

Interact with your data and create interactive plots with R Shiny

Dan Feng, Pfizer (Wuhan) Research and Development Co., Ltd, Wuhan, China

ABSTRACT

Shiny is an R package that makes it easy to straightly build interactive web apps from R. The scope of this presentation is to introduce how R shiny app helps with data exploration and data visualization. It will take the model-based meta-analysis (MBMA) visualization tool as an example to explore and visualize MBMA data. This Shiny app enables to import and export the dataset, subset the dataset by row and column, display summary statistics, check the data availability. In the meantime, after selected target endpoint and graph parameters, this tool will automatically create interactive scatter plot, box plot, longitudinal plot, forest plot, funnel plot, and heat map.

INTRODUCTION

Shiny enables statistical programmers to build up interactive web applications by simply using R language without JavaScript. In the meantime, it can be also written directly in HTML, CSS, and JavaScript for more flexibility. Shiny is designed for fully interactive visualization, it instantly updates itself whenever the user makes a change.

This paper focuses on the “interaction” with data and plots. It uses the MBMA visualization tool as an example to demonstrate how to link the data with plots, how to create different types of interactive plots. Instead of generating one-time tables, listings and figures when data changes, shiny app provides a new option to visualize the data with user friendly interaction interface and high efficiency.

HOW TO BUILD A SHINY APP

The basic structure of a shiny app contains three components: user interface (UI), server instructions and shinyApp which combines UI and server into an app. UI nests R functions that assemble an HTML user interface for your app. Server is a function with instructions on how to build and rebuild the R objects displayed in the UI. Below is the template for a shiny app:

```
library(shiny)
ui <- fluidPage()
server <- function(input, output) {}
shinyApp(ui = ui, server = server)
```

For building up an interactive shiny app, it should include inputs and outputs. Inputs collect values from the user and outputs add R output to the UI (Figure 1). There are various forms of input (Figure 2) such as select box, radio buttons, check box, etc. You can pick up any input by simply using available shiny functions when design the user interface. The outputs of the shiny apps include interactive table, text, plot, and image.

Once all of the components are stored in a folder just run app.R then you can get access to your app. You can also publish the app at shinyapps.io (free) or your own server. After publishing the app, it's easy to get access to the app with a website link.

