Introduction to Shiny
Introduction to Data Science

Joseph N Paulson

paulson.joseph@gene.com

Several slides modified from Garrett Grolemund / Rstudio

What is Shiny?

Shiny is an R package developed by RStudio that allows one to create interactive websites and visualizations.

- Overview:
  - http://shiny.rstudio.com/
- Gallery:
  - http://shiny.rstudio.com/gallery/
- Documentation:
  - http://shiny.rstudio.com/articles/
  - http://shiny.rstudio.com/reference/shiny/
- Deployment:
  - http://shiny.rstudio.com/deploy/
- Cheat sheet:
  - Link here

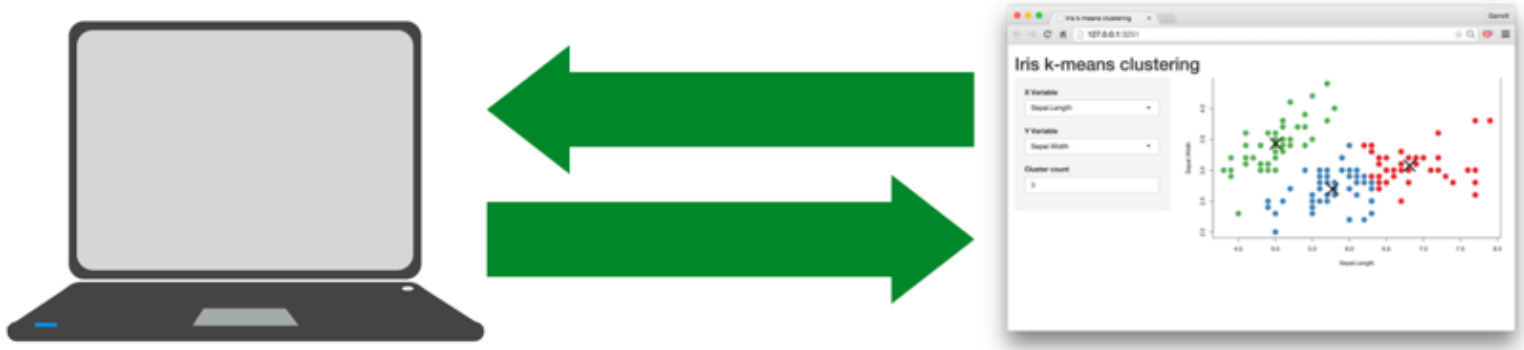
For this presentation, shiny is not a shiny art installation

Examples

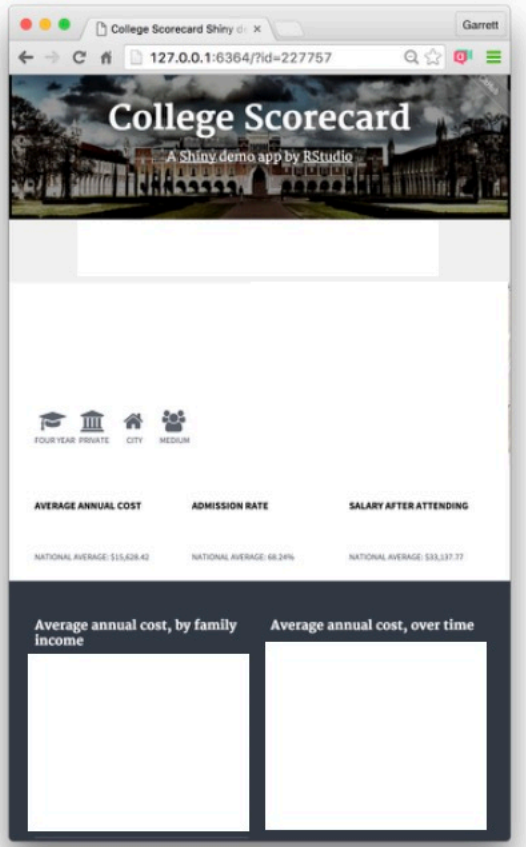
- A few examples we will go over (Lots of examples in the gallery)
  - Simple
    - Malnutrition visualization example – Github Code
    - http://shiny.rstudio.com/gallery/single-file-shiny-app.html
    - http://shiny.rstudio.com/gallery/kmeans-example.html
  - Slightly more advanced
    - Diarrheal microbiome cohort – Github Code
    - http://shiny.rstudio.com/gallery/movie-explorer.html

Shiny overview

Every Shiny app is maintained by a computer (laptop, desktop or server) running R



Slides modified / borrowed from Garrett Grolemund



Server computer running R and Shiny

A Web browser

Template

- Lets create an R app through RStudio.
- It’ll come with a ton of stuff.
- Lets remove it and replace with a minimalist product that will produce an empty (visually) webpage.

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

Just Do It.

The following can be put within a single file, eg. **app.R** the ui and server portions can be separated into two files:

ui.R and **server.R**

and run locally with the **runApp** function.

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

Assessment – run this

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

Overview - 1

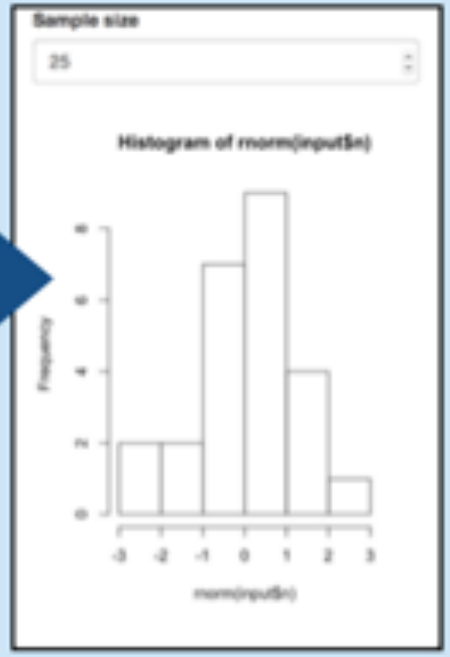
Add inputs to the UI with ***Input()** functions

Add outputs with ***Output()** functions

Tell server how to render outputs with R in the server function. To do this:

1. Refer to outputs with **output$<id>**
2. Refer to inputs with **input$<id>**
3. Wrap code in a **render*()** function before saving to output

```
library(shiny)

ui <- fluidPage(
  numericInput(inputId = "n",
    "Sample size", value = 25),
  plotOutput(outputId = "hist")
)

server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$n))
  })
}

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



Overview - 2

Save your template as **app.R**. Alternatively, split your template into two files named **ui.R** and **server.R**.

```
library(shiny)

ui <- fluidPage(
  numericInput(inputId = "n",
    "Sample size", value = 25),
  plotOutput(outputId = "hist")
)

server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$n))
  })
}

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

```
# ui.R

fluidPage(
  numericInput(inputId = "n",
    "Sample size", value = 25),
  plotOutput(outputId = "hist")
)
```

```
# server.R

function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$n))
  })
}
```

ui.R contains everything you would save to ui.

server.R ends with the function you would save to server.

No need to call **shinyApp()**.

Save each app as a directory that contains an **app.R** file (or a **server.R** file and a **ui.R** file) plus optional extra files.

The directory name is the name of the app

(optional) defines objects available to both ui.R and server.R

(optional) used in showcase mode

(optional) data, scripts, etc.

