PharmaSUG 2018 - Paper HT-03

Shiny Happy People: Using RShiny and SDTM Data to generate a Quick Interactive Dashboard

Saranya Duraismy, Nate Mockler, Biogen

ABSTRACT

This workshop will show how to use the "shiny" package in R and SDTM data to create an open-source, interactive and modular dashboard that can take your SDTM data and create insights that can saved time and much less work. A basic understanding of R and SDTM data is recommended.

INTRODUCTION

With the torrent of data cascading all around us, trying to make sense of it has required more than standard static visualizations. There is a need to modify displays in real-time, often to tease out the insight that previously would take multiple revisions and lots of time. A multitude of products have been developed to fill that need, but for many one has to either learn a new language, or deal with limited functionality.

Shiny, a package developed by RStudio, allows you to use a language you already know – R – to create interactive visualizations that can be either used individually or in the cloud. In this hands-on workshop, we will explore how to create a basic input, and then apply it to a basic dashboard. A basic knowledge of R is required.

WHAT MAKES A GOOD DASHBOARD?

Before starting a new dashboard, we should explore quickly what makes a good dashboard. To do that, let's first explore what makes a *bad* dashboard. One of the thought leaders in the area of information is Stephen Few, and his company Perceptual Edge. One of his white papers *Common Pitfalls in Dashboard Design* lists 13 pitfalls (Few, 2018):

- Exceeding the boundaries of a single screen
- Supplying inadequate context for the data
- Displaying excessive detail or precision
- Expressing measures indirectly
- Choosing inappropriate media of display
- Introducing meaningless variety
- Using poorly designed display media
- Encoding quantitative data inaccurately
- Arranging the data poorly
- Ineffectively highlighting what's important
- Cluttering the screen with useless decoration
- Misusing or overusing color
- Designing an unappealing visual display

