PharmaSUG China 2024 - Paper DV-10064

Enhancing Tumor Response Data Monitoring with R-Shiny Visualization and Analytics Tools

Keting Lv, Daiichi Sankyo (China) Holdings Co., Ltd.

ABSTRACT

In the realm of solid tumor research, monitoring tumor response data in line with RECIST 1.1 guidelines is pivotal. This intricate process requires not only a harmonious alignment among the study team but also a seamless integration of multifaceted data sources. However, traditional SAS-based listing reviews often struggle to pinpoint critical or abnormal data points within tons of listings. This paper presents an R-Shiny powered interactive application, meticulously crafted to refine the efficacy data review process. It boasts the capability to autonomously generate reconciled listings for tumor responses, thereby facilitating the identification of data anomalies. The application also underscores discrepancies in tumor imaging assessment timelines, spotlighting any deviations from the prescribed schedule. The application enhances the data monitoring workflow by ensuring uniformity in examination protocols and visit chronology. It also generates standard efficacy plots—waterfall, spider, and swimmer plots—to augment data analysis.

The application accommodates input data from CDISC datasets SDTM or alternative formats, contingent on user-provided minimal data mapping. It is versatile, catering to both investigator and independent review data. The prospect of expanding R-Shiny efficacy data checking tools to encompass irRECIST, RANO, and RANO-BM criteria is being explored for wider applicability.

INTRODUCTION

We want our R-Shiny application to be developed with the following distinctions:

- **Interactive Dashboard**: The application consolidates multiple outputs into a single, dynamic dashboard, effectively preventing information overload and facilitating a more organized data presentation.
- **Real-Time Data Handling**: The system processes SDTM or raw data with minimal adjustments, bypassing the need for ADaM formatting and simplifying data preparation.
- **Tailored for Investigators and Independent Review**: While the primary focus for DMC/Interim Analysis/submissions remains on Investigator and Independent Adjudicated results, our application extends its capabilities for comprehensive data review. It includes additional filters and options for evaluators, such as buttons for Independent Reader 1 and Independent Reader 2, ensuring a broader spectrum of information is accessible.

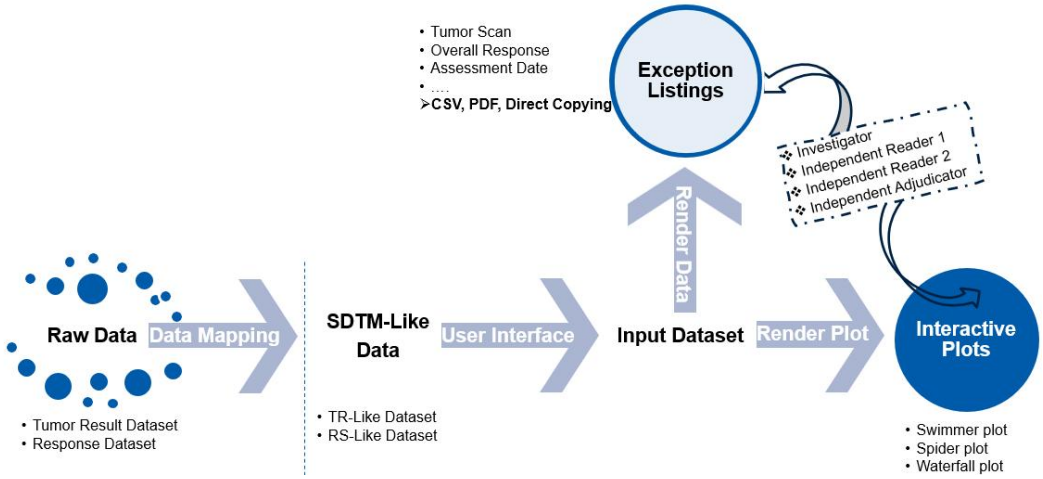
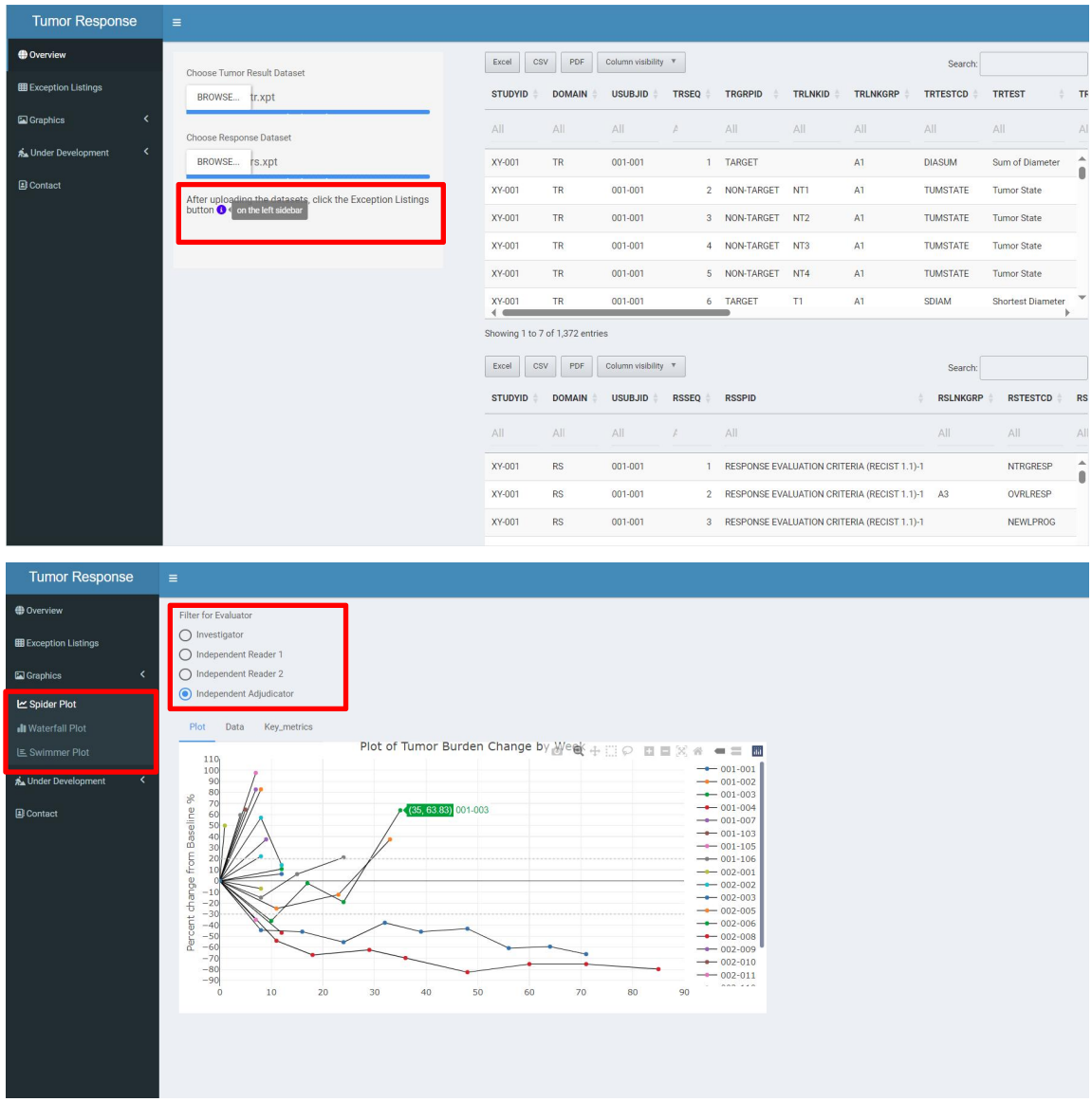


