PHUSE EU Connect 2022

Presentation No: DV03

Seeing the Full Picture with a R Shiny Data Visualization Toolbox

Anastasiia Konvisarova, Kharkiv, Ukraine

1 Abstract

As more organizations are embracing the functionality of using open-source languages, we are beginning to see the advantages of being able to build tools to help us in our work. There are many situations in which having a tool that creates data visualizations without the usual hassle of creating custom programs for each figure is advantageous.

This presentation takes us through the development and demonstration of a Shiny app which has the purpose of helping programmers and statisticians make more clear and proficient visualizations. The toolkit is designed to help check every plot in more detail. It is especially useful for tasks which use complicated functions, calculations or statistical methods and helps to give the user a complete picture of their analysis.

2 Introduction

Whilst creating data evaluation visualizations for example for publication materials, it was noted that a more efficient production of graphs would be desirable. Hence, a toolkit for producing displays was created as a R Shiny application. The toolkit utilizes ggplot for building the graphs and has the advantage that it can be used with ADaM datsets.

3 Application Layout and functionality

The current application contains four main parts: Data Input, Data Table Output, Visualization Toolbox, Visualization output. In order to view the Data Table Output and Visualization Output parts, the data must be chosen and checked.

Figure 1 shows the landing page with the first set of fields to be completed. On this page the dataset, variables for two scales and parameters for the particular dataset can be selected. Also, it is possible to choose a dataset from the file system (Button “Choose…”). The dataset can be imported using the .sas7dat extension to create the figure and use the toolbox. Fabricated datasets have been used to mimic the standard ADaM datasets for this demonstration and to test the application. After data is input and checked for appropriateness, areas with Tables and Visualization Output tabs are shown in Figure 2.

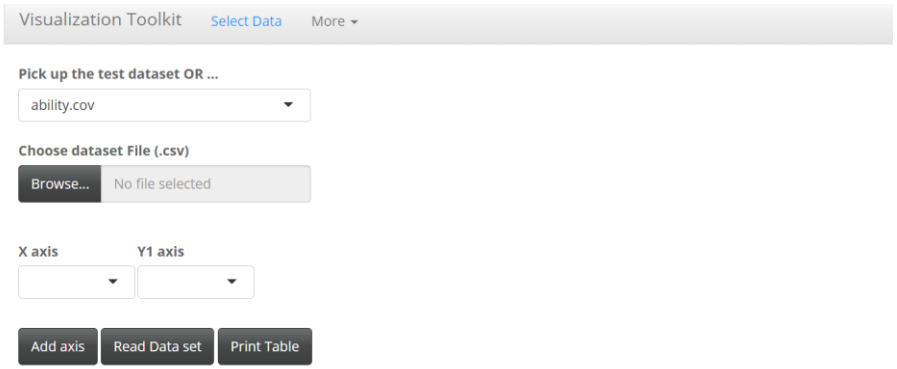

Figure 1 – Main Layout of the application

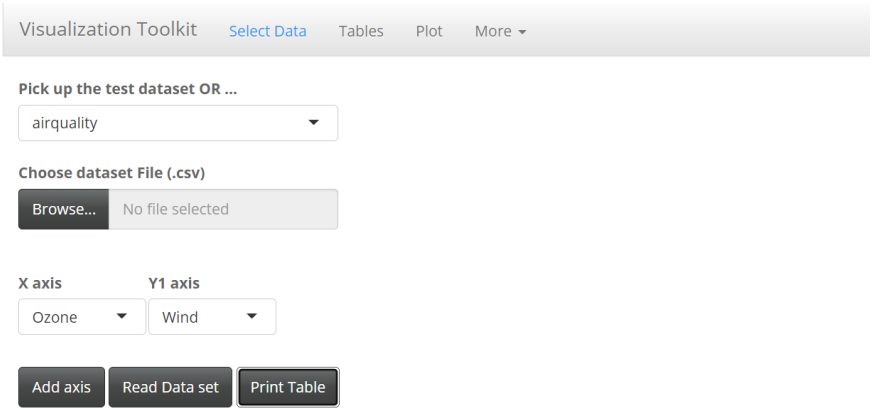

Figure 2 – Hidden tabs after the data set is chosen and checked

The tables with chosen data and chosen subset are available and shown after pushing “Print Table” in “Tables” tab (Figure 3). On the left side it is the full dataset, which was read in the previous section, on the right side is a subset of the dataset, which was chosen.