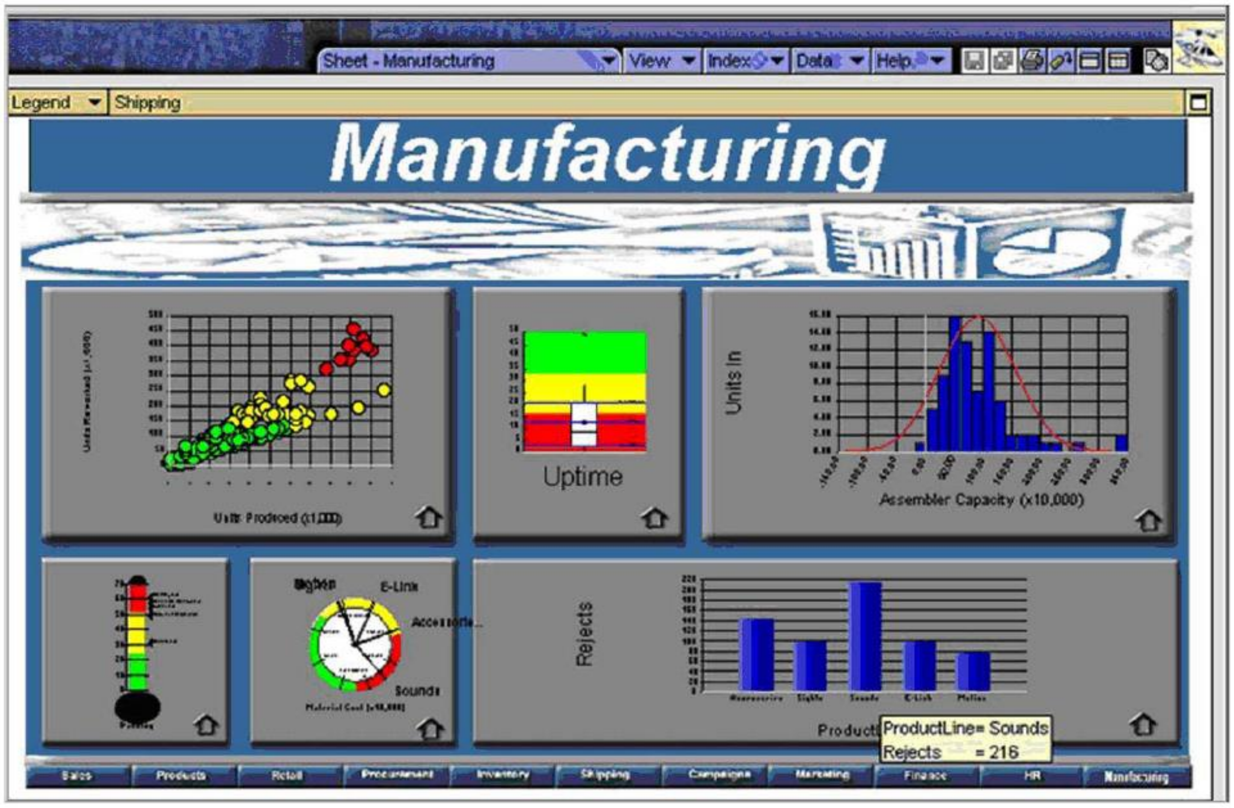


Figure 1: This thing is hideous, and unhelpful (Few, 2018a)

So what makes a good dashboard? Few describes a good dashboard as one that:

- Easily Scan the Big Picture
- Zoom in on important specifics
- Link to Supporting Details

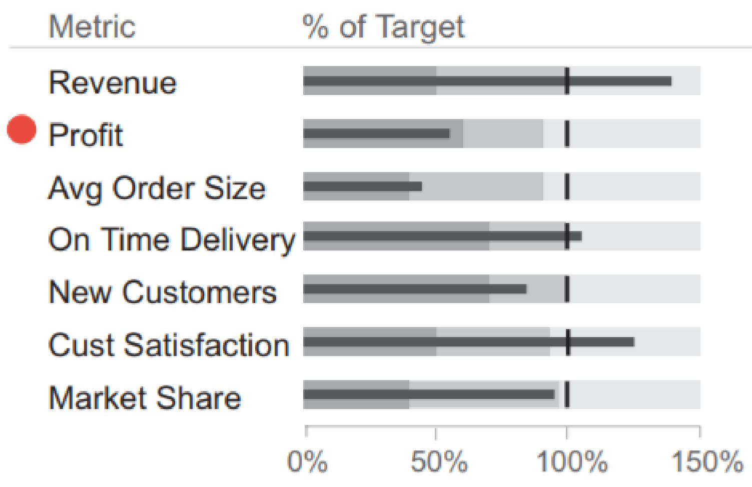


Figure 2: I can easily tell Profit is Bad (Few, 2018b)