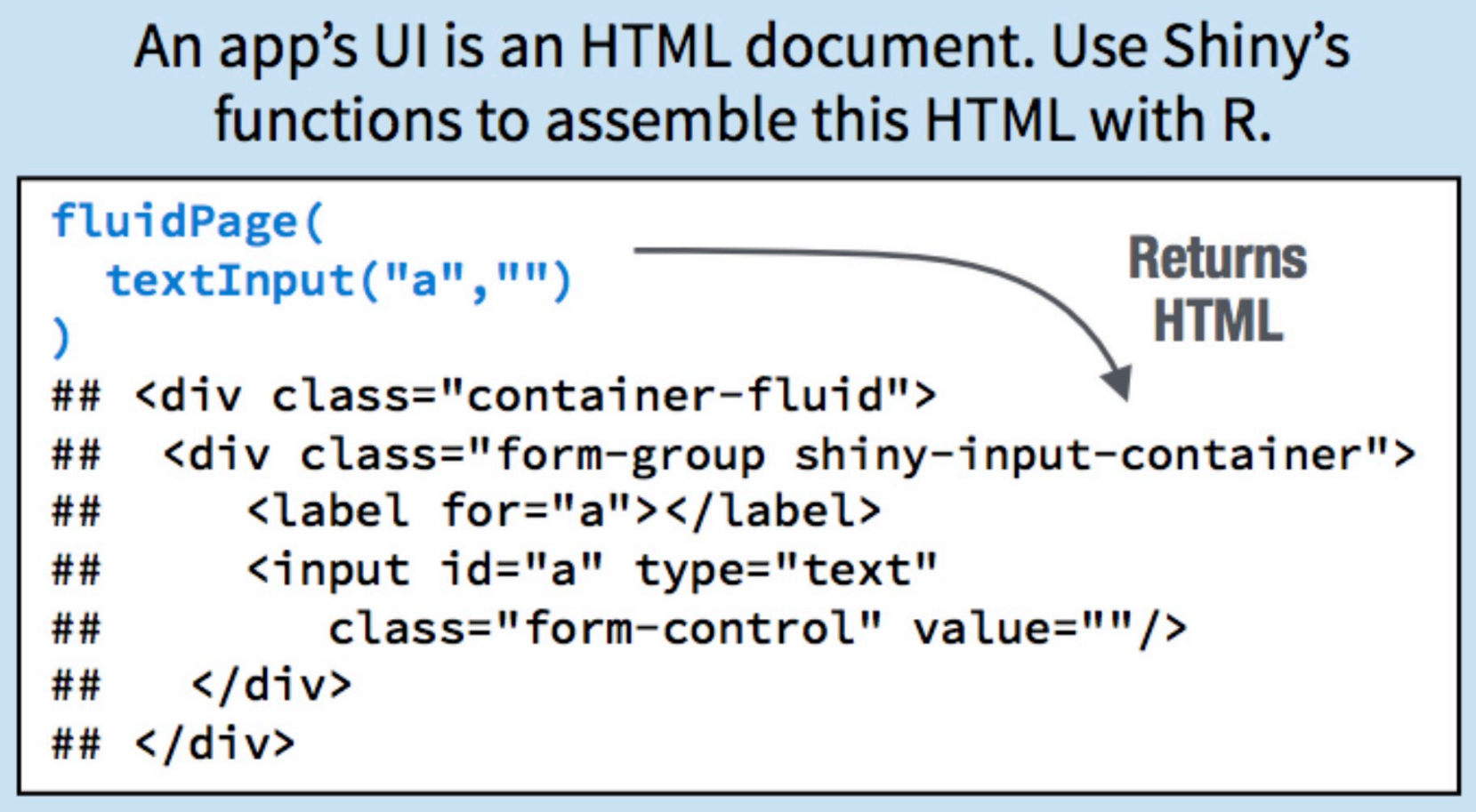
(optional) directory of files to share with web browsers (images, CSS, .js, etc.) Must be named "**www**"

Launch apps with `runApp(<path to directory>)`

UI Breakdown

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

Digging a bit deeper.



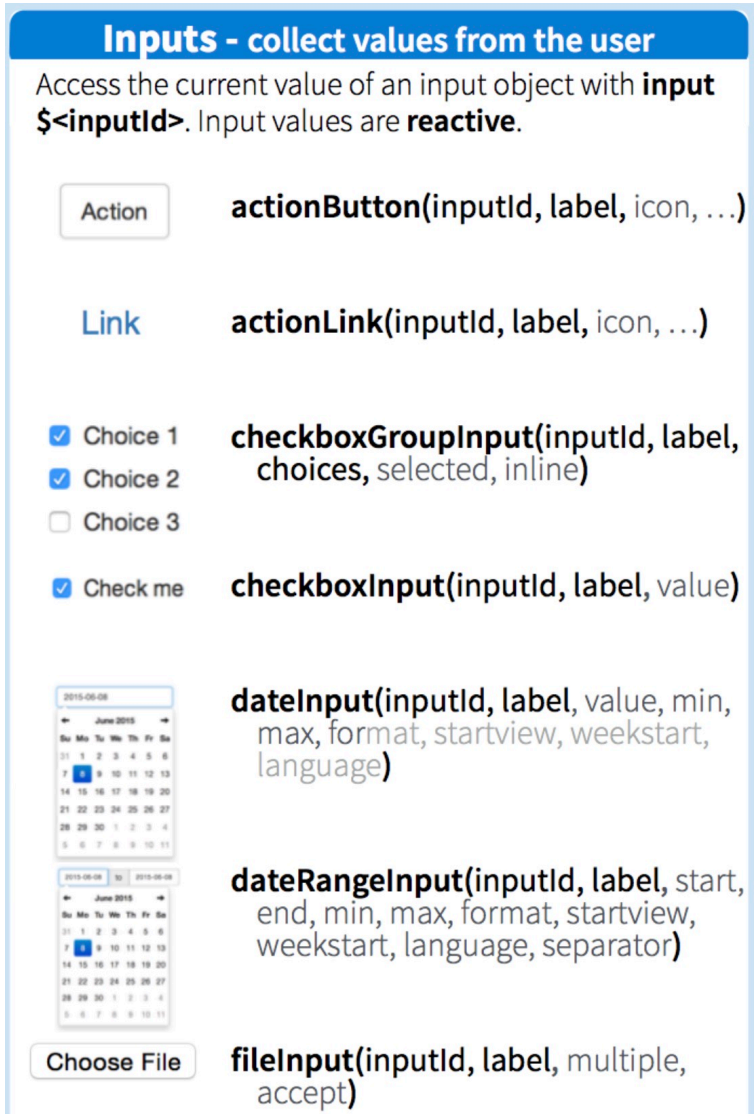
UI - Inputs

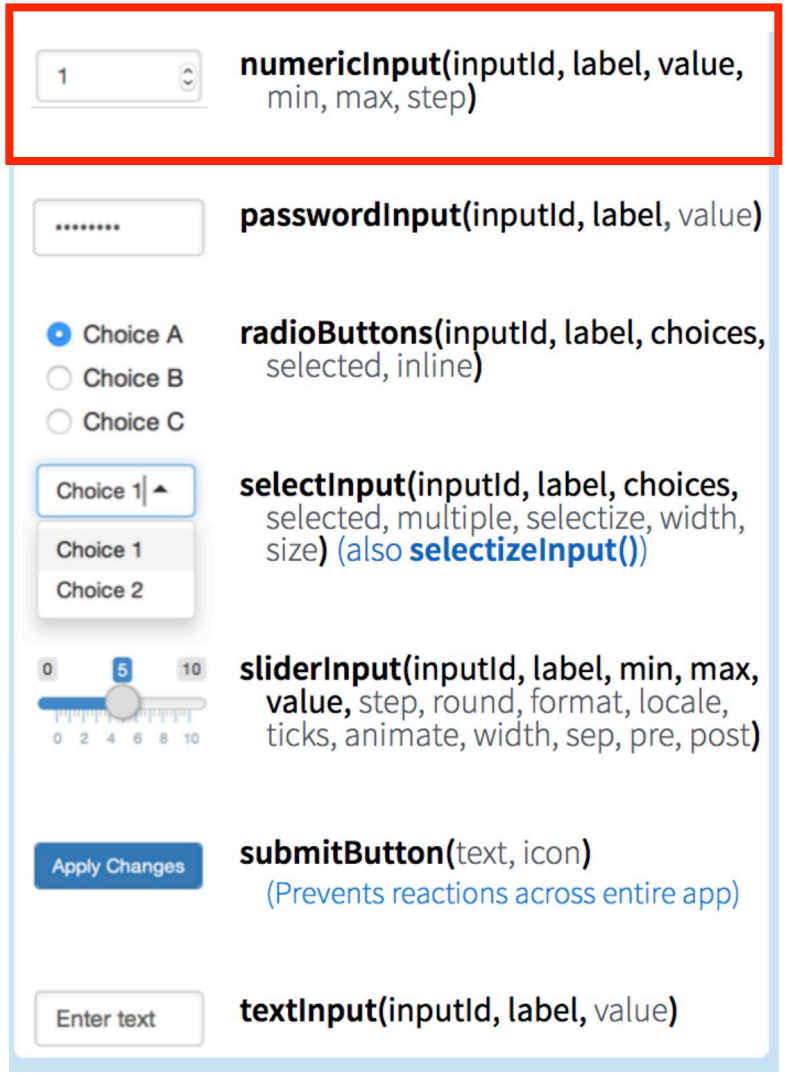
Inputs for the shiny app go within the ui component of the shiny app.

They create the sliders, radio buttons, and clickable values for the HTML user interface.

