PharmaSUG China 2025 – Paper DV-185

Advancing Biostatistical Workflows: R-Based Interactive Visualization for Tables, Listings, and Automated Code Generation

Zichuan TAN, Sanofi

ABSTRACT

In today's pharmaceutical industry, data visualization has become increasingly critical. Departments such as sales have adopted commercial tools like Tableau and Power-BI to create interactive dashboards, enabling decision-makers to access intuitive analytics and improving managerial efficiency. However, in biostatistics, visualization lacks consensus—both in methods and content. This discrepancy stems from two key factors: first, statistical programmers have traditionally handled most analytical coding, fostering a path dependency on this linear workflow; second, legacy platforms like SAS, which dominated the field, lacked essential UI design capabilities, rendering visualization impractical.

The rise of R as a statistical programming language offers new opportunities. Its open-source nature allows for flexible visualization solutions. This paper introduces an innovative approach that leverages RShiny, rlang, and other tidyverse packages to develop a Shiny app capable of generating visualized, interactive Table & Listing outputs alongside automated R code. The design of the app emphasizes user-friendly parameter customization and dynamic data presentation, addressing gaps left by conventional tools. the paper also presents the core principles of the design, optimizations for user interaction, and future prospects.

By embedding visualization and interactivity into biostatistical workflows, this framework aims to enhance reproducibility, minimize manual coding efforts, and facilitate adaptive analytics in clinical research. R-based visualization app may addresses long-standing limitations, offering a scalable and flexible solution for modern statistical programming challenges.

INTRODUCTION

Statistical analysis has long relied on traditional tools like SAS for generating static outputs such as tables and listings. However, these methods often lack flexibility in visualizing complex datasets, particularly in clinical trials where dynamic data interpretation is critical. The limitations of legacy platforms—such as their inability to support interactive dashboards or real-time parameter adjustments—have hindered the adoption of modern visualization techniques in biostatistical workflows. Recent advancements in open-source programming languages, particularly R, have introduced new possibilities for integrating interactivity and visualization into statistical programming. Packages like RShiny, rlang, and tidyverse enable the creation of customizable, user-driven interfaces that streamline data exploration while maintaining reproducibility [1].

A key challenge in biostatistics lies in translating static analytical results into actionable insights for stakeholders, including clinicians and regulatory bodies. Traditional workflows often require manual coding for each adjustment, increasing the risk of errors and reducing efficiency. To address this, innovative frameworks are emerging that combine automated code generation with interactive visualization. For instance, RShiny-based applications allow users to modify parameters dynamically—such as adjusting subgroup analyses or filtering datasets—without requiring deep programming expertise.

The integration of visualization and interactivity into statistical workflows is particularly relevant in clinical research, where decisions often depend on nuanced interpretations of efficacy and safety data. For instance, in pharmacokinetic/pharmacodynamic (PK/PD) modeling, interactive visualizations can help clarify dose-response relationships by enabling real-time adjustments to model parameters.

This paper explores a novel framework that leverages R’s ecosystem to create an interactive Shiny app for generating Table & Listing outputs alongside visualized data. The design emphasizes user-friendly customization, automated code generation, and seamless integration with existing statistical standards. The discussion includes core principles of the framework, optimizations for usability, and potential applicati ons in clinical trials. By addressing the limitations of conventional tools, this approach aims to

redefine how biostatisticians collaborate with cross-functional teams, ultimately improving decision-making in drug development.

