

# R SHINY WORKSHOP

16<sup>TH</sup> OCTOBER 2020

*given by Andy McKenzie*

# What we'll cover

- 1) Setting up a R Shiny app and how the parts of it interact.
- 2) Controlling the layout of the app (i.e. how it looks).
- 3) Adding control widgets (e.g. selector boxes, sliders).
- 4) Putting a R Shiny on the web.
- 5) Next steps

# Initial setup

Make a directory called “RShiny workshop”

Then a sub-directory to this called “workshop apps”

RShiny workshop  
    workshop apps

# Geyser data set

In the R console

```
# data sets available in R  
data()
```

```
# geyser data set  
data("faithful")
```

```
# find out more about this data  
help("faithful")
```

# Geyser data set

# view the data

View(faithful)

	eruptions	waiting
1	3.600	79
2	1.800	54
3	3.333	74
4	2.283	62
5	4.533	85
6	2.883	55
7	4.700	88
8	3.600	85
9	1.950	51
10	4.350	85

eruptions = eruption duration (minutes)

waiting = time to next eruption (minutes)

# Geyser data set: histogram of time until next eruption

Make a new R script file, as below, and save it to the “RShiny workshop” directory.  
Call it something like “introductory script.R”

```
RShiny workshop
  workshop apps
    introductory script.R
```

```
num.bins <- 20
```

```
x <- faithful[, 2]
```

```
min(x)
```

```
max(x)
```

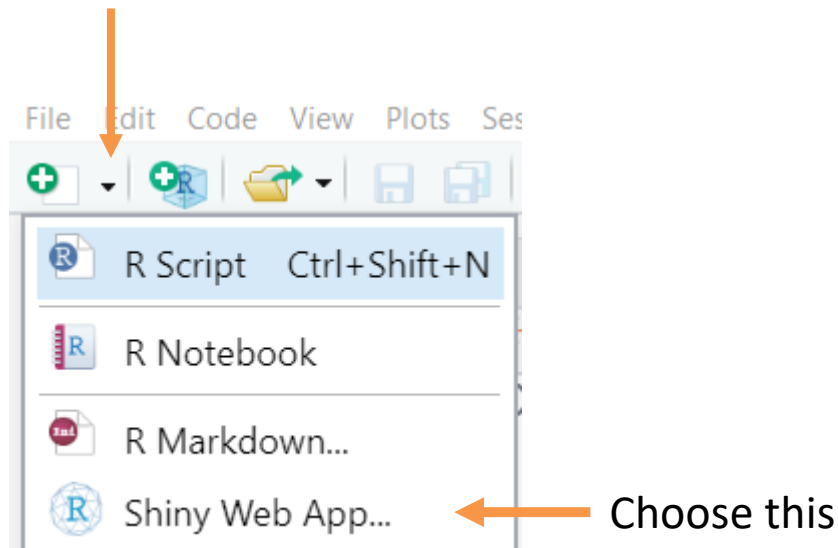
```
bins <- seq(min(x), max(x), length.out = num.bins + 1)
```

```
hist(x, breaks = bins, col = 'darkgray', border = 'white')
```

