Water Quality App Instructions

Build an interactive application that uses water quality data collected by the DWQ buoys in several central Utah lakes.

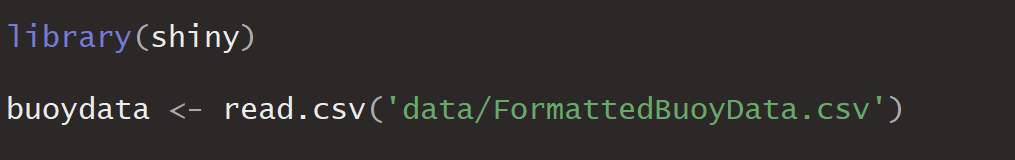
Download the WaterQualityApp folder. This folder contains the base files (ui.R, server.R, data folder, and R project file) that you will use to create an application in shiny.

Open the WaterQualityApp Project in RStudio.

Create a new script in the app/project directory called “global.R”. You can load any packages or data files in this file that can be used in your ui.R and server.R scripts. In the gloabal.R script, you will want to load the shiny package.

# Step 1 – Connect the data file (FormattedBuoyData.csv[[1]](#footnote-1)) to the app

Read in the data file by adding the following line of code after the line that loads the shiny package in the global.R script:



The dates will need to be reclassified from factors to dates. Use:

buoydata$DateTime <- as.Date(buoydata$DateTime,format="%m/%d/%Y")

to format this column (in the global.R script).

# Step 2 – Add a user input control widget for site selection:

Add the code allowing the user to select the site via a dropdown menu (selectInput) to the sidebarPanel (ui.R):

selectInput(inputId=”site”,

label=”Site”,

choices=unique(buoydata$SiteName),

selected = NULL,

multiple = FALSE,

selectize = TRUE,

width = NULL,

size = NULL)

Run your app. It should look something like this:

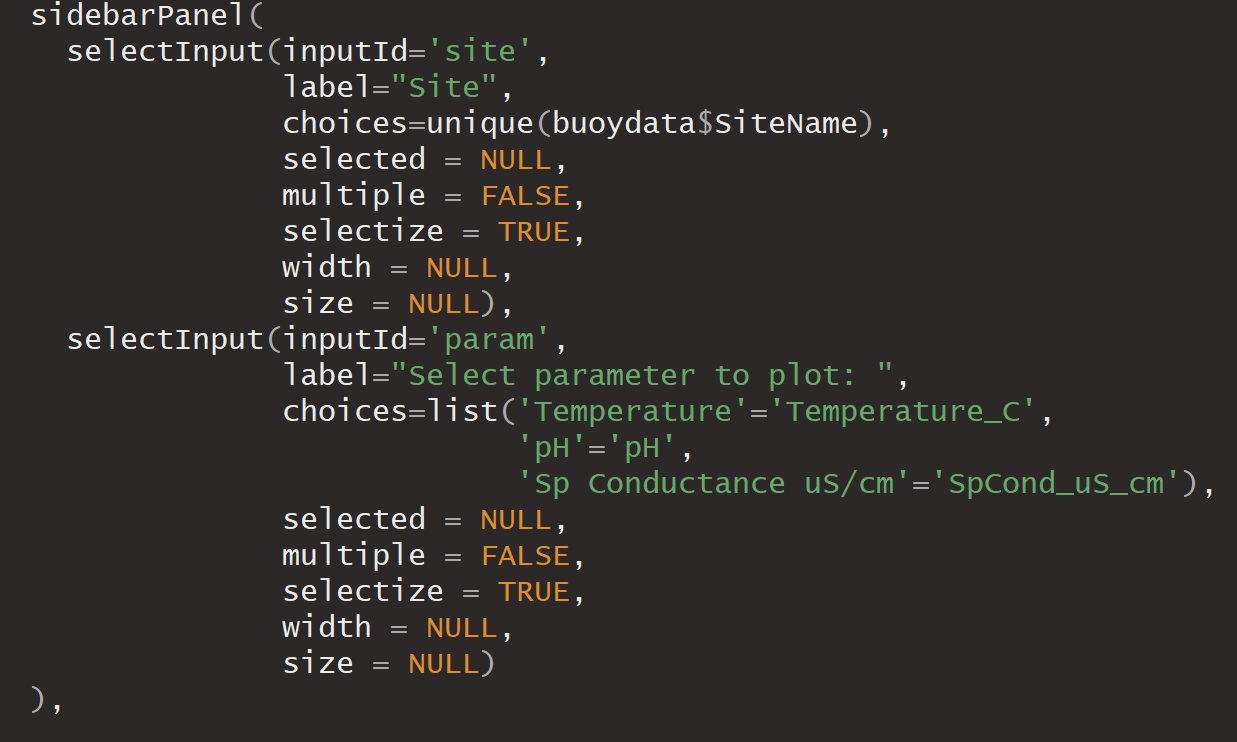


# Step 3 – Add user input control widget for the parameter selection

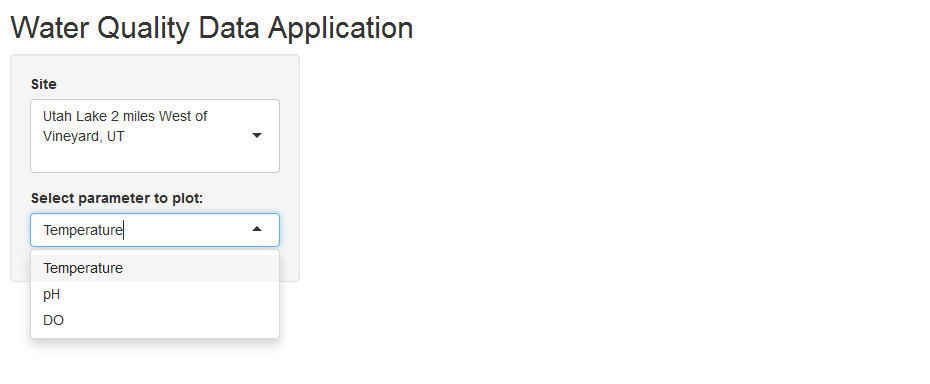
Add this code to the sidebarPanel, below the site selection widget created in Step 2. Options are provided for either A) a dropdown menu or B) radio buttons.

## Option A – Dropdown menu:

Provide a list of at least 3 water quality parameters (choosing from the parameters included in the buoydata dataframe). They do not need to be the same as those pictured. Code will be similar to the dropdown menu (selectInput) created in Step 2, except for the choices.

