Programming with R, Week 7

Denis Vrdoljak

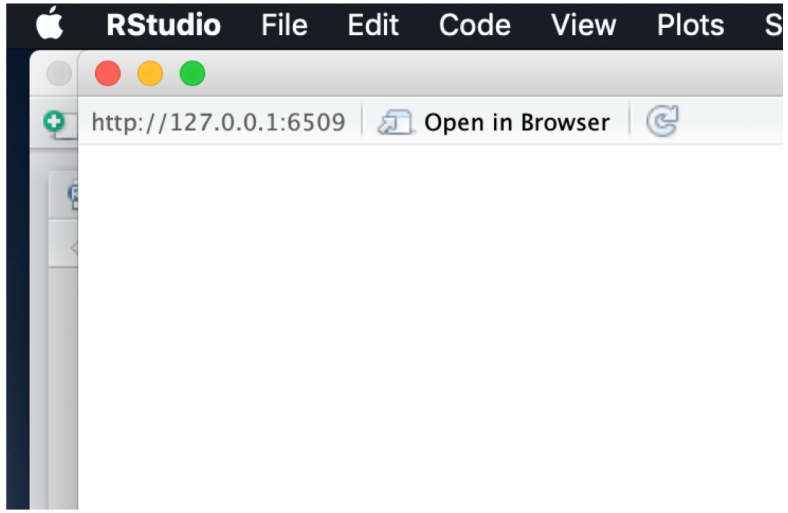
Week 7

- Shiny R
- Order by

Shiny R

Template

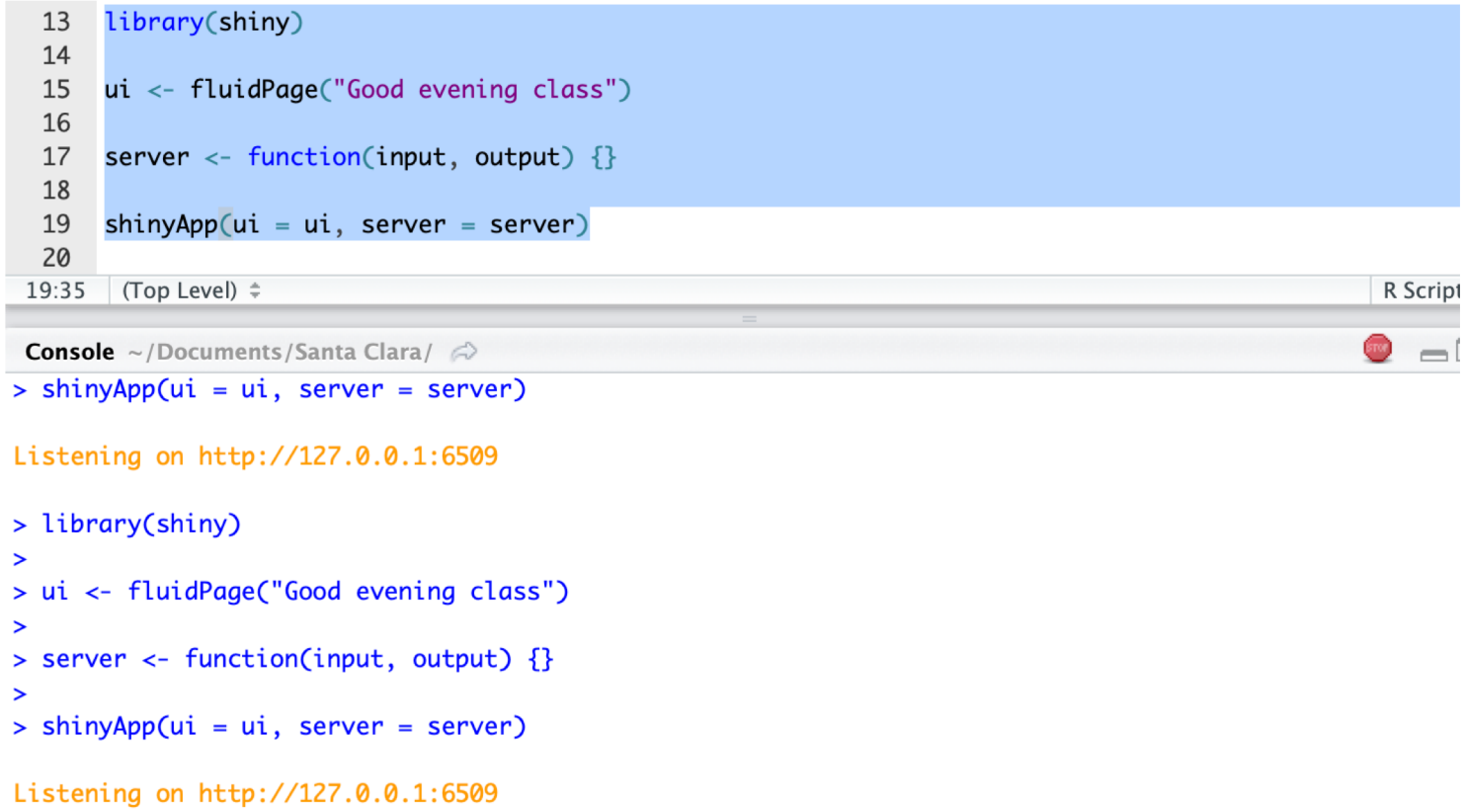
```
# This is a minimum template required to run Shiny R
# It just opens an empty window

library(shiny)

ui <- fluidPage()

server <- function(input, output) {}

shinyApp(ui = ui, server = server)

# Creates an empty page
```



Close an App

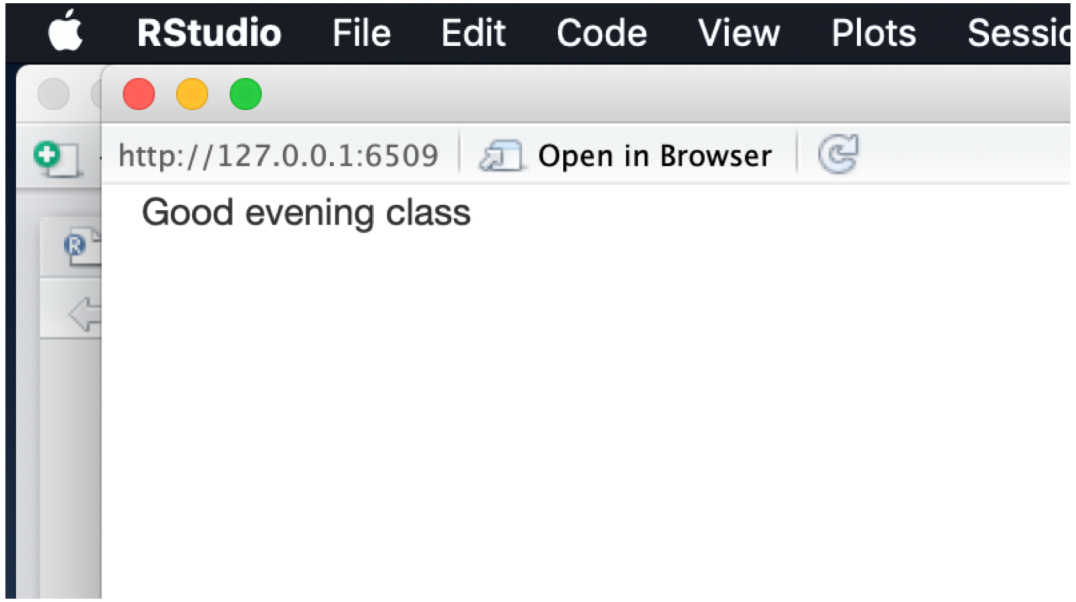
To stop the app from the R Studio, click on the Stop button.

You will need to do that before each code run.



