R Shiny Tutorial

Tuhin Sheikh

Department of Statistics,
University of Connecticut, Storrs, CT.
Email: mdtuhin.sheikh@uconn.edu

April 23, 2020

Outline

1. Basics of RShiny
2. Getting started
3. Building basic shiny app
4. Examples of different input-output types
5. References

What is Shiny?

- Shiny is an open source web application framework for R.
- Shiny makes it easier to present your work to a wider audience.
- Audience with little to no knowledge of R can also explore shiny app with minimal instructions.

Requirements

- Knowledge of R and RStudio.
- Please install the R package: "Shiny" .
- Knowledge of HTML and CSS can be helpful but not required.

Getting started with input-output dynamics

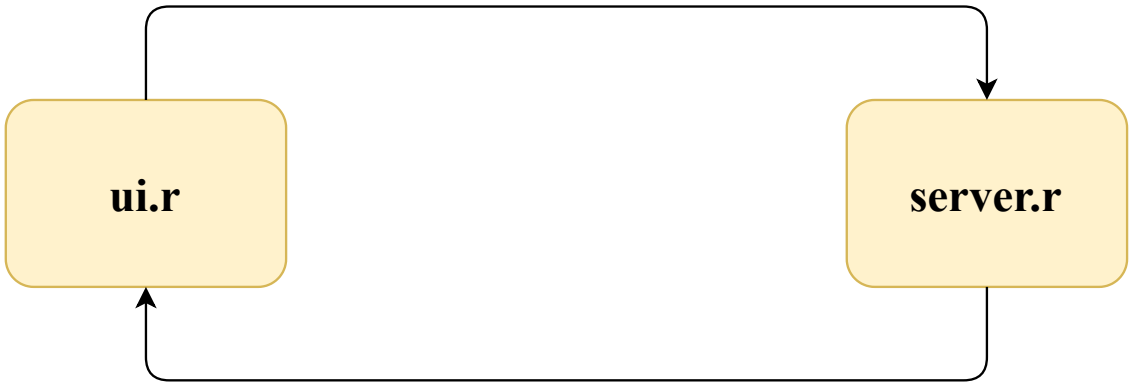

Figure 1: Input-output dynamics in RShiny

More on input-output

ui.r

- Create page layout
- Use different input widgets
- Control type of inputs
- Play with the appearance, etc.

More on input-output cont. . .

server.r

- Give instruction on the inputs
- Control the output types
- Control the display of your outputs, etc.

Example 1: Basic skeleton

```
library(shiny)

ui <- fluidPage(
  titlePanel(title = "This is the title"),
  sidebarLayout(position = "right",
    sidebarPanel(h3("this is a side bar"),
                 h4("widget4"), h5("widget5")),
    mainPanel(h4("this is the main panel text"),
              h5("this is the output"))
  )
)

server <- function(input, output){

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

Basic skeleton cont. . .

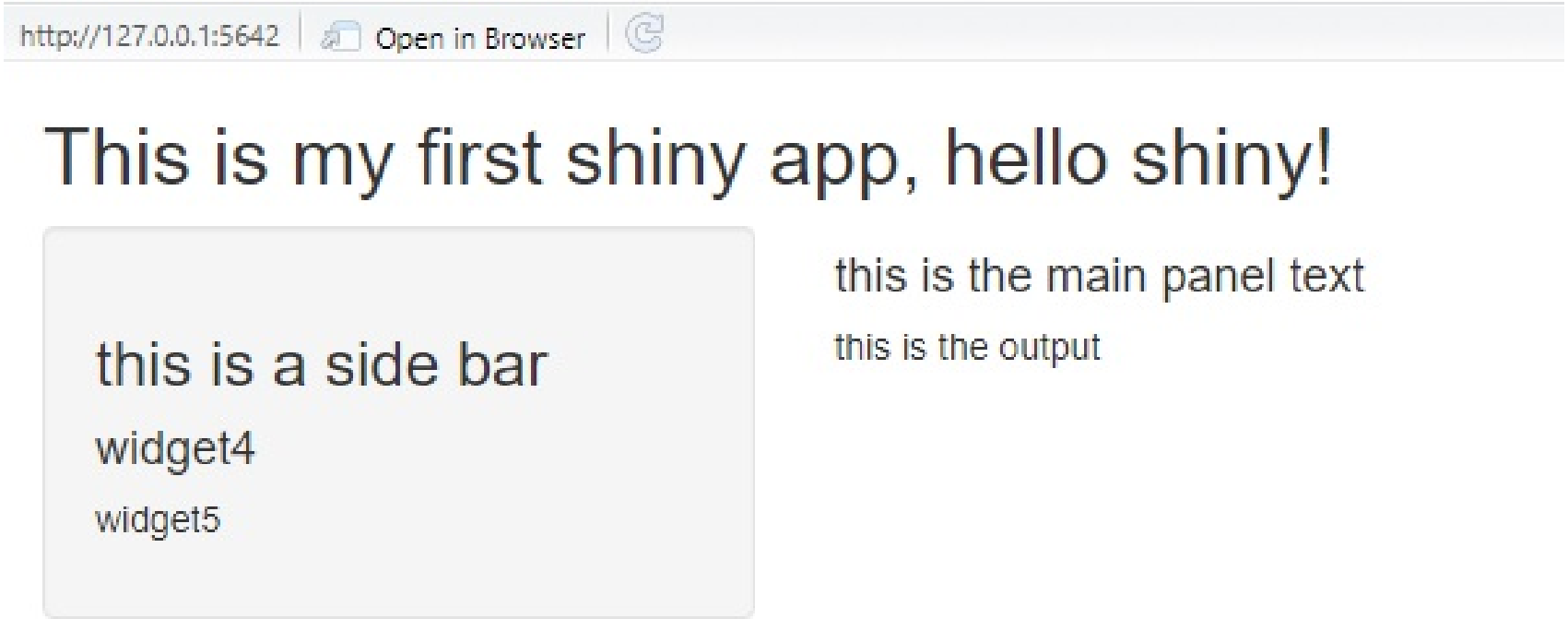

Figure 2: The basic RShiny app

Input and Output in RShiny

- In the 'ui.r', we define all our inputs and output functions.
  - Add inputs with ***Input()** functions.
  - Add outputs with ***Output()** functions.
- The 'server.r' instructs how to produce those outputs with the given inputs.
  - Refer to outputs with **output$<id>**.
  - Refer to inputs with **input$<id>**.
  - Use output type specific **render*()** function to display output.

Example 2: Text Input and Output in RShiny

