SESUG Paper 207-2019

# Make your Data shine with R-Shiny

Pavan Vemuri(Sri Pavan Vemuri), Alkermes;

## ABSTRACT

Data visualization and analysis is vital to the Pharma/CRO industry. Real time data access can enhance and streamline the ease of information transfer. This Paper aims to provide an example and serve as a template for other applications by making Patient profiles data reactive. R and R-shiny are used to build the profiles data as a web service instead of static reports giving the ability to access/visualize data in real time.

## INTRODUCTION

Data signals are key drivers in Clinical trials. Patient profiles are a fitting example to corroborate the above statement as they analyze key patient data. The data in pursuit for these reports can be safety and/or efficacy data. Output format is usually static and in RTF/PDF. Having real time access (will be used interchangeably as “reactive data”) will simplify the data transfer process and in turn the analysis time. The program in this presentation achieves this by building a web based application for patient profiles using R-Shiny. While demonstration here is specific to Patient profiles, the logic and approach can be implemented in any situation where real time data access is beneficial.

### 1. Building the Patient Profiles Application:

The purpose of this patient profiles application is to mimic the RTF/PDF report while providing the benefit of reactive data. The steps involved in implementation process are explained below and can be broken down into four components as shown in the flow chart “Figure 1.”

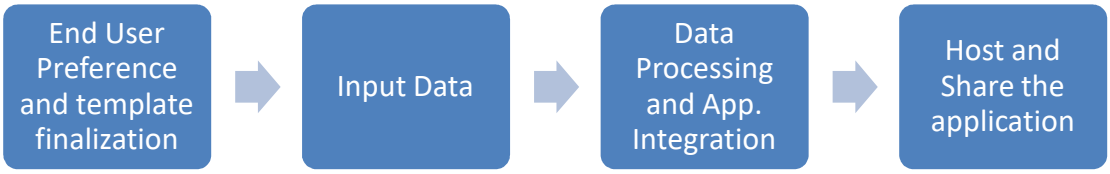



**Figure 1. Process flow**

### 1.1 End User Preference and Template:

The preliminary step involves identifying the audience and understanding the information they seek. This determines the data, as well as the layout. The focus of implementation should be on delivering the information in a precise and simple manner. In our scenario, this translates to a simple layout with the ability to show overall as well as individual subject data by utilizing “patient id” as the trigger for selection. The data is displayed as Tables and graphs with text fields in between to enhance readability. A single page layout is selected vs. tab layout due to ease of navigation.

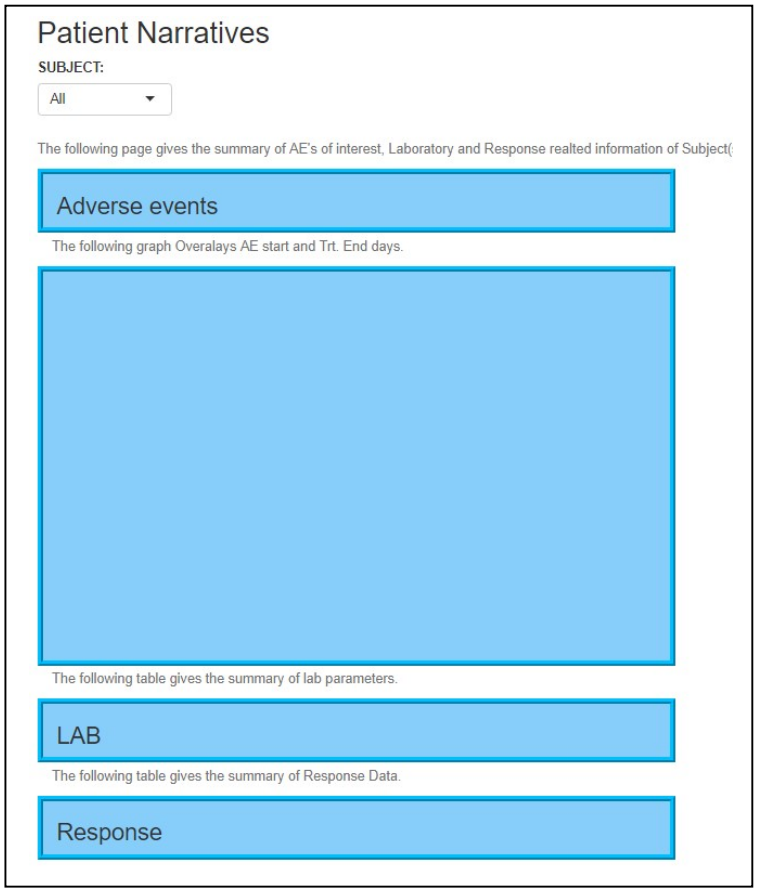
An html patient profiles page can be accomplished using various languages. R and R-Shiny are selected due to ease of syntax esp. in data manipulation and hosting data. A high-level logic flow and R packages used are explained below with basic R programming knowledge left to the user. The complete program is shown in the appendix section “2.1 Code”.