METHODOLOGY AND IMPLEMENTATION

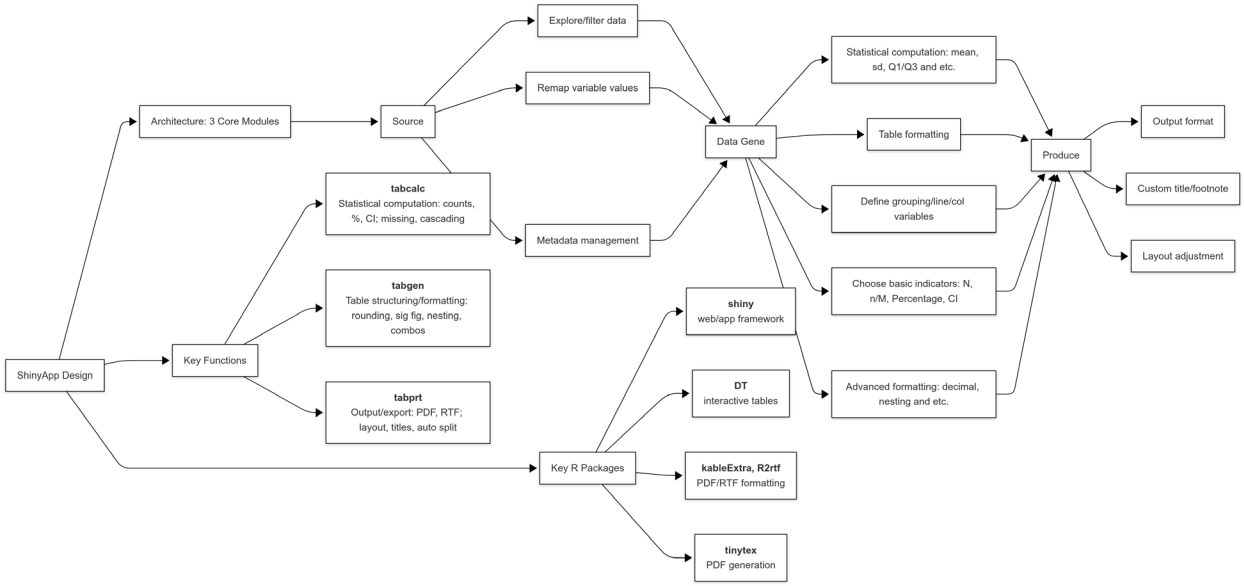

Figure 1. App design process flow

The Shiny application is designed to streamline the generation and customization of statistical tables from clinical data. Its architecture is built around three core modules: Source, Data Gene, and Produce, each serving distinct yet interconnected functionalities. The Source module provides an interactive interface for data exploration, allowing users to view and filter datasets, apply custom remapping of variable values, and manage metadata. The Data Gene module acts as the central engine for statistical computation and table formatting, enabling users to define grouping variables, line variables, column variables, and statistical indicators (e.g., counts, means, confidence intervals) while applying advanced formatting options such as decimal precision control, indicator combinations, and hierarchical nesting. The Produce module finalizes the output by generating publication-ready tables in formats like PDF and RTF, with options for custom titles, footnotes, and layout adjustments.

The functionality is powered by a suite of R packages that facilitate data manipulation, visualization, and output generation. At its core, Shiny provides the web-based framework, enabling dynamic user interactions and real-time updates to the application interface. DT (DataTables) is used to render interactive, paginated tables in the Source module, allowing users to explore large datasets with ease. KableExtra and R2rtf play critical roles in formatting tables for LaTeX and RTF outputs, while Tinytex facilitates PDF generation via LaTeX compilation. Additionally, there are Dplyr and Tidyr for efficient data processing, Stringr for string manipulation, particularly in remapping operations and Rlang for dynamic evaluation of user inputs.

Apart from R packages, The application's workflow is orchestrated by three key encapsulated custom functions: tabcalc, tabgen, and tabprt. Tabcalc serves as the computational backbone, calculating statistical metrics (e.g., counts, percentages, confidence intervals) based on user-defined groupings, line variables, and conditions. It dynamically adjusts for missing data and applies cascading counting policies to ensure accurate denominator calculations for percentages and confidence intervals. Tabgen transforms the raw statistical output into a structured, user-customizable table. It applies formatting rules such as decimal rounding, significant figures, and indicator combinations (e.g., "n(%)", "(95% CI)"). This function also supports compact hierarchical displays for nested groupings, ensuring clarity in multi-level categorical data. Tabprt finalizes the output by exporting the formatted table to PDF or RTF files. It integrates user-specified titles, footnotes, and layout settings, while also handling tasks like column width adjustments, paper size configurations, and automatic splitting of wide tables across pages. Together,

these functions form a seamless pipeline from raw data to polished, publication-ready tables, emphasizing both analytical rigor and user-centric design.