Figure 1. Data flow

- **Guided Interface**: Our goal is to unveil an application featuring a streamlined interface that utilizes succinct text for straightforward communication. Subtle navigational cues are provided through mouse-hover interactions and distinctive icons, enhancing user experience.
- **Multiple Outputs**: The application is designed to cater to diverse user needs by generating outputs in various formats such as CSV, XLSX, PDF, or enabling direct copying. Additionally, it offers interactive visualizations, allowing users to engage with the data more intuitively.

The application comprises two principal components: exception listings for checks and interactive plots for visual data analysis, each with distinct functionalities yet complementary capabilities.

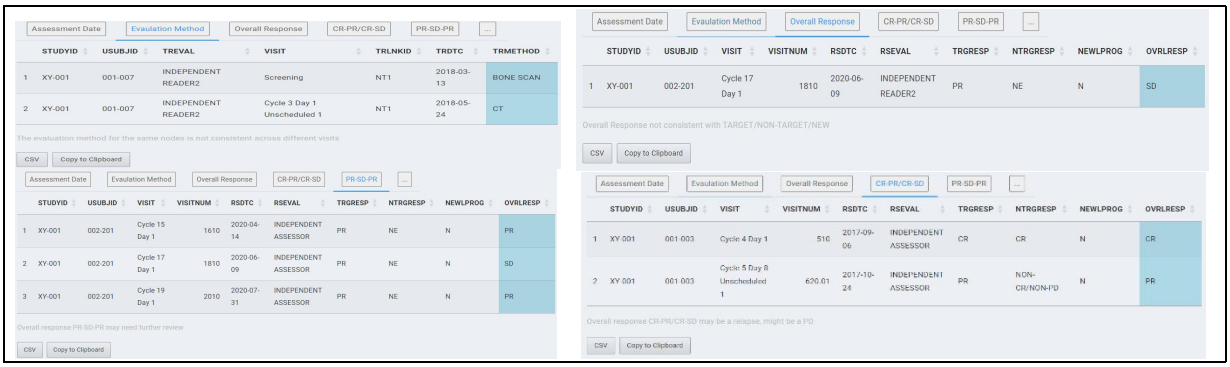
For illustrative purposes, this paper will present simplified dummy data, a streamlined checking example, figures, and condensed R-code.



Display 1. Screen capture of user interface

EXCEPTION LISTINGS

The first component of the R-Shiny application is exception listings. The variables that cause potential issues needing review will be highlighted.



Output 1. Example exception listing generated by this application

The example R code of the checking list CR followed by PR/SD can be found at appendix I.

#	Input Dataset	Target Item	Check Points	Check Logic
1	TR	Tumor Scan	Consistency of scan method	The evaluation method for the same nodes is not consistent across different visits
2	RS	Overall Response	Consistency of CRF response and derived results	Not consistent with TARGET/NON-TARGET/NEW per RECIST 1.1 guideline
3	RS	CR-PR	PR might implicitly be a PD	CR followed by PR
4	RS	CR-SD	SD might implicitly be a PD	CR followed by SD
5	RS	PR-SD-PR	Overall response SD appears between two PRs	SD between two PRs
6	RS	Assessment Date	Date of Overall Response; Date of Scan/Examination	Date of Overall Response is prior to the Date of Scan/Examination of a non- target tumor at the same assessment visit

Table 1. Exception listings check points examples

GRAPHICS

This application is designed to generate subject-level plots for tumor response data monitoring purposes. It includes a “Filter for Evaluator” button, allowing users to easily select their desired evaluator.

ADVANTAGES OF INTERACTIVE PLOTS

Traditional static plots sometimes struggle to balance conciseness with sufficient information disclosure. For instance, spider plots typically do not display subject IDs, preventing reviewers from identifying the subjects associated with each line. Adding subject IDs directly to the plot lines can lead to overlapping or crowding issues. However, interactive plots could fully utilize mouse-hover features to display subject ID information. Moreover, plots from different evaluators can be incorporated into a single dashboard.

Interactive plots can also facilitate the addition of widgets to filter specific populations or color-code subgroups within the dashboard, thus enhancing the review process. Currently, this application does not include subgroup analysis features, as its primary function is data monitoring, which typically requires reviewing all subjects.

CATEGORIES OF PLOTS

The application produces subject-level plots such as spider, swimmer, and waterfall plots, each supporting distinct aspects of data monitoring.

Swimmer plot

This plot provides a visualization of subjects' responses over time, delineated by weeks. It portrays each subject's progression with a variety of colors and shapes to denote different response statuses. Refer to Figure 2 for an illustration, with subjects 001-003 serving as examples. An instance of CR-PR may indicate a data issue, which the data manager can identify from the plot. This plot can then be utilized to facilitate communication with the central review vendor. The ggplot object is transformed into a plotly object to enhance interactivity. For the sample R code, see Appendix IV.

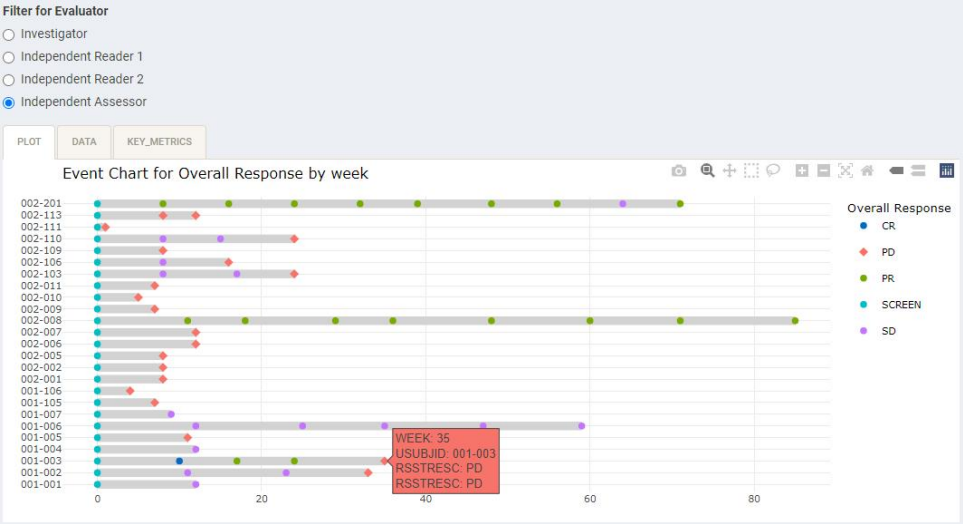


Figure 2. Interactive Swimmer Plot

Spider plot

The spider plot illustrates the percentage change in tumor burden from the baseline, aiding in the monitoring of tumor size alterations (e.g., detection of outliers) and verifying the consistency between tumor size and overall response. Hovering over the data points reveals the subject ID, along with the corresponding week (x-value) and percentage change (y-value).