```
ui <- fluidPage(
  titlePanel(title = "This is the title"),
  sidebarLayout(
    sidebarPanel(
      ...,
      textInput("name", "Type your name")
    ),
    mainPanel(...,
              textOutput("nameout"))
  )
)
server <- function(input, output){
      output$nameout <- renderText({
      paste(input$name)
    })
}
```

Figure 3: Text Input and Output in RShiny

Example 2: Text Input and Output in RShiny

Figure 4: Text Input and Output in RShiny

Example 3: SelectInput()

```
ui <- fluidPage(
        sidebarPanel(
        ...,
          selectInput("country", "Where do you from?",
                      choices = c("Bangladesh", "USA"),
                      selected = NULL)),
          mainPanel(
              paste("My name is"),
              textOutput("nameout"),
              textOutput("country"))
       )
)

server <- function(input, output){
    ...
    output$country <- renderText({
      paste("I am from ", input$country)
    })
}
```

Figure 5: SelectInput() in RShiny

Example 3: SelectInput() cont. . .

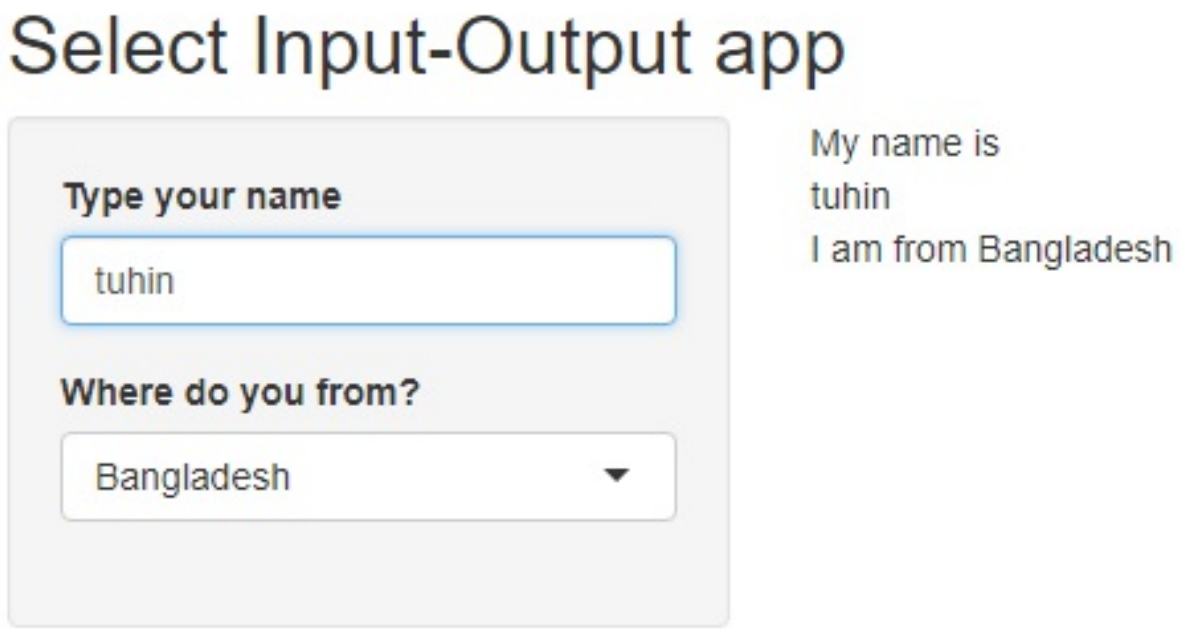

Figure 6: Select Input-output app in RShiny

Plots in RShiny: plotOutput() and renderPlot()

- To display plots on the shiny app, plotOutput() and renderPlot() are used.
- ui.r: defines the id for the plot, e.g. plotOutput("plotID").
- server.r: gives the instruction on the plot construction, e.g. output$plotID <- renderPlot(...).

Example 4: ui.r with plotOutput()

