Learning Shiny

Robin Lovelace

01/20/2015

Contents

1 Introduction

This tutorial briefly describes **shiny**, an R package that enables development, testing and deployment of interactive web applications.

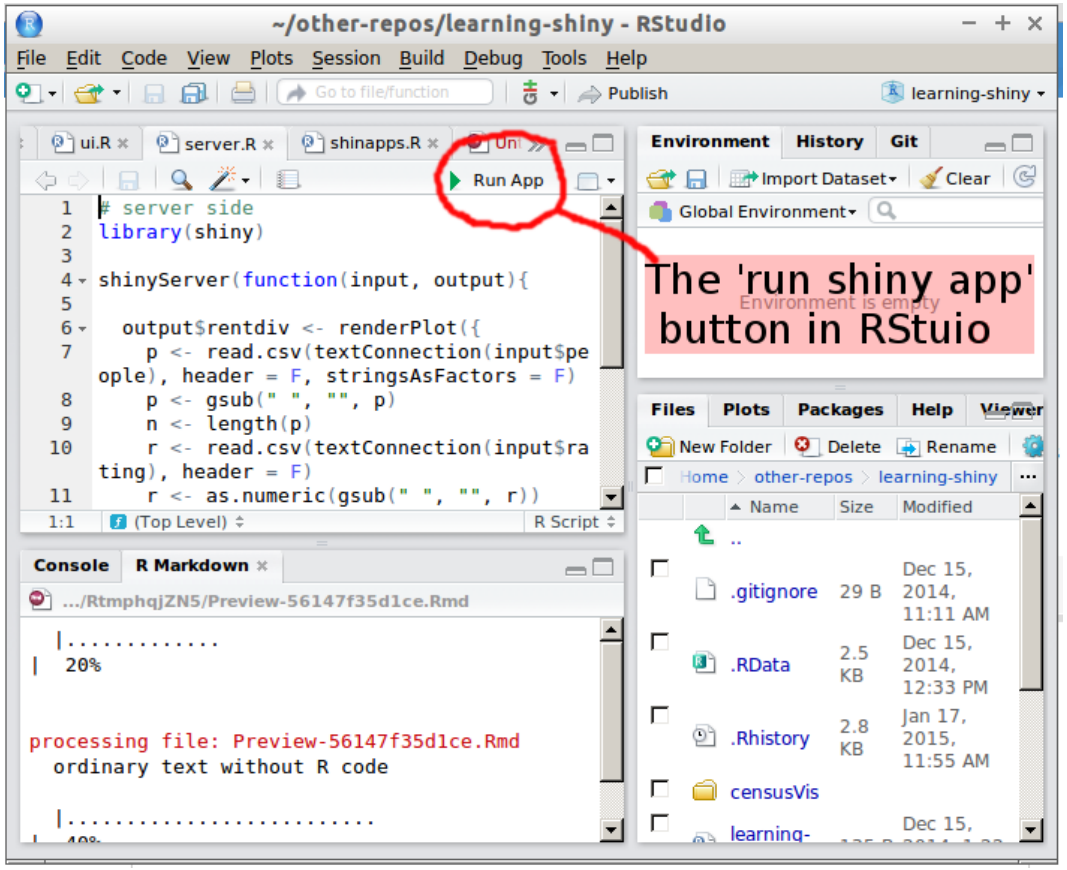
Shiny is extremely flexible, so you can build an application to help solve almost any real (or fictitious) quantifiable problem you can think of. It's a cliche worth repeating: with **shiny**, you really are limited only by your imagination.

Because **Shiny** is an open source software project with a very strong community, it has excellent documentation, including a user group, a Gallery of examples and (most recommended of all) a fantastic up-to-date online tutorial maintained by RStudio. Unless you're an R user [who lives under a rock], you'll be aware of RStudio which has revolutionised R's user appeal with a slick user interface and dozens of tools that make life easier for R beginners and developers alike. RStudio is a profit making company that has done more than any other organisation to promote, develop and communicate R to the new generation. To make a (perhaps unfair) analogy, RStudio is to R what Canonical is/was to Linux: a commercial enterprise that greatly improved the user-friendliness of the underlying program.

