

OSOS 2024

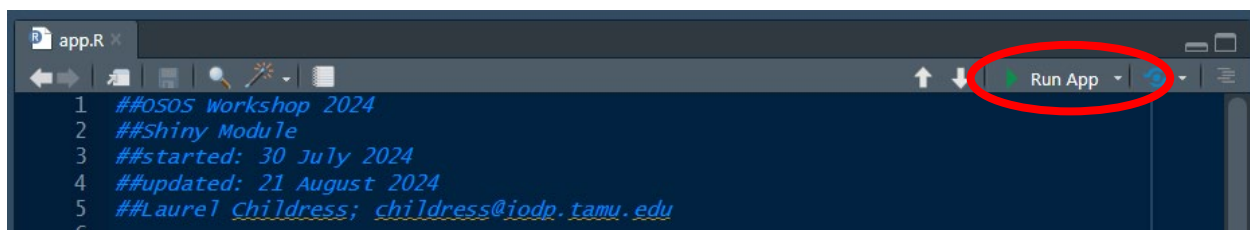
Shiny Module – Detailed Workflow

1) Introduction

We will start with the most basic components of a Shiny App.

When you open the file “app.R” you will see that most of it is currently commented out, save for a few rows.

To run any Shiny App locally, **click the ‘Run App’ button** in the upper right corner of the Source panel in RStudio.



When we do this a pop-up window appears. It is all white right now. Boring! But this is exactly what we have told the Shiny App to do so far.

Most of the commented lines are removed here so it is easier to see. Right now, this is all the Shiny App knows. The UI and the server are the two framework pieces of a Shiny App.

```
1 library(shiny) # load the shiny package
2
3 ui <- fluidPage(
4   ## How the app LOOKS
5   ## this is the UI (user interface) for your Shiny App; in this section
6   ## set the visualizations for the user, including elements such as user input
7 )
8
9 server <- function(input, output, session) {
10   ## How the app WORKS
11   ## this is the server side of your Shiny app. Here is where user input is
12   ## applied to data to produce the graph and table.
13 }
14
15 shinyApp(ui, server)
16 ## construct and start a Shiny application from UI and server
```

You may also notice in your Console panel that R is now “Listening on <http://127.0.0.1:4242>....” We are running the Shiny App locally for now (the easiest way to build and test before deploying) and this is your local host and port.

2) Data for our App - Earthquakes

For the module we will use the dataset 'quakes', which is part of The R Datasets Package (same as 'iris' or 'mtcars' which you might be familiar with).

Included in 'quakes' are the locations of 1000 seismic events of MB > 4.0. The events occurred in a cube near Fiji since 1964.

