PharmaSUG China 2021 - Paper DV-39

Utilizing Plotly in R to Review Clinical Data with Interactive Plots

Kailu Song, Beigene (Beijing) Co., Ltd.

ABSTRACT

Visualization is always a significant method to monitor the clinical trial process and review data. R shiny has been widely used by pharmaceutical companies to explore deep in data during the trial execution phase. Plotly is one of the best visualization packages which works directly with R shiny to build flexible, interactive, ready-for-publication, and high-quality graphs. Plotly is an open-source package built from JavaScript, html and css, and is compatible with multiple tools, including not only R but also python, MATLAB, and JavaScript etc. This paper demonstrates how to use Plotly in R to generate basic interactive plots using dummy ADaM dataset and how these plots communicate with R shiny app, and provides an example of how an R shiny app reviews clinical data using Plotly figure.

INTRODUCTION

Reviewing clinical data reports can be a tedious and time-consuming process, even with the standard plots. However, with the help of interactive plots, the time of data reviewing can be largely reduced, and the data issues can be identified and thus resolved more efficiently. Plotly is such a package designed to build interactive plots with just a few lines of code and adapts to multiple platforms. Its JavaScript-based functionality makes it an ideal tool for displaying big data on the website. Moreover, Plotly creates numerous readable and high-quality plots for different scenarios with customized attributes and build-in formulas which also allows adjusting the view of the output plots flexibly. In addition, Plotly has an online open-source graphing library where you can find examples of various plots code that can directly adapt to your dataset. The paper briefly introduces the basics of Plotly in R studio, how to use Plotly to demonstrate clinical data, as well as how to make flexible adjustment to the plots. Plotly normally works with R shiny app to produce the interactive web app, so I also give an example about using Plotly figure to demonstrate clinical efficacy data and enabling the plot to communicate with R shiny app. What is worth noticing is that this paper only focuses on how to produce Plotly figures instead of building an entire R shiny app.

PLOTLY BASICS

1. INTRO TO PLOTLY

Plotly is an R package which can be installed using CRAN, and you should load the package at first. Then we can create a Plotly object by using function plot_ly(). Below is a simple example of a scatter plot (shown in Figure 1). I use the build-in dataset BOD in R, and Plotly will find a reasonable way to represent the dataset if a trace type is not provided; in this case, Plotly represents the data in a scatter plot. According to the Figure 1, with a simple one-line code, the plot automatically adds hover labels in the figure where you can reveal the point data by moving the cursor over the point and having the label show up. In addition, on the top right corner of Figure 1, you can see Plotly add those interactive features to the plot, for example, saving the image in a static format, zooming in/ zooming out the plot, and toggling spike lines, to name but a few.

```
install.packages("plotly")
library("plotly")
plot_ly(BOD, x =~Time,y =~demand)
```

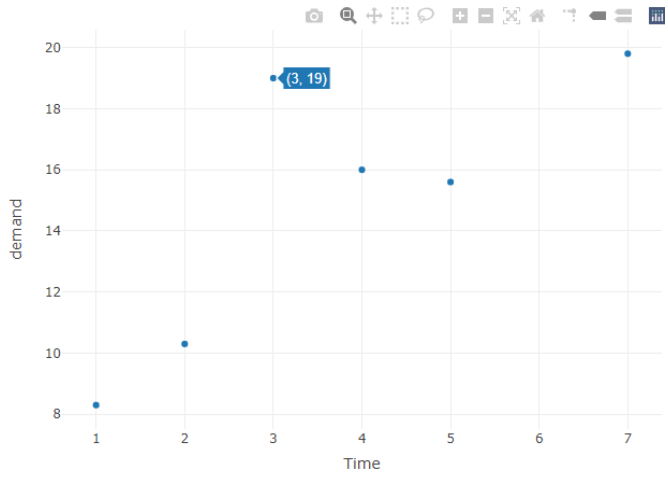


Figure 1. Figure 1Plotly Sample Plot

2. BASE PLOTS

There are mainly two functions in Plotly package that allow the users to output customized interactive plots, which are plot_ly() and layout() functions. The function plot_ly() is used to initiate a Plotly visualization, while layout() is used to customize the layout of the plot. There is another type of function in Plotly used frequently-the add_*() function, such as add_trace(), add_text(), add_markers(), add_lines(), add_paths(), and add_segments(). Those add_*() functions are applied to add scatter-based plot on the top of the Plotly object, which give us a more flexible interface to some special cases of the scatter trace. Other than plot_ly(), plot_geo() and plot_mapbox() are also functions to initiate Plotly plot with a map-based layer, since we do not normally use those two functions in clinical trial, we will not discuss more in this paper. In this section, I will demonstrate Plotly figures using dummy ADaM dataset.

