PharmaSUG 2018 - Paper AD-24

Application Development Framework for R/Shiny

Ashok Gunuganti, Pfizer Inc

ABSTRACT

This paper presents a basic framework for developing R/Shiny applications to accomplish various tasks as a better alternative than doing those tasks in SAS. The primary advantage of Apps developed in R/shiny over base SAS is that the applications can be published to the Web. They are also more dynamic, interactive, and are able to incorporate the versatile R packages into Shiny which makes them very powerful. Since Shiny applications can be deployed to the Web, the end user can use them with zero knowledge of R.

INTRODUCTION

Most statistical programmers work on daily basis with SAS datasets to perform tasks such as creation/review of SDTM data, ADaM data, and TLFs using SAS. R can also perform most of these tasks because it has great packages designed to work with SAS datasets. One of these packages is Shiny, a tool that provides a framework to develop GUI applications. Shiny's straightforward implementation of UI/Server interface makes it an ideal tool to create applications that can produce great efficiency for the repetitive tasks that a statistical programmer often encounters. The paper briefly introduces Shiny and presents a step-by-step approach to create a first Shiny application. The general framework presented in this paper can be extended to create robust and scalable Shiny applications. The focus of this paper is to demonstrate the building of Shiny Apps rather than a detailed discussion of Shiny components and how to deploy Apps on the Web

SETTINGUP YOUR APPLICATION DEVELOPMENT ENVIRONMENT

First, download and install the latest versions of

- R
- RStudio

After R and RStudio are installed successfully, launch RStudio and install the packages listed in **Error! Reference source not found.**. This is accomplished by typing install.Packages ("<package name>") at the RStudio console.

R Package	Purpose
Shiny	Shiny is an open source R package that provides an elegant and powerful Web framework for building Web applications using R. Shiny helps turn your analyses into interactive Web applications without requiring HTML, CSS, or JavaScript
Tidyverse	The tidyverse is a collection of R packages designed for data science. It includes the following packages that are relevant to the current context *haven* for importing SAS datasets *readxl* for importing .xls and .xlsx sheets *ggplot2* for creating graphics, based on the grammar of graphics *dplyr* for data manipulation *tidyr* – set of functions that help to tidy your data

R Package	**Purpose**
	stringr – provides functions to work with strings
DT	**DT** provides an R interface to the JavaScript library **DataTables**. R data objects (matrices or data frames) can be displayed as tables on HTML pages, and DataTables provide filtering, pagination, sorting, and many other features in the tables.
Summarytools	Provides tools to quickly and neatly summarize data
Plotly	For creating interactive Web-based graphics
qwraps2	A collection of (wrapper) functions the creator found useful for quickly placing data summaries and formatted regression results into '.Rnw' or '.Rmd' files

Table 1: R packages

SHINY APP TEMPLATE

A basic Shiny app program template is presented below:

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

The program consists of three main components:

1. *ui* - nested R functions that assemble an HTML user interface for your app
2. *server* - a function with instructions on how to build and rebuild the R objects displayed in the ui
3. *shinyApp* - combines ui and server into a functioning app.

A BASIC EXTENDABLE SHINY UI LAYOUT

There are several types of layouts available to organize the UI. In this paper, I layer tabpanels on top of each other that are contained in tabsetpanel. The user can navigate the tabpanels by clicking on the tabs shown in figure below (TAB1, TAB2, and TAB3). Each tabpanel then can be setup independent of the other tabs to perfrom specific tasks like create live summary tables, review of SAS datasets for data quality and compliance, (raw, SDTM, ADaM etc) or create graphical patient profiles etc.