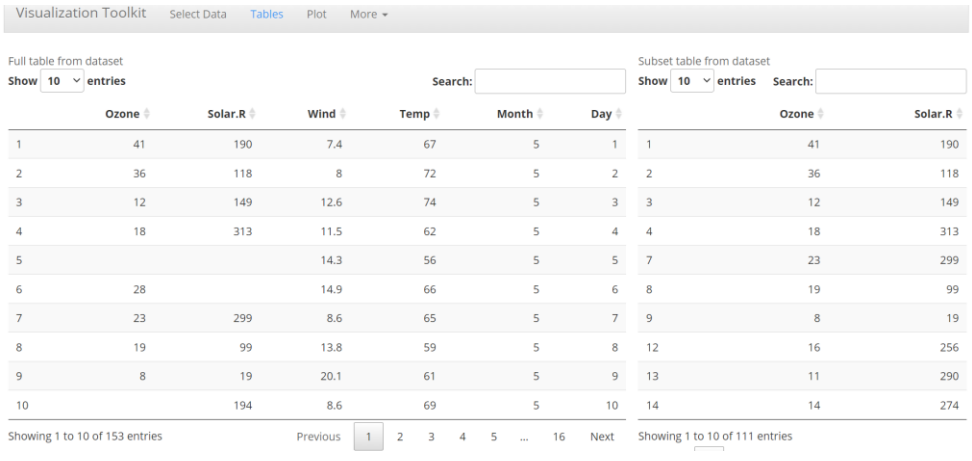

Figure 3 – Tables output visualization

After the dataset was selected, the table field is filled with the data from this dataset. Figure 4 shows the output graph after the “Build” button is clicked. On the left side is the visualization toolbox with 6 options that can be used for manipulating the graph. In the middle is the plot visualization with the graph built based on the subset dataset. On the right side is a table with the data points, which are visualized in the graph. It is possible to use 6 different options in the toolkit: Reset, Pan, Subset, Point, Zoom and All Points on the plot created from the selected variables. Functional code for these options is available in Appendix A.

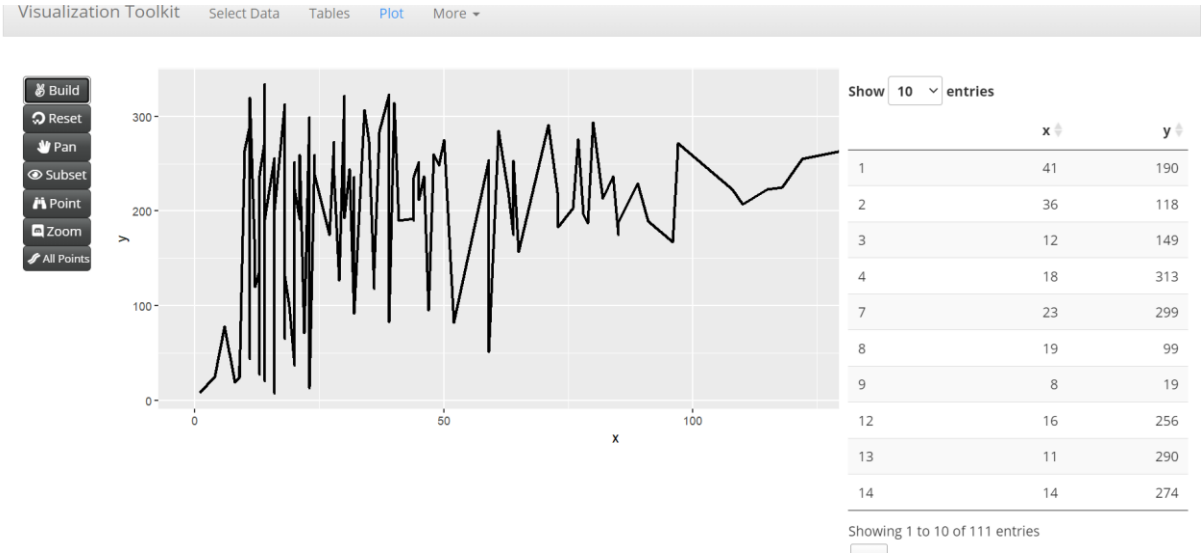

Figure 4 – Plotted graph using Build function

Reset – cleans the previous data on the graph.

Pan – selects an area. The selected part can be moved and scaled. After it is selected, a subset of the initial plot is presented. (Figure 5 shows “Pan” result). Pan could be applied without clicking the button, as this function is built in the plot area itself.

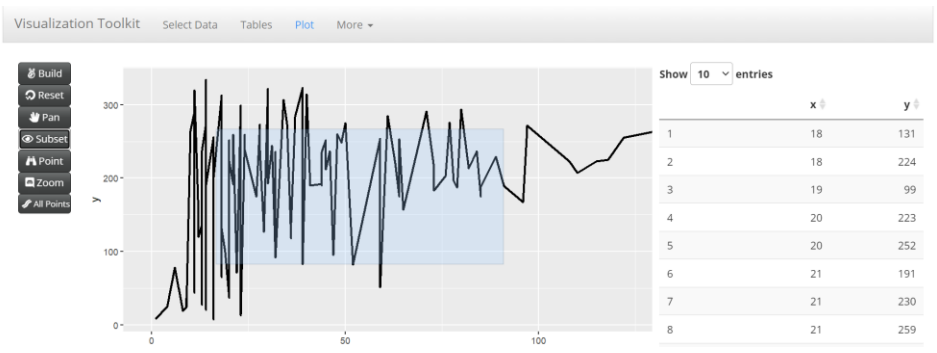

Figure 5 – Usage of Pan function

Subset – this is used to subset the plot and further manipulate the graph. Before clicking on the Subset button, first select the area. (Figure 5 shows "Subset" result of the selected area). The table on the right side shows the data subset from the selection on the plot.