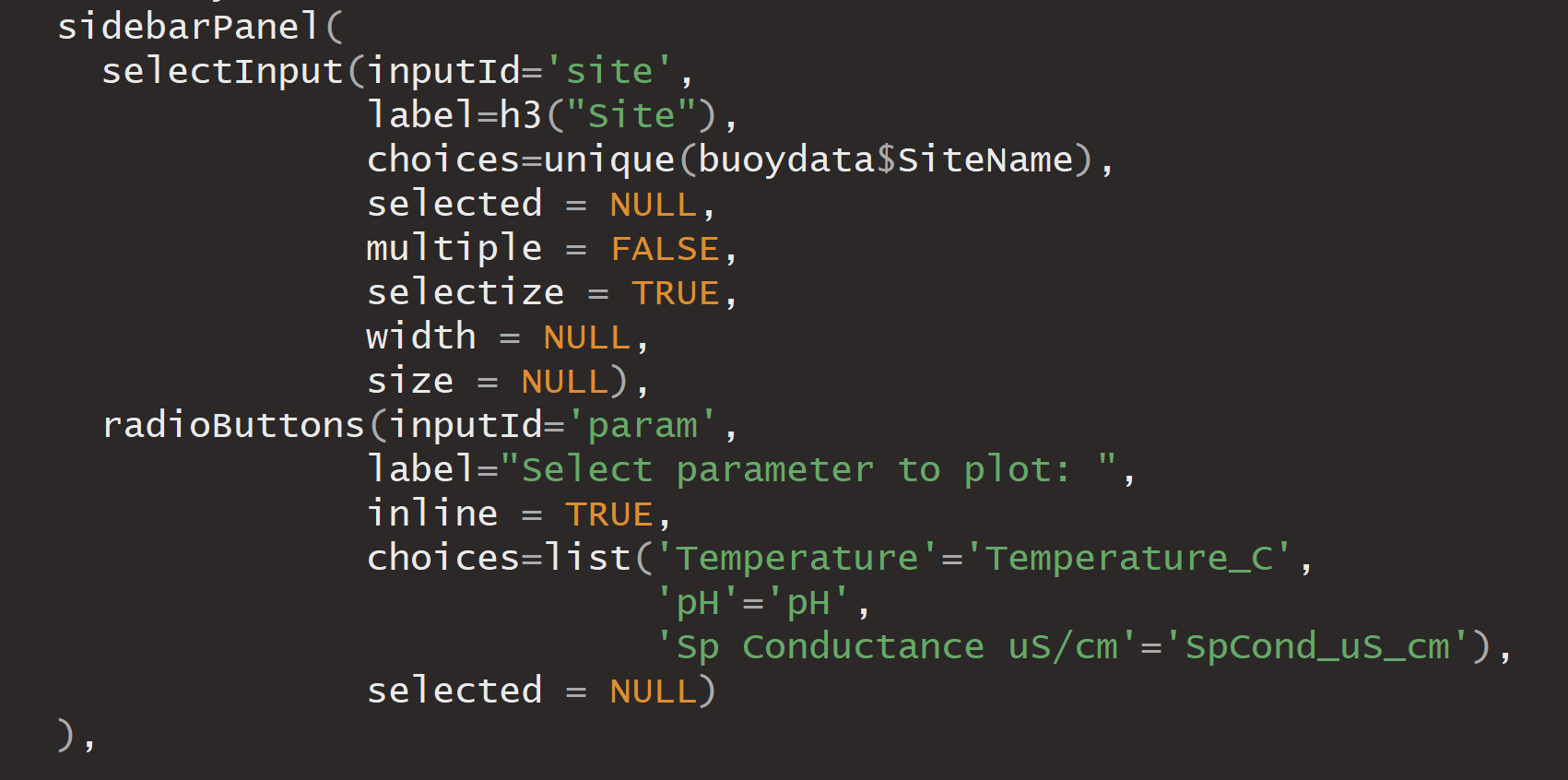


Run your app. It should look something like this (depending on which parameters you use):

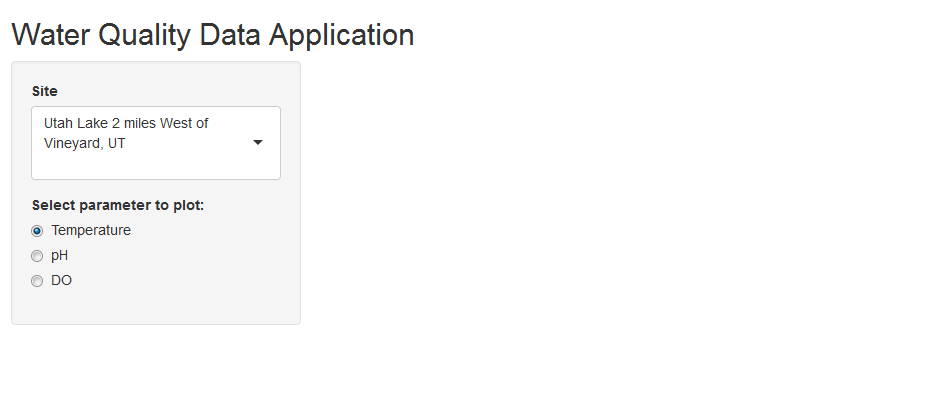


## Option B – Radio Buttons:

Provide a list of at least 3 parameters (choosing from the parameters included in the buoydata dataframe). They do not need to be the same as those pictured.



Run your app. It should look something like this (depending on which parameters you use):



# Step 4 – Add user input for date range

Go to <https://shiny.rstudio.com/gallery/widget-gallery.html>, and look at the example for the Date range widget. Click “See Code” and copy the line (up to but not including the comma) for dateRangeInput() in the ui.R tab. Paste this below the other user inputs from Step 2 and 3 in the ui.R script. Don’t forget to put a comma after the previous input widget!

Note: You’ll notice the label is formatted using an “h3” wrapper:

label = h3("Date range")

h3 indicates the label should be stylized as “header 3” style (think HTML header styles)[[2]](#footnote-2).

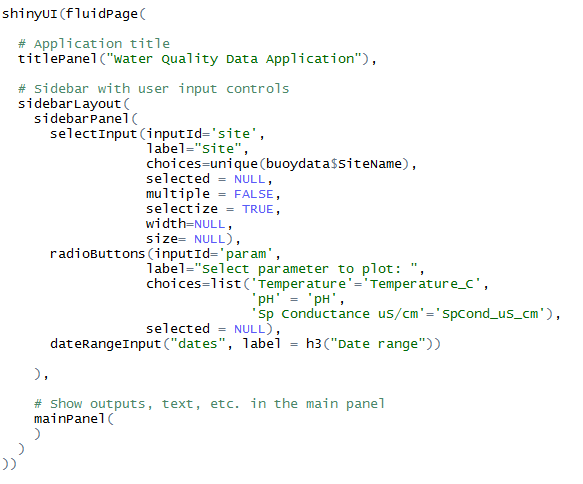
You can either remove this wrapper so that it reads:

dateRangeInput("dates", label = "Date range")