This tool is invaluable for researchers and clinicians monitoring treatment efficacy through the visualization of tumor size fluctuations throughout a study. The plot's interactive features allow for a comprehensive examination of individual patient reactions to treatments and the choice of different evaluators. The spider plot is generated using a loop method to add traces. Refer to Appendix III for the sample R code.

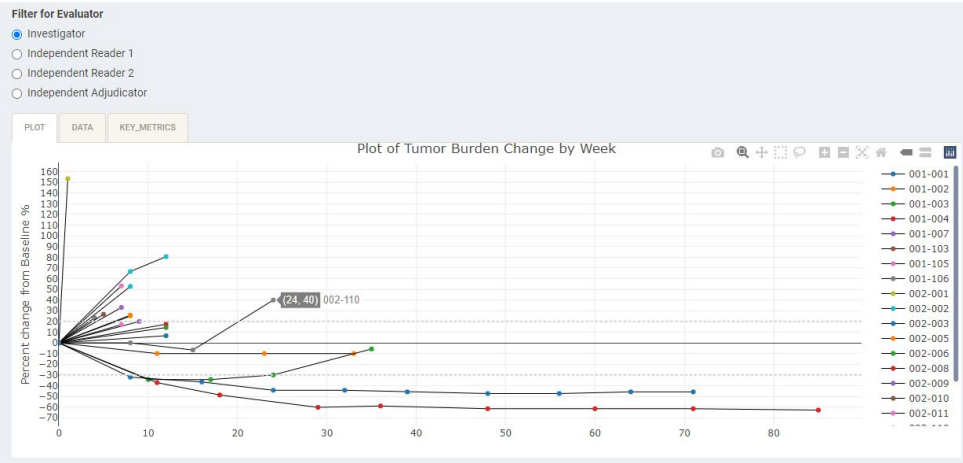


Figure 3. Interactive Spider Plot Displaying Patient ID on Hover

Waterfall plot

Focusing on the evolution of tumor size, as quantified by the sum of diameters, this plot highlights the subject's best percentage change from the baseline. For instance, in the example plot, the selected subject's best percentage change is less than 20%, yet a new lesion is displayed in the accompanying table, it indicates that the patient's BOR PD is attributable to the new lesion rather than an increase in the size of the target lesion.

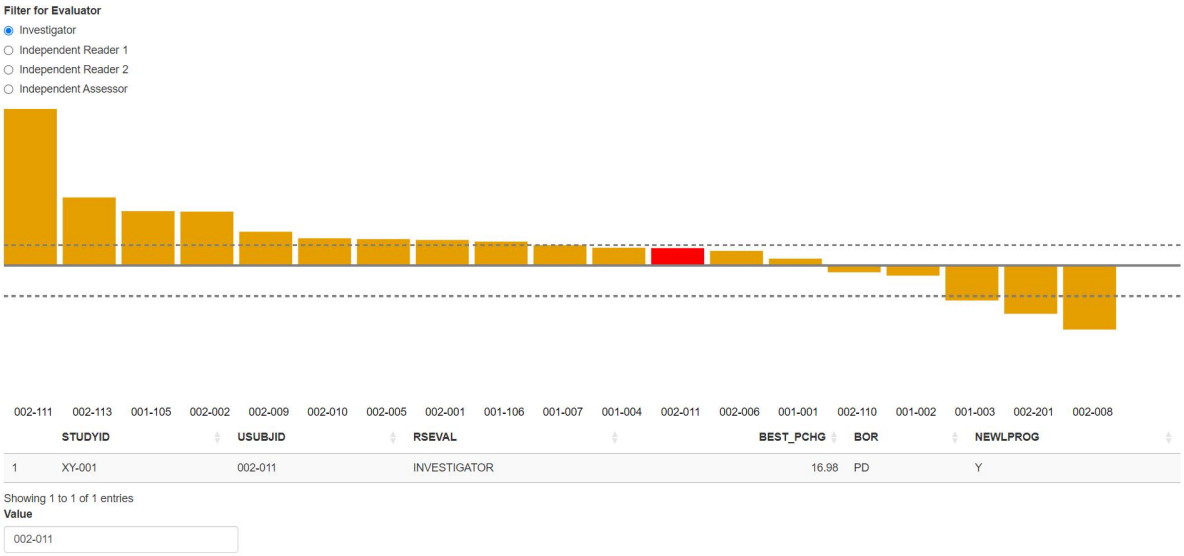


Figure 4. Interactive Waterfall Plot by R2D3

The r2d3_script, the second segment of the code, is a script for crafting an interactive bar chart utilizing the r2d3 package, which amalgamates D3.js with R. The r2d3 integration offers a dynamic mode of data presentation. However, due to the potential complexity of the D3 script and the length of the code, r2d3 should be considered only when specific interactivity—such as click-through actions, bespoke interactive functions, or fluid transitions—is required. The sample R code can be found at Appendix IV.