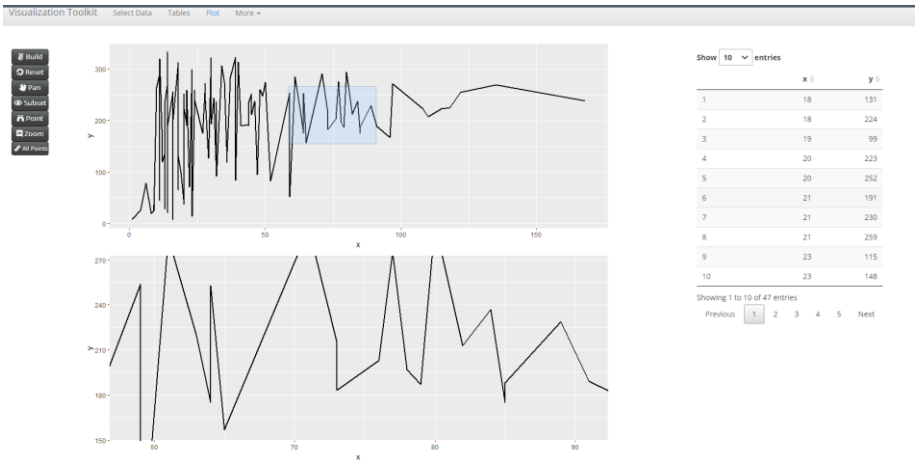

Figure 6 – Usage of Subset function

Point – shows all the active points in the selected area using a red colour and can also be used to select particular points on the graph which are in the current dataset.

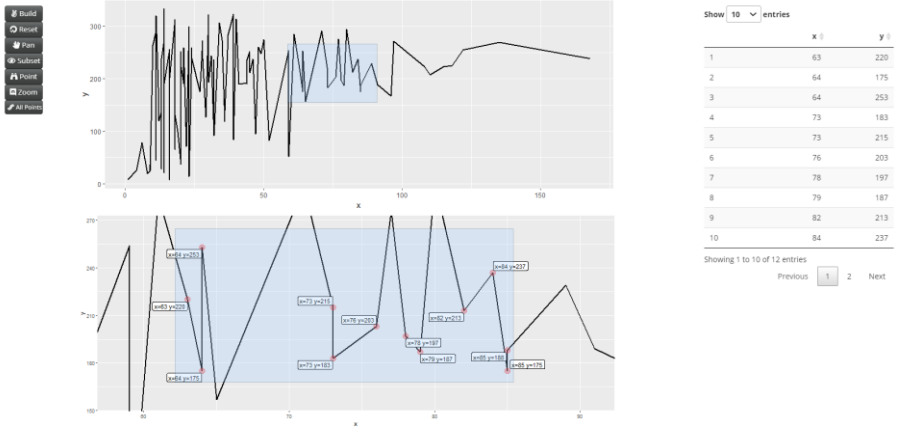

Figure 7 – Usage of Pan function to make a subset of points

Figure 7 (above) shows the selected area with all active points ("Pan" function) in red.

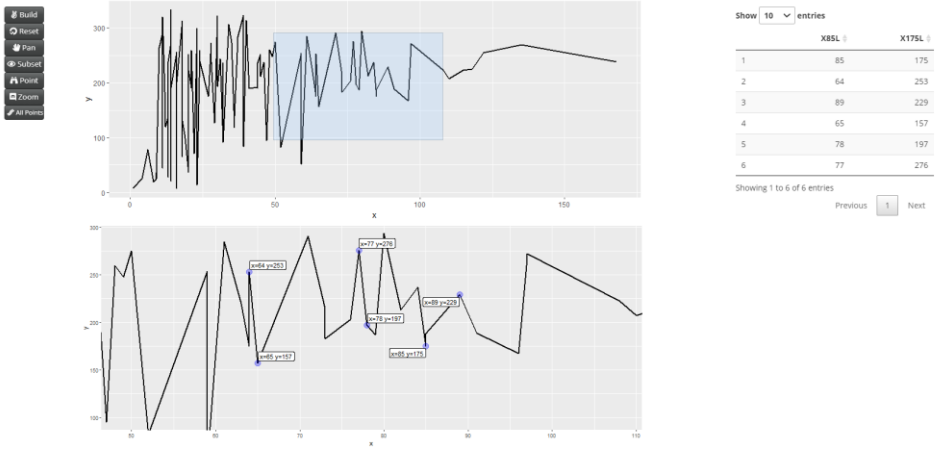

Figure 8 – Usage of Point function to make a particular selection of points

Figure 8 (above) shows the selected free points ("Point" function) in blue. All the points could be deleted by double clicking on any point. The table is interactive and updated using the selected points or selected area. By clicking on the row in the table, the particular point will be highlighted as shown in Figure 9 (below).

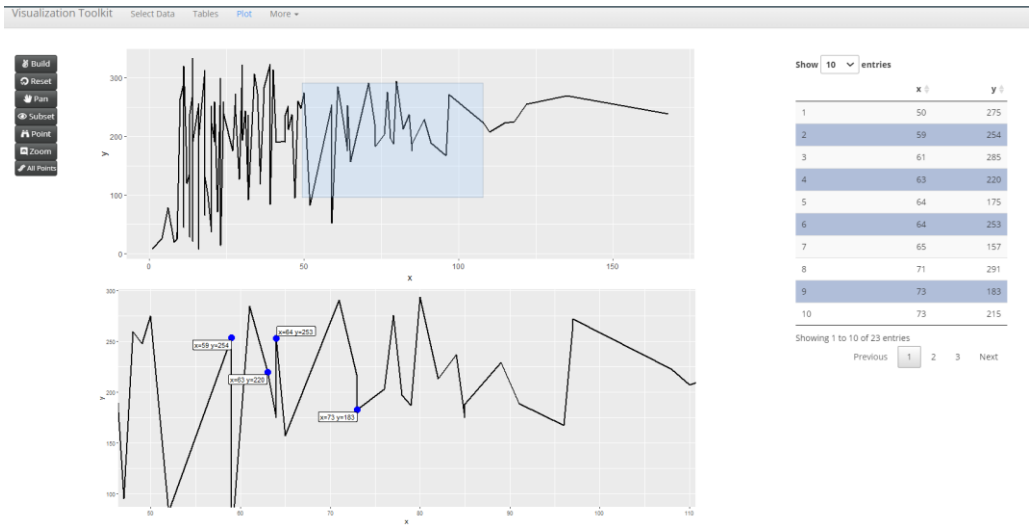

Figure 9 – Choose the particular points in table to highlight those on the plot

Zoom – gives the ability to zoom in on any selected area (see Figure 10 below).