By integrating these components, the application offers a cohesive environment for data analysts to perform exploratory analyses, validate statistical assumptions, and produce standardized reports. The modular design ensures scalability, allowing users to adapt workflows to diverse datasets and reporting requirements. Its reliance on R's ecosystem of packages ensures compatibility with existing data science tools while maintaining a high degree of reproducibility. This combination of interactivity, customization, and automation positions it as a versatile solution for statistical reporting in clinical research and beyond.

UI DESIGN & INTERACTION OPTIMIZATION

In the realm of user interface (UI) design and interaction optimization, the Shiny application exemplifies a sophisticated blend of aesthetic appeal and functional efficiency. The UI design leverages several key modules and functions to create an intuitive and responsive interface that enhances user experience. At its core, the application utilizes Shiny's built-in UI components such as fluidPage, sidebarLayout, tabsetPanel, and DT to structure the layout and manage data presentation. These components are essential for organizing content in a way that is both visually appealing and easy to navigate.

The UI design begins with the foundational structure provided by fluidPage, which offers a flexible grid system for arranging elements on the page. This is complemented by sidebarLayout, allowing for a clear separation between navigation controls and main content areas. Within these layouts, tabsetPanel facilitates the organization of different functionalities into distinct tabs, ensuring that users can easily switch between tasks without feeling overwhelmed. For instance, the Source, Data Gene, and Produce tabs each house specific tools and settings relevant to their respective roles within the application. Additionally, DT is used extensively to render interactive tables, enabling users to explore large datasets dynamically.

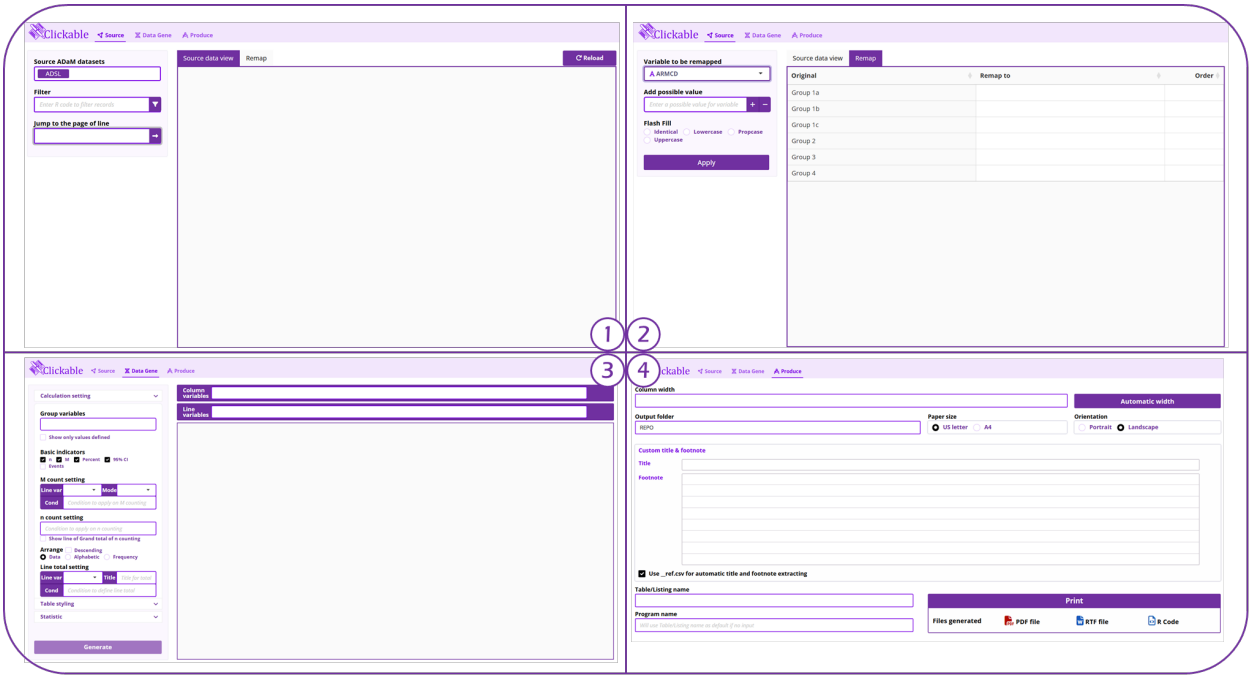

Figure 2. User Interface

To further enhance the user experience, the application incorporates custom render settings and plugins of DT and specialized input handling mechanisms.