RStudio's impact on R is hard to overstate and it's employees are responsible for a number of the most frequently downloaded packages including ggplot2, dplyr and stringr.

This tutorial was even written and compiled in RStudio, with its seamless integration with knitr and RMarkdown which is now my default text editor for writing tutorials, vignettes and even academic papers.[1]

To cut a long story short, I RStudio for this tutorial: it even provides a button to test your applications with a single click[2]


~/other-repos/learning-shiny - RStudio
File Edit Code View Plots Session Build Debug Tools Help
Go to file/function
Publish
learning-shiny
ui.R
server.R
shinapps.R
Run App
Environment History Git
Import Dataset
Clear
Global Environment
The 'run shiny app' button in RStuio
Environment is empty
1:1
(Top Level)
R Script
Console R Markdown
.../RtmphqjZN5/Preview-56147f35d1ce.Rmd
Files Plots Packages Help Viewer
New Folder Delete Rename
Home other-repos learning-shiny

```
# server side
library(shiny)

shinyServer(function(input, output){

  output$rentdiv <- renderPlot({
    p <- read.csv(textConnection(input$people), header = F, stringsAsFactors = F)
    p <- gsub(" ", "", p)
    n <- length(p)
    r <- read.csv(textConnection(input$rating), header = F)
    r <- as.numeric(gsub(" ", "", r))
```

```
  |.............
|   20%


processing file: Preview-56147f35d1ce.Rmd
  ordinary text without R code

  |.........................
```

Name	Size	Modified
..		
.gitignore	29 B	Dec 15, 2014, 11:11 AM
.RData	2.5 KB	Dec 15, 2014, 12:33 PM
.Rhistory	2.8 KB	Jan 17, 2015, 11:55 AM
censusVis		
learning-		Dec 15,

Because there is so much high quality material on shiny and web app development overall, it would be easy to repeat what's already out there. That is not the intention, so this tutorial builds on and creates a narrative for existing tutorials, rather than trying to rebuild the wheel. The aim is simple: **to navigate new users of R through the exciting new world of online visualisation that it opens up via Shiny**. Before we proceed I'd like you to take a quick peak at the following shiny resources, to get a sense of what else is out there:

- RStudio's shiny website.
- The shiny github repo.
- Stackexchange questions about shiny.
- An academic paper outlining the potential of shiny in precision farming.
- The shiny cheat sheet.

[1]RMarkdown's recently added support for Pandoc citations has enabled it to be used as a fully fledged text editor. If I were to write my thesis again, I'd probably do it in RMarkdown to save all the \s needed for pure Late X!

[2]RStudio seemingly does **not** provide keyboard shortcuts for Shiny, something you'll see by tapping `Ctl-Shift-K` from your RStudio console now (hint hint RStudio developers!).

2 What shiny can (and cannot) do

In addition to checking out the above resources, aspiring 'web app' creators should think about what **shiny** is well suited for before diving in.

shiny is good for:

- Relatively simple visualisations of small datasets that can be processed 'in the cloud'.
- 'Proof of concept' demonstrations of the applications you are thinking of.
- Rapid deployment if you already know R

shiny is not so good at:

- Very large applications.
- Map serving: this may be best left to something heavier duty like GeoServer
- Collecting user information (at present)

To see what **shiny** does well and not-so-well, it's worth taking a look at some examples.

- An excellent real world example is the Ebola-Dynamic-Model which provides an interactive model of the spread of Ebola.
- The 'word cloud' example in RStudio's gallery demonstrates that it's not *just* quantitative information that can be produced.
- The 'google-charts' example shows how **shiny** can integrate with existing web visualisation tools.
- My favourite example is the superzip also from the gallery. This demonstrates the ability to place data on-top of interactive maps.
- There is an academic paper explaining how shiny was used to create a proof-of-concept app for precision farming (Jahanshiri and Shariff 2014).