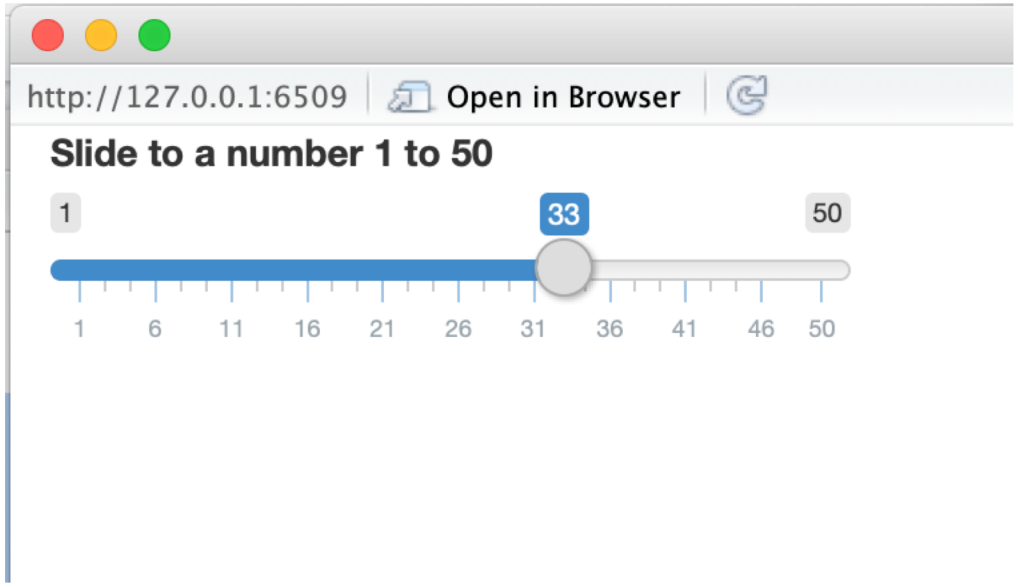
Add Text

```
# We will now add a simple text

library(shiny)

ui <- fluidPage("Good evening class")

server <- function(input, output) {}

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



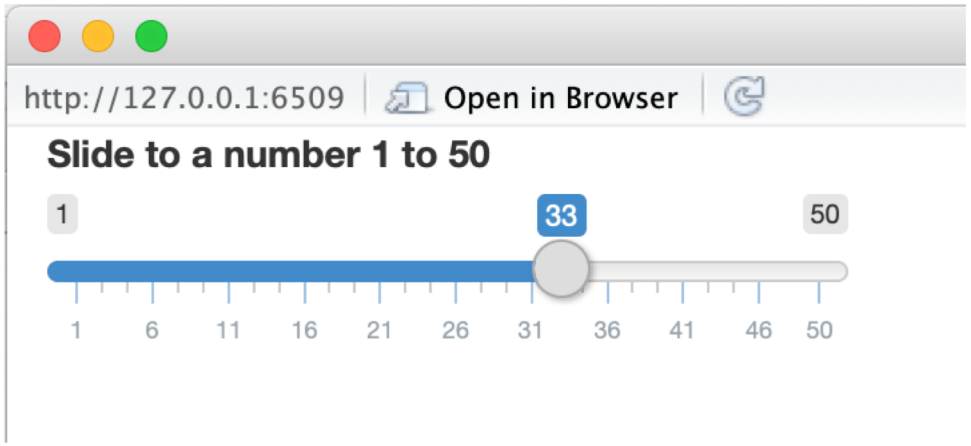
Add Input

```
#Add a slider as an Input
library(shiny)
ui <- fluidPage(sliderInput(inputId = "num",    label = "Slide to a number 1 to 50",    value = 33, min = 1, max = 50))
server <- function(input, output) {}
shinyApp(ui = ui, server = server)
```



Adding Output ID

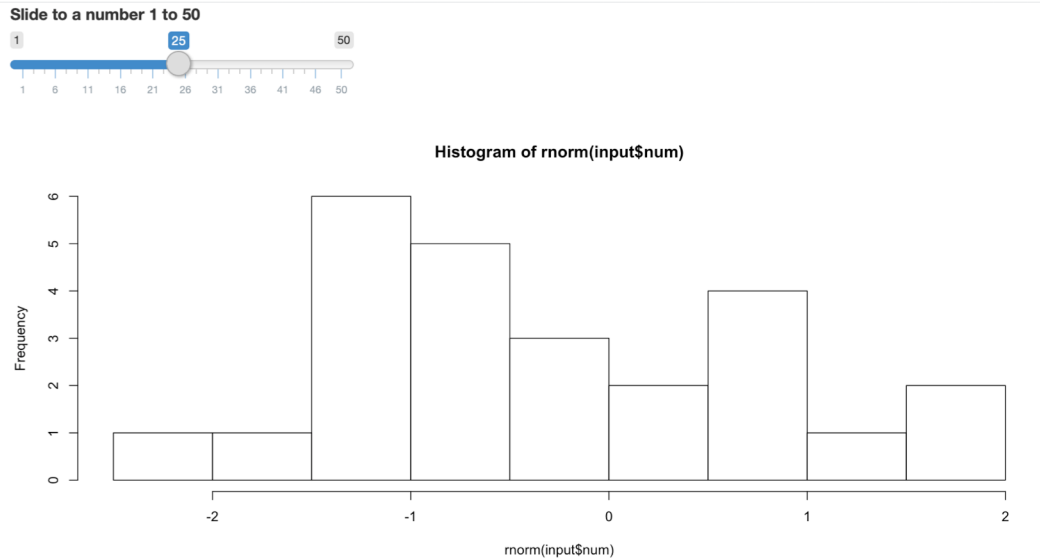
```
#Adding output ID “hist”
library(shiny)
    ui <- fluidPage(
    sliderInput(inputId = "num",
    label = "Slide to a number 1 to 50",
    value = 33, min = 1, max = 50),
    plotOutput("hist"))
server <- function(input, output) {}
shinyApp(ui = ui, server = server)
```



- No change on a screen, because we did not build it in program yet
- Output() just created a space for an R object which must be built in the server function

Adding Output in the Server Function

```
#Adding output to the server function
library(shiny)
ui <- fluidPage(
      sliderInput(inputId = "num",
      label = "Slide to a number 1 to 50",
      value = 33, min = 1, max = 50),
      plotOutput("hist"))

server <- function(input, output) {
            output$hist <- renderPlot({
            hist(rnorm(25))
})}

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

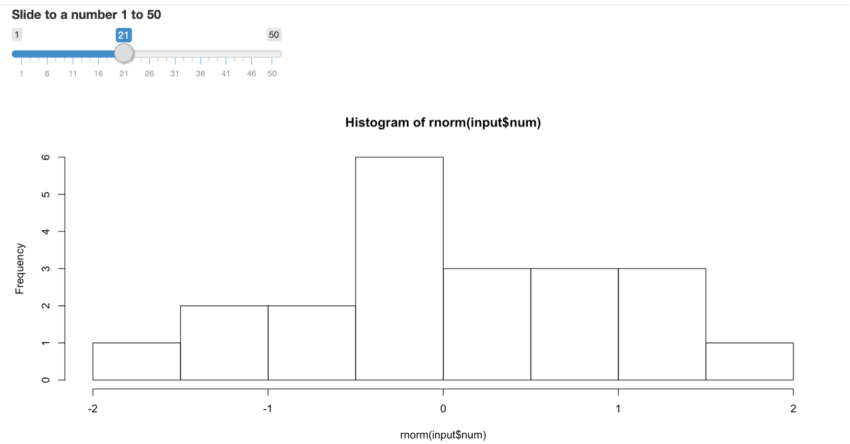


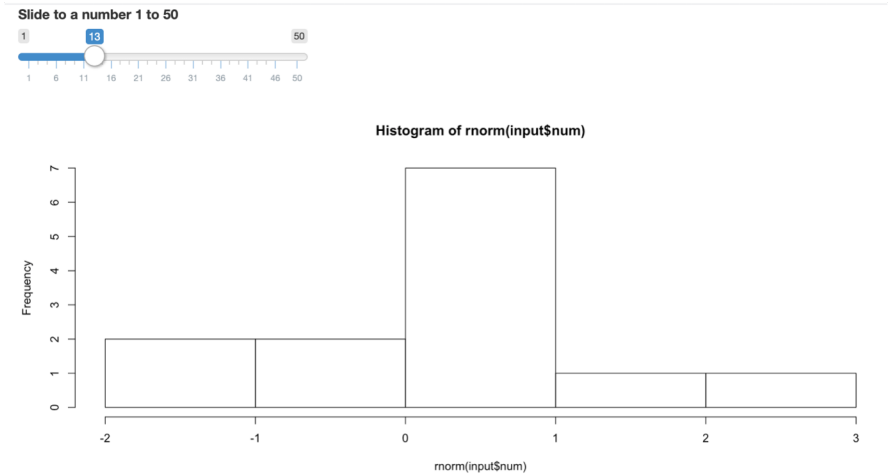
Server Function

- Assembles inputs and outputs
- Syntax: server <- function(input, output) {}
- Saves the output to output$ (output$var1)
- Builts the output with render() function
- Access input values with input$ (input$var2)

