Session 7: An Introduction to Shiny

Dave Margraf

July 26, 2018

Contents

The shiny package

- Build interactive web apps using R.

Look at some published examples.

Download the package if needed:

```
install.packages("shiny")
```

List examples provided in the package with 'runExample()'.

```
## Valid examples are "01_hello", "02_text", "03_reactivity", "04_mpg", "05_sliders", "06_tabsets", "07_
```

Run a local example.

- Open a new R script (Ctrl + Shift + N) and paste the following code:

```
library(shiny)
runExample("05_sliders")
```

Copying and modifying existing apps is a useful way to get used to the structure and functionality of Shiny.

```r
runExample("01_hello")      # a histogram
runExample("02_text")       # tables and data frames
runExample("03_reactivity") # a reactive expression
runExample("04_mpg")        # global variables
runExample("05_sliders")    # slider bars
runExample("06_tabsets")    # tabbed panels
runExample("07_widgets")    # help text and submit buttons
runExample("08_html")       # Shiny app built from HTML
runExample("09_upload")     # file upload wizard
runExample("10_download")   # file download wizard
runExample("11_timer")      # an automated timer
```

Shiny components

- A Shiny app requires a user interface, a server function, and a call to the 'shinyApp()' function.
- The user interface
  - Defines inputs
  - Inputs and outputs laid out
- Server function
  - Creates output and other data
- These may be in separate files or together.

A basic single file template:

```r
library(shiny)

ui <- fluidPage()

server <- function(input, output){}

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

Open the R file named 'shinySingleFile.R' if you would like to build off of it.

User interface

- Use the 'fluidPage()' function to define the layout of your app.
- Section of the page are broken up into panels or rows based on an underlying grid.
- The sidebar and grid layouts are common and easy to begin with.

Sidebar Layout

- Provides a sidebar for inputs and a large main area for output.

```
ui <- fluidPage(

  titlePanel("Sidebar layout example"),

  sidebarLayout(

    sidebarPanel(
      # Add widgets here
    ),

    mainPanel(
      # Add plots here
    )
  )
))
```

Grid layout

- Rows are created by the 'fluidRow()' function.
- Columns defined by the column() function.
- Column widths are 12-wide grid system within a 'fluidRow()'.

```
ui <- fluidPage(

  titlePanel("Grid layout example"),

  fluidRow(
    column(4,
    ),
    column(4,
    ),
    column(4,
    )
  ),

  fluidRow(
    column(2,
    ),
    column(6,
    ),
    column(4,
    )
  )
)
```

Control widgets

- Widgets add web elements to your Shiny app.
- Users can interact with widgets provide values to Shiny.

Standard control widgets:

Function	Widget
'actionButton()'	Action Button
'checkboxGroupInput()'	A group of check boxes
'checkboxInput()'	A single check box
'dateInput()'	A calendar to aid date selection
'dateRangeInput()'	A pair of calendars for selecting a date range
'fileInput()'	A file upload control wizard
'helpText()'	Help text that can be added to an input form
'numericInput()'	A field to enter numbers
'radioButtons()'	A set of radio buttons
'selectInput()'	A box with choices to select from
'sliderInput()'	A slider bar
'submitButton()'	A submit button
'textInput()'	A field to enter text

Open 'widgetGallery.R' to test some of these and press 'Run App'.

Server function

The 'server()' builds a list-like object named output that contains all of the code needed to update the 'R' objects in your app. Each R object needs to have its own entry in the list.

Display reactive output

You can create reactive output by adding an object to your user interface and telling 'R' to build the object in the server function.

```
server <- function(input, output){}
```

Reactivity is acheived connecting the widget values (the source) of 'input' to the objects (the endpoints) in 'output' in the above code.

Output function	Creates
dataTableOutput	DataTable
htmlOutput	raw HTML
imageOutput	image
plotOutput	plot
tableOutput	table
textOutput	text
uiOutput	raw HTML
verbatimTextOutput	text

Add output to the user interface 'sidebarPanel()', 'mainPanel()', or 'column()' in the 'ui' to tell Shiny where to display your object.

Next, provide code to build the object in the 'server()' function.

Render function	Creates
renderDataTable	DataTable

Render function	Creates
renderImage	images (saved as a link to a source file)
renderPlot	plots
renderPrint	any printed output
renderTable	data frame, matrix, other table like structures
renderText	character strings
renderUI	a Shiny tag object or HTML

Create an entry in the server output list by defining a new element within the 'server()' function. The element name should match the name of the reactive element that you created in the 'ui'.

For example, the element 'output$selectOut' is defined by the 'renderPrint()' function in the 'server()' function and matches names with 'verbatimTextOutput("selectOut")' of the 'ui' in the 'widgetGallery.R' file. If you can understand that gibberish you're well on your way to understaning Shiny.

Let's take a look at a simple example:

- Open a new R script (Ctrl + Shift + N) and paste the following code:

```
library(shiny)
runExample("01_hello")
```

The reactive source, 'input$obs', is used by the reactive endpoint, 'output$distPlot'. Whenever 'input$obs' changes, 'output$distPlot' is notified that it needs to re-execute.