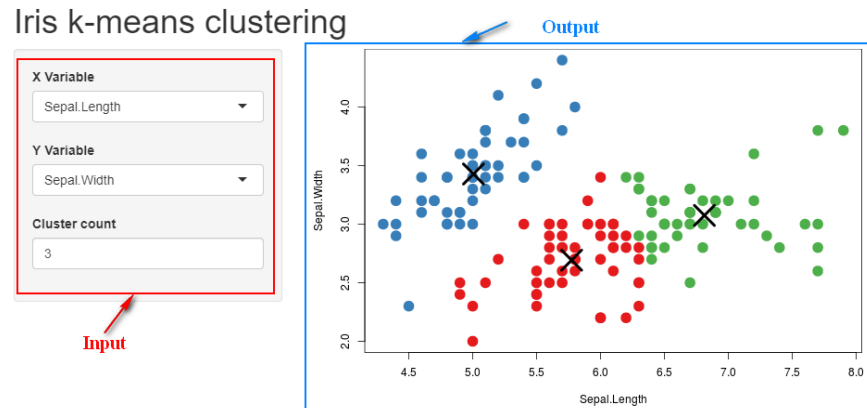


Figure 1. Shiny User Interface with Input and Output

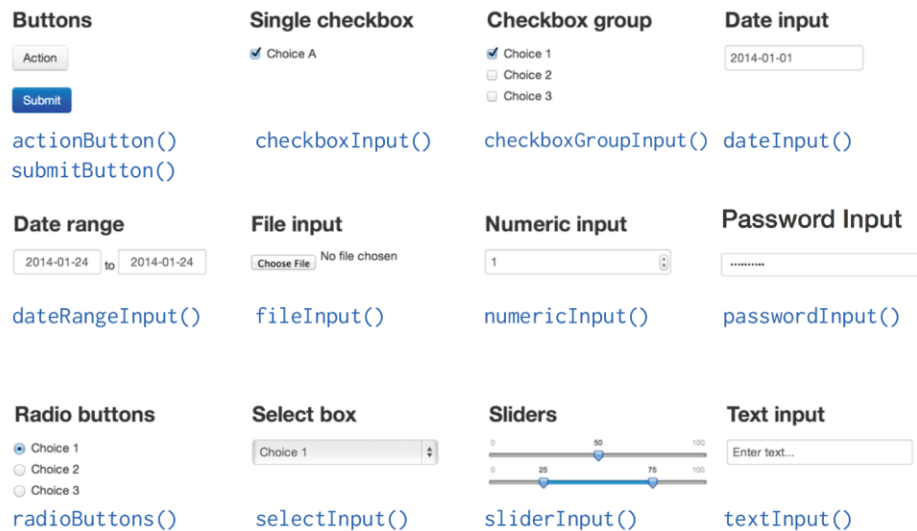


Figure 2. Shiny Inputs

MBMA VISUALIZATION TOOL OVERVIEW

The MBMA visualization tool is a shiny app used to visualize the model-based meta-analysis curation data. The meta-analysis data assembles large collection of analysis results from individual studies for integrating the findings. The curated data file is from the systematic review of literature through standard data format definitions. The study information, data source, treatment, demographic, endpoint and indication are collected in standard format. The tool enables the user to load the curated data for different indications, then explore and visualize the data.

INTERACT WITH DATA

DATA IMPORT AND OVERVIEW

The app starts with uploading the input data from the local computer through the file input box. The fileInput() function loads the data in multiple format, such as excel, csv, R dataset. After the data is loaded, a text about data set information including indication, endpoints, drugs and number of literatures is displayed. The text output is assembled by using HTML elements with tags. The blue text changes if

different file is loaded. InfoBox function (Input Data Overview) is a more straightforward way to present the data summary. Figure 3 is the example for data import and overview.

Example code for infoBox function:

```
infoBox(title = "Number of publications", value = nLit, width = 6,
color="blue", icon = shiny::icon("book"))
```

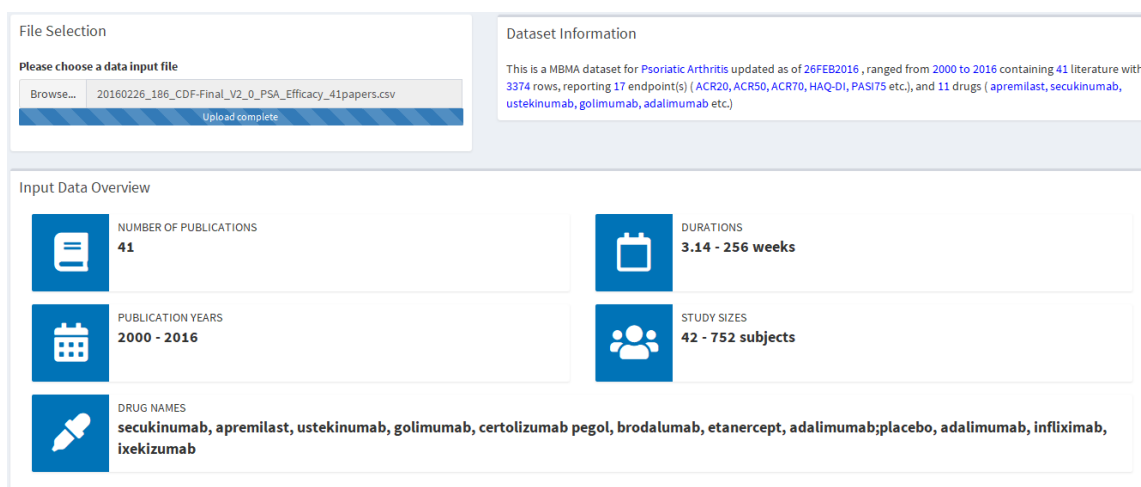


Figure 3. Data Import and Overview

DATA VALIDATION

Since the input files are in the standard format, it validates the data by cross checking with the general data definition and checking the availability and characteristics of the key variables which are used in the plot sessions. These text output for data validation is also implemented by using HTML.