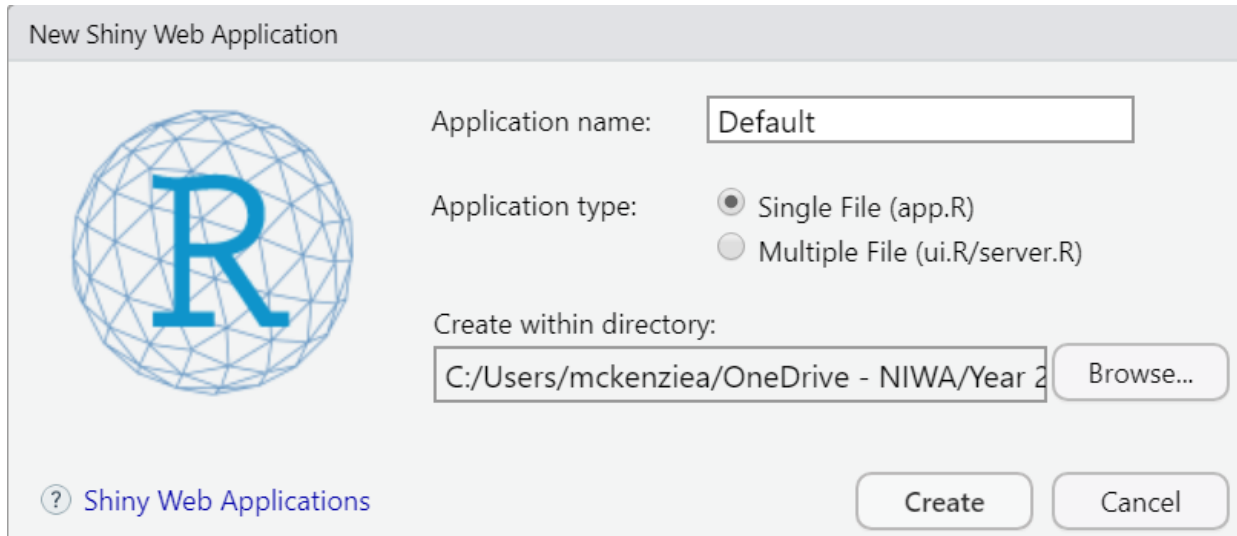
# Your first R Shiny app!

We'll set up the default demo Shiny app

Click on the little black triangle to bring up different script options



# Your first R Shiny app!



New Shiny Web Application

Application name:

Application type: ☒ Single File (app.R)  
☐ Multiple File (ui.R/server.R)

Create within directory:

[? Shiny Web Applications](#)

- (1) For the Application name put: Default
- (2) Leave the **Application type** as Single File (app.R)
- (3) For **Create within directory** browse to choose “workshop apps”
- (4) Click Create





# Your first R Shiny app!

This will

(1) make a sub-directory called **Default**

(2) and in this directory will be app.R (code for the app)

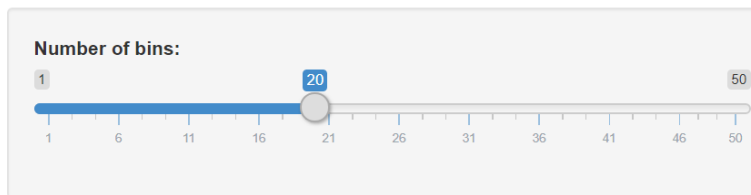
Year 2021 > RShiny workshop > workshop apps > Default		
^	Name	Status
	 app.R	

# app.R (running it)

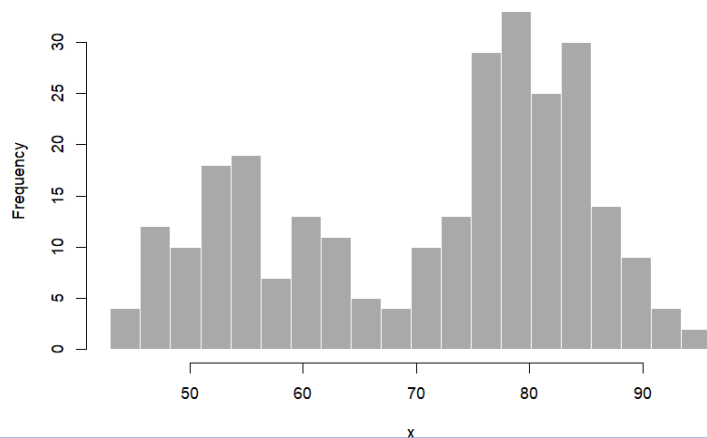
If someone has the shiny library installed, they can run this code to get the app going. Try running it line-by-line.

<http://127.0.0.1:13749> | [Open in Browser](#) | [Pub](#)

## Old Faithful Geyser Data

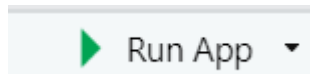


Histogram of x



# app.R (running it)

Or even easier, click on the Run App icon (top-right of editor window).