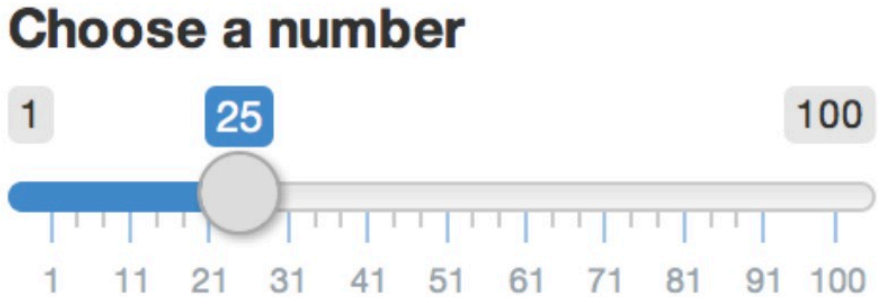
The values from the webpage (UI) are transferred to R in a list called 'input'.

The values can be extracted and processed through R and are available as **input$<inputID>.**





Syntax




```
sliderInput(inputId = "num", label = "Choose a number", …)
```

input name
(for internal use)

Notice:
Id not ID

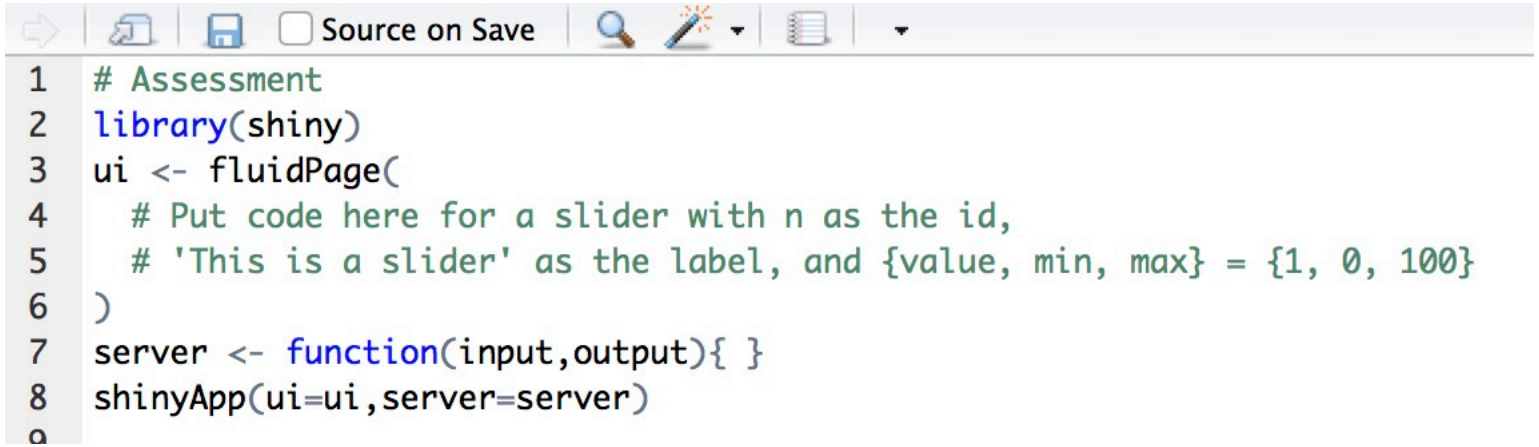
label to display

input specific arguments

```
?sliderInput
```

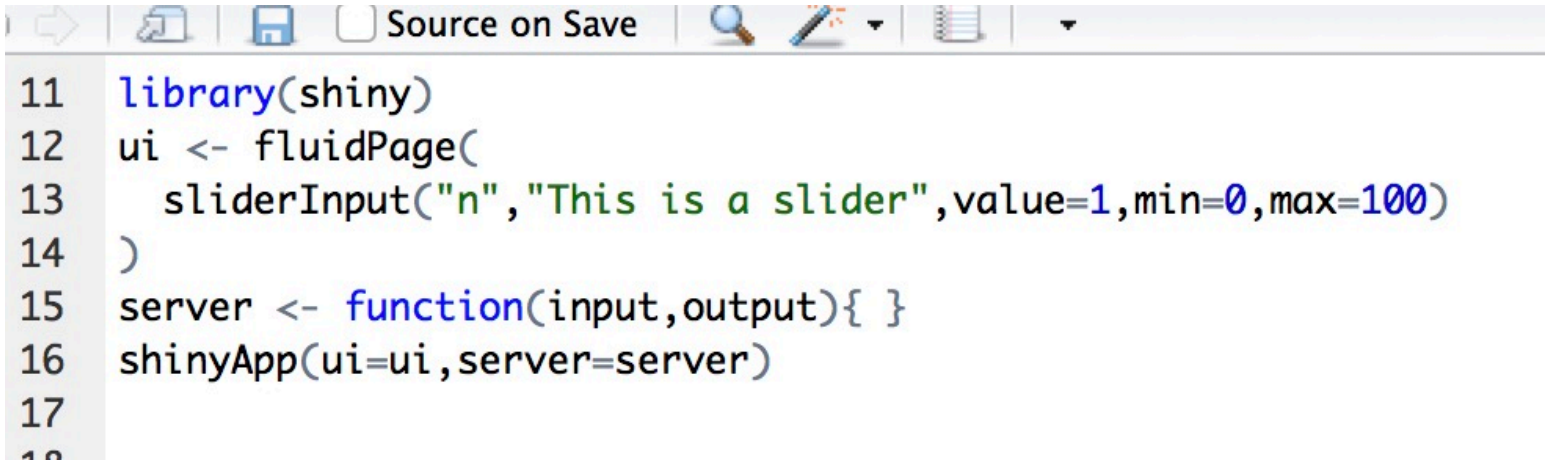
Assessment – UI inputs

Extend the basic template to create a **sliderInput** with a starting value of 1, min of 0 and max of 100.



Solution

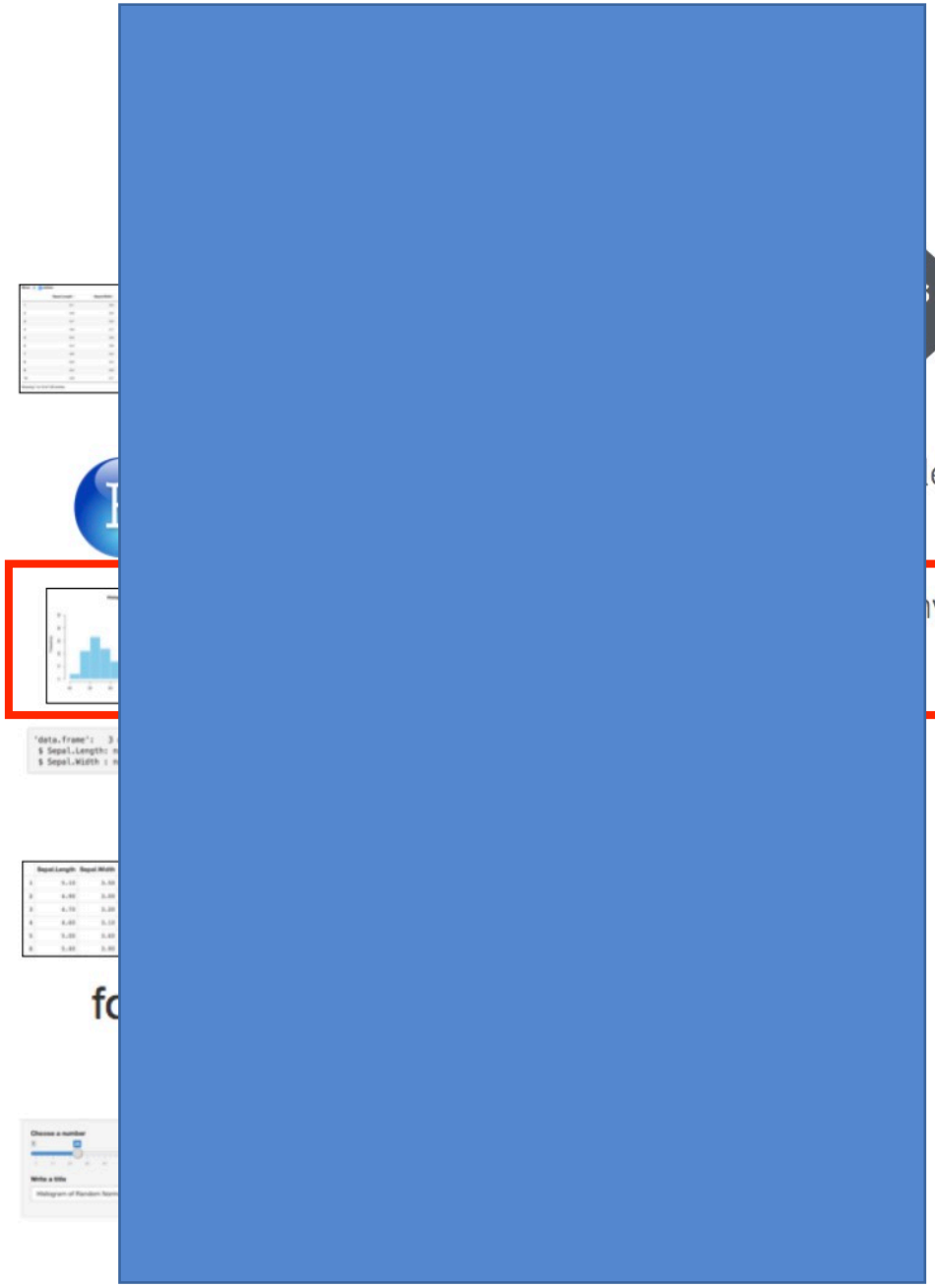
Extend the basic template to create a **sliderInput.**



UI – Output

Other pieces go within the UI specifically to display the 'server' processed information.