Example code for using HTML to generate text output:

```
paste("<p><span style='color:red'>1.Required variables which are not  
existed in current import data:</span><span style='color:blue'>  
",lack_varString, "!</span></p>")
```

DATA SUBSET AND TABLE VIEW

The app provides flexibility to define the key variables such as endpoints and response values right after the data is loaded even if they are not in standard format. User can define the columns which contains the endpoints and response values by picking up variables from the dropdown list (Figure 4).

The figure shows a form with four sections for selecting key variables:

- Choose column containing endpoint(s):** A dropdown menu with "ENDPOINT" selected.
- Choose Endpoint name(s):** A text input field.
- Choose column containing Response Field:** A dropdown menu with "CHBSL_VAL" selected.
- Select a normalizing unit for time:** A dropdown menu with "Week" selected.

Figure 4. Select Key Variables for the App

To subset the data set (Figure 5), the select box with a dropdown list of the variables names enables the user to choose the variable(s) to subset. The corresponding variables with their available values will

automatically pop up right after they have been selected as the variable to subset. Slider bar is used to control the values for numeric variables and selection box is used to choose the values for character variables.

Example code for create slider input and selection box input:

```
if (is.numeric(unlist(Dataforfilter()[[i]])) == FALSE) {
  selectizeInput(paste0('filter_row', i),
    label=i,
    choices=unique(Dataforfilter()[[i]]),
    selected=NULL,
    multiple=TRUE) }
else{
  sliderInput(paste0('filter_row', i),
    label=i,
    min=min(Dataforfilter()[[i]], na.rm=TRUE),
    max=max(Dataforfilter()[[i]], na.rm=TRUE),
    value=c(min(Dataforfilter()[[i]], na.rm=TRUE), max(Dataforfilter()[[i]], na.rm=TRUE))) } }
```

Select the variable(s) to subset the data by

ARM.AGE ARM.DOSE

The input data for all plot views will be based on the Endpoint(s) selected in the left hand pane and this subset of the data.

ARM.AGE

43.5 44.5 45.5 46.5 47.5 48.5 49.5 50.5 51.5 52.5 53

ARM.DOSE

Download data subset for personal use

Figure 5. Data Subset

Figure 6 is the example of interactive data table output. The data table refreshes automatically when user subset the data in figure 5. User can choose variables to view and use the filter box under the variable name to check the data availability.

Example code for create the interactive data table output:

```
output$filter_Data<- DT::renderDataTable(
  DT::datatable(Datafilter1()[c(1:nrow(Datafilter1()))], input$variable),
  filter = 'top', escape = FALSE, options = list(pageLength = 25,
    scrollX='500px', autoWidth = TRUE)) )
```

Select variables to view in the table of subsetting data below:

ARM.TIME1 ARM.TIME1U ARM.TRT DSID ENDPOINT RSP.VAL

Show 10 entries

Search:

	ARM.TIME1	ARM.TIME1U	ARM.TRT	DSID	ENDPOINT	RSP.VAL
15	1	WK	placebo	5	ACR20	6.85122
16	2	WK	placebo	5	ACR20	14.2152
17	3	WK	placebo	5	ACR20	15.0316
18	4	WK	placebo	5	ACR20	16.2572
19	8	WK	placebo	5	ACR20	21.3646
20	12	WK	placebo	5	ACR20	25.3466
21	16	WK	placebo	5	ACR20	18.2801
22	20	WK	placebo	5	ACR20	16.1241
23	24	WK	placebo	5	ACR20	15
24	1	WK	secukinumab	5	ACR20	12.1715

