Project Report_COP5090

Sai Srujana Gunda, Brunda Uppalapati, Dhrishya Menon

2023-11-25

PROJECT REPORT- BOOK RECOMMENDER APP

I. INTRODUCTION

The objective of this project is to develop a *Shiny app for book recommendation* based on the provided dataset. The primary objective of the Book Recommender app is to *offer users a curated list of the top 10 books based on their entered keyword(s)*. Upon submitting a keyword or keywords into the app, users will receive a tailored list of the highest-rated books that match their search criteria.

The dataset for this project was obtained from **Kaggle**, the data science competition platform. The dataset contains the most popular data science books that has been scraped from Amazon.com.

The dataset consists of a total of 946 books. There are in total 18 columns in the dataset out of which mainly only 4 columns `title` , `author` , `ISBN-13` and `avg_reviews` are used for the project.

II. PRELIMINARY STEPS

Data Pre-processing:

- The data preprocessing phase involved several steps which were undertaken to refine the dataset for analysis purposes.
- The four columns `title` , `author` , `ISBN-13` and `avg_reviews` in the dataset are renamed to `Title` , `Author` , `ISBN` , `Rating` . Notably, a crucial step involved the addition of a new column named `contents` . The column has been compiled from multiple sources and houses the table of contents for each of the books. This addition greatly enhances the dataset by providing a detailed perspective of each book's theme elements and structural arrangement.
- The `contents` column was essential to improving the recommendation system's effectiveness and accuracy by providing a comprehensive understanding of the content organization in each book.

Data Cleaning:

- The next vital step that was performed was data cleaning which focused on ensuring the dataset's integrity and readiness for analysis by addressing various data quality issues.
- The initial data cleaning step involved addressing missing values or empty values. To improve uniformity and standardize the data, the blank strings in the `Author` column were carefully handled as NA (Not Available) values. These rows were subsequently removed, ensuring the dataset's robustness and completeness.
- The data cleaning steps enabled creating a refined dataset that could be used for exact analysis and recommendation-making in the project.

III. DEVELOPMENT OF SHINY APP

The library `shiny` is deployed in the project. It is a powerful and versatile framework in R and assists in the creation of interactive web applications. Shiny is designed to make it simple for users to turn their R scripts into interactive web apps that support real-time data manipulation, user interaction, and ideal visualizations. It helps in creating dynamic web-based interfaces without having to learn complicated web development languages. The framework's straightforward design, combined with its broad range of UI components and reactive programming features, makes it an important tool for creating engaging and interactive data-driven apps in R.

UI:

- The development of a user-friendly and aesthetically pleasing user interface (UI) for the Shiny app was a key component of the project's development. Utilizing a blend of HTML, CSS, and Shiny's layout features, the user interface was designed to provide an interesting and intuitive experience.
- An ==engaging background image== was a key component of the UI design. The well-chosen picture, depicting a library filled with books, had an essential part in creating the app's overall theme. The background application of the picture was set up with cover size, fixed attachment, and no-repeat attribute constraints, which gave the interface more visual appeal and depth.
- The readability and aesthetic appeal were enhanced with the help of ==font styles==. *Georgia* and *Lato* were integrated into the interface to provide a visually pleasing and sophisticated typographic display.
- The ==color scheme== was chosen to provide a nice visual contrast and ease of navigation. It is primarily defined by a palette of dark tones for text *#333* which corresponds to an equal intensity of red, green, and blue at about 20% of their full capacity, resulting in a dark gray color.
- The ==input and output panels== were arranged with ease using the `sidebarLayout` function, which was used to optimize the layout structure for user interaction. A well-designed text input box for entering keywords and an action button with the phrase *Get Recommendations* for simple navigation were features of the `sidebarPanel` , which contained user inputs. The result was effectively shown by the `mainPanel` , which included a tidy and well-organized table structure for the books that were suggested.
- In order to improve visual appeal and user engagement, the UI components were organized and styled using CSS features like *padding, border-radius, and color schemes*. These design components were used to create a user interface (UI) for the Shiny app that was intended to make book suggestions easy to navigate, readable, and engaging for users.