Specific R packages perform specific functions. The functions from a package become available in the program after the respective library call. The core packages used in this paper are:

- DPLYR: DPLYR is probably the most used R package for data manipulation. In our example, sub setting/processing the data based on input selection is achieved using the DPLYR package functions.
- PLOTLY: Plotly R package utilizes JavaScript library plotly.js. It is used to generate interactive and standalone graphs. In our Paper it is used to generate the Adverse Events graph with attributes like Hover text, zoom, pan etc. Figure 3 illustrates the Hover text feature.
- DT Package: The DT package converts the data frame(R data structures) into Java tables. This helps in embedding your data directly into a web page. In our application every table seen is a result of the DT package.
- SHINY: Shiny is a versatile web package. It is the main component of the paper and is used in packaging the tables and graphs into User Interface and Server-side functions that work in tandem as input/output modules. In other words, it helps to build the interface and establish connection to the data.

The programming aspect of the application can be broken down into User Interface (UI) and the Server side modules. The UI primarily deals with providing the Interface and a gateway for user to provide input. The server side provides the processing and output mechanism for the data. The data can be populated in the UI side, Server side or both. In this paper, Subject selection is implemented on the UI side and the data for tables, graphs and reactive text boxes is performed on the server side.

Shown in Figure 2 is a blank template which gives an insight into how the final report would look like.



### Figure 2. Template layout without data

The code needed to generate the template is shown below. It is worthwhile to note the minimal code needed.

```
#program begins


    # define UI for application
1.  ui <-
2.    fluidPage(
                  #### create a html style
3.       tags$style(HTML("
                  #first {
4.                    border-style: ridge;
5.                    border-width: 5px;
6.                    border-color:  DeepSkyBlue;
7.                    background-color: LightSkyBlue;
8.
9.                    }
10.                  ")),
11.    titlePanel("Patient Narratives"),
                    # Create a new Row in the UI for selectInputs
                   ##select input column
12.    fluidRow(
13.      column(2,
14.               selectInput("USUBJID",
15.                           "SUBJECT:",
16.                           c("All",
17.                             unique(adae2$USUBJID)))             )
18.    ),
                   # Create a new row for the table.
19.    helpText("Text filed for information"),
                    ##reactive text
20.    textOutput("Text entered here changes with the input selection"),
                  ##Panel for AE table (table1).
21.    column(8,offset=0.5, id="first", h3("Adverse events"),dataTableOutput("Table1")),
                    ##text to add infromation about graph
22.    column(12, helpText("The following graph Overalays AE start and Trt. End days.")),
                   ##AE plot (AEP).
23.    column(8,offset=0.5,id="first",plotlyOutput("AEP")),
                   ##text to add infromation about Lab data
24.    column(12, helpText("The following table gives the summary of lab parameters.")),
                  ##Panel for Lab table (table2).
25.    column(8,  offset=0.5, h3("LAB"),id="first", dataTableOutput("table2")),
                  ##text to add infromation about Response data
```

```
26.     column(12, helpText("The following table gives the summary of Response Data.")),
                   ##Panel for Response table (table3).
27.     column(8,  offset=0.5, h3("Response"),id="first", dataTableOutput("table3"))
28.   )


29.  server <- function(input, output) {}
                ## Serve side code comes
                     ## Run the application
30. shinyApp(ui = ui, server = server)
```