Showing 1 to 10 of 540 entries

Previous 1 2 3 4 5 ... 54 Next

Figure 6. Data Table View

CREATE INTERACTIVE PLOTS

INPUT AND OUTPUT OF THE INTERACTIVE PLOTS

Figure 7 is the example about the input and output of the interactive plot. The inputs include choose x-axis, color group variables, ranges for both x and y axes, and labels and title for the plot etc. The plot output and data table output will instantly update according to selections. In addition, the click of row in the data table will directly link to the point in the plot.

Example code for create the interactive plot:

```
plot <- ggplot(plotData, aes(TIME, GRAPHCOL,color=END)) +  
  geom_point() +  
  geom_line(aes(linetype=lineGroup)) +  
  scale_x_continuous(input$plot_lab_x, plotData$TIME) +  
  scale_y_continuous(ylab) +  
  ggtitle(title) +  
  coord_cartesian(xlim = input$xtime, ylim = input$endrange)+  
  labs(col='ARM.DATA',size='N.ARM')
```

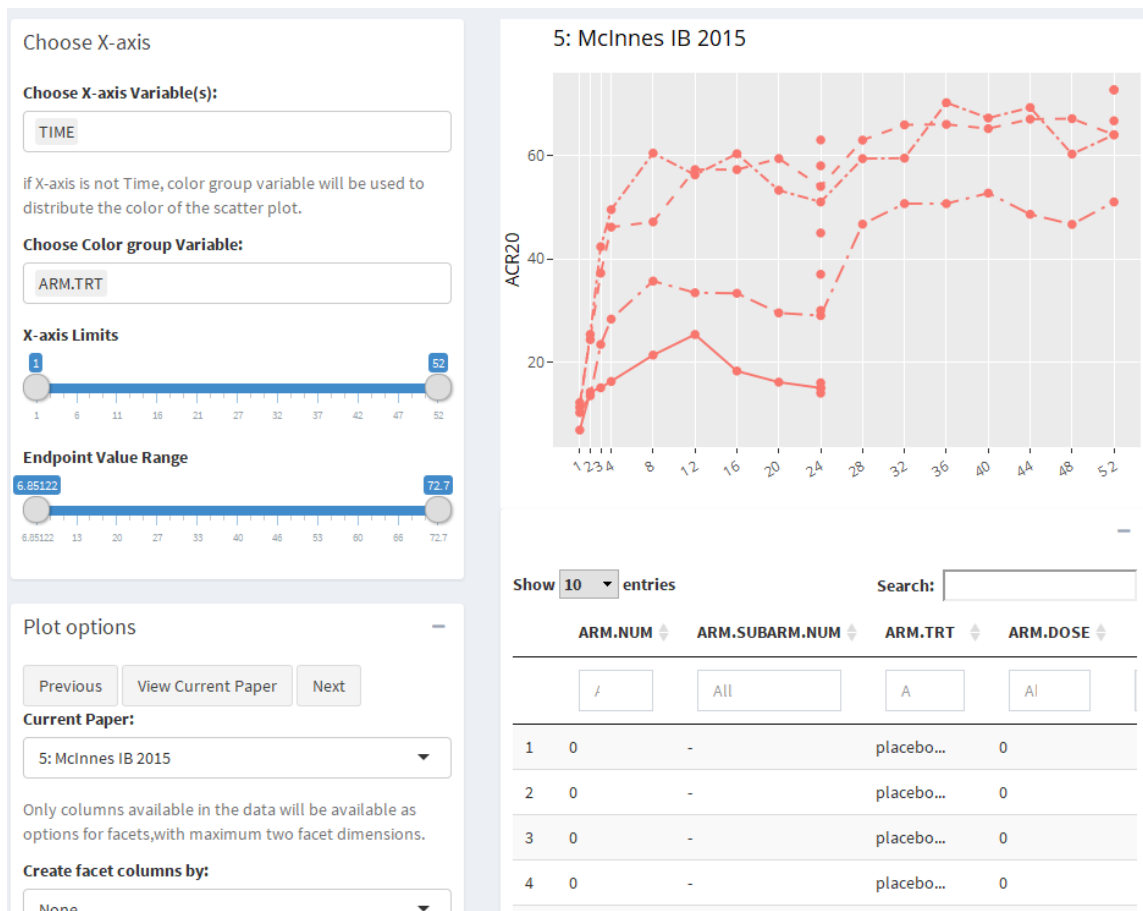


Figure 7. Input and Output of Time vs. Endpoint Plot