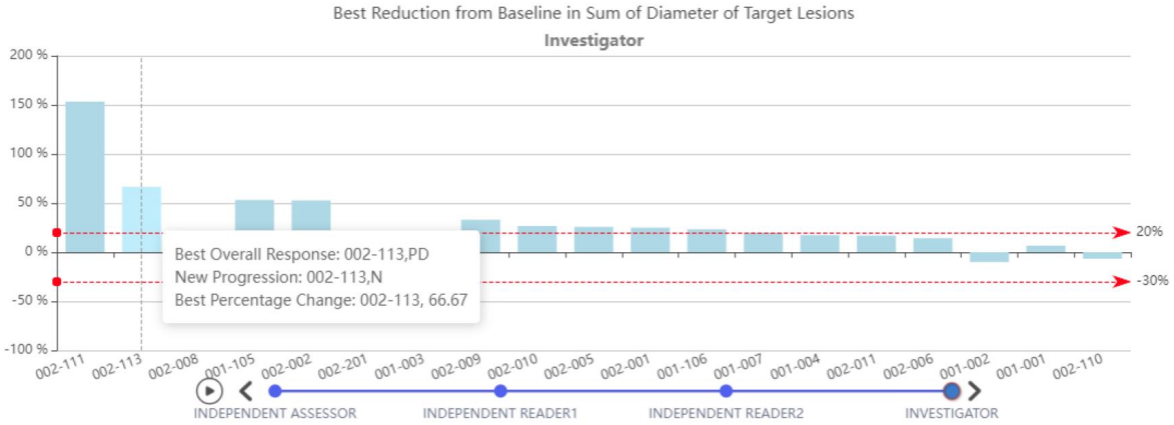


Figure 5. Interactive waterfall plot by echarts4r

Numerous valuable packages from the htmlwidgets for R, such as the echarts4r package, offer excellent options for generating efficacy plots in R-Shiny. Here is an example of a waterfall plot created using the echarts4r package, which serves as an alternative waterfall plot for this R-Shiny app. The syntax of the echarts4r package is very concise and easy to learn. In this example, the e_timeline_opts() function is introduced as a tool for selecting evaluators. Although its name suggests a "timeline," it can be considered an enhanced version of "grouping", effectively adding a component to the entire chart container that allows switching between different groups of data.

COMPARISON OF DIFFERENT R-SHINY PLOT APPROACHES

The application employs four distinct coding methodologies for the three different kinds of plots. The table below summarizes their unique features.

Coding Method	**ggplot to plotly**	**Loop to add trace with plotly**	**R2D3**	**Echarts4r**
Example	Swimmer plot	Spider Plot	Waterfall plot	Waterfall plot
R Package	library(ggplot2), library(plotly)	library(plotly)	library(r2d3)	library(echarts4r)
Core Function	ggplotly()	add_trace(),plot_ly()	d3Output(), r2d3()	e_charts(),e_timeline_opts()
Language	R	R	JavaScript and R	R and JavaScript
Advantage	Concise and user-friendly	Only needs plotly package	Flexibility in click-through interactivity	Directly call JavaScript
Drawback	Limited in click-through interactivity, not smooth transition	Rendering multiple traces can impact performance	Complicated D3 script brings more work	Not that flexible for specific function

Code Structure	output$Waterplot <- renderPlotly({
	# Create a ggplot object
	g <- ggplot() +
	geom_bar() +
	geom_abline() +
	geom_text(aes() +
	geom_point() +
	theme_bw() +
	labs(title = "",x = "", y = "") +
	# Convert it to an interactive plotly object
	ggplotly(g)
	})
		server <- function(input, output) {
		
		output$Spiderplot <- renderPlotly({
		## Initialize a plotly object with layout
		p <- plot_ly() %>%
		layout()
		
		## Identify columns to add as traces
		ToAdd <- ##
		
		## Add the traces one at a time
		for(i in ToAdd){
		p <- p %>% add_trace()
		}
		## Returns the plotly object 'p' as the final output
		p
		})
			r2d3_script <- "
			// D3 code inside an R variable
			"
			# Save D3 code into a temporary file
			r2d3_file <- tempfile()
			writeLines(r2d3_script, r2d3_file)
			
			ui <- fluidPage(
			d3Output("d3")
			)
			
			server <- function(input, output, session) {
			output$d3 <- renderD3({
			r2d3(r2d3_file)
			})}
			
			shinyApp(ui = ui, server = server)
				server <- function(input, output) {
				output$plot <- renderEcharts4r({
				trrso3 %>%
				
				group_by(RSEVAL) %>%
				
				e_charts(USUBJID, timeline = TRUE) %>%
				
				e_bar(BEST_PCHG, realtimeSort = TRUE, seriesLayoutBy = "column") %>%
				
				e_timeline_serie() %>%
				e_tooltip(trigger = "axis", formatter = htmlwidgets::JS())
				})
				}

Reference Gallery	Plotly r graphing library in R		Gallery • r2d3 (rstudio.github.io)	Chart Types • echarts4r (john-coene.com)

Table 2 Comparison of different methods in generating R-shiny interactive graphs

VISION

We aim to expand R-Shiny efficacy data checking tools to include irRECIST, RANO, and RANO-BM criteria for broader applicability. Below is an example of an Overall Response Check using the RANO criteria.

The RANO (Response Assessment in Neuro-Oncology) criteria, developed specifically for brain tumors, assess not only tumor size but also incorporate neurological examinations and corticosteroid use, making it often preferred over RECIST for brain tumor evaluation.

Type of Assessment	**Complete Response (CR)**	**Partial Response (PR)**	**Stable Disease (SD)**	**Progressive Disease (PD)**
Target response	None	**≥50%** decrease in **sum of product of perpendicular diameters** relative to baseline	<50% decrease <25% increase, relative to baseline	≥25% increase in sum of product of perpendicular diameters relative to **minimum**
Enhancing Non-target response	None	Stable or improved	Stable or improved	Unequivocal PD
Non-target Non-enhancing (T2/FLAIR)	Stable or decrease	Stable or decrease	Stable or decrease	Increase
New Lesions	None	None	None	Present
Steroid use	None	Stable or decrease	Stable or decreased	Not applicable
Clinical status	Stable or improved	Stable or improved	Stable or improved	Worse
Requirement for overall response	All	All	All	Any

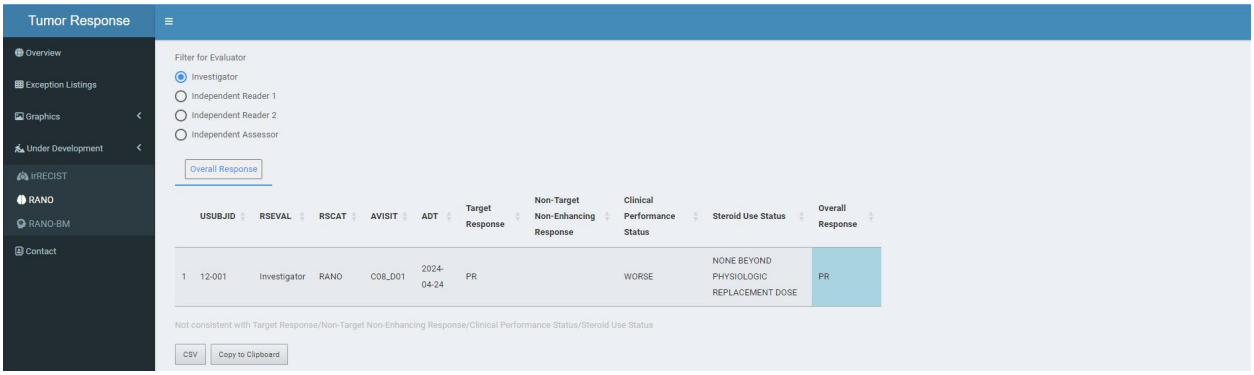
Table 3 RANO Overall Response Criteria

Type of Assessment	**Complete Response (CR)**	**Partial Response (PR)**	**Stable Disease (SD)**	**Progressive Disease (PD)**
Target response	None	≥30% decrease in sum of longest diameter relative to baseline	<30% decrease <20% increase, relative to minimum	≥20% increase, relative to minimum
Non-target Lesions	None	Stable or improved	Stable or improved	Unequivocal PD
New Lesions	None	None	None	Present
Requirement for overall response	All	All	All	Any
Target response	None	≥30% decrease in sum of longest diameter relative to baseline	<30% decrease <20% increase, relative to minimum	≥20% increase, relative to minimum
Non-target Lesions	None	Stable or improved	Stable or improved	Unequivocal PD
New Lesions	None	None	None	Present