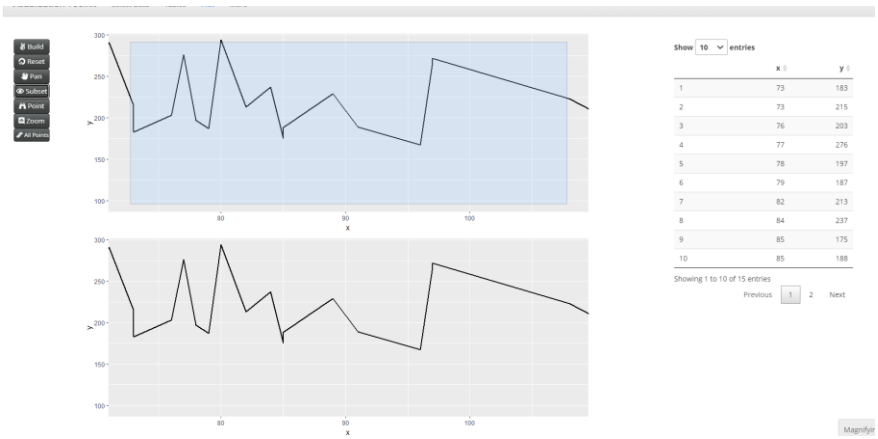

Figure 10 – Zoom in function

All Points – this function will select all points and highlight them on the subset of the plot. In the table on the right, all the points are shown and can be interactively chosen (Figure 11)

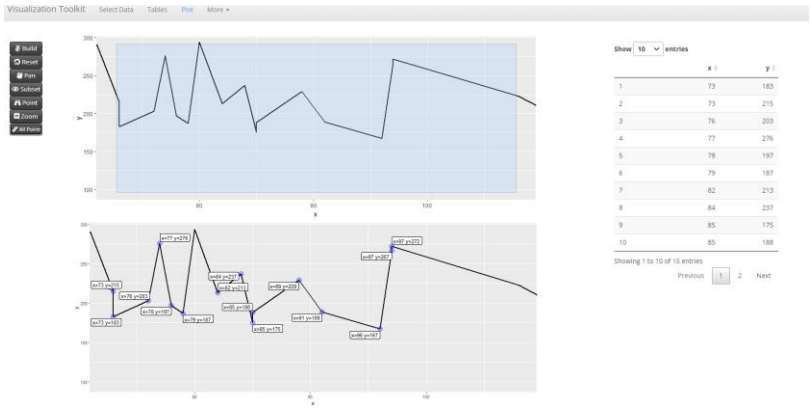

Figure 11 – All points function

To deselect all the point, double click the left mouse button.

All the figures are created using the ggplot package, functions for each of the options were written separately. For both areas (initial graph and subset) it is possible to apply any of the functions in the toolkit. Clicking on the Reset button will erase all plotted areas, so that new graphs and data can be visualized. Furthermore, all the plots are interactive and changeable, so the package can be used for different clinical trials with any data.

3 Conclusion

In clinical trials it is very important to present clear outputs with all the data in the correct and proper order. Visualizations are a good way of presenting results and to do this it is not complex, but it does require the right data to be input. The R Toolbox package can help with distinguishing particular areas and data points on some figures with the help of the options in the toolkit. The toolkit can be used with different statistical methods as it uses standard functions and programming procedures. In addition, it assists programmers and statisticians to get accurate results.

Appendix A

A.1 UI.R

```
#
# This is the user-interface definition of a Shiny web application. You can
# run the application by clicking 'Run App' above.
#
# Find out more about building applications with Shiny here:
#
#    http://shiny.rstudio.com/
#

library(shiny)
library(shinydashboard)
library(shinythemes)
library(mailtoR)
# Define UI for application that draws a histogram

source('tabsets.R')


ui <- fluidPage(
         tags$head(
           tags$style(HTML('.shiny-split-layout>div {overflow: hidden;}')),
         ),
         navbarPage(id="vis_too", "Visualization Toolkit",  collapsible = TRUE, theme =
shinytheme("spacelab"),
                              position = "static-top",
                tab1,
                tab2,
                tab3,

                navbarMenu("More", tab4, tab5)
))
```