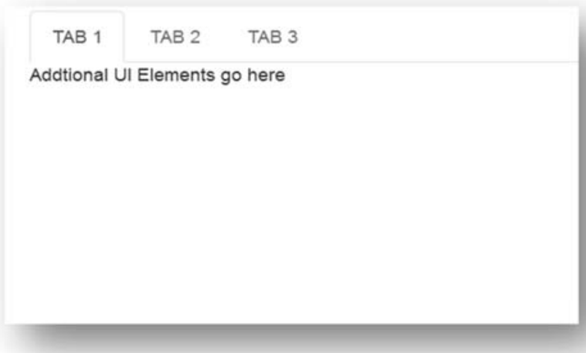


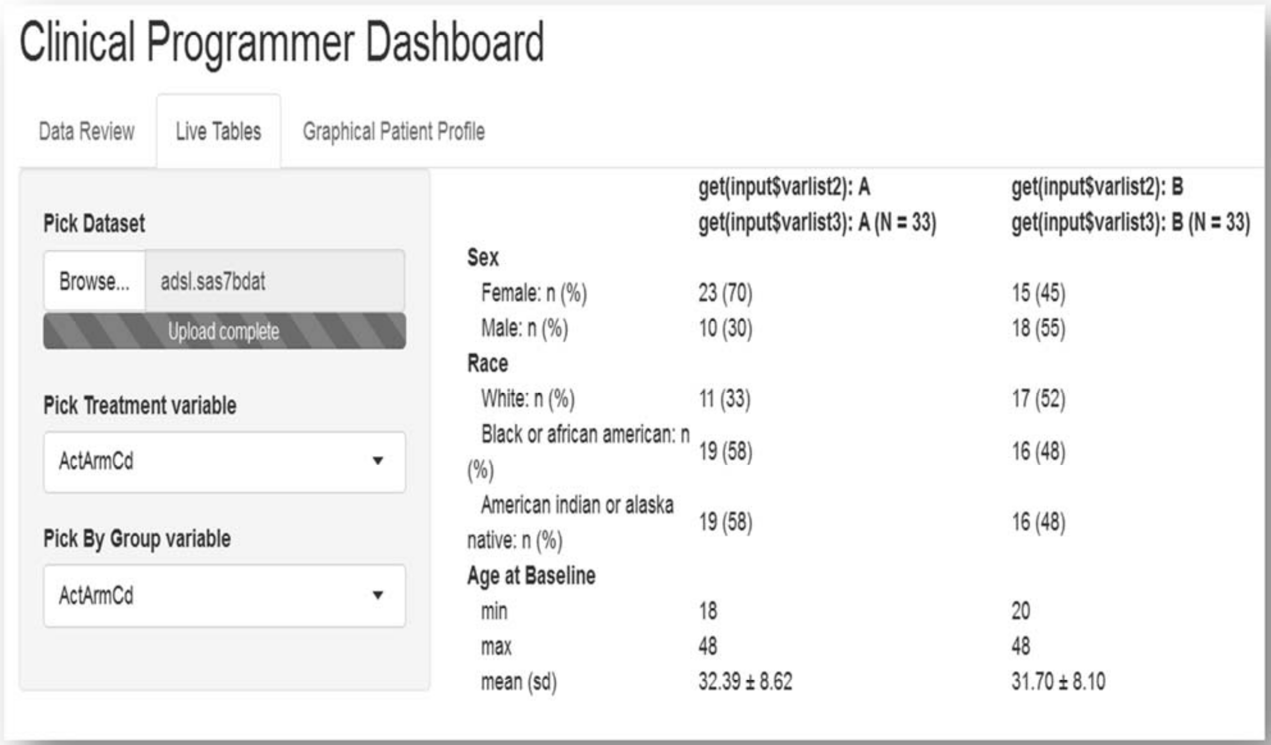
Figure 1: Basic UI

Following is the code to create the UI in figure 1, as you can see it is very simple and straightforward.

```
fluidPage( tabsetPanel(
tabPanel("TAB 1", mainPanel(uiOutput('Addtional UI Elements go here'))),
tabPanel("TAB 2", mainPanel(uiOutput('Addtional UI Elements go here'))),
tabPanel("TAB 3", mainPanel(uiOutput('Addtional UI Elements go here')))
                        )
          )
```

LIVE SUMMARY CREATION APPLICATION.