Alternatives to **shiny** are:

- Google charts which integrates with R (and shiny) via the **googleVis** package
- Plotly, which has excellent integration with R via the **plotly** package.
- d3 is an extremely flexible JavaScript visualisation package, which interfaces to R via the d3Network package.

Having browsed these options (and perhaps others) and deciding that **shiny** is the tool for the job, it's time to get started.

3 Getting started with shiny

Shiny in fact contains a wide range of pre-built apps that can be explored and modified by the user to see how the system works.

To see what examples are available, use the `runExample()` function:

```r
runExample()
```

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

Based on the results from this, you can proceed. Let's run the widgets example:

```r
runExample("07_widgets")
```

The most informative example for beginners is probably the first in most cases:

```r
runExample("01_hello")
```

Spend some time taking a look at the `ui.R` and `server.R` files. Note that **every shiny app must contain these two files**, placed in a folder of the app's name. That's fundamental to the app's structure:

```
|-- hi
|   |-- change-app-lines.R
|   |-- server.R
|   `-- ui.R
|-- rentSplit
|   |-- server.R
|   |-- shinyapps
|   |   `-- robinlovelace
|   |       `-- rentSplit.dcf
|   `-- ui.R
|-- learning-shiny.Rmd
`-- learning-shiny.R
```

The directory structure illustrated above shows a typical R working director with `shinyapps.R` being the only file in the route directory, which can be used to run the apps in the sub-folders `hi` and `rentSplit`. These are apps, containing the critical **ui** and **server** components. Note that `rentSplit` contains another sub-directory `shinyapps`. This repository does not affect the overall performance of the app: it's used to embed the app online on the free app hosting site shinyapps.io.

The rentSplit app that we are going to build today is hosted on this website and available to anyone worldwide, through the following link: https://robinlovelace.shinyapps.io/rentSplit/

Before building your own app (the next section), it's worth looking at the contents of the `runExample("01_hello")` app in some detail. We will modify this example for our own needs.

3.1 Downloading the project repository

Navigate to the project's repository: https://github.com/Robinlovelace/learning-shiny

Click on 'Download ZIP' on the right hand side of this page. Unzip the folder. Navigate into this folder and open the file `learning-shiny.Rproj` in RStudio. Take a look around and make yourself at home in this folder: it will be your digital home throughout this tutorial!