The ***Output** functions take the output of functions rendered within the server. The rendered values are transferred from the server in **output$<outputID>** and rendered in the webpage (UI) using the ***Output** functions.



Put into **ui** function

dataTableOutput(outputId, icon, …)

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

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

verbatimTextOutput(outputId)

tableOutput(outputId)

textOutput(outputId, container, inline)

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

*Output()

To display output, add it to `fluidPage()` with an `*Output()` function

```
plotOutput(outputId = "plot")
```

the type of output to display

name to give to the output object

Assessment - UI output

Create a Shiny App using the following template to create a histogram of a user-defined number of draws from a normal distribution (or any distribution).

The general structure for the ui and server are in place as is the code required for plotting the histogram and drawing the 'n' draws.

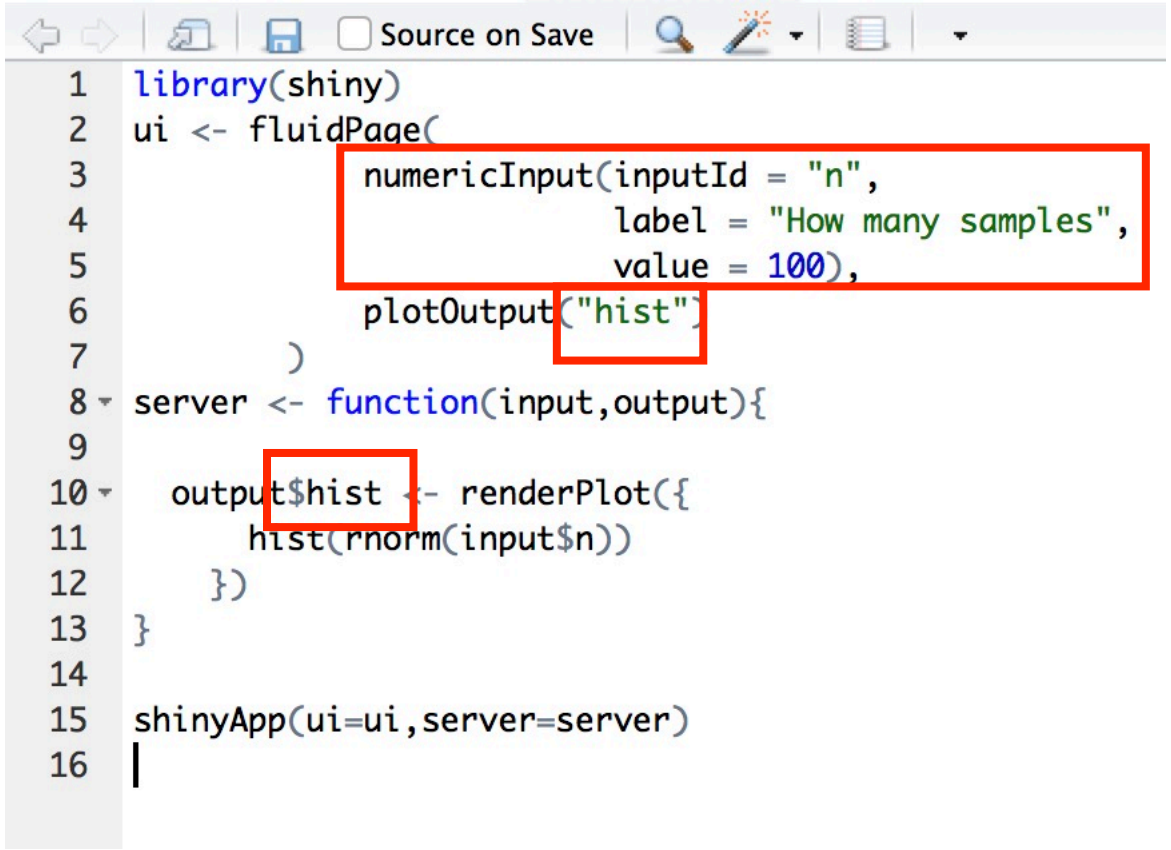
In particular, you only need to prepare the plot. We only need to create a **numericInput** that will say how many draws to make from a distribution and a **plotOutput** that will plot the **hist**ogram.

Source on Save

```
1
2  # Assessment 1
3  library(shiny)
4  ui <- fluidPage(
5    # Place code here
6    # hint you need a 'numericInput' with labelId = n
7    # hint you also need to 'plot' the 'Output'
8  )
9  server <- function(input,output){
10
11     output$hist <- renderPlot({
12       hist(rnorm(input$n))
13     })
14
15 }
16 shinyApp(ui=ui,server=server)
```

Solution

We added a **numericInput** to say how many draws to make from a distribution (eg. **rnorm**) and provided the **hist** function within **output$hist** the **input$n** normal distribution draws.



Server Breakdown

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

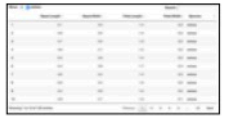
Server

These pieces go within the **server** function and are the workhorse between taking the hard written functions and display.

The render functions perform the opposite task of the input functions and grab the output values and converts it into HTML code.

The rendered values are transferred from the server in **output$<outputID>** and rendered in the webpage (UI).

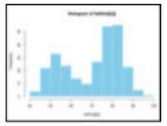
Put into **server** portion



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

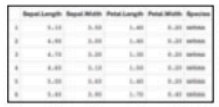
works with

renderImage(**expr**, env, quoted, deleteFile)



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

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



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

foo

renderText(**expr**, env, quoted, func)

renderUI(**expr**, env, quoted, func)

render*()

Builds reactive output to display in UI

```
renderPlot({ hist(iris$Sepal.Length) })
```

type of object to build

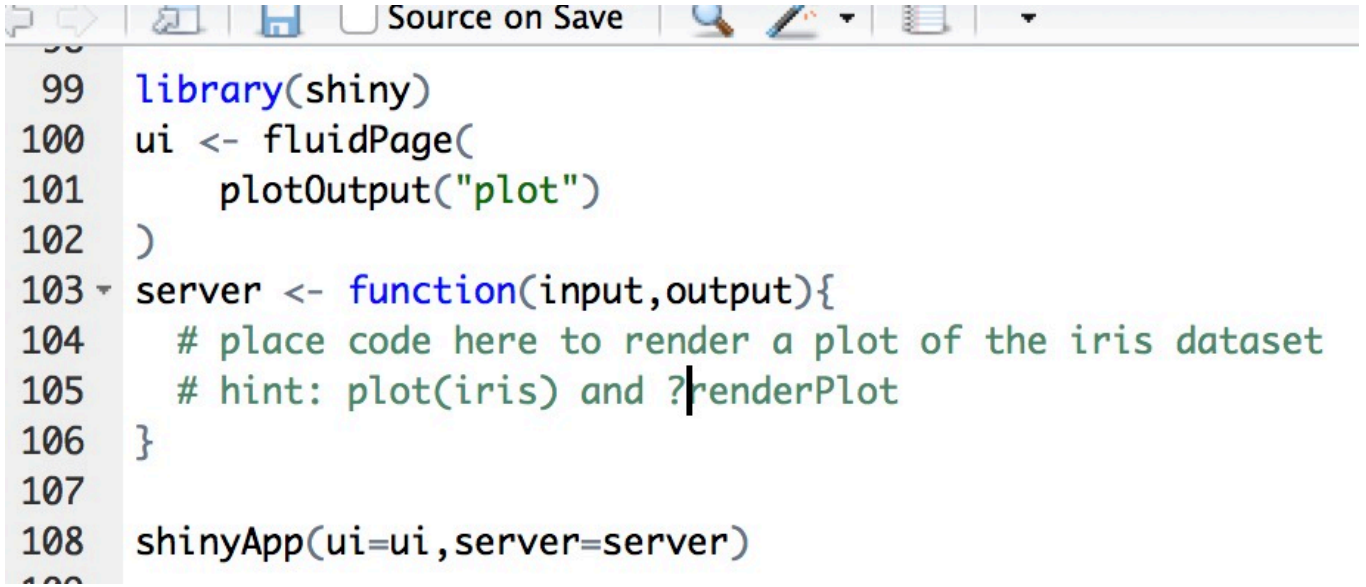
code that builds the object

Assessment - Render output

Create a Shiny App using the following template to plot the iris dataset given:

The general structure for the ui and server are in place as is the code to plot the output from the server.

