

## Interactive Clinical Plot Gallery with Cross-Plot Subject Linking in R Shiny

You Lyu (Lea), Johnson & Johnson

### ABSTRACT

Clinical trial data visualization plays a critical role in understanding treatment effects, safety profiles, and individual subject responses. Traditional static reporting tools often lack the interactivity needed for exploratory data analysis during study review. In this paper, we present a user-driven clinical plot gallery built with R Shiny, featuring eleven plot types commonly used in oncology clinical trials: line plots, waterfall plots, scatter plots, box plots, spider plots, swimmer plots, multi-Y plots, tornado plots, bar plots, Kaplan-Meier plots, and forest plots. A key feature of this plot gallery is the cross-plot subject linking system, which allows users to highlight individual subjects across all visualizations simultaneously, enabling seamless exploration of subject-level data. The plot gallery also supports a user-configurable layout where users can select and arrange plots of interest side by side. A pattern-based highlighting system enables users to identify and visually emphasize subgroups of subjects sharing clinically meaningful response patterns across compatible plots. A click-to-explore feature on selected plots triggers a subject profile modal displaying laboratory, adverse event, and best overall response data. A subject journey timeline provides a longitudinal, event-level view of each highlighted subject's clinical course. An integrated statistical options panel allows users to overlay hypothesis test results, effect sizes, and multiple comparison adjustments directly on selected plots. Additionally, a per-plot code editor powered by the shinyAce package allows users to modify, preview, and apply custom plot functions directly within the dashboard. This paper describes the design, implementation, and key technical considerations of building such a user-driven platform using R Shiny and the shinydashboard, plotly, shinyWidgets, and shinyAce packages.

### INTRODUCTION

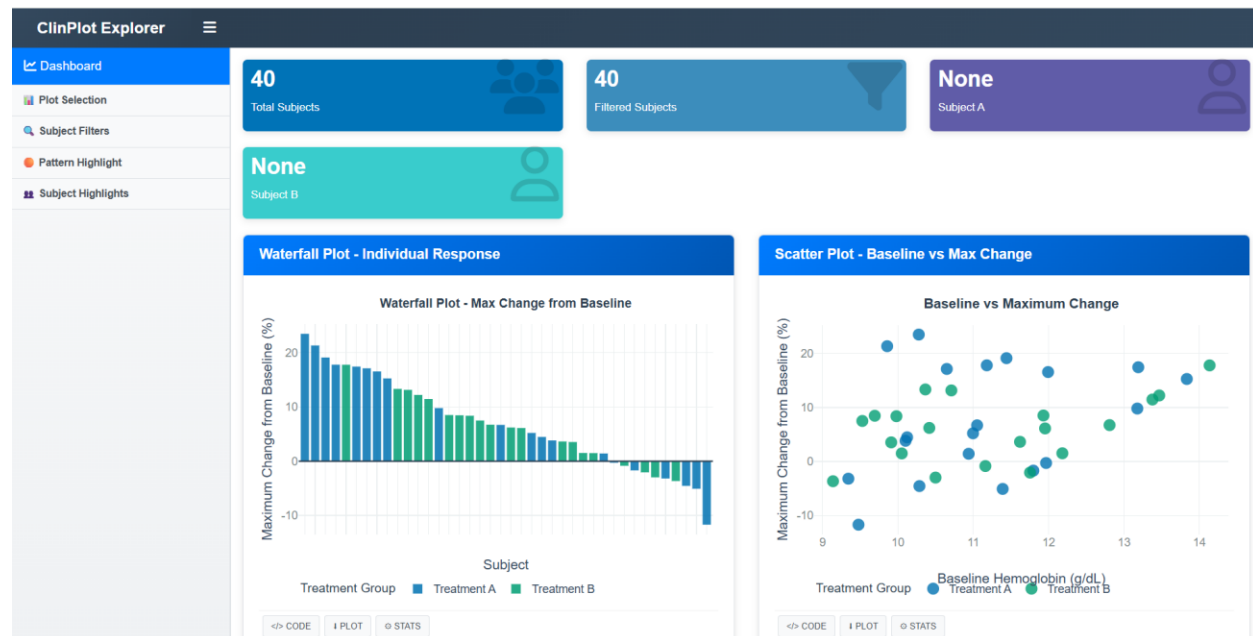
The increasing complexity of oncology clinical trials demands more sophisticated tools for data review and exploration. Regulatory submissions and internal study reviews often rely on a fixed set of static outputs, which may not adequately support the iterative and exploratory nature of clinical data analysis. Interactive visualization tools have the potential to bridge this gap by enabling analysts and clinicians to explore data dynamically, identify patterns, and drill down into individual subject profiles without generating new static outputs for each query.