Until now I have talked about the basic UI and have gone through the steps on how to set up your development environment and install the packages you need. Next, you are going to use this setup to create your first Shiny App. In this section, I will go over how to start adding UI elements to one of the tabs and how to go about adding server code to dynamically interact with the UI. Figure 2 is the Live summary creation App which dynamically creates the baseline summary of sex, race, and age at baseline using any SAS datasets presented by the user. The only requirement is that the data to support summary is in the SAS dataset.



	get(input$varlist2): A get(input$varlist3): A (N = 33)	get(input$varlist2): B get(input$varlist3): B (N = 33)
Sex		
Female: n (%)	23 (70)	15 (45)
Male: n (%)	10 (30)	18 (55)
Race		
White: n (%)	11 (33)	17 (52)
Black or african american: n (%)	19 (58)	16 (48)
American indian or alaska native: n (%)	19 (58)	16 (48)
Age at Baseline		
min	18	20
max	48	48
mean (sd)	32.39 ± 8.62	31.70 ± 8.10

Figure 2: Live Summary Tables

The App has 3 UI elements

1. *fileInput* - Enables the user to choose file from a directory.
2. *selectInput* – Two drop downs to pick the variables (treatment variable and by group variable).
3. A main panel area where the server sends the summary.

The flow chart in figure 3 illustrates how the inputs are passed to the server which then generates the summary.