In particular, you only need to prepare and **renderPlot** the plot.



Solution

Create a Shiny App using the following template to plot the iris dataset given:

The general structure for the ui and server are in place as is the code to plot the output from the server.

In particular, you only need to prepare and **renderPlot** the plot.

```
110  library(shiny)
111  ui <- fluidPage(
112    plotOutput("plot")
113  )
114  server <- function(input,output){
115
116    output$plot <- renderPlot({
117      plot(iris)
118    })
119  }
120
121  shinyApp(ui=ui,server=server)
```

Reactivity

When the sliders, text, numeric, really any of the inputs are modified the functions within the server *react* to the changed input and re-render

Reactivity 101

Reactivity automatically occurs whenever you use an input value to render an output object

```
function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
})
```

Reactions

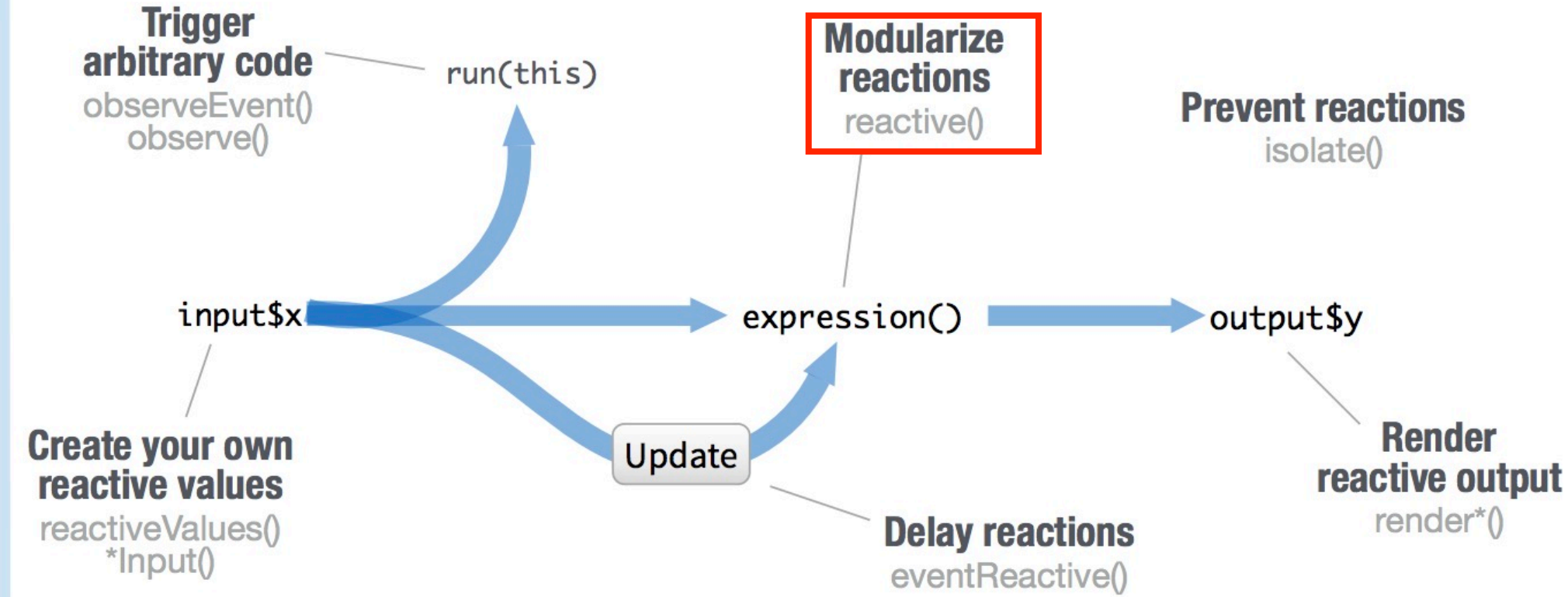
```
library(shiny)
ui <- fluidPage(
  sliderInput("num", "", 1, 100, 25),
  plotOutput("hist")
)
server <- function(input, output) {
  output$hist <- renderPlot({
     hist(rnorm(input$num))
  })
}
shinyApp(ui = ui, server = server)
```

Reactivity automatically occurs whenever you use an input value to render an output object

An input value

Reactivity

Reactive values work together with reactive functions. Call a reactive value from within the arguments of one of these functions to avoid the error `Operation not allowed without an active reactive context.`



Reactivity

- As your apps become more complicated, it can be useful to simplify your code and create **reactive** functions within the server to use across the multiple functions within the server.

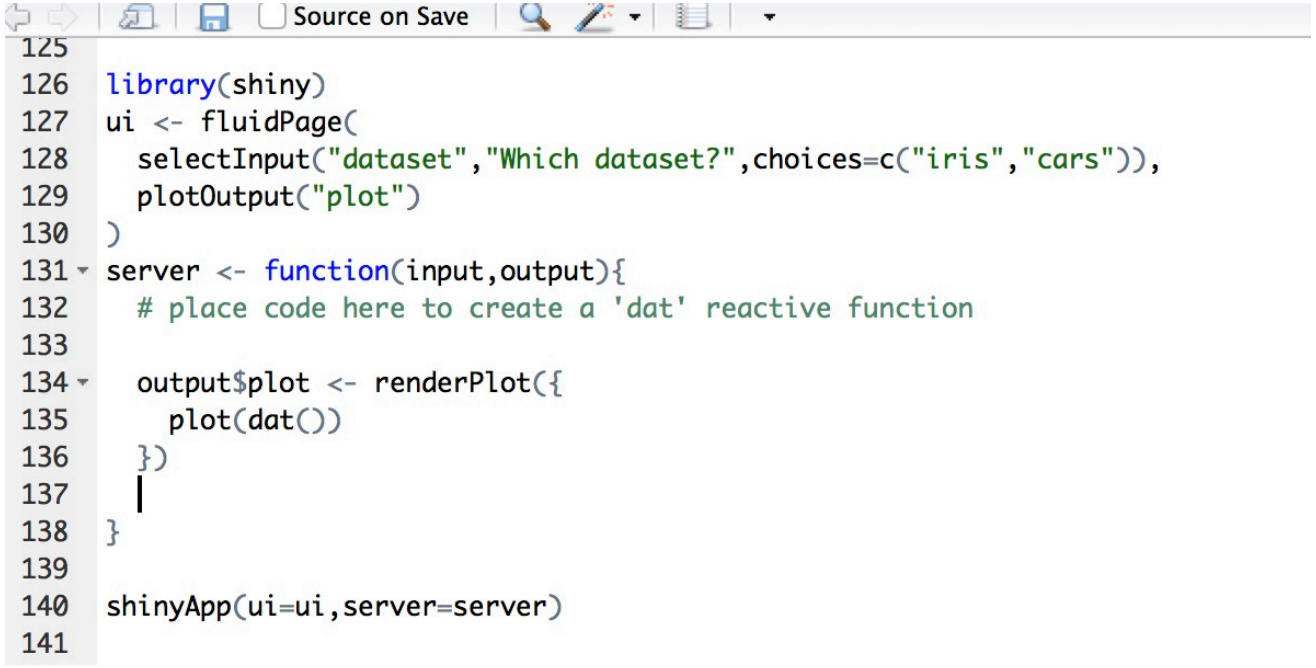
- For example, we can create **reactive** data to use within multiple functions

Assessment - Reactivity

Create a Shiny App using the following template to plot the iris dataset given:

The general structure for the ui and server are in place as is the code to plot the output from the server.

