An Introduction to R Shiny

Building interactive R based web apps using Shiny

What is Shiny ?

Open Source web application framework for R, developed by R Studio.

Shiny makes it easy to turn analytical analysis into stylish, interactive web apps, presentable to a wider audience.

Link to a few examples of Interactive web apps made using Shiny:

http://shiny.rstudio.com/gallery/

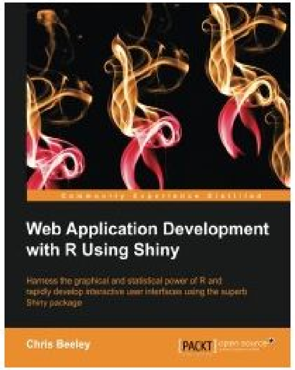
http://showmeshiny.com

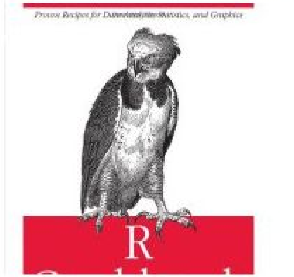
www.showmeshiny.com

About

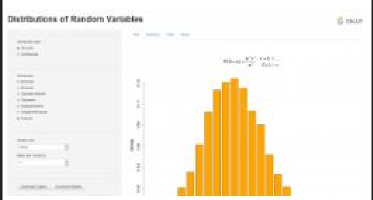
Search ... Search

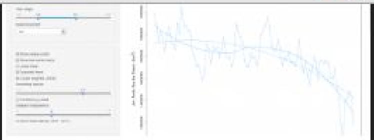
GIFT GUIDE


Community Experience Distilled
Web Application Development with R Using Shiny
Harness the graphical and statistical power of R and rapidly develop interactive user interfaces using the superb Shiny package
Chris Beeley
PACKT open source



functions.



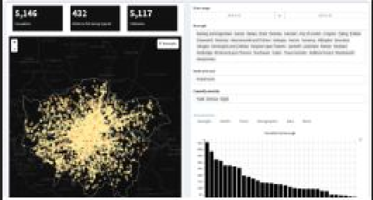


DALLAS POLICE LIVE TRACKER

Keep up with Dallas police calls in real time, with a map and table.

LONDON ROAD CASUALTIES

Road casualties reported in Greater London between 2005 and 2014.



BIKE SHARE MAP

Map of bike share systems around the world, including current availability for some.

WITTGENSTEIN CENTRE DATA EXPLORER

LABOR FORCE STATISTICS EXPLORER

RCHARTS NEW YORK TIMES

Advertiser disclosure - When you are assisted by links on this site some retailers might pay us a small commission. Hide

What is needed to build the app using Shiny

Latest version of R installed in system.

“Shiny” package installed.

Yes, you need to have some knowledge of HTML, CSS or Javascript. However, Shiny makes it simple and easy.

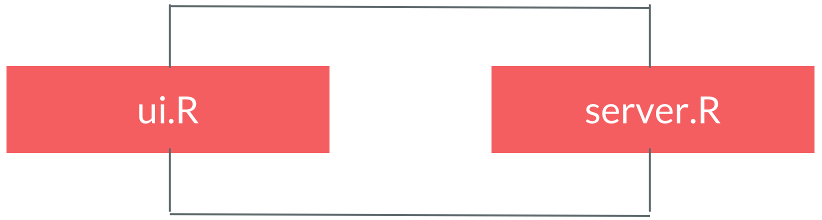
We will use R Studio for programming.

Getting Started : **UI.R** and **Server.R**

Install.packages("shiny")

Create user interface in UI.R file.

Create the server.R file for computational purposes,



Create user interface, control the layout, widgets, interface that captures user inputs.
Also, displays the output.

Eg. the title, page layout, text input, radio buttons etc.

Set of instructions that uses the input provided by the user and processes them to produce the required output which is further displayed by ui.R script.

