Demo: Interactive R Markdown

Environmental Data Analytics | John Fay & Luana Lima

Fall 2024

Overview

RMarkdown is a powerful tool for creating dynamic documents that combine code, text, and graphics. With the addition of widgets, it's possible to make these documents even more interactive and engaging. In this lesson, we will explore how to create interactive HTML files using RMarkdown widgets.

LESSON OBJECTIVES

1. Create plotting functions that accept variables
2. Use widgets from the `shiny` library to make interactive HTML documents

1. Using variables in plotting code

Set up the coding environment

Create a base plot

We will plot Total P as a function of time, coloring data by depth, and symbolizing by lake.

Using variables in plots

Now tweak the above code so that the variable we plot (`tp_ug`) is specified as a variable rather than hard coded in our ggplot command. We will also set the start and end dates to variables as well.

To do this we need to convince R that the value we associate with the `y` aesthetic is a variable, not a column name. This is done by the odd convention of inserting the variable within the code `!!sym()`. The `!!` (pronounced "bang-bang") combined with `sym()` are "Tidy eval helpers" necessary because tidy-R does away with quotations. See more info here.

Creating functions to generated plots with specific inputs

Next, we'll one of the uses of using variables in our plotting commands: we'll create a function to generate our plot, making it super easy to make repeate plots with slight modifications.

To do this, rather than setting our variable values in the code chunk, we just set them as inputs to a function. Then we can copy and paste out plotting code above as the code the function executes (i.e. between the `{ }`). From there, we just call the function with the inputs we want to generate a plot.

2. Interactive R with Shiny Widgets

Using the RMarkdown Cheat Sheet

Have a look at the RMarkdown cheat sheet (`Help>Cheat Sheets>RMarkdown Cheat Sheet`). Note in the lower right corner of the second page, the section on `Interactivity`. I've copied the code in this section to the Rmd document called `Demo_WidgetDemo.Rmd`. Open that document and have a look.

- Notice that there's no Knit menu; it has been replaced by `Run Document`
- Run the document: A web page appears
- Change the value in the `How many cars?` section and see how the page responds.
- Close the browser window or hit the stop sign in the Console to end the interactive session.

If you look at the code in the document, perhaps you can see how it relates to what appeared on the interactive page. If not, no worries. We'll explore some more..

Making This Rmd Document Interactive

[*** You may want to stage and commit this document before moving on… ***]

Edit the YAML to switch the Rmd to be interactve To enable your document to be run (instead of knit) you must alter your `YAML` section, that region in the very top surrounded by `---`.

- Change `output: pdf_document` to be `output: html_document`
- Just below that line, add a new line: `runtime: shiny`
- Save your Rmd and the `Run Document` option should appear.

Check that code chunks are/are not dispayed or run as desired. Before running our script, check the setting of each code chunk. In particular, note which ones are not shown in the output (echo=FALSE), not run (eval=FALSE), have warnings suppressed (warning=FALSE). This is useful for streamlining your output while not having to delete code. Note, however, that all markdown (the text outside the code chunks) will still appear in the output. We'd have to delete that, but we'll just ignore it for now.

Add and explore our interactive code Copy the code chunk from the `Demo_WidgetDemo.Rmd` and paste into the chunk below.

Run the document. Along with all the Markdown, you'll see the same interactive app as before.

Anatomy of an interaction: Widgets and Renderer For interaction, we need a **widget** and typically a **renderer**.

A widget constructs an interface prompting a user for input at runtime. The widget we just used is the `numericInput()` widget and it asks the user to provide a number. View the help on the `numericInput()` widget. The widget parameters included in our code are: - `n` is the a variable name associated with the value the user provides (`inputId`) - `How many cars?` is the prompt shown with the widget (`label`) - `5` is the default value assigned to `n` (`value`)

A renderer listens for any changes in the webpage; for example if the user changes the number in the `numericInput` widget. If it detects a change, code associated with the renderer runs. The renderer in our code is `renderTable()`, which as its name implies, renderes a table. The data used to populate the table is generated from the code inside the curly braces: `head(cars, input$n)`. In other words, it's going to create a table showing the first few rows of the "cars" dataframe. How many rows shown is defined by the `input$n` expression. And `input$n` gets its value from the widget (input) with the named variable "n", i.e. the value in our `numericInput()` widget.

- So, try modifying the variable name in the first line of code, changing `n` to `user_val`. For the renderer to get the correct value, `input$n` will have to also be updated to `input$user_val`.
- Alternatively, add another renderer to the code: `renderPrint()` which simply prints a message. Set the argument to `print(paste,"You selected",input$n)`. It should simply print the value selected in the `numericInput()` widget.

Other widgets Shiny has many other input widgets: https://shiny.rstudio.com/gallery/widget-gallery.html. Below we'll use a Select Box widget to specify the variable shown in our plot. We'll also add a slider range to limit the date shown in our plot.

> Which widgets would be appropriate for the inputs to our plotting function?

Creating our interactive plotting app

Now we'll create code that will call our plotting function, but with inputs derived from widgets! We'll use a `selectInput()` widget to set the nutrient that we want to plot, and a `sliderRange()` widget to set the start and end year to plot. And then we'll insert our `plot_it()` function created above within a `renderPlot()` renderer to show out plot!

Note: Before running this plot, set the previous code chunk to not evaluate