R Shiny has emerged as a powerful framework for building interactive web applications directly within the R ecosystem, making it accessible to statisticians and programmers already familiar with R. Combined with packages such as plotly for interactive graphics, shinydashboard for structured layouts, shinyWidgets for enhanced UI components, and shinyAce for in-browser code editing, R Shiny provides a comprehensive platform for clinical data visualization.

This paper presents a dashboard designed for oncology clinical trial data review. The application uses simulated datasets following CDISC ADaM conventions, including ADSL, ADHEMA, ADAE, ADTTE, and ADEFF, ensuring that the data structure and variable naming are consistent with industry standards. The plot gallery is designed with four core principles: comprehensiveness of plot types, interactivity at the subject level, flexibility in user-driven layout configuration, and extensibility through in-dashboard code editing.

## DASHBOARD OVERVIEW

The plot gallery is built using the shinydashboard package, which provides a structured three-panel layout consisting of a header, a collapsible sidebar, and a main content area (Display 1). The sidebar contains all user controls organized into collapsible sections using HTML details and summary elements: plot selection, layout options, subject filters, pattern highlighting, and subject highlight selectors. The main content area dynamically renders the selected plots in a user-defined grid layout.



(Display 1. Interface of Plot Gallery)

## PLOT CATEGORIES

The plot gallery supports eleven plot types covering the major visualization needs in oncology clinical trials. Each plot is implemented as a standalone R function that accepts the filtered dataset and optional subject highlight parameters, making the codebase modular and maintainable.

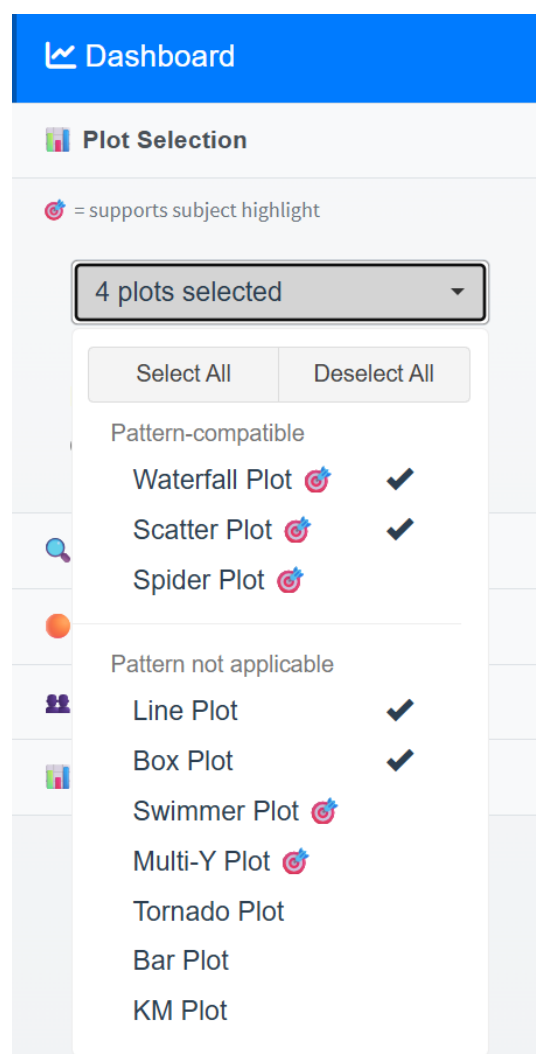
The line plot displays mean biomarker changes over time with standard error bars, stratified by treatment group. The waterfall plot shows maximum percent change from baseline for each subject, sorted in descending order and colored by treatment group. The scatter plot visualizes the relationship between baseline values and maximum change from baseline. The box plot displays the distribution of percent change from baseline at each study visit by treatment group. The spider plot shows individual subject trajectories over time, with reference lines at clinically meaningful thresholds. The swimmer plot illustrates treatment duration and best overall response for each subject. The multi-Y plot overlays two biomarkers on separate y-axes to show their co-evolution over time.

