phuse·virtual

Single Day Event

R Shiny App for Clinical Data Review & Analysis

Jagadish Katam,
Princeps Technologies Inc.

phuse.global

Disclaimer

The opinions expressed in this presentation and on the following slides are solely those of the presenter and not necessarily those of Princeps Technologies. Princeps Technologies does not guarantee the accuracy or reliability of the information provided herein.

Objectives

- Understand what is Shiny?
- Basic structure of a Shiny App
- App Template
- How does Shiny work?
- A look into the script
- Reactivity
- Example Apps
- Uses of Shiny
- Resources

Shiny

- Shiny is an open source R package that provides an elegant and powerful web framework for building web applications using R. Shiny helps you turn your analyses into interactive web applications without requiring HTML, CSS, or JavaScript knowledge.

- Install the “shiny” package in R.
  - (Tools>Install Packages)
  - install.packages(‘shiny’)

- Load the package into R by
  - library(shiny)

Create your own App

- Create a new project in Rstudio

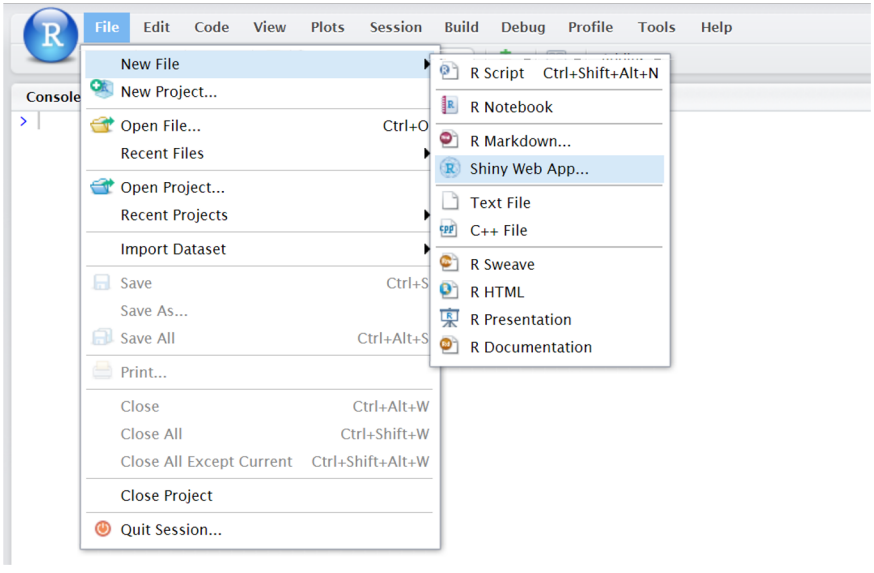
  - New File>New Directory>Shiny Web Application
  - Give it a unique name
  - This will generate folders with the necessary structure