The code until line 28 generates the UI. UI comprises of page layout and procedures that capture the data in a specified format Ex; page style elements, text boxes, graph, tables etc. Some of the important elements of the UI code are discussed in this paragraph. The “fluidpage” keyword in line 2 automatically adjusts the placement and spacing of panels which hold data. Tag$style keyword in line 3 generates CSS elements which give the page coloring, borders and additional formatting. The subject selection is achieved using “Selectinput” keyword in line 14 which provides filter mechanism to input data. Similarly “RenderTable”, “Rendergraph” and “Textoutput” keywords provide mechanism to show Tables, graphs and textboxes respectively. Each of these elements provides procedures to render output and each of them is given a unique id. This Id value is used on server side module to link the data. Line 29 is the beginning of server side code implementation. The server side code is explained in detail in “section 1.3”. Line 30 calls the App and the result of the call is Figure 2.

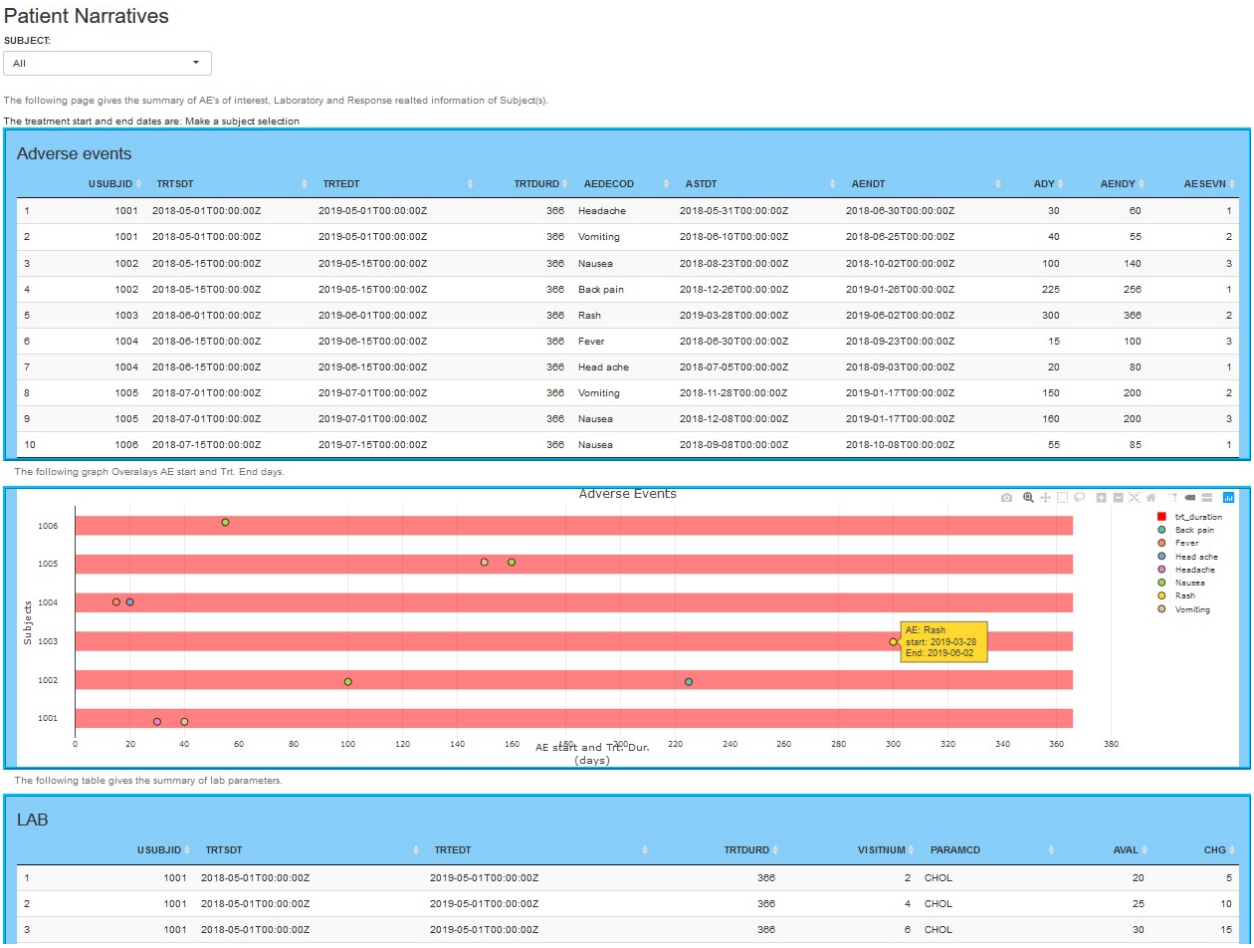
## 1.2 Input Data:

Input data is determined by the information being pursued and will need to be validated to preserve the integrity. The format itself (SAS, EXCEL, CSV etc.) can be decided by programmer or based on the convention of the department. In this paper, the data is stored in excel format for simplicity. The input data utilized is displayed in tables 1, 2 and 3 in appendix section.

## 1.3 Server side code and Data Integration:

The server side code is where the UI elements are linked to the data using their unique IDs and provide an input output mechanism to populate the data. The appendix section 2.1 shows the complete code. A brief overview and program flow is explained here. The lines 62-70 help in populating the treatment start and stop dates for the particular subject selection. The procedure(s) inside the brackets are executed first and the data is passed to the appropriate element using its ID. Similarly lines 71-96 are responsible for the AE, Lab, Response tables and lines 97-116 generate the graph.

When the program is executed initially, the result is an html page with all the subjects data shown. This will be called the landing page as this is the first glimpse of output a user will see.




## Figure 3. Landing page

The text box labeled “SUBJECT:” provides the filtering mechanism via a drop down menu. The text as well as tables and graph reflect specific subject data based on selection. When a selection is made, the UI passes the selection information to the server side which performs data selection/processing and returns back the results. Shown below is the screen shot when a subject 1004 is selected.

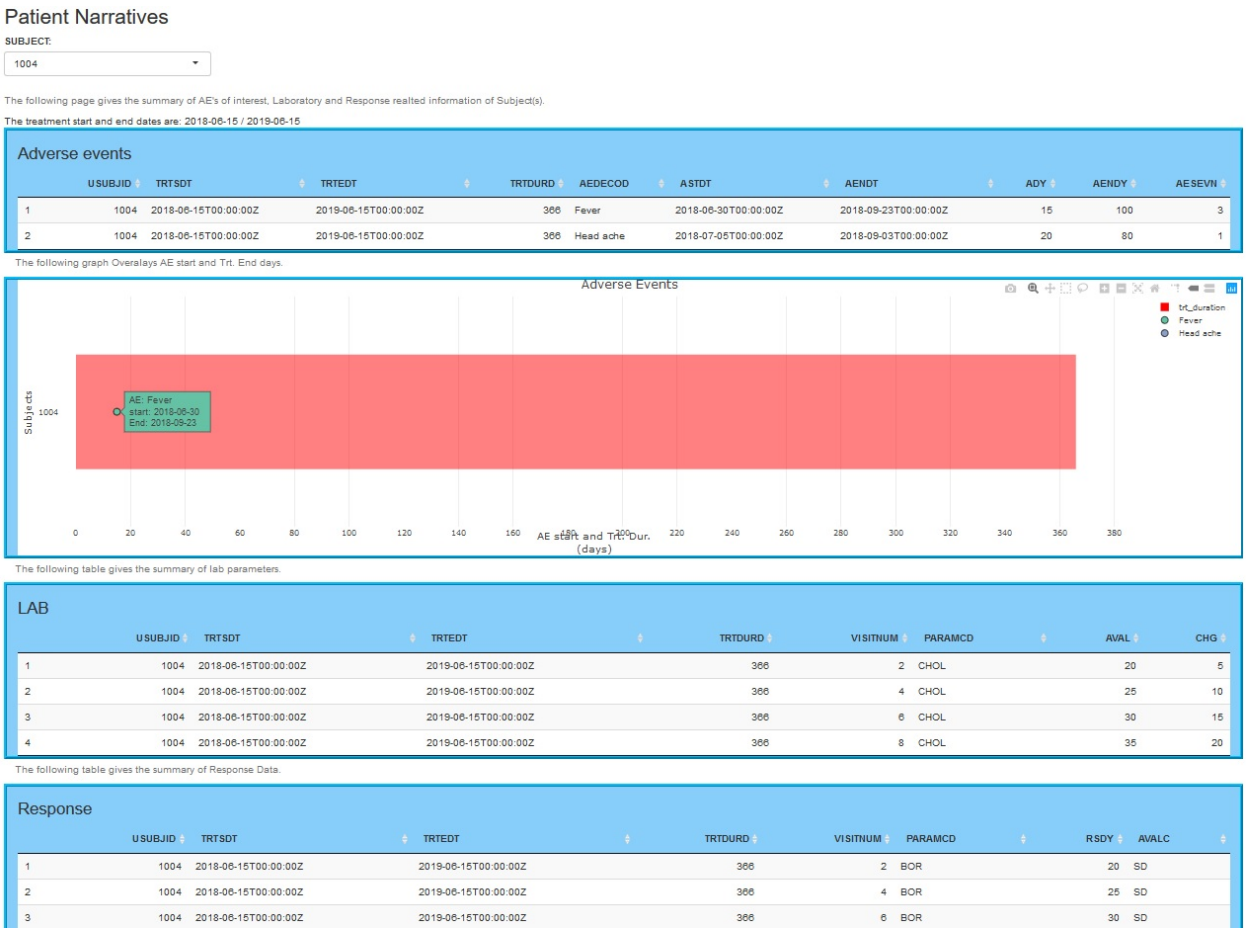