INTERACT WITH PLOT BY USING PLOTLY

Plotly is an R package for creating interactive web-based graphs via the open source JavaScript graphing library plotly.js. It converts your static plots to an interactive web-based version. Figure 8 and Table 1

shows the plotly graph functions. The hover function helps better view the plot and one click download button enables to download the plot in png format.

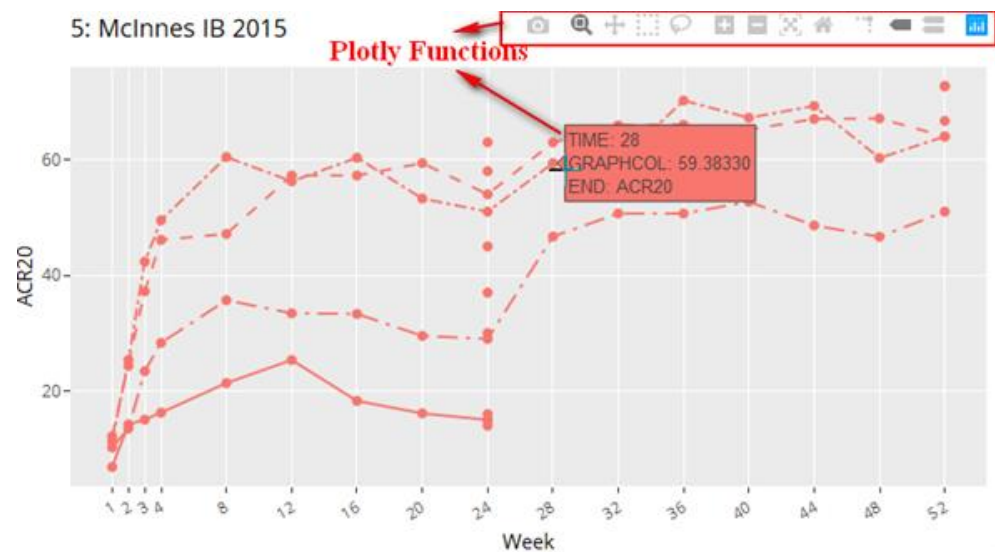


Figure 8. Plotly Graphs Functions









	Download Plot as PNG		Autoscale
	Zoom		Reset Axes
	Pan		Show Closest Data on Hover
	Zoom In/ Zoom Out		Compare Data on Hover

Table 1. Plotly Graphs Functions

CREATE PLOT WITH FACET

You can also lay out the panels in a grid by using the `facet_grid()` function. It forms a matrix of panels defined by row and column faceting variables. Figure 9 is the example for the plot which facet by row (dose) and column (treatment). The selection box input provides list of variables that can be selected as facet row and column.

