PharmaSUG China 2023 - Paper DV-171

Interactive Table Cross-Querying and Spotlight Effects in Shiny Apps with jQuery

Hengyi Yang, Qilu Pharmaceutical Co., Ltd

ABSTRACT

R Shiny is a web application framework in R that allows you to build interactive web applications. It can be particularly useful in the field of clinical research and development by providing a platform to create dynamic and customizable applications for data analysis, visualization, and reporting.

JavaScript and CSS act as the artistic brushstrokes that breathe life into RShiny applications, with JavaScript providing dynamic interactivity and data manipulation while CSS adds visual styling and layout, resulting in a captivating and visually immersive user experience.

This paper will provide multiple examples showcasing the practical implementation of JavaScript applications in the field of clinical research and their direct application in the development

INTRODUCTION

Shiny apps have emerged as a popular data analysis tool, particularly in the field of clinical research and development. These interactive web applications provide a user-friendly interface for visualizing and exploring data, making them valuable resources for researchers and analysts. However, there are instances where traditional Shiny apps fall short in meeting everyday requirements.

Using JavaScript and CSS, we can enhance the flexibility and aesthetics of a shiny app. Below, we'll leverage jQuery, a powerful JavaScript framework, along with CSS, to implement some features that improve the user experience:

- Table Subquery: By injecting jQuery, we can enable a functionality where users can click on a specific content in one table to query another table. This allows for seamless retrieval of related information.
- Table Spotlight Effect: We can create a visually appealing effect where the content in a table dynamically follows the movement of the mouse cursor, creating a spotlight-like effect. This draws attention to the table's content and enhances user engagement.

By incorporating these JavaScript and CSS techniques, we can elevate the overall interactivity and visual appeal of the shiny app, resulting in a more intuitive and delightful user experience.

TABLE SUBQUERY

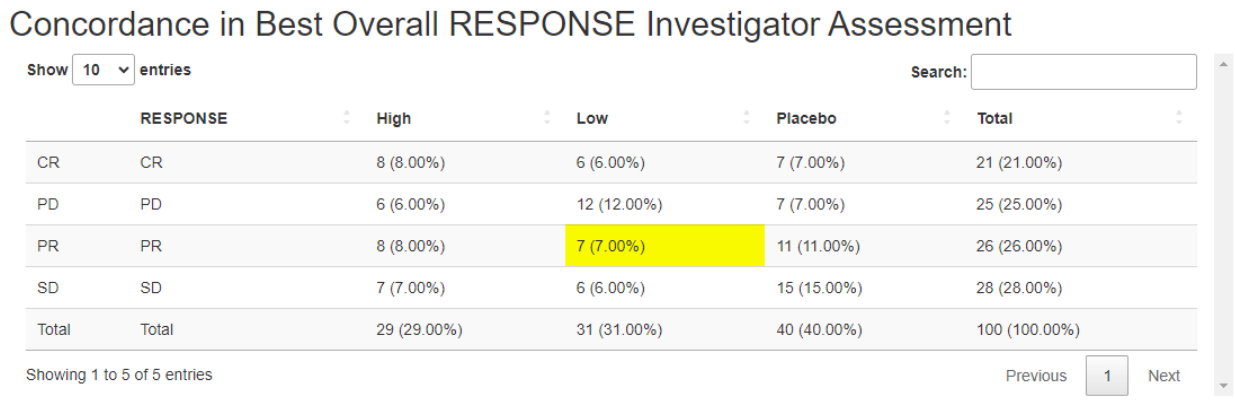
We often come across situations where we review tables and wish to click on certain numbers in the summary table to query the detailed information of the subjects from which those numbers originated. This allows us to gain a clearer understanding of each subject's situation as well as the overall project. What's exciting is that we can easily achieve this functionality using the DT package within the Shiny app by injecting jQuery. Below is an example to describe the implementation process..

SUMMARY TABLE

It provides a user interface (UI) with a table displaying statistical information based on grouping variables.