```
ui <- fluidPage(
  sidebarPanel(
    selectInput("var", "Dependent Variable",
                choices = names(mtcars)),
    sliderInput("bins",
                "Number of bins:",
                min = 1,
                max = 50,
                value = 30)
  ),
  mainPanel(
    plotOutput('Hist')
  )

)
```

Figure 7: ui.r of histogram app

Example 4: server.r with renderPlot()

```
server <- function(input, output, session) {
    output$Hist <- renderPlot({
    x <- mtcars[, input$var]
    bins <- seq(min(x), max(x),
                length.out = input$bins + 1)
    hist(mtcars[, input$var],
         breaks = bins,
         col = 'darkgray',
         border = 'white')
  })
}
```

Figure 8: server.r of histogram app

Example 4: histogram shiny app

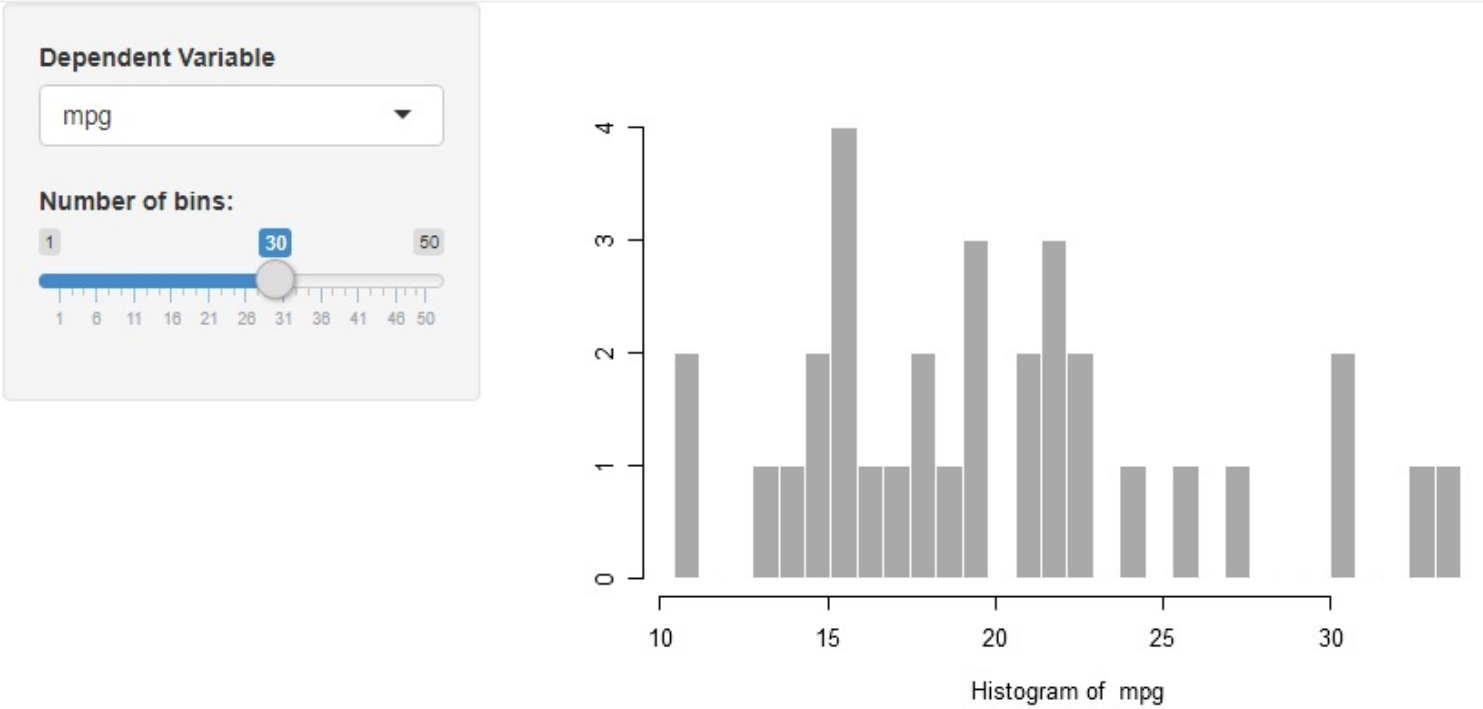

Figure 9: Shiny app of histogram for mtcars data

Useful functions: reactive(), verbatimTextOutput()

- verbatimTextOutput():
  - This function in **ui.r** allows to show the outputs using R functions (e.g. summary(), str(), etc.)
- reactive():
  - In Example 4, x is defined within renderPlot() environment and cannot be used outside this environment.
  - For single task, it is okay. However, what if we need to use this variable information multiple times!
  - The reactive() facilitates to use certain information throughout the app.

Example 5: ui.r with verbatimTextOutput()