Type

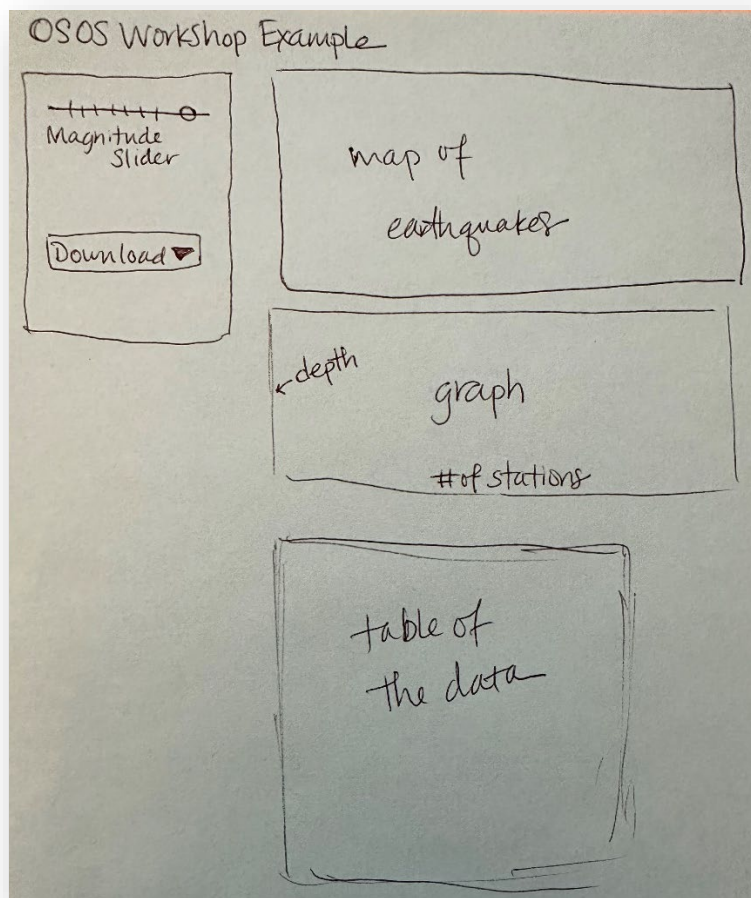
?quakes

in your Console panel and run it.

In the Help Panel information about the dataset will appear.

This data includes coordinate (latitude and longitude) information, which will allow us to make a map. There is also data on the magnitude of each earthquake, as well as the depth from which we can make a graph.

Knowing what is in the data, here is an idea for how we will make an app to present and share the data.



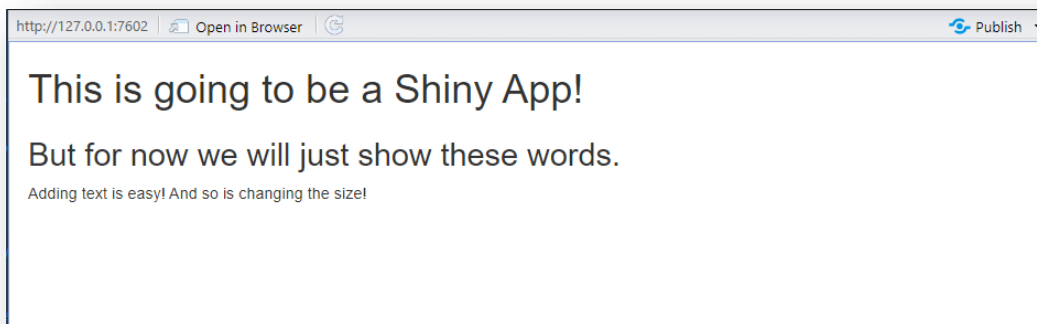
3) Basic Outline for our App

Uncomment lines 52 – 54 and click the ‘Run App’ button.

Tip: an easy way to comment/uncomment lines is to highlight the line(s) and use Control + Shift + C (Command + Shift + C for Mac).

Many other keyboard shortcuts here: <https://support.posit.co/hc/en-us/articles/200711853-Keybaord-Shortcuts-in-the-RStudio-IDE>

Note: if you did not Save this change to the app you will get a pop-up asking you to Save. After any change to an app be sure to Control + S (Command + S for Mac). Over time this will become second nature if you are working in a lot of Shiny Apps.



Text can be written directly; however I like to have more control over the font size and thus use h1 through h6 to control the relative font sizes in my app.

Uncomment line 35 and click the ‘Run App’ button.

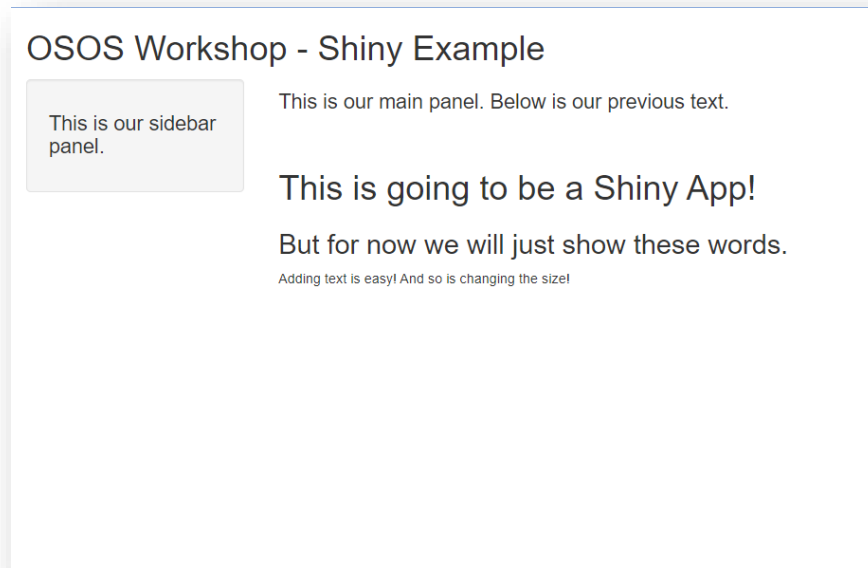
We have now added a title to our app. Without any other structure this just looks like more of the same text. So let’s add some additional framework elements to our app.