Now that we have that discussed, let’s discuss Shiny.

INTRO TO SHINY

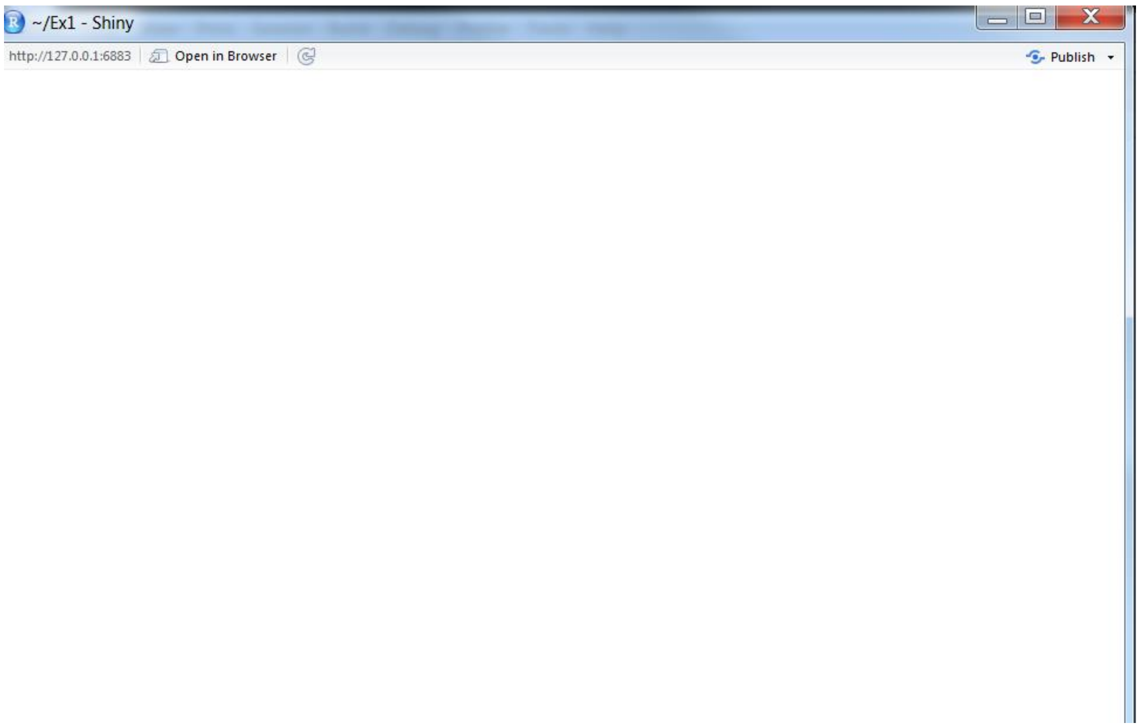
Shiny is an R package that makes it easy to build interactive web apps straight with R. Created with Javascript and CSS, we can create web apps right from R. The best part is that we can do it right within the language – no additional work required.