```
ui <- fluidPage(
  sidebarPanel(
    selectInput("var", "Dependent Variable",
                choices = names(mtcars)),
    sliderInput("bins",
                "Number of bins:",
                min = 1,
                max = 50,
                value = 30)
  ),
  mainPanel(
    textOutput("text"),
    verbatimTextOutput("summary"),
    plotOutput('Hist')
  )

)
```

Figure 10: ui.r with verbatimTextOutpu() function

Example 5: server.r with reactive()

```
server <- function(input, output, session) {
  xvar <- reactive({
     mtcars[, input$var]
  })

  output$text <- renderText({
    paste("Summary of ", input$var)
  })

  output$summary <- renderPrint({
    summary(xvar())
  })

  output$Hist <- renderPlot({
    bins <- seq(min(xvar()), max(xvar()),
                length.out = input$bins + 1)
    hist(xvar(), breaks = bins, col = 'darkgray',
         border = 'white', main = "", ylab = "",
         xlab = paste("Histogram of ", input$var))
  })
}
```

xvar() can be used throughout the app due to the reactive function

Figure 11: server.r with reactive() function

Example 5: reactive() and verbatimTextOutput() app

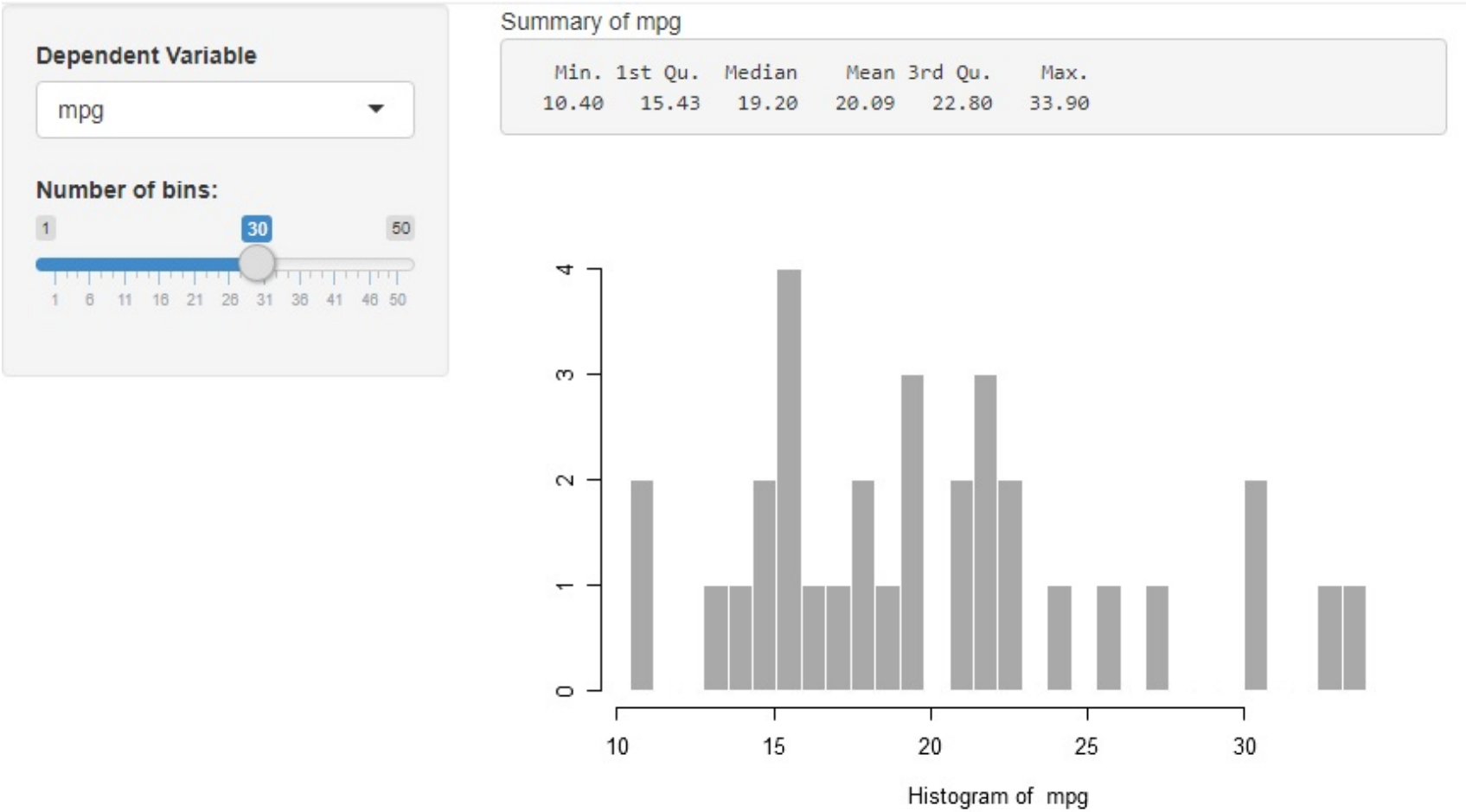

Figure 12: Shiny app using reactive() and verbatimTextOutpu()

Dynamic user interface: uiOutput() and renderUI()

- The earlier examples consider that we have static data and variable names are from that particular data.
- We might want to make the selection of variables that change dynamically with the change in data.
- ui.r: defines uiOutput() as an output that depends on the known input.
- server.r: uses that output as an input thereafter.

[Note: this dynamics can be compared with the pipe function (e.g. %>%) in R.]

Example 6: ui.r with uiOutput()