The UI consists of a title panel that displays the title " Concordance in Best Overall RESPONSE Investigator Assessment " and a main panel that includes a table output area.

Figure 1. Summary table of Best overall RESPONSE Investigator Assessment information



Concordance in Best Overall RESPONSE Investigator Assessment

Show 10 entries Search:

	RESPONSE	High	Low	Placebo	Total
CR	CR	8 (8.00%)	6 (6.00%)	7 (7.00%)	21 (21.00%)
PD	PD	6 (6.00%)	12 (12.00%)	7 (7.00%)	25 (25.00%)
PR	PR	8 (8.00%)	7 (7.00%)	11 (11.00%)	26 (26.00%)
SD	SD	7 (7.00%)	6 (6.00%)	15 (15.00%)	28 (28.00%)
Total	Total	29 (29.00%)	31 (31.00%)	40 (40.00%)	100 (100.00%)

Showing 1 to 5 of 5 entries Previous 1 Next

USER CLICKS ON DIGITAL INTERACTION

The Server logic preprocesses the data by modifying certain columns and merging datasets. It calculates the total count for each unique value of TRT01A and groups the data by RESPONSE and TRT01A, calculating the count and percentage for each combination. The resulting data is then rendered as an interactive datatable using the renderDT function.

Within the renderDT function, JavaScript is used to handle click events on table cells. When a cell is clicked, the JavaScript code retrieves information such as the row index, column index, and cell values. It logs this information and passes it to the Shiny application using Shiny.setInputValue. The clicked cell is visually highlighted by adding a CSS class to it.

The Server code also handles the selected data by filtering the original dataset based on user selections. The filtered data is merged with another dataset and rendered in a separate datatable. A modal dialog is displayed to show the selected data, and a "Dismiss" button is provided to remove the highlighted cell and close the dialog. Overall, the Server part effectively processes and presents the data, allowing users to interact with the datatable and explore selected information in a modal dialog.

Within the renderDT function, a JavaScript callback function is defined using the callback argument of the datatable function. This callback function handles the click event on table cells.

When a cell is clicked, the following steps are executed:

1. A JavaScript callback function is defined within the renderDT function to handle click events on table cells.
2. When a cell is clicked, the callback function retrieves the clicked cell and extracts the row and column indices.
3. The value of the first column (yasix) in the clicked row is obtained.
4. The text content of the clicked column header (group) is extracted.
5. The yasix and group values are logged, passed to the Shiny application using Shiny.setInputValue, and the clicked cell is visually highlighted by adding a CSS class to it.

In summary, the JavaScript code within the callback function captures click events, retrieves relevant information from the clicked cell and column header, logs the values, passes them to the Shiny application, and adds a CSS class to highlight the clicked cell.

Here are two examples. Example 1 can be directly executed in a single Shiny application, while Example 2 is standalone JS code that can be applied to an enterprise-level R Shiny project. Example 2 utilizes bound page elements to retrieve information about the selected cell's row and column. It can be used to retrieve information from any table.