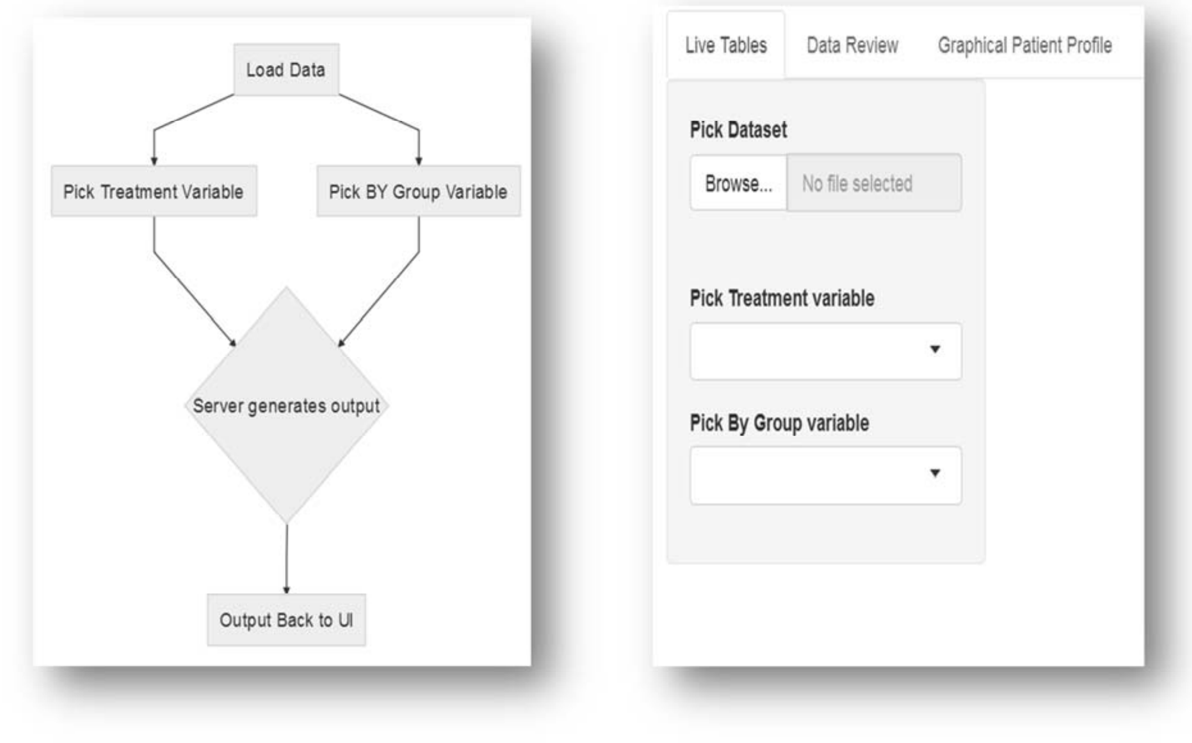


Figure 3: App Flow Chart and UI

Code for building UI

```
library(shiny)
tabPanel( "Live Tables",
sidebarLayout(

sidebarPanel(

 fileInput('sfile2', 'Pick Dataset',
accept=c('text/csv',
'text/comma-separated-values,text/plain',        1
'.sas7bdat')),

selectInput("varlist2","Pick Treatment variable",choices=c(colnames(data()))),  2

 selectInput("varlist3","Pick By Group variable",choices=c(colnames(data())))
                                                                                3
), # end sidebar panel

mainPanel(
uiOutput("datasum2")

                     4
)
)
) #tabpanel  end
```

UI Code Explanation

[1] Creates the UI file input box which enables the user to select the input dataset.

[2] After the dataset is loaded, it is passed to the server to populate selectInput with a list of variables from the dataset

[3] The second selectInput is also loaded with the variables from the dataset

[4] The variable selections from the input boxes are passed to the server where the summary table is created and passed back to the main panel section of the Ui which displays the table.

Server code

```
our_summary1 <-    1
  list("Sex" =
         list("Female: n (%)" = ~ qwraps2::n_perc0(Sex == 'F'),
              "Male: n (%)" = ~ qwraps2::n_perc0(Sex == 'M')
              ),
       "Race" =
         list("White: n (%)" = ~ qwraps2::n_perc0(Race == 'WHITE'),
              "Black or african american: n (%)" = ~ qwraps2::n_perc0(Race == 'BLACK O
R AFRICAN AMERICAN'),
              "American indian or alaska native: n (%)" = ~ qwraps2::n_perc0(Race == '
BLACK OR AFRICAN AMERICAN')
         ),
       "Age at Baseline" =
         list("min" = ~ min(Age),
              "max" = ~ max(Age),
              "mean (sd)" = ~ qwraps2::mean_sd(Age))

  )

data2 <- reactive({
file2 <- input$sfile2
if(is.null(file2)){return()} [1]    2
read_sas(file2$datapath)

})


observeEvent(input$sfile2,{
updateSelectInput(session,"varlist2",choices=c(colnames(data2())))    3
})


observeEvent(input$sfile2,{
updateSelectInput(session,"varlist3",choices=c(colnames(data2())))    4
})


observeEvent(input$varlist2,{
output$datasum2 <- renderUI(
{ if(is.null(data2())){return ()}    5
ds<- data2()
sumtabl <- summary_table(filter(ds,SAFFL == 'Y') %>% group_by(get(input$varlist2),get(
input$varlist3)) ,our_summary1)
sumtabl <- HTML(knit2html(text=capture.output(sumtabl), fragment.only=TRUE))
sumtabl})
})
```

Server Code Explanation

[1] Creates a template for the summary table that is sent to the UI - Function *n_perc0* computes n and percent, *Min* computes the minimum, *Max* computes the maximum and *mean_sd* function computes the mean and standard deviations for the variables referenced. Display labels are created in this section as well.

[2] Once the user picks a dataset, reactive functions are used to run the code to pick up the dataset, to use the datapath associated with the file, and read in the SAS dataset.

[3] Update the varlist1 with variable names from the dataset selected after the dataset load event is observed.

[4] Update the varlist2 with variable names from the dataset selected after the dataset load event is observed.

[5] Creates summary table which is called by the UI main panel.

As the user picks different treatment variables and by groups, the output is updated dynamically.

EXTEND SETUP

Based on the setup so far, you can create additional tabs that house additional Apps. In the following pages, a couple of such Apps are presented.

Simple Data Review Tool

Figure 4 shows the UI inputs of the app for quick review of data quality. Next the dataset selected by the user is loaded, the pick variable selectInput is populated with the variables from the dataset, and 1 or more check boxes appear with the distinct values of the variable chosen. Figure 5 shows the UI outputs which consists of a data table and stats summary of the subset below it.
These are updated dynamically as user changes inputs in UI.