```
ui <- fluidPage(
  titlePanel(title = "Dynamic user interface"),
  sidebarLayout(
    sidebarPanel(
      selectInput("dataset", "select data",
                  choices = c("iris", "mtcars")),
      uiOutput("vx"),
      sliderInput("bins",
                  "Number of bins:",
                  min = 1,
                  max = 50,
                  value = 30)
    ),
    mainPanel(
       textOutput("dataName"),
       verbatimTextOutput("structure"),
       textOutput("varName"),
       verbatimTextOutput("summary"),
       plotOutput("Hist")
    )
  )
)
```

Figure 13: ui.r with uiOutput()

Example 6: server.r with renderUI()

```
server <- function(input, output) {
  dat <- reactive({
    get(input$dataset)
  })
  ...
  output$vx <- renderUI({
     selectInput("variablex", "Select variable",
                 choices = names(dat()))
  })

  var <- reactive({
    dat()[, input$variablex]
  })
  ...
  output$Hist <- renderPlot({
    bins <- seq(min(var()), max(var()),
                length.out = input$bins + 1)
    hist(var(),
         breaks = bins, col = 'darkgray', border = 'white', ylab = "",
         main = "", xlab = paste("Histogram of ", input$varablex)
  })

}
```

Output based on input data

uiOutput as input

Figure 14: server.r with renderUI()

Example 6: shiny app using renderUI()

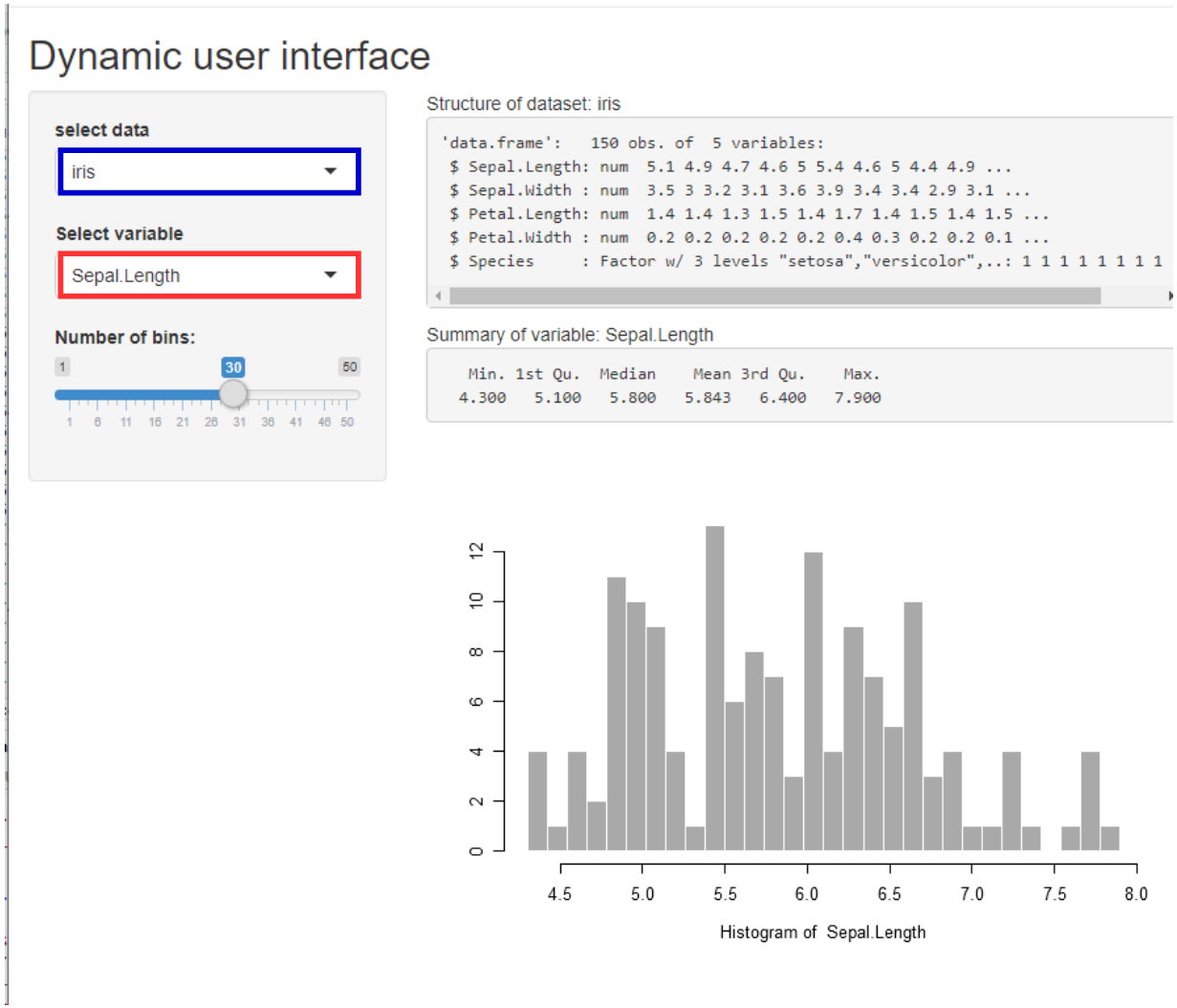

Figure 15: Shiny app with dynamic user interface