Structure of ui.R

```
library(shiny)          #Remember to load the shiny package

#Define UI for the shiny application here.

shinyUI(fluidPage(

#Application title

titlePanel(),

#sidebarLayout(

        sidebarPanel(),

        mainPanel(),

))

)
```

Structure of server.R

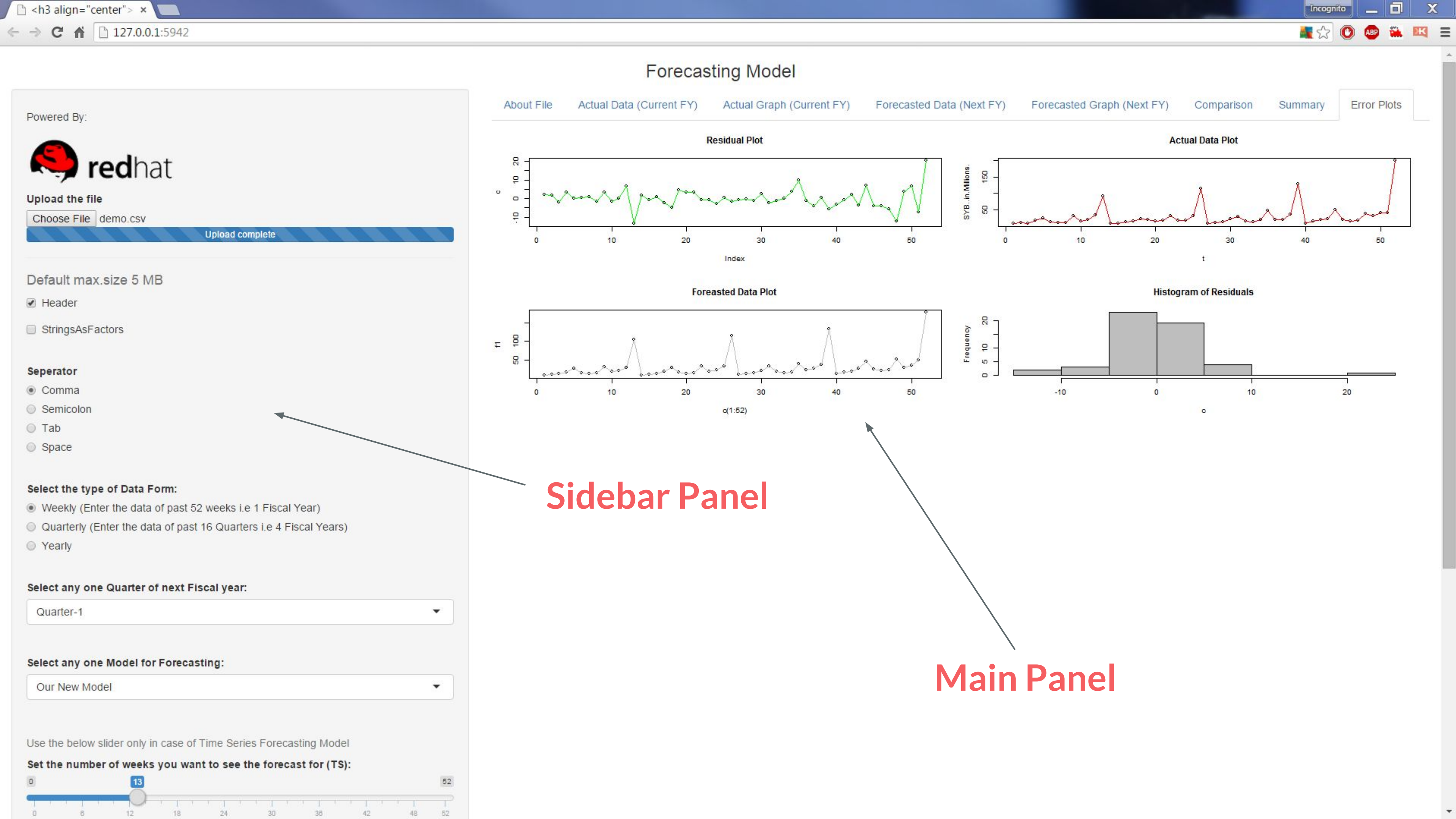
```
library(shiny)

shinyServer(

        function(input, output) {

        }

)
```



Forecasting Model

Powered By:

redhat

Upload the file

demo.csv

Default max.size 5 MB

[✓ Header]

StringsAsFactors

Seperator

[✓ Comma]
Semicolon
Tab
Space

Select the type of Data Form:

[✓ Weekly (Enter the data of past 52 weeks i.e 1 Fiscal Year)]
Quarterly (Enter the data of past 16 Quarters i.e 4 Fiscal Years)
Yearly

Select any one Quarter of next Fiscal year:

Quarter-1

Select any one Model for Forecasting:

Our New Model

Use the below slider only in case of Time Series Forecasting Model

Set the number of weeks you want to see the forecast for (TS):

0 — 13 — 52

Running the Shiny App

Save both the files ui.R and server.R in the working directory of R or any other folder.

