Paper DV05

End to End Interactive TLF using R Shiny

Rohit Banga, Bayer Inc., Toronto, Canada

ABSTRACT

This paper will demonstrate how to design an end to end web application to make interactive statistical outputs - Tables, Listings and Figures (TLF) using R Shiny Dashboard. The paper will describe in detail how SAS XPT files from different studies can be loaded in R. The read data will then be used to create interactive statistical reports - summary tables, graphs, and data listings at real time using. The paper will then describe how these reports can be downloaded in different formats such as CSV, PDF and RTF etc. Finally, the paper will detail how the shiny app can be deployed on web and made available to end users.

This method has great application in creating data science and data monitoring applications that can optimize data management tasks and statistical analysis workflow. Statistical reports and interactive dashboards can be created and made available in real-time, using open source tools, and without waiting for programmers to create reports.

INTRODUCTION

SAS and R are perhaps the two most widely used statistical analysis software within the pharmaceutical industry. SAS has been the predominant market leader in analytics space. SAS provides a vast array of statistical functions; it is easy to learn and has an excellent technical support. Most pharmaceutical companies have a validated SAS environment that produces GXPs compliant outputs to be submitted to FDA. However, SAS being a commercial enterprise software is an expensive option and is not always enriched with latest statistical functions.

R is an open source software and new packages/libraries are continuously added. It has seen an exponential growth in the past years. It also helps that students primarily use R during their academics and research, and they take this familiarity with R to their jobs. To a statistician working on a new statistical method, the language of choice is R when the algorithm is implemented and tested. It also offers support for emerging technologies such as advanced deep learning integrations. R is cost-effective and has a big online community but has no customer support as such.

R also offers extensive capabilities for producing interactive graphics - tables, listings and figures (TLF). But a lot of programmers struggle to design an end to end application using R. This paper will detail the different steps for implementing a web application using R Shiny. An application such as this can automate the TLF generation process and can instantly make TLFs available as soon as analysis data is made available.

In the subsequent sections, we will first develop our R Shiny application on our desktop using local R installation. We will begin by programming a functionality that will load analysis datasets (XPT files) of different studies in R and then write the code to generate TLF. We will then program different functionalities for displaying TLFs in an interactive way. Then we will write a program to help download the TLF in whatever format we want. After we finish the development of our R Shiny application, we will learn how to deploy that application on an Ubuntu machine using R Studio Shiny sever. This will make our application available to users on the web.

SETUP A SHINY APP

R Shiny is a web application framework for R and R Studio's Shiny server makes shiny applications available over the web. We will first develop an R Shiny application on our local machine and then learn to deploy this application on an ubuntu server to make it available on the web.

The code used in this example is uploaded on GitHub. This is the link to GitHub repository -

https://github.com/rohitbanga/rshinytlf

These are steps to setup a Shiny App –

- Create a directory on your local machine. Let's say we name it - "RShinyTLF". This is the directory in which we will store all our application files.
- Create an empty R file called - "app.R" in that directory. Please remember that this file can only be named "app.R". This will be our main application file, which will have a UI and server element.