**Figure 4. Subject Selection 1004.**

### 1.4 Hosting the data onto a server:

This is the Final step and can be accomplished in more than one way. For corporate implementation, It is advisable to communicate with the IT department to ensure the company security policies and guidelines are followed. To establish complete control and customization, one will need to host the data in the corporate server. Alternatively the data can also be hosted by R-server using a paid service. The input data references will need to be made dynamic before hosting the application. This will ensure the data is available when the application is run without any broken links. The application shown in the paper is hosted on R server. Hosting data on R server for personal purpose is free which our case is.

## 2.0 Appendix:

| USUBJID | TRTSDT | TRTEDT | TRTDURD | AEDECOD | ASTDT | AENDT | ADY | AESEVN |
|---|---|---|---|---|---|---|---|---|
| 1001 | 5/1/2018 | 5/1/2019 | 366 | Headache | 5/31/2018 | 6/30/2018 | 30 | 1 |
| 1001 | 5/1/2018 | 5/1/2019 | 366 | Vomiting | 6/10/2018 | 6/25/2018 | 40 | 2 |
| 1002 | 5/15/2018 | 5/15/2019 | 366 | Nausea | 8/23/2018 | 10/2/2018 | 100 | 3 |
| 1002 | 5/15/2018 | 5/15/2019 | 366 | Back pain | 12/26/2018 | 1/26/2019 | 225 | 1 |
| 1003 | 6/1/2018 | 6/1/2019 | 366 | Rash | 3/28/2019 | 6/2/2019 | 300 | 2 |
| 1004 | 6/15/2018 | 6/15/2019 | 366 | Fever | 6/30/2018 | 9/23/2018 | 15 | 3 |
| 1004 | 6/15/2018 | 6/15/2019 | 366 | Head ache | 7/5/2018 | 9/3/2018 | 20 | 1 |
| 1005 | 7/1/2018 | 7/1/2019 | 366 | Vomiting | 11/28/2018 | 1/17/2019 | 150 | 2 |
| 1005 | 7/1/2018 | 7/1/2019 | 366 | Nausea | 12/8/2018 | 1/17/2019 | 160 | 3 |
| 1006 | 7/15/2018 | 7/15/2019 | 366 | Nausea | 9/8/2018 | 10/8/2018 | 55 | 1 |