Run the app in local system using runApp() command specifying the folder name or use the runApp icon from the R Studio. There is option to run the app in browser as well.

Share the app over the web (using github, dropbox, gist or ShinyApps etc).

Chapter - 2

More Deeper into Shiny

My first shiny App - Hello Shiny

Create a very simple shiny app - no computations but just display some text in the title panel, sidebar panel and the main panel

- Create the UI and Server files.
- Save the files in the working directory &
- Finally, run the app.

ui.R

```r
shinyUI(fluidPage(
))
```

server.R

```r
shinyServer(function(input, output) {
})
```

This code is the bare minimum needed to create a Shiny app. The result is an empty app with a blank user-interface, an appropriate starting point for this lesson.

Layout

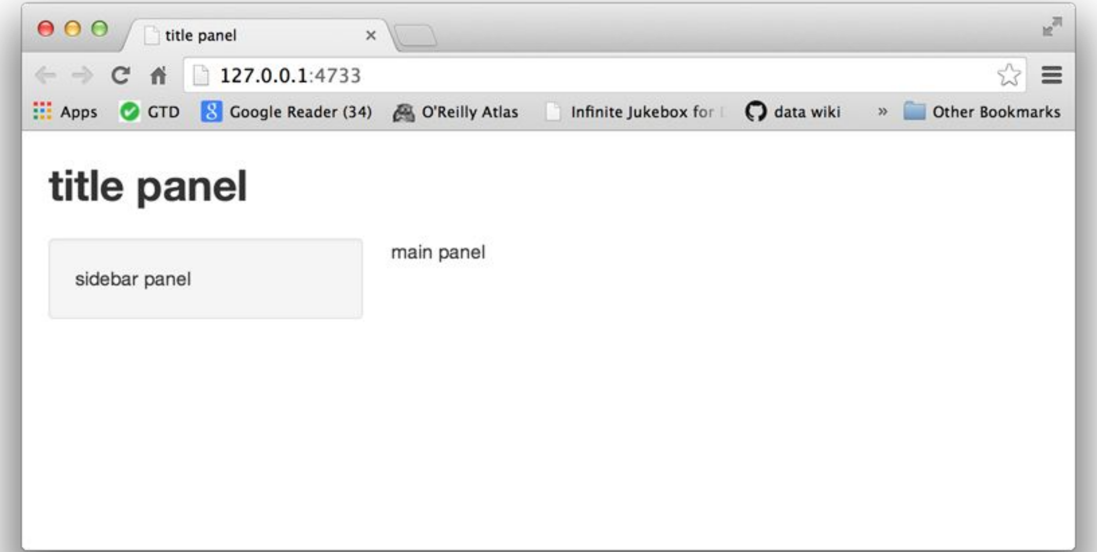
Shiny `ui.R` scripts use the function `fluidPage` to create a display that automatically adjusts to the dimensions of your user's browser window. You lay out your app by placing elements in the `fluidPage`function.

For example, the `ui.R` script below creates a user-interface that has a title panel and then a sidebar layout, which includes a sidebar panel and a main panel. Note that these elements are placed within the`fluidPage` function.

```
# ui.R

shinyUI(fluidPage(
  titlePanel("title panel"),

  sidebarLayout(
    sidebarPanel( "sidebar panel"),
    mainPanel("main panel")
  )
))
```



title panel

sidebar panel

main panel

titlePanel and sidebarLayout are the two most popular elements to add to fluidPage. They create a basic Shiny app with a sidebar.

sidebarLayout always takes two arguments:

- sidebarPanel function output
- mainPanel function output

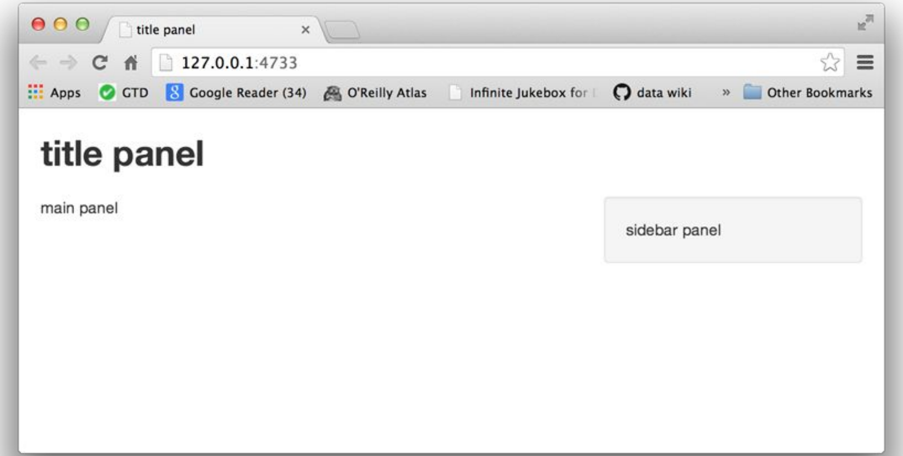
These functions place content in either the sidebar or the main panels. The sidebar panel will appear on the left side of your app by default. You can move it to the right side by giving sidebarLayout the optional argument position = "right".

```r
# ui.R

shinyUI(fluidPage(
  titlePanel("title panel"),

  sidebarLayout(position = "right",
    sidebarPanel( "sidebar panel"),
    mainPanel("main panel")
  )
))
```

`titlePanel` and `sidebarLayout` create a basic layout for your Shiny app, but you can also create more advanced layouts. You can use `navbarPage` to give your app a multi-page user-interface that includes a navigation bar. Or you can use `fluidRow` and `column` to build your layout up from a grid system.



title panel

main panel

sidebar panel

Images

Images can enhance the appearance of your app and help your users understand the content. Shiny looks for the `img` function to place image files in your app.

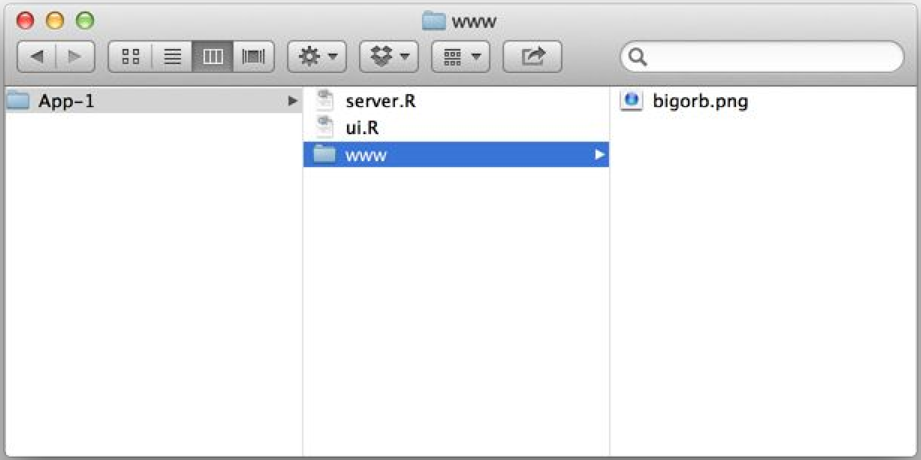
To insert an image, give the `img` function the name of your image file as the `src` argument (e.g.,`img(src = "my_image.png")`). You must spell out this argument since `img` passes your input to an HTML tag, and `src` is what the tag expects.

You can also include other HTML friendly parameters such as height and width. Note that height and width numbers will refer to pixels.

```
img(src = "my_image.png", height = 72, width = 72)
```

The `img` function looks for your image file in a specific place. Your file *must* be in a folder named `www` in the same directory as the `ui.R` script. Shiny treats this directory in a special way. Shiny will share any file placed here with your user's web browser, which makes `www` a great place to put images, style sheets, and other things the browser will need to build the wep components of your Shiny app.