The app.R file will have this basic structure -

```
# Load necessary packages
library(shiny)
library(shinydashboard)
…..

# Define UI element for
ui <- dashboardPage(

        # Sidebar Panel for Displaying – Study selection dropdown and TLFs
        dashboardSidebar (
                sidebarMenu(
                        selectInput(<<Drop down for Study List>>)
                        , menuItem(<<Display Selection button for Table 1>>)
                        , menuItem(actionButton(<<Display a button to refresh database>>))),

        # Main panel for displaying outputs
        dashboardBody (
                tabItems(
                        tabItem(<<Display contents of Table 1>>))))

# Define server logic to define events and run programs for creating TLFs
server <- function(input, output, session) {

observeEvent(input$study, {
        <<Load Study Datasets when a study is selected from dropdown >>
        <<Make output tables>> })
```


```
observeEvent(input$refresh_btn, {
        <<Run a code when a Refresh Database button is pressed >> })

}

# Run the app

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

LOADING STUDY DATA IN R ENVIRONMENT

Now that we have a directory to store our application files and an app.R skeleton, we will program the functionality to load data from different studies and make it available to our application. For this purpose, we will write a R program - “load_db.R” that will perform the following steps -

- Input <selected_study> parameter from app.R program
- Convert SAS analysis datasets from <selected_study> folder into R data frame. We will utilize function sasxport.get from Hmisc package to read XPT files and convert them into R data frames.
- Run code of each TLF and create R data frames corresponding to each TLF. We will create data frames such as table_1, listing_1, figure_1 etc. and these data frames would be used in app.R to display different TLF on the shiny dashboard.
- Store analysis data frames and TLF data frames in a R workspace image. We want to save the data frames in a workspace image because we want the results to be available immediately to users when they select a study. If the SAS analysis datasets change, then we will call this program again to re-load the new datasets and remake the TLF data frames. (we can call this program through a button – Refresh Database - that we will make on the dashboard sidebar).
- Call a program - “output_tlf_rtf.R” that will convert table_1, listing_1, figure_1 etc. data frames into RTF files (or any other format the user desires) and store it in the current directory. Then we will program a button – Download All Files - on the dashboard sidebar which will help download the results of output_tlf_rtf.R program.

load_db.R file will have this basic structure –

```
# Load necessary libraries
library(Hmisc)

#Import datasets
adsl = sasxport.get(paste0(selected_study,"/adsl.xpt"))
….
#Run Code for Table 1
source("table_1.R",local = TRUE)

#Save imported study data frames and TLF data frames into R image file
save.image(file = paste0(selected_study,".RData"))

# Run program to use TLF dataframe and save them in RTF format
source("output_tlf_rtf.R")
```

MAKE TLF PROGRAMS

The program “load_db.R” calls individual TLF programs. These programs use the analysis data frames and create output TLF data frames that will be displayed on the dashboard. A simplified example of such a program is shown below –

```
# Make count of gender by treatment
table_1a = sqldf("select trt01a
                        ,sex as Category
                        ,count(distinct usubjid) as count
                 from adsl
                 group by trt01a
                 ,sex"
)

#Transpose the results
table_1 = dcast(table_1a, Category ~ trt01a,value.var = "count")
table_1_columns = colnames(table_1)
```

We will make individual programs for each output such as this. Please note that this program only calculates the numbers. All manipulations done to visualize the output in a better way are done in app.R.

MAKE PROGRAM TO SAVE ALL TLF AS RTF (OUTPUT_TLF_RTF.R)

This program is also called from load_db.R program. The purpose of this program is to convert all TLF data frames and save them as RTF. This program will be called from a button – Download All Files – from the dashboard sidebar. A simplified example of such a program is shown below –

```
library(rtf)

outfile = paste0("outputs_",selected_study,".rtf")


rtf <- RTF(outfile)
addHeader(rtf,title="Section 1 - Tables", subtitle="This Section contains all tables")
addParagraph(rtf, "Table 1 - Demographic Data:\n")
addTable(rtf, table_1)

addPageBreak(rtf, width=8.5, height=11, omi=c(1,1,1,1))
addHeader(rtf,title="Section 2 - Figures", subtitle="This Section contains all figures")
addParagraph(rtf, "Figure 1 - Cog Analysis - Mean change for baseline by visit:\n")
addPlot(rtf,plot.fun=print,width=5,height=4,res=300, fig_1)

done(rtf)
```

- This program uses RTF package to convert R data frames into RTF.
- Programmers can use addHeader, addParagraph, addTable, addPlot and addPageBreak functions to add various elements to RTF

- Programmers can of course make individual RTF and combine them in a zip file. The idea is that as soon as data is refreshed, this code generates all outputs and make them available to end user from the dashboard.

MAKING DROPDOWN FOR SELECTING DIFFERENT STUDIES

Since we have the load_db.R in place, we have the TLF data frames loaded into our environment and TLF output ready to be downloaded. Now we need to configure our dashboard and display TLF from different studies in our application. The first step in making the dashboard is to program a dropdown with different study names. Upon selecting a study from dropdown, the following steps will happen -

- Display the list of studies in the dropdown (using selectInput function).
- The application will then search for the R workspace image consisting of TLF data frames corresponding to the selected study (using observeEvent function)
- Load the study's R workspace image if available and make TLF data frames available in current environment.
- If study workspace image is not found, then run program - "load_db.R" to make study's R workspace image.

This and the subsequent functionalities will be programmed in app.R. A simplified example of implementation of this functionality in app.R is shown below –