Example code for facet:

```
if(input$facetRows == "None" & input$facetCols == "None"){plot}
else {plot<-plot + facet_grid(facet)}
```

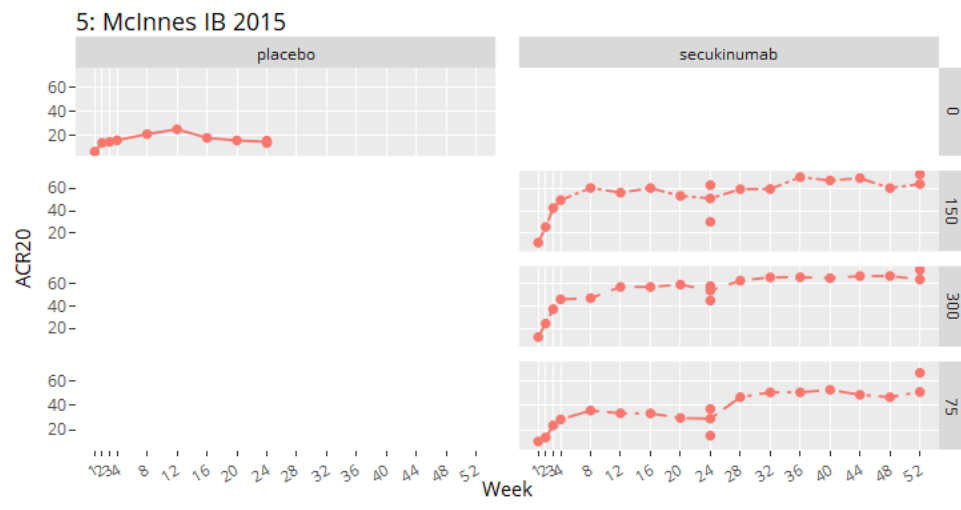


Figure 9. Plot with Facet

OTHER INTERACTIVE PLOTS

The follow examples are other types of interactive plots.

- Scatter Plot (Figure 10)



Figure 10. Scatter Plot

- Box Plot (Figure 11)

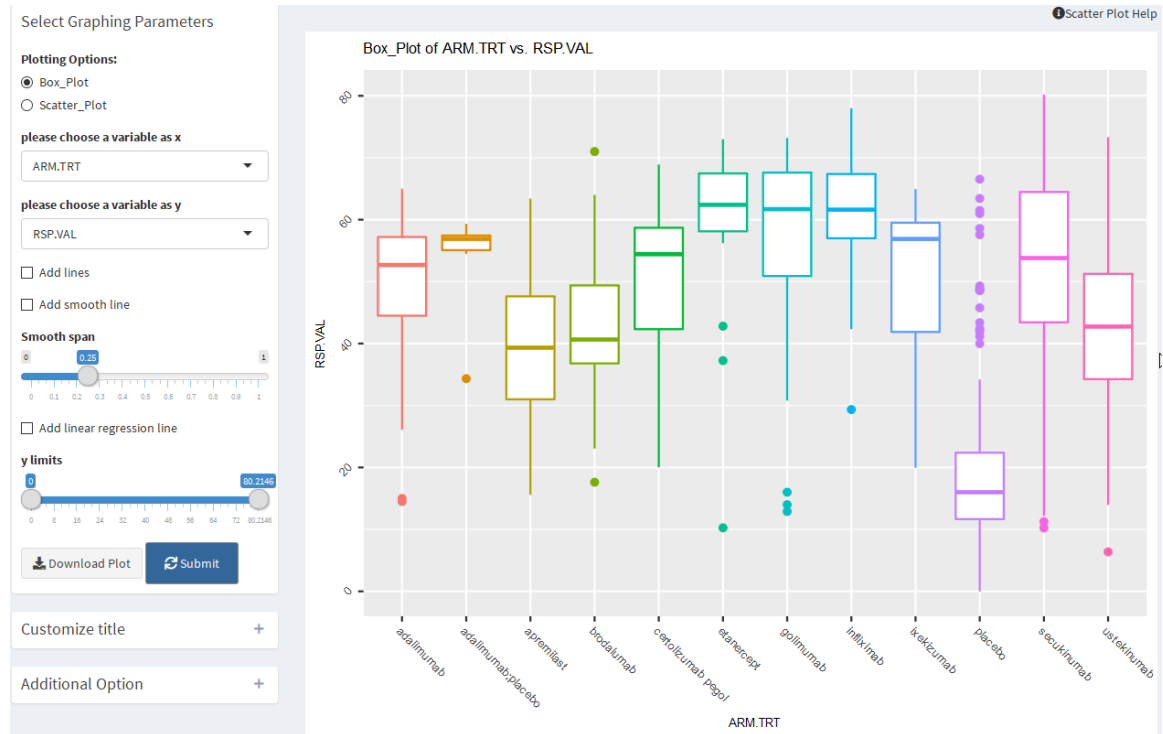


Figure 11. Box Plot

- Forest Plot (Figure 12)