This is the same as typing in the console window (if your working directory is **RShiny workshop**):

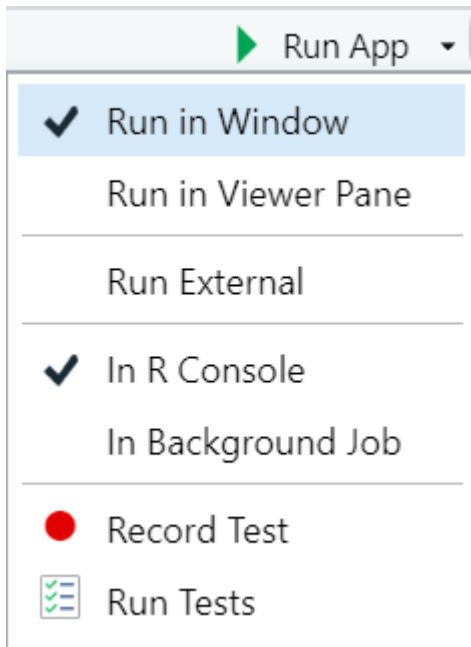
```
runApp('workshop apps/Default')
```

Or if the working directory is **workshop apps** then:

```
runApp('Default')
```

# app.R (running it)

Can set other options for what happens when you click on Run App:



One of the tabs in bottom-right pane of RStudio

Turns up in your default browser (I prefer this)

Runs app from console (so you can't use the console)

Runs in background (CAN use the console)

# app.R (the components)

# functions to construct and run shiny apps

```
library(shiny)
```

# user interface (what you see when you run the app)

```
ui <- ...
```

# calculations and plots (the usual R code)

```
server <-
```

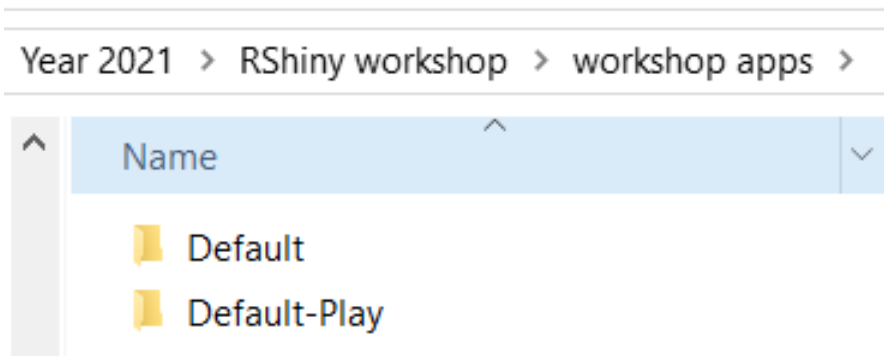
# Bring together the user interface and server, and run the application

```
shinyApp(ui = ui, server = server)
```

# app.R (play time)

In the “workshop apps” directory, make a new shiny app called “Default-Play”

Some procedure as before, but make the Application Name: Default-Play



# app.R (play time)

For your Default-Play app

- (i) change the starting value for the number of bins from 30 to 20
- (ii) change the colour of the histograms to purple (yikes!).
- (iii) have a general muck around with changing the code, and the way the app is run.

Check your changes by running the app often.

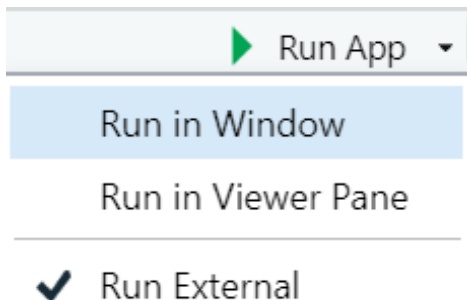
# app.R (display mode)

## Display mode

See the code and running app at the same time by typing in the console: `runApp(display.mode = "showcase")`

Click on the “show with app” icon to display the code next to the running app.