Normally, DT package is widely used for rendering interactive tables in Shiny apps, it is typically limited to functioning as a uneditable table viewer. In practice, developers and users often need to make further

modifications to the table after it has been generated, making a purely read-only interface somewhat restrictive. This is where cell editing becomes an essential enhancement. By integrating KeyTable, a plugin for DT, users can directly edit individual cells within the table in real time. These edited table is then sent back to the server, and can be converted into a standard tibble or data frame, which make available for further processing within the R environment.

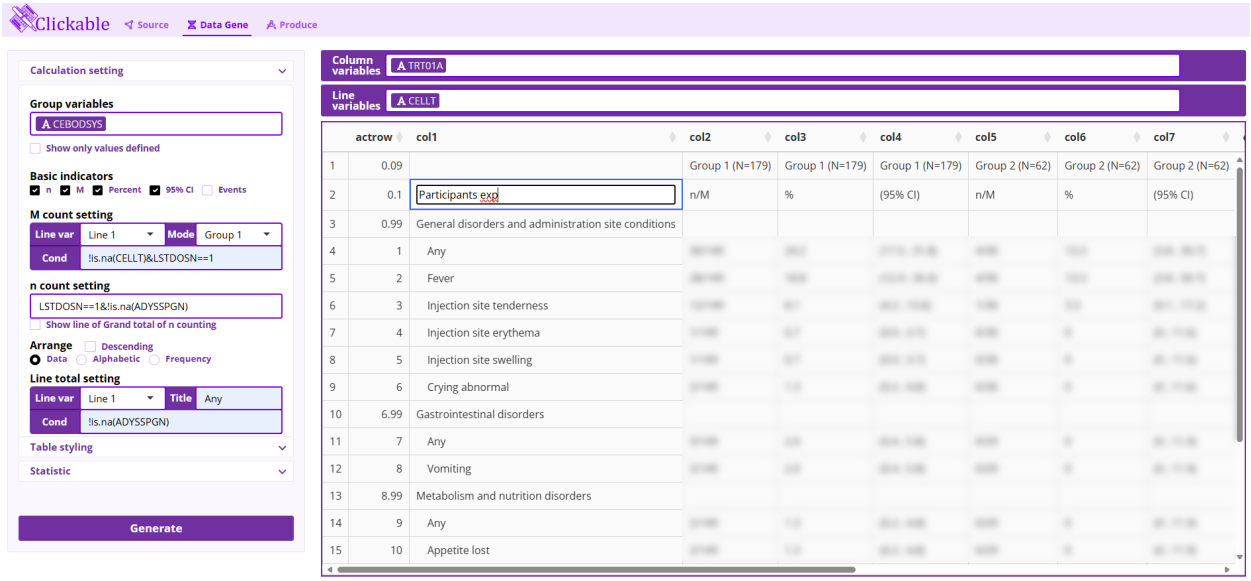

Figure 3. Inputting text into top-left cell of a DT-rendered table

Another critical aspect of the UI design is the use of InputSet_multi and InputSet_single functions, which significantly improve the usability of select inputs. These functions employ JavaScript to customize the appearance and behavior of dropdown menus, making them more informative and easier to navigate. Specifically, InputSet_multi handles multi-select inputs, while InputSet_single manages single-select scenarios. Both functions utilize the selectize.js library, which provides powerful autocomplete and tagging features. Through the use of custom templates and icons, these functions transform plain text options into visually distinctive items, enhancing readability and reducing cognitive load for users.