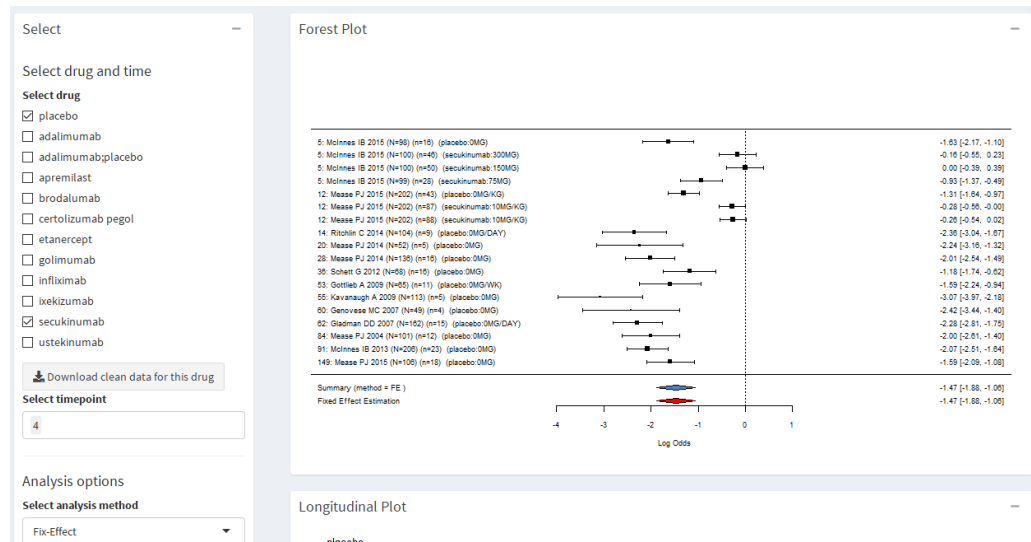


Figure 12. Forest Plot

- Heat Map (Figure 13)



Figure 13. Heat Map

CONCLUSION

Although this example is based on the meta-analysis data, using shiny to create the interactive data table and plots are also applicable for clinical trial data. Take oncology studies as example, team may request bi-weekly or monthly safety and efficacy data review. The waterfall plot, spider plot, histogram plot and Kaplan-Meier plot can also be created by shiny apps. Once the shiny app is developed, the team members can easily visualize the data by loading the latest data. Not only for oncology studies, shiny app is also the powerful tool to visualize the data at the compound or asset level. For example, since pooled data is available for a compound from several trails, shiny app can create the interactive plots by different subgroups, dose levels and compare the endpoints across subgroups.

This paper gives examples about various shiny inputs and outputs, demonstrates different types of interactive plots to explore and visualize the data. It provides an interactive way to explore the data and visualize the data.

REFERENCES

<https://shiny.rstudio.com/>
<https://plot.ly/r/getting-started/>
https://ggplot2.tidyverse.org/reference/facet_grid.html

ACKNOWLEDGMENTS

Thanks for the guidance and support from the MBMA working group of Pfizer Clinical Pharmacology department. Thanks to Gaoli Du, Fan Xiao and Jingwei Huang for supporting the development the Shiny App. Thanks to James Kim and Maggie Wang for review and comment of the tool.

CONTACT INFORMATION

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

Dan Feng, M.S.,
 Pfizer (Wuhan) Research and Development Co., Ltd
Dan.Feng@pfizer.com