat, anthony, alan)**
   1. Describe whether and how you modified and implemented the initial screen mock-ups developed for [Report #1](https://www.ece.rutgers.edu/~marsic/Teaching/SE/report1.html). Comment only on significant changes in your user interface, those that reduce (or increase) the [user effort](https://www.ece.rutgers.edu/~marsic/Teaching/SE/report1-appA.html). Changes of colors or styles are less important and should be omitted from your report.

From the mock-up, it was originally planned to display the books with their images on the homepage of the site. However, on the home page the only features that will appear on there are just the search bar and a horizontal menu which consists of a genre button and a favorites button. The genre button gives a dropdown of all possible genres for books and the favorites gives a dropdown of the user’s favorite books. The horizontal menu will be positioned at the top and the search bar is centered in the middle of the home page.

* 1. The textbook does not deal much with the GUI design. Excellent guidelines for GUI design can be found here:   
     Sun Microsystems, Inc. Java Look and Feel Design Guidelines. Mountain View, CA, 1999. Available at: <http://java.sun.com/products/jlf/ed2/book/>
  2. “Ease-of-use” is generally considered a key characteristic of user interface. “Ease-of-use” should not be confused with a flashy interface, with lots of colors, picture, graphics, etc. On the contrary, you should avoid flashy user interfaces. “Ease-of-use” means that interface is intuitive, easy to understand and operate, without having to ask many questions or read voluminous documentation. A minimal user interface that is well organized should be sufficient. You already considered the user effort as part of [Report #1](https://www.ece.rutgers.edu/~marsic/Teaching/SE/report1.html), and here you should strive to minimize the user effort, thus maximizing the “ease-of-use”.

1. Design of Tests **(Shaz, Seymour, Vedanta, Akshat, Anthony)**

Note that for this report you are just designing your tests; you will program and run those tests as part of work for your first demo, [see the list here](https://www.ece.rutgers.edu/~marsic/Teaching/SE/demo1.html#GRADING).

* 1. List and describe the test cases that will be programmed and used for unit testing of your software **(Seymour, Akshat)**
     1. The test cases for our program are fairly simple because we just have to input in various books ranging from genres that are related like Harry Potter and Lord of the Rings, to books that are different like Romeo and Juliet and the Hunger Games. We will also input incorrect spelling of titles, null values, and books that do not exist. (enter some example test cases if you can)
     2. The main idea of the list data structure is in order break down the describing characteristics of our user input, and put all the information of the inputted tags all in one place (a different list). So the testing for this data structure was fairly simple as we only needed to make sure that the number of times certain tags came up in our inputted list was recorded. So being as that we already have developed a work-in-progress database, we used some of the books from that database in order to test different quantities of inputs from a user, and made sure that the list accurately batched up all of the data from those inputs accurately.
  2. Discuss the test coverage of your tests **(Vedanta, Alan)**
     1. The test coverage of our tests is mainly for the front end because by the time the inputs arrive to the backend to be used in the algorithm they should already be preprocessed to make sure the spelling is correct, the books exist, and the correct numbers of inputs have typed. The front end is in charge of telling the user to keep on entering the correct inputs until the desired format of inputs is correct. At that point we can assume, the program will return the recommended books without any problem. Like any recommendation programs, depending on the inputs sometimes you may get back books that are actually similar to the books you inputted, but if you input a bunch of drastically different books the program will have a hard time returning back similar books for you to read.
     2. Testing for the backend will require more effort. Using a sample set of books and estimating return values, we can adjust our algorithm multipliers (for different tag types) in order to return books that are most like the books we’ve inputted. With repetition we should be able to enter any list of books and receive a list of books which has strong correlation to those entered.
  3. Describe your Integration Testing strategy and plans on how you will conduct it **(Vedanta, Akshat)**
     1. Our integration testing is going to be focused on multiple book input and ensuring that having more than one changing variable will not drastically affect the outcome of our product.
     2. We are going to show that our system can take infinite book inputs and generate the recommended output almost instantly. For this, the length of book inputs is until the last input added in the search bar (of our front end site) until the Search Icon is clicked.
     3. We are going to show that the system is able to detect spelling errors and predict an appropriate book name based on the list we have in our database. For this, we will intentionally make some spelling mistakes for some book of our choice and show that the suggestion matches the input we are looking for.
  4. Describe also your plans for testing any algorithms, non-functional requirements, or user interface requirements that you might have stated in your Report #1 **(Shaz, Anthony)**
     1. User Interface Testing: Test how the predictive text matches up. What happens when we start typing random letters into the search bar.
     2. Algorithm: We will give one specific input and know all tags that are associated with that book. We will then see if the tag collector gets all the tags properly and how to we order/ weight them.

1. Project Management and Plan of Work **(Avani, Kutay)**

\*Everyone has to reference their independent contributions\*

* 1. Merging the Contributions from Individual Team Members  
     Compiling the final copy of the report from everyone’s work, ensuring consistency, uniform formatting and appearance.  
     Describe what issues were encountered and how they were tackled.
  2. Project Coordination and Progress Report  
     What use cases have been implemented?   
     What is already functional, what is currently being tackled?  
     List and describe other relevant [project management](https://www.ece.rutgers.edu/~marsic/Teaching/SE/projects.html#TEAMS) activities.
     1. Done with the recommendation algorithm and test cases-Alan
  3. Plan of Work  
     List the projected milestones and dates by which you plan to accomplish them. Preferably, you should use [Gantt charts](http://www.ganttchart.com/) for planning and scheduling your project.
  4. Breakdown of Responsibilities
     1. List the names of modules and classes that each team member is currently responsible for developing, coding, and testing
     2. Who will coordinate the integration?
     3. Who will perform and integration testing? (The assumption is that the unit testing will be done for each unit by the student who developed that unit.)