Tabs in the Shiny app: tabsetPanel()

- It is often desirable to have tabs on the shiny app to separate the outputs in an organized fashion.
- The main function to be used in ui.r is tabsetPanel()

```
tabsetPanel(type = "tab",
    tabPanel("Summary", verbatimTextOutput("summary")),
    tabPanel("Structure", verbatimTextOutput("str")),
    tabPanel("Data", tableOutput("data")),
    tabPanel("Plot", plotOutput("Hist"))
)
```

- Calling the input id in the server.r remains same as shown earlier.

Example 7: ui.r with tabsetPanel()

```
ui <- fluidPage(
  titlePanel(title = "Shiny app with tabs"),
  sidebarLayout(
    sidebarPanel(
      selectInput("dataset", "select data",
                  choices = c("iris", "mtcars")),
      uiOutput("vx"),
      sliderInput("bins",
                  "Number of bins:",
                  min = 1,
                  max = 50,
                  value = 30),
      radioButtons("color", "select color",
                   choices = c("Green", "Red", "Yellow"),
                   selected = "Green")
    ),
    mainPanel(
      tabsetPanel(type = "tab",
          tabPanel("Summary", verbatimTextOutput("summary")),
          tabPanel("Structure", verbatimTextOutput("str")),
          tabPanel("Data", tableOutput("data")),
          tabPanel("Plot", plotOutput("Hist"))
      )
    )
  )
)
```

Figure 16: ui.r of shiny app using tabsetPanel()

Example 7: server.r with tabsetPanel()