Reactivity

```
#Using reactive value “num” from slider to dynamically change
#histogram
library(shiny)
      ui <- fluidPage(
      sliderInput(inputId = "num",
      label = "Slide to a number 1 to 50",
      value = 33, min = 1, max = 50),
      plotOutput("hist"))
server <- function(input, output) {
      output$hist <- renderPlot({
      hist(rnorm(input$num))
       })}
shinyApp(ui = ui, server = server)
```





Reactivity Elements

Reactive Values

- Must use with reactive functions
  - renderPlot({hist(input$num)})
  - Otherwise error “operation not allowed)
- When value is changed
  - Notifies function
  - Object created by the reactive function responds
  - Output changes

Reactive Functions

- Use code to build an object
- Object will respond when reactive values change
- Types of functions
  - Render
  - Reactive
  - Isolate
  - Delayed reaction

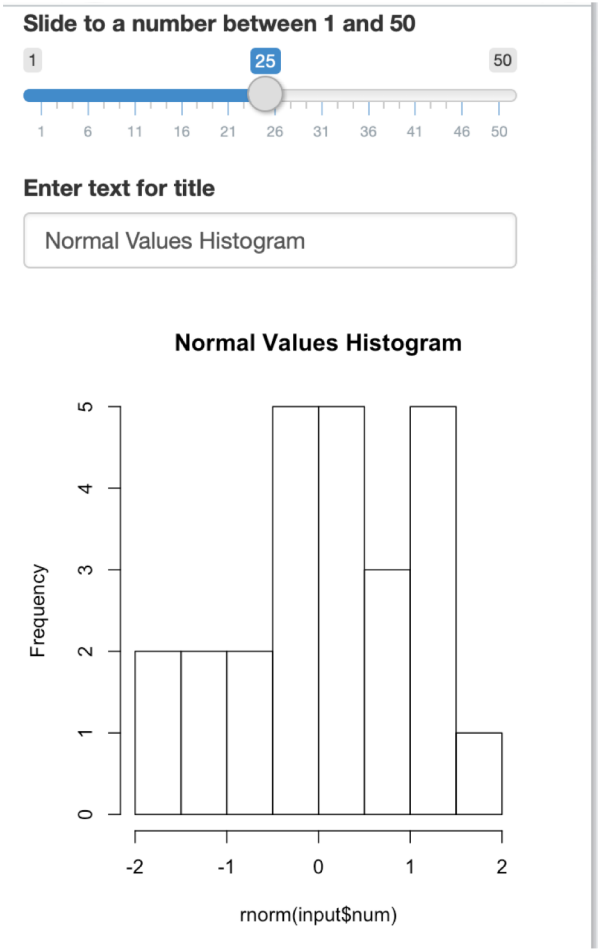
Render Functions

Displays an object
Saves results to output$
Usage: renderPlot({code})

function	creates
renderDataTable()	Interactive table
renderImage()	Image
renderPlot()	Plot
renderPrint()	Printed output
renderTable()	A table
renderText()	Character string
renderUI()	Shiny UI element

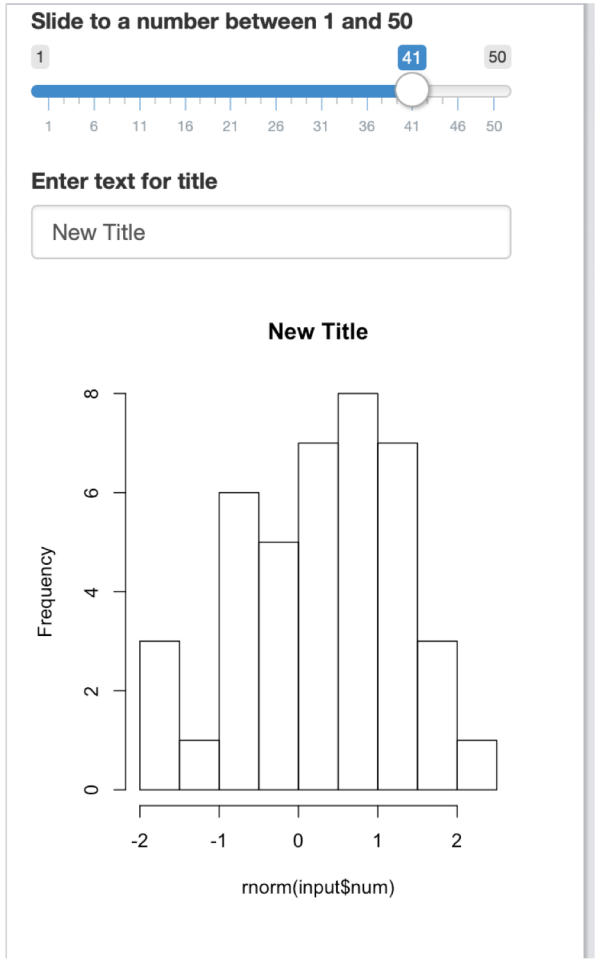
Example of a Render

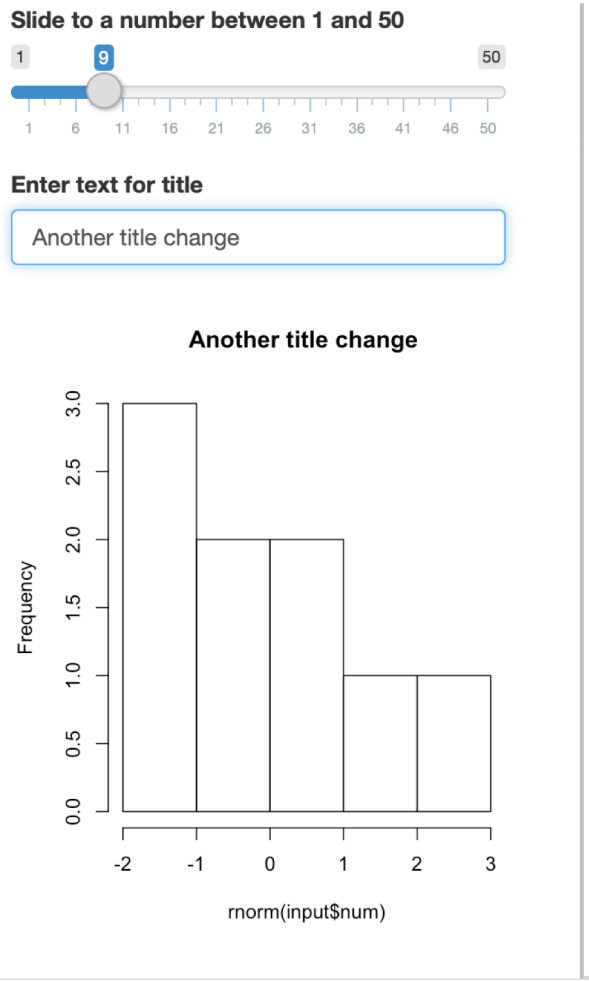
```
#this is example with two types of input: slider and text
#slider input to histogram
#text input to enable changing a title

library(shiny)
ui <- fluidPage(
      sliderInput(inputId = "num",
            label = "Slide to a number between 1 and 50",
            value = 25, min = 1, max = 50),
      textInput(inputId = "title",
            label = "Enter text for title",
            value = "Normal Values Histogram"),
            plotOutput("hist")
      )
server <- function(input, output) {
output$hist <- renderPlot({
hist(rnorm(input$num), main = input$title)
})}
shinyApp(ui = ui, server = server)
```



Render Example cont

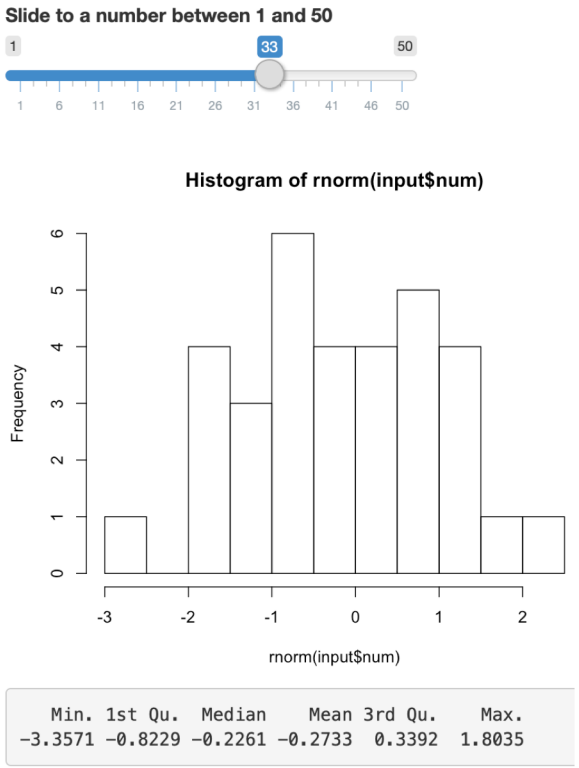
As number and title dynamically change, it is reflected in the output