```
#Define list of studies.
study_list = c("Demo Study 1", "Demo Study 2" )

#Define the dropdown in the UI
ui = dashboardPage(
        dashboardSidebar(
                sidebarMenu(
                        selectInput("study", "Study Name:", study_list)
                        ….)))



# Define server logic ----
server <- function(input, output, session){

        #Initialize selected_study object
        selected_study <<- 'none'

        #Define the Event which gets triggered when a study is selected
        observeEvent(input$study, {
                print("Setting selected_study")
                #Set selected_study object equal to study selected from dropdown
                selected_study <<- input$study

                if (selected_study!= "none"){
                #Check if study workspace image exists
                if (!file.exists(paste0(selected_study,".RData"))) {
                        #If image doesn't exist run the load_db.R program to create the image.
                        showModal(modalDialog("Refreshing Database. Please wait 2-3 minutes until the
                                process finishes.", footer=NULL))
                        source("load_db.R")
                        removeModal()
                            }
```

```
                    #Load selected study workspace image
                    load(paste0(selected_study,".RData"))

                output$table_1 <<- …
                output$listing_1 <<- …

}
```

- The observeEvent(input$study) function gets triggered every time input study changes.
- When observeEvent(input$study) function gets triggered, the corresponding data for the study gets loaded in the R environment and the output TLFs are re-visualized.

DISPLAY TLF ON DASHBOARD

We will now use the TLF data frames (table_1, listing_1, figure_1etc.) created from individual TLF programs (table_1.R, listing_1, figure_1.R etc.) to visualize the outputs on the R Shiny Dashboard.

- This is how the UI section of app.R will be organized -

```
ui = dashboardPage(
        dashboardSidebar(
                sidebarMenu(
                  menuItem("Table 1 - Demographic Information", tabName = "table1", icon = icon("angle-right"))
                , menuItem("Listing 1 - Adverse Events", tabName = "listing1", icon = icon("angle-right"))
                ,menuItem("Figure 1 - Cog Analysis "   , tabName = "figure1"     , icon = icon("angle-right"))
                ….)))
                  ,dashboardBody(
                    tabItems(
                      tabItem(tabName = "table1"    , h2("Table 1 - Demographic Information")  ,div(style =
'overflow-x: scroll;white-space: nowrap;', DT::dataTableOutput("table_1")%>%
withSpinner(color="#0dc5c1")))
                    ,tabItem(tabName = "listing1"  , h2("Listing 1 - Adverse Event")         ,div(style = 'overflow-x:
scroll;white-space: nowrap;', DT::dataTableOutput("listing_1")%>% withSpinner(color="#0dc5c1"))
                                        , h2("Listing 1 - Legend")              ,div(width = 'overflow-x: scroll;white-
space: nowrap;', DT::dataTableOutput('legend_l1')%>% withSpinner(color="#0dc5c1"))
                                )
                    ,tabItem(tabName = "figure1"    , h2("Figure 1 - Cog Analysis - Mean change for baseline by
visit")  ,div(style = 'overflow-x: scroll;white-space: nowrap;', selectInput("param", "Parameter Name:",
figure_1_param)
```

- In the sidebar, we are using menuItem function to display different tabs corresponding to different outputs. We use tabName parameter to assign a unique name to each output. The tabName in sidebar is used to link to tabItem in dashboard body.
- In the Dashboard body, we define tabItems corresponding to each tabName in the sidebar. Multiple outputs on the same page will have the same tabName.
- We use the option withSpinner to display a loading icon till the time the output loads.

This is how the Server part of app.R will be organized

