An Introduction to R Shiny

(*shiny* is an R package by R Studio)

- A web application framework for R
- R Shiny makes it very easy to build interactive web applications with R
- Much of this introductory information is taken directly from the available tutorial from R Studio
  - http://shiny.rstudio.com/tutorial/

Tutorial “Hello Shiny!”

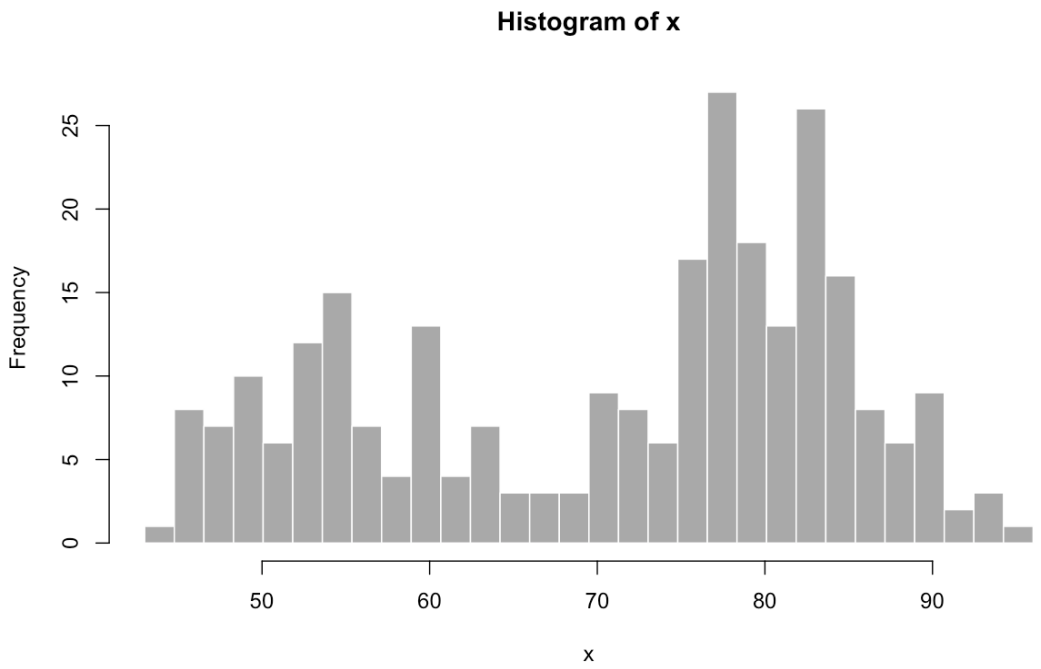
Open R and run the tutorial example

```
> library(shiny)
> runExample("01_hello")
```

Hello Shiny!

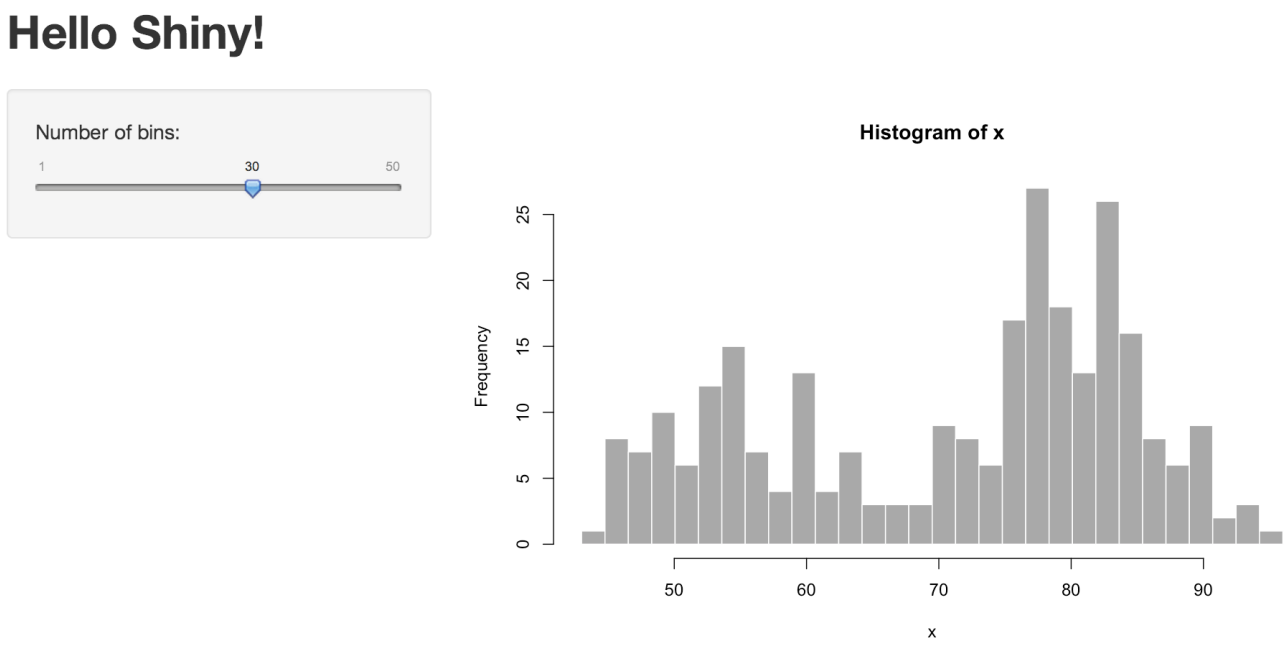
Number of bins:

1 30 50



- Shiny applications have two components:
  - **a user-interface definition (UI)** file called `ui.R`
    - This source code is used to set-up what the user will actually see in the web app, i.e. the layout of the web page
      - Title, sliders, widgets, plots, location of items on the page, etc.
    - This source code is also used to accept input from the user
      - e.g. It recognizes what the user has entered in the slider
  - **a server script** file called `server.R`
    - This source code does the computational R work "under the hood" with familiar functions such as *hist(), plot()*, etc.
    - This source code contains the instructions that your computer needs to build your app
- These two source files work together to create your R Shiny web application

Let's take a close look at these two files for the "Hello Shiny!" app



Example `ui.r` file from tutorial "Hello Shiny!"

(setting-up the structure of the web page)

ui.R

```
library(shiny)

# Define UI for application that draws a histogram
shinyUI(fluidPage(

  # Application title
  titlePanel("Hello Shiny!"),

  # Sidebar with a slider input for the 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")
    )
  )
))
```

Create a layout with a sidebar (1/3 space of page) and main area (2/3 space of page).

Define sidebar: Put the slider for input in the sidebar panel and name the input as "bins".

Define your slider and set initial settings for slider (value=30).

Define main panel: Put the generated plot in the main panel.

Give your output plot a name, such as "`distPlot`". This name will also be used in the `server.r` file.

Example `server.r` file from tutorial "Hello Shiny!"

(the "under the hood" computations)