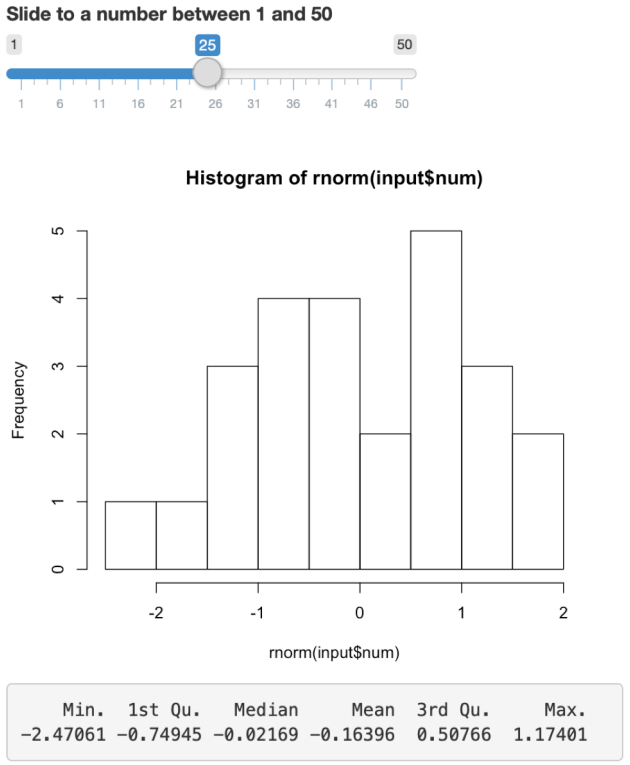




Modularize

```
#using reactive value to display two diagrams
#histogram and stats

library(shiny)
ui <- fluidPage(
      sliderInput(inputId = "num",
            label = "Slide to a number between 1 an
            value = 25, min = 1, max = 50),
            plotOutput("hist"),
      verbatimTextOutput("stats"))
server <- function(input, output) {
      output$hist <- renderPlot({
            hist(rnorm(input$num))    })
      output$stats <- renderPrint({
            summary(rnorm(input$num))
})}
shinyApp(ui = ui, server = server)
```



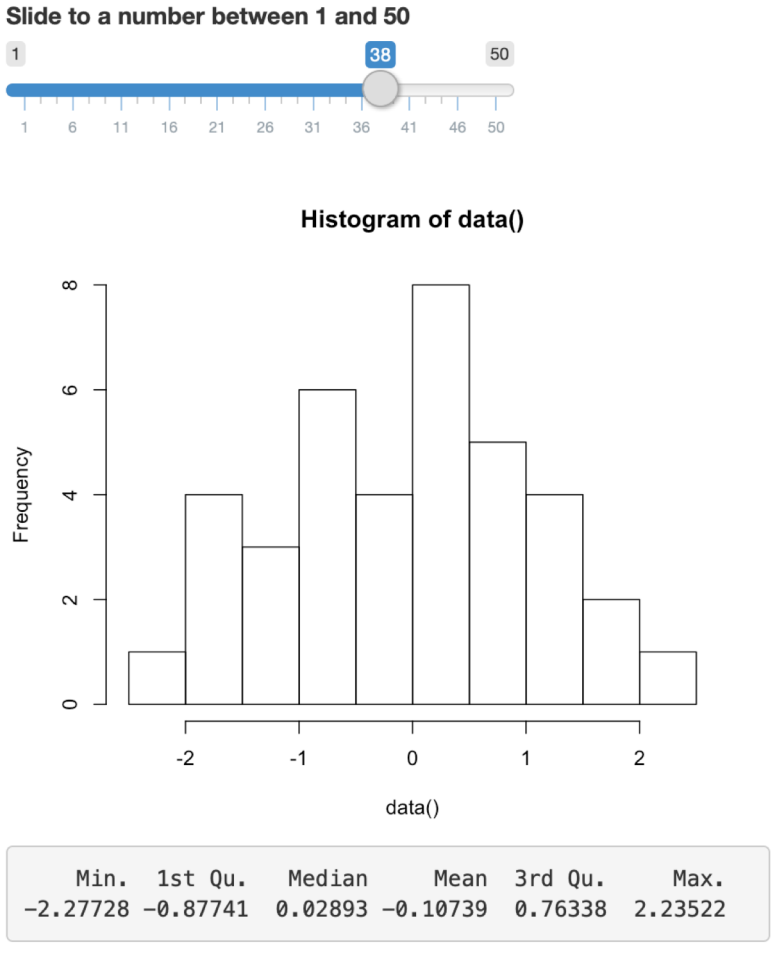


Reactive Object

- Builds a reactive object/reactive expressions to be used as functions
- Usage: data <- reactive({code})
- Example:
  - Another way to apply reactive value to multiple objects
  - Creating an reactive object first
  - Then using that object in render function (in server section)

Reactive Object Example

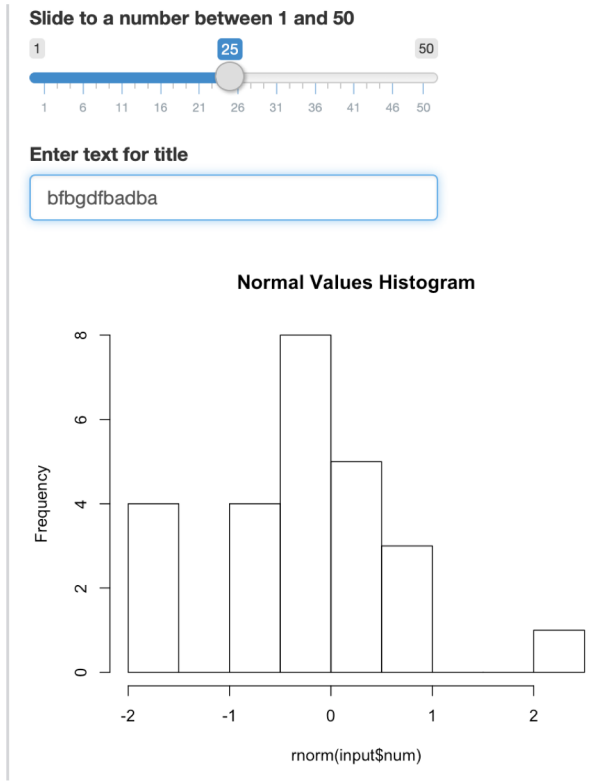
```
library(shiny)
ui <- fluidPage(
        sliderInput(inputId = "num",
        label = "Slide to a number between 1 and 50",
        value = 25, min = 1, max = 50),
    plotOutput("hist"),
        verbatimTextOutput("stats"))
server <- function(input, output) {
    data <- reactive({
        rnorm(input$num)
     })
    output$hist <- renderPlot({
        hist(data())
     })
    output$stats <- renderPrint({
        summary(data())
})}
shinyApp(ui = ui, server = server)
```



Isolate

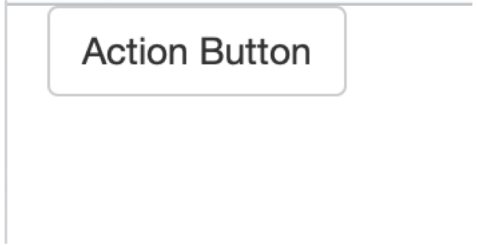
- Makes object non reactive
- Use isolate() to disable reactivity

```
library(shiny)
ui <- fluidPage(
        sliderInput(inputId = "num",
                label = "Slide to a number between 1 and
                50",
                value = 25, min = 1, max = 50),
        textInput(inputId = "title",
                label = "Enter text for title",
                value = "Normal Values Histogram"),
        plotOutput("hist"))
server <- function(input, output) {
        output$hist <- renderPlot({
                hist(rnorm(input$num),
                        main = isolate({input$title}))
        })}
shinyApp(ui = ui, server = server)
```