The way that the package works is that it runs a function – shinyApp, that takes a ui object and server function. The server can be your own computer (the default), or uploaded to a cloud-based service that will let other users look at it.

The structure of a basic shiny function is shown below:

```
library(shiny)

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

The first line loads in the shiny library. Following that, it creates the ui object with nothing in it. It initiates the server function as well – calculating nothing, and then runs the application.



Output 1: Not Much to Look at, yet…

Not very exciting, but it will be the canvas in which you will paint your dashboard (figuratively: don't paint your screen, please).

Most of the work in getting a Shiny output is defined in the relationship between the UI object and the Server function, as shown below:

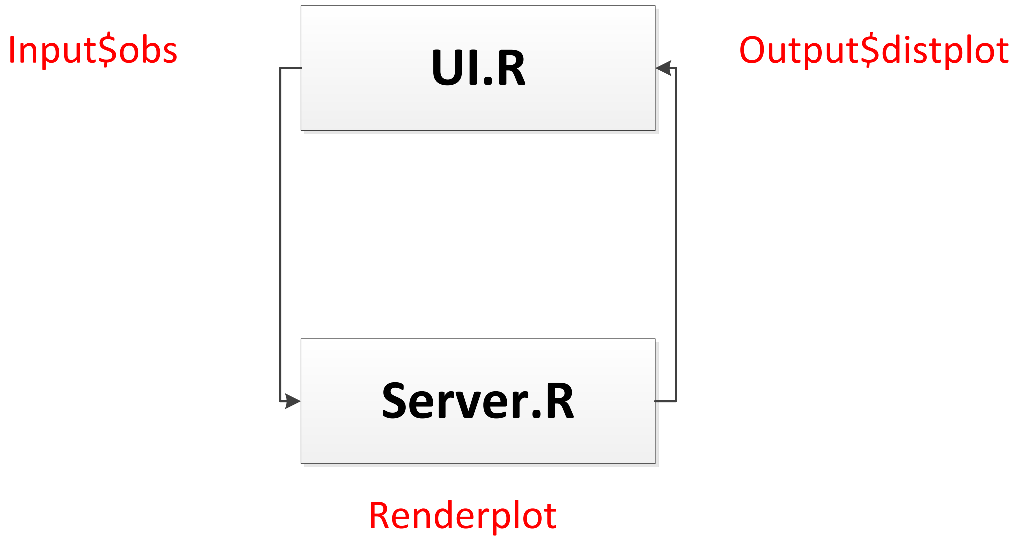


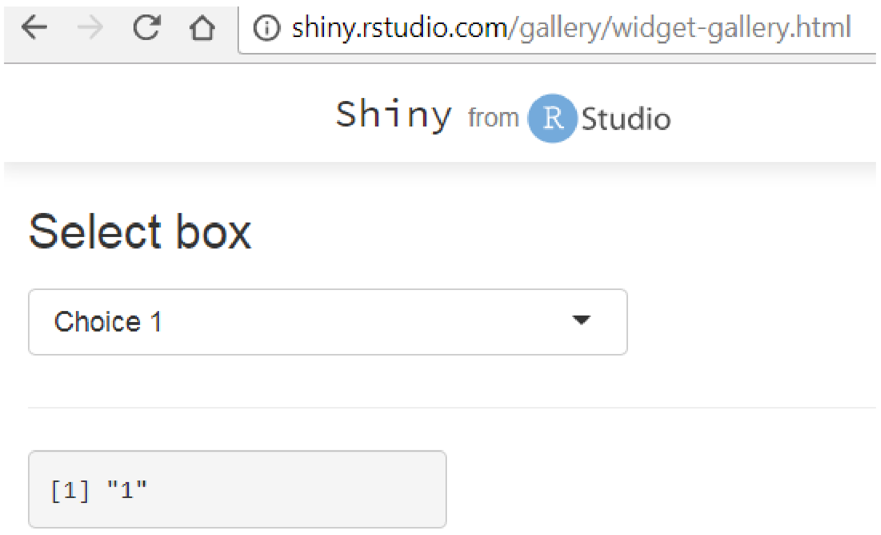
Figure 3: UI and Server – like Chocolate and Peanut Butter

The key to understanding shiny is in between the relationship between the UI object and the Server function. The UI object is responsible for accepting the inputs of the user, and communicating the outputs. The Server object is responsible to doing all the “R Stuff” (generating the plots, tables, doing the calculations, etc). Understanding that is the key to working with Shiny.

Another key to understanding Shiny are “widgets”, functions that will create html objects and communicate the inputs that the user select. An example is