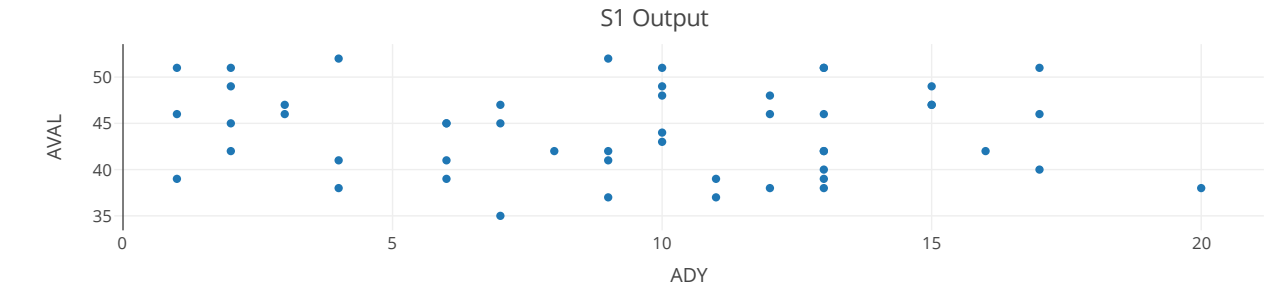
A. Scatter and Line Plot

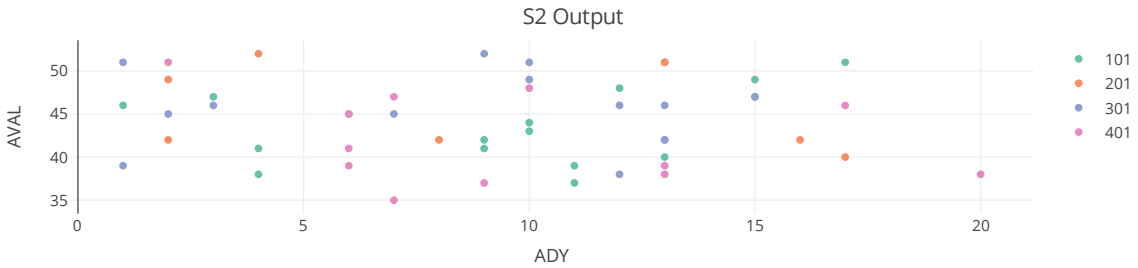
Scatter plot can be useful during laboratory data reviewing. The example below illustrates the relationship between analysis relative day and laboratory value from ADLB. You can produce scatter plot by specifying input dataset, variable names for x and y axis in plot_ly() function, as shown in S1. You can also output the scatter plot which has a group variable, as shown in S2. If I attempt to differentiate the laboratory value between multiple sites, I put the group variable SITEID in the color argument, so the different sites display different colors in the plot automatically. You can also use a self-defined color for group variable by defining the colors argument in S3, but you need to ensure that the number of self-defined colors equals the number of groups in the group variable. In this case, I have four sites, so I need to specify four colors in the function. Instead of using color to differentiate the groups, you can also use the symbol in S4. I also add a title and labels for two axes by adding the layout() function. The detail layout for X and Y axis needs to be in a list format, in which you can modify the title, font size, range, and ticks, etc. Apart from changing the color and shape of the scatter plot, you can also change the size. As shown in S5, I use AVAL to define the dot size, and the higher the value is, the bigger the dot size is. You can double click the legend and remove sites or just display the selected sites, which is the benefit of the interactive plot. You can edit the hover label by adding the text argument in S6.