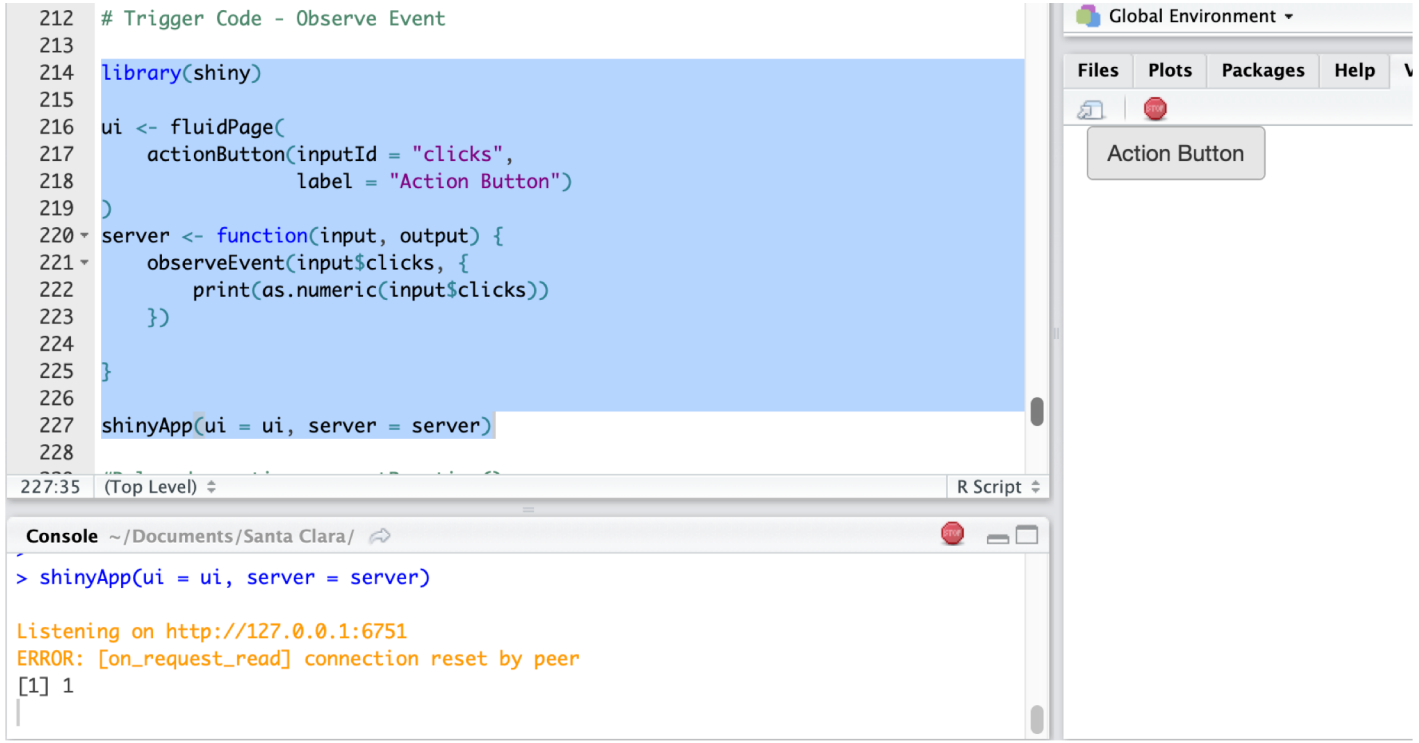
Action Button

```
# This example shows how to create an Action Button

library(shiny)
ui <- fluidPage(
      actionButton(inputId = "clicks",
      label = "Action Button"))
server <- function(input, output) {    }
shinyApp(ui = ui, server = server)
```



Trigger Code

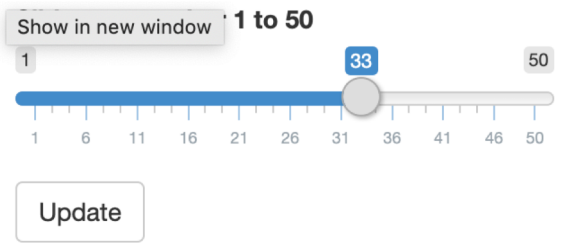
- Trigger code can be used to activate code when Action Button is pressed
- Triggers code to run on the server
- Usage: observeEvent(reactive_value, {code})

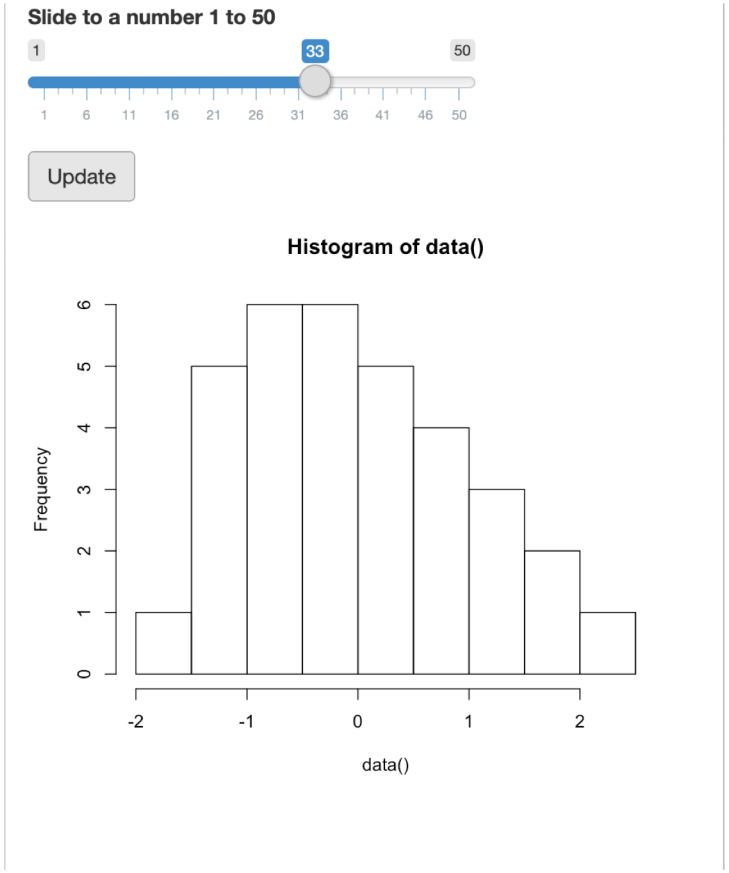


Delayed Reaction

- Does not react to the change in valu
- Recats to the Update button
- eventReactive(input$go, {code})

```
library(shiny)
ui <- fluidPage(
        sliderInput(inputId = "num",
                label = "Slide to a number 1 to 50",
                value = 33, min = 1, max = 50),
        actionButton(inputId = "go",
                label = "Update"),
        plotOutput("hist"))
server <- function(input, output) {
        data <- eventReactive(input$go, {
                rnorm(input$num)
 })
output$hist <- renderPlot({
         hist(data())
 })}
shinyApp(ui = ui, server = server)
```





