PharmaSUG China 2024 - Paper PO-10026

R Shiny-based Visualization for IDMC

Qiong Wu, HappyLifeTech

ABSTRACT

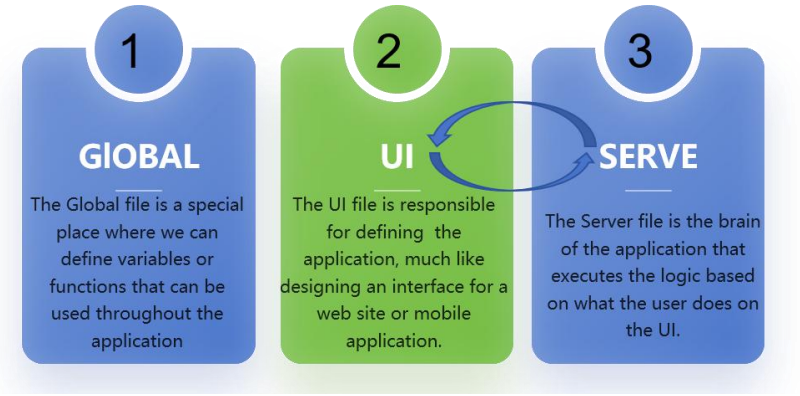
IDMC are responsible for regularly reviewing cumulative data from ongoing clinical trials to protect subject safety, ensure trial reliability, and validate trial results. In order to better support IDMC, this paper visualizes clinical trial data and results using R Shiny, which is an excellent tool for observing data distribution, composition, comparison, and trends due to its ease of use, real-time data updates, variety of chart types, customization, etc. R Shiny consists of two main modules etc. the user interface (ui) and the server logic (server). The ui.r is used to design appearance, interactive elements, and presentation of results, while the server.r responds to user actions and performs calculations. Eventually, the test information is presented in the form of graphs or icons, which transforms boring data into easy-to-understand images and facilitates the extraction of useful information from the data.

INTRODUCTION

IDMC plays a crucial role in drug clinical trials, and its independence, professionalism and standardization are essential to protect the interests of subjects and improve the quality of trials. The data that the IDMC needs to analyze may include safety data (such as adverse event) , validity data (such as prespecified endpoints) , and so on, therefore, how to display the data comprehensively, effectively and intuitively is a topic worth discussing. Now R Shiny provides a powerful data visualization framework for building interactive charts and graphs to help researchers quickly grasp data and monitor research progress and results, such as the patient's condition, treatment effect etc. R Shiny helps IDMC members better understand and interpret data from clinical trials by displaying dynamic changes in data in interactive charts and graphs.

GENERAL FRAMEWORK

The body of the code is divided into three parts: Global. R, UI. R and Serve. R.



- About global.r, you have some code that needs to be used in both the ui and server, so a global file is a good choice. This way, when you need to update this part of the code, you only need to make changes in one place. The global file can also help you avoid repeating the same code across multiple files, increasing development efficiency.
- In the ui, we can add text boxes, sliders, check boxes, and so on, which allow the user to enter data or select different options, and the ui can also display charts, tables, or text, these are generated dynamically based on the user's actions.
- The server file contains all the data processing and analysis code. It can calculate results based on

user input or update the presentation on the ui. Server files can respond to a variety of different user inputs without having to write code repeatedly.

GLOBAL.R

Figure 1. Shows imported packages, external functions, and default datasets.

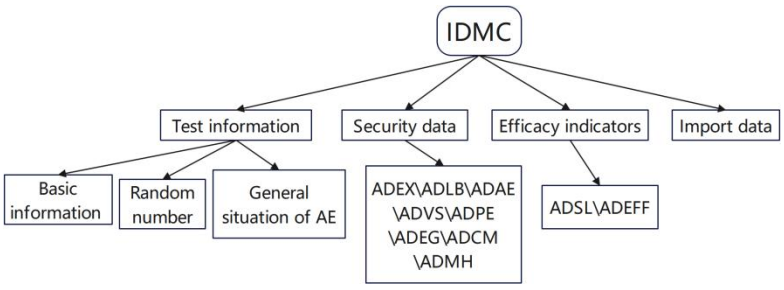
Usually, the data sets are in *.sas7bdat* format. Fortunately, the haven package has facilitated a way to load *.sas7bdat* data in R via the *read_sas()* function. This paper first imports packages and external functions, followed by *list.* file. The *files* function lists the current working directory (. stands for current directory) and all extensions under its subdirectories named.*sas7bdat* file.