```
selectInput("select", label = h3("Select box"),
    choices = list("Choice 1" = 1, "Choice 2" = 2, "Choice 3" = 3),
    selected = 1),
```

which generates:



```
[1] "1"
```

Output 2: A box that gives you three choices

Now that we have that, let’s create a simple dropbox and a table.

A SIMPLE DROPBOX AND A TABLE

Let’s review what we have:

```
library(shiny)
ui <- fluidPage(
)
server <- function(input, output) {
}
# Run the application
shinyApp(ui = ui, server = server)
```

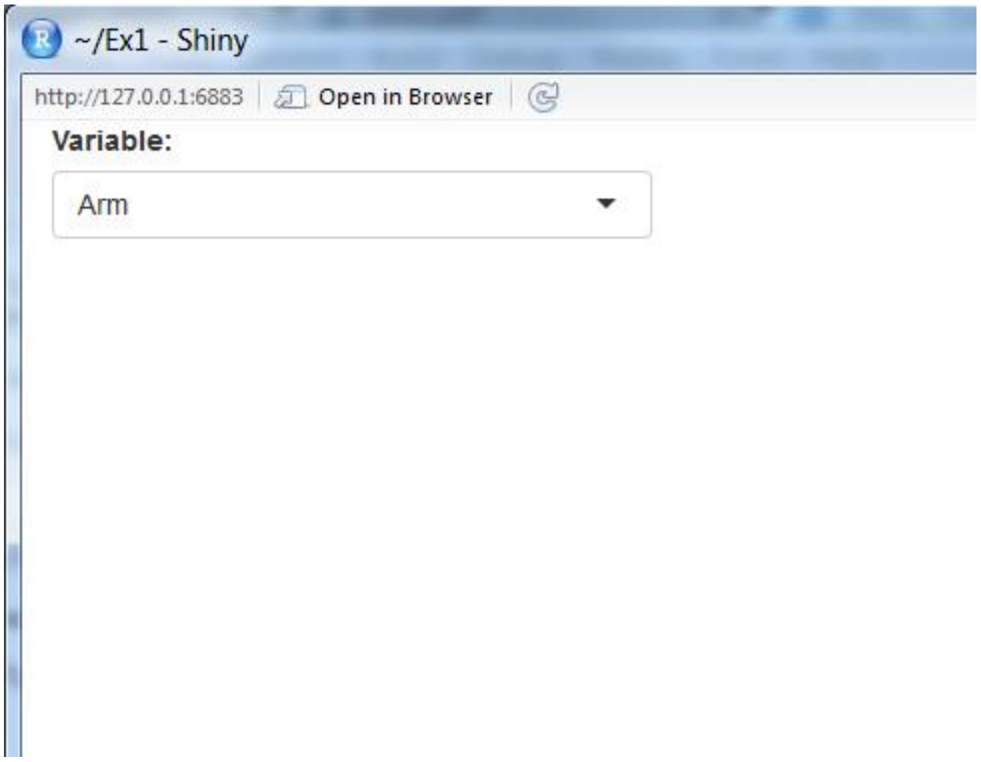
Let’s first add some data to the workspace. This will not change any of the output:

```
library(shiny)
library(haven)
dm <- read_sas("J:/drug/study/R_training/dm.sas7bdat")

ui <- fluidPage(
)

server <- function(input, output) {
}


# Run the application
shinyApp(ui = ui, server = server)
```

Now that we have done that, the next step is to create the input. We’ll use the selectInput function discussed prior and add it to the ui object.

