Practical training Shiny interactive map

Willem Stolte

February 25, 2016

This is an instruction for building a simple Shiny app in R. We will use online data. This is the document with instructions, a ready to use app can be found in the projectfolder (CourseApps/app.R)

Start a new Shiny app source document

- In Rstudio, choose from file menu: New File / Shiny Web App …
- You can choose between
  - Single file (UI and Server) are defined as two functions in one file
  - Multiple file (ui/server) (UI and Server are defined in separate files). For more complex apps, this can be more convenient. In that case the UI and Server files can be displayed simultaneouslyon one screen.
- Here we choose the "single file" option
- The document opens with some standard, functional, content to get going
- Try and run the document by pressing the "run App" button just above the document in Rstudio
  - a Web Application appears showing a working example. It is run in Rstudio's own viewer/browser. At the top of the App window, there is a button to choose to display it in your browser.

Add new data

Now, we are going to use data from an online source. Use your own data, or use the example data from the Marine Projects database. The address is:

```
url <- "http://marineprojects.openearth.nl/geoserver/mep-nsw/ows?service=WFS&version=1.0.0&request=GetF
require(readr) # faster than standard read functions
pvis <- read_csv(url)
# or, you can specify column types and skip columns by adding a textstring
# d double, c character, i integer,  _ skip
pvis <- read_csv(url, col_types = "__cdd_____c____c_c_________d__")
# for large data, performance increases using the last option
# look whats inside
str(pvis)
View(pvis)
```

Note that there are two kinds of measurements in the dataset, aantal (number) and lengte (length).

Let's make an interactive map of the fish data, where we can choose the species, and where the size of the symbols in the map indicates the number of individuals of that species. In that case we need to specify the

Where to put code

An overview of all available functions can be found here

You should consider that

All reactive code will be run each time you make new input in the User Input. The reactive code should be minimized for performance

Reactive code is code within a reactive function. reactive() and renderPlot() are examples of reactive functions.

Reading in data and all general calculations can be done in the first part of the App source file, so above the specification of ui and server.

data preparation (above the UI definition)

Here we can make a subset of the data with parameter length. This has only to be done once. It is also possible to aggregate per year and/or location for example. All data preparations done here

user interface code

In this part of the code, the user interface is specified. You can define what is in a side panel and main panel. Input of data to be handled by the server script can be defined as "input functions". An example of such a function

- selectInput
  - Create a select list input control
- sliderInput (animationOptions)
  - Slider Input Widget
- textInput
  - Create a text input control

In general, type the function name (tab can be used to autocomplete) and then tab or F1 for help on the function.

In our example, we will make a list input with all different species of fish.

Server code

The server code will provide the content of the web app.

you can choose to serve tables, text, or graphs. If Shiny is combined in a markdown document, also markdown formatting can be used.

The server code should perform an action on the data for which input from the user interface is needed. Input from the input is stored in a list called "input" can can be used by the server script by calling the named element of the list, in our case e.g. "input$species". In the UI, species should be the object where the input of the organisme.naam is stored.

The served object is often a plot The function renderPlot() is used by the server to generate the plot. The plot is stored in a list called "output" for transfer to the userinterface. In our example app:

```
output$mapPlot <- renderPlot( here comes the definition of the plot   )
```

In the UI, the plot is called by the same name.

Try to rebuild the pvisApp from scratch.

tips: ctrl-a ctr-i outlines the code. This is useful in order to keep track of all the nesting