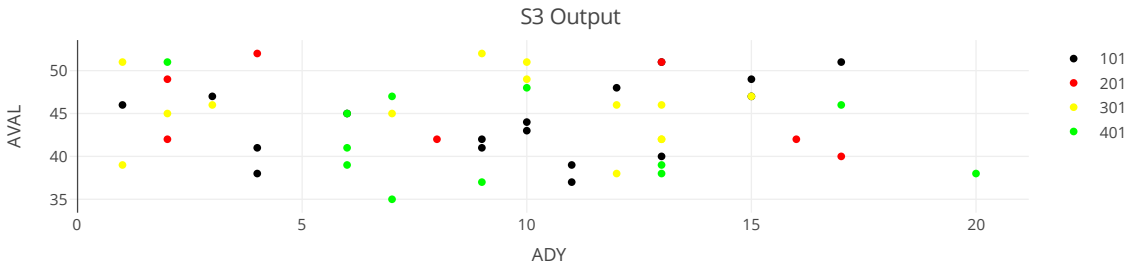
The source code for scatter plots is shown below and the corresponding plots are shown in Figure 3:

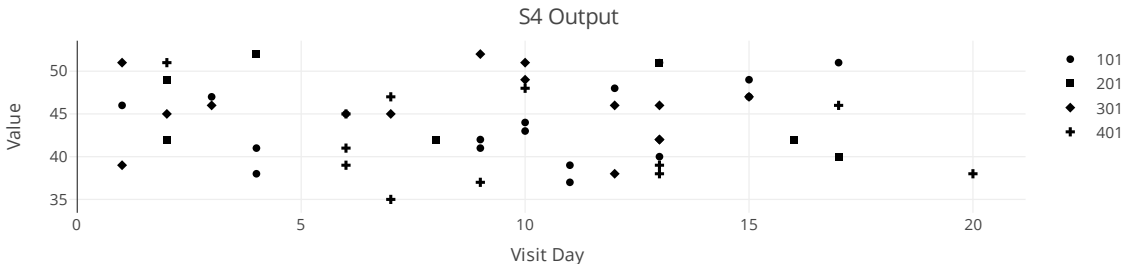
```
S1 <- plot_ly(data=adlb, x=~ADY, y =~AVAL)
S2 <- plot_ly(data=adlb, x=~ADY, y =~AVAL, color = ~SITEID)
S3 <- plot_ly(data=adlb, x=~ADY, y =~AVAL, color = ~SITEID,
      colors = c("black","red","yellow","green"))
S4 <- plot_ly(data=adlb, x=~ADY, y =~AVAL, symbol =~SITEID,
```

```
        symbols = c("circle","square","diamond","cross"),
        color = I("black")) %>%
        layout( title = "S4 Output",
                xaxis = list(title = "Visit Day"),
                yaxis = list(title = "Value"))
S5 <- plot_ly(data=adlb, x=~ADY, y =~AVAL, color = ~SITEID, size =~AVAL)
S6 <- plot_ly(data=adlb, x=~ADY, y =~AVAL, color = ~SITEID, size =~AVAL,
                text = ~paste0("SITE:",SITEID, "\n","VALUE:", AVAL))
```