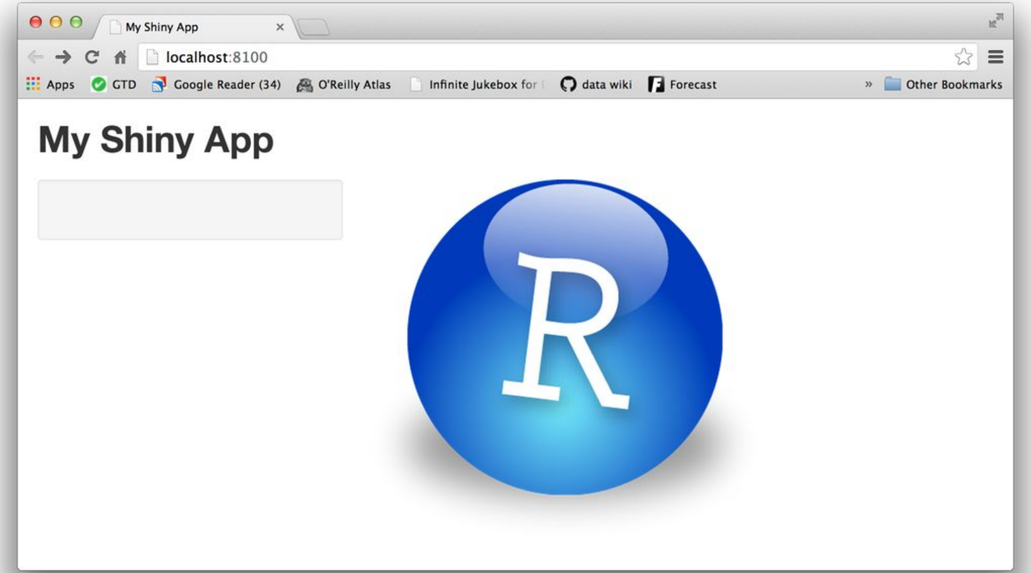
So if you want to use an image named bigorb.png, your `App-1` directory should look like this one:



With this file arrangement, the `ui.R` script below can create this app.

```r
# ui.R

shinyUI(fluidPage(
  titlePanel("My Shiny App"),
  sidebarLayout(
    sidebarPanel(),
    mainPanel(
      img(src="bigorb.png", height = 400, width = 400)
    )
  )
))
```

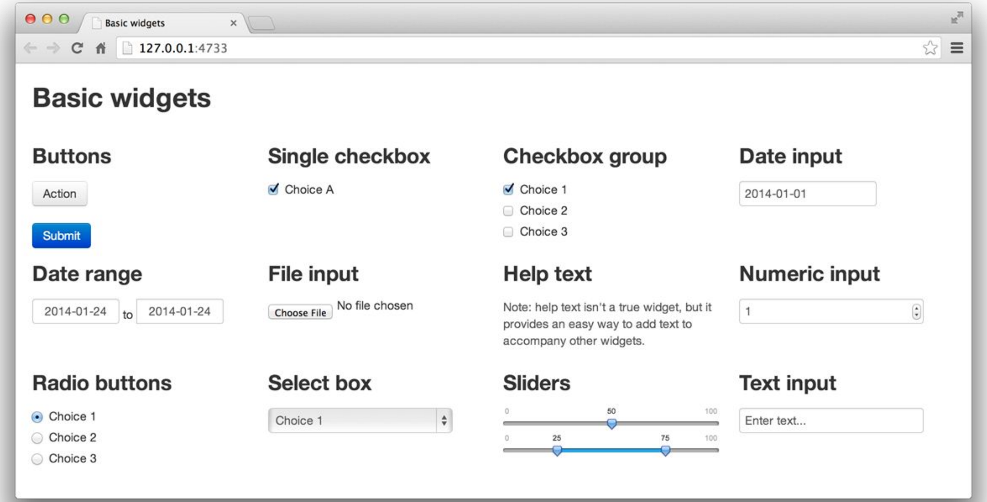


My Shiny App

Chapter - 3

Add 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



Note: help text isn't a true widget, but it provides an easy way to add text to accompany other widgets.

Each widget function requires several arguments. The first two arguments for each widget are

- A **Name for the widget.** The user will not see this name, but you can use it to access the widget's value. The name should be a character string.
- A **label.** This label will appear with the widget in your app. It should be a character string, but it can be an empty string `""`.