```
output$listing_1 <<- DT::renderDataTable({
 options(DT.options = list(dom = 'Bfrtip'
                    ,class = 'cell-border stripe dt-center nowrap'
                    ,extensions = list('Buttons' = NULL
```


```
                                    ,'FixedColumns' = NULL)
                        ,buttons = list(list(extend = 'copy')
                                    ,list(extend = 'csv',filename= 'Listing 1 - Adverse Events')
                                    ,list(extend = 'pdf',filename= 'Listing 1 - Adverse Events')
                                    ,list(extend = 'print')
                                    ,list(extend = 'excel',filename= 'Listing 1 - Adverse Events'))
                        ,paging = TRUE
                        ,pageLength = 50
                        ,scrollX = TRUE
                        ,fixedColumns = list(leftColumns = 5)
    ))
    dat = DT::datatable(listing_1
                        ,rownames = FALSE) %>%
      DT::formatStyle("aeser"
                      ,target = "row"
                      ,backgroundColor = DT::styleEqual(c('Y'), c('red'))
      )
    return(dat)
  })
```

- We use the options dom = 'Bfrtip' and buttons to enable downloading the output in CSV, PDF, EXCEL format.
- We used the option fixedColumns to freeze the first column.
- We use the option formatStyle to conditionally color rows of the output.

MAKE REFRESH DATABASE BUTTON

We need to program a button that will call the load_db.R program when clicked. This is needed to reload datasets and recalculate outputs if the study data gets updated.

We will make the button in the UI Sidebar using the actionButton function and then assign an observeEvent to it in the Server section of app.R as shown below -

```
ui = dashboardPage(
        dashboardSidebar(
                sidebarMenu(
                        menuItem(actionButton("refresh_btn", "Refresh Database"))
                        ….)))


# Define server logic ----
server <- function(input, output, session){

  observeEvent(input$refresh_btn, {
      showModal(modalDialog("Refreshing Database. Please wait 2-3 minutes until the process finishes.",
                            footer=NULL))
    source("load_db.R")
    removeModal()
    session$reload()
  })
}
```

MAKE “DOWNLOAD ALL FILES” BUTTON

Just as refresh database button, we need an additional button to help download all outputs in the RTF format when clicked.

We will make the button in the UI Sidebar using the downloadButton function and then assign an observeEvent to it in the Server section of app.R as shown below -

```
ui = dashboardPage(
        dashboardSidebar(
                sidebarMenu(
                        menuItem(downloadButton("all_rep", "Download All Files"))
                        ….)))


# Define server logic ----
server <- function(input, output, session){

  output$all_rep =
    downloadHandler(
      filename = function(){
      paste0("outputs_",selected_study,".rtf")},
      content = function(file){
        print(paste0("Selected Report -","outputs_",selected_study,".rtf"))
        file.copy(paste0("outputs_",selected_study,".rtf"),file)}
    )
}
```

INSTALLING RSTUDIO SHINY SERVER

To make our application available on the web, we need to install RStudio Shiny Server on a server machine. The recommended solution is to install RStudio Shiny Server on an Ubuntu machine. One can provision an ubuntu virtual machine on AWS (Amazon EC2), Azure (Microsoft) or Google Cloud Platform. Provisioning an ubuntu server on a cloud platform or a personal computer is outside the scope of this paper (For instance, follow steps in Reference point 2 to get started with AWS Linux instance). Please note that you need at least 1GB of RAM to install some packages in R. Therefore, it is possible that the most basic configuration machines offered by the cloud providers would not work (For example, a t2-micro EC2 instance of AWS will not able to handle this installation). After an ubuntu server is up and running, make sure that the security groups/firewall of the machine allows access to the following ports –

- Port 22 for SSH
- Port 3838 for RStudio Shiny Server

Make sure that you can SSH into the machine. Mac users can use Terminal and Windows users can use “Putty” to access the machine. Once inside the ubuntu machine, follow these steps to install Base R, R Shiny Package and RStudio server.

(Please note that the first line of code below is valid for installing R on Ubuntu 18.04. For installing R on other Ubuntu versions, please see reference point 5 and change the below line accordingly)