For safety and efficacy summaries, the tornado plot displays adverse event incidence by toxicity grade for the top events, with treatment groups shown on opposite sides of a central axis. The bar plot shows the incidence of treatment-emergent adverse events by exposure duration category. The Kaplan-Meier plot displays overall survival curves with censoring indicators. The forest plot presents subgroup analysis results with hazard ratios and 95% confidence intervals.

All plots share a consistent `academic_theme()` built on `ggplot2`'s `theme_minimal()`, using a coordinated color palette with Treatment A in blue (#0072B2) and Treatment B in green (#009E73), following colorblind-friendly conventions from the Wong color palette.

## USER-DRIVEN PLOT GALLERY

A key design feature of the dashboard is the user-driven plot gallery (Display 2). Rather than displaying all plots in a fixed layout, users can select which plots to display using a dropdown multi-select input implemented with the `pickerInput()` function from the `shinyWidgets` package. The dropdown groups plots into two categories: pattern-compatible plots that support pattern-based highlighting, and other plots. Each plot name is annotated with a target symbol to indicate support for subject-level highlighting.



(Display 2. Layout of Plot Selection)

Users can also choose between a one-column and two-column layout using a radio button control. The plot gallery is rendered dynamically using `renderUI()` in the server, which constructs the appropriate grid of shinydashboard `box()` elements based on the user's selections. Plots that support click-to-explore functionality are rendered as `plotlyOutput()` elements, while other plots use standard `plotOutput()` elements.

## SUBJECT FILTERS AND REACTIVE DATA FLOW

The sidebar provides filtering controls that apply globally across all plots. Users can filter by treatment group, sex, and age range using select inputs and a slider. A "Clear All Filters" button resets all filters and subject highlight selections simultaneously. All plots update reactively when filters are changed, using Shiny's `reactive()` framework to propagate filtered datasets to all plot rendering functions. Subject dropdown choices are updated reactively based on the filtered population, ensuring that only subjects present in the current filtered dataset are available for highlight selection.

## PATTERN-BASED HIGHLIGHTING

In addition to subject-level highlighting, the plot gallery supports pattern-based highlighting through a pattern filter dropdown in the sidebar. Users can select from four predefined clinical response patterns: consistently decreasing, rebound, consistently increasing, and deep response.

Subjects matching the selected pattern are identified algorithmically from the laboratory data using `dplyr`-based logic applied to the reactive filtered dataset. Matched subjects are highlighted with a distinct red color (`#e74c3c`) across all pattern-compatible plots, which include the waterfall, scatter, spider, line, and box plots. The pattern highlight is rendered as a separate geom layer added on top of the base plot, using semi-transparent filled circles for point-based plots and bold overlaid lines for trajectory plots.

The sidebar displays a description of the selected pattern along with a match count, showing the number and percentage of currently filtered subjects who meet the pattern criteria. This real-time feedback allows users to quickly assess how common a given response pattern is within the filtered population before drilling into individual subjects.