In this example, the name is "action" and the label is "Action":`actionButton("action", label = "Action")`

The remaining arguments vary from widget to widget, depending on what the widget needs to do its job. They include things the widget needs to do its job, like initial values, ranges, and increments. You can find the exact arguments needed by a widget on the widget function's help page, (e.g., `?selectInput`).

The `ui.R` script below makes the app pictured above. Change your own `App-1` `ui.R` script to match it, and then launch the app (`runApp("App-1")`, select Run App, or use shortcuts).

```r
shinyUI(fluidPage(
  titlePanel("Basic widgets"),

  fluidRow(
      actionButton("action", label = "Action"),
      br(),
      br(),
      submitButton("Submit")),

      h3("Single checkbox"),
      checkboxInput("checkbox", label = "Choice A", value = TRUE)),

      checkboxGroupInput("checkGroup",
        label = h3("Checkbox group"),
        choices = list("Choice 1" = 1,
           "Choice 2" = 2, "Choice 3" = 3),
        selected = 1)),

      dateInput("date",
        label = h3("Date input"),
        value = "2014-01-01"))
  ),
  fluidRow(
      dateRangeInput("dates", label = h3("Date range"))),
      fileInput("file", label = h3("File input"))),
      h3("Help text"),
      helpText("Note: help text isn't a true widget,",  "but it provides an easy way to add text to",
        "accompany other widgets.")),
      numericInput("num", label = h3("Numeric input"),
        value = 1))
  ),
  fluidRow(
      radioButtons("radio", label = h3("Radio buttons"),
        choices = list("Choice 1" = 1, "Choice 2" = 2, "Choice 3" = 3),selected = 1)),

      selectInput("select", label = h3("Select box"),
        choices = list("Choice 1" = 1, "Choice 2" = 2, "Choice 3" = 3), selected = 1)),
          sliderInput("slider1", label = h3("Sliders"),
        min = 0, max = 100, value = 50),
      sliderInput("slider2", "", min = 0, max = 100, value = c(25, 75))
      ),
      textInput("text", label = h3("Text input"),
        value = "Enter text..."))
  )
))
```

Chapter - 4

Reactive Output

Two steps

You can create reactive output with a two step process.

1. Add an R object to your user-interface with `ui.R`.
2. Tell Shiny how to build the object in `server.R`. The object will be reactive if the code that builds it calls a widget value.

```r
# ui.R
shinyUI(fluidPage(
  titlePanel("censusVis"),
  sidebarLayout(
    sidebarPanel(
      helpText("Create demographic maps with information from the 2010 US Census."),
      selectInput("var",
        label = "Choose a variable to display",
        choices = c("Percent White", "Percent Black","Percent Hispanic", "Percent Asian"),
        selected = "Percent White"),
      sliderInput("range",
        label = "Range of interest:",
        min = 0, max = 100, value = c(0, 100))),

    mainPanel(
      textOutput("text1")))))
```

```r
# server.R

shinyServer(function(input, output) {
     output$text1 <- renderText({
          "You have selected this"
     })
  }
)
```

Shiny tracks which outputs depend on which widgets. When a user changes a widget, Shiny will rebuild all of the outputs that depend on the widget, using the new value of the widget as it goes. As a result, the rebuilt objects will be completely up-to-date.

This is how you create reactivity with Shiny, by connecting the values of `input` to the objects in `output`. Shiny takes care of all of the other details.

Use widget values

If you run the `server.R` script above, the Shiny app will display “You have selected this” in the main panel. However, the text will not be reactive. It will not change even if you manipulate the widgets of your app.

You can make the text reactive by asking Shiny to call a widget value when it builds the text. Let’s look at how to do this.