Table 4 RECIST Overall Response Criteria

Key differences between RANO and RECIST include measurement methods (RANO uses the product of perpendicular diameters, while RECIST uses the sum of the longest diameter), thresholds for partial response (RANO PR: ⩾50% decrease; RECIST PR: ⩾30% decrease), and stable disease ranges (RANO SD: -50% to 25%; RECIST SD: -30% to 20%). Additionally, RANO PR and SD compare with the

baseline, while PD compares with the minimum, and RECIST PR compares with the baseline, while SD and PD compare with the minimum. RANO CR and PR must be confirmed after at least 4 weeks. For non-target lesions, RANO includes Enhancing Non-target response or T2/FLAIR, with no difference in new lesion standards between RANO and RECIST. RANO also considers corticosteroid use and clinical status (stable, improved, decreased), but an increase in corticosteroids alone does not indicate progression without persistent clinical deterioration. For both RANO and RECIST, all conditions must be met for CR, PR, and SD, while for PD, meeting any condition is sufficient.



Output 2. Example of exception listing by RANO criteria

CONCLUSION

Overall, this R-Shiny application serves as a dashboard for monitoring and validating tumor data in clinical trials, ensuring data quality and consistency with predefined scenarios. It provides an interactive platform for data analysts to upload datasets, apply checks, and review results efficiently.

REFERENCES

[1] Eisenhauer, E A et al. “New response evaluation criteria in solid tumours: revised RECIST guideline (version 1.1).” European journal of cancer (Oxford, England : 1990) vol. 45,2 (2009): 228-47.

[2] Gaillard F, Glick Y, Campos A, et al. RANO criteria for glioma. Reference article, Radiopaedia.org (Accessed on 05 Jul 2024) https://doi.org/10.53347/rID-29155

ACKNOWLEDGMENTS

The author would like to thank Daiichi Sankyo Biostatistics and Data Management Department tor reviewing the paper and providing valuable feedback.

CONTACT INFORMATION

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

Keting Lv
Daiichi Sankyo (China) Holdings Co., Ltd.
86-18106850068
lv.keting.kn@daiichisankyo.com.cn
https://bit.ly/Portfolio-klyu

APPENDIX I: DEMO R CODE OF EXCEPTION LIST

```
library(shiny)
library(shinythemes)
library(shinydashboard)
library(tidyverse)
library(DT)
library(shinyBS)
library(sassy)
library(plotly)
library(echarts4r)

# Define the user interface (UI)
ui <- fluidPage(
  theme = shinytheme("paper"), # Set the theme for the UI
  dashboardPage(
    skin = "blue", # Set the color theme for the dashboard
    header = dashboardHeader(title="Tumor Response"), # Create a header for
the dashboard
    sidebar = dashboardSidebar( # Create a sidebar for navigation
      sidebarMenu(
        menuItem("Overview", icon = icon("globe"), tabName = "overview"), #
Add menu items to the sidebar
        menuItem("Exception Listings", icon = icon("th"), tabName =
"utilities")
      )
    ),
    body = dashboardBody( # Define the body of the dashboard
      tabItems(
        # Create tabs for different sections of the dashboard
        tabItem(tabName = "overview",
                sidebarLayout(
                  sidebarPanel(
                    style = "height: 30vh; overflow-y: auto;", # Style the
sidebar panel
                    tags$div(style="margin-bottom: 20px;"),
                    fileInput("sasdata3", "Choose Response Dataset",
multiple = FALSE, accept = ".xpt"), # File input for uploading datasets
                    h4("After uploading the datasets, click the Exception
Listings button",
                       icon("info-circle", id = "info-icon-2", style =
"color: blue; cursor: pointer;")
                    )
                  ),
                  mainPanel(
                    DT::DTOutput("alldata2"), # Display data tables in the
main panel
                    div(style = "height:70px"),
                    DT::DTOutput("alldata3")
                  )
                ),
                bsTooltip(id = "info-icon-2", title = "on the left sidebar",
placement = "right", trigger = "hover")
        ),

        tabItem(tabName = "utilities",
                mainPanel(
```

```
                  radioButtons("rseval", "Filter for Evaluator", # Radio
buttons for filtering data
                               choices = c(
                                 "Investigator" = 'INVESTIGATOR',
                                 "Independent Reader 1" = 'INDEPENDENT
READER1',
                                 "Independent Reader 2" = 'INDEPENDENT
READER2',
                                 "Independent Assessor" = 'INDEPENDENT
ASSESSOR'
                               ),
                               selected = 'INVESTIGATOR'
                  ),
                  tabsetPanel(
                    id = 'dataset',
                    tabPanel(
                      title = tags$button(
                        "CR-PR/CR-SD",
                        id = "scenario1-btn",
                        title = "Overall response CR-PR/CR-SD may be a
relapse, might be a PD"
                      ),
                      DT::dataTableOutput("mytable4") # Output for the data
table
                    ),
                  )
                )
        ),

        tabItem(tabName = "contact", # Placeholder for contact information
tab
        )
      )
    )
  )
)

# Define the server logic
server <- function(input, output, session) {
  # Reactive expression to read the uploaded dataset
  DAT_C <- reactive({
    req(input$sasdata3) # Ensure that a file is uploaded
    inData3 <- input$sasdata3
    if (is.null(inData3)){ return(NULL) } # Return NULL if no file is
uploaded
    mydata3 <- haven::read_xpt(inData3$datapath) # Read the SAS XPT file
  })

  # Reactive expression to filter the dataset based on evaluator selection
  DAT_C_F <- reactive({
    DAT_C() %>%
      dplyr::filter(RSEVAL==input$rseval) # Filter the data by evaluator
  })

  # Render the data table for all data
  output$alldata3= DT::renderDataTable({
    DT::datatable(
```

```
      DAT_C(),
      filter = 'top', extensions = c('Buttons', 'Scroller'),
      options = list(
        scrollY = 250,
        scrollX = 500,
        deferRender = TRUE,
        scroller = TRUE,
        buttons = list('excel','csv','pdf',
                       list(extend = 'colvis', targets = 0, visible = FALSE)
        ),
        dom = 'lBfrtip',
        fixedColumns = TRUE
      ),
      rownames = FALSE
    )
  })

  # Render the data table for filtered data
  output$mytable4 <- DT::renderDataTable({
    d_all <- DAT_C_F()%>%

select(STUDYID,USUBJID,RSEVAL,RSTESTCD,RSSTRESC,VISITNUM,VISIT,RSDTC,RSDY)%
>%
      proc_transpose(

by=c("STUDYID","USUBJID","RSEVAL","VISITNUM","VISIT","RSDTC","RSDY"),
        var=c("RSSTRESC"),
        id=c("RSTESTCD")
      )%>%
      arrange(USUBJID,VISITNUM,RSDTC)%>%
      group_by(USUBJID, RSEVAL) %>%
      mutate(LAG_OR=lag(OVRLRESP),LEAD_OR=lead(OVRLRESP))%>%
      filter(
        OVRLRESP=="CR" & LEAD_OR=="PR"|
          OVRLRESP=="PR" & LAG_OR=="CR"|
          OVRLRESP=="CR" & LEAD_OR=="SD"|
          OVRLRESP=="SD" & LAG_OR=="CR"
      )%>%

select(STUDYID,USUBJID,VISIT,VISITNUM,RSDTC,RSEVAL,TRGRESP,NTRGRESP,NEWLPRO
G,OVRLRESP)
    DT::datatable(
      d_all,
      caption = tags$caption(
        style = "caption-side: bottom; text-align: left; margin: 8px 0;",
        "Overall response CR-PR/CR-SD may be a relapse, might be a PD"
      ),
      extensions = 'Buttons',
      options = list(
        dom = 'tB',
        paging = TRUE,
        searching = TRUE,
        fixedColumns = TRUE,
        autoWidth = TRUE,
        ordering = TRUE,
        buttons = list(
          list(extend = 'csv', filename = "CR followed by PRSD"),
```

```
          list(extend = 'copy', text = "Copy to Clipboard")
        )
      )
    )%>%
      formatStyle(
        columns = c('OVRLRESP'),
        backgroundColor = 'lightblue'
      )
  })
}

# Run the Shiny application
shinyApp(ui = ui, server = server)
```