Example 1 :

```
server <- function(input, output) {
  output$tableOutput <- renderDT({
    # Calculate frequency table using ut_freq function
    result_matrix <- ut_freq(adrs, "TRT01A", "RESPONSE")

    # Render the interactive datatable
    datatable(result_matrix, selection = 'none',
        callback = JS("
           table.on('click', 'td', function() {
            var cell = $(this);
            var rowIndex = cell.parent().index();
            var colIndex = cell.index();
            var yasix = table.cell(rowIndex, 0).data();
            var group = table.column(colIndex).header().textContent;
            console.log('Clicked yasix: ' + yasix + ', group: ' + group);
            Shiny.setInputValue('selectedData', { yasix: yasix, group:
      group }, { priority: 'event' });
             // Highlight clicked cell
             cell.addClass('highlighted');
                });
              "))
  })

  # Reactive function for selected data
  selectedData <- reactive({
    input$selectedData
  })

  output$selectedDataTable <- renderDataTable({
    req(selectedData())
    yasix <- selectedData()$yasix
    group <- selectedData()$group

    if (yasix == "Total" & group == "Total") {
      filteredData <- adrs
    } else if (yasix == "Total") {
      filteredData <- adrs[adrs$TRT01A == group, ]
    } else if (group == "Total") {
      filteredData <- adrs[adrs$RESPONSE == yasix, ]
    } else {
      filteredData <- adrs[adrs$RESPONSE == yasix & adrs$TRT01A == group, ]
    }
```

```
    merged_data <- merge(filteredData, TR, by = c("USUBJID","TRT01A",
"VISIT"))
    datatable(merged_data)
  })

  observeEvent(selectedData(), {
    showModal(modalDialog(
      title = "Query Result",
      size = "l",
      class = "my-modal",
      fluidRow(
        column(width = 12, style = "max-height: 800px; overflow-y: scroll;",
               div(class = "my-modal-content",
dataTableOutput("selectedDataTable")))
      ),
      footer = tagList(
        actionButton("dismissButton", "Dismiss")
      )
    ))
  })

  # Remove highlighted class when Dismiss button is clicked
  observeEvent(input$dismissButton, {
    shinyjs::runjs("$('.highlighted').removeClass('highlighted');")
    removeModal()
  })
}
```

Example 2 :

```
$( document ).ready(function() {
    $('.my-modal-content').on('click', 'td', function() {
        var cell = $(this);
        var table = cell.closest('table').DataTable();
        var rowIndex = cell.parent().index();
        var colIndex = cell.index();
        var yasix = table.cell(rowIndex, 0).data();
        var group = table.column(colIndex).header().textContent;
        console.log('Clicked yasix: ' + yasix + ', group: ' + group);
        Shiny.setInputValue('selectedData', { yasix: yasix, group: group },
{ priority: 'event' });
                    cell.addClass('highlighted');
        });
});
```