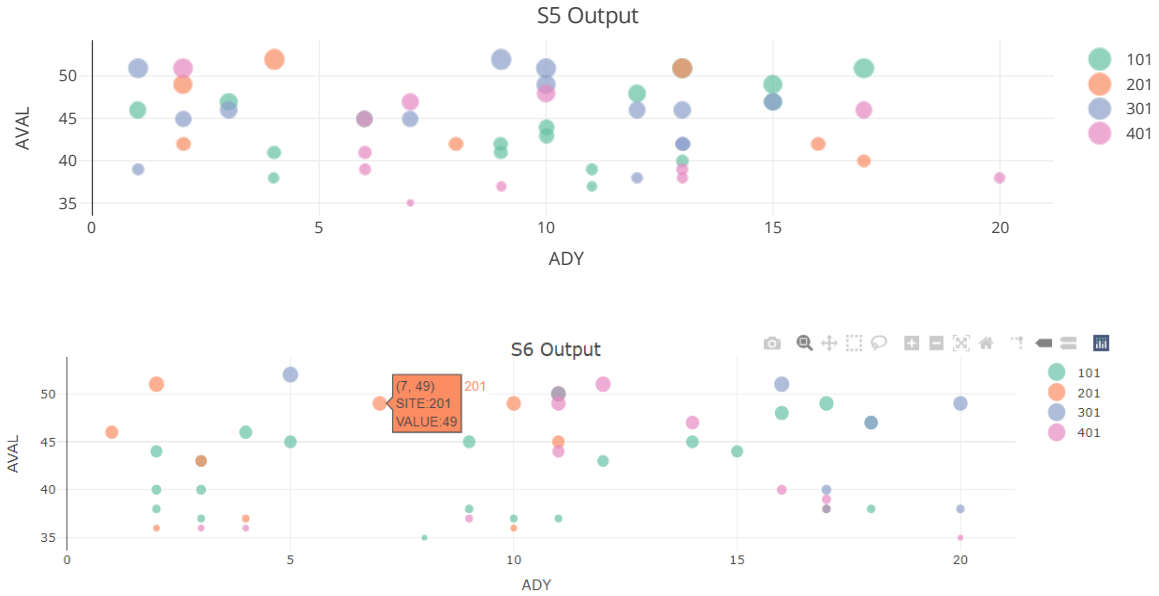


Figure 2. Scatter Plot Examples

Line plot can be drawn using the idea of scatter plot, where you change the mode to lines, since we have multiple sites, it will be helpful if we put all the site value together in the same plot, in such case, we can use add_trace() function on the top of plot_ly() function. In the example below, I first filter out ADLB datasets by site ID, and then, I define a Plotly object by using plot_ly() function without specifying any arguments. Plotly functions work with pipe (%>%), so we can use consecutive pipe to define each induvial plot and Plotly will put the plots in one figure for us. I put three different traces in the same figure: The first one is a line, the second is a line with marker, and the last is a scatter plot, in which we can control the type of figure by changing the mode argument. The source code is shown below with output figure:

```
adlb_1site <- adlb %>%
              filter(SITEID == "101")
adlb_2site <- adlb %>%
              filter(SITEID == "201")
adlb_3site <- adlb %>%
              filter(SITEID == "301")
L1 <- plot_ly() %>%
      add_trace(data=adlb_1site, x=~ADY, y =~AVAL, name="Site 1",
                mode = "lines") %>%
      add_trace(data=adlb_2site, x=~ADY, y =~AVAL, name="Site 2",
                mode = "lines+markers") %>%
      add_trace(data=adlb_3site, x=~ADY, y =~AVAL, name="Site 3",
                mode = "markers")
```

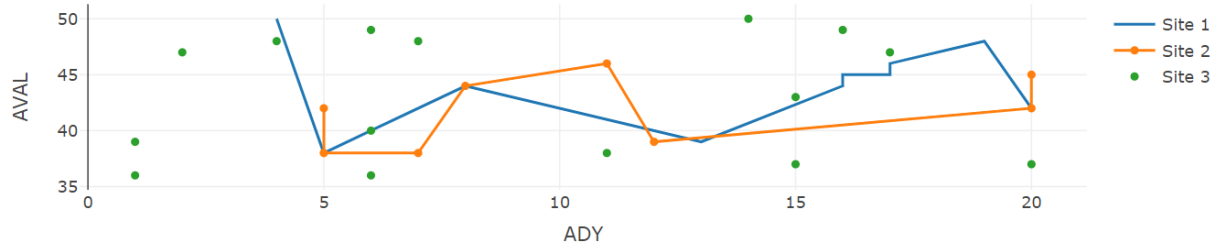
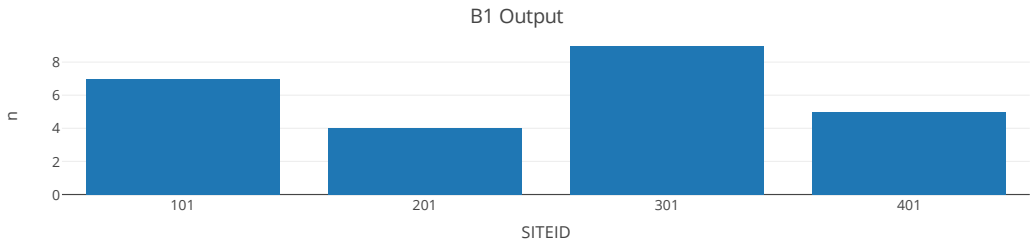