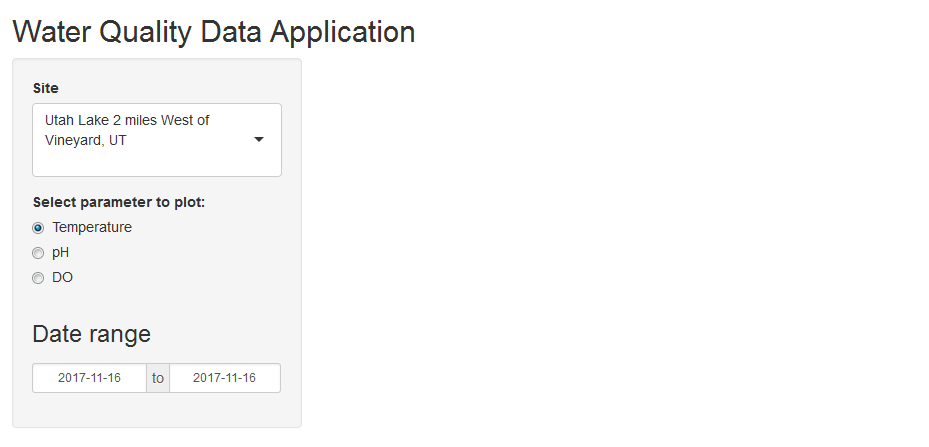
or you may wish to update the labels for the other inputs to match this style and make your app more uniform. You may want to include an argument that sets your default start date (the buoy record starts on August 26, 2016). Then, your code will look like:

dateRangeInput("dates", label = "Date range", start="2016-08-26")

Your ui.R code should look similar to the following (depending on which parameter input option you chose):



Run your app. It should look something like this (depending on which option you chose for the parameter input widget):



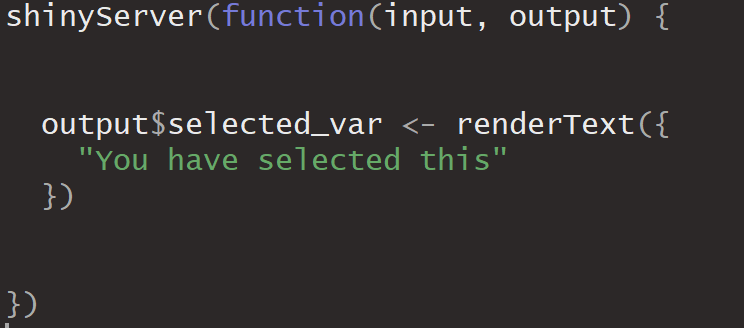
# Step 5 – Create a reactive text output

Go to <https://shiny.rstudio.com/tutorial/written-tutorial/lesson4/>.

Add a textOutput function in the mainPanel (following Step 1 of the example).



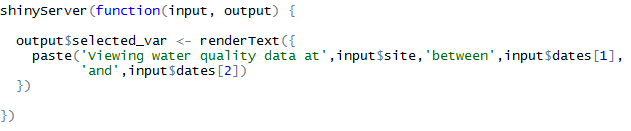
Continue following the example (Step 2 of the example) to create an object in the server.R script called “output$selected\_var”. This will be a renderText() function.



Replace the example text “You have selected this” text with the following:

paste(“Viewing water quality data at”,input$site,”between”,input$dates[1],”and”,input$dates[2])

Your code should now look like this:



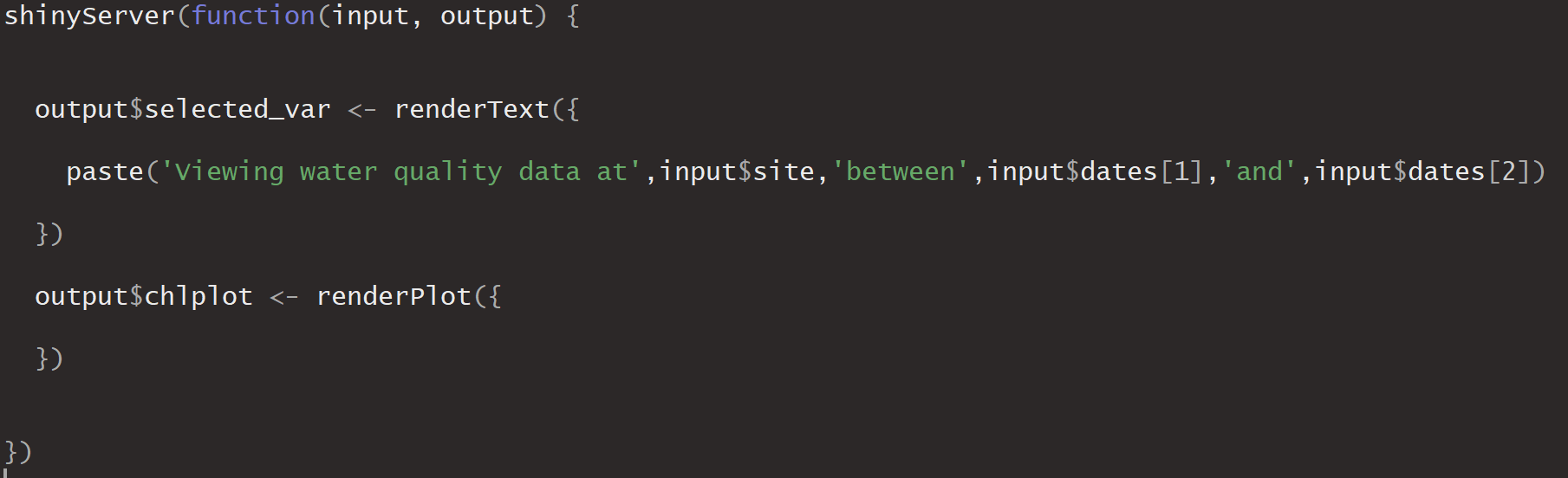
Run your app. Note that the text updates dynamically based on the user inputs.