## SUBJECT JOURNEY TIMELINE

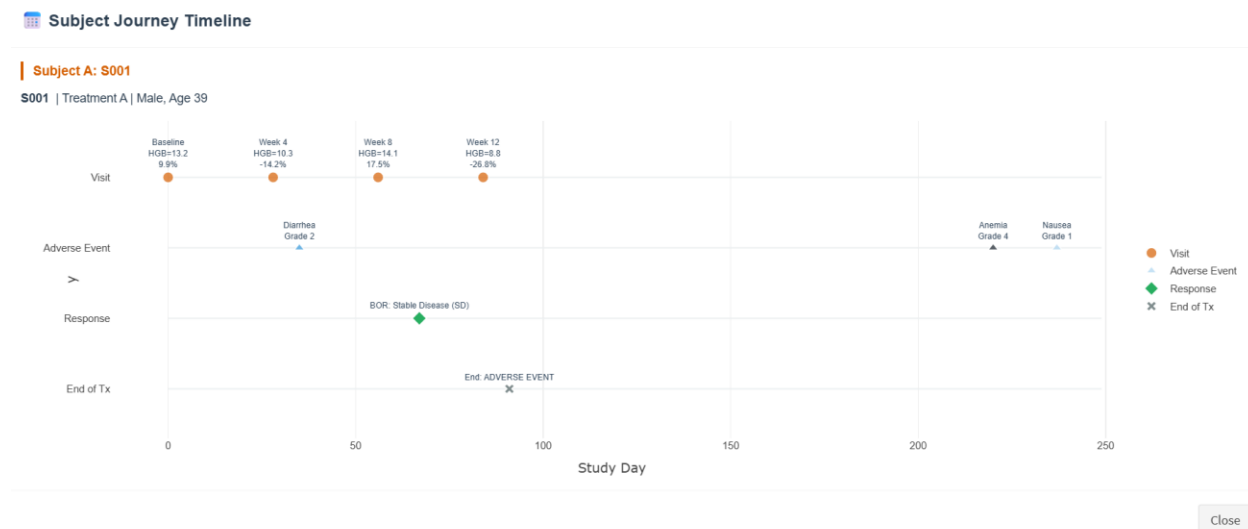
A subject journey timeline provides a longitudinal, event-level view of each highlighted subject's full clinical course. The timeline is accessible through a "Journey" button in the Subject Highlights section of the sidebar. When clicked, a full-width modal dialog is displayed showing a `plotly`-based timeline for Subject A, Subject B, or both subjects stacked vertically.

The journey plot is built directly with `plotly` rather than `ggplot2`, using four horizontal event lanes: visits (hemoglobin measurements), adverse events, best overall response, and end of treatment. Each event type uses a distinct marker shape and color scheme. Visit events are displayed with circular markers colored in the subject's highlight color, with hover text showing the visit name, hemoglobin value, and percent change. Adverse events are displayed as upward triangles with colors mapped to toxicity grade, ranging from light blue for Grade 1 to near-black for Grade 4. Response events use diamond markers in green, and end-of-treatment events use cross markers in gray.

Horizontal lane lines are rendered as background segments to visually separate the four event categories. The plot title displays the subject identifier, treatment group, sex, and age. When both subjects are selected, each subject's timeline is rendered in a separate `plotlyOutput()` at reduced height within the same modal, separated by a horizontal rule.

The journey timeline provides a subject-level narrative that complements the cross-plot highlighting system. While the highlighting system shows where a subject falls relative to the population distribution,

the journey timeline reveals the sequence and timing of clinical events for that subject, supporting a more complete understanding of individual treatment experience.



(Display 3. Example of Subject Journey Timeline)

## CROSS-PLOT SUBJECT LINKING

The cross-plot subject linking system is the interactive feature of the plot gallery. Users can select up to two subjects of interest from dropdown menus in the sidebar, labeled Subject A and Subject B. Subject A is highlighted in vermilion (#D55E00) and Subject B in rose pink (#CC79A7), providing clear visual distinction.

When a subject is selected, all plots that support subject highlighting are automatically updated to overlay the subject's data. For line and spider plots, the subject's individual trajectory is drawn as a dashed line with circular markers. For the scatter plot, the subject's data point is marked with a hollow circle. In the waterfall plot, the subject's bar is recolored. The swimmer plot recolors and labels the subject's bar. In the multi-Y plot, the subject's individual biomarker trends are overlaid on the group mean lines.