Figure 3. Line Plot Example

B. Bar Plot

Another useful plot type is bar plot, which is often used to compare count value among different groups. In the examples below, I use ADSL dataset to compare sex within or between sites. We can output bar plot by specifying the type to the 'bar' as in B1. If we want to match more than one bar to each site, just add another bar plot using pipe in B2. If you want to have stack bar plot, just set the layout bar mode to "stack" in B3. I can also display the number of count for each bar code by adding text and setting the text position through text position argument in B4.

```
adsl_sex <- adsl %>% group_by(SITEID,SEX) %>% count()
adsl_F <- adsl_sex %>% filter(SEX == "F")
adsl_M <- adsl_sex %>% filter(SEX == "M")
adsl_t <- adsl_sex %>% mutate(SITEID) %>% spread(SEX,n) %>%
          rename(female = F,male = M)
B1 <- plot_ly(data=adsl_F, type = "bar", x=~SITEID, y = ~n)
B2 <- plot_ly(data=adsl_t) %>%
      add_trace(type = "bar", x=~SITEID, y =~female, name = "Female") %>%
      add_trace(type = "bar", x=~SITEID, y =~male, name = "Male")
B3 <- plot_ly(data=adsl_t) %>%
      add_trace(type = "bar", x=~SITEID, y =~female, name = "Female") %>%
      add_trace(type = "bar", x=~SITEID, y =~male, name = "Male") %>%
      layout(barmode = "stack")
B4 <- plot_ly(data=adsl_t) %>%
      add_trace(type = "bar", x=~SITEID, y =~female, name = "Female",
                text = ~female, textposition = "auto") %>%
      add_trace(type = "bar", x=~SITEID, y =~male, name = "Male",
                text = ~male, textposition = "auto") %>%
      layout(yaxis = list(title = "Number of Subjects"),
             xaxis = list(title = "Site ID"))
```



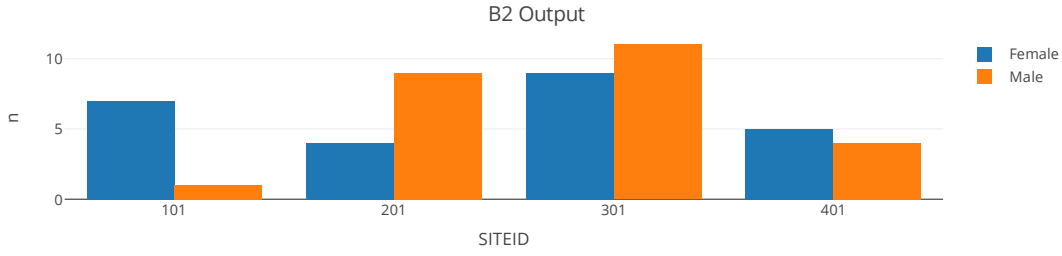
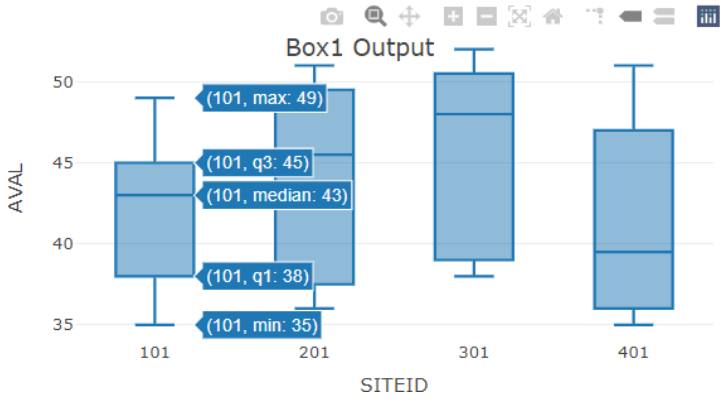