```
server <- function(input, output) {
  dat <- reactive({
    get(input$dataset)
  })
  output$vx <- renderUI({
    selectInput("variablex", "Select variable",
                choices = names(dat()))
  })
  output$str <- renderPrint({
    str(dat())
  })
  output$summary <- renderPrint({
    summary(dat())
  })
  output$data <- renderTable({
    head(dat())
  })
  var <- reactive({
    dat()[, input$variablex]
  })

  output$Hist <- renderPlot({
    bins <- seq(min(var()), max(var()), length.out = input$bins + 1)
    hist(var(), breaks = bins, col = input$color, border = 'white',
         main = "", xlab = paste("Histogram of ", input$variablex),
         ylab="")
  })
}
```

Calling the output ID is same as basic skeleton

Figure 17: server.r of shiny app with tabs

Example 7: shiny app with tabs

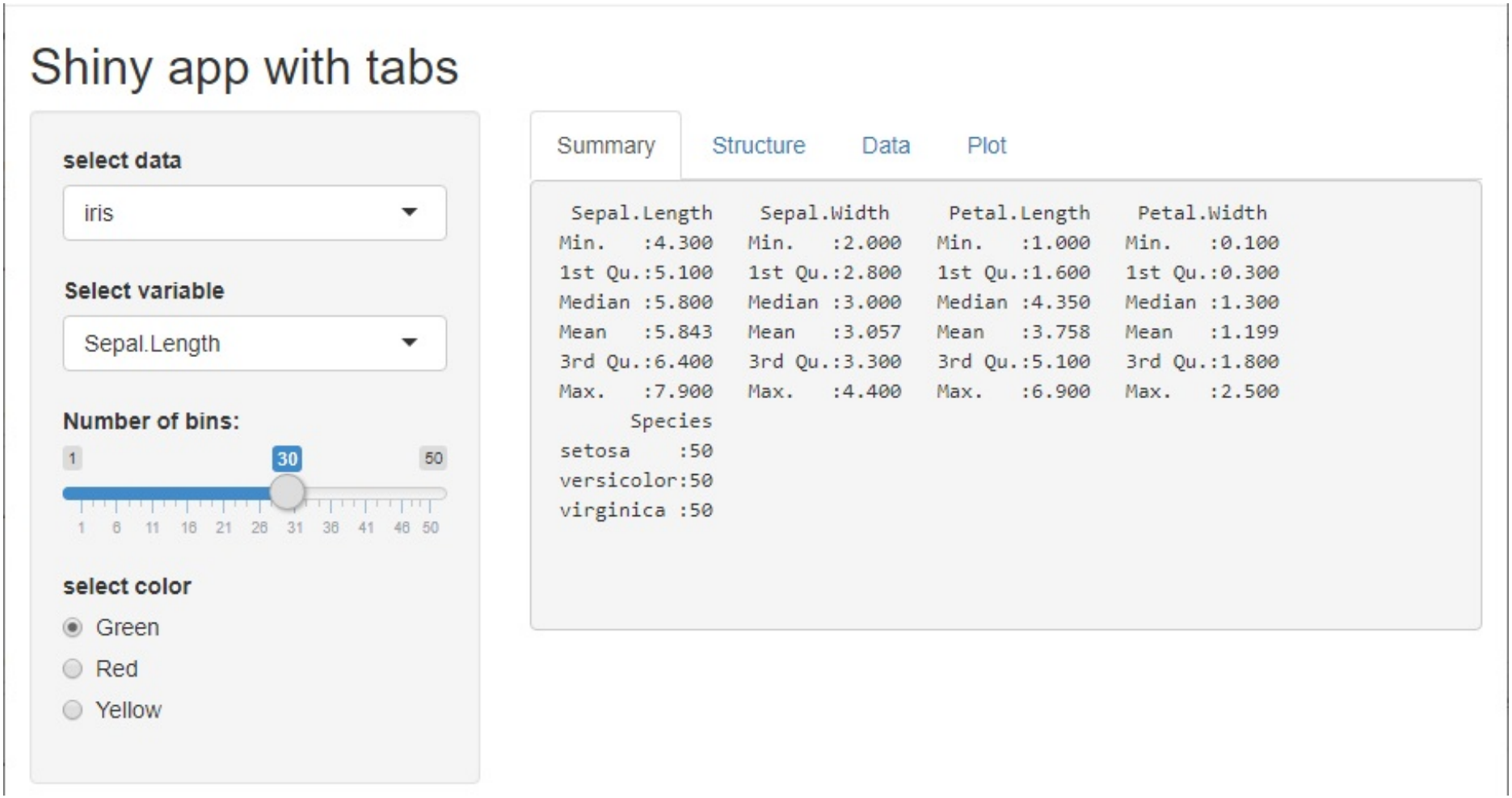

Figure 18: Shiny app using renderUI()

Conditional panels on shiny app

- Having the privilege of conditional inputs makes the shiny app readable and presentable.
- For instance, one might want to have some inputs **only if** certain conditions are met.
- The main function that we need is conditionalPanel().