```
sudo add-apt-repository 'deb https://cloud.r-project.org/bin/linux/ubuntu bionic-cran35/'
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys E298A3A825C0D65DFD57CBB651716619E084DAB9
sudo apt update && sudo apt upgrade
sudo apt install r-base


sudo su - \
```

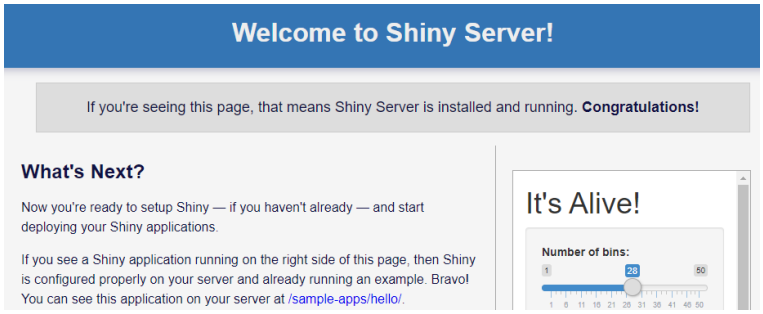
-c "R -e \"install.packages('shiny', repos='https://cran.rstudio.com/')\""

sudo apt-get install gdebi-core
wget https://download3.rstudio.org/ubuntu-14.04/x86_64/shiny-server-1.5.9.923-amd64.deb
sudo gdebi shiny-server-1.5.9.923-amd64.deb

Once the above steps are completed, a shiny-server application would be installed and up and running. Verify if everything is installed correctly by visiting this URL in your browser –

http://<Your Virtual Machine IP Address>:3838/

You will see a welcome to R Shiny Server page as shown below -



DEPLOY APP ON SHINY SERVER
Now that we have a shiny server up and running, we need to deploy the app that we developed on this server. These are the steps to deploy the app on the server

- SSH into the ubuntu machine and run this command to make directory /srv/shiny-server writable–

  sudo chmod -R 777 /srv/shiny-server

- Copy the “RShinyTLF” directory from the local machine to /srv/shiny-server. Now, this can be complicated for programmers who are not used to programming in linux or dealing with virtual machines. The simplest solution is to download an FTP client such WinSCP (https://winscp.net/). When you enter the connection details on your virtual machine on this FTP client, it will give you a window’s like interface for the ubuntu server. Use this tool to copy over RShinyTLF directory to /srv/shiny-sever.

- Install R packages on the server.

  sudo chmod -R 777 /usr/lib/R/site-library
  sudo chmod -R 777 /usr/local/lib/R/site-library
  R
  install.packages("Hmisc", dependencies=TRUE)
  install.packages("sqldf", dependencies=TRUE)
  install.packages("plyr", dependencies=TRUE)
  install.packages("dplyr", dependencies=TRUE)
  install.packages("reshape2", dependencies=TRUE)
  install.packages("DT", dependencies=TRUE)
  install.packages("expss", dependencies=TRUE)
  install.packages("shiny", dependencies=TRUE)
  install.packages("shinydashboard", dependencies=TRUE)

```
install.packages("shinyWidgets", dependencies=TRUE)
install.packages("shinyjs", dependencies=TRUE)
install.packages("shinycssloaders", dependencies=TRUE)
install.packages("ggplot2", dependencies=TRUE)
install.packages("rtf", dependencies=TRUE)
```

- Update shiny-server configuration file. In the ubuntu terminal, write this line -

```
sudo nano /etc/shiny-server/shiny-server.conf
```

- Make sure the shiny-server.conf looks like this (added app_dir and modified log_dir parameters) –

```
# Instruct Shiny Server to run applications as the user "shiny"
run_as shiny;

# Define a server that listens on port 3838
server {
  listen 3838;

  # Define a location at the base URL
  location / {

    # Host the directory of Shiny Apps stored in this directory
    site_dir /srv/shiny-server;

    app_dir /srv/shiny-server/RShinyTLF;
    log_dir /var/log/shiny-server/RShinyTLF;