Server:

- The server-side logic for the Book Recommender app was developed by building a dependable and adaptable system for screening books according to user input. The reactive logic for the same was created to respond to user-entered keywords in a dynamic way, offering a personalized selection of highly recommended books or appropriate statements if no matches were discovered.
- The development of specific functions entrusted with filtering the dataset according to user-inputted keywords was a crucial component of the server-side logic. These thoughtfully developed routines scanned the dataset to find and extract books that closely matched the supplied keyword or keywords.
- By using lowercase conversions and string-matching algorithms, the functions found suitable titles, authors, or content descriptions that matched the user's input. *Lowercase conversion* is a common text processing technique used to normalize text data, the function `tolower()` is used in the project for this purpose. All characters in a string are converted to lowercase, this especially helps in ensuring consistent comparisons.
- The occurrences of a particular pattern or sequence of characters within a larger string or dataset are identified by *string-matching algorithms*.The `grep()` is a versatile function used for pattern matching within strings or vectors. The function showcases immaculate flexibility in handling patterns and extracting

information from strings or vectors which ultimately makes it an indispensable tool for data analysis and manipulation.

- The algorithm performs a thorough sorting process and a list of the best-rated books from the filtered dataset after a successful match of keywords is displayed. Books were ranked in order of preference by the sorting process, which made sure that the top ten highly rated books were consistently recommended.
- When the keywords submitted by the user did not match any entries in the dataset, the server-side logic was effectively developed to respond with relevant messages. The user is appropriately informed by the notifications that no books match the keywords submitted. This strategy was designed to be user-friendly, ensuring clarity and assisting users in fine-tuning their search parameters.

The server-side logic can be considered as the backbone of the book recommender app organizing flawless interactions, dynamic data processing, and the transmission of tailored book suggestions or relevant messages based on user input.

IV. EXPLANATION OF THE CODE

The `library(shiny)` statement initiates the utilization of the Shiny package in R. Subsequently, the `read.csv()` function is employed to load the *book_dataset_clean.csv* file which is the name of the dataset used in the project that consists of columns such as *titles, authors, ISBNs, and ratings*.

UI:

Once the dataset has been loaded, within the ui definition, the `tags$head()` section, along with `tags$style()` and `HTML()` , enables the incorporation of custom CSS styling for the app's user interface. - The `body` section of the of code customizes the appearance and layout of the app by embedding CSS properties and rules. - The `background-image` is used to set an image as the background for the entire app. The URL specified within this line of code links to an image. - Background Attributes such as `background-size: cover` is used to ensure that the image covers the entire background, `background-repeat: no-repeat` is used to prevent the image from repeating, and `background-attachment: fixed` is used to fix the background image in place while scrolling. - The `font-family` is used to define the fonts used. The font *Georgia* is used as the primary font and *Lato* is used as the fallback font. Additionally, color *#333* is implemented to establish a dark shade for text elements effectively enhancing readability.