# Step 6 – Add a Plot

Add a line to load the ggplot2 package in the global.R script.

## Option A - Relationships between chlorophyll and some other parameters:

Create another output, this time a renderPlot output, called “output$chlplot” after the renderText function. You do not need a comma to separate the output functions in the shinyServer function. This plot should show the relationship between a user-selected parameter and chlorophyll.



Subset the data to reflect the user inputs. Within the renderPlot function, add the following code:

plotdata <- subset(buoydata,SiteName==input$site &

DateTime >= input$dates[1] &

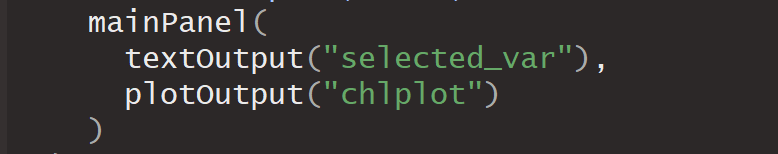
DateTime<= input$dates[2])

Then create a plot by adding the following code within the renderPlot function.

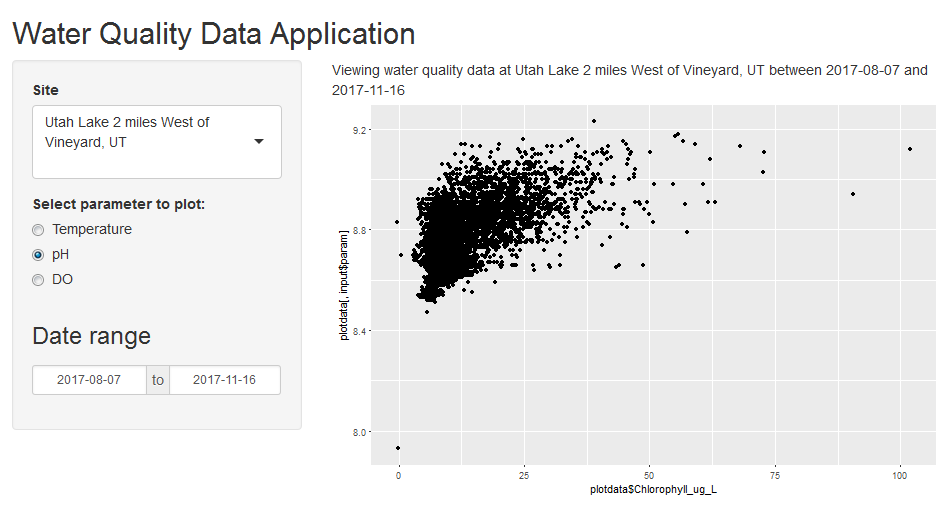
ggplot(data=plotdata,aes(x=plotdata$Chlorophyll\_ug\_L,y=plotdata[,input$param]))+geom\_point()

Customize the plot (add a title, labels, change color, change theme, etc.)

Add the plot object to the user interface, below the textOutput function in the mainPanel.