A.2 Tabsets.R

```
1.  tab1 <- tabPanel(inputId="sel_d", "Select Data", value="tab1_1",
2.                              selectInput("dataset", "Pick up the test dataset OR ...",
    ls("package:datasets")),
3.                              fileInput("file1", label="Choose dataset File (.csv)", accept =
    c(".csv", ".sas7bdat")),
4.                              #selectInput("column", "Select columns to draw a figure",
    character(0), multiple = TRUE),
5.                              div(id = 'placeholder',
6.                                style = "display: grid;
7.                                   grid-template-columns: 10% repeat(2, 10%); ## same as
    repeat(4, 20%)
8.                                   grid-gap: 2px;",
9.
10.                               selectInput("x", "X axis", character(0), multiple=FALSE,
    width="120px"),
11.                               selectInput("y1", "Y1 axis", character(0),
    multiple=FALSE,  width="120px")),
12.                               actionButton(inputId="add_ax", label="Add axis"),
13.                               actionButton(inputId="read", label="Read Data set"),
14.                               actionButton(inputId="table", label="Print Table"))
15.                             #column(4, offset=0, selectInput("x", "X axis", character(0),
    multiple=TRUE, width="120px")),
16.                             #column(4, selectInput("y1", "Y1 axis", character(0),
    multiple=FALSE, width="120px")))
17.
18. tab2 <- tabPanel("Tables",
19.                             fluidRow(splitLayout(cellWidths = c("65%", "35%"), cellArgs =
    list(style = "padding: 6px"),
20.                                          box("Full table from dataset", width = 20, height =
    "200",solidHeader = T,
21.                                              dataTableOutput("preview")),
22.                                          box("Subset table from dataset",  width = 15, height =
    "200",solidHeader = T,
23.                                              dataTableOutput("psub")))))
24. tab3 <- tabPanel("Plot",
25.                  splitLayout(cellWidths = c("7%", "63%", "30%"), cellArgs = list(style =
    "padding: 6px"),
26.                       fluidRow( box(style='width:100px;overflow-x:
    hidden;height:1000px;overflow-y: hidden; position:fixed;',
27.                         verticalLayout(
28.                           actionButton(inputId="build", label="Build", icon=icon("angellist"),
    style='padding:4px; font-size:95%', width=70),
29.                           actionButton(inputId="reset", label="Reset", icon=icon("digital-
    ocean"), style='padding:4px; font-size:95%', width=70),
30.                           actionButton(inputId="pan", label="Pan", icon=icon("hand-spock"),
    style='padding:4px; font-size:95%', width=70),
31.                           actionButton(inputId="eye", label="Subset", icon=icon("eye"),
    style='padding:4px; font-size:95%', width=70),
32.                           actionButton(inputId="point", label="Point", icon=icon("binoculars"),
    style='padding:4px; font-size:95%', width=70),
33.                           actionButton(inputId="zoom", label="Zoom", icon=icon("discord"),
    style='padding:4px; font-size:95%', width=70),
34.                           actionButton(inputId="s_all", label="All Points", icon=icon("bacon"),
    style='padding:4px; font-size:80%', width=70)))),
35.                        fluidRow(box(style='width:1000px;overflow-x:',
36.                                     plotOutput("hist",
37.                                            click="plot_click",
38.                                            dblclick = "plot_dblclick",
39.                                            hover="plot_hover",
40.                                            brush="plot_brush"
41.                                                    ),
42.                                     plotOutput("subpl",
```

```
                                          click="subplot_click",
                                          hover="subplot_hover",
                                          dblclick="subplot_dblclick",
                                          brush="subplot_brush")
                                    )),
                        fluidRow(box(style='width:350px;overflow-x: hidden; position:fixed;',
                                     dataTableOutput("dtsel"))
                        )))

tab4 <- tabPanel("Outreach", fluidPage(tags$h3("CONTACT VERAMED"),
                                       tags$h2(" "),
                                       tags$h4("Contact us via website"),
                                       tags$a(href="https://www.veramed.co.uk/?gclid=CjwKCAjwq5-WBhB7EiwAl-HEkhemwYwwio4obQnis9OOmp24PnmTdtLQQQGwyOvF_Oeru-Pq38BdrBoCkVIQAvD_BwE",
                                              "Go to the website: veramed.co.uk"),
                                       tags$h2(" "),
                                       tags$h4("Contact us via email"),
                                       mailtoR(email="anastasiia.konvisarova@veramed.co.uk",
                                               text="Email to Anastasiia Konvisarova"),
                                       use_mailtoR(),
                                       tags$h2(" "),
                                       tags$h4("Or please contact Veramed directly:"),
                                       mailtoR(email="info@veramed.co.uk",
                                               text="Email Veramed"),
                                       use_mailtoR(),
                                       tags$h2(" "),
                                       tags$h4("Contact us via tel."),
                                       tags$h5("UK: +44(0) 20 3696 7240"),
                                       tags$h5("US: +1 617 444 8513"),
                                       tags$h5("Ukraine: +38 044 393 1062"),))

tab5 <- tabPanel("About")
```

