A guide to publishing an R Shiny app online

@huiwen-goy | October 11, 2024

Packages and the basic elements

STEP 1. Install R package ‘shiny’.

Other potentially useful packages:
‘DT’ (more options for formatting tables)
‘shinydashboard’ (more options for UI layout)
‘shinyWidgets’ (more options for buttons and sliders)
‘rsconnect’ (for publishing the app online in shinyapps.io)

STEP 2. Figure out what elements are needed: the general layout of the page, what inputs to take, and what outputs to generate.

STEP 3. Create an empty R script called **app.R** in a folder where you want to store the app’s files, e.g., datasets. It’s best if each app lives in a unique directory. Some people write separate **ui.R** and **server.R** files instead of a single app.R file containing everything; either way is fine.

Each app.R script should contain 3 things (example code on next page):

1. ‘**ui**’, which specifies the layout and appearance, e.g. where elements are located on the page, text content, font size and spacing

```
ui <- fluidPage( … )
```

2. ‘**server**’, which specifies how to build the displayed content, e.g., using numbers entered into the UI to change a plot

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

3. a call to ‘**shinyApp**’ at the end of the script, which runs the app by combining the instructions in ‘ui’ and ‘server’

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

The different elements on a page are called “**panels**”. The ‘**fluidPage**’ function is used to lay out panels in a specified number of rows and columns. "Fluid" means that the heights and widths of the panels self-adjust as the size of the UI window changes. Other functions can be used within the ‘fluidPage’ function, like ‘fluidRow’ or ‘mainPanel’.

Within ‘**fluidPage**’, you can specify what types of input (e.g. numeric, checkbox, slider, drop-down menu) should be in different panels. These inputs can be called later by the ‘**server**’ and used to generate some output, like a calculated variable or a plot. Inside ‘server’, variables can be specified as being “**reactive**”, meaning that the user can keep entering different information into the UI (e.g. adjusting a slider on a scale), and the output will change in response.

Example app.R

```
library(shiny)
library(ggplot2)

x_values <- c(1:5)   #make a fixed set of x-axis values

#prepare additional geoms that can be added by changing options in the UI
add_A_lines_high <- list(
  geom_hline(yintercept = 7, col="red", linetype="dotted"),
  geom_hline(yintercept = 9, col="red", linetype="dashed"),
  geom_text(aes(label = "High", x = 2, y = 8), col = "red", hjust = 0)
  )
add_A_lines_low <- list(
  geom_hline(yintercept = 3, col="blue", linetype="dotted"),
  geom_hline(yintercept = 5, col="blue", linetype="dashed"),
  geom_text(aes(label = "Low", x = 2, y = 4), col = "blue", hjust = 0)
  )

poly_x <- c(3, 4, 4, 3, 2, 2)
poly_y <- c(3, 4, 6, 7, 6, 4)
poly_df <- data.frame(poly_x, poly_y)
add_B_polygon <- list(
  geom_polygon(aes(poly_x, poly_y), data = poly_df, inherit.aes = F, fill="green", alpha=0.5),
  geom_text(aes(label = "Centre", x = 3, y = 5), col = "darkgreen")
  )

############################

ui <- fluidPage(

  #make a sidebarLayout with a plot on left and inputs on right
  sidebarLayout(
      position = c("left"),
      fluid = TRUE,

      #mainPanel in sidebarLayout, with plot
      mainPanel(
        width = 7,
        plotOutput("testplot")
        ), #end mainPanel

      #sidebarPanel in sidebarLayout, with inputs
      sidebarPanel(
        width = 5,
        #first set of discrete reactive inputs for plot (the y-values)
        fluidRow(numericInput(inputId = "x_1", label = "x1", value=NULL, step=1, min=0, max=10)),
        fluidRow(numericInput(inputId = "x_2", label = "x2", value=NULL, step=1, min=0, max=10)),
        fluidRow(numericInput(inputId = "x_3", label = "x3", value=NULL, step=1, min=0, max=10)),
        fluidRow(numericInput(inputId = "x_4", label = "x4", value=NULL, step=1, min=0, max=10)),
        fluidRow(numericInput(inputId = "x_5", label = "x5", value=NULL, step=1, min=0, max=10)),
```

```
        #reactive input A for plot; first option does nothing; other options add 2 geom_lines
        fluidRow(radioButtons(inputId = "add_A",
                              label = p("Option A"),
                              choiceNames = c("None", "Plot low range", "Plot high range"),
                              choiceValues = c("none", "low", "high"),
                              inline = FALSE)),

        #reactive input B for plot; first option does nothing; second option adds 1 geom_polygon
        fluidRow(radioButtons(inputId = "add_B",
                              label = p("Option B"),
                              choiceNames = c("None", "Plot polygon"),
                              choiceValues = c("none", "polygon"),
                              inline = FALSE)),

      ) #end sidebarPanel

  ) #end sidebarLayout

) #end fluidPage


############################

server <- function(input, output){

  #collect inputs into a dataframe
  df_input <- reactive({
                    data.frame(x_values = c(1:5),
                    y_values = c(input$x_1, input$x_2, input$x_3, input$x_4, input$x_5))
              })

  output$testplot <- renderPlot({

    #create basic plot
    p1 <- ggplot() +
      geom_point(data = df_input(), aes(x = x_values, y = y_values)) +
      coord_cartesian(xlim=c(1, 5), ylim=c(0, 10), default=TRUE) +
      theme_bw()

    if(input$add_A == "low"){ p1 <-  p1 + add_A_lines_low }
    if(input$add_A == "high"){ p1 <-  p1 + add_A_lines_high }
    if(input$add_B == "polygon"){ p1 <-  p1 + add_B_polygon }

    p1

  }) #end renderPlot

} #end server


############################

shinyApp(ui, server)     #put the 'ui' and 'server' together into an app
```

Running an R Shiny app on your local computer

Method A. With the app.R script open, click ‘Run’ button in RStudio

Method B. In the console:
```
setwd(‘point/to/the/directory’)
runApp(‘app.R’)
```

While the app is running, the console window will be busy. To stop the app, close the App window or click on the Stop sign.

If you make changes to the code in the app.R file, save those changes, then click ‘Reload App’ to incorporate the changes in the running app.

Publishing your first R Shiny app online

Install package ‘rsconnect’.

Create an account in shinyapps.io. There is a one-time process to authorize a connection between the ShinyApps website and Rstudio on your computer.

On the website, go to Account > Tokens to generate a token.

Then type some commands in the RStudio console (check the instructions in the user guide):
`rsconnect::setAccountInfo(name="…", token="…", secret="…")`

There should be an app.R file (or separate ui.R and server.R files) and any other required files (e.g., csv) stored in some folder on the local computer.

In RStudio, set the working directory to where the app.R and associated files are stored, and type `runApp("filename")` in the console to check that the app works properly.

If the app runs properly, click the Publish button in the top right corner of RStudio.

Check the project name, and project files. Change the project name if needed, and select/unselect any associated files to be published. There should be some progress messages displayed in the RStudio console while the app is being published.

When the app has been published online, a browser window will pop up with the UI and the web address of the new Shiny app, e.g., https://username.shinyapps.io/app_name/.

Publishing subsequent R Shiny apps online

Log into your shinyapps.io account to check that it's still active, and view previous apps there.

In RStudio, set the local working directory to where the app.R and associated files (e.g. csv) are stored, and type `runApp("filename")`. If everything looks good, click "Publish" at the top of the user interface. Change the project name if needed, and select/unselect any associated files to be published. A browser window will pop up with the UI and the web address of the new Shiny app.