## Table 1. Adverse Events

| USUBJID | TRTSDT | TRTEDT | TRTDURD | VISITNUM | PARAMCD | AVAL | CHG |
|---|---|---|---|---|---|---|---|
| 1001 | 5/1/2018 | 5/1/2019 | 366 | 2 | CHOL | 20 | 5 |
| 1001 | 5/1/2018 | 5/1/2019 | 366 | 4 | CHOL | 25 | 10 |
| 1001 | 5/1/2018 | 5/1/2019 | 366 | 6 | CHOL | 30 | 15 |
| 1001 | 5/1/2018 | 5/1/2019 | 366 | 8 | CHOL | 35 | 20 |
| 1002 | 5/15/2018 | 5/15/2019 | 366 | 2 | CHOL | 20 | 5 |
| 1002 | 5/15/2018 | 5/15/2019 | 366 | 4 | CHOL | 25 | 10 |
| 1002 | 5/15/2018 | 5/15/2019 | 366 | 6 | CHOL | 30 | 15 |
| 1002 | 5/15/2018 | 5/15/2019 | 366 | 8 | CHOL | 35 | 20 |
| 1003 | 6/1/2018 | 6/1/2019 | 366 | 2 | CHOL | 20 | 5 |
| 1003 | 6/1/2018 | 6/1/2019 | 366 | 4 | CHOL | 25 | 10 |
| 1003 | 6/1/2018 | 6/1/2019 | 366 | 6 | CHOL | 30 | 15 |
| 1003 | 6/1/2018 | 6/1/2019 | 366 | 8 | CHOL | 35 | 20 |
| 1004 | 6/15/2018 | 6/15/2019 | 366 | 2 | CHOL | 20 | 5 |
| 1004 | 6/15/2018 | 6/15/2019 | 366 | 4 | CHOL | 25 | 10 |
| 1004 | 6/15/2018 | 6/15/2019 | 366 | 6 | CHOL | 30 | 15 |
| 1004 | 6/15/2018 | 6/15/2019 | 366 | 8 | CHOL | 35 | 20 |
| 1005 | 7/1/2018 | 7/1/2019 | 366 | 2 | CHOL | 20 | 5 |
| 1005 | 7/1/2018 | 7/1/2019 | 366 | 4 | CHOL | 25 | 10 |
| 1005 | 7/1/2018 | 7/1/2019 | 366 | 6 | CHOL | 30 | 15 |
| 1005 | 7/1/2018 | 7/1/2019 | 366 | 8 | CHOL | 35 | 20 |
| 1006 | 7/15/2018 | 7/15/2019 | 366 | 2 | CHOL | 20 | 5 |
| 1006 | 7/15/2018 | 7/15/2019 | 366 | 4 | CHOL | 25 | 10 |
| 1006 | 7/15/2018 | 7/15/2019 | 366 | 6 | CHOL | 30 | 15 |
| 1006 | 7/15/2018 | 7/15/2019 | 366 | 8 | CHOL | 35 | 20 |