The highlight layers are implemented using ggplot2's `inherit.aes = FALSE` mechanism, which allows additional geom layers to be added to an existing plot without inheriting the base aesthetic mappings. This approach keeps the highlight logic separate from the base plot construction, making the code modular and easy to maintain.

## CLICK-TO-EXPLORE SUBJECT PROFILE

Three plots support an additional click-to-explore feature: the scatter plot, waterfall plot, and spider plot. These plots are rendered using `ggplotly()` from the `plotly` package, which converts `ggplot2` objects into interactive `plotly` visualizations supporting zoom, pan, and hover interactions.

When a highlighted subject's data point is clicked, a subject profile modal is triggered using Shiny's `showModal()` and `modalDialog()` functions. The modal displays the subject's demographic information, treatment assignment, sex, and age in the title bar. The modal body presents three panels side by side: a

laboratory data table showing hemoglobin values and percent changes at each study visit, an adverse event table listing all recorded events with toxicity grades and study day of onset, and a best overall response table showing the subject's response category and assessment day.

Each data panel is rendered as an interactive `DT::dataTableOutput()` element, allowing users to sort and scroll within the modal. If no adverse events or response data are available for the subject, a plain-text message is displayed in place of the table.

The click detection is implemented using plotly's `event_data("plotly_click", source = pid)` function. To ensure that only highlighted subjects trigger the modal, the server checks whether the clicked subject matches the currently selected Subject A or Subject B before displaying the modal. A subtitle annotation added to each clickable plot through plotly's `layout(annotations = ...)` mechanism displays the message "Click on highlighted subject to explore" in the highlight color, providing an in-plot cue to users.

## INTEGRATED STATISTICAL OPTIONS

The dashboard provides an optional statistical overlay panel accessible through a per-plot "Stats" button rendered in the toolbar below each plot. Clicking the button opens a modal dialog with statistical controls on the left and a live preview plot on the right. Users can configure statistical options, click "Run Stats" to update the preview and generate a plain-language interpretation, and then click "Apply to Dashboard" to push the annotated plot to the main gallery.

Depending on the plot type, available options include hypothesis tests (t-test, Wilcoxon rank-sum, log-rank, Gehan-Breslow-Wilcoxon, Fisher's exact, or chi-square), effect size measures (Cohen's d, mean difference, odds ratio, or risk difference), multiple comparison adjustments (Bonferroni or Benjamini-Hochberg FDR), and summary statistics such as hazard ratios, median survival, response rates, and treatment duration. Significant results are annotated directly on the plot with significance symbols and p-values.

Each statistical modal includes a plain-language interpretation section generated programmatically from the computed results, describing the direction, magnitude, and significance of observed differences in full sentences. A "Clear Stats" button resets all options to their default off state.

## CODE EDITING AND EXPORT

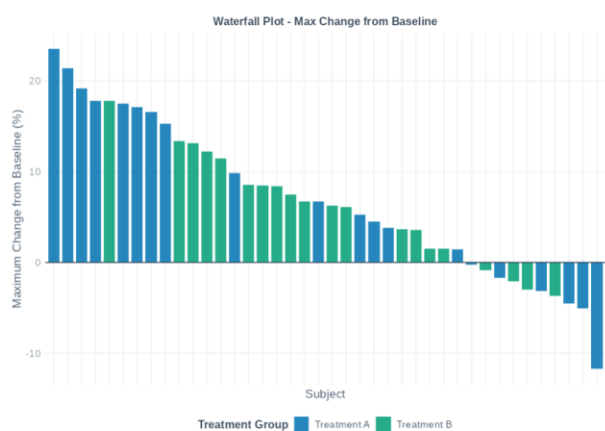
Each plot in the gallery is equipped with a toolbar rendered directly below the plot area, providing access to code editing and export functionality.