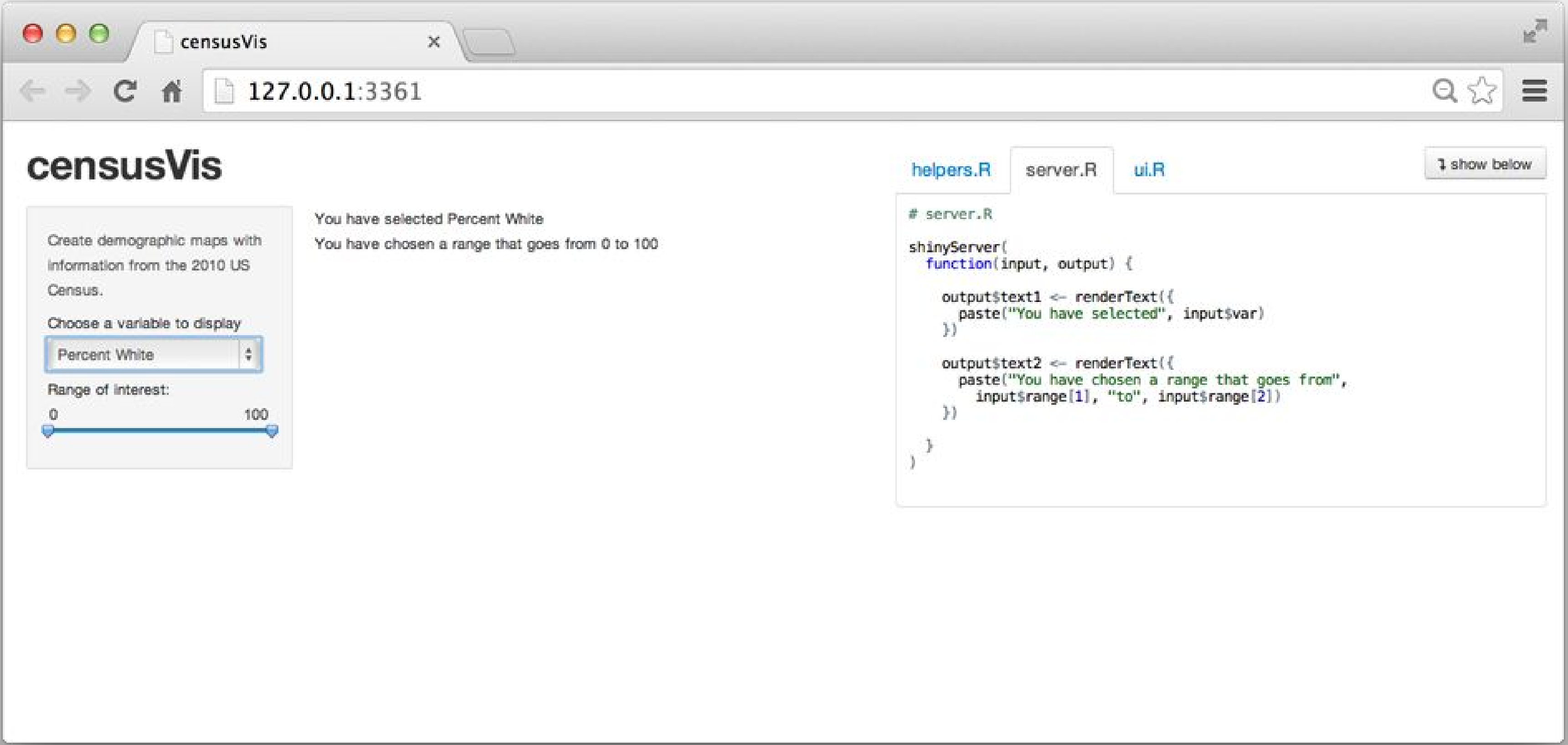
Take a look at the first line of code in `server.R`. Do you notice that the unnamed function mentions *two*arguments, `input` and `output`? You already saw that `output` is a list-like object that stores instructions for building the R objects in your app.

`input` is a second list-like object. It stores the current values of all of the widgets in your app. These values will be saved under the names that you gave the widgets in `ui.R`.

```r
# server.R
shinyServer(
  function(input, output) {

    output$text1 <- renderText({
      paste("You have selected", input$var)
    })

  }
)
```



censusVis

Create demographic maps with information from the 2010 US Census.

Choose a variable to display

Percent White

Range of interest:

0 100

You have selected Percent White

You have chosen a range that goes from 0 to 100

```
# server.R

shinyServer(
  function(input, output) {

    output$text1 <- renderText({
      paste("You have selected", input$var)
    })

    output$text2 <- renderText({
      paste("You have chosen a range that goes from",
        input$range[1], "to", input$range[2])
    })

  }
)
```

Chapter - 5

Loading R scripts and datasets

This lesson will show you how to load data, R Scripts, and packages to use in your Shiny apps. Along the way, you will build a sophisticated app that visualizes US Census data.