Figure 4. Bar plot examples

C. Box Plot

Box plot can be easily drawn by Plotly without any statistical calculation. As illustrated in the example Box 1, I only need to put the input dataset and specify X and Y variables and set the type to the “box”. Then Plotly will do calculation automatically and in the interactive plot, you can see the results by putting the mouse over the box in Box 1. You can also change the box direction by reversing the X and Y axis as shown in Box 2. If you need to add jittered points, you can follow the example of Box3. You can display either all the points or only the outliers. By default, as Plotly uses a linear method of computed quartiles, you can also choose to use an exclusive or inclusive algorithm by setting quartile method argument to “exclusive” or “inclusive”.

```
Box1 <- plot_ly(data=adlb, x =~SITEID, y =~AVAL, type = "box")
Box2 <- plot_ly(data=adlb, y =~SITEID, x =~AVAL, type = "box")
Box3 <- plot_ly(data=adlb, x =~SITEID, y =~AVAL, type = "box",
                boxpoints = "all", jitter = 0.5, pointpos = -2)
```



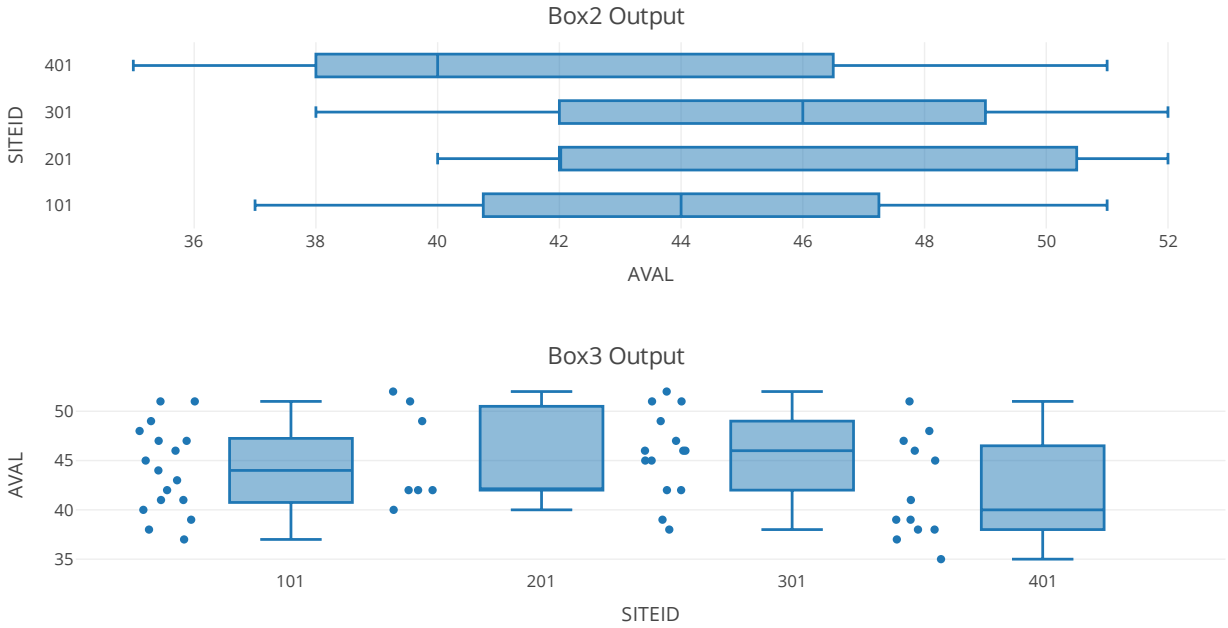


Figure 5. Box Plot Examples

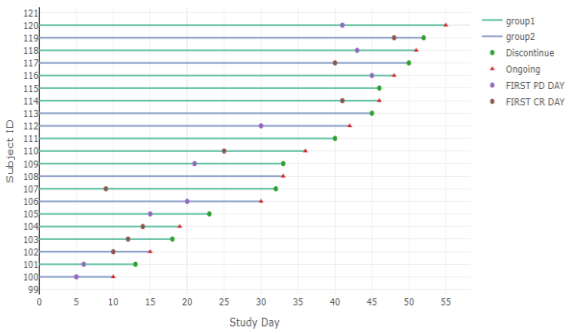
APPLY PLOTLY IN R SHINY APP

Plotly function can create an interactive plot used in an R shiny app. In the example below, I build a simpler shiny app, which displays the subjects treatment status and tumor evaluation status. In this app, when you click a scatter point in the figure, another table will be displayed, showing the detail information about this subject. Plot_ly function provides a method for user to communicate with other elements, a reactive table in this example, in a shiny app. We first register the Plotly by adding source argument and key argument in the plot_ly() function. In this figure, I set the source name of the figure as “plotSource”, and use SUBJID as a key variable. This source name can be used across the server function. To output the reactive table, I use the event_data() function to access the event of the Plotly figure, and it can be a click event or a hover event. I use plotly_click to trigger the event. Every time a user clicks the figure, the key information is read out. Then I use this key information to filter out the df dataset, and then output the table.