```
library(shiny)
library(haven)

dm <- read_sas("J:/drug/study/R_training/dm.sas7bdat")

ui <- fluidPage(  selectInput("variable", "Variable:",
                              c("Arm" = "ARM",
                                "Country" = "COUNTRY",
                                "Race" = "RACE"))
              )

server <- function(input, output) {

}

# Run the application
shinyApp(ui = ui, server = server)
```

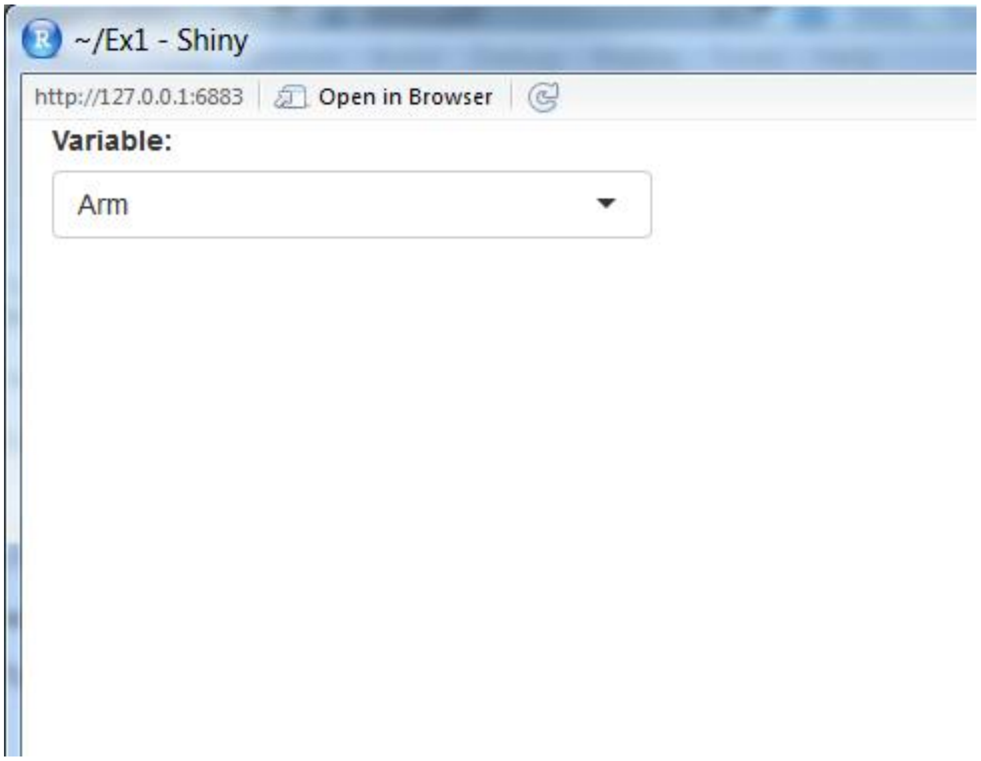
This now creates an attribute of the “input” object called variable that is the option that the user selects between “Arm”, “Country” and “Race”. As you can see below:



Output 3: Slow Progress….

Now we can use the input from the user to create a table

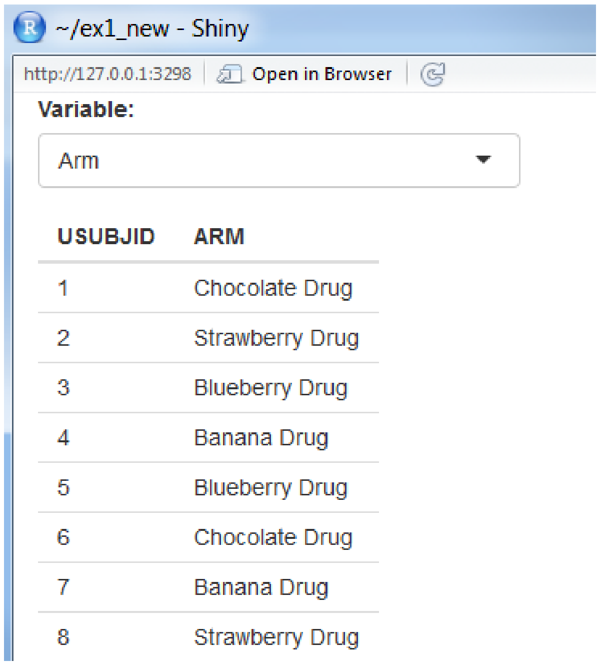
```
library(shiny)
library(haven)