Saving and Running the App

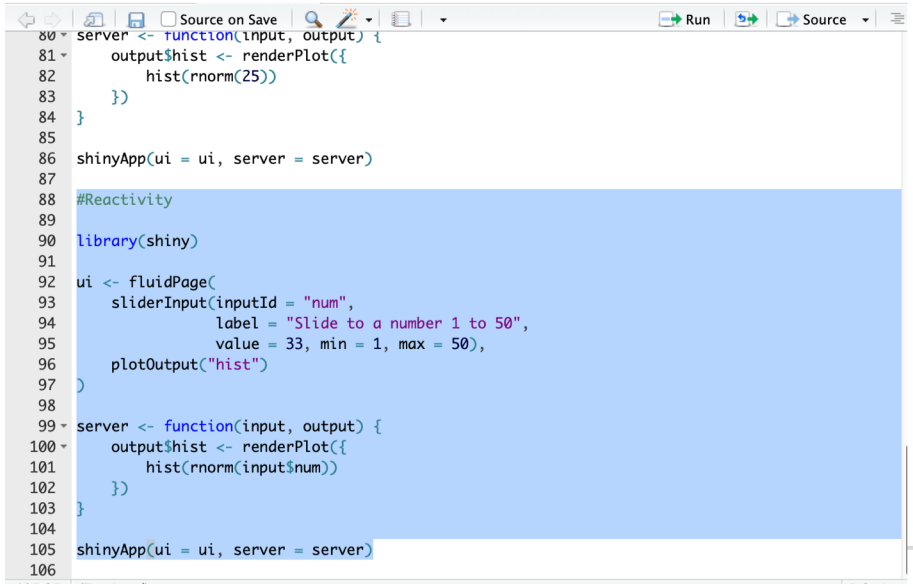
Launch the App from R Studio

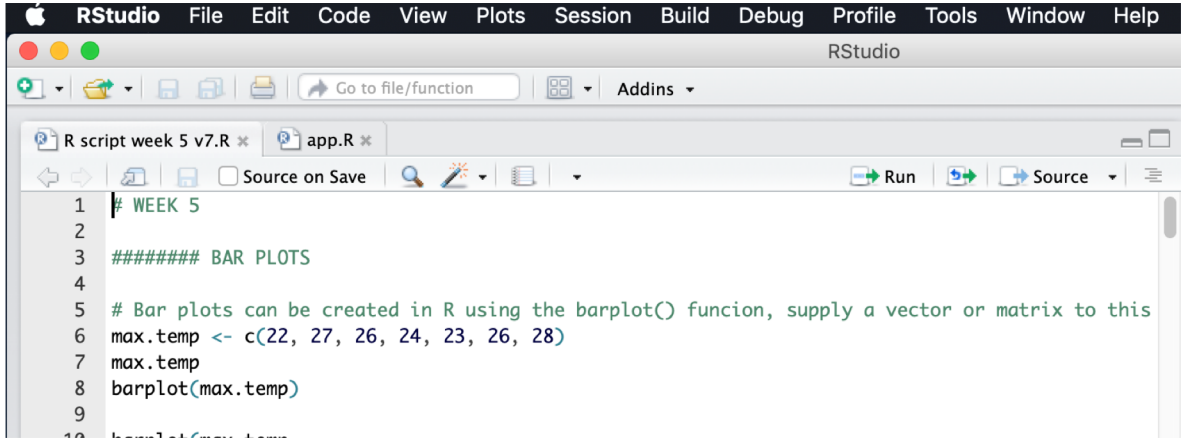
If file has been created as a R script

run the content of the file by using the Run button

To run portion of the file

Highlight the code and hit the Run button





Saving an App

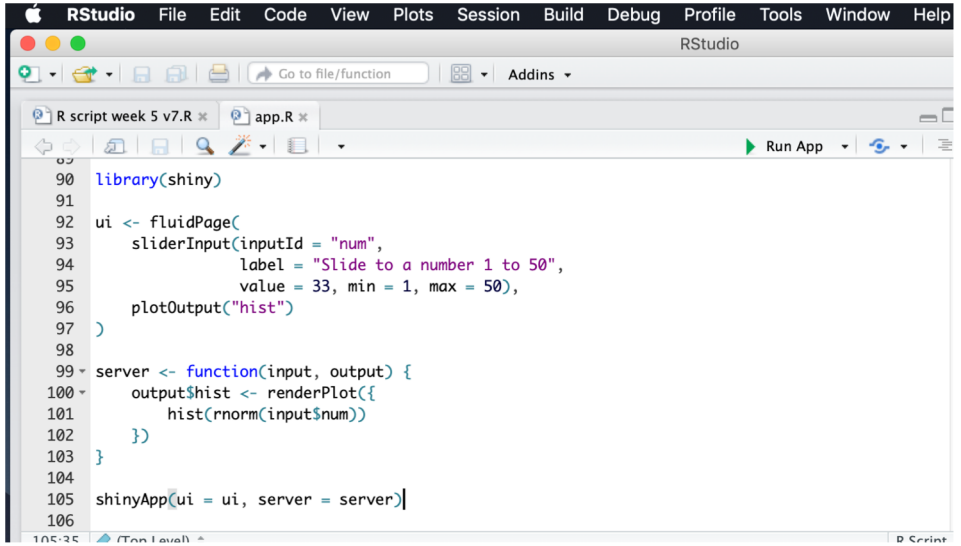
- Save the R script as app.R
- In the same directory save
  - Datasets
  - Images
  - Helper Scripts
  - Etc

- For larger scripts you can separate ui from the server portion of the script and save them as two files. This was default behaviour in older versions
  - ui.R
  - server.R

Running the App

- Shiny server will first look for app.R
- If app.R is not present, it will look for server.R and ui.R
- Names must be exact as noted

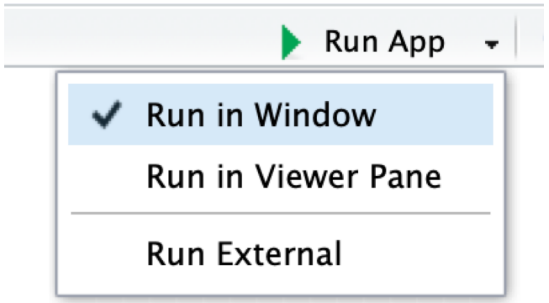
- If scripts is named properly,
  - R Studio will add the “Run App” button
  - Options how to run it



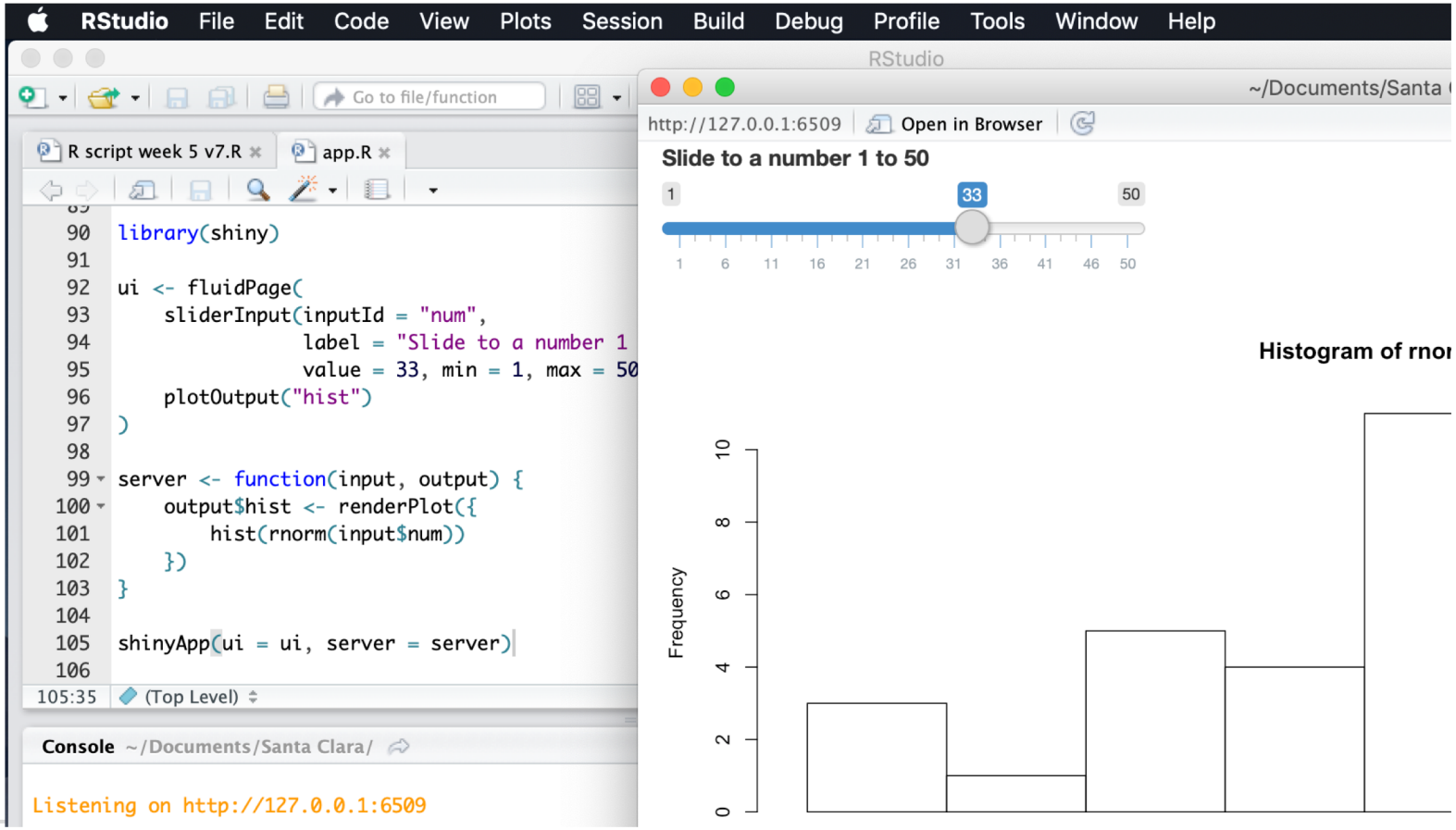
Options for Display

- We can choose how the window will be opened:
  - Run in window
  - Run in Viewer Pane
  - Run Externally, in a browser