The source code of this app and view of the app is shown blow:

```
library(tidyr)

library(plotly)

library(shiny)

library(DT)


### Create dummy dataset ###

df <- data.frame(SUBJID = seq(100, 120, by=1),
                 STDY = 0,
                 ENDY= c(10,13,15,18,19,23,30,32,33,33,36,40,42,45,46,46,
                         48,50,51,52,55),
                 COHORT = sample(c("group1","group2"),21,replace = TRUE),
                 Status = c("Ongoing","Discontinue","Ongoing",
                            "Discontinue","Ongoing","Discontinue",
                            "Ongoing", "Discontinue", "Ongoing",
                            "Discontinue", "Ongoing", "Discontinue",
```

```
                              "Ongoing", "Discontinue","Ongoing",
                              "Discontinue", "Ongoing", "Discontinue",
                              "Ongoing", "Discontinue","Ongoing"),
                    DISC = c(10,13,15,18,19,23,30,32,33,33,36,40,42,45,46,
                             46,48,50,51,52,55),
                    PDDAY = c(5,6,NA,NA,NA,15,20,NA,NA,21,NA,NA,30,NA,NA,
                              NA,45,NA,43,NA,41),
                    CRDAY = c(NA,NA,10,12,14,NA,NA,9,NA,NA,25,NA,NA,NA,41,
                              NA,NA,40,NA,48,NA))  %>%
        arrange(SUBJID, ENDY)

### Shiny App ###
ui <- fluidPage(
         fluidRow(column(6,plotlyOutput("plot")),
                  htmlOutput("hoverDataOut"),
                  DT::dataTableOutput("table")))
server <- function(input, output, session) {
   output$plot <- renderPlotly({
      p<- plot_ly(data=df, key=~SUBJID, source = "plotSource") %>%
          add_segments(x =~STDY, xend =~ENDY, y =~SUBJID, yend = ~SUBJID,
                       color = ~COHORT) %>%
          add_markers(x =~ENDY, y=~SUBJID, name = ~Status,
                      symbol = ~Status,
                      symbols = c("x","triangle-left-open")) %>%
          add_markers(x=~PDDAY, y=~SUBJID, name = "FIRST PD DAY") %>%
          add_markers(x=~CRDAY, y=~SUBJID, name = "FIRST CR DAY") %>%
          layout(xaxis = list(title = "Study Day",
                              autotick = FALSE,
                              dtick = 5),
                 yaxis = list(title = "Subject ID",
                              autotick = FALSE,
                              range = c(99,121),
                              dtick = 1))
               })
  output$table <- DT::renderDataTable({
     event.data <- event_data(event="plotly_click",source = "plotSource")
     if (!is.null(event.data$key)){
        df <- df %>% filter(SUBJID == as.numeric(event.data$key))
     datatable(df)
    }
  })
}
shinyApp(ui, server)
```



Show 50 entries Search:

	SUBJID	STDY	ENDY	COHORT	Status	DISC	PDDAY	CRDAY
1	120	0	55	group1	Ongoing	55	41	

Showing 1 to 1 of 1 entries Previous 1 Next

Figure 6. Shiny App

CONCLUSION

In this paper, the basic usage of Plotly package in R has been demonstrated with dummy clinical data. The scatter, line and bar plots can be widely applied to produce other plots, such as swimmer plot, forest plot and even gantt chart. Those Plotly figures can be easily embedded into an R shiny app and trigger reactive events. Plotly has low learning threshold and can be used in multiple platforms with similar programming logic. With the help of Plotly, the researchers can easily review the clinical data with interactive visualization. Using Plotly, we can also create fancier R shiny app.

REFERENCES

1. Siever, Carson. 2019. “Interactive Web-based Data Visualization with R, Plotly, and Shiny.” Available at https://plotly-r.com/index.html.

CONTACT INFORMATION

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

Kailu Song
kailu.song@beigene.com