```
conditionalPanel(condition = "some condition",
  tasks when the condition is met)
```

Example 8: ui.r with conditionalPanel()

```
ui <- fluidPage(
  headerPanel(title = "Conditional panel app"),
  sidebarPanel(
    selectInput("n", "Select Option", choices = c("option1",
                "option2"), selected = "option1"),
    conditionalPanel(
      condition = "input.n == 'option1'",
      selectInput("b", "Select Options within option1",
                  choices = c("A", "B")),
      conditionalPanel(
        condition = "input.b == 'A'",
        titlePanel("Options A")),
      conditionalPanel(
        condition = "input.b == 'B'",
        titlePanel("Options B"))
    ),
    conditionalPanel(
      condition = "input.n == 'option2'",
      selectInput("d", "Select Options within option2",
                  choices = c("C", "D")),
      uiOutput("vx")
    )
  ),
  mainPanel(verbatimTextOutput("text"))
)
```

Figure 19: ui.r of shiny app using conditionalPanel()

Example 8: server.r with conditionalPanel()

```
server <- function(input, output){
  output$text <- renderText({
    if(input$n == 'option1'){
      paste("1st Choice ", input$n, "choice within 1st ",
            input$b, sep = "\n")
    } else {
      output$vx <- renderUI({
        selectInput("test", "Options within second choice",
                    choices = c("choice Opt2 1", "choice Opt2 2"),
                    selected = "test1")
      })
      paste("1st Choice ", input$n, "choice within 1st ", input$d,
            "third choice ", input$test, sep = "\n")
    }
  })
}
```

Figure 20: server.r of shiny app using conditionalPanel()

Example 8: shiny app with conditionalPanel()

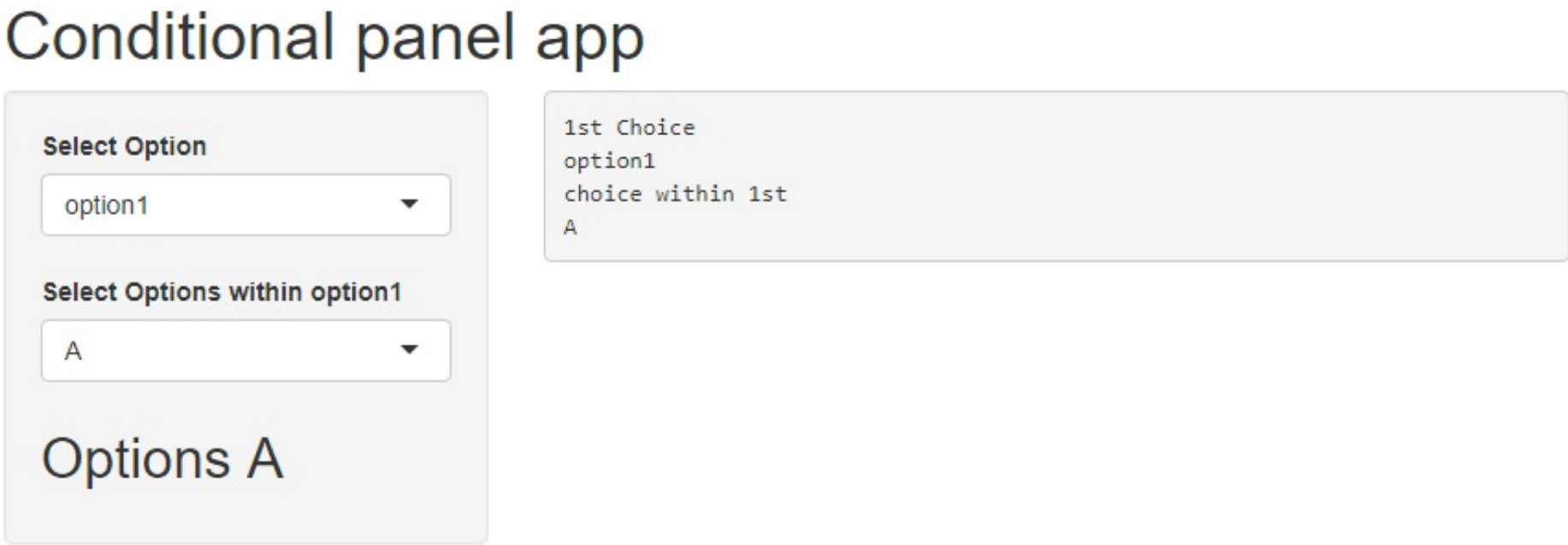

Figure 21: Shiny app with conditionalPanel()

References

1. https://shiny.rstudio.com/tutorial/
2. https://rstudio.github.io/shiny/tutorial/

Thank You