We want to display a plot for either the Iris or Car datasets depending on the selection

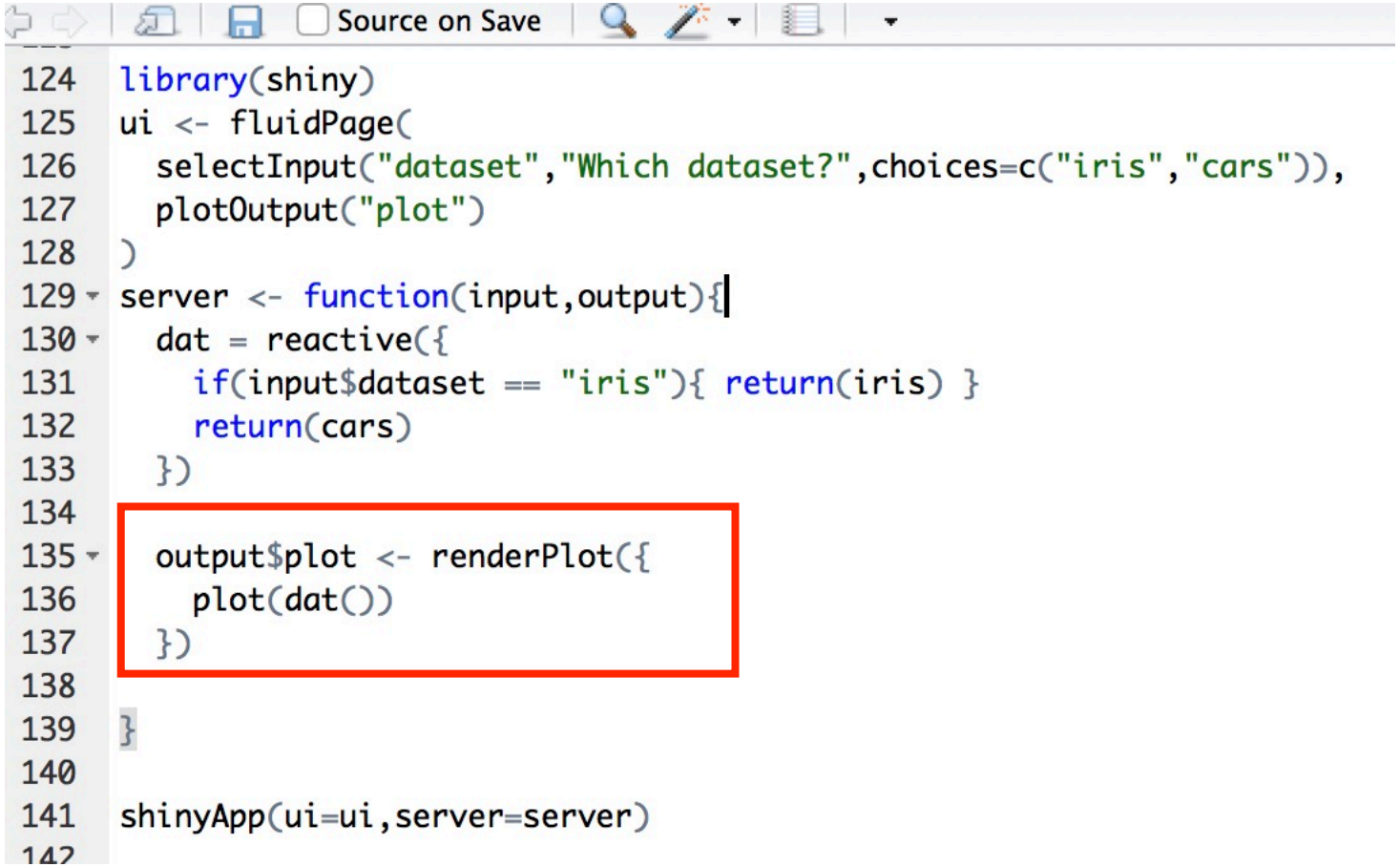


Solution

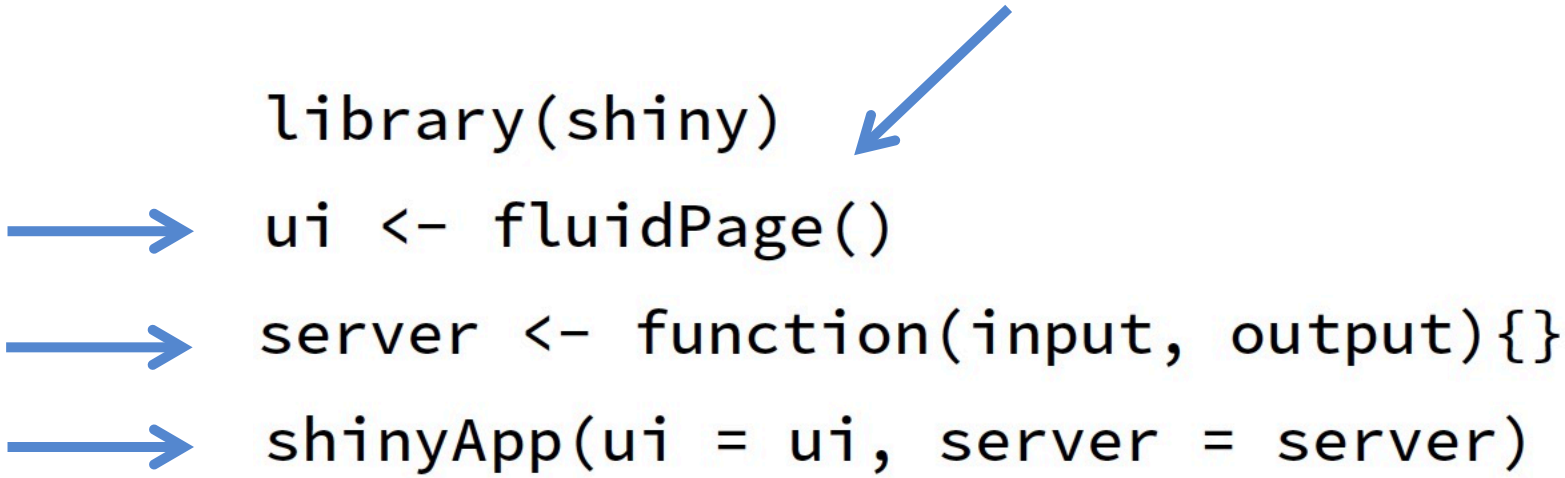
Create a Shiny App using the following template to plot the iris dataset given:

The general structure for the ui and server are in place as is the code to plot the output from the server.

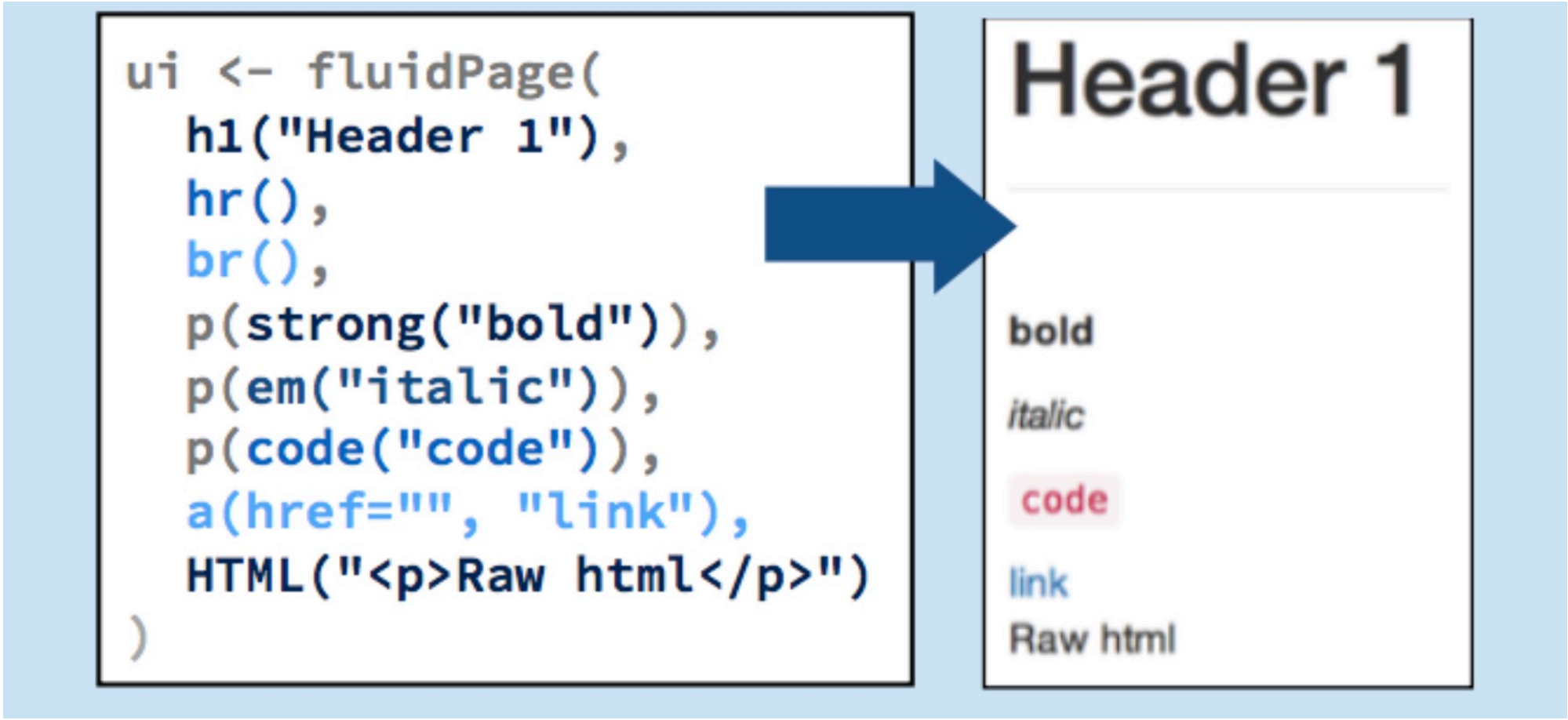
We want to display a plot for either the Iris or Car datasets depending on the selection