You can traverse the index of all filenames in filex through a *for loop*. In the circulation, do.call function is used to call a function dynamically. The function it calls here is the *assignment operation* <-, which means it assigns the expression on the right to the variable on the left.

UI.R

The ui in R Shiny framework is rich, easy to build and customize, highly flexible, dynamically responsive, integrated and extensible, and user-friendly. These advantages make shiny one of the first tools for data scientists and analysts to create interactive data visualization and analytics applications. It is also a powerful tool for creating interactive data visualization and analyzing applications.

Figure 2. An overview of UI



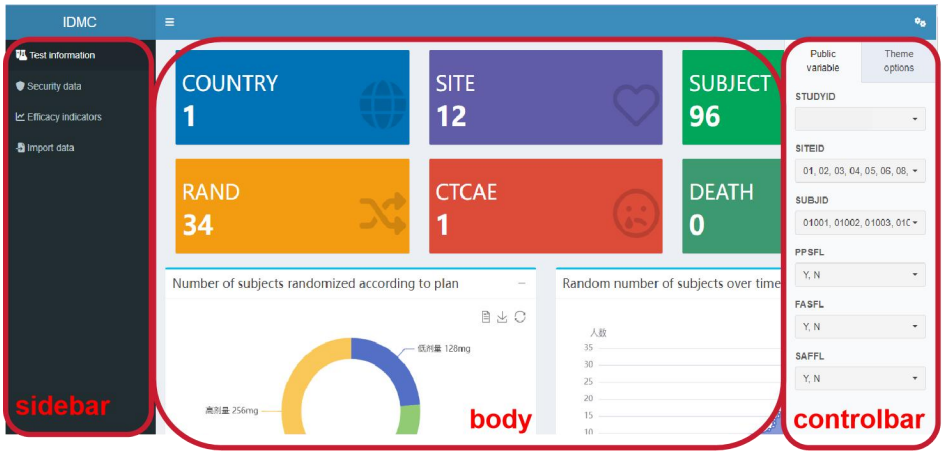
Through the design of ui, we can better display and interpret the data in clinical trials, such as basic trial information, safety results, efficacy indicators etc. Through scatterplot or box-line diagram of the intuitive display, to help researchers quickly grasp the data, monitoring the progress of research and results.

Reference R Shiny clinical prediction model and decision support system to obtain more accurate prediction and more personalized recommendations for IDMC decision-making.

First, Shiny provides a rich set of ui elements, such as hearder, sidebar, controlbar, and so on, that allow programmers to create professional-looking and highly interactive Web applications, and these ui elements can be defined directly in R code without writing additional HTML, CSS, or JavaScript code.

The following illustration shows a simple ui interface.