## Table 2. Laboratory data

| USUBJID | TRTSDT | TRTEDT | TRTDURD | VISITNUM | PARAMCD | RSDY | AVALC |
|---|---|---|---|---|---|---|---|
| 1001 | 5/1/2018 | 5/1/2019 | 366 | 2 | BOR | 20 | SD |
| 1001 | 5/1/2018 | 5/1/2019 | 366 | 4 | BOR | 25 | SD |
| 1001 | 5/1/2018 | 5/1/2019 | 366 | 6 | BOR | 30 | PR |
| 1001 | 5/1/2018 | 5/1/2019 | 366 | 8 | BOR | 35 | CR |
| 1002 | 5/15/2018 | 5/15/2019 | 366 | 2 | BOR | 20 | SD |
| 1002 | 5/15/2018 | 5/15/2019 | 366 | 4 | BOR | 25 | SD |
| 1002 | 5/15/2018 | 5/15/2019 | 366 | 6 | BOR | 30 | CR |
| 1002 | 5/15/2018 | 5/15/2019 | 366 | 8 | BOR | 35 | SD |
| 1003 | 6/1/2018 | 6/1/2019 | 366 | 2 | BOR | 20 | CR |
| 1003 | 6/1/2018 | 6/1/2019 | 366 | 4 | BOR | 25 | SD |
| 1003 | 6/1/2018 | 6/1/2019 | 366 | 6 | BOR | 30 | PR |
| 1003 | 6/1/2018 | 6/1/2019 | 366 | 8 | BOR | 35 | PR |
| 1004 | 6/15/2018 | 6/15/2019 | 366 | 2 | BOR | 20 | SD |
| 1004 | 6/15/2018 | 6/15/2019 | 366 | 4 | BOR | 25 | SD |
| 1004 | 6/15/2018 | 6/15/2019 | 366 | 6 | BOR | 30 | SD |
| 1004 | 6/15/2018 | 6/15/2019 | 366 | 8 | BOR | 35 | SD |
| 1005 | 7/1/2018 | 7/1/2019 | 366 | 2 | BOR | 20 | CR |
| 1005 | 7/1/2018 | 7/1/2019 | 366 | 4 | BOR | 25 | PR |
| 1005 | 7/1/2018 | 7/1/2019 | 366 | 6 | BOR | 30 | CR |
| 1005 | 7/1/2018 | 7/1/2019 | 366 | 8 | BOR | 35 | CR |
| 1006 | 7/15/2018 | 7/15/2019 | 366 | 2 | BOR | 20 | PR |
| 1006 | 7/15/2018 | 7/15/2019 | 366 | 4 | BOR | 25 | PR |
| 1006 | 7/15/2018 | 7/15/2019 | 366 | 6 | BOR | 30 | PR |
| 1006 | 7/15/2018 | 7/15/2019 | 366 | 8 | BOR | 35 | SD |