Uncomment lines

- 36 – 38 open and start our sidebar panel
- 46 close the sidebar panel
- 49 – 51 open and start our main panel
- 68 – 69 close the main panel and finish using sidebar layout function

and click the ‘Run App’ button.

Now it's really starting to look like something. We have a title for our app, a sidebar panel on the left side, and a main panel area.



4) Design the Sidebar

Next let's design what the sidebar will look like. I wanted to have a slider in there, where the user could filter which magnitude earthquakes to see on the map/graph/table. I also wanted a button where the user could download the data they select.

Uncomment lines 39 – 42 and click the 'Run App' button.

Currently, the app will throw an error.

The two most common 'typo' errors made in Shiny are unclosed parentheses (just like in R), and either missing or extra commas.

Shiny uses a lot of commas!

Looking at lines 39 – 42 we are both missing a comma and we have an extra comma.

We are missing a comma at the end of line 38.

We have an extra comma at the end of line 42. Line 42 currently does not need a comma because it is the final piece of the sidebar panel right now.

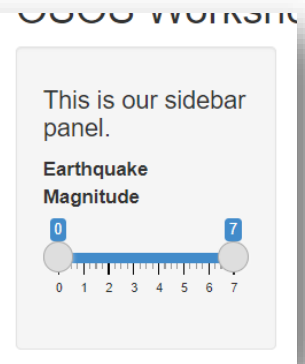
Add a comma at the end of line 38.

Delete the comma at the end of line 42.

Run the app.

We now have a slider in our sidebar, and we can control the range and the steps of the slider.

```
30 #####
31 ## How the app LOOKS
32 ## this is the ui (user interface) for your Shiny App; in this section
33 ## set the visualizations for the user, including elements such as user input
34 ui <- fluidPage(
35   titlePanel("OSOS Workshop - Shiny Example"), #page title
36   sidebarLayout( ## layout a side bar panel
37     sidebarPanel(width = 3, # space in Shiny is based on 12, so this is 1/4
38       h4("This is our sidebar panel."),
39       sliderInput("mag_Input", "Earthquake Magnitude",
40         min = 0, max = 7,
41         value = c(0, 7),
42         step = 0.5)
43     #
44     # downloadButton("downloadData", "Download Selected Data"),
45     #
```



I also wanted a button in the sidebar where the user could download the data. Let's add that next.

Uncomment lines 43 – 45 and click the 'Run App' button.

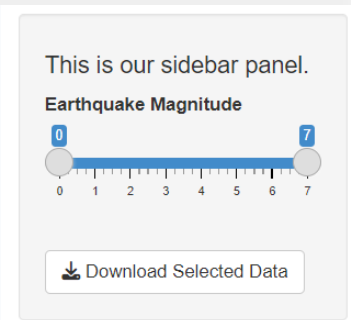
Same problem, we've got a comma issue!

Add a comma at the end of line 42.

Run the app.

We've now got a nice looking sidebar with a slider for user input and a way for the user to download the data.

```
33 ## set the visualizations for the user, including elements such as user input
34 ui <- fluidPage(
35   titlePanel("OSOS Workshop - Shiny Example"), #page title
36   sidebarLayout( ## layout a side bar panel
37     sidebarPanel(width = 3, # space in Shiny is based on 12, so this is 1/4
38       h4("This is our sidebar panel."),
39       sliderInput("mag_Input", "Earthquake Magnitude",
40         min = 0, max = 7,
41         value = c(0, 7),
42         step = 0.5),
43     br(),
44     downloadButton("downloadData", "Download Selected Data"),
45     br()
46   ), #close the sidebar
47   #####
48   ##
```



5) Design the Main Panel

Next let's design what the main panel will look like. I wanted a map of the earthquakes, a graph, and a table of the data.

Uncomment lines 63 – 67 and click the 'Run App' button.