The UI styling portion mainly consists of 9 elements, - the `.title` being the first one is a CSS class that centers the text and sets the font size to 36 pixels with a bottom margin of 20 pixels. This can be applied to titles or headers in the app for consistent styling. - The second element `.sidebar` is used to define the style for the sidebar, including padding, border radius, maximum width, and centering it inside the app's layout. The background color is set to rgba(255, 255, 255, 0.8) with a small transparency. - Third element `.main-panel` customizes the look of the main panel by using the sidebar's background color, padding, and border radius. - Fourth element `#keywordInput` is used to set the width, padding, margin, border radius, and appearance of the keyword text input box. - The fifth and sixth CSS elements respectively, `#submit` is used to style the `Get Recommendations` button by giving it the dimensions, padding, border radius, text color, background color, and click-through capability and `#submit:hover` determines how the button will appear when it is hovered over, altering the cursor style and background color. The CSS rules of `table` , `th` and `td` specify the width, text alignment, padding, border collapse, and bottom border for table rows, as well as the style and layout for tables and their cells. The structural layout of the Book recommender app's user interface (UI) is employed by 4 main Shiny functions. The application's title panel is created by using the `titlePanel` (*Book Recommender*) function. The panel functions as a prominent header, displaying the app's title and giving users a clear idea of its role. - The sidebar and main panel are the two separate areas of the application layout that are defined by the `sidebarLayout()` method and it constructively improves the app's usability and organization. The `sidebarPanel()` method creates a sidebar panel inside the `sidebarLayout`. Interactive elements that enable users to enter data or start activities are contained within the panel. A text input box `textInput()` is present where users can enter keywords and the action button `actionButton()` labeled *Get Recommendations* initiates the search for book recommendations. The `mainPanel()` function is used to build the main portion of the application interface, in contrast to the sidebar.It usually shows the conclusions or consequences that occur from user inputs or actions. `bookList` a tableOutput() element that is used in the main panel, is intended to display the suggested books in a tabular manner.

Server:

Server-Side Processing is the backend logic of the Book recommender app. The server component is essential for handling user-inputted data and for performing reactive logic. The server-side section covers all of the necessary features, with a particular emphasis on leveraging user-entered keywords to filter data and improve book suggestions.

- The heart of the server logic resides within the `filtered_books` reactive function. The function dynamically filters the dataset and responds to the user input in real-time. As a user enters keywords into the app's text input, a process to match and retrieve relevant books from the loaded dataset is initiated.
- Transforming user input into lowercase letters is an essential component built into the filtering method. By transforming the input to lowercase, the app ensures a case-insensitive matching approach. The method improves search accuracy and user experience by allowing the program to get results regardless of the input's case. The `filtered_books` reactive function proceeds by using the `grep()` function to search for keywords in the dataset's *contents* column without regard to case. One of the important feature of the `grep()` function is its ability to quickly search over character vectors and find patterns.
- Users may extract certain pieces or filter datasets according to predefined criteria, facilitating pattern matching with the help of this function. By providing a pattern to match and a target vector, `grep()` swiftly scans the elements, returning the indices of matches that have been found. `grep()` has commendable flexibility that allows customization with parameters for case sensitivity, its utility is also enhanced in various data-related tasks. The function `grep()` plays a crucial role in data manipulation and pattern discovery, greatly enhancing the analytical capabilities of R programming.
- The algorithm subsequently concentrates on data preparation and purification following keyword matching. In order to handle any data discrepancies, empty strings are converted to NA values in the *Author* field. Furthermore, the `complete.cases()` method is used to filter out rows that include crucial information in columns like *Title, Author, ISBN* and *Rating* that is either missing or incomplete. By ensuring that the dataset contains accurate and pertinent information, this phase improves the accuracy of subsequent analyses.
- After the dataset has been refined, the logic determines if there are matching books available based on the user's query. The code sorts the books by their ratings (*Rating* column) if the `matching_books` subset has entries (i.e., `nrow(matching_books)` > 0). The top 10 suggestions are then obtained by extracting the highly rated books using the `head()` method. Conversely, if no matches are found, a short message, *No matching books found for the entered keyword* is generated within the data frame structure and returned. Ultimately a more engaging and satisfying user experience is ensured with the help of backend logic.The function `output$bookList` specifies in what manner the Shiny app's `bookList` output element will be rendered. It specifies how the output will be displayed on the user interface.
- The `req(input$submit)` statement ensures that the code within the block executes only when the user clicks the *Get Recommendations* button `(input$submit)`. It helps avoid unnecessary computations before the user triggers the action. The filtered dataset is retrieved by `filtered_data` it contains either the top-rated books based on user-entered keywords or a message if no matches were found. The data is obtained by processing the user's input from the `filtered_books()` reactive expression. Conditional Rendering is performed by `if (is.null(filtered_data))`, it is helpful in verifying if any filtered data is accessible. It

returns NULL, indicating no content to display in this particular case. The `else if` condition detects if the filtered data contains a column named *message*. If the column is present it is implied that no matching books were found based on the user query. It then formats and returns a data frame specifically structured to display the message to the user. The `else` condition chooses the particular columns *ISBN, Title, Author, Rating* from the filtered dataset `filtered_data` . It then prepares and returns a subset of the data containing information about the top-rated books matching the user's query.