Figure 3. Display the UI

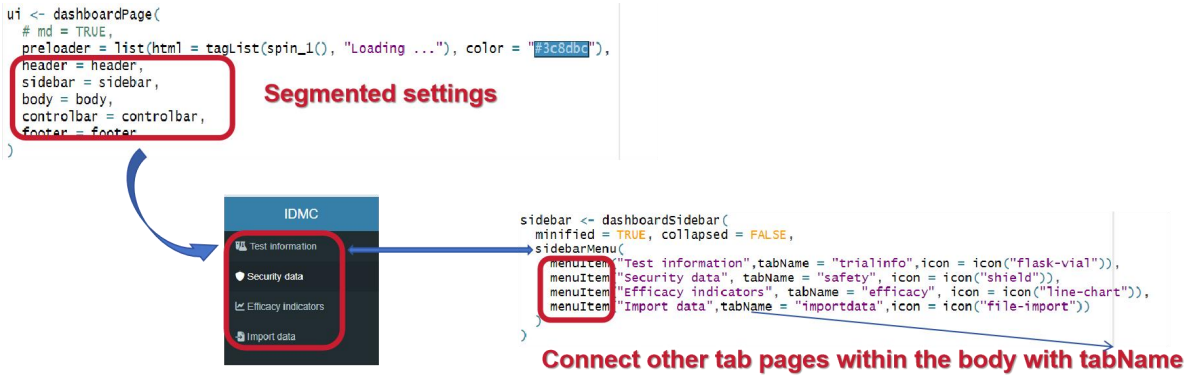


By using shiny ui functions, developers can quickly create various forms of input controls (such as sliders, buttons, drop-down menus, and text boxes) and output functions (such as charts, tables, and text displays) , to capture user input and display data analysis or visualization results. The main body of the ui is set up in chunks, and we use *dashboardpage* to integrate the parts, using themes developed by Google as well as customizations for the design process.

And you can use *tabName* to connect to other tab pages within the body, allowing interaction between different tab pages to accommodate complex data analysis and visualization requirements. A pre-loader can be defined at design time that displays a rotating icon (generated by *spin_1()*) and the text 'Loading...' as the application loads.

The following figure shows some of the code and the results.

Figure 4. UI sidebar



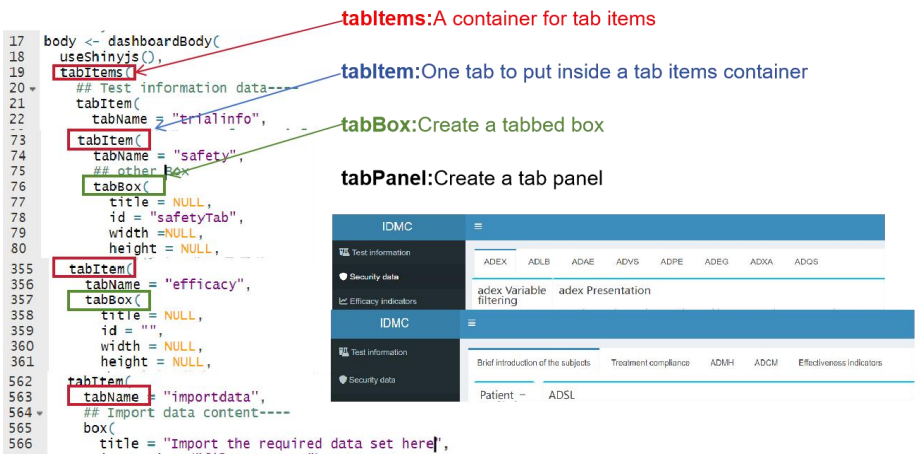
With respect to page layouts, tabpanel are used to create tabbed panels, and often layout functions, such as *fluidrow* and *column*, are used to design rows and columns inside a tabPanel. The width parameter of

the *column* function is used to specify the width of the column. It takes a number between 1 and 12(or between 1/12 and 1) to indicate the number of grid cells occupied by the column.

By default, a page has 12 grid cells, so you can adjust the width parameter to create columns of different widths. You can also nest *fluidrows* and *column* to create more complex layouts.

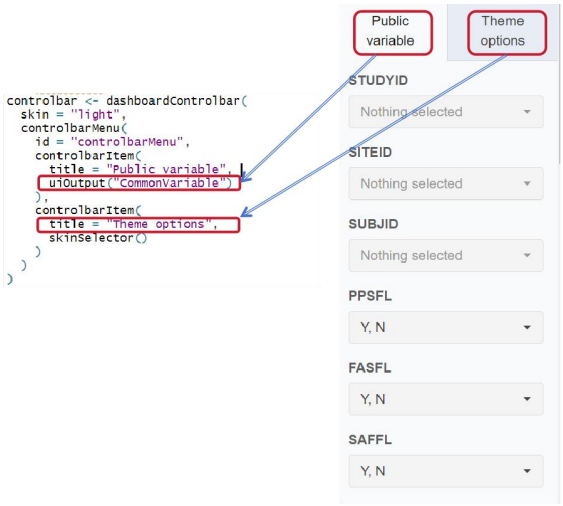
The following figure shows some of the major code and results.