The code editor is implemented using the shinyAce package. Each plot has a "CODE" button in its toolbar. Clicking the button opens a modal dialog with an ACE editor on the left displaying the current source code of the plot's R function, and a live preview plot on the right. The plot function source code is extracted at runtime using a custom `extract_fn_code()` function that reads the application's own source file and locates the function definition by name using brace-depth tracking. A filter summary comment block is prepended to the extracted code, showing the currently active treatment, sex, and age filters along with the number of filtered subjects, providing context for the code being edited.

```

1 # -- Active Filters
2 # Treatment : All
3 # Sex : All
4 # Age Range : 25 - 75
5 # Subjects : 40 / 40 (after filter)
6
7 create_waterfall_plot <- function(data, subj_a = NULL, subj_b = NULL, pattern_sub
8                               stat_orr = FALSE, stat_thr = -30) {
9   plot_data <- data[sadhema %>%
10     group_by(SUBJID, TRT01A) %>%
11     summarise(max_pchg = max(PCMG, na.rm = TRUE), .groups = "drop") %>%
12     arrange(desc(max_pchg)) %>%
13     mutate(SUBJID_order = row_number(),
14            SUBJID = factor(SUBJID, levels = SUBJID))
15
16   p <- ggplot(plot_data, aes(x = SUBJID, y = max_pchg, fill = TRT01A,
17                             text = as.character(SUBJID), customdata = as.charac
18   geom_col(width = 0.8, alpha = 0.85) +
19   geom_hline(yintercept = 0, color = "#2c3e50", linewidth = 0.5) +
20   scale_fill_manual(name = "Treatment Group", values = TRT_COLORS) +
21   labs(title = "Waterfall Plot - Max Change from Baseline",
22        x = "Subject", y = "Maximum Change from Baseline (%)") +
23   academic_theme() +
24   theme(axis.text.x = element_blank(), axis.ticks.x = element_blank(),
25         plot.title = element_text(size = 12))
26
27   if (isTRUE(stat_orr)) {
28     p <- p + geom_hline(yintercept = stat_thr, linetype = "dashed", color = "#e74
29     orr <- plot_data %>%
30     group_by(TRT01A) %>%
31     summarise(orr = mean(max_pchg <= stat_thr) * 100, .groups = "drop")
32     ann <- paste(orr$TRT01A, sprintf("ORR=%0f%%", orr$orr), sep = ": ", collapse
33     p <- p + labs(caption = paste0("Response threshold: ", stat_thr, "% | ", an
34

```



Close

Download Code

Run Stats

Apply to Dashboard

#### (Display 4. Interface of Code Editor)

Users can modify the plot function code directly in the editor and click "Run" to evaluate the modified code in a sandboxed environment using `eval(parse(text = code), envir = new.env())`. If the evaluation succeeds, the preview plot is updated using the new function. If an error occurs, a formatted error message is displayed below the preview area. Clicking "Apply to Dashboard" replaces the plot in the main gallery with the custom version for the current session. A "Download Code" button allows users to export the edited function as a standalone R script. This feature bridges the gap between dashboard-based exploration and programmatic customization, allowing experienced R users to extend or modify any plot without leaving the dashboard environment.

For export, the toolbar provides PNG and PDF download options implemented using Shiny's `downloadHandler()` function, which renders the plot to a temporary file using `ggplot2`'s `ggsave()` and streams it to the user's browser. PDF export uses the `cairo_pdf` device to ensure proper font rendering. The toolbar is generated dynamically as part of the `renderUI()` gallery construction, so it is present for every plot regardless of which plots the user has selected.

## TECHNICAL IMPLEMENTATION CONSIDERATIONS

Several technical considerations were important in building a stable and performant dashboard. Reactive data flow was carefully structured to avoid unnecessary re-rendering. The filtered dataset is computed once in a `reactive()` expression and shared across all plot rendering functions. Subject highlight selections are passed as function arguments rather than accessed directly inside reactive expressions where possible, to minimize the scope of reactive dependencies.