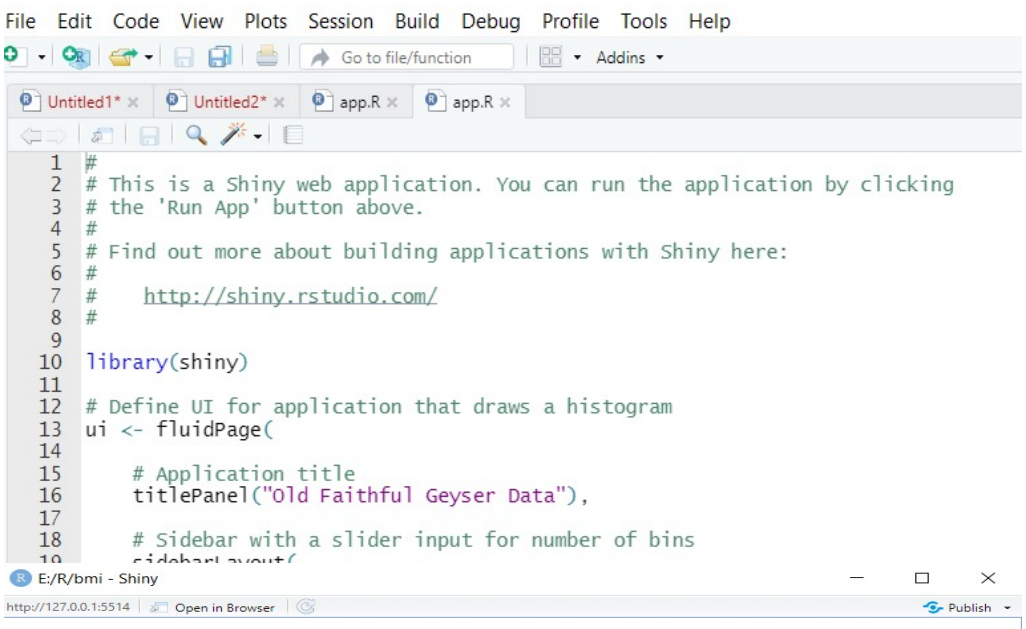
Structure of a Shiny App

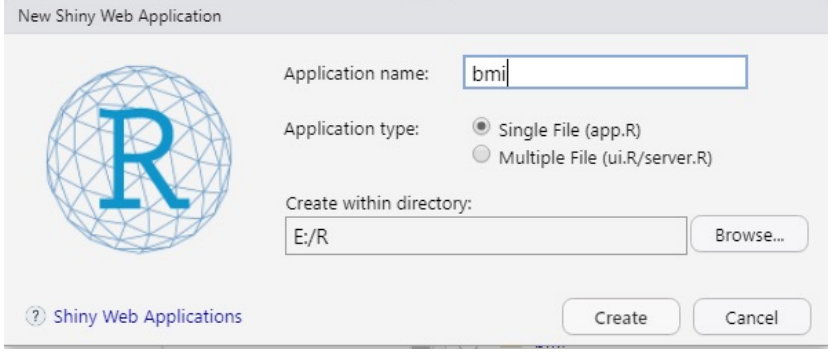
- There are two types of Apps we can create

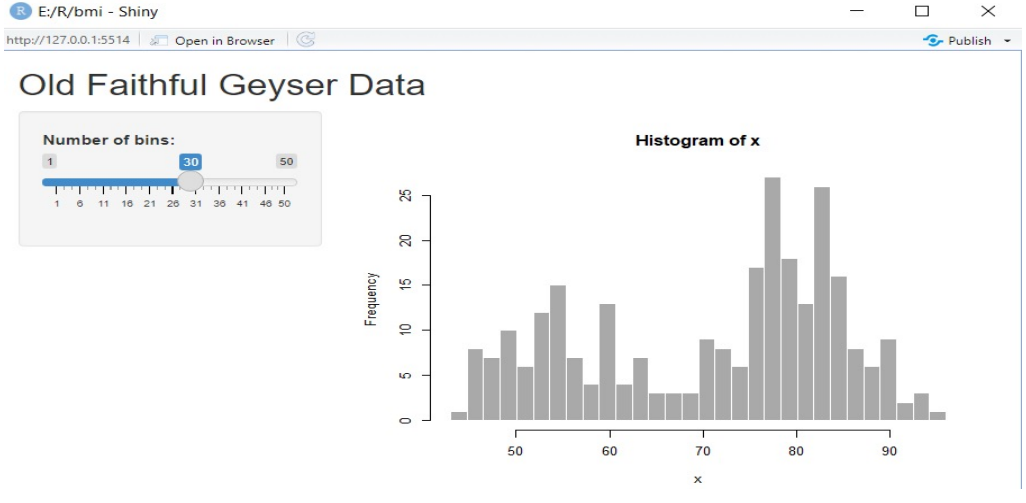
  1. Single file App
     - ui and server are stored in one script
     - used when developing simple Shiny App
     - name of the file has to be app.R
  2. Two file App
     - ui and server are stored in separate scripts
     - Clear separation between the front end and back end
     - Highly preferable when developing advanced Shiny App
     - Names of files have to be ui.R and server.R

- We are going to develop Shiny Apps using the Single file method.


