YintingWang_A12

Yunting Wang

3/30/2022

R Markdown

Load packages —- library(shiny) library(shinythemes) library(tidyverse)

Load data —- nutrient_data <- read_csv("Data/NTL-LTER_Lake_Nutrients_PeterPaul_Processed.csv") nutrient_data$sampledate < -as.Date(nutrient_data ata$sampledate, format = "%Y-%m-%d") nutrient_data <- nutrient_data %>% filter(depth_id > 0) %>% select(lakename, sampledate:po4)

Define UI —- ui <- fluidPage(theme = shinytheme("yeti"), titlePanel("Nutrients in Peter Lake and Paul Lake"), sidebarLayout(sidebarPanel(

```
  # Select nutrient to plot
  selectInput(inputId = "y",
              label = "Nutrient",
              choices = c("tn_ug", "tp_ug", "nh34", "no23", "po4"),
              selected = "tp_ug"),

  # Select depth
  checkboxGroupInput(inputId = "fill",
                     label = "Depth ID",
                     choices = unique(nutrient_data$depth_id),
                     selected = c(1, 7)),

  # Select lake
  checkboxGroupInput(inputId = "shape",
                     label = "Lake",
                     choices = c("Peter Lake", "Paul Lake"),
                     selected = "Peter Lake"),

  # Select date range to be plotted
  sliderInput(inputId = "x",
              label = "Date",
              min = as.Date("1991-05-01"),
              max = as.Date("2016-12-31"),
              value = c(as.Date("1995-01-01"), as.Date("1999-12-31")))),

# Output
mainPanel(
  plotOutput("scatterplot", brush = brushOpts(id = "scatterplot_brush")),
  tableOutput("mytable")
)))
```

Define server —- server <- function(input, output) {

```
# Define reactive formatting for filtering within columns
 filtered_nutrient_data <- reactive({
   nutrient_data %>%
     filter(sampledate >= input$x[1] & sampledate <= input$x[2]) %>%
     filter(depth_id %in% input$fill) %>%
     filter(lakename %in% input$shape)
 })

# Create a ggplot object for the type of plot you have defined in the UI
   output$scatterplot <- renderPlot({
    ggplot(filtered_nutrient_data(),
           aes_string(x = "sampledate", y = input$y,
                      fill = "depth_id", shape = "lakename")) +
      geom_point(alpha = 0.8, size = 2) +
      theme_classic(base_size = 14) +
      scale_shape_manual(values = c(21, 24)) +
      labs(x = "Date", y = expression(Concentration ~ (mu*g / L)),
           shape = "Lake", fill = "Depth ID") +
      scale_fill_distiller(palette = "YlOrBr", guide = "colorbar",
                           direction = 1)
      #scale_fill_viridis_c(option = "viridis", begin = 0, end = 0.8, direction = -1)
  })

# Create a table that generates data for each point selected on the graph
   output$mytable <- renderTable({
     brush_out <- brushedPoints(filtered_nutrient_data(),
                                input$scatterplot_brush)
   })


}
```

Create the Shiny app object —- shinyApp(ui = ui, server = server)

Questions for coding challenge —- #1. Play with changing the options on the sidebar. # Choose a shinytheme that you like. The default here is "yeti" # How do you change the default settings? # How does each type of widget differ in its code and how it references the dataframe? #2. How is the mainPanel component of the UI structured? # How does the output appear based on this code? #3. Explore the reactive formatting within the server. # Which variables need to have reactive formatting? # How does this relate to selecting rows vs. columns from the original data frame? #4. Analyze the similarities and differences between ggplot code for a rendered vs. static plot. # Why are the aesthetics for x, y, fill, and shape formatted the way they are? # Note: the data frame has a "()" after it. This is necessary for reactive formatting. # Adjust the aesthetics, playing with different shapes, colors, fills, sizes, transparencies, etc. #5. Analyze the code used for the renderTable function. # Notice where each bit of code comes from in the UI and server. # Note: renderTable doesn't work well with dates. "sampledate" appears as # of days since 1970.