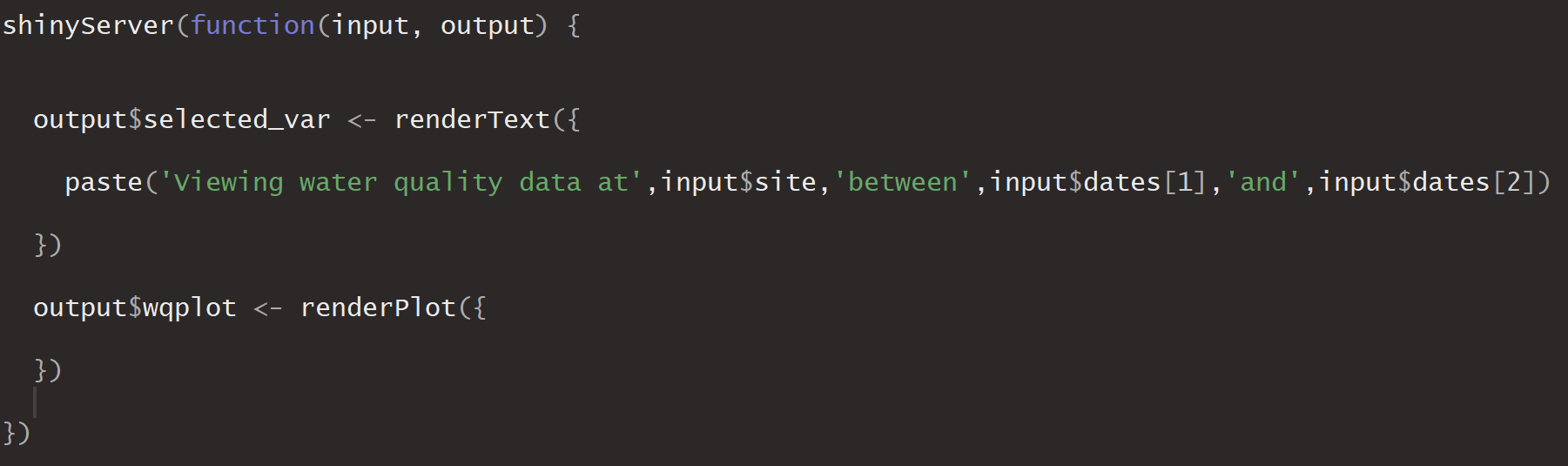


Run your app. Both the text and the plot should update based on your input. Note that the date range may need to be adjusted before any data show up in the plot. It should look something like this (depending on which option you chose for the parameter input widget ):



## Option B - Some parameter over Time:

Create another output, called “output$wqplot” after the text output function. You do not need a comma to separate the output functions in the shinyServer function. This plot should show the relationship between a user-selected parameter and time.



Subset the data to reflect the user inputs. Within the renderPlot function, add the following code:

plotdata <- subset(buoydata,SiteName==input$site &

DateTime >= input$dates[1] &

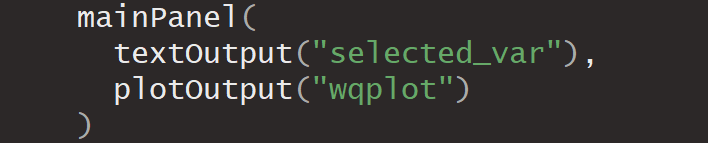
DateTime<= input$dates[2])

Then create a plot using ggplot().

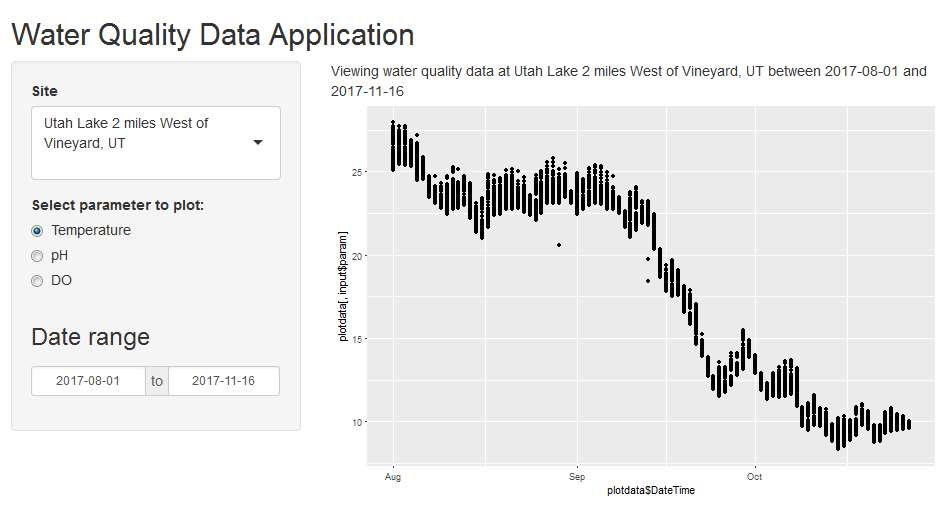
ggplot(data=plotdata,aes(x=plotdata$DateTime,y=plotdata[,input$param]))+ geom\_point()

Customize the plot (add a title, labels, change color, change theme, etc.)

Add the plot object to the user interface (ui.R), below the textOutput function in the mainPanel.



Run your app. Both the text and the plot should update based on your input. Note that the date range may need to be adjusted before any data show up in the plot. It should look something like this (depending on which option you chose for the parameter input widget ):



# Step 7 – Add a reactive function that performs a simple analysis

Create a reactive function within the shinyServer function that will create a simple linear model using lm() between chlorophyll and the selected parameter.

chlmod <- reactive({

plotdata <- subset(buoydata,SiteName==input$site &

DateTime >= input$dates[1] &

DateTime<= input$dates[2])

mod <- lm(plotdata$Chlorophyll\_ug\_L~plotdata[,input$param])

modsummary <- summary(mod)

return(modsummary)

})

Add an output within the shinyServer function that will display a result of this model.