APPENDIX II: SAMPLE R CODE OF SWIMMER PLOT

```
rs1 <- sdtm$rs %>%
  filter(RSTESTCD == "OVRLRESP") %>%
  mutate(WEEK = round(RSDY/7, digits = 0)) %>%
  select(USUBJID, RSEVAL, RSSTRESC, WEEK)

results, setting response as 'SCREEN' and week as 0
rs0 <- rs1 %>%
  distinct(USUBJID, RSEVAL) %>%
  mutate(RSSTRESC = "SCREEN", WEEK = 0)

rs2 <- bind_rows(rs1, rs0) %>%
  arrange(USUBJID, WEEK)

ui <- fluidPage(
  mainPanel(
    plotlyOutput("plot")
  )
)

server <- function(input, output) {
  output$plot <- renderPlotly({
    # Create a ggplot object with 'rs2' data, mapping weeks to x-axis and
subject IDs to y-axis
    sw <- ggplot(rs2, aes(x = WEEK, y = USUBJID)) +
      geom_line(color = "lightgray", size = 2.5) + # Add lines in light
gray color
      geom_point(aes(color = RSSTRESC, shape = RSSTRESC), size = 1.5) + #
Color and shape points based on 'RSSTRESC'
      scale_color_manual(values = c("SCREEN" = "#00BFC4", "CR" = "#036BC1",
"PR" = "#78AB00", "SD" = "#C475FF", "PD" = "#F8746B", "NE" = "#7F7F7F"),
                         limits = c("SCREEN", "CR", "PR", "SD", "PD", "NE"))
+
      scale_shape_manual(values = c("SCREEN" = 16, "CR" = 16, "PR" = 16,
"SD" = 16, "PD" = 18, "NE" = 16),
                         limits = c("SCREEN", "CR", "PR", "SD", "PD", "NE"))
+ # Use diamond shape for 'PD'
      labs(y = "", x = "",
           title = "Event Chart for Overall Response by Week") +
      theme_minimal(base_size = 14) + # Use a minimal theme with base font
size 14
      guides(color = guide_legend(title = "Overall Response"),
             shape = guide_legend(title = "Overall Response")) +
      guides(color = FALSE)


    # Convert the ggplot object to an interactive plotly object
    ggplotly(sw)
  })
}

shinyApp(ui = ui, server = server)
```

APPENDIX III: SAMPLE R CODE OF SPIDER PLOT

```
ui <- fluidPage(

  mainPanel(
    # Output
    tabsetPanel(type = "tabs",
                tabPanel("Plot", plotlyOutput('Spiderplot')),
                tabPanel("Data", DT::dataTableOutput("Spidertable")),
                tabPanel("Key_metrics",
DT::dataTableOutput("SpiderKey_metrics")))
  )
)

# Create a new data frame 'tr1' from 'sdtm$tr'
tr1 <- sdtm$tr %>%
  filter(TRTEST == "Sum of Diameter") %>%
  select(STUDYID, USUBJID, TREVAL, TRSTRESN, VISIT, TRDY, TRDY) %>%
  group_by(USUBJID) %>%
  # Create new columns 'BASE' for baseline and 'PCHG' for percent change
  mutate(
    BASE = TRSTRESN[VISIT == "Screening"],
    PCHG = round(if_else(VISIT != "Screening", (TRSTRESN - BASE) / BASE *
100, 0), digits=2)
  ) %>%
  ungroup() %>%
  # Modify 'TRDY' column for Screening visit and round the days to weeks
  mutate(TRDY = ifelse(VISIT == "Screening", 0, TRDY),
         TRDY = round(TRDY / 7, digits = 0)) %>%
  # Transpose the data frame to wide format
  proc_transpose(
    by = c("STUDYID", "TREVAL", "TRDY"),
    var = c("PCHG"),
    id = c("USUBJID")
  )

# Define a function 'hline' to create horizontal lines for plots
hline <- function(y = 0, color = "lightgray") {
  list(
    type = "line",
    x0 = 0,
    x1 = 1,
    xref = "paper",
    y0 = y,
    y1 = y,
    line = list(color = color, dash = "dot")
  )
}

server <- function(input, output) {

  output$Spiderplot <- renderPlotly({
    # Initialize a plotly object
    p <- plot_ly() %>%
      # Set the layout for the plot title and axes
      layout(title = "Plot of Tumor Burden Change by Week",
             xaxis = list(title = "", rangemode = 'tozero'),
```

```
            yaxis = list(title = "Percent change from Baseline %", dtick =
10)) %>%
      # Add horizontal lines at y=20 and y=-30
      layout(shapes = list(hline(20), hline(-30)))

    # Identify columns to add to the plot that are not part of the id
variables
    ToAdd <- setdiff(colnames(tr1), c("STUDYID", "TREVAL", "TRDY", "NAME"))

    # Loop through each column to add as a trace to the plot
    for (i in ToAdd) {
      p <- p %>%
        add_trace(x = tr1[["TRDY"]], y = tr1[[i]], name = i,
                  type = 'scatter',
                  mode = 'line+markers',
                  line = list(color = 'black', width = 1))
    }
    # Return the plot object
    p
  })
}

shinyApp(ui,server)
```