File Edit Code View Plots Session Build Debug Profile Tools Help
New File
New Project...
Open File... Ctrl+O
Recent Files
Open Project...
Recent Projects
Import Dataset
Save Ctrl+S
Save As...
Save All Ctrl+Alt+S
Print...
Close Ctrl+Alt+W
Close All Ctrl+Shift+W
Close All Except Current Ctrl+Shift+Alt+W
Close Project
Quit Session...
R Script Ctrl+Shift+Alt+N
R Notebook
R Markdown...
Shiny Web App...
Text File
C++ File
R Sweave
R HTML
R Presentation
R Documentation


File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Untitled1*
Untitled2*
app.R
app.R
1 #
2 # This is a Shiny web application. You can run the application by clicking
3 # the 'Run App' button above.
4 #
5 # Find out more about building applications with Shiny here:
6 #
7 # http://shiny.rstudio.com/
8 #
9
10 library(shiny)
11
12 # Define UI for application that draws a histogram
13 ui <- fluidPage(
14
15 # Application title
16 titlePanel("Old Faithful Geyser Data"),
17
18 # Sidebar with a slider input for number of bins


New Shiny Web Application
Application name: bmi
Application type:
Single File (app.R)
Multiple File (ui.R/server.R)
Create within directory:
E:/R
Browse...
Shiny Web Applications
Create
Cancel


E:/R/bmi - Shiny
http://127.0.0.1:5514
Open in Browser
Publish
Old Faithful Geyser Data
Number of bins:
1
30
50
1 6 11 16 21 26 31 36 41 46 50
Histogram of x
Frequency
0 5 10 15 20 25
50 60 70 80 90
x

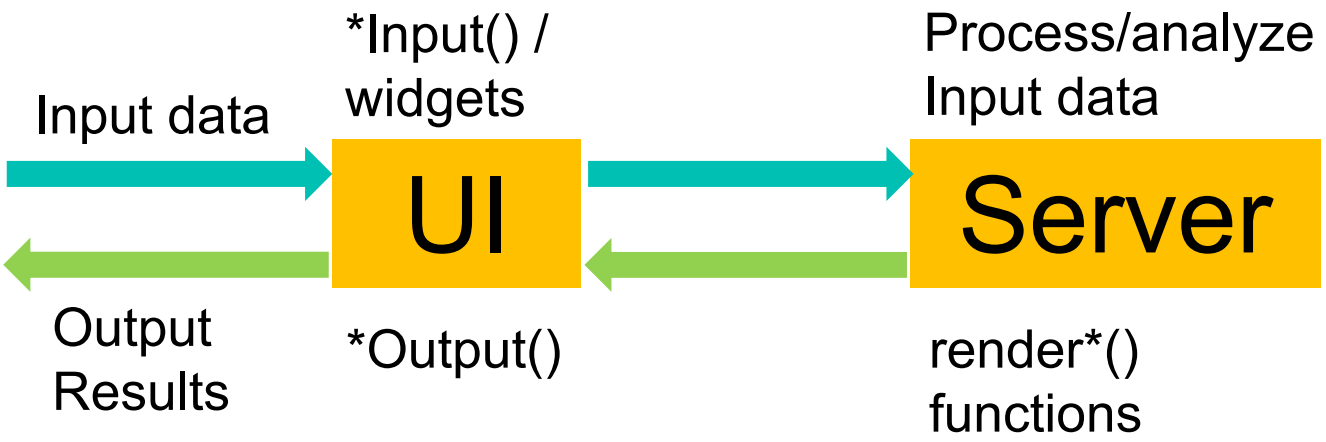
App Template

app.R

Run App

```
1  library(shiny)
2  ui <- fluidPage()
3  server <- function(input, output) {}
4  shinyApp(ui = ui, server = server)
```

Inputs and Outputs



* Name of specific function

Add control widgets via *Input()

- Widgets are web elements that users can interact with. They provide a way for users to send messages to the Shiny app.