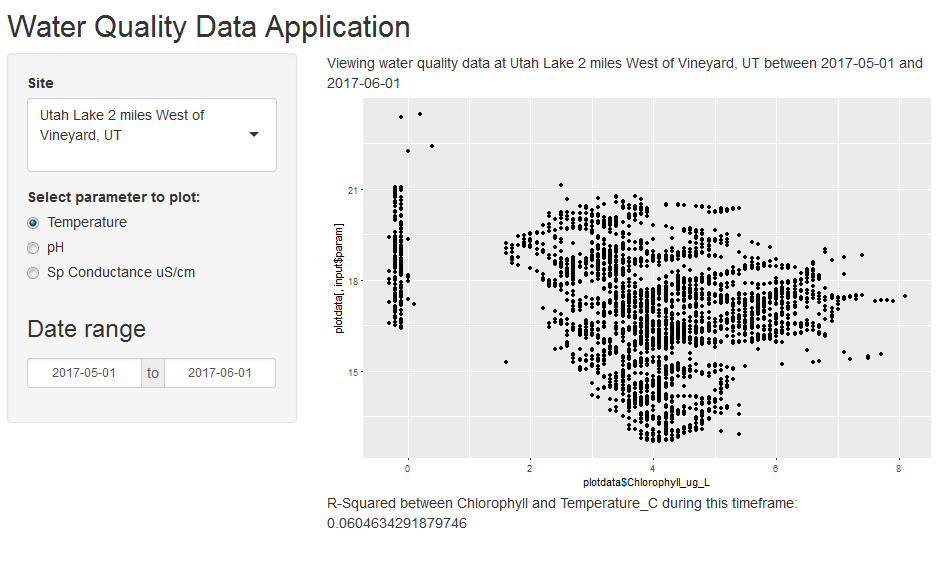
output$modelresults <- renderText({

paste("R-Squared between Chlorophyll and",input$param," during this timeframe:",chlmod()$r.squared)

})

Add a textOutput to the mainPanel (in the ui.R script) for the modelresults, similar to when you added the textOutput in step 5.

Run your app. Note that there will be an error displayed if no data exists for the selected date range. It should look something like this (depending on which option you chose for the parameter input widget ):



# Step 8 – Customize!

## Change the title of your app!

Change the text in the titlePanel object of the ui.R script.

## Use a different data source

* Instead of DWQ buoy data, use a different dataset. You could even use web services (like those in dataRetrieval or WaterML to download different parameters)

## Use different widgets:

* Check out other user inputs in the gallery for other user input widget options: <http://shiny.rstudio.com/gallery/widget-gallery.html>
* The shinyWidgets package has additional options for user inputs: <https://dreamrs.github.io/shinyWidgets/>