Opening in an external browser is a good choice in display mode





# app.R (“live” editing version one)

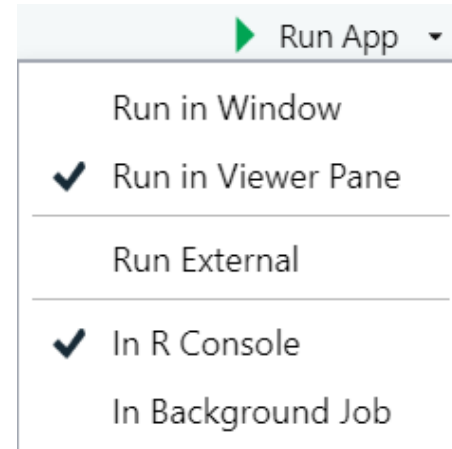
## “Live” editing

(1) Turn on “Run in View Pane”

(2) Run the app

(3) Make edits to the app

(4) Click on  This will save then run the app file.

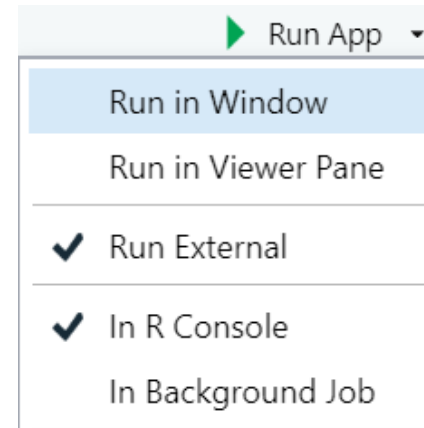


# app.R (“live” editing version two)

(1) Turn on “Run External”

(2) Run the app

(3) Make edits to the app



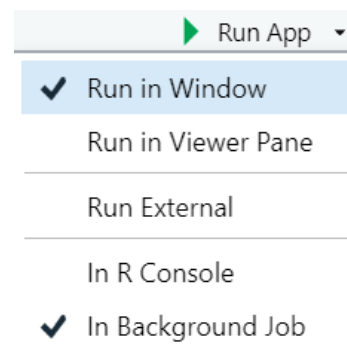
(4) Save the app file, then update the external browser window to see the changes

# app.R (background jobs)

(1) Turn on “In Background Job”

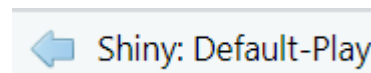
(2) Either “Run in Viewer Pane”/  
“Run External”

(3) Run app



Bottom-left pane will move to “Jobs” tab. Console tab is still free to run commands in.

To see the background jobs running click on



Click on stop in the “Jobs” tab to stop an app (closing a window will leave a job still running).

# app.R (the components)

# functions to construct and run shiny apps

```
library(shiny)
```

# user interface (what you see when you run the app)

```
ui <- fluidpage(
```

```
)
```

# calculations and plots (the usual R code)

```
server <- function(input, output) {
```

```
}
```

# Bring together the user interface and server, and run the application

```
shinyApp(ui = ui, server = server)
```

# app.R (user interface)

```
# Define UI for application that draws a histogram
ui <- fluidPage(
```

```
  # Application title
```

```
  titlePanel("Old Faithful Geyser Data"),
```

```
  # Sidebar with a slider input for number of bins
```

```
  sidebarLayout(
    sidebarPanel(
      sliderInput("bins",
        "Number of bins:",
        min = 1,
        max = 50,
        value = 30)
```

```
    ),
```

```
    # Show a plot of the generated distribution
```