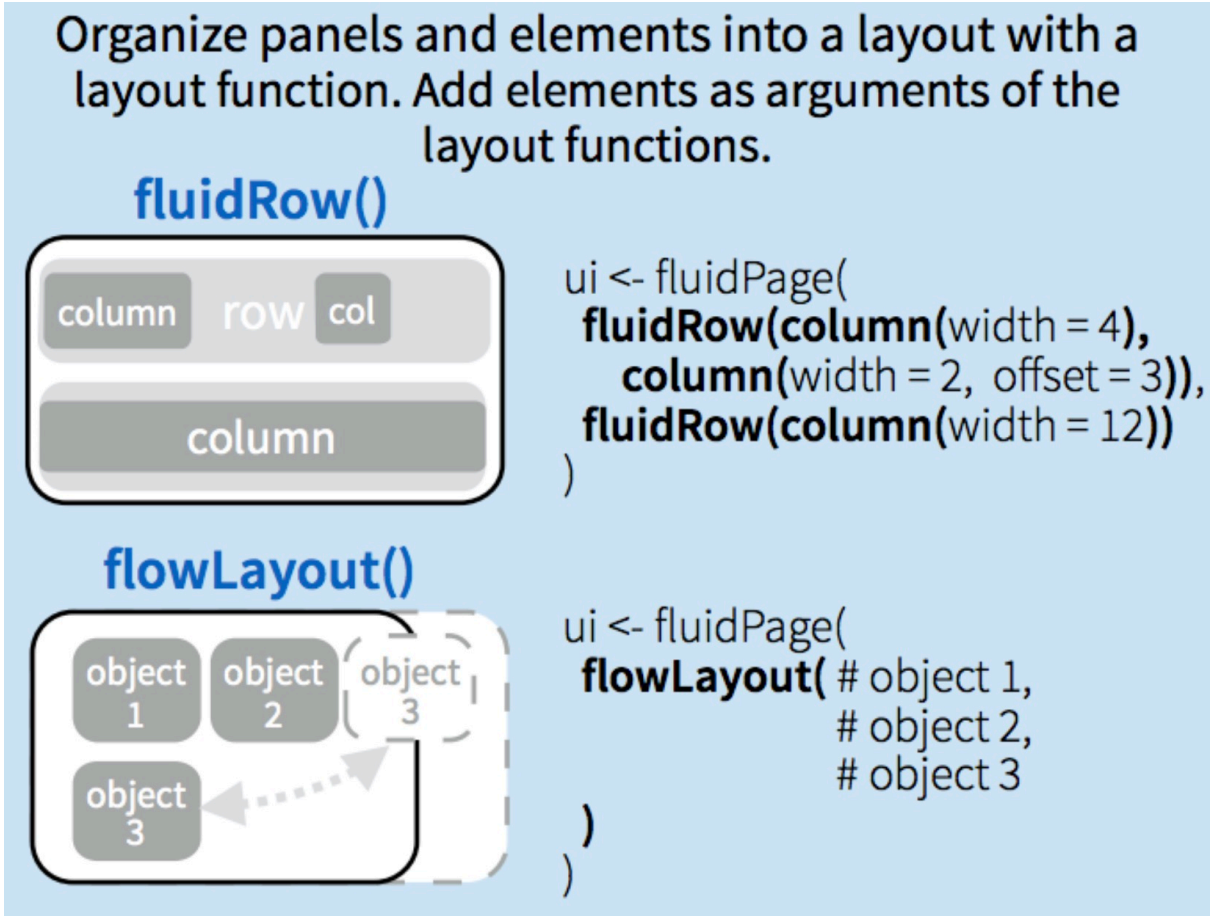
Digging deeper.



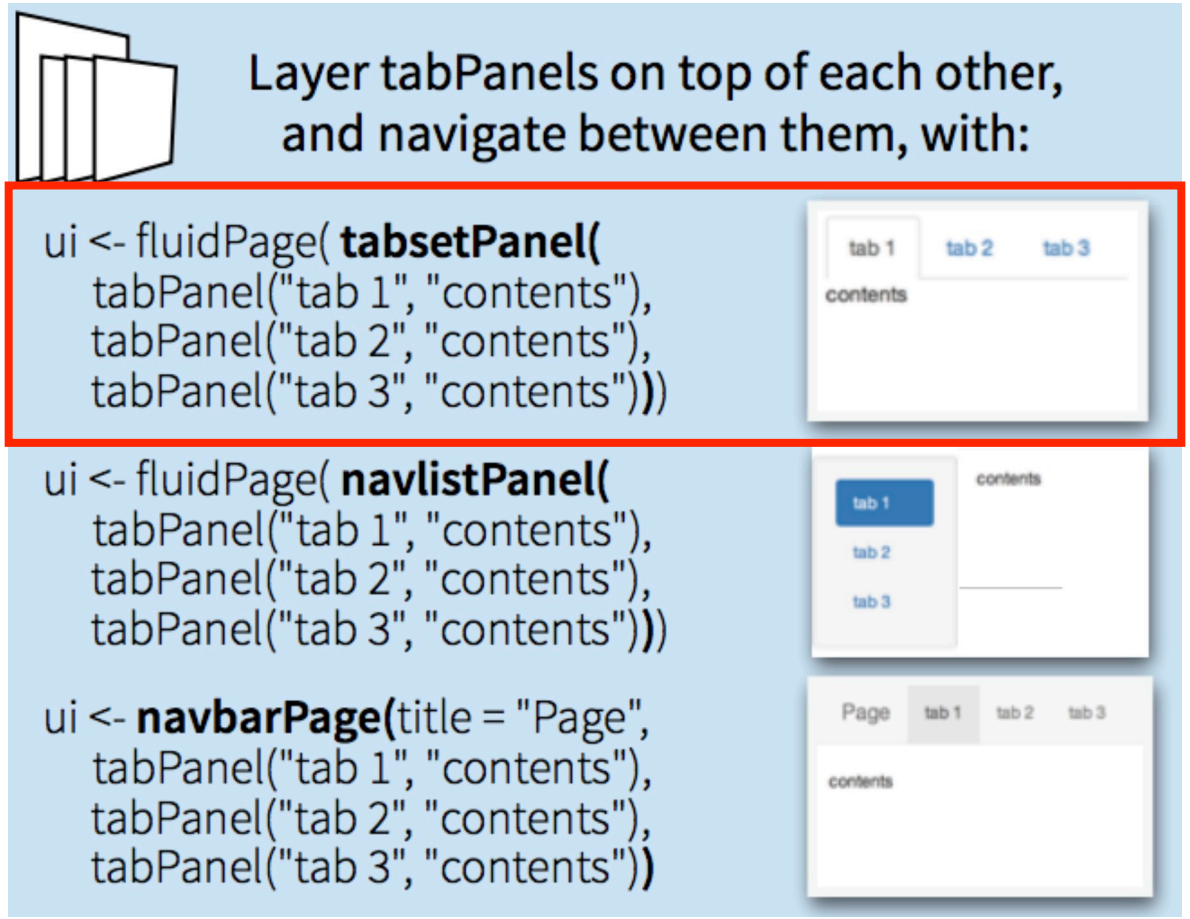
Digging a bit deeper.



Digging a bit deeper.