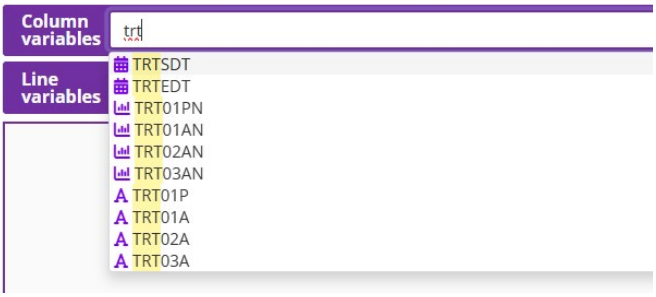

Figure 4. Specific icons are placed before variables to indicate the datatype

Specifically, InputSet_multi dynamically adjusts the display of options based on their type, using different icons to represent character, numeric, date, and remap variables.

This visual differentiation helps users quickly identify the nature of each variable without needing to read through lengthy descriptions. These enhancements not only make the interface more aesthetically pleasing but also streamline the selection process, leading to faster and more accurate data manipulation.

Underpinning these customizations is a robust programming approach that combines R with JavaScript to achieve seamless integration between backend computations and frontend interactions. The use of HTML and CSS within Shiny's tags$style and tags$head elements allows for precise control over the appearance of various UI components. For instance, custom styles applied to buttons, dropdowns, and

tables ensure consistency across the application, creating a cohesive look and feel. Below is an example of defining default and active state appearance for .btn-default buttons, including color, size, and hover effects.

Beyond visual enhancements, effective data presentation also hinges on how variables are represented in tables. A common scenario is that the values (or modalities) listed in the required table, and/or the order in which they appear, may differ from those of the corresponding variable in the dataset. Additionally, it is possible that not all values from the dataset need to be displayed in the table, or conversely, some values required in the table may not yet be present in the dataset (thus programmers may need to “force” the display of that all-zero row). To address this, programmers often need to "remap" the variable to bridge the gap. The “Remap” functionality shown in Figure 5. streamlines the process by automatically listing the distinct available values of the selected variable. Users can easily redefine values or add absent values to create a new variable that fits the intended output—making the task even faster when using "flash fill" for minor adjustments. Besides, users can define a custom display order for each remapped value, which determines how the values are arranged in the final output table.

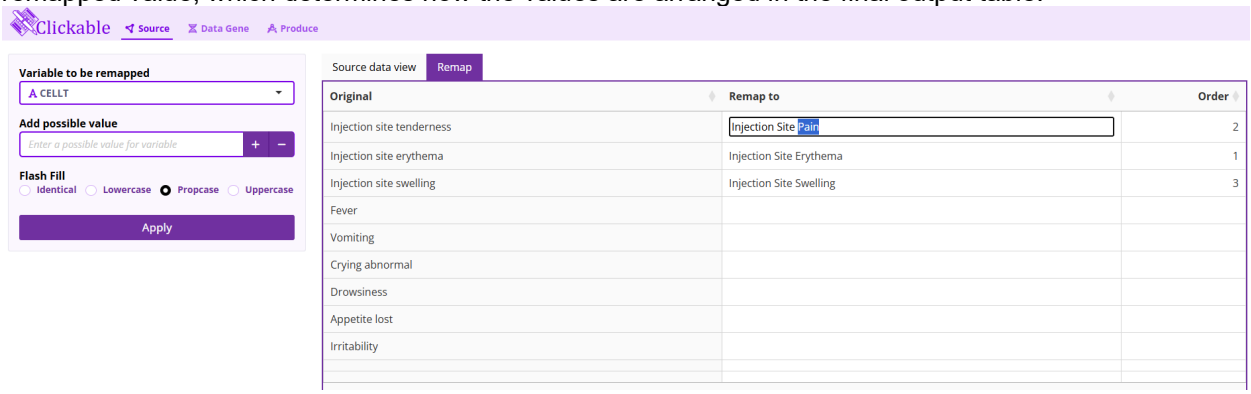

Figure 5. Remap a variable and define value order in “Remap” page

In summary, the UI design and interaction enhancements aim to align with common user needs and expectations. By integrating Shiny’s flexible layout tools with custom JavaScript functions and thoughtful styling, the application strives to provide a more engaging and efficient experience for generating and interacting with statistical tables. Features like editable DT table, InputSet_multi, InputSet_single and “Remap” were developed with the intention of improving usability and clarity, drawing on common programming practices to support smoother user interactions. While there is always room for further improvement, these design choices reflect an ongoing effort to create a more intuitive and functional interface for data exploration and analysis.