User interface – Inputs

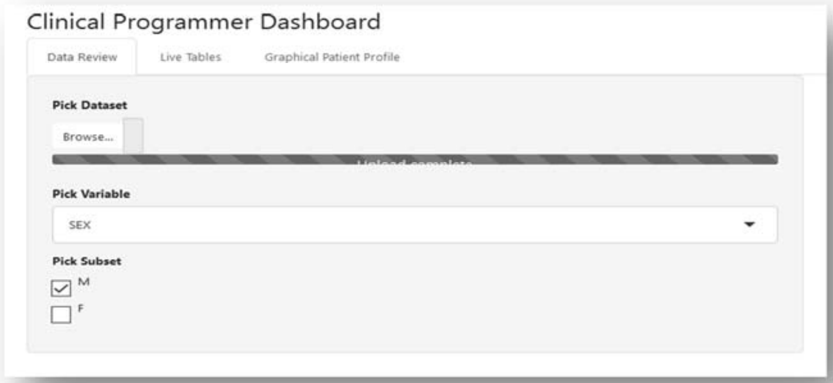
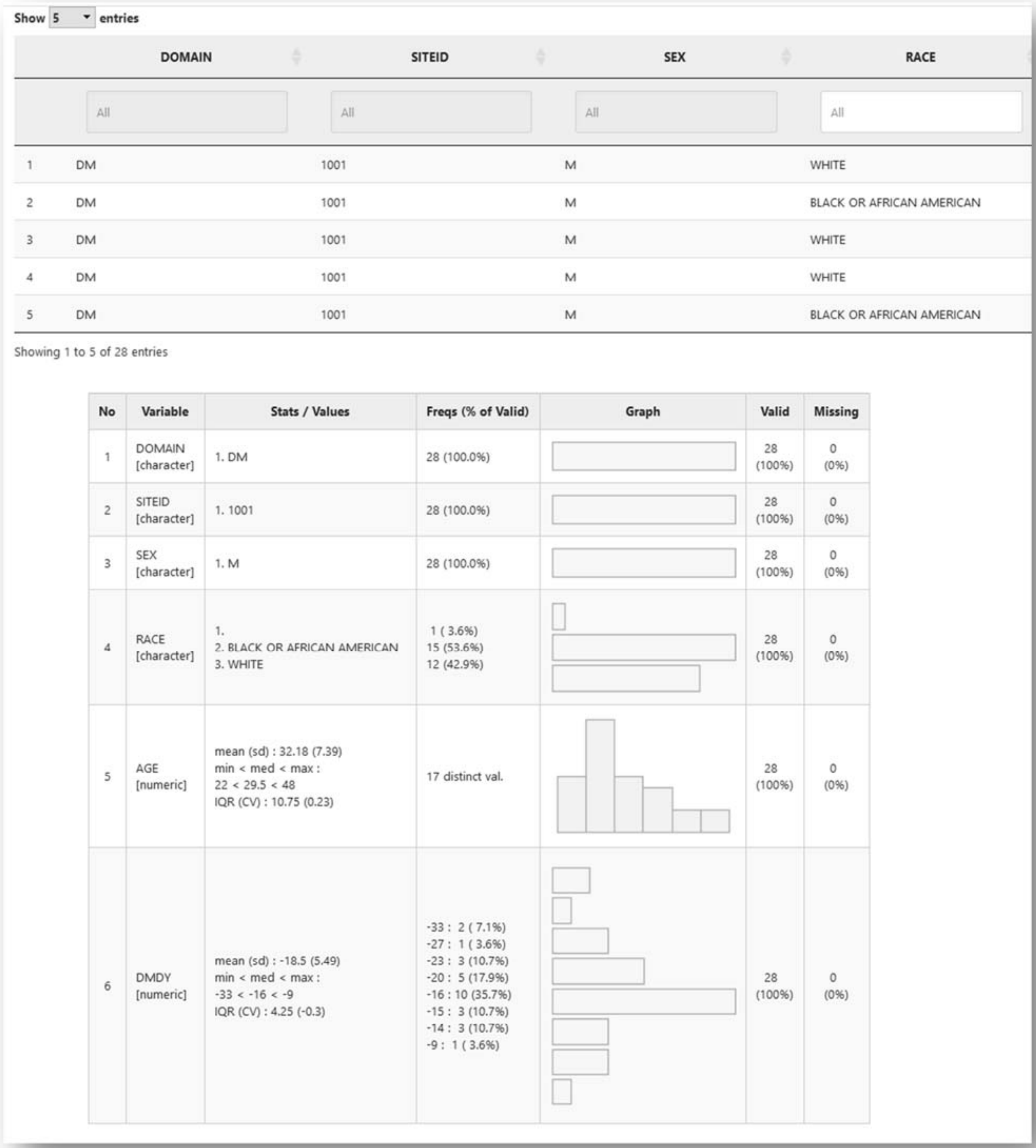