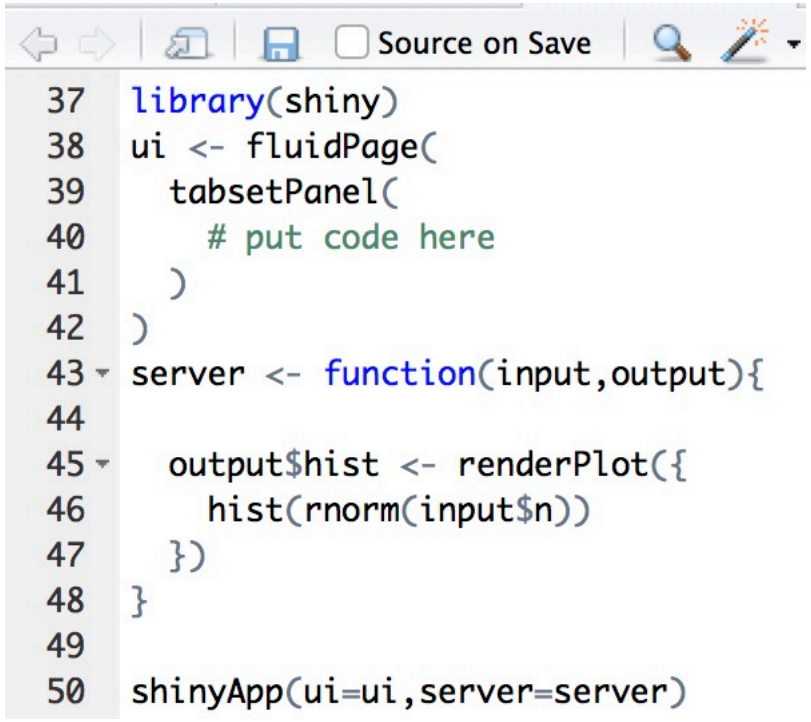
Digging a bit deeper.



Assessment – tabs and other HTML

Create a Shiny App using the following template to create a '**tabsetPanel**'ized app with **tabPanel**s that will recreate the app we made earlier.

But this time, will make one tab for inputs (**numericInput** for our n observations) and another tab for the histogram output (using **plotOutput**)



Solution

Create a Shiny App using the following template to create a '**tabsetPanel**'ized app with **tabPanel**s that will recreate the app we made earlier.

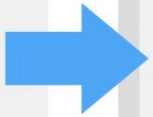
But this time, will make one tab for inputs (**numericInput** for our n observations) and another tab for the histogram output (using **plotOutput**)

```
58  library(shiny)
59  ui <- fluidPage(
60    tabsetPanel(
61      # put code here
62      tabPanel("Input",
63               numericInput(inputId = "n",
64                            label = "How many samples",
65                            value = 100)
66      ),
67      tabPanel("Output",
68               plotOutput("hist")
69      )
70    )
71  )
72  server <- function(input,output){
73
74    output$hist <- renderPlot({
75      hist(rnorm(input$n))
76    })
77  }
78
79  shinyApp(ui=ui,server=server)
80
```

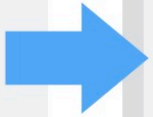
App design

Two file apps

```
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist")
)

server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}

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



```
# ui.R
library(shiny)
fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist")
)
```



```
# server.R
library(shiny)
function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}
```

App design

runApp

You can launch any app from the command line with `runApp`

```
runApp("~/Documents/App-1")
```

File path to app directory.
R will append the file path to the working directory, if path does not begin at the home directory

Deploying your app

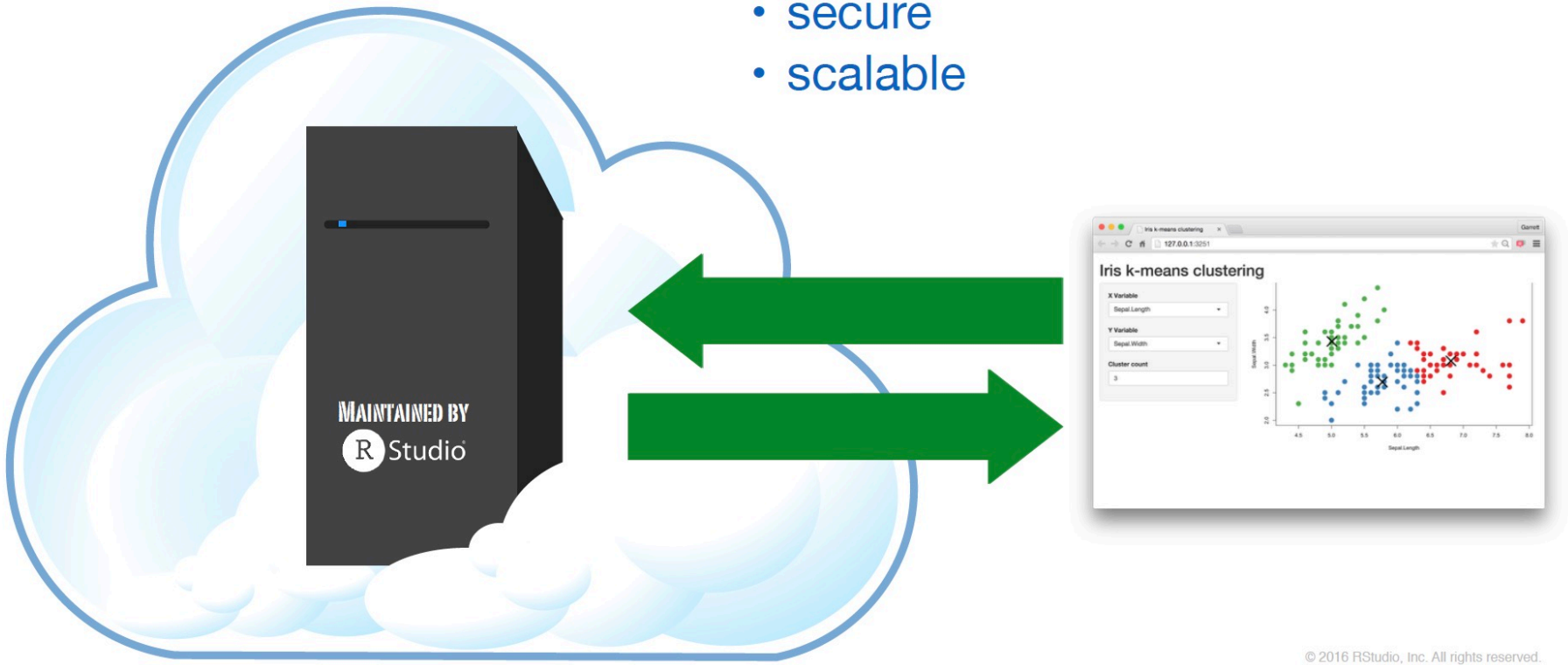
- Shinyapps.io

- Local shiny server

Shinyapps.io

A server maintained by RStudio

- easy to use
- secure
- scalable



Example deployment workflow

- Create cloud server instance
  - I prefer to use either Amazon Web Services (AWS) or DigitalOcean
- Install Shiny Server [optionally Rstudio Server as well]
- Go to Shiny App directory [default: /srv/shiny-server/shiny]
- Make folder for your project along with server.R and ui.R
  - Optionally do this through Rstudio server and allow it to create the files and example scripts
- Modify
- Enjoy and Profit @ your-ip-address:3838/shiny/your-project-folder-name

Example deployment workflow - 2

- Can place shiny scripts into a github repository and anyone can run your apps locally with the runGithub function (or url and runUrl).
  - For example
    - https://github.com/jnpaulson/MSD1000
    - https://github.com/jnpaulson/helpmeviz-malnutrition

```
library("shiny")
runGitHub("helpmeviz-malnutrition", "nosson")
```

Example workflow - 3

- Develop shiny code within an Rmarkdown file.

- This allows for static components and then shiny/interactive components within the HTML produced document.
  - Download the example here
    - Note: requires no ShinyApp function call (nor ui / server function definitions)

Other shiny topics

- Modules - https://shiny.rstudio.com/articles/modules.html

- Reactive Inputs and other Reactivity - https://shiny.rstudio.com/articles/reactivity-overview.html

Internship @ Genentech (Summer 2017)

Genentech

South San Francisco, California

Number of Positions: 4–6

Type of Student: PhD in statistics, biostatistics, or related field

Deadline for Applying: January 29, 2017, but offers may be made earlier

Our biostatistics summer interns work for 10–12 weeks under the supervision of experienced biostatisticians on theoretical or applied problems with direct relevance to ongoing clinical or nonclinical drug development research in areas such as oncology, immunology, infectious disease, ophthalmology, and neuroscience. Specific topics include problems in genomics and, more generally, translational research to late-stage clinical trials and post-marketing evaluations. At the end of the internship, students give a department-wide presentation on their research topic. It is not uncommon for an intern to summarize their work in a peer-reviewed publication.

Applicants must be at least 18 and pursuing a PhD in statistics or biostatistics. They must have at least one year of graduate work by May 2016 and be returning to school in the fall of 2016. The applicant must be legally authorized to work in the United States. In addition, applicants should have a good working knowledge of R, S-Plus, or SAS and good communication skills.

Contact: Please send CV, personal statement of interest, and a letter of recommendation to *gnebiostatsummerintern@gene.com*.