## Table 3. Response Data

## 2.1 Code

```
1. ##Back up
2. library(shiny)
3. library(dplyr)
4. library(DT)
```

```
library(tibble)
library(plotly)
library(readxl)
##Read the input files.
adae2 <- read_excel("./data/test_adae.xlsx")
adlb <- read_excel("./data/test_adlb.xlsx")
adrs <- read_excel("./data/test_adrs.xlsx")
## Define UI for application that draws a histogram
ui <-
fluidPage(
## create html style elements for border and colors
tags$style(HTML("
#first {
border-style: ridge;
border-width: 5px;
border-color:  DeepSkyBlue;
background-color: LightSkyBlue;
}
")),
titlePanel("Patient Narratives"),
## Create a new Row in the UI to select subjects
##select input column. This will hold all the unique subjects
fluidRow(
column(2,
selectInput("USUBJID",
"SUBJECT:",
c("All",
unique(adae2$USUBJID)))
)
),
## Create a new row for the table.
```

```
36.  helpText("The following page gives the summary of AE's of interest, Laboratory and Response
37.  realted information of Subject(s)."),


38.  ##reactive text to hold the start and end dates of treatment
39.  textOutput("trt_start"),
40.  #Data table for AE
41.  column(8,offset=0.5, id="first", h3("Adverse events"),dataTableOutput("table")),
42.  ##help text
43.  column(12, helpText("The following graph Overalays AE start and Trt. End days.")),
44.  ##AE Plot
45.  column(8,offset=0.5,id="first",plotlyOutput("AEP")),
46.  ##help text
47.  column(12, helpText("The following table gives the summary of lab parameters.")),
48.  ##Table for Lab
49.  column(8,  offset=0.5, h3("LAB"),id="first", dataTableOutput("table0")),
50.  column(12, helpText("The following table gives the summary of Response Data.")),
51.  ##Table for Response Data
52.  column(8,  offset=0.5, h3("Response"),id="first", dataTableOutput("table1"))
53.  )
54.  ## Server side implementation
55.  server <- function(input, output) {
56.  ##text outupt
57.  output$trt_start <- renderText({


58.  ##subset data and show treatment start and end dates based on selected subject
59.  data0 <- as_tibble(adae2)
60.  if (input$USUBJID != "All") {
61.  data0 <- as_tibble(adae2) %>% filter(USUBJID== input$USUBJID)
62.  data0_ <- c("The treatment start and end dates are:",unique(as.character(data0$TRTSDT)), "/",
63.  unique(as.character(data0$TRTEDT)))
64.  }
```

```
65.  ##output a generic statment if all subjects are selected
66.  if (input$USUBJID == "All") {
67.  data0_ <- c("The treatment start and end dates are: Make a subject selection")
68.  }
69.  data0_
70.  })
71.  ## Filter AE data based on subject selections
72.  output$table <-  renderDataTable(datatable({
73.  data <- as_tibble(adae2)
74.  if (input$USUBJID != "All") {
75.  data <- as_tibble(adae2) %>% filter(USUBJID== input$USUBJID)
76.  }
77.  data
78.  }, options = list(dom = 't')))
79.  ## Filter Lab data based on subject  selections
80.  output$table0 <- renderDataTable(datatable({
81.  data1 <- as_tibble(adlb)
82.  if (input$USUBJID != "All") {
83.  data1 <- as_tibble(adlb) %>% filter(USUBJID== input$USUBJID)
84.  #data <- filter(adae2, USUBJID==input$USUBJID)
85.  }
86.  data1
87.  }, options = list(dom = 't')))
88.  ## Filter Response data based on Subject selections
89.  output$table1 <- renderDataTable(datatable({
90.  data2 <- as_tibble(adrs)
91.  if (input$USUBJID != "All") {
92.  data2 <- as_tibble(adrs) %>% filter(USUBJID== input$USUBJID)
93.  #data <- filter(adae2, USUBJID==input$USUBJID)
94.  }
95.  data2
```

```
96.   }, options = list(dom = 't')))

97.   output$AEP <- renderPlotly({


98.   data <- as_tibble(adae2)

99.   adsl <- as_tibble(adae2) %>% distinct(USUBJID,TRTDURD,TRTSDT,TRTEDT)

100. if (input$USUBJID != "All") {

101. data <- as_tibble(adae2) %>% filter(USUBJID== input$USUBJID)

102. adsl <- as_tibble(adae2)  %>%  filter(USUBJID== input$USUBJID) %>%
       distinct(USUBJID,TRTDURD,TRTSDT,TRTEDT)

103. }

104. plot_ly(adsl) %>% add_trace(x = ~TRTDURD, y = ~USUBJID,  type = 'bar',

105. orientation = 'h', name = 'trt_duration',marker = list(color = 'red',opacity=0.5),width=0.5,hoverinfo ="text",

106. text = ~paste( 'TRT_END:',TRTEDT)) %>%  add_trace(x=~data$ADY, y = ~data$USUBJID,color=~data$AEDECOD,

107. type = 'scatter', mode = 'markers', yaxis = 'y2',  marker = list( size=10,opacity = 1, line = list(color='black',width = 1)),

108. hoverinfo = "text", text = ~paste('</br> AE:', data$AEDECOD, '</br> start:',

109. data$ASTDT,'</br> End:',data$AENDT)) %>%

110. layout(title = 'Adverse Events',    xaxis = list(title = "AE start and Trt. Dur.

111. (days)",dtick=20),

112. yaxis = list(side = 'left', title = 'Subjects', showgrid = FALSE, zeroline = FALSE,  type

113. = 'category'),yaxis2 = list(side = 'right', overlaying = "y",showticklabels = FALSE,

114. title = '', showgrid = FALSE, zeroline = FALSE))

115. })

116. }

117. #Run the application

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

## CONCLUSION

The paper aims to demonstrate the benefit of reactive data using R and R-Shiny. The vision is to have various application that involve inter departmental collaboration be built in this concept. This will help achieve the idea of “data on demand”.

## RECOMMENDED READING

- *R-shiny:*

  *https://shiny.rstudio.com/tutorial/written-tutorial/lesson1/*

- *R-Basics online course*

  *https://online-learning.harvard.edu/course/data-science-r-basics*

## CONTACT INFORMATION

Your comments and questions are valued and encouraged. Please contact me at:

<Pavan Vemuri>
<Alkermes
<sripavanv@gmail.com >