Figure 4: Data Review - Inputs

User Interface – Outputs



Show 5 entries

	DOMAIN	SITEID	SEX	RACE
	All	All	All	All
1	DM	1001	M	WHITE
2	DM	1001	M	BLACK OR AFRICAN AMERICAN
3	DM	1001	M	WHITE
4	DM	1001	M	WHITE
5	DM	1001	M	BLACK OR AFRICAN AMERICAN

Showing 1 to 5 of 28 entries

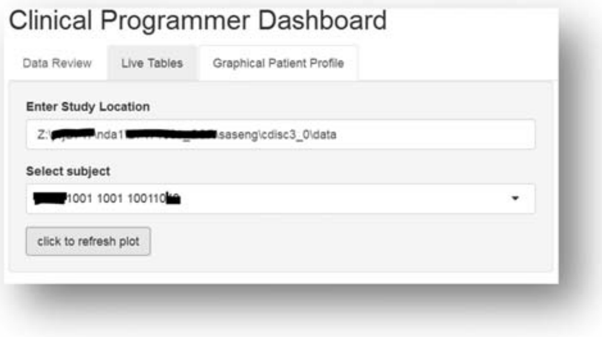
No	Variable	Stats / Values	Freqs (% of Valid)	Graph	Valid	Missing
1	DOMAIN [character]	1. DM	28 (100.0%)		28 (100%)	0 (0%)
2	SITEID [character]	1. 1001	28 (100.0%)		28 (100%)	0 (0%)
3	SEX [character]	1. M	28 (100.0%)		28 (100%)	0 (0%)
4	RACE [character]	1. 2. BLACK OR AFRICAN AMERICAN 3. WHITE	1 (3.6%) 15 (53.6%) 12 (42.9%)		28 (100%)	0 (0%)
5	AGE [numeric]	mean (sd) : 32.18 (7.39) min < med < max : 22 < 29.5 < 48 IQR (CV) : 10.75 (0.23)	17 distinct val.		28 (100%)	0 (0%)
6	DMDY [numeric]	mean (sd) : -18.5 (5.49) min < med < max : -33 < -16 < -9 IQR (CV) : 4.25 (-0.3)	-33 : 2 (7.1%) -27 : 1 (3.6%) -23 : 3 (10.7%) -20 : 5 (17.9%) -16 : 10 (35.7%) -15 : 3 (10.7%) -14 : 3 (10.7%) -9 : 1 (3.6%)		28 (100%)	0 (0%)

Figure 5: Data review - Outputs

Graphical Patient Profile

Figure 6 shows the UI inputs of the app for generating a graphical safety profile for subjects in a study. The user copies the location of the study SDTM data in the text box below. Safety SDTM datasets are copied to the work area following that *selectInput* is populated with the subject Id's from the DM domain. As the user selects a specific subject, a safety profile is generated as shown in Figure 7. As the user updates the subject selection the plots are updated dynamically.