    # When a user visits the base URL rather than a particular application,
    # an index of the applications available in this directory will be shown.
    directory_index off;
  }
}
```

Press Ctrl+X, then press “Y” and then Enter.

- Restart shiny-server

```
sudo systemctl restart shiny-server
```

- Our app is visible on –

http://<Your Virtual Machine IP Address>:3838/

- Access to this machine can be made available only to authorized users. One can also build a user authentication system before the application using RStudio server pro or 3rd party tools (See references Point 4).

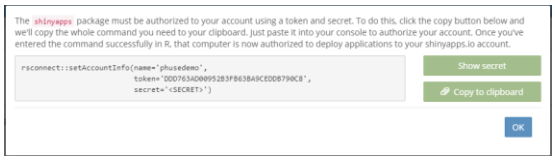
So, we have successfully developed and deployed our R Shiny Application on a virtual machine!

PUBLISHING APP ON SHINYAPPS.IO

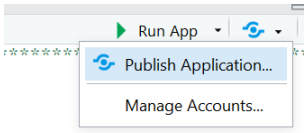
Users can also publish their apps on Shinyapps.io if a virtual machine is not available. Shinyapp.io will host your RShiny applications for free which is an excellent option for those starting out. The downside of this approach is that the app will be open to public. Shinyapps.io also have paid options if you want to use shiny apps in a production environment.

Programmers can deploy their R Shiny application directly from their local R Studio installation to shinyapps.io. To do that you just need to configure shinyapps.io token and secret in R Studio. These are the steps to publish your application on shinyapps.io –

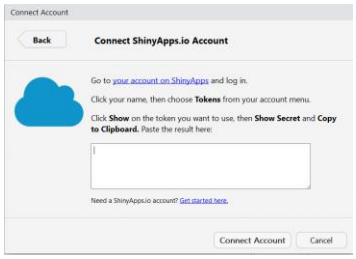
- Sign up for an account on https://www.shinyapps.io/
- Upon sign up you will see a getting started screen that will explain how to publish on shinyapps.io using R.
- To publish using R Studio, note down the token and secret from the shinyapps.io admin portal → Accounts → Token → Show secret.



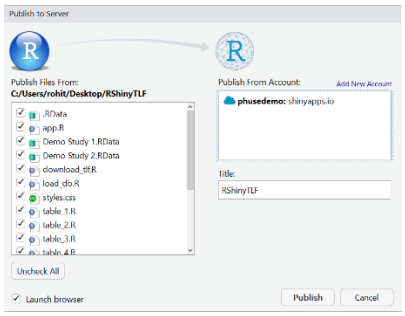
- Click on copy to clipboard and copy the code.
- In your local R Studio, click on the blue icon and click publish application



- In “Publish From Account”, click on “Add new account” → ShinyApps.io and paste the code copied above



- You will see the connected shinyapps.io account. Select the connected account and click on Publish.



- Your application should open automatically in the browser upon success deployment. For example, the application made in the paper is deployed here –

  https://phusedemo.shinyapps.io/RShinyTLF/

CONCLUSION

This paper outlined all the steps to develop an end-to-end application using R Shiny. We saw how easily and quickly we can build a web application that loads SAS XPT Files and creates interactive dashboards. We also saw how this application can be deployed on the web and made available to end users. Such applications can greatly streamline the statistical report generation process.

REFERENCES

1. Ubuntu Virtual Machine
    a. Amazon EC2 - https://aws.amazon.com/ec2/
    b. Azure Virtual Machines - https://azure.microsoft.com/en-ca/services/virtual-machines/
    c. Google Cloud Virtual Machines - https://cloud.google.com/compute/docs/instances
2. Getting Started with Amazon EC2 Linux Instances - https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2_GetStarted.html
3. How to build a Shiny App - https://shiny.rstudio.com/articles/build.html
4. Adding Authentication to Shiny Server in 4 Simple Steps - https://auth0.com/blog/adding-authentication-to-shiny-server/
5. Ubuntu packages for R - https://cran.r-project.org/bin/linux/ubuntu/README.html