Figure 2. Subqueries between tables

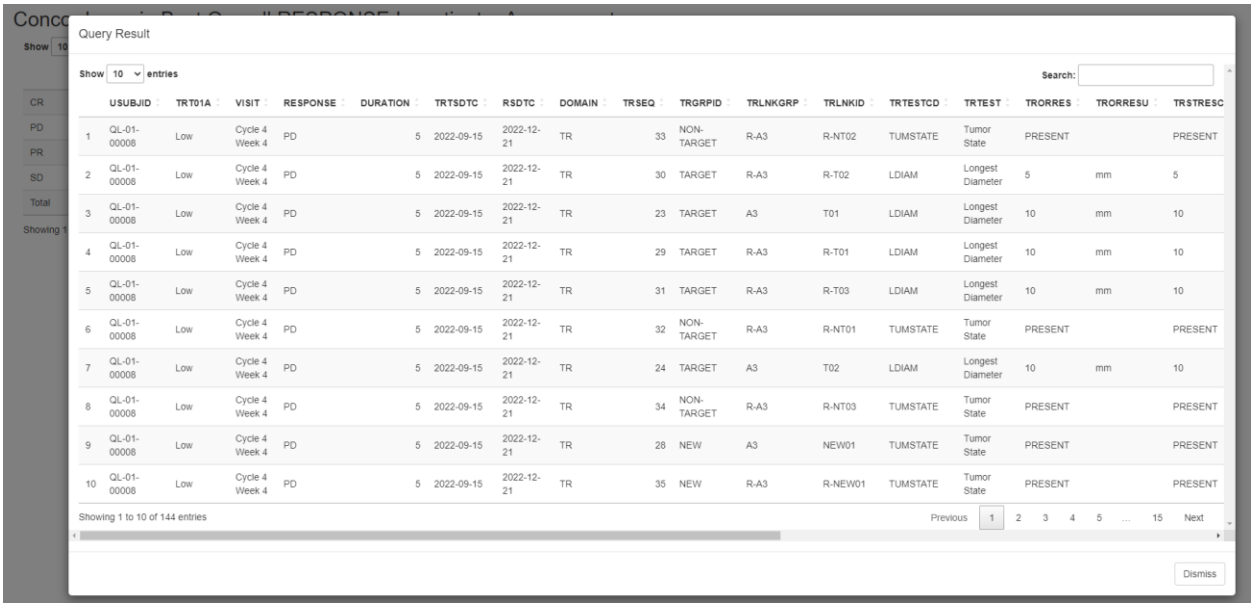


TABLE SPOTLIGHT EFFECT

To achieve a spotlight effect on a specific data point while viewing a large table in a Shiny app, jQuery can be easily employed. The following scenario shows how to implement this spotlight feature using jQuery.

First, we define a data frame called "TR" containing contains tumor examination information for the subject. This data will be visualized in the data table. Moving on to the user interface (UI) section, we use the fluidPage function to create a responsive layout page.

To enhance the user experience, we apply some custom CSS styles to the table cells using the tags$style function. This snippet adds a highlighting effect to the cells when the mouse hovers over them:

```
tags$style(
  HTML("
    .highlight-cell { background-color: #7AC5CD !important; }
    .highlight-row  { background-color: #BBFFFF !important; }
    .highlight-col  { background-color: #BBFFFF !important; }
  ")
)
```

Furthermore, we employ jQuery to handle mouse events and dynamically apply the highlighting effect.

1. Mouseover Event:
   a) When the user hovers the mouse over a table cell (`td`), the code triggers the mouseover event.
   b) It identifies the column index of the current cell and increments it by 1 to match jQuery's selector syntax.
   c) The code adds the "highlight-row" class to the closest row of the current cell, highlighting the entire row with a light blue background.

d) Additionally, it adds the "highlight-col" class to all cells in the current column, creating a light blue background for the entire column.

e) To avoid the current cell being part of the column highlight, the code removes the "highlight-col" class from the current cell.

f) Finally, the code adds the "highlight-cell" class to the current cell, applying a dark blue background to emphasize it.

2. Mouseout Event:

a) When the user moves the mouse out of a table cell, the mouseout event is triggered.

b) The code retrieves the column index of the current cell and increments it by 1 to match jQuery's selector syntax.

c) It removes the "highlight-row" class from the closest row of the current cell, removing the row's highlight effect.

d) Similarly, it removes the "highlight-col" class from all cells in the current column, removing the column's highlight effect.

e) Lastly, the code removes the "highlight-cell" class from the current cell, eliminating the distinct highlight effect.

3. Overall Effect:

a) With the above implementation, the code dynamically highlights the entire row and column while hovering over a cell, creating a visually interactive experience.

b) The current cell receives a separate highlight to draw attention to it, ensuring a clear focus on the specific cell.

c) When the mouse moves away from the cell, the highlights are removed, returning the table to its normal appearance.

The following code snippet, added within the tags$script function, accomplishes this:

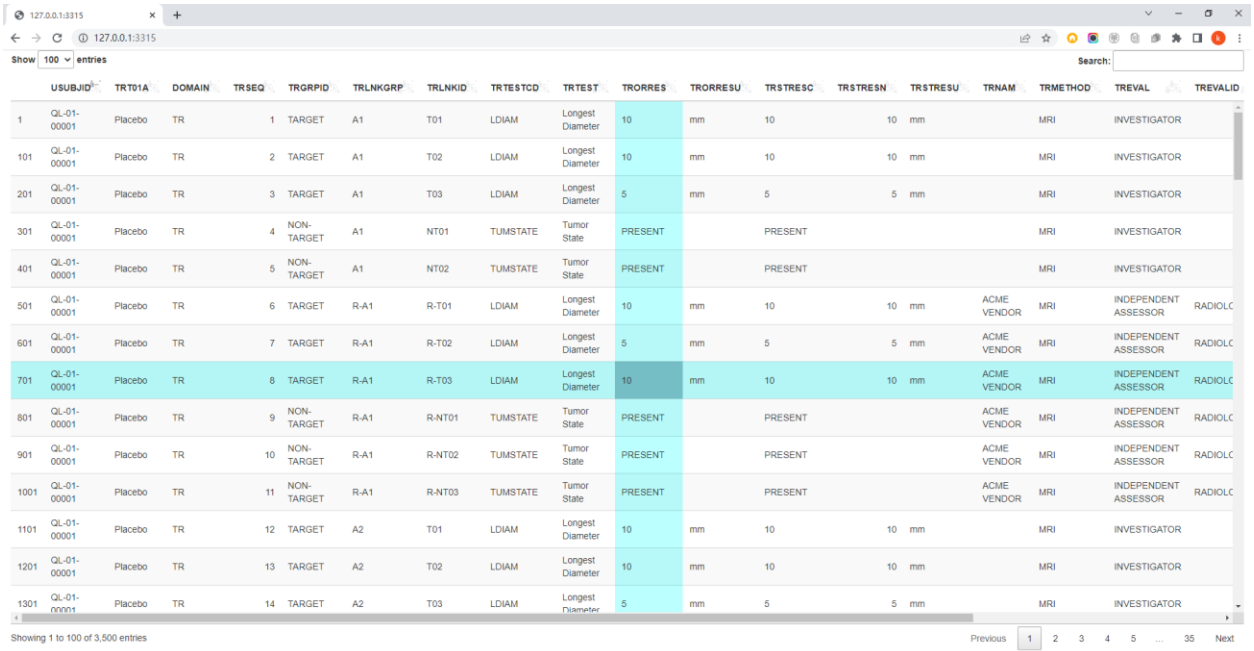
```
tags$script('
  $(document).ready(function(){
    $(document).on("mouseover", "td", function(){
      var colIndex = $(this).index() + 1;
      $(this).closest("tr").addClass("highlight-row");
      $("td:nth-child(" + colIndex + ")").addClass("highlight-col");
      $(this).removeClass("highlight-col");
      $(this).addClass("highlight-cell");
    });

    $(document).on("mouseout", "td", function(){
      var colIndex = $(this).index() + 1;
      $(this).closest("tr").removeClass("highlight-row");
      $("td:nth-child(" + colIndex + ")").removeClass("highlight-col");
      $(this).removeClass("highlight-cell");
    });
  });
')
```

In the server section, the renderDataTable function is used to render the "adsl" data frame into an interactive data table. Lastly, we combine the UI and server components using the shinyApp function to create a complete Shiny application.

By integrating these code snippets, you can build a compelling interactive data table application. Users will have the ability to explore and interact with the data, while the added CSS and jQuery code provides an engaging highlighting effect, enhancing the visual experience.

Figure 3. Cross spotlight effect in the table



CONCLUSION

These two small jQuery examples have the power to work wonders for a shiny app, significantly improving its usability and making it more intuitive for users. The subquery functionality and table spotlight effect provided by jQuery make the app more user-friendly, enhancing the overall user experience.

REFERENCES

DT: An R interface to the DataTables library

https://rstudio.github.io/DT/

SDTM Examples for Oncology Use Cases

https://wiki.cdisc.org/download/attachments/28453863/SDTM%20Examples%20for%20Oncology%20Use%20Cases.xlsx?version=1&modificationDate=1443373028095&api=v2

ACKNOWLEDGMENTS

Special thanks to Yingying Yan for the suggestions on fine-tuning the sub-query algorithm and to Tao Zhang for the valuable feedback on the paper.

Special thanks to the shiny team and the statistical programming team for their outstanding support and collaboration.

CONTACT INFORMATION

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

Name :Hengyi Yang
Qilu Pharmaceutical Co., Ltd
+86 15067820117
hengyi.yang@qilu-pharma.com