A.3 Server.R

```
#
# This is the server logic of a Shiny web application. You can run the
# application by clicking 'Run App' above.
#
# Find out more about building applications with Shiny here:
#
#    http://shiny.rstudio.com/
#

library(shiny)
library(ggplot2)
library(dplyr)
library("sas7bdat")
library(ggrepel)
library(DT)

# Define server logic required to draw a histogram
shinyServer(function(input, output, session) {

    hideTab(inputId = "vis_too", target = "Plot")
    hideTab(inputId = "vis_too", target = "Tables")

    #0. Add button for creating new var on the graph
    observeEvent(input$add_ax, {
        ctn(ctn() + 1)

        insertUI(
            selector = '#placeholder',
            ui = div(
                id = Id()('div'),
                selectInput(Id()('y'), 'Y Axis:', character(0), multiple = FALSE),
                uiOutput(Id()('input'))
            )
        )
    })

    observeEvent(ctn(), {
        id <- Id()('input')
        selection <- Id()('y')
        output[[id]] <- renderUI({
            req(input[[Id()('y')]])
            switch(
                updateSelectInput(inputId = "y2", choices=names(data()))
            )
        })
    }, ignoreInit = TRUE)

    #1. function for inputing the dataset

    observeEvent(input$read, {

        output$preview <<- NULL
        output$column <<- NULL

        file <- reactive({input$file1})


        #1.a Read from data set funtion

        if (length(file()) == 0) {
             print("There is no file selected, you can choose from the test ones")
             data <- reactive({
```

```
63.                 dstinput(input$dataset)
64.             })
65.
66.             # if (is.empty(data())) {
67.
68.
69.             #     message("Dataset is not complete")
70.             # }
71.
72.             output$preview <- renderDataTable(
73.                 datatable(data(),
74.                           options=list(selection = "multiple",
75.                                        pageLength = 10)
76.
77.                 ))
78.
79.             observeEvent(input$dataset, {
80.                 updateSelectInput(inputId = "column", choices=names(data()))
81.                 updateSelectInput(inputId = "x", choices=names(data()))
82.                 updateSelectInput(inputId = "y1", choices=names(data()))
83.                 updateSelectInput(inputId = "y", choices=names(data()))
84.             })
85.
86.             selectCn <<- reactive({
87.                 data()[,c(input$x, input$y1)]
88.             })
89.
90.
91.         }
92.         else {
93.             ext <- tools::file_ext(file()$datapath)
94.
95.             req(file())
96.             validate(need(ext == c("csv", "sas7bdat"), "Please upload a csv or sas7bdat
    file"))
97.
98.             if (ext == "csv") {
99.                 data<-reactive({read.csv(file()$datapath)})
100.            }
101.            if (ext == "sas7bdat") {
102.                data<-reactive({read.sas7bdat(file()$datapath)})
103.            }
104.
105.            output$preview <- renderDataTable(
106.                datatable(data(),
107.                          options=list(selection = "multiple",
108.                                       pageLength = 10)
109.
110.                ))
111.
112.            observeEvent(input$dataset, {
113.                updateSelectInput(inputId = "column", choices=names(data()))
114.                updateSelectInput(inputId = "x", choices=names(data()))
115.                updateSelectInput(inputId = "y1", choices=names(data()))
116.            })
117.
118.            selectCn <<- reactive({
119.                data()[,c(input$x, input$y1)]
120.            })
121.        }
122.
123.    })
124.
125.    #1.b Write to the table
126.    # observeEvent(input$add_ax, {
```

```
127.    #     message("Here is an axis added")
128.    #
129.    # })
130.
131.    #1.c Write to the table
132.    observeEvent(input$table, {
133.
134.        showTab(inputId = "vis_too", target = "Plot")
135.        showTab(inputId = "vis_too", target = "Tables")
136.
137.        selc <- reactive(na.omit(selectCn()))
138.
139.            #1.b print dataset to table
140.
141.            x1 <<- reactive(selc()[1])
142.            x2 <<- reactive(selc()[2])
143.
144.            df <- reactive({data.frame(
145.                x=x1(),
146.                y=x2()
147.            )})
148.
149.            #1.c print the subset of the dataset
150.            output$psub <- renderDataTable(
151.                datatable(df(),
152.                          options=list(selection = "multiple",
153.                                       pageLength = 10)
154.
155.                ))
156.
157.
158.            finaldst <<- reactive({dstoutput(df())})
159.
160.        #}
161.    })
162.
163.    #2. events on buttons
164.        #2.1 Build Button
165.
166.    observeEvent(input$build, {
167.        output$hist <- renderPlot({
168.            subplot(finaldst(), xmin=min(finaldst()["x"]),
169.                                xmax=max(finaldst()["x"]),
170.                                ymin=min(finaldst()["y"]),
171.                                ymax=max(finaldst()["y"]))
172.        }, res = 96)
173.
174.        output$dtsel <- renderDataTable(
175.            datatable(finaldst(),
176.                      options=list(selection = "multiple", searching = FALSE,
177.                                   pageLength = 10)
178.
179.            ))
180.
181.    })
182.
183.        #2.2 Reset button
184.    observeEvent(input$reset, {
185.        hideTab(inputId = "vis_too", target = "Plot")
186.        hideTab(inputId = "vis_too", target = "Tables")
187.
188.        output$hist <<- NULL
189.        output$subpl <<- NULL
190.        output$preview <<- NULL
191.        output$psub <<- NULL
```

```
        output$dtsel <<- NULL
    })

    #3. Function to build a subplot from the chosen area

    limits <- reactive(xy_reqturn(input$plot_brush))

    pl<-reactive(subplot(finaldst(),
                         xmin=limits()[["xmin"]],
                         xmax=limits()[["xmax"]],
                         ymin=limits()[["ymin"]],
                         ymax=limits()[["ymax"]]))

     observeEvent(input$eye, {
        showNotification("Magnifying glass")

        subpn<- reactive(get_points(finaldst(), finaldst()[["x"]], finaldst()[["y"]],
    limits()$xmin, limits()$xmax, limits()$ymin, limits()$ymax))

        #print(subpn())

        output$dtsel <- renderDataTable(
            datatable(subpn(),
                      options=list(selection = "multiple", searching = FALSE,
                                   pageLength = 10)

            ))


        output$subpl <- renderPlot({

            tryCatch({
                    pl()
                },
            warning=function(w) {
                message('No data in chosen Analysis', w)
                showModal(
                    modalDialog(
                        div("Magnifying glass: There is no area selected on the graph")
                    )
                )
                return(NULL)
            },
            error=function(e) {
                message('No data in chosen Analysis', e)
                showModal(
                    modalDialog(
                        div("Magnifying glass: There is no area selected on the graph")

                                )
                )
                return(NULL)
            }
            )
    }, res=96)

    })
     #4. Function to select area with interested points
     observeEvent(input$pan, {
         #get coordinatess on click for the particular area with points
         clicks <- reactiveVal(
             data.frame(x = c(), y = c())
         )

         clicks1 <- reactive({
```

```
256.             test_data <- clicks()