APPENDIX IV: SAMPLE R CODE OF WATERFALL PLOT BY R2D3

```
rstr <- sdtm$tr%>%
  filter(TRTEST == "Sum of Diameter") %>%
  group_by(STUDYID, TREVAL, USUBJID) %>%
  mutate(
    BASE = TRSTRESN[VISIT == "Screening"],
    PCHG = round(if_else(VISIT != "Screening", (TRSTRESN - BASE) / BASE *
100, 0), digits=2),
    TRDY = if_else(VISIT == "Screening", 0, TRDY),
    TRDY = round(TRDY / 7, digits = 0)
  ) %>%
  ungroup() %>%
  filter(VISIT != "Screening") %>%
  group_by(STUDYID, TREVAL, USUBJID) %>%
  summarize(BEST_PCHG = min(PCHG), .groups = "drop") %>%
  mutate(RSEVAL = TREVAL) %>% # remove later
  left_join(
    sdtm$rs %>%
      filter(RSTESTCD %in% c("OVRLRESP","NEWLPROG"))%>%
      proc_transpose(
        by=c("STUDYID",
"USUBJID","RSEVAL","VISITNUM","VISIT","RSDTC","RSDY"),
        var=c("RSSTRESC"),
        id=c("RSTESTCD"))%>%
      arrange(USUBJID,VISITNUM,RSDTC)%>%
      select(STUDYID,USUBJID,RSEVAL,OVRLRESP,NEWLPROG) %>%
      mutate(OVRLRESP = factor(OVRLRESP, levels = c("CR", "PR", "SD", "PD",
"NE"), ordered = TRUE)) %>%
      group_by(STUDYID, RSEVAL, USUBJID) %>%
      slice(1)%>%
      mutate(BOR = OVRLRESP),
    by = c("STUDYID", "RSEVAL", "USUBJID")
  ) %>%
  arrange(BEST_PCHG) %>%
  mutate(
    BOR_TEXT = case_when(
      BEST_PCHG >= 0 ~ BEST_PCHG + 5,
      BEST_PCHG < 0 ~ BEST_PCHG - 5
    ),
  )


r2d3_script <- "
// Define the r2d3 script as a string
function svg_height() {return parseInt(svg.style('height'))}
function svg_width()  {return parseInt(svg.style('width'))}
function row_left() {return svg_width() * 0; }
function row_bottom()  {return svg_height() * 0.5; }
function actual_max() {return d3.max(data, function (d) {return d.y; }); }
function row_height() {return (svg_height() / actual_max()) * 0.5; }
function row_width()  {return svg_width() / data.length * 0.95; }
var bars = svg.selectAll('rect').data(data);

// For each new data point, append a 'rect' element
bars.enter().append('rect')
```

```
    .attr('x',      function(d, i) { return i * row_width() + row_left(); })
// Set the x position
    .attr('y',      function(d) { if(d.y<0) {return row_bottom();} else
{return row_bottom() - d.y * row_height();}}) // Adjust y position based on
'y' value
    .attr('width',  row_width() * 0.9) // Set the width of the bar
    .attr('height', function(d) { return Math.abs(d.y) * row_height(); })
// Set the height of the bar
    .attr('fill',   function(d) {return d.fill; }) // Set the fill color
    .attr('id',     function(d) {return (d.label); }) // Set the id
attribute
    .on('click', function(){
      Shiny.setInputValue('bar_clicked', d3.select(this).attr('id'),
{priority: 'event'}); // On click, set the 'bar_clicked' input value in
Shiny
    })
    .on('mouseover', function(){
      d3.select(this).attr('fill', function(d) {return d.mouseover; }); //
On mouseover, change the fill color
    })
    .on('mouseout', function(){
      d3.select(this).attr('fill', function(d) {return d.fill; }); // On
mouseout, revert the fill color
    });

// Transition the bars to new positions with a duration of 500ms
bars.transition()
  .duration(500)
    .attr('x',      function(d, i) { return i * row_width() + row_left(); })
// Animate the x position
    .attr('y',      function(d) { if(d.y<0) {return row_bottom();} else
{return row_bottom() - d.y * row_height();}}) // Animate the y position
    .attr('width',  row_width() * 0.9) // Animate the width of the bar
    .attr('height', function(d) { return Math.abs(d.y) * row_height(); })
// Animate the height of the bar
    .attr('fill',   function(d) {return d.fill; }) // Animate the fill
color
    .attr('id',     function(d) {return (d.label); }); // Animate the id
attribute

// Remove any bars that are no longer bound to data
bars.exit().remove();

// Function to add a reference line at a specific 'y' value with a given
stroke style
function addReferenceLine(yValue, strokeStyle) {
  svg.append('line')
    .attr('x1', 0)
    .attr('y1', row_bottom() - yValue * row_height()) // Set the starting y
position
    .attr('x2', svg_width()) // Set the ending x position
    .attr('y2', row_bottom() - yValue * row_height()) // Set the ending y
position
    .attr('stroke', 'gray') // Set the stroke color
    .attr('stroke-width', 2) // Set the stroke width
    .attr('stroke-dasharray', strokeStyle); // Set the stroke dash array
(solid or dashed line)
```

```
}

// Add a solid reference line at y = 0
addReferenceLine(0, '0');

// Add dashed reference lines at y = 20 and y = -30
addReferenceLine(20, '5,5');
addReferenceLine(-30, '5,5');

// Select all 'text' elements and bind them to the data for labels
var txt = svg.selectAll('text').data(data);

// For each new label, append a 'text' element
txt.enter().append('text')
    .attr('x', function(d, i)  { return i * row_width() + row_left() +
(row_width() / 2); }) // Set the x position
    .attr('y', svg_height() * 0.98) // Set the y position
    .text(function(d) {return d.label; }) // Set the text content
    .style('font-family', 'sans-serif') // Set the font family
    .style('text-anchor', 'middle'); // Set the text anchor to middle

// Transition the labels to new positions with a duration of 1000ms
txt.transition()
    .duration(1000)
    .attr('x', function(d, i)  { return i * row_width() + row_left() +
(row_width() / 2); }) // Animate the x position
    .attr('y', svg_height() * 0.98) // Animate the y position
    .text(function(d) {return d.label; }); // Animate the text content

// Remove any labels that are no longer bound to data
txt.exit().remove();

"

# Save the r2d3 script into a temporary file
r2d3_file <- tempfile()
writeLines(r2d3_script, r2d3_file)

# Define the UI for the Shiny application
ui <- fluidPage(
  radioButtons("var", "Filter for Evaluator",
               choices = c("Investigator" = 'INVESTIGATOR',
                           "Independent Reader 1" = 'INDEPENDENT READER1',
                           "Independent Reader 2" = 'INDEPENDENT READER2',
                           "Independent Assessor" = 'INDEPENDENT ASSESSOR'),
               selected = 'INVESTIGATOR'), # Radio buttons to select the
evaluator
  d3Output("d3"), # Output for the D3 visualization
  DT::dataTableOutput("table"), # Output for the data table
  textInput("val", "Value", "USUBJID") # Text input for the user subject ID
)

# Define the server logic for the Shiny application
server <- function(input, output, session) {
  filteredData <- reactive({
    rstr %>%
```

```
      filter(RSEVAL == input$var) # Filter the data based on the selected
evaluator
  })

  output$d3 <- renderD3({
    filteredData() %>%
      mutate(label = USUBJID) %>% # Add a label column
    arrange(desc(BEST_PCHG)) %>% # Arrange the data by best percentage
change
    mutate(
      y = BEST_PCHG, # Set the 'y' value
      ylabel = prettyNum(BEST_PCHG, big.mark = ","), # Format the 'ylabel'
      fill = ifelse(USUBJID != input$val, "#E69F00", "red"), # Set the fill
color based on the user subject ID
      mouseover = "#0072B2" # Set the mouseover color
    ) %>%
      r2d3(r2d3_file) # Render the D3 visualization
  })

  observeEvent(input$bar_clicked, {
    updateTextInput(session, "val", value = input$bar_clicked) # Update the
text input when a bar is clicked
  })

  output$table <- renderDataTable({
    filteredData() %>%
      filter(USUBJID == input$val) %>% # Filter the data by user subject ID
    select(STUDYID, USUBJID, RSEVAL, BEST_PCHG, BOR, NEWLPROG) %>% # Select
relevant columns
    datatable(options = list(
      paging = FALSE,
      searching = FALSE # Disable paging and searching for the table
    ))
  })
}

# Run the Shiny application
shinyApp(ui = ui, server = server)
```