## Add text explaining what the app does/how to use it:

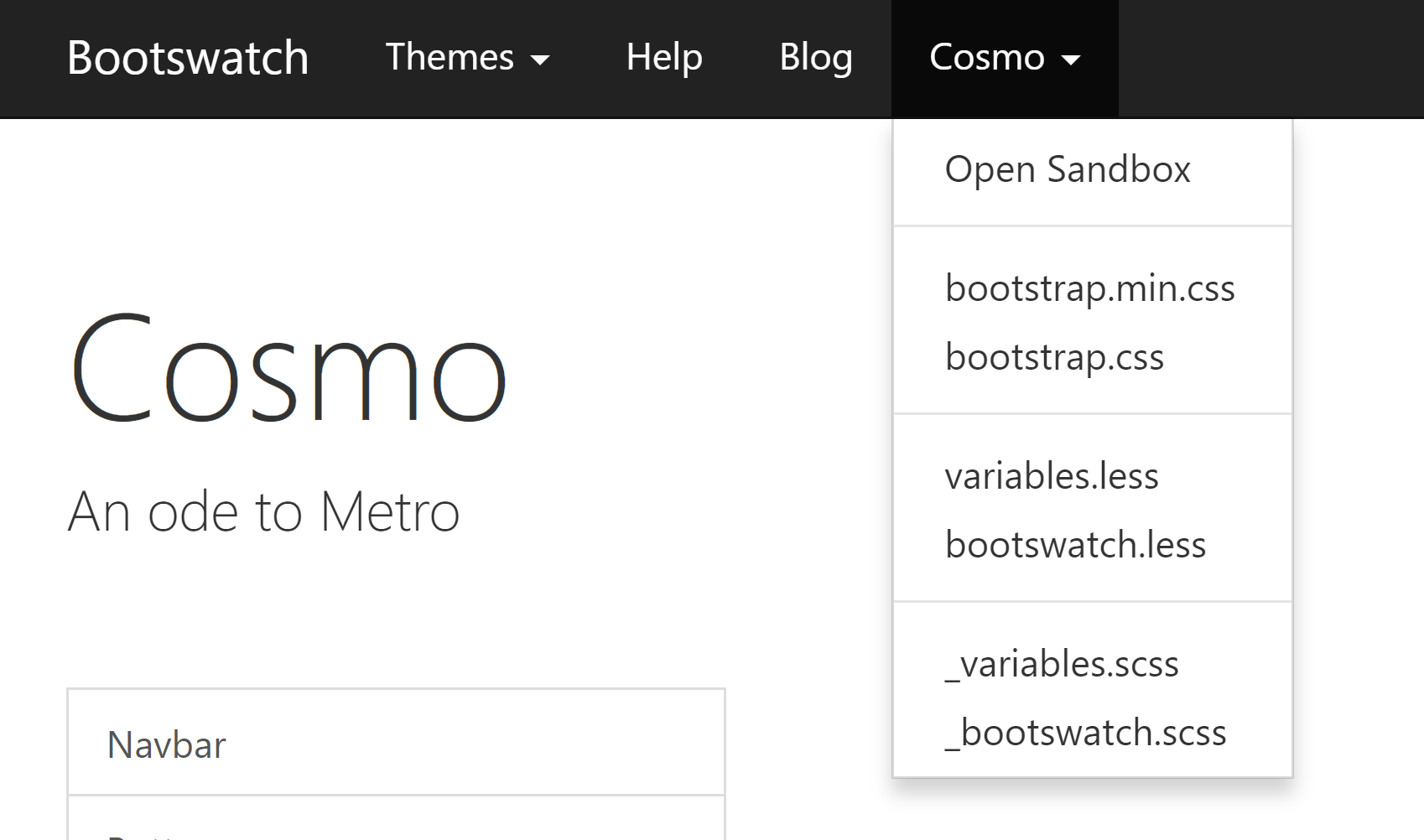
* Use the helpText widget to add (dynamic or non-dynamic) text to your interface: <https://shiny.rstudio.com/reference/shiny/0.11/helpText.html>
* or add text within a panel (see the “Formatted Text” section of <https://shiny.rstudio.com/tutorial/written-tutorial/lesson2/>)

## Add a map:

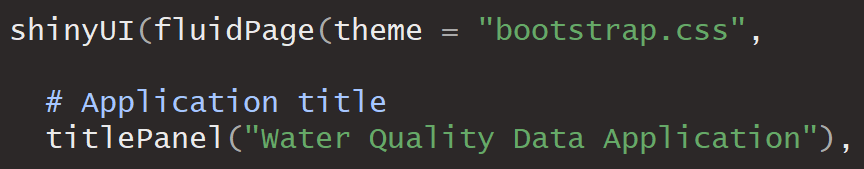
* Use ggmap or leaflet to add a map (static or interactive) to your shiny app
* <https://github.com/dkahle/ggmap>
* <https://rstudio.github.io/leaflet/shiny.html>

## Style your app:

* Add a logo/image: <https://stackoverflow.com/questions/21996887/embedding-image-in-shiny-app>
* Try a different layout: <https://shiny.rstudio.com/articles/layout-guide.html>
* Use html-style wrappers to style the text in your app. See the “HTML content” section of <https://shiny.rstudio.com/tutorial/written-tutorial/lesson2/>
* Use css to apply a “custom stylesheet” to your app. Create a subdirectory named “www” in your Shiny app directory. This subdirectory name www is special. Shiny makes every file in www available to your user’s browser. The www subdirectory is a great place to put CSS files, images, and other things a browser needs to build your Shiny App. Check out <https://bootswatch.com/> for free .css files that will change the look of your app (especially if you’ve styled it using the html style tags). Just browse the themes, then select the theme name > bootstrap.css.



This will open a text file in your browser. Save the file in your www folder (right click and save as ‘.css’ file in your www folder). Then, in your ui.R script, assign the theme within the fluidPage object.



## Other ideas:

* Try more advanced analysis (like k-means clustering <https://shiny.rstudio.com/gallery/kmeans-example.html>)
* Instead of choosing a single site, allow users to compare data across different sites (look into user input widgets that allow multiple selections).
* Create different kinds of plots.
* Look into the plotly package for an even more interactive application: <https://plot.ly/r/shiny-tutorial/>
* Display the data in an interactive data table: <https://shiny.rstudio.com/gallery/basic-datatable.html>
* Add other data sources – think about using data from other locations, or other methods (such as dataRetrieval or WaterML)

1. Additional information regarding units, parameters, ranges, equipment, etc. are included in the WQMetadata.txt file. [↑](#footnote-ref-1)
2. <https://shiny.rstudio.com/tutorial/written-tutorial/lesson2/>

   [↑](#footnote-ref-2)