- Shiny widgets collect a value from the user. When a user changes the widget, the value will change as well.

*Input() Function	widget
numericInput()	A field to enter numeric values
textInput()	A field to enter text
sliderInput()	A slider bar
selectInput()	A box with choices to select from
radioButtons()	Radio buttons used to select an item from a list.

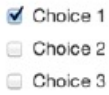
Buttons

`actionButton()`
`submitButton()`

Single checkbox

`checkboxInput()`

Checkbox group



`checkboxGroupInput()`

Date input

`dateInput()`

Date range

`dateRangeInput()`

File input

`fileInput()`

Numeric input

`numericInput()`

Password Input

`passwordInput()`

Radio buttons

`radioButtons()`

Select box

`selectInput()`

Sliders

`sliderInput()`

Text input

`textInput()`

*Output()

- Shiny provides a family of functions that turn R objects into output for the user interface. Each function creates a specific type of output.

*Output() Function	Creates
htmlOutput()	An HTML output element that can be included in a panel
textOutput()	text output element
plotOutput()	plot or image output element
tableOutput()	table output element

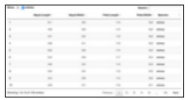
render*()

- Each entry to output should contain the output of one of Shiny's render* functions. These functions capture an R expression and do some light pre-processing on the expression.

- Use the render* function that corresponds to the type of reactive object that you want to output.

render*() Function	Creates
renderText()	Text Output
renderPlot()	Plot Output
renderTable()	Table Output

Outputs - render*() and *Output() functions work together to add R output to the UI



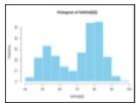
DT::**renderDataTable**(expr, options, callback, escape, env, quoted)

works with

dataTableOutput(outputId, icon, …)

renderImage(expr, env, quoted, deleteFile)

imageOutput(outputId, width, height, click, dblclick, hover, hoverDelay, hoverDelayType, brush, clickId, hoverId, inline)

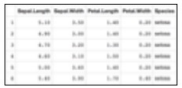


renderPlot(expr, width, height, res, …, env, quoted, func)

plotOutput(outputId, width, height, click, dblclick, hover, hoverDelay, hoverDelayType, brush, clickId, hoverId, inline)

renderPrint(expr, env, quoted, func, width)

verbatimTextOutput(outputId)



renderTable(expr,…, env, quoted, func)

tableOutput(outputId)

foo

renderText(expr, env, quoted, func)

textOutput(outputId, container, inline)

renderUI(expr, env, quoted, func)

uiOutput(outputId, inline, container, …)

& **htmlOutput**(outputId, inline, container, …)

Reactivity

- Shiny uses reactive programming
- Outputs react to changes in input
- All inputs in Shiny are reactive, hence we always use input$<inputid> inside render function which will cause output to re-render.

```
output$bmiout2 <- renderText({
    bmi <-  round(input$weight/(input$height/100)^2,2)
    paste0('Classification: ', bmif(bmi))
})
```

- output$bmiout2 depends on input$weight and input$height
  - If input$weight and input$height changes then output$bmiout2 reacts