User interface – Inputs



Clinical Programmer Dashboard

Enter Study Location

Z:\[redacted]nda1[redacted]saseng\cdisc3_0\data

Select subject

[redacted]1001 1001 100110[redacted]

Figure 6: Patient profile – Inputs

User Interface – Outputs

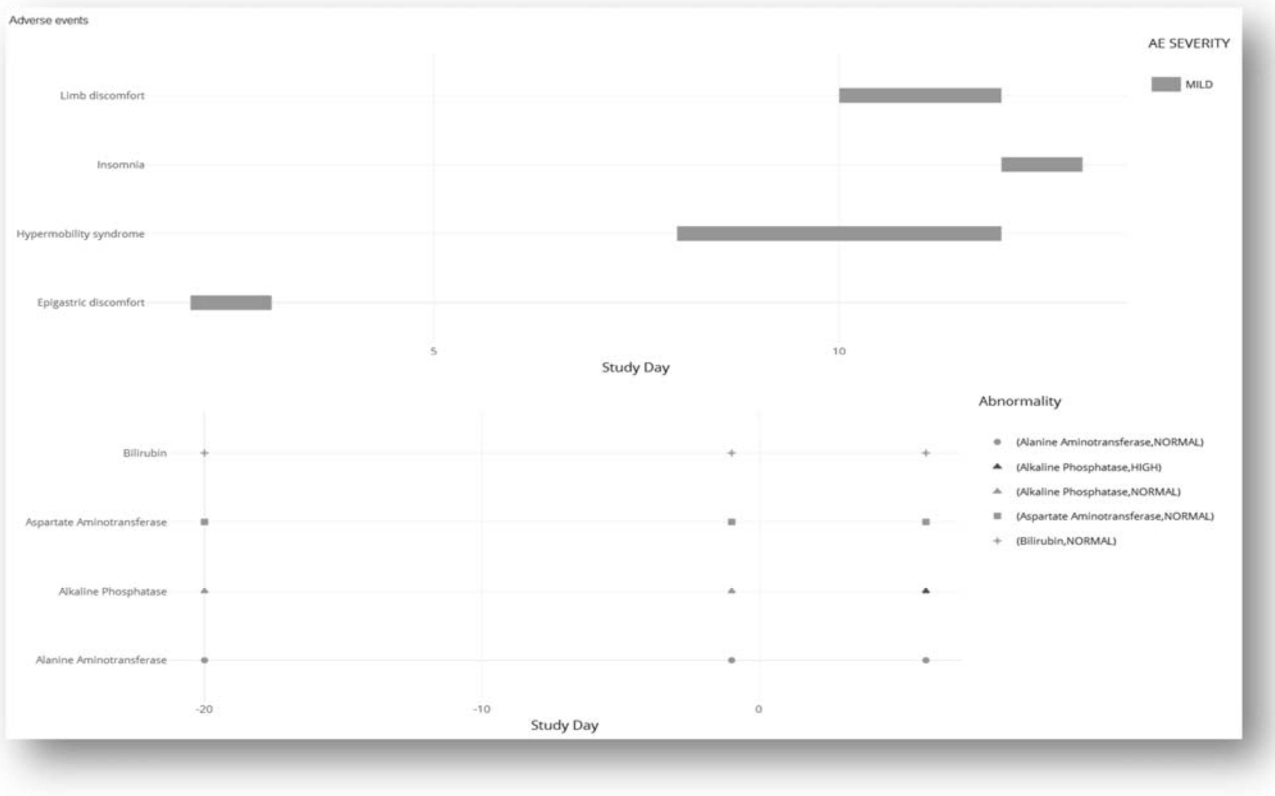


Figure 7: Patient profile – Outputs

CONCLUSION

Using the framework presented in this paper as a starting point, you can quickly develop dynamic Web-enabled applications that can be customized to a very high degree. The dynamic nature of the Shiny Apps makes them very powerful. They can automate route tasks and provide great efficiency. For a SAS programmer, Shiny Apps are similar to SAS macros, but they are much more dynamic whenever the user provides the macro parameters interactively.

REFERENCES

https://shiny.rstudio.com/images/shiny-cheatsheet.pdf

https://cran.r-project.org/web/packages/qwraps2/vignettes/summary-statistics.html

https://stackoverflow.com/questions/tagged/shiny

ACKNOWLEDGMENTS

The author would like to thank Liping Zhang for her invaluable input.

CONTACT INFORMATION

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

Ashok Gunuganti
Pfizer Inc.
Ashovardhan.gunuganti@pfizer.com