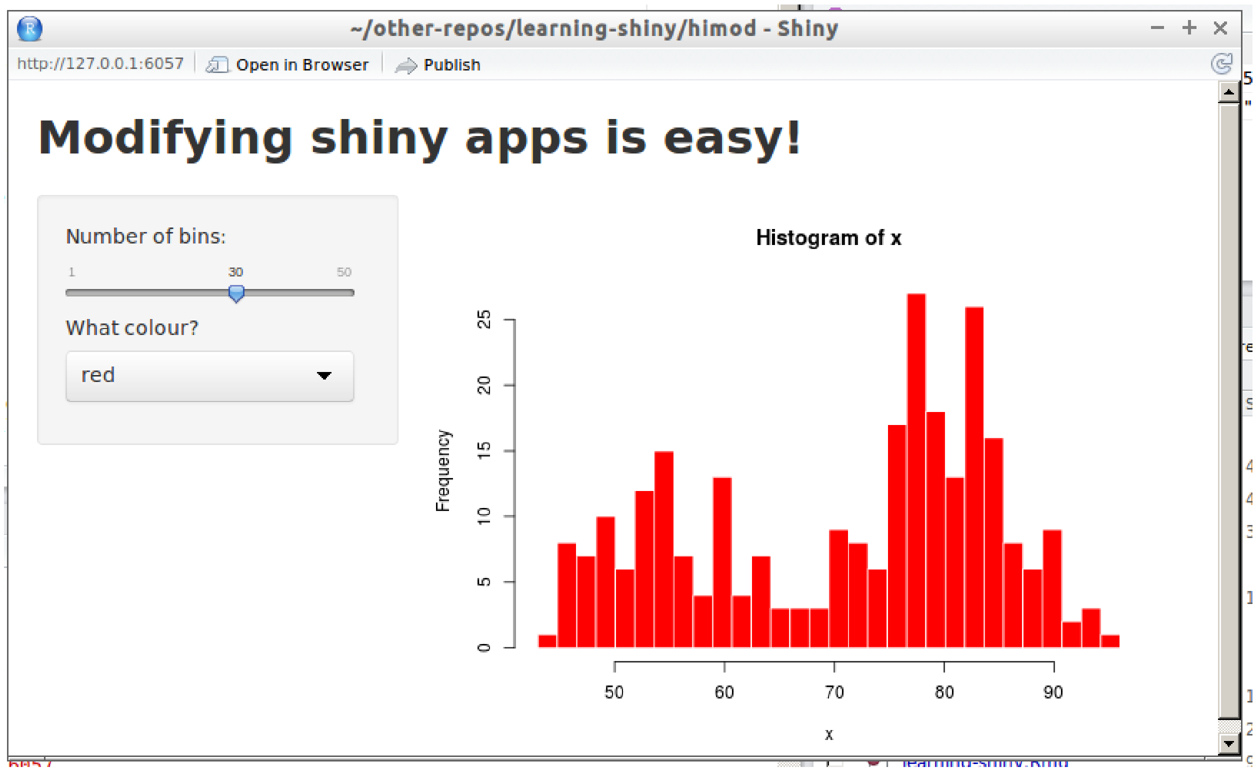
4 Modifying an app for your own needs

The cornerstone of scientific progress can be summarised in a single phrase by Isaac Newton:

> "Building on the shoulders of giants"

This means that instead of starting from scratch every time, sometimes we can move forward faster by modifying what someone else has already done, giving due credit to the original source. This is ethic underlying software allows open source options such as Linux and R to triumph over proprietary programs such as Microsoft Windows and SAS.

So let's make a modification to the `01_hello` app, so that the user can *decide* which colour they want the histogram to be. The end result should look like this:



How do we create this? First I'd suggest creating a new folder, perhaps called `himod` to contain the new app's code so it runs from the project's root directory. Next we must identify some key features of the shiny app.

4.1 The user interface (ui.R)

This is probably the best place to start as it's where you define the user's input into the project. `surver.R` can then be instructed to take the user's input and base the output on the input (yes, `server.R` creates a new function with `function(input, output)`!).

We must look past some essential but less interactive elements before getting to the interactive widgets that can be changed. These usually include:

```
shinyUI(fluidPage(
  titlePanel("Hello Shiny!"),
  sidebarLayout(
    sidebarPanel(
#       ... content here
     )
    )
))
```

We don't need to go into detail about what these are and what they do. This is explained on RStudio's shiny tutorial and shiny's internal documentation system. Typing `?sidebarLayout`, for example, will display some very useful help! As Lesson 2 from RStudio's tutorial explains, `shinyUI(fluidPage( ))` alone will create the display. It is *fluid* because it automatically adjusts to the width of the user's device.

What we are interested in is how to add additional widgets, a full list of which is presented here: http://shiny.rstudio.com/tutorial/lesson3/ .

To add an option selector widget, we use, logically enough, the `selectInput()` widget function. The essential arguments are as follows:

```
selectInput(inputId = "inputID",
  label = "What the user sees",
  choices = list("list", "of", "options"),
  selected = "default option" )
```

4.2 Task 1a: adding a color selector option

You can add new widgets to the `ui.R` script without affecting the server side. Do this now to create the drop down menu of color selection illustrated above.

> Hint: Check RStudio's page on shiny widgets (lesson 3 at shiny.rstudio.com/tutorial/) for a list of available widgets

When you are complete, run the app either by typing `runApp("hi")` or by clicking on the 'Run App' button in RStudio. Note that when new content is added to the user interface side of shiny, it does not necessarily affect the app (in this case, we can select a colour without any impact on the resulting graph). To change the output, we need to move to the server side.