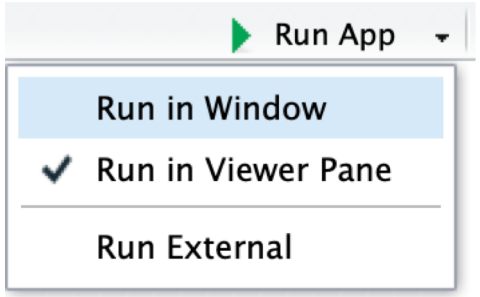
Run in Window



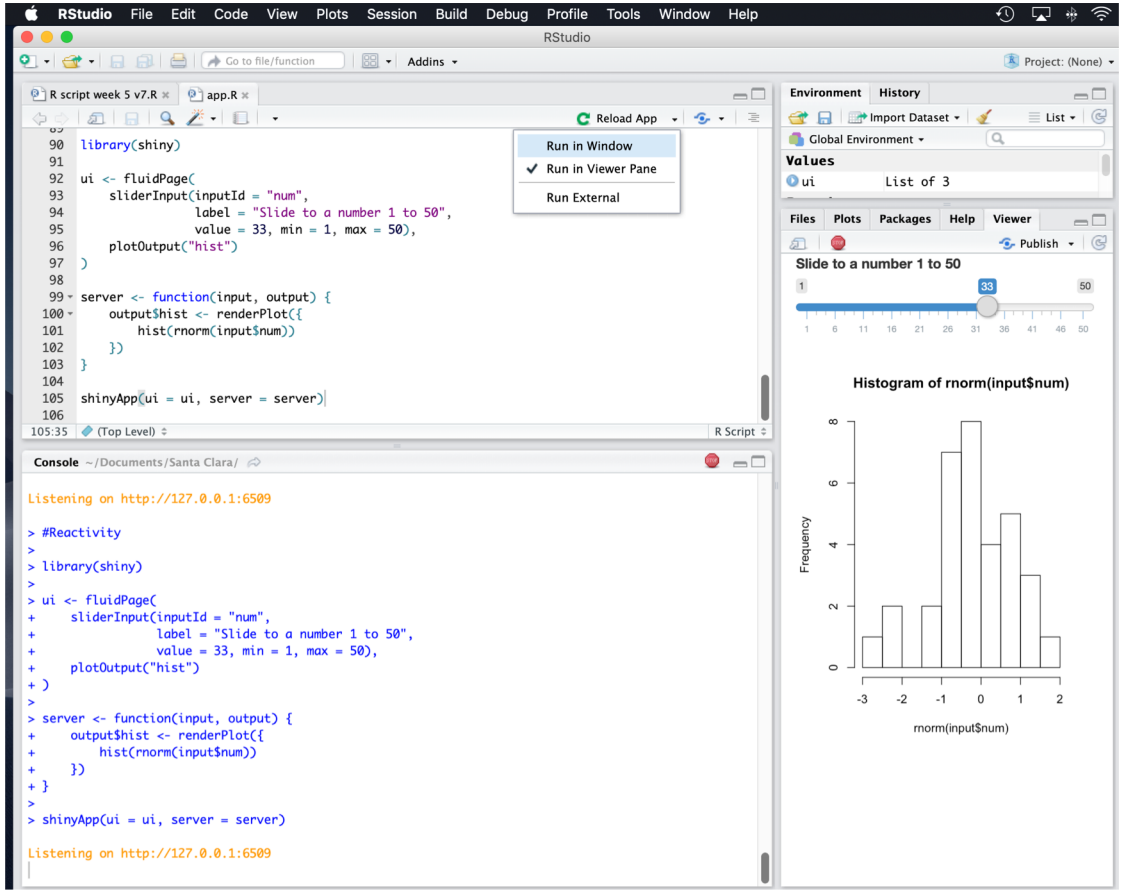
Opens a separate window



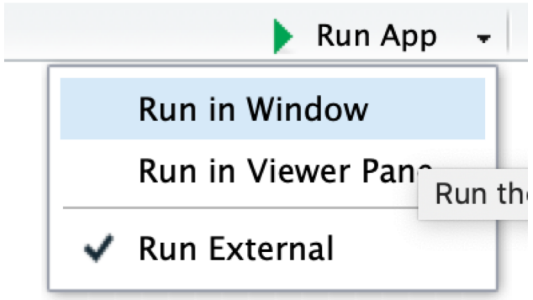
Run in Viewer Pane



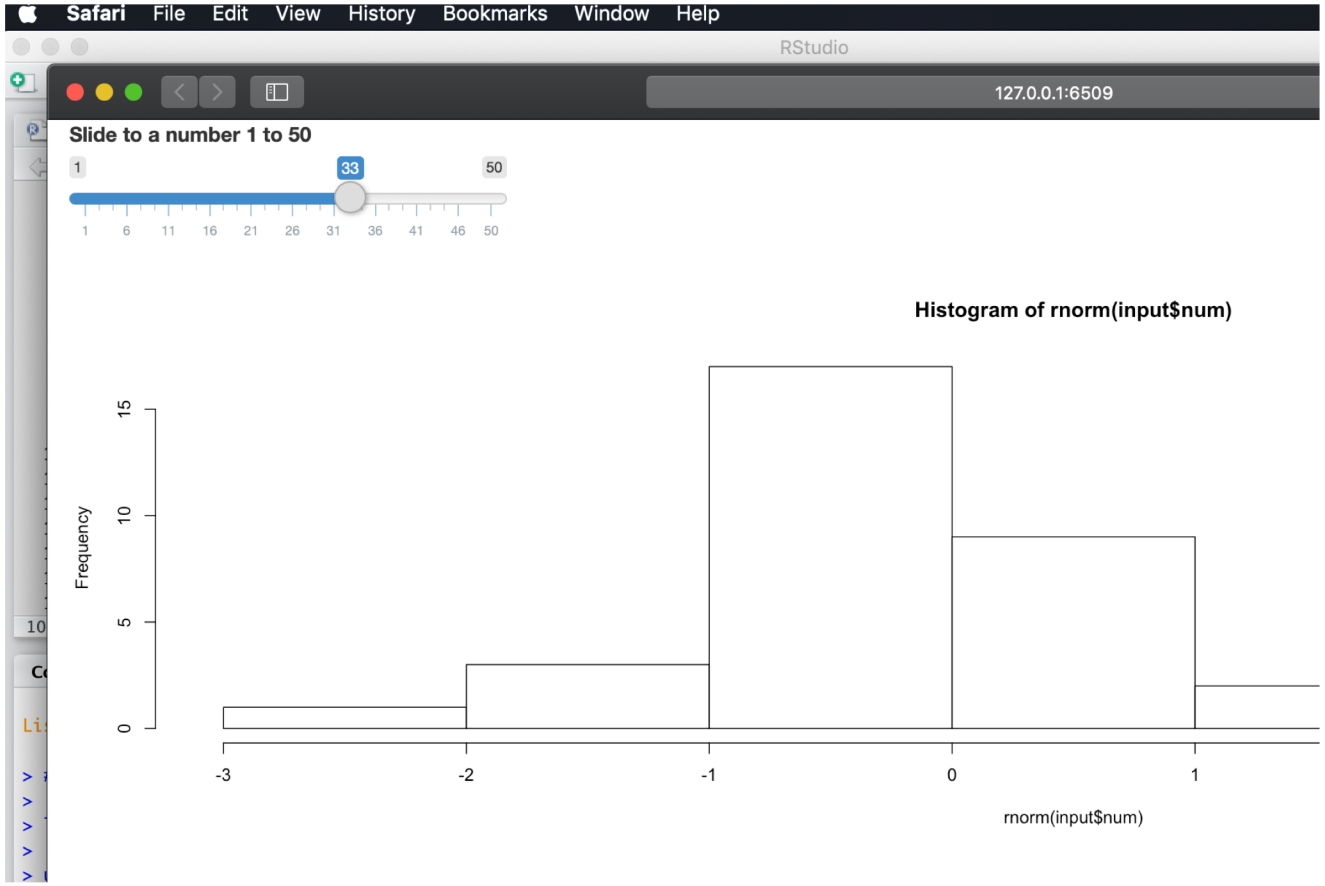
Displays output in a viewer pane



Run External



Opens a browser

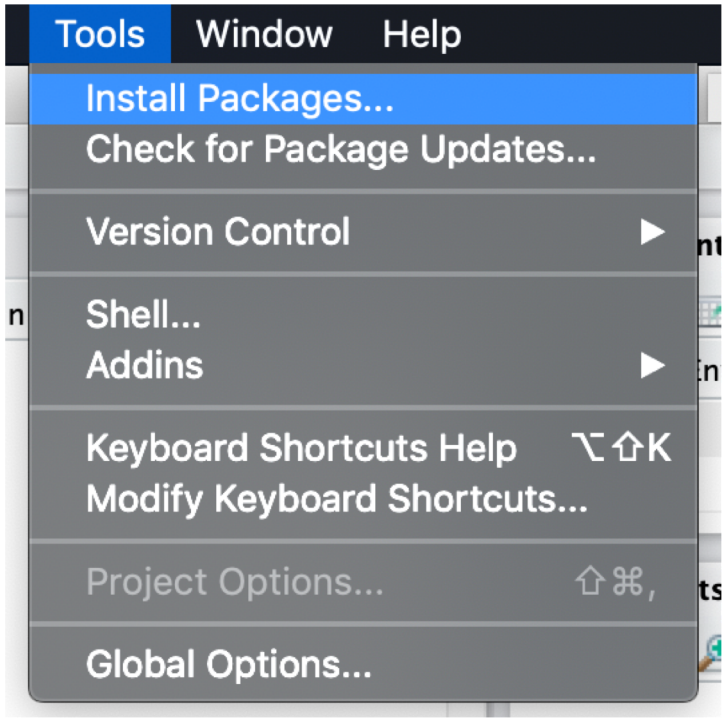


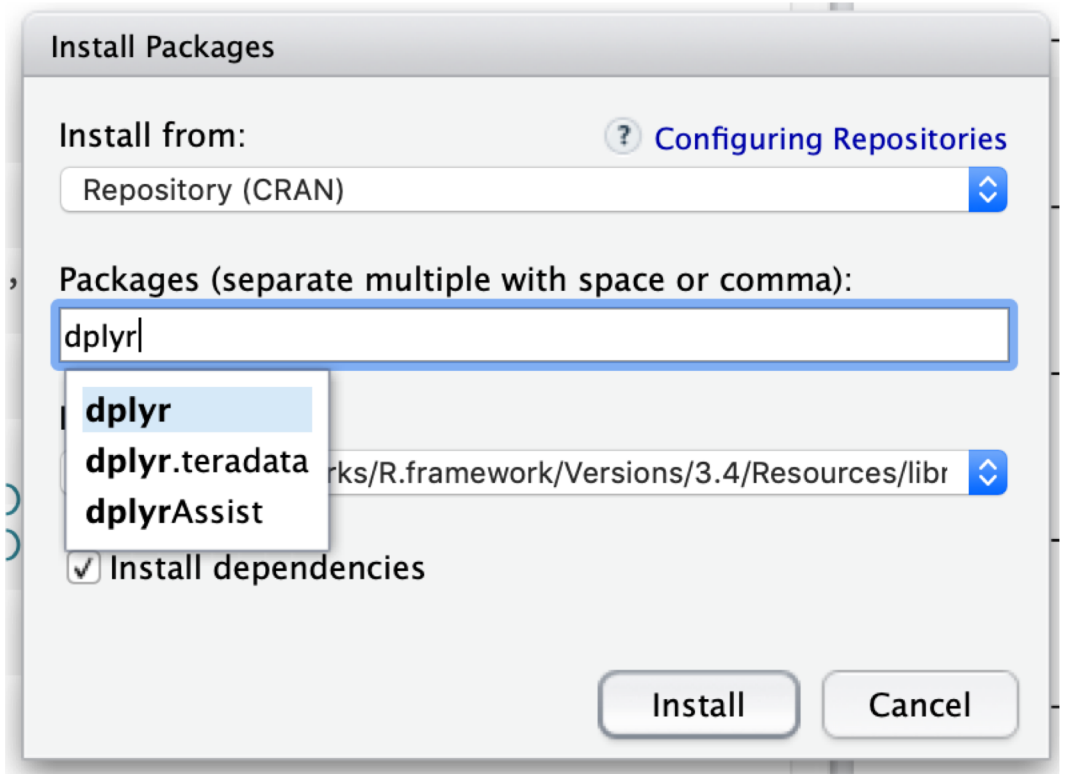
Data Wrangling with dplyr

What is dplyr

- dplyr is R package for common data manipulation operations such as
  - Tidy data
  - Reshape Data
  - Subsets
  - Summarise
  - Add new variables
  - Combine Data Sets
- Requires library splyr
- Data sets used in examples are from library datasets

Install dplyr Library





Install dplyr Library from Console

```
> install.packages("dplyr")
also installing the dependencies ‘ellipsis’, ‘zeallot’, ‘utf8’, ‘vctrs’, ‘cli’, ‘crayon’, ‘fansi’, ‘p
illar’, ‘purrr’, ‘assertthat’, ‘pkgconfig’, ‘Rcpp’, ‘rlang’, ‘tibble’, ‘tidyselect’, ‘BH’, ‘plogr’


  There are binary versions available but the source versions are later:
          binary source needs_compilation
ellipsis    0.1.0  0.3.0              TRUE
vctrs       0.1.0  0.2.0              TRUE
pillar      1.3.1  1.4.2             FALSE
purrr       0.3.2  0.3.3              TRUE
pkgconfig   2.0.2  2.0.3             FALSE
Rcpp        1.0.1  1.0.2              TRUE
rlang       0.3.1  0.4.1              TRUE
tibble      2.1.1  2.1.3              TRUE
dplyr     0.8.0.1  0.8.3              TRUE

Do you want to install from sources the packages which need compilation?
y/n: y
trying URL 'https://cran.rstudio.com/bin/macosx/el-capitan/contrib/3.4/zeallot_0.1.0.tgz'
Content type 'application/x-gzip' length 43124 bytes (42 KB)
==================================================
downloaded 42 KB

trying URL 'https://cran.rstudio.com/bin/macosx/el-capitan/contrib/3.4/utf8_1.1.4.tgz'
Content type 'application/x-gzip' length 172781 bytes (168 KB)
```

Head

- Display the head of the dataset

```
> library(dplyr)
>
> head(airquality)
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