Reactivity

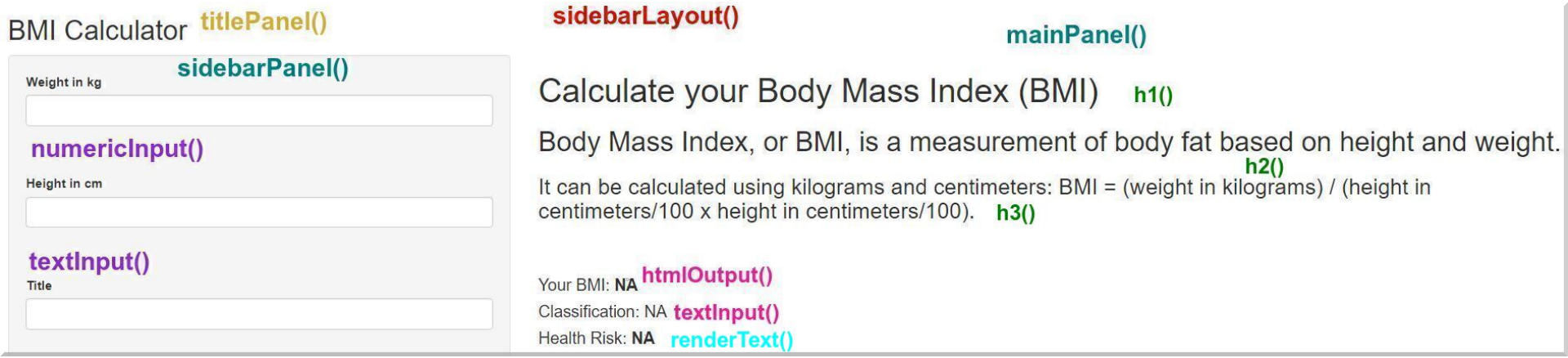
- The reactive values can only be used inside the reactive context like render* or if you want to repeat the reactive expression (same dataset) for multiple render* then we need to create a reactive expression (dataset) using the reactive() function.

```
# generate bmi based on input$weight and input$height from ui.R
bmi     <-  reactive(
    round(input$weight/(input$height/100)^2,2)
    )
```

- Accessing reactive values outside of reactive context results in error.
- The way to use a reactive expression is as below

```
output$bmiout2 <- renderText({
    paste0('Classification: ', bmif(bmi()))
})
```

First App – BMI Calculator



R Shiny Script - BMI Calculator

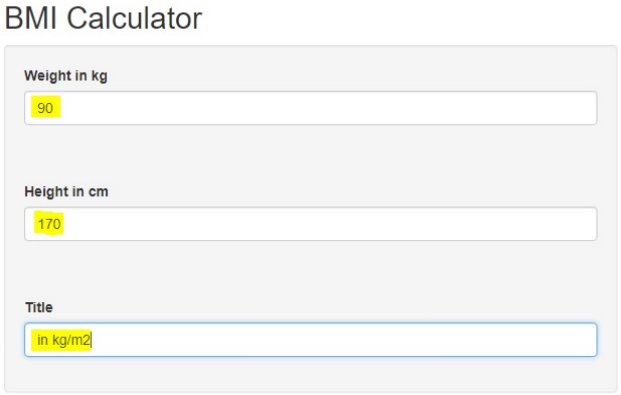
ui

```
# Define UI for application that generates a BMI Calculator
ui <- fluidPage(

    # Application title
    titlePanel("BMI Calculator"),

    # Sidebar with a numeric input and text input
    sidebarLayout(
        sidebarPanel(
            numericInput("weight",
                         "Weight in kg",
                         NULL),

            br(),
            br(),

            numericInput("height",
                        "Height in cm",
                        NULL),

            br(),
            br(),

            textInput('title','Title',value = "")

        ), #sidebarPanel

        # Show the results
        mainPanel(
            h1(htmlOutput('titleout')),
            h2('Body Mass Index, or BMI, is a measurement of body fat based on height and w
            h3('It can be calculated using kilograms and centimeters:  BMI = (weight in kil

            tags$br(),
            tags$br(),

            h4(htmlOutput("bmiout1")),
            h4(textOutput("bmiout2")),
            h4(htmlOutput("bmiout3")),
        ) #mainPanel
    ) #sidebarLayout
) #fluidPage
```