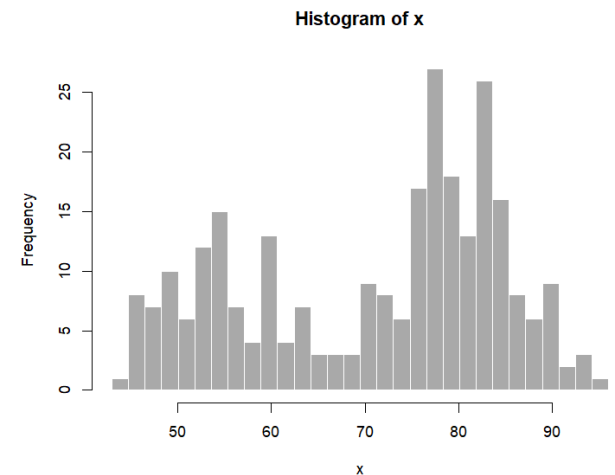
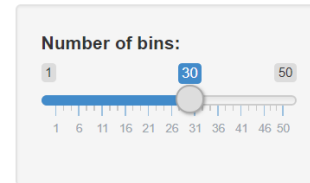
```
    mainPanel(
      plotOutput("distPlot")
```

```
    )
```

```
  )
```

```
)
```

Old Faithful Geyser Data



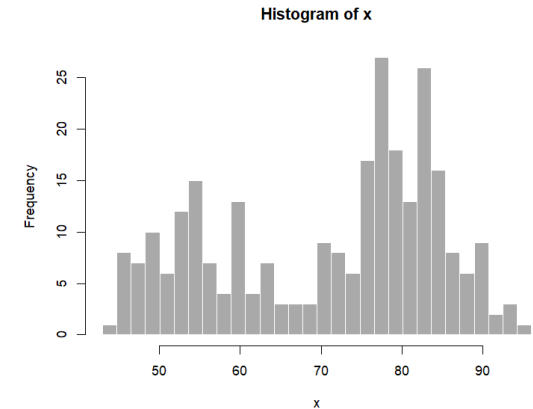
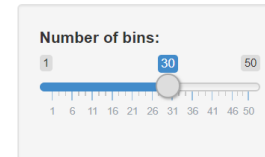
# app.R (server)

```
# Define server logic required to draw a histogram
server <- function(input, output) {
```

```
  output$distPlot <- renderPlot({
    # generate bins based on input$bins from ui.R
    x <- faithful[, 2]
    bins <- seq(min(x), max(x), length.out = input$bins + 1)

    # draw the histogram with the specified number of bins
    hist(x, breaks = bins, col = 'red', border = 'white')
  })
}
```

Old Faithful Geyser Data



# app.R (reactivity)

**Reactivity.** Change the slider value in the user interface (ui) and the plot in the server section will update, and what is displayed in the user interface is updated.

Reactivity and linking the user interface and server is done via **input/output** variables in the code

**# user interface (what you see when you run the app)**

```
ui <- fluidpage(
```

```
)
```

**# calculations and plots (the usual R code)**

```
server <- function(input, output) {
```

```
}
```

# app.R (reactivity)

## Change the slider input value

# user interface (what you see when you run the app)

```
ui <- fluidpage(
```

```
  sliderInput("bins", → input$bins
```

```
    "Number of bins:",
```

```
    min = 1,
```

```
    max = 50,
```

```
    value = 20)
```

```
)
```

# calculations and plots (the usual R code)

```
server <- function(input, output) {
```

```
  bins <- seq(min(x), max(x), length.out = input$bins + 1)
```

```
}
```

**sliderInput** is a reactive input function. If the slider is moved then the `input$bins` value is updated.

There's a whole lot of reactive input functions with names:

**XXXXXInput**



# app.R (reactivity)

## An updated plot to show in the user interface

# user interface (what you see when you run the app)

```
ui <- fluidpage(
```

```
  mainPanel(
```

```
    plotOutput("distPlot")
```

```
  )
```

```
)
```

# calculations and plots (the usual R code)

```
server <- function(input, output) {
```

```
  output$distPlot <- renderPlot({
```

```
    # draw the histogram with the specified number of bins
```

```
    hist(x, breaks = bins, col = 'blue', border = 'white')
```

```
  })
```

```
}
```

**renderPlot** is a reactive function. If the plot inside the function changes then the user interface display is updated.

There's a whole lot of reactive output functions with names:

**renderXXXXX**