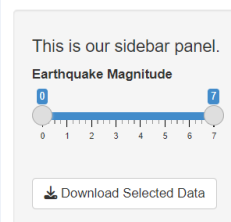
Same problem, we've got a comma issue! Yes, again. This is something that will occur a lot, at least for me, when initially writing apps in Shiny so hopefully this helps you get used to looking for this issue. RStudio is fairly good to give you a red-x warning about the problem, but if you have a big/long app then that may be off your current screen.

Add a comma at the end of line 54.

Run the app.

```
47 ##### 2) Layout the sidebar #####
48 ## Layout the sidebar
49 sidebarPanel(
50   h4("This is our sidebar panel. Below is our previous text."),
51   br(), #this adds space between the lines of text
52   h2("This is going to be a Shiny App!"),
53   h3("But for now we will just show these words."),
54   h6("Adding text is easy! And so is changing the size!"),
55 ##### 3) Layout the main panel #####
56 # fluidRow() #alternate version
57 #   column(width = 5,
58 #     leafletOutput("mymap"),
59 #     column(width = 7,
60 #       plotOutput("coolplot"),
61 #     ),
62 #   )
63 leafletOutput("mymap"), # a map of the data
64 br(), # a break to put space below the map
65 plotOutput("coolplot"), #a graph of the data
66 br(), br(), # two breaks for better visual
67 tableOutput("results") #a table of the users selection
68 ) #close the main panel
69 ) #close the side bar layout
70 ) #close the ui
```

OSOS Workshop - Shiny Example



This is our main panel. Below is our previous text.

This is going to be a Shiny App!

But for now we will just show these words.

Adding text is easy! And so is changing the size!

Everything looks – the same? Yes.

We have defined in the UI where we want the map, graph, and table to appear and have defined those variables but we have not yet given the app any information on what data to use, how to make the map, how to make the graph, or how to make the table.

So let's do that!

6) Build the Server Side Functions

a. Define what data to use

We want to use the quakes data set. We also want the user to manipulate which magnitude earthquakes are shown in the map/graph/table using the slider.

Uncomment lines 78 – 83 and click the ‘Run App’ button.

It still looks the same! Yes. At this point we have only defined what data to use. We have not defined the map, graph, or table variables.

Type

`head(quakes)`

in your Console panel and run it.

```
76 # read the user input and subset the dataset
77 ## read the user input and subset the dataset
78 quake_select <- reactive({ # subset will change based on user input
79   quakes %>% # the quakes dataset
80     filter(mag >= input$mag_Input[1], # limit the data to user choices
81           mag <= input$mag_Input[2]
82   )
83 })
84
```

However, we have used the input from the user. Remember that we defined the variable for the slider input.

```
39 sliderInput("mag_Input", "Earthquake Magnitude",
40             min = 0, max = 7,
41             value = c(0, 7),
42             step = 0.5),
```

b. Define what the map looks like

Next, let's define what the map looks like. This will finally change how the app looks!

Uncomment lines 86 – 95.

```

84 ▾ #####
85 #generate a map based on the user selection
86 ▾ output$mymap <- renderLeaflet({
87   leaflet(quake_select()) %>%
88     addTiles() %>% #related to our background, can allow us to label
89     addCircles(lng = ~long, lat = ~lat, #add our exp points
90       popup = paste("Magnitude:", quake_select()$mag, "<br>",
91         "Depth (km):", quake_select()$depth), #generate a pop-up on click
92       weight = 10, radius = ~ sqrt(mag) #size of the circles
93     ) %>%
94     addProviderTiles("Esri.WorldImagery") #nice looking background
95 ▸ })
96 ▾ #####

```


Run the app.

Finally, our app is starting to look like something!

OSOS Workshop - Shiny Example

This is our sidebar panel.

Earthquake Magnitude




Download Selected Data

This is our main panel. Below is our previous text.

This is going to be a Shiny App!

But for now we will just show these words.

Adding text is easy! And so is changing the size!