server.R

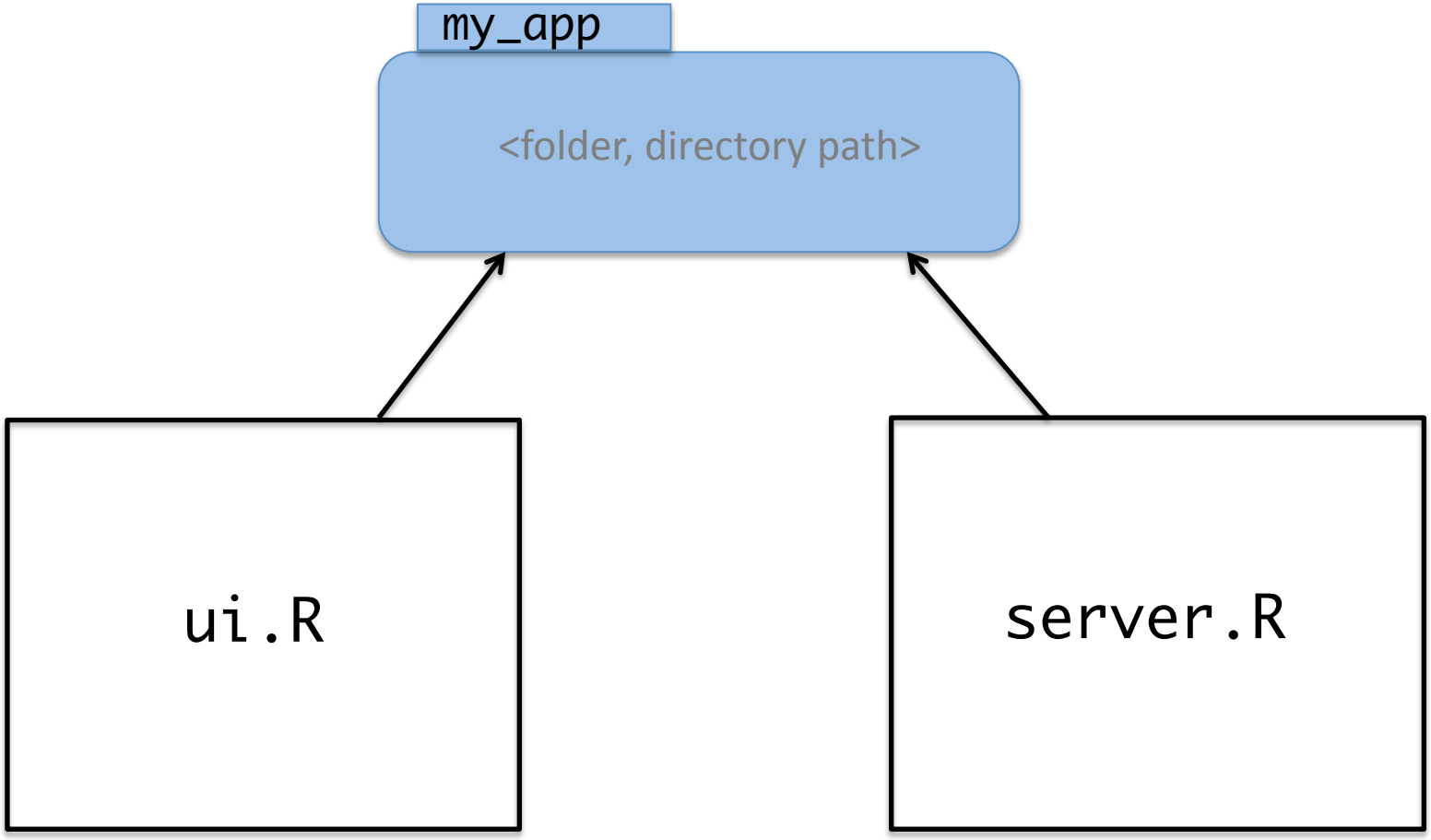
```
library(shiny)

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

  # Expression that generates a histogram. The expression is
  # wrapped in a call to renderPlot to indicate that:
  #
  #  1) It is "reactive" and therefore should re-execute automatically
  #     when inputs change
  #  2) Its output type is a plot

  output$distPlot <- renderPlot({
    x    <- faithful[, 2]  # Old Faithful Geyser data
    bins <- seq(min(x), max(x), length.out = input$bins + 1)

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

Name of output plot stated in the `ui.r` file, or "`distPlot`".

Set-up arguments for the *hist()* function based on user-input "bins" from web app.

Generate the *hist()* plot with given arguments.

Folder/File structure for R shiny app



To make an R Shiny app, start with this folder/file/filename structure. Put both files (named exactly `ui.R` and `server.R`) into a single folder named for your app.

This is the 'bare-bones' structure for a Shiny app. As you get more complex, you may include other things in this folder, such as a data file, or the 'global.R' file, but that's further down the road.

Running an R Shiny App

- Every Shiny app has the same structure:
  - two R scripts saved together in a directory. At a minimum, a Shiny app has ui.R and server.R files.
- You can create a Shiny app by making a new file directory and saving a ui.R and server.R file inside it. **Each app will need its own unique directory (or folder)**.
- You can run a Shiny app by giving the name of its directory to the function *runApp()*.

```
> library(shiny)
> runApp("my_app")
```

Running the “Hello shiny” app directly from the `ui.R` and `server.R` files

- After putting the two files from the “Hello Shiny!” tutorial into my local directory called `"Stat_6220/R-shiny/hello_shiny"`, I was able to run the same app with the following commands:

```
> setwd("Stat_6220/R-shiny")
> library(shiny)
> runApp("hello_shiny")
```

- Which means I can now edit and play around with the code to create a new app.

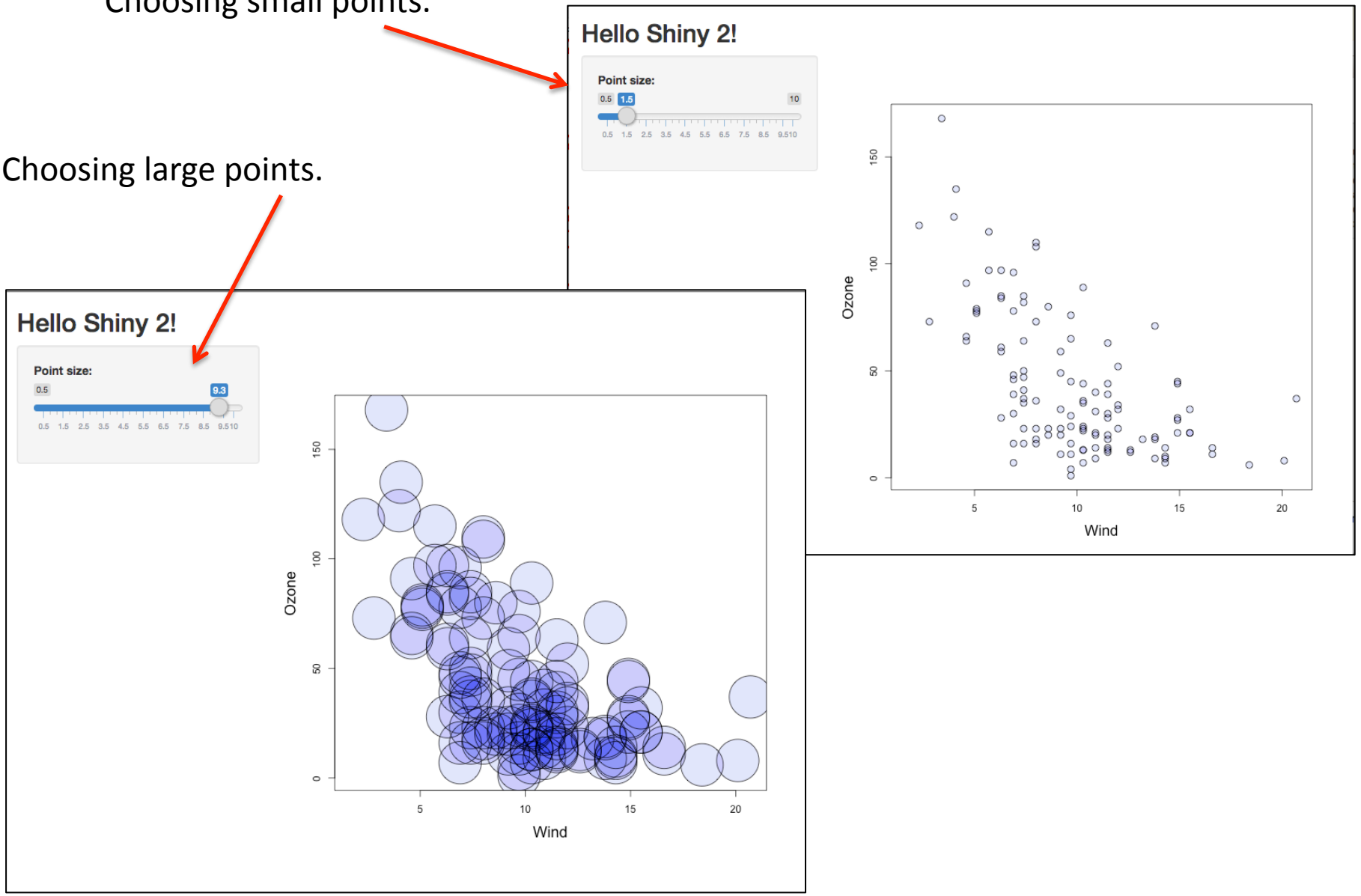
My new R Shiny app

- Airquality scatterplot: Ozone vs. Wind
  - Input slider: size of points

```
> setwd("Stat_6220/R-shiny")
> library(shiny)
> runApp("hello_shiny_2")
```

Choosing small points.

Choosing large points.



Web accessible version: http://shiny.rhondadecook.com/hello_shiny_2/

Example `ui.r` file from tutorial "Hello Shiny 2!"

(setting-up the structure of the web page)

```
library(shiny)

shinyUI(fluidPage(
    titlePanel("Hello Shiny 2!"),
    sidebarLayout(
        sidebarPanel(
            sliderInput("size",
                        "Point size:",
                        min=0.5,
                        max=10,
                        value=1)
        ),
        mainPanel(
            plotOutput("airPlot")
        )
    )
))
```

Create a layout with a sidebar (1/3 space) and main area (2/3 space).

Put the slider for inputs over in the sidebar panel.

Define your slider, user-defined input, and initial settings for slider.

Put the generated plot in the main panel.

Give your output plot a name, such as "`airPlot`". This name will also be used in the `server.r` file.

Example `server.r` file from tutorial "Hello Shiny 2!"

(the "under the hood" computations)

Name of plot stated in the `ui.r` file, or "`airPlot`".

```
library(shiny)

shinyServer(function(input,output){
    output$airPlot<-renderPlot({
        x<-airquality[,3]
        y<-airquality[,1]
        plot(x,y,col="#0000ff20",pch=16,
             cex=input$size, xlab="Wind",
             ylab="Ozone",cex.lab=1.5)
        points(x,y,col=1,pch=1,cex=input$size)
    }, height = 400, width = 400)
})
```