APPENDIX V: SAMPLE R CODE OF WATERFALL PLOT BY ECHARTS4R

```
trrso3 <- tr %>%
  filter(TRTEST == "Sum of Diameter") %>%
  group_by(STUDYID, TREVAL, USUBJID) %>%
  mutate(
    BASE = TRSTRESN[VISIT == "Screening"],
    PCHG = round(if_else(VISIT != "Screening", (TRSTRESN - BASE) / BASE *
100, 0), digits=2),
    TRDY = if_else(VISIT == "Screening", 0, TRDY),
    TRDY = round(TRDY / 7, digits = 0)
  ) %>%
  ungroup() %>%
  filter(VISIT != "Screening") %>%
  group_by(STUDYID, TREVAL, USUBJID) %>%
  summarize(BEST_PCHG = min(PCHG), .groups = "drop") %>%
  mutate(RSEVAL = TREVAL) %>% # remove later
  left_join(
    rs %>%
      filter(RSTESTCD %in% c("OVRLRESP","NEWLPROG"))%>%
      proc_transpose(
        by=c("STUDYID",
"USUBJID","RSEVAL","VISITNUM","VISIT","RSDTC","RSDY"),
        var=c("RSSTRESC"),
        id=c("RSTESTCD"))%>%
      arrange(USUBJID,VISITNUM,RSDTC)%>%
      select(STUDYID,USUBJID,RSEVAL,OVRLRESP,NEWLPROG) %>%
      mutate(OVRLRESP = factor(OVRLRESP, levels = c("CR", "PR", "SD", "PD",
"NE"), ordered = TRUE)) %>%
      group_by(STUDYID, RSEVAL, USUBJID) %>%
      slice(1)%>%
      mutate(BOR = OVRLRESP),
    by = c("STUDYID", "RSEVAL", "USUBJID")
  ) %>%
  arrange(BEST_PCHG) %>%
  mutate(
    BOR_TEXT = case_when(
      BEST_PCHG >= 0 ~ BEST_PCHG + 5,
      BEST_PCHG < 0 ~ BEST_PCHG - 5
    ),
    #USUBJID = factor(USUBJID, levels = USUBJID[order(BEST_PCHG, decreasing
= TRUE)])
  )

ck <- trrso3%>%arrange(RSEVAL,desc(BEST_PCHG))

# Define the user interface
ui <- fluidPage(
  mainPanel(
    echarts4rOutput("plot")
  )
)

server <- function(input, output) {
  output$plot <- renderEcharts4r({
    trrso3 %>%
      group_by(RSEVAL) %>%
```

```
      e_charts(USUBJID, timeline = TRUE) %>%
      e_scatter(BOR, symbolSize = 10, name = "Best Overall Response",
symbol="circle") %>%
      e_scatter(NEWLPROG, symbolSize = 10, name = "New Progression",
symbol="circle") %>%
      e_bar(BEST_PCHG, name = "Best Percentage Change", realtimeSort = TRUE,
seriesLayoutBy = "column") %>%
      e_title(left = "center", top = 10) %>%
      e_timeline_serie(title = list(
        list(
          text = "Best Reduction from Baseline in Sum of Diameter of Target
Lesions",
          textStyle = list(
            fontWeight = "normal",
            fontSize = 15
          ),
          subtext = "Independent Adjudicator",
          subtextStyle = list(
            fontWeight = "bold",
            fontSize = 15
          )
        ),
        list(text = "Best Reduction from Baseline in Sum of Diameter of
Target Lesions", subtext = "Independent Reader 1"),
        list(text = "Best Reduction from Baseline in Sum of Diameter of
Target Lesions", subtext = "Independent Reader 2"),
        list(text = "Best Reduction from Baseline in Sum of Diameter of
Target Lesions", subtext = "Investigator")
      )) %>%
      e_legend(show = FALSE) %>%
      e_mark_line(data = list(yAxis = 20), title = "20%", lineStyle =
list(type = "dashed", color = "red")) %>%
      e_mark_line(data = list(yAxis = -30), title = "-30%", lineStyle =
list(type = "dashed", color = "red")) %>%
      e_tooltip(trigger = "axis", formatter = htmlwidgets::JS("
        function(params) {
          var tooltipText = '';
          params.forEach(function(item) {
            if (item.seriesName !== 'USUBJID') {
              tooltipText += item.seriesName + ': ' + item.value + '<br>';
            }
          });
          return tooltipText;
        }
      ")) %>%
      e_x_axis(axisLabel = list(rotate = 20)) %>%
      e_y_axis(formatter = "{value} %")%>%
      e_theme("weforum")%>%
      e_color("lightblue")
  })
}

# Run the application
shinyApp(ui = ui, server = server)
```