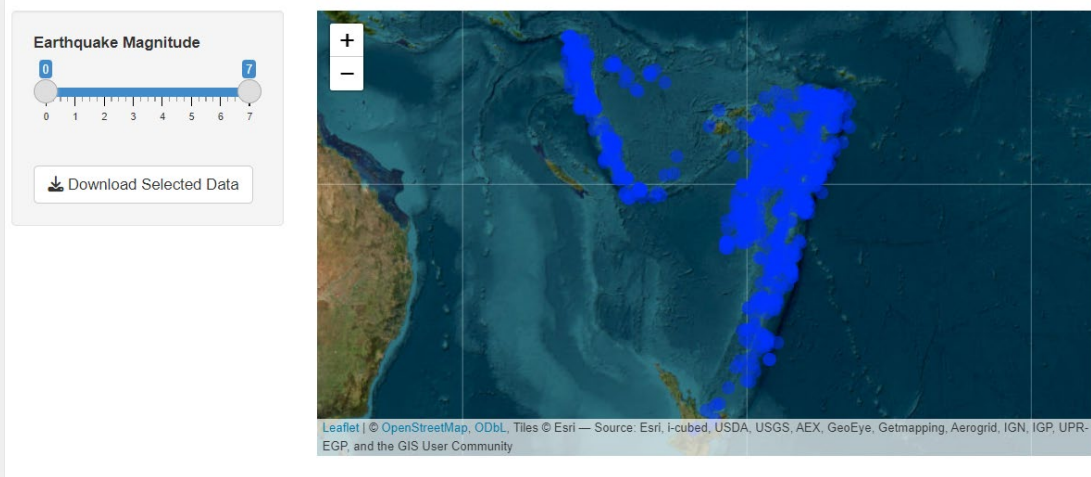
Leaflet | © OpenStreetMap, ODbL, Tiles © Esri — Source: Esri, i-cubed, USDA, USGS, AEX, GeoEye, Getmapping, Aerogrid, IGN, IGP, UPR-EGP, and the GIS User Community

Now that we are adding real components, let's clean up some of that example text we started with.

Comment out lines 38 and 50 - 54 and click the 'Run App' button.

Much nicer!

OSOS Workshop - Shiny Example



The dots are hard to see in the default blue, since the ocean is also blue. And while the size of the circles is proportional to the magnitude, it would be nice if there was also a color gradient.

First we can define our color palette.

Uncomment lines 26 – 28.

Then we will use this color palette in our map.

After line 92 add:

color = ~ pal(mag) #color of the circles

Run the app.

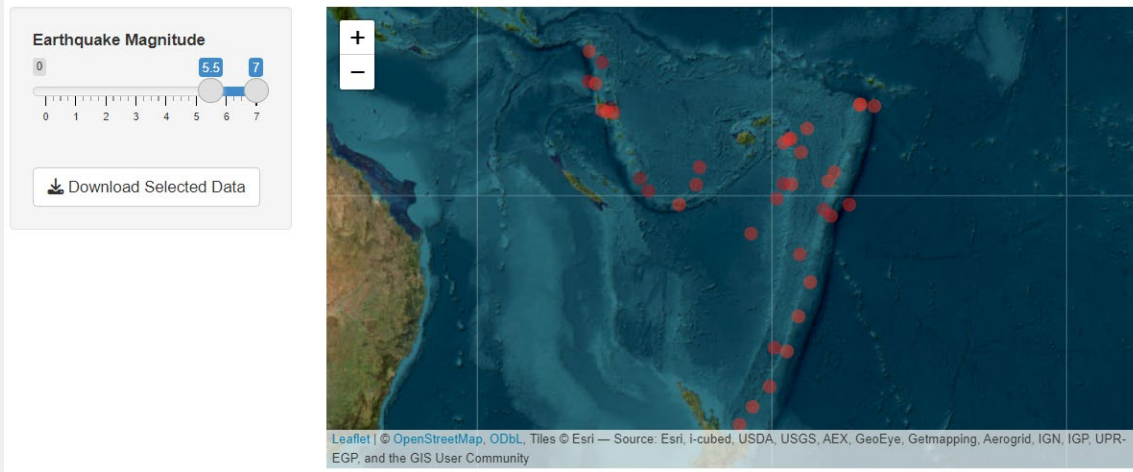
Those pesky commas again!

Add a comma at the end of line 92.

Run the app.

Beautiful! Play around with the slider and you will notice that the number of points on the map will change.

OSOS Workshop - Shiny Example



It would also be nice if the map had a legend, since we are scaling the color based on the magnitude. Let's add a legend.