server

```
# Define server logic required to generate a BMI Calculator
server <- function(input, output) {

    # generate reactive expression bmi based on input$weight and input$height from ui.R
    bmi    <-  reactive(
        round(input$weight/(input$height/100)^2,2)
        )


     output$bmiout1 <- renderText({

      paste0('Your BMI: ', '<b>', bmi() ,'<br>')

    })

    output$bmiout2 <- renderText({

        paste0('Classification: ', bmif(bmi()))

    })

    output$bmiout3 <- renderText({

        paste0('Health Risk: ', '<b>', bmih(bmi()) ,'<br>')
    })

    output$titleout <- renderText({

        paste0('Calculate your Body Mass Index (BMI) ',input$title)

    })

}
```

BMI Calculator



Calculate your Body Mass Index (BMI) in kg/m2

Body Mass Index, or BMI, is a measurement of body fat based on height and weight.

It can be calculated using kilograms and centimeters: BMI = (weight in kilograms) / (height in centimeters/100 x height in centimeters/100).

Your BMI: **31.14**
Classification: Obese
Health Risk: **High**

Demo

Second App – Safety Analysis

Safety Analysis **titlePanel()**

sidebarLayout()

mainPanel()

sidebarPanel()

Safety Flag

Y

selectInput()

Treatment Emergent Flag

Y

Severity:

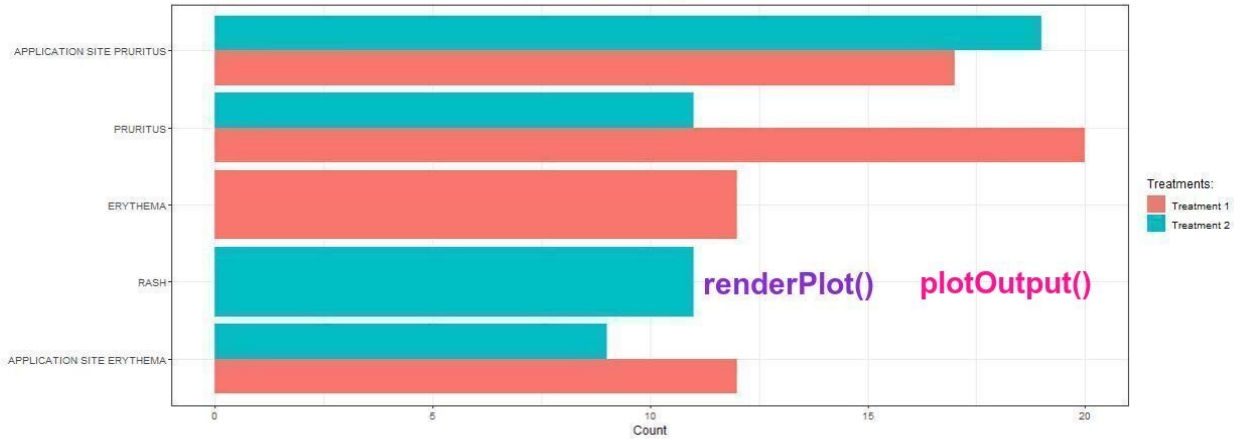
- MILD
- MODERATE
- SEVERE

radioButtons()

Number of Adverse Events to Display:

sliderInput()

0 5 10 15 20 25



renderPlot() **plotOutput()**

AEPT	Treatment 1	Treatment 2
PRURITUS	20	11
APPLICATION SITE PRURITUS	17	19
ERYTHEMA	12	NA
APPLICATION SITE ERYTHEMA	12	9
RASH	NA	11