Figure 5. UI body



We also enrich the options by adding a menu, the actions shown below, and the results.

Figure 6. Meau



Even without the server, you can get a nice interface by stacking the ui components, but the problem is that we don't just need a panel to display the results, we also need user interaction, and new parameter values keep coming in, but the ui can not do data analysis, so variable names and feature categories can not be automatically extracted from the default dataset independently, so you need to create output variables to connect the ui and serve.

With the responsive programming model in R shiny, when the input in the ui changes, the server immediately responds and recalculates the output. This model enables Shiny applications to respond to user actions in real time, providing a high degree of interactivity and dynamism. Now let's see how the data analysis is done in server.

SERVE.R

First, Server processes the data. The first step is to upload the data. Serve needs to replace the default data with the newly uploaded data. The default data has already been read into global, and the variable names are filenames such as adae, ADEX etc. Import the dataset; the code is divided into UI and server sections.

The ui part mainly lets the user have a window to upload data and display the files uploaded by the user; the server part renders the table displayed and listens for the uploading of files and updates the data to new data files. The two parts are connected by id.

The following figure shows how to upload data in serve. This involves *reactivevalue*, which can be used to store objects that other expressions can depend on, and is a *reactive* list.

Figure 7. Data upload

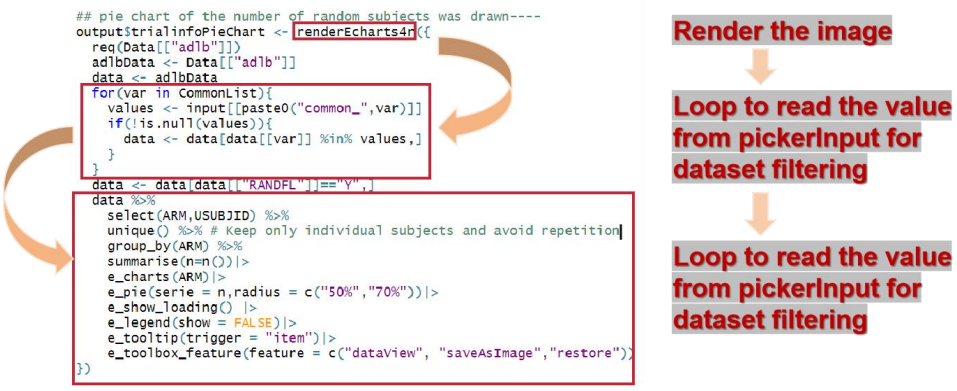
Second, build common variables and theme options for filtering, where you need to loop through the variables you want to filter to generate the *pickerInput* component so that the previous variable change is detected and the options are updated automatically.

Once the filter component is established and the dataset is available, how do you implement the common variable filter to get the filtered data and plot it?

First, you need to loop through the values from the *pickerInput* and filter the data set, then render the image using r*enderEcharts4r*, and finally use the drawing function as normal, you need to make sure that the server part matches the local identifier of the ui output.

The following illustration shows some of the programs that draw on filtered data.

Figure 8. Based on filtered data and drawing



ui functions typically contain a series of Shiny ui components, such as imgInput, sliderInput, plotOutput, and so on. These components are built using R's expression syntax and the ui functions provided by Shiny. However, in more complex ui components, you may need to dynamically generate ui elements based on server-side logic.

In this case, you can use the *uiOutput* function to define a UI placeholder and the *renderUI* function in the server function to dynamically generate the ui element. The *renderXX* function is commonly used in server functions to render text, tables, images, and so on.

For example, use the *renderDataTable* function to render a table that can be displayed in the ui using placeholders defined by the *dataTableOutput* function. The *renderDataTable* function allows you to display data frames or other tabular data structures in R to the user as interactive HTML tables.