257.             colnames(test_data) <- c('x','y')
258.             test_data
259.         })
260. 
261.         observeEvent(input$subplot_brush, {
262.             limsub <- reactive(xy_reqturn(input$subplot_brush))
263. 
264.             subpoints<- reactive(get_points(finaldst(), finaldst()[["x"]], finaldst()[["y"]],
    limsub()$xmin, limsub()$xmax, limsub()$ymin, limsub()$ymax))
265. 
266.             output$dtsel <- renderDataTable(
267.                 datatable(subpoints(),
268.                           options=list(selection = "multiple", searching = FALSE,
269.                                        pageLength = 10)
270. 
271.                 ))
272. 
273. 
274.             output$subpl <- renderPlot({
275. 
276.                 tryCatch({
277.                     pl() + geom_point(data=subpoints(), aes(x=x, y=y), size=5, color="red",
    alpha = 0.3)+
278.                         geom_label_repel(data=subpoints(), aes(x=x, y=y,
    label=paste0("x=",x," y=",y)))
279.                 },
280.                 warning=function(w) {
281.                     pl()
282.                 },
283.                 error=function(e) {
284.                     pl()
285.                 })
286. 
287.             })
288.         })
289. 
290.         observeEvent(input$subplot_click,
291.                      {clicks(rbind(clicks(), c(xy_str(input$subplot_click)[["x"]],
    xy_str(input$subplot_click)[["y"]])))
292.                      )}
293.         )
294. 
295.         #5. Function to delete points on dbl click
296.         dclicks <- reactiveVal(
297.             data.frame(x = c(), y = c())
298.         )
299. 
300.         dclicks1 <- reactive({
301.             dtest_data <- dclicks()
302.             colnames(dtest_data) <- c('x','y')
303.             dtest_data
304.         })
305. 
306.         observeEvent(input$subplot_dblclick,
307.                      {clicks(-c(xy_str(input$subplot_dblclick)[["x"]],
    xy_str(input$subplot_dblclick)[["y"]]))
308.                          output$dtsel <<- NULL
309.                      }
310.         )
311. 
312.     })
313. 
314.     observeEvent(input$point, {
315.         #get coordinatess on click for the particular points
```

```
316.         clicks <- reactiveVal(
317.             data.frame(x = c(), y = c())
318.         )
319.
320.         clicks1 <- reactive({
321.             test_data <- clicks()
322.             colnames(test_data) <- c('x','y')
323.             test_data
324.         })
325.
326.         observeEvent(input$subplot_click,
327.                      {clicks(rbind(clicks(), c(nearPoints(finaldst(), input$subplot_click,
    threshold=100, maxpoints=1)[["x"]],
328.                                                nearPoints(finaldst(), input$subplot_click,
    threshold=100, maxpoints=1)[["y"]]))
329.                      )}
330.         )
331.
332.         observeEvent(input$subplot_click, {
333.
334.             output$dtsel <- renderDataTable(
335.                 datatable(clicks(),
336.                           options=list(selection = "multiple", searching = FALSE,
337.                                        pageLength = 10)
338.
339.                 ))
340.
341.
342.             output$subpl <- renderPlot({
343.
344.                 tryCatch({
345.
346.                     pl() + geom_point(data=clicks1(), aes(x=x, y=y), size=5, color="blue",
    alpha = 0.3)+
347.                         geom_label_repel(data=clicks1(), aes(x=x, y=y, label=paste0("x=",x,"
    y=",y)))
348.
349.
350.                 },
351.                 warning=function(w) {
352.                     showNotification("There is no area chosen or the x and y are not
    numbers")
353.                     print("There is no area chosen or the x and y are not numbers")
354.                     pl()
355.                 },
356.                 error=function(e) {
357.                     showNotification("There is no area chosen or the x and y are not
    numbers")
358.                     print("There is no area chosen or the x and y are not numbers")
359.                     pl()
360.                 })
361.
362.             })
363.             #get points to delete on dbl click
364.             dclicks <- reactiveVal(
365.                 data.frame(x = c(), y = c())
366.             )
367.
368.             dclicks1 <- reactive({
369.                 dtest_data <- dclicks()
370.                 colnames(dtest_data) <- c('x','y')
371.                 dtest_data
372.             })
373.
374.             observeEvent(input$subplot_dblclick,
```

```
                        {clicks(-c(xy_str(input$subplot_dblclick)[["x"]],
    xy_str(input$subplot_dblclick)[["y"]]))
                        }
            )

        })

    })

    #6. Function to click the points separately
    observeEvent(input$s_all, {
        #get coordinatess on click for the particular points

        dt <- reactive({ggplot_build(pl())})

        subp_global <- reactive ({
            data.frame(dt()$data[[1]]["x"], dt()$data[[1]]["y"])
        })

        subpnt<- reactive(get_points(finaldst(), finaldst()[["x"]], finaldst()[["y"]],
    limits()$xmin, limits()$xmax, limits()$ymin, limits()$ymax))

        output$dtsel <- renderDataTable(datatable(subpnt(),
                options=list(selection = "multiple", searching = FALSE,
                pageLength = 10)
            ))

            output$subpl <- renderPlot({

            tryCatch({

                pl() + geom_point(data=subpnt(), aes(x=x, y=y), size=5, color="blue", alpha =
    0.3)+
                       geom_label_repel(data=subpnt(), aes(x=x, y=y, label=paste0("x=",x,"
    y=",y)))

            },
            warning=function(w) {
                showNotification("There is no area chosen or the x and y are not numbers")
                print("There is no area chosen or the x and y are not numbers")
                pl()
            },
            error=function(e) {
                showNotification("There is no area chosen or the x and y are not numbers")
                print("There is no area chosen or the x and y are not numbers")
                pl()
            })
            })

            observe({
                req(input$dtsel_rows_selected)
                selRow <- reactive({subpnt()[input$dtsel_rows_selected,]})
                print(selRow()[[1]])
                output$subpl <- renderPlot({
                        pl() + geom_point(data=selRow(), aes(x=x, y=y), size=5, color="blue",
    alpha = 1)+
                        geom_label_repel(data=selRow(), aes(x=x, y=y, label=paste0("x=",x,"
    y=",y)))
                })
    })

    })

    observeEvent(input$zoom, {
        showNotification("Magnifying glass")
```

```
434.
435.         output$hist <- renderPlot({
436.             tryCatch({
437.                 pl()
438.             },
439.             warning=function(w) {
440.                 message('No data in chosen Analysis', w)
441.                 showModal(
442.                     modalDialog(
443.                         div("Magnifying glass: There is no area selected on the graph")
444.                     )
445.                 )
446.                 return(NULL)
447.             },
448.             error=function(e) {
449.                 message('No data in chosen Analysis', e)
450.                 showModal(
451.                     modalDialog(
452.                         div("Magnifying glass: There is no area selected on the graph")
453.
454.                     )
455.                 )
456.                 return(NULL)
457.             }
458.             )
459.         }, res=96)
460.
461.     })
462.
463.     ctn <- reactiveVal(0)
464.     Id <- reactive({
465.         function(id){
466.             paste0(id, ctn())
467.         }
468.     })
469.
470.
471.
472.})
```