renderTable() **tableOutput()**

R Shiny Script - Safety Analysis

ui

```
ui <- fluidPage(

    # Application title
    titlePanel("Safety Analysis"),

    # Sidebar with select input
    sidebarLayout(
        sidebarPanel(
            selectInput("saffl",
                        "Safety Flag",
                        choices = c('Y')),

            br(),
            br(),


            selectInput("aetrtem",
                        "Treatment Emergent Flag",
                        choices = c('Y','N')),

            br(),
            br(),

            radioButtons("aesev",
                        "Severity:",
                        choices=unique(factor(adae$aesev)),
                        selected = 'MILD'),

            br(),
            br(),

            sliderInput("aept",
                        "Number of Adverse Events to Display:",
                        min=0,max=25,
                        value = 5,
                        step = 5,
                        ticks = TRUE)
        ),

        # Show a plot of the generated distribution
        mainPanel(
           plotOutput("plot"),
           br(),
           br(),
           # Show a table of the generated distribution
           tableOutput("table")
        )
    )
)
```

server

```
# Define server logic required to get the safety analysis
server <- function(input, output) {

    adsl_adae2 <- reactive(
        inner_join(adsl1,adae,by='usubjid') %>% rename(saffl=saffl.x,trt01a=trt01a.x) %>%
        filter(saffl==input$saffl,aetrtem==input$aetrtem,aesev==input$aesev) %>%
        distinct(usubjid,trt01a,aedecod) %>%
            group_by(trt01a,aedecod) %>%
            summarise(n=n(),.groups = "drop_last") %>%
        arrange(trt01a,n,aedecod) %>%
        mutate(rank=order(trt01a,n,aedecod,decreasing = TRUE),trt01an=ifelse(trt01a=='Treatment 1',1,2)) %>%
            ungroup() %>%
            filter(rank<=(input$aept-1)) %>%
        arrange(rank,trt01an,trt01a,aedecod)

    )

    adsl_adae3 <- reactive(
        inner_join(adsl1,adae,by='usubjid') %>% rename(saffl=saffl.x,trt01a=trt01a.x) %>%
            filter(saffl==input$saffl,aetrtem==input$aetrtem,aesev==input$aesev) %>%
            distinct(usubjid,trt01a,aedecod) %>%
            group_by(trt01a,aedecod) %>%
            summarise(n=n(),.groups = "drop_last") %>%
            arrange(trt01a,n,aedecod) %>%
            mutate(rank=order(trt01a,n,aedecod,decreasing = TRUE)) %>%
            arrange(trt01a,rank,aedecod) %>%
            ungroup() %>%
            filter(rank<=(input$aept-1)) %>%
            pivot_wider(-rank,names_from = trt01a,values_from = n) %>%
            rename(PT=aedecod) %>%
            rowwise() %>%
            mutate(n_sort=max(c_across(starts_with('Treatment')),na.rm = TRUE),AEPT=fct_reorder(PT,n_sort)) %
            arrange(AEPT) %>%
            select(AEPT,starts_with('Treat'))
    )

    output$plot <- renderPlot({

        ggplot(adsl_adae2(),aes(n,reorder(aedecod,n),fill=trt01a,order=trt01an)) +
            geom_col(position = 'dodge' ) +
            xlab("Count") + ylab(" ") +
            theme_bw() +
            guides(fill=guide_legend(title="Treatments:"))

    })

    output$table <- renderTable({

                    adsl_adae3()
    })
```

Uses of Shiny

- It is a great way to
  - Share data analysis results
  - Allow user to explore data
  - Explain statistical concepts
  - Controlled Data visualization
  - It is especially good for repetitive tasks
  - Data cleaning

Resources

- R Studio tutorials are great
  - https://shiny.rstudio.com/tutorial/
- R Studio gallery has lots of small examples to copy
  - https://shiny.rstudio.com/gallery/
- Shiny Apps: Development and Deployment
  - https://www.mzes.uni-mannheim.de/socialsciencedatalab/article/shiny-apps/
- Reactivity 101
  - https://s3.amazonaws.com/assets.datacamp.com/production/course_20680/slides/chapter3.pdf

Thank you

jagadish.katam@princepstechnologies.com

Jagadish Katam | LinkedIn