The use of `lapply()` to register `renderPlot()` and `renderPlotly()` outputs for all plots in the registry allows the server code to remain concise and consistent. Each plot in the registry is defined with its title, rendering function, height, and flags indicating whether it supports pattern highlighting and click-to-explore functionality. This registry-driven approach makes it straightforward to add new plot types by adding an entry to the registry and implementing the corresponding plot function.

For plotly plots, the `ggplotly()` conversion introduces some rendering differences compared to static `ggplot2` output, particularly for marker shapes and annotation text. Annotations added via `ggplot2`'s `annotate()` function are not preserved by `ggplotly()`, so subject labels for plotly plots are added directly through plotly's `layout(annotations = ...)` mechanism instead. Highlight traces generated by `ggplot2`'s additional geom layers sometimes lack `customdata`, so a post-processing step iterates over the plotly trace list to inject subject identifiers into single-point marker traces by matching their coordinates against the main data traces.

The per-plot code editor uses brace-depth tracking to extract function boundaries from the source file without requiring a formal R parser, which is sufficient for well-formatted function definitions but assumes consistent brace placement. The sandboxed `eval()` environment inherits from the application's global environment, giving edited functions access to all helper functions and color constants defined at the top level.

## CONCLUSION

In this paper, we have presented an interactive clinical trial data visualization dashboard built with R Shiny, demonstrating how modern R packages can be combined to create a comprehensive and user-friendly platform for clinical data exploration. The plot gallery's key contributions include a user-driven gallery supporting eleven oncology-relevant plot types, a cross-plot subject linking system enabling simultaneous subject-level exploration across all visualizations, a pattern-based highlighting system for identifying clinically meaningful response subgroups, a click-to-explore subject profile modal providing instant access to laboratory, adverse event, and response data, a subject journey timeline offering a longitudinal event-level view of individual clinical courses, an integrated statistical options panel for overlaying hypothesis tests and effect sizes with plain-language interpretation, a per-plot code editor enabling in-dashboard customization of plot functions using the `shinyAce` package, and plot export functionality supporting PNG and PDF formats.

The modular design of the application, with a registry-driven plot management system, a shared reactive data flow, and a separation between base plot logic and highlight layers, makes it straightforward to extend the plot gallery with additional plot types or adapt it to real study data by replacing the simulated datasets with actual ADaM datasets. The consistent use of a colorblind-friendly color palette and a unified academic theme ensures that the visualizations are suitable for both internal review and presentation purposes.

Future enhancements could include integration with actual SAS datasets via the `haven` package, user authentication for multi-user deployment, and session state persistence to allow users to save and restore filter and highlight configurations across sessions.

## REFERENCES

Chang, W., Cheng, J., Allaire, J., Sievert, C., Schloerke, B., Xie, Y., Allen, J., McPherson, J., Dipert, A., & Borges, B. 2023. Shiny: Web Application Framework for R. R package version 1.7.5. <https://CRAN.R-project.org/package=shiny>.

Sievert, C. 2020. Interactive Web-Based Data Visualization with R, plotly, and shiny. Chapman and Hall/CRC. <https://plotly-r.com>.

Chang, W. & Borges Ribeiro, B. 2021. shinydashboard: Create Dashboards with Shiny. R package version 0.7.2. <https://CRAN.R-project.org/package=shinydashboard>.

Perrier, V., Meyer, F., & Granjon, D. 2023. shinyWidgets: Custom Inputs Widgets for Shiny. R package version 0.7.6. <https://CRAN.R-project.org/package=shinyWidgets>.

Wickham, H. 2016. ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag New York. <https://ggplot2.tidyverse.org>.

Trestle Technology, LLC. 2023. shinyAce: Ace Editor Bindings for Shiny. R package version 0.4.2. <https://CRAN.R-project.org/package=shinyAce>.

Wong, B. 2011. Color blindness. Nature Methods, 8(6), 441.

## ACKNOWLEDGMENTS

I would like to thank colleagues in C&SP for their continued support and feedback during the development of this plot gallery.

## CONTACT INFORMATION

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

You Lyu (Lea)  
Johnson & Johnson  
[ylv13@its.jnj.com](mailto:ylv13@its.jnj.com)