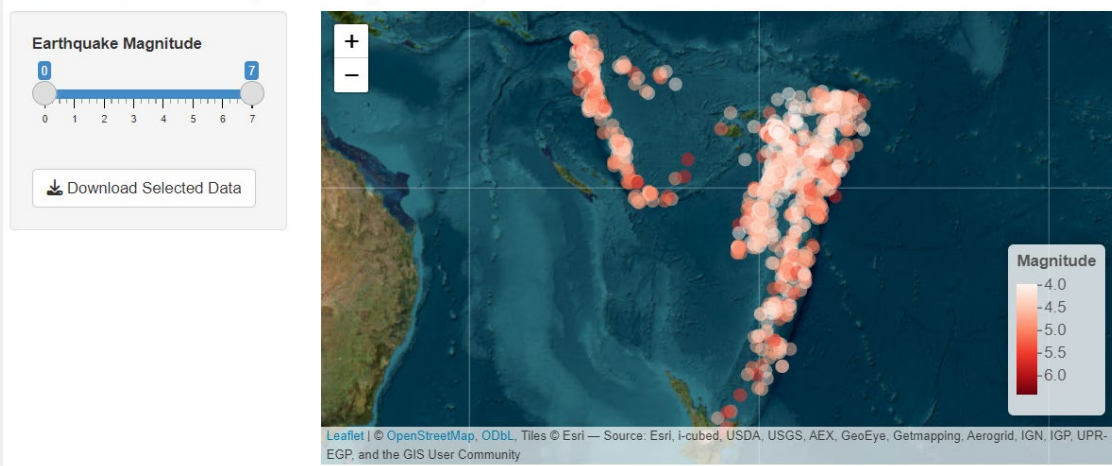
After line 94 add:

```
addLegend("bottomright", pal = pal, values = ~mag,  
title = "Magnitude",  
opacity = 1) %>%
```

Run the app.

The map now looks great, and we can see it is responsive to the user input.

OSOS Workshop - Shiny Example



c. Define what the graph looks like

Next, let's define what the graph looks like.

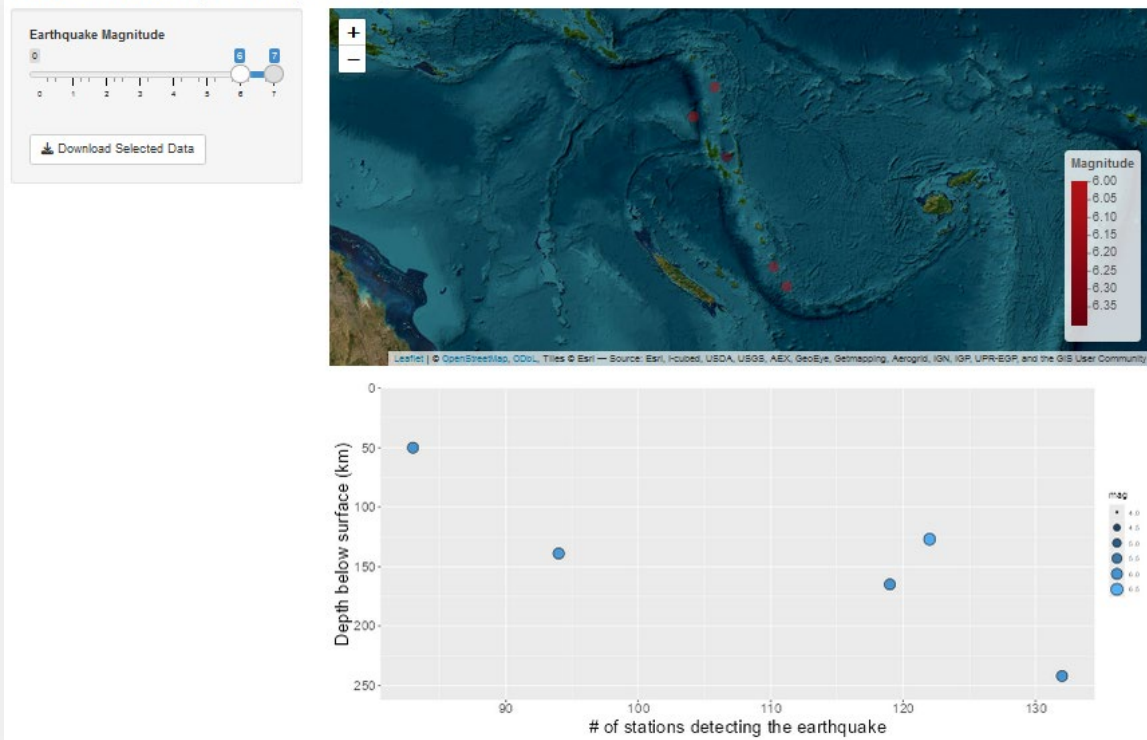
Uncomment lines 102 – 113.

```
100 ▾ #####
101 ## generate the graph of the data based on the user selection
102 ▾ output$coolplot <- renderPlot({
103   ggplot() +
104     geom_point(data = quake_select(), aes(y = depth, x = stations,
105     size = mag, fill = mag), shape = 21) +
106     labs(x = "# of stations detecting the earthquake", y = "Depth below surface (km)") +
107     scale_fill_continuous(limits=c(4, 6.5), breaks=seq(4, 6.5, by = 0.5)) +
108     scale_size_continuous(limits=c(4, 6.5), breaks=seq(4, 6.5, by = 0.5)) +
109     scale_y_reverse(expand = c(0, 0), limits = c(max(quake_select()$depth) + 20,0)) +
110     theme(axis.text = element_text(size = 15),
111           axis.title = element_text(size = 20)) +
112     guides(color = guide_legend(), fill = guide_legend())
113 ▴ })
114 ▾ #####
```