User interface (UI) and *server-side logic (server)* components of Shiny are combined by the `shinyApp()` function, which acts as the binding agent. It acts as the entry point to launch the application, encapsulating the defined UI layout and the server's reactive behavior. The layout structure, which specifies the web application's visual components, input widgets, and general design, is stored in the UI parameter. Concurrently, the server parameter contains the reactive logic that controls user interactions, data processing, and the dynamic updating of the user interface in response to input from the user. These elements work together to create a responsive and interactive web application that satisfies the requirements outlined in the UI and server code sections. A seamless interaction between the UI and server is put together by the `shinyApp()` function.

V. TESTING AND VERIFICATION

Thorough testing and validation are essential foundations that guarantee the resilience and dependability of the features that have been implemented. The UI development phase for the Book Recommender application involved an extensive search for the most suitable background image that resonated with the app's theme and purpose. Multiple rounds were carried out, examining many images in order to choose the one that best captured the spirit of the application—a procedure that included assessing a multitude of choices until the ideal background was found.

Furthermore, much care was taken to refine font styles and colors, with various combinations examined to determine those that blended in with the general layout and design scheme. The evaluation also included experimenting with padding and alignments, precisely adjusting these components to produce an aesthetic that guaranteed a consistent and engaging user interface experience. Through iterative testing and adjustments, the UI elements were carefully crafted to create a captivating and user-friendly interface within the Book Recommender app.

During the server-side development stage, the filtering mechanism's effectiveness was tested to make sure it worked. This involved conducting searches with different terms in order to determine how accurate the filtering procedure was. Testing revealed that several rows in the dataset had missing values in the important columns of `Title` , `Author` , `ISBN` , and `Rating` . In order to address this problem, a methodical cleaning procedure was carried out, eliminating rows in these crucial columns that included NA or empty values. But closer inspection showed that there were several books that had no author information at all, leaving empty strings in the `Author` column. To address this, a specific line of code was introduced to convert these empty strings to NA values, ensuring data consistency and completeness within the Book Recommender application. These iterative steps in refining the server-side logic were instrumental in enhancing the reliability and completeness of the data used for book recommendations.

VI. SCOPE FOR IMPROVEMENTS

> There are several ways to improve the Book Recommender app. The accuracy of recommendations for books might be greatly increased by improving the recommendation algorithm to use more complex techniques like *collaborative filtering or machine learning models*. These models have the potential to produce more relevant and customized suggestions by utilizing user behavior, preferences, and book features.

Including more elements for customization and user involvement may increase user engagement. Having the *ability to rate books, mark favorites, or leave comments* might improve the user experience and better personalize suggestions. Adding information from other sources or growing the dataset could increase the selection of books that can be suggested. A recommendation system that incorporates data from other APIs or various book databases, such as genres, topics, or user reviews, might be more varied and comprehensive. It is essential to continuously improve the user interface and overall experience. An intuitive and visually beautiful application may be achieved by honing the UI components further, experimenting with layouts, colors, and fonts, and doing user testing to get feedback.

> Enhancing the *scalability* of the codebase and deploying the application on cloud platforms could make it more widely accessible and accommodate a larger user base.

Ensuring robustness and performance as the user traffic grows is essential for a successful deployment. Ongoing testing and validation are crucial for maintaining the application’s integrity. Potential problems and flaws may be found and fixed with the use of extended test cases, edge case scenarios, and frequent code reviews.