A.4 plot_fun.R

```
#create a plot for the dataset
subplot <- function(df, xmin=0, xmax=0, ymin=0, ymax=0) {
  sbplt<-ggplot(df, aes(x, y)) +
    geom_line(size=1)+
    coord_cartesian(xlim=c(xmin, xmax), ylim=c(ymin, ymax))+
    scale_color_brewer(palette="Dark2")
  return(sbplt)
}

#get data from mouse
xy_str <- function(e) {
  if(is.null(e)) return("NULL\n")
  paste0("x=", round(e$x, 4), " y=", round(e$y, 4), "\n")
  paramxy <- list(x=round(e$x,4), y=round(e$y, 4))
  return(paramxy)
}
xy_range_str <- function(e) {
  if(is.null(e)) return("NULL\n")

  paste0("xmin=", round(e$xmin, 4), " xmax=", round(e$xmax, 4),
         " ymin=", round(e$ymin, 4), " ymax=", round(e$ymax,4))

}
xy_reqturn <- function(e) {
  if(is.null(e)) return("NULL\n")
  paramlist <- list(xmin=round(e$xmin, 4), xmax=round(e$xmax, 4),
                    ymin=round(e$ymin, 4), ymax=round(e$ymax, 4))
  #print(paramlist)
  return(paramlist)
}

#dataset to select input
dstinput <- function(dstinp) {
  out <- get(dstinp, "package:datasets")
  if (!is.data.frame(out)) {
    validate(paste0("'", dstinp, "' is not a data frame"))
  }
  return (out)
}

#converting dataset to final
dstoutput <- function(dstout) {

    test_data <- dstout
    colnames(test_data) <- c('x','y')
    test_data

  return (test_data)
}

#get all point in the selected area from MAIN plot
get_points <- function(dst, dstx, dsty, xmin, xmax, ymin, ymax) {
  tempdsty <- reactive({
    ifelse(round(dsty,4) > round(ymin, 4) ,
           ifelse(round(dsty,4) < round(ymax, 4),dsty,NA),NA)
  } )

  tempdstx <- reactive({
    ifelse(round(dstx,4) > round(xmin, 4) ,
           ifelse(round(dstx,4) < round(xmax, 4),dstx,NA),NA)
  } )
```

```
62.
63.   substx <- (data.frame(na.omit(tempdstx()), y=0))
64.   substy <- data.frame(x=0, na.omit(tempdsty()))
65.
66.   substx1 <- {
67.     dtest_data <- substx
68.     colnames(dtest_data) <- c('x', 'y')
69.     dtest_data
70.   }
71.   substy1 <- {
72.     dtest_data <- substy
73.     colnames(dtest_data) <- c('x', 'y')
74.     dtest_data
75.   }
76.
77.   substx2 <- {
78.     dtest_data <- merge(dst, substx1, by = "x")
79.     dtest_data
80.   }
81.
82.   substy2 <- {
83.     dtest_data <- merge(dst, substy1, by = "y")
84.     dtest_data
85.   }
86.
87.   subst <- {
88.     temp <- merge(substy2, substx2)
89.     temp1 <- subset(temp, select = c(x,y) )
90.     temp2 <- inner_join(temp1, dst)
91.     temp2
92.     }
93.
94.   return(distinct(subst))
95.
96. }
```