Run the app.

Drag the slider around and both the map and graph will change the points displayed.

OSOS Workshop - Shiny Example



d. Define what the table looks like

Next, let's define what the table looks like.

Uncomment lines 116 – 120.

```
114 ▾ #####
115   ## generate the table of the data based on the user selection
116 ▾ output$results <- renderTable({
117   quake_select()
118 ▲   },
119   hover = TRUE, bordered = TRUE
120 )
121 ▾ #####
```

Run the app.

Try changing 'bordered' or 'hover' to FALSE and see how this updates the table.

This is the most basic version of table in Shiny. Using the 'DT' package, you can generate far more elegant and interactive tables (pagination, searching, sortable columns) for your app.

Using that format, our table could look something like this:

Show	10 ▾	entries	Search: <input type="text"/>		
Latitude ▴ ▾	Longitude ▴ ▾	Depth (km) ▴ ▾	Magnitude ▴ ▾	# of Stations ▴ ▾	
-20.4200	181.6200	562	4.8	41	
-20.6200	181.0300	650	4.2	15	
-26.0000	184.1000	42	5.4	43	
-17.9700	181.6600	626	4.1	19	
-20.4200	181.9600	649	4	11	
-19.6800	184.3100	195	4	12	
-11.7000	166.1000	82	4.8	43	
-28.1100	181.9300	194	4.4	15	
-28.7400	181.7400	211	4.7	35	
-17.4700	179.5900	622	4.3	19	
Showing 1 to 10 of 1,000 entries			Previous	1	2 3 4 5 ... 100 Next

e. Define the user downloadable csv of the data

Finally, let's define the downloadable csv file for our users.

Uncomment lines 123 – 130.

```
121. #####
122. ## generate a downloadable csv based on the user selection
123. output$downloadData <- downloadHandler(
124.   filename = function() { #generate a default filename
125.     paste("Quakes_filtered", ".csv", sep = "")
126.   },
127.   content = function(file) { #generate the data file
128.     write.csv(quake_select(), file, row.names = FALSE)
129.   }
130. )
```

Run the app.

Download the file and save it somewhere easy, like your desktop.

[This *may* not work locally on a Mac, but will work when the app is implemented in the server online.]

Our app is now complete!

f. Publishing the app

This is the final step for any Shiny app – making it available to others.

If using shinyapps.io then publishing can be accomplished directly from RStudio by setting up a connection to your shinyapps.io account.

Once you have established the connection between your RStudio and shinyapps.io using the 'rsconnect' package, publishing is as easy as pressing the 'Publish' button.

Setting up everyone's account is more than we have time to work through today. While it takes a little time to set-up, the process is quite easy. <https://shiny.posit.co/r/articles/share/shinyapps/>

shinyapps.io is free for up to 5 applications and 25 active hours per month, so I also don't want to waste any of your hours with the example app from this module.