6    28      NA 14.9   66     5   6
```

Filter

- Filter function will return only raws that meet the criteria

```
> library(dplyr)
>
> filter(airquality, Day == 1 & Wind > 5)
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    NA     286  8.6   78     6   1
3    39      83  6.9   81     8   1
4    96     167  6.9   91     9   1
>
```

Mutate

- Mutate is used to add new variable

```
> mutate(airquality, Celsius = (Temp -32) *5 / 9)
    Ozone Solar.R Wind Temp Month Day  Celsius
1      41     190  7.4   67     5   1 19.44444
2      36     118  8.0   72     5   2 22.22222
3      12     149 12.6   74     5   3 23.33333
4      18     313 11.5   62     5   4 16.66667
5      NA      NA 14.3   56     5   5 13.33333
6      28      NA 14.9   66     5   6 18.88889
7      23     299  8.6   65     5   7 18.33333
8      19      99 13.8   59     5   8 15.00000
9       8      19 20.1   61     5   9 16.11111
10     NA     194  8.6   69     5  10 20.55556
11      7      NA  6.9   74     5  11 23.33333
```

Count

```
> library(dplyr)
>
> count(airquality, Month)
# A tibble: 5 x 2
  Month     n
  <int> <int>
1     5    31
2     6    30
3     7    31
4     8    31
5     9    30
>
```

Group Data

- Group data into rows of the same value of specified
- Usage
  - library(dplyr)
  - group_by(object, row_names, ….)
  - Ungroup(object)

Order-By Example

```
library(dplyr)
airgrp <- group_by(airquality, Month, Day)
airgrp
```

```
> library(dplyr)
>
> airgrp <- group_by(airquality, Month, Day)
>
> airgrp
# A tibble: 153 x 6
# Groups:   Month, Day [153]
   Ozone Solar.R  Wind  Temp Month   Day
   <int>   <int> <dbl> <int> <int> <int>
 1    41     190   7.4    67     5     1
 2    36     118   8      72     5     2
 3    12     149  12.6    74     5     3
 4    18     313  11.5    62     5     4
 5    NA      NA  14.3    56     5     5
 6    28      NA  14.9    66     5     6
 7    23     299   8.6    65     5     7
 8    19      99  13.8    59     5     8
 9     8      19  20.1    61     5     9
10    NA     194   8.6    69     5    10
# … with 143 more rows
```

Thank You