AUTOMATED CODE GENERATION

Automated code generation within this Shiny application revolves around systematically translating the functionalities of various modules into executable R scripts. By understanding the functional logic behind each module, it becomes theoretically possible to automate this process. The basic functionality of this Shiny app focuses on data preprocessing, which includes dataset selection, data filtering, and remapping. For dataset selection, the ADSL dataset is included by default and mandatorily, ensuring that essential clinical trial information is always available. Other datasets such as ADAE and ADLB are integrated with ADSL using the left_join function from the dplyr package. This ensures that all relevant datasets are seamlessly merged, forming a comprehensive dataset for further analysis.

Following dataset integration, users can apply custom filters to refine the combined dataset according to specific criteria. These filters are based on user-defined conditions, allowing for targeted subsets of the data to be extracted for more focused analyses. For remap functionality, it allows users to redefine variable values and order of values based on mappings specified in an editable remapping table. This table lists original variable values alongside their corresponding mapped values and order, determining the creation of new variables within the merged dataset.

The simplicity yet effectiveness of implementing these functionalities through R code cannot be overstated. Taking dataset selection as an example, the following R code demonstrates how this can be automated:

Once the data has been preprocessed, it is ready for table generation. The Shiny app employs three encapsulated functions: tabcalc, tabgen, and tabprt. These functions handle calculations, table styling, and pdf/rtf output respectively. Since these functions are already defined, then simply reading the parameters set or input by users via the Shiny interface and inserting these values into the appropriate arguments of the functions would be sufficient for automation.

This process automates the generation of R code tailored to the user's specifications without requiring manual intervention, it enhances reproducibility and supports efficient data analysis, making the workflows more accessible and convincible.

CONCLUSION

This Shinyapp offers a practical solution for automating various data processing and table generation tasks within clinical research. By utilizing R's versatile packages such as dplyr, tidyr, and kableExtra, the app integrates functionalities ranging from initial data preprocessing to final output formatting. The key feature of this application is its ability to generate R scripts automatically based on user inputs through an intuitive Shiny interface. This approach aims to reduce the complexity involved in manual coding and enhance the reproducibility of data analysis processes. Users can select datasets, apply filters, remap variables, and create tables with ease, ensuring that all necessary data transformations are accurately reflected in the generated R code.

Another notable benefit of this approach is its accessibility to users with varying levels of programming experience. Researchers who may not be proficient in R can still perform complex analyses by interacting with user-friendly components. This design helps bridge the gap between advanced data analysis tools and those who need them but might lack extensive programming skills. By automating repetitive tasks such as dataset merging and variable remapping, the app allows researchers to focus more on interpreting results rather than dealing with the intricacies of data manipulation.

Looking forward, there are several potential areas for improvement and expansion. Developing a template system is the key to further improve the efficiency and reduce the operational threshold. Expanding the range of supported statistical methods and enhancing customization options for table outputs could make the tool more adaptable to different research needs. These enhancements aim to address current limitations and cater to evolving research requirements.

REFERENCES

[1] Wickham, H. (2019). Advanced R, Second Edition (2nd ed.). Chapman and Hall/CRC.

ACKNOWLEDGMENTS

Deep gratitude to the vibrant R community and the ever-expanding world of AI.

In a realm where knowledge knows no bounds, it is they who have broken down barriers and opened doors for all who seek to learn and grow.

CONTACT INFORMATION

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

Zichuan(Zack) TAN
Sanofi (China) Investment Co., Ltd.
Zack.TAN@sanofi.com