4.3 The server side (server.R)

One key argument you should have noticed in the previous section is that all widgets must have an ID, such as `bins` in the "hi" app that we're working on:

```
      sliderInput("bins",
        "Number of bins:",
        min = 1,
        max = 50,
        value = 30)
    ),
```

In the above code chunk there are five arguments, the latter of which (`min`, `max`, and `value`) are specific to the slider: `textInput()`, `fileInput()` and the other 10+ widgets will take slightly different arguments. However, **all widgets require an ID and a label**. These are always the first two arguments in shiny widgets and they must be user-defined *character strings* (this means text surrounded by quote marks, single `'like this'` or double `"like this"`). The arguments accepted by `textInput()` and `fileInput()` widget functions are shown below to illustrate the point: the first two arguments are essential and remain unchanged for all shiny widget functions.

```
textInput(inputId = ..., label = ..., value = ...,)
fileInput(inputId = ..., label = ..., multiple = ..., accept = ...)
```

Although we do not need to explicitly state the arguments in our code (`sliderInput(inputId = "bins", label = "nbins", ...)` is identical to `sliderInput("bins", "nbins", ...)` because of positional matching in R functions (Wickham 2014), it is good practice to state the arguments explicitly, especially when you are learning new functions and communicating the code to novices. `inputID` is the first argument in every widget function because it is the most important: **it is the only link between ui.R and server.R**.

When we look at the code in `server.R` in the "hi" app example, we can see how the `bins` input object (referred to using its ID, `"bins"`) is called:

```
    bins <- seq(min(x), max(x), length.out = input$bins + 1)
```

Thus, to refer to objects created in ui.R, we must use the following notation:

```
input$inputID
```

Understanding this is critical to completing the next task.

4.4 Task 1b: make the graph's color change by modifying server.R

Based on the above description of the link between `ui.R` and `server.R`, make the colour selected in the user interface implemented in **task 1a** be an active input into the graph: make the graph change colour.

When you've completed tasks 1a and 1b, it's time to create a shiny app from scratch. Following the learning by doing ethic, it's recommended that you type the entirety of the code for this app (comprising of only 2 files, `server.R` and `ui.R`), rather than copy-and-pasting from the rentSplit example already contained in this project.

As always, before we jump into writing the code, we should think a little about the wider context: think more, work less!

5 Building RentSplitR

To see how shiny works, there is no better way than starting with a *pen and paper*. That's because your web application should be driven by a clear sense of purpose. Just learning **shiny** for the hell of it, with no use cases in mind could be a waste of time: you'll only retain the techniques if you use them frequently on a project that is worth the time invested. So here's a mock example of the need for a new app. The need is as follows:

- Many people who share accommodation are not happy with their room/rent.
- It's difficult to judge how to set the rent fairly, based on how good different rooms are.
- Therefore there is a need for an interactive tool to help decide the right rent.

5.1 Task 2: re-create RentSplit to make it your own

Based on the code contained in the rentSplit folder, re-write this app. Resisting the temptation to copy-and-paste, and referring frequently to the RStudio tutorial will improve your understanding. It's likely that error messages will appear the first few times you try to run the app - this is normal! Persevere and you'll feel a sense of satisfaction when the app, written entirely by you, finally works.

6 Task 3: write your own app

By carefully completing the previous two tasks you should have built-up the skills needed to build and deploy your own apps. This final task is open ended: create your own app that meets a real world need!

References

Jahanshiri, Ebrahim, and Abdul Rashid Mohd Shariff. 2014. "Developing Web-Based Data Analysis Tools for Precision Farming Using R and Shiny." *IOP Conference Series: Earth and Environmental Science* 20 (June): 012014. doi:10.1088/1755-1315/20/1/012014. http://stacks.iop.org/1755-1315/20/i=1/a=012014?key=crossref.6d48229e96a8bbfd5a553a3d0055472c.

Wickham, Hadley. 2014. *Advanced R*. CRC Press. http://www.crcpress.com/product/isbn/9781466586963http://adv-r.had.co.nzhttp://www.crcpress.com/product/isbn/9781466586963http://adv-r.had.co.nz.