dm <- read_sas("J:/drug/study/R_training/dm.sas7bdat")

ui <- fluidPage(  selectInput("variable", "Variable:",
                            c("Arm" = "ARM",
                              "Country" = "COUNTRY",
                              "Race" = "RACE"))
              )

server <- function(input, output) {
  output$data <- renderTable({
    dm[, c("USUBJID", input$variable), drop = FALSE]
  }, rownames = TRUE)
}

# Run the application
shinyApp(ui = ui, server = server)
```



Output 4: I thought there was supposed to be a table here?

I'm sure you note that there is no table here, that is because after the table is generated on the server side, it has to be rendered on the UI side. So, we have to use the RenderTable statement:

```
library(shiny)
library(haven)
dm <- read_sas("J:/drug/study/R_training/dm.sas7bdat")

ui <- fluidPage(
  selectInput("variable", "Variable:",
              c("Arm" = "ARM",
                "Country" = "COUNTRY",
                "Race" = "RACE")),
  tableOutput("data")
)

server <- function(input, output) {
  output$data <- renderTable({
    dm[, c("USUBJID", input$variable), drop = FALSE]
  }, rownames = TRUE)
}

# Run the application
shinyApp(ui = ui, server = server)
```



Variable: Arm

USUBJID	ARM
1	Chocolate Drug
2	Strawberry Drug
3	Blueberry Drug
4	Banana Drug
5	Blueberry Drug
6	Chocolate Drug
7	Banana Drug
8	Strawberry Drug

Output 5: We did it! An interactive table

Now it’s your turn:

EXERCISE 1: SHINY FUNDAMENTALS

- *Load app.R from \Shiny_Ex1*
- *Run it and explore it*
- *Replace selectInput with RadioButton (all parameters are the same)*
- *Add Age as a variable to output$data*
- *Reminder: the c() function combines arguments into a vector.*

EXTENDING THE FUNCTIONALITY

First, congrats on creating your first project with Shiny. I’d shake your hand if I was there:

Figure 4: This is not my actual hand.

Everything that was just done is the core of Shiny. Everything else is just adding additional inputs and outputs, or nesting it within tabs or dashboards. If you can understand the basic loop we just did, everything else will be a breeze.

As an example, let's replace the table with a histogram. To do this, we'll use 2 different functions: tableOutput to replace plotOutput, and renderTable to replace renderPlot.

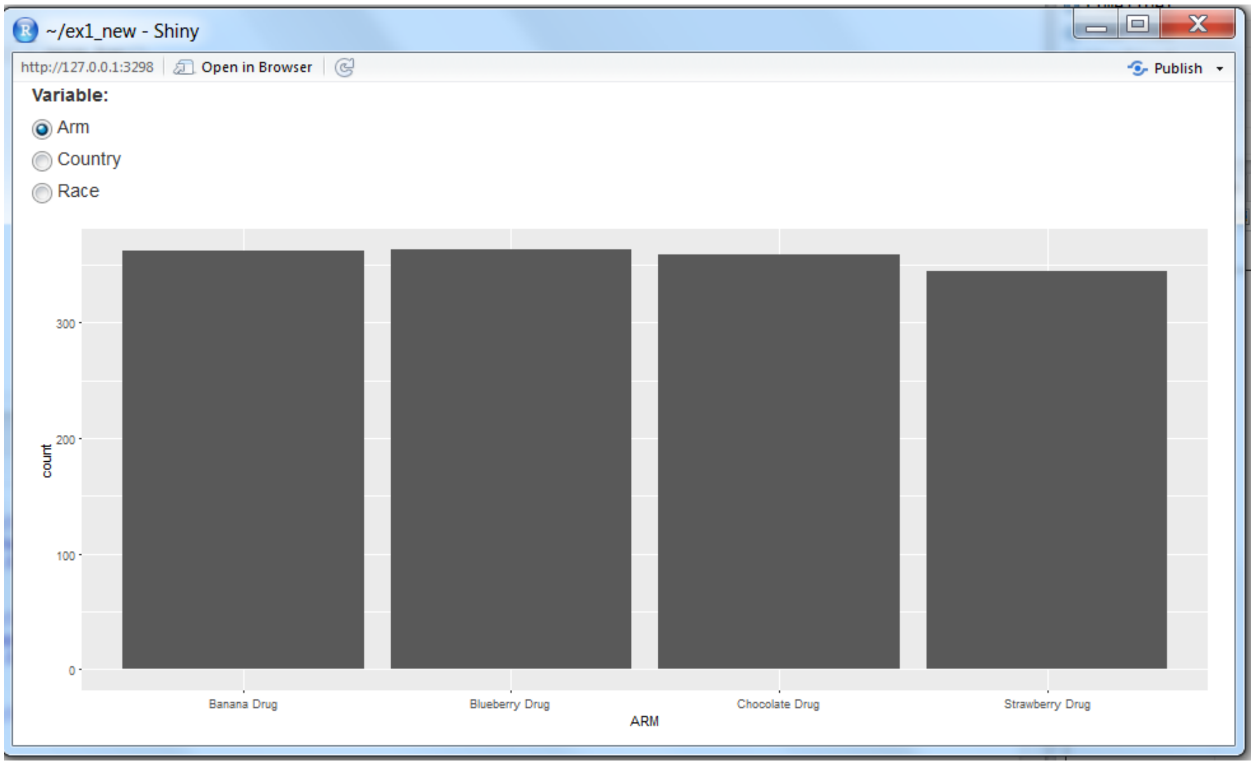
So let's take the code we just generated:

```
library(shiny)
library(haven)
dm <- read_sas("J:/drug/study/R_training/dm.sas7bdat")

ui <- fluidPage(
  selectInput("variable", "Variable:",
              c("Arm" = "ARM",
                "Country" = "COUNTRY",
                "Race" = "RACE")),
  tableOutput("data")
)

server <- function(input, output) {
  output$data <- renderTable({
    dm[, c("USUBJID", input$variable), drop = FALSE]
  }, rownames = TRUE)
}

# Run the application
shinyApp(ui = ui, server = server)
```

and make those replacements:

```
library(shiny)
library(haven)

dm <- read_sas("J:/drug/study/R_training/dm.sas7bdat")

ui <- fluidPage(  radioButtons("variable", "Variable:",
                           c("Arm" = "ARM",
                             "Country" = "COUNTRY",
                             "Race" = "RACE")),
                  plotOutput("plot")
              )

server <- function(input, output) {
  output$plot <- renderPlot({
    ggplot(dm, aes_string(input$variable)) + geom_bar()
  })
}

# Run the application
shinyApp(ui = ui, server = server)
```

You get an interactive plot just like that:



Output 6: Easy as Pie

EXERCISE 2: PLOT TWIST!

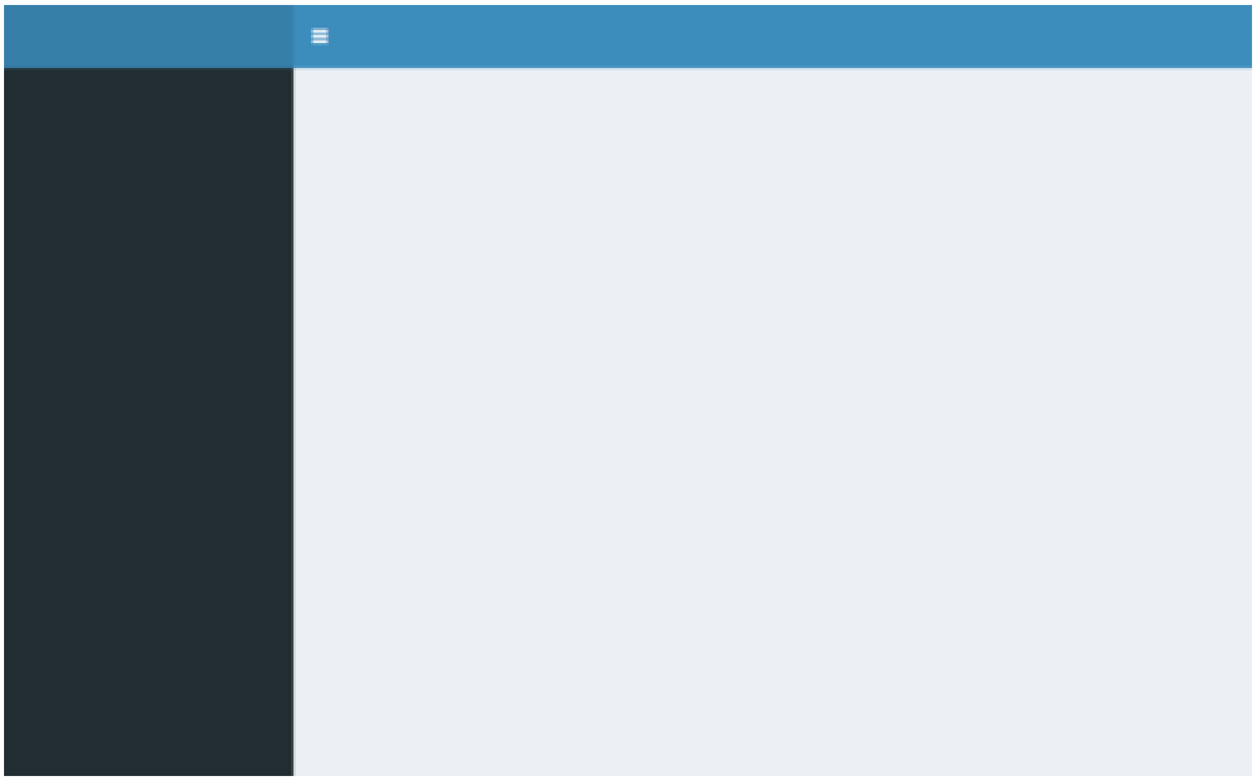
Take your program from exercise 1. Using our example, add a plot that will do a histogram of <variable> by sex (remember we need to use facet)

DASHBOARD CONFESSIONALS

Now that you understand how to create and modify both table and plot objects, let's actually look at a dashboard. For this, we will need the shinydashboard package. Running the following code:

```
library(shiny)
library(shinydashboard)

ui <- dashboardPage(
  dashboardHeader(),
  dashboardSidebar(),
  dashboardBody()
)

server <- function(input, output) { }

shinyApp(ui, server)
```

which should look familiar to you, generates the following:



Output 7: Stephen Few Would Not Be Happy

The only differences are the UI object has another object within in… a dashboard object. This dashboard object has three parts: header (which contains the title), Sidebar (which deals with moving through different "pages" of a dashboard and is beyond the score of this paper), and Body. Body is basically the output that we already have. Consequently, things that we put in ui we now put in DashboardBody. The good part is that the program will now handle aligning the layout, so you can focus on more importantly matters, like deciding what actually to put in there.

EXERCISE 3: PARADISE BY THE DASHBOARD

First, install the shinydashboard package if you have not done so already. Using the shell described above, add the program that you created in Exercise 2 into the body. If you have extra time, try to put the table and plot both in the DashboardBody and see how the program aligns everything.

CONCLUSION

This short hands-on session was but a taste of what Shiny can do. Fully interactive dashboards that can be deployed in the cloud can be prototyped, shown, and iterated on quickly, which can allow the user to focus on the data, and not creating the structure around it. For more examples, please go to shiny.rstudio.com/gallery or www.showmeshiny.com , for many examples with the coe included.

REFERENCES

Few, Stephen. "Common Pitfalls in Dashboard Design." Accessed Feburary 1, 2018. Available at http://www.perceptualedge.com/articles/Whitepapers/Common_Pitfalls.pdf .

Few, Stephen. "Rich Data, Poor Data: Designing Dashboards to Inform" Accessed Feburary 1, 2018. Available at http://www.perceptualedge.com/articles/Whitepapers/Rich_Data_Poor_Data.pdf

CONTACT INFORMATION

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

Nate Mockler
Nate.mockler@biogen.com

Saranya Duraisamy
saranya.duraisamy@biogen.com