The application’s reach may be increased and its inclusivity for a varied user base can be strengthened by taking *localization* into account by supporting numerous languages and guaranteeing compliance with accessibility requirements. The Book Recommender project may develop into a more comprehensive and user-centric application, offering enhanced recommendations and an improved user experience. Its continuous success and expansion are largely dependent on its ability to iterate and adapt in response to user input and technical improvements.

VII. CONCLUSION

The Book Recommender project is an exciting solution designed for those interested in data science books, offering personalized and insightful book recommendations within this specialized domain. Through diligent data cleaning, thoughtful UI/UX design, and robust server-side logic, this application offers a user-friendly interface and accurate data science book suggestions based on keyword inputs.

Although the current version achieves its primary objective, there are still opportunities for more development and optimization. The process of creating this application demonstrated the value of iterative development, user-centric design, and high-quality data. The project has room to grow going ahead, with opportunities to deploy more sophisticated recommendation algorithms, broaden the scope of data sources, and improve user interfaces.

The Book Recommender project lays the groundwork for further advancements that will enable it to provide users with personalized recommendations for stimulating reading.

APPENDIX

REFERENCES

1. [Kaggle Website] (https://www.kaggle.com/datasets/die9origephit/amazon-data-science-books (https://www.kaggle.com/datasets/die9origephit/amazon-data-science-books))

2. [Shiny Website] (https://shiny.posit.co/r/getstarted/shiny-basics/lesson1/ (https://shiny.posit.co/r/getstarted/shiny-basics/lesson1/))

3. [How to use CSS in R Shiny] (https://appsilon.com/howto-css-and-shiny/ (https://appsilon.com/howto-css-and-shiny/))

4. [Wallpaper Link] (https://www.wallpapertip.com/ (https://www.wallpapertip.com/))

5. [CSS Color Schemes Website] (https://palettes.shecodes.io/ (https://palettes.shecodes.io/))

6. [Font Famliy] (https://developer.mozilla.org/en-US/docs/Web/CSS/font-family (https://developer.mozilla.org/en-US/docs/Web/CSS/font-family))

7. [GitHUb Link] (https://github.com/Dhrish96/Scientific-Computation-Project/tree/main (https://github.com/Dhrish96/Scientific-Computation-Project/tree/main))

IMAGES OF THE APP

1. [Initial Screenshot] (https://github.com/Dhrish96/Scientific-Computation-Project/blob/main/BookRApp%201-Initial.png (https://github.com/Dhrish96/Scientific-Computation-Project/blob/main/BookRApp%201-Initial.png))

2. [Example Screenshot] (https://github.com/Dhrish96/Scientific-Computation-Project/blob/main/Example%20Screenshot.png (https://github.com/Dhrish96/Scientific-Computation-Project/blob/main/Example%20Screenshot.png))

3. [Message Displayed Screeshot] (https://github.com/Dhrish96/Scientific-Computation-Project/blob/main/Message%20Displayed%20Screenshot.png (https://github.com/Dhrish96/Scientific-Computation-Project/blob/main/Message%20Displayed%20Screenshot.png))