The following table lists some common ui output functions and *render* functions in serve.

Table 1. Data upload

Output function	Output object	render	packages
dataTableOutput	Table	renderDataTable	shiny
htmlOutput	Website	renderUI	shiny
imgOutput	Picture	renderImage	shiny
plotOutput	Drawing	renderPlot	shiny
tableOutput	Form	renderTable	shiny
textOutput	Text	renderText	shiny
uiOutput	Website	renderUI	shiny
verbatimTextOutput	Text	renderPrint	shiny
plotlyOutput	Interaction diagram	renderPlotly	plotly

Do you have to rewrite the filter loop every time you draw or present data, and how do you ensure that the results of variable filtering are reused?

You Don't have to recalculate it in every *render* * function when you want to be able to use an interactive variable controlled by input multiple times in the server function. You can use the reactive function instead, defines an interactive variable that caches values and knows when to update them. It is the response expression in Shiny to prevent Shiny from rerunning unnecessary code, making the application faster.

The first time a response expression is run, the expression saves its results to the computer's memory, and the next time a response expression is called, it checks whether the saved value has been updated (that is, whether the widget it depends on has changed) , and if the value has not been updated, the previously saved value can be called without any calculation.

if it has been updated, when using a response expression, we need to enclose its symbol with parentheses and treat it like a function. For example, data <-*reactive ({ XXX })* , which is subsequently used using *data ()* .

RESPONSIVE PROGRAMMING

Responsive programming-we express high-level goals or describe constraints, and then rely on others to decide how and when to translate them into action. This is the programming style we use in Shiny. In Shiny, we express the service logic using responsive programming.

The core idea is to specify a dependency graph so that when input changes, all related outputs are automatically updated. This makes the process of writing Shiny applications fairly simple, but it takes some time to understand how they fit together.

- Inertia

One advantage of responsive programming in Shiny is that it allows applications to be very lazy. The Shiny application does as little work as possible to complete the required updates to the resulting control.

- Response diagram

Shiny inertia has another important property. In most R code you can read the code from beginning to end to understand the sequence of program execution. This is not useful in Shiny, however, because Shiny runs on demand. That is, the order in which responsive code executes is determined by the response diagram, not by the order in which it is placed. You can use the *reactlog* package to draw a response diagram.

- Response expression

An important component of the response diagram is the response expression, which we can now think of as a tool to reduce code duplication. For example, we saw reactive earlier.

CONCLUSION

- In summary, the ui and server play different roles in the R Shiny framework but are closely related. The ui interacts with the user and provides the presentation, while the server processes the input and generates the output. With the responsive programming model, the ui and the server can respond to each other's changes in real time, allowing for a high degree of interactivity and dynamism.
- R Shiny offers a wealth of ui controls and layout options that customize the look and function to the needs of the IDMC to provide faster insight into the data. And R Shiny can connect to the data source in real time, ensuring that the data is up to date and accurate.

  Applications can automatically update charts and results when the data in the data source changes. R Shiny supports a variety of data sources (such as databases, CSV files, apis, and so on) for the unified management and analysis of data.
- Using R Shiny to power IDMC visualization brings many benefits, including interactivity, ease of use and sharing, customization, real-time data updates, team collaboration, maintainability, and extensibility. These benefits will help improve the efficiency and quality of data analysis and visualization for effective management and decision-making in IDMC.

REFERENCES

Awesome RShiny Project

https://gitcode.com/grabear/awesome-rshiny

Visualizing metabolomics data with R

https://pubmed.ncbi.nlm.nih.gov/36373190/

https://shiny.rstudio.com/tutorial/

https://mastering-shiny.org/index.html

Mastering Shiny for R Web Development

ACKNOWLEDGMENTS

Thanks for all the help from HLT statistical programming department, and look forward to working together in the future to develop efficient tools to aid the pharmaceutical industry.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Qiong Wu
HappyLifeTech
E-mail:qiong.wu01@hlifetech.com