CODE

```
# Load the Shiny library
library(shiny)
```

```
## Warning: package 'shiny' was built under R version 4.3.2
```

```
# Load the book_dataset_cleanv.csv file
local_csv <- read.csv("C:/Users/dhris/OneDrive/Desktop/my_app/book_dataset_cleanv.csv")

# Define the user interface layout
ui <- fluidPage(
  tags$head(         # Define the head section of the HTML
    tags$style(      # Define the style section within the head
      HTML(          # Use HTML to define CSS styles
        "
        body {
          background-image: url('https://www.wallpapertip.com/wmimgs/9-97960_26102-title-room-of
-knowledge-man-made-book.jpg');
          background-size: cover;
          background-repeat: no-repeat;
          background-attachment: fixed;
          font-family: 'Georgia', Lato;
          color: #333;
        }
        /* CSS styles for different elements in the app */
        .title {
          text-align: center;
          font-size: 36px;
          margin-bottom: 20px;
        }
        /* Sidebar styles */
        .sidebar {
          background-color: rgba(255, 255, 255, 0.8);
          padding: 20px;
          border-radius: 10px;
          max-width: 600px;
          margin: 0 auto;

        }
        /* Main panel styles */
        .main-panel {
          background-color: rgba(255, 255, 255, 0.8);
          padding: 20px;
          border-radius: 10px;
        }
        /* Keyword input styles */
        #keywordInput {
          width: 100%;
          padding: 8px;
          margin-bottom: 10px;
          border-radius: 5px;
          border: 1px solid #ccc;
        }
        /* Submit button styles */
        #submit {
          width: 100%;
          padding: 8px;
          border-radius: 5px;
          background-color: #A0522D;
          color: white;
          border: none;
        }
        #submit:hover {
          background-color: #8B4513;
          cursor: pointer;
        }
        /* Table styles */
        table {
          width: 100%;
          border-collapse: collapse;
          margin-top: 20px;
        }
        th, td {
          padding: 8px;
          text-align: left;
          border-bottom: 1px solid #ddd;
        }
        "
      )
    )
  ),
  # Display the title panel with text
  titlePanel("Book Recommender"),
  sidebarLayout(                      # Define a layout with sidebar and main panel
    sidebarPanel(                     # Define the sidebar panel
      div(class = "sidebar",          # Create a div with the class "sidebar"
          textInput("keywordInput", "Enter keyword:"),   # Input field to enter a keyword
          actionButton("submit", "Get Recommendations")  # Button to trigger book recommendation
s
      )
    ),
    # Define the main panel
    mainPanel(
      div(class = "main-panel",       # Create div with the class "main-panel"
          tableOutput("bookList")     # Display the output table for book recommendations
      )
    )
  )
)
# Define the server logic
server <- function(input, output) {
  filtered_books <- reactive({      # Create a reactive expression to filter books based on keyw
ord input
    keyword <- input$keywordInput   # Get the keyword entered by the user

    # Checking if the keyword is not empty
    if (nchar(keyword) > 0) {
      keyword <- tolower(keyword)   # Convert the keyword to lowercase

      # search for books that contain the specific keyword and store in the 'matching_books' var
```

```
iable
      matching_books <- local_csv[grep(keyword, tolower(local_csv$contents)), ]

      # Convert empty strings in 'Author' column to NA
      matching_books$Author[matching_books$Author == ""] <- NA

      # Remove rows with NA or empty values in specified columns
      matching_books <- matching_books[complete.cases(matching_books$Title, matching_books$Autho
r, matching_books$ISBN, matching_books$Rating), ]

      # Checking if there are matching books after filtering
      if (nrow(matching_books) > 0) {
        # Sort by 'Rating' column to get top-rated books
        top_books <- matching_books[order(matching_books$Rating, decreasing = TRUE), ]

        # Return the top 10 books
        return(head(top_books, 10))
      } else {
        # Return a message if no matching books found
        return(data.frame(message = "No matching books found for the entered keyword."))
      }
    } else {
      return(NULL)     # Return NULL if keyword is empty
    }
  })

  output$bookList <- renderTable({         # Render the table output for book recommendations
    req(input$submit)                      # Ensure that the submit button is pressed
    filtered_data <- req(filtered_books()) # Get the filtered data from the function 'filtered_b
ooks'

    if (is.null(filtered_data)) {
      return(NULL)                         # Return NULL if filtered data is NULL
    } else if ("message" %in% colnames(filtered_data)) {
      return(data.frame(message = filtered_data$message))  # Return a data frame with a message
if no books found
    } else {
      filtered_books_subset <- filtered_data[, c("ISBN","Title", "Author", "Rating")] # Creating
a subset of the filtered data containing specific columns ISBN, Title, Author, Rating
      return(filtered_books_subset)       # Return the subset of filtered data
    }
  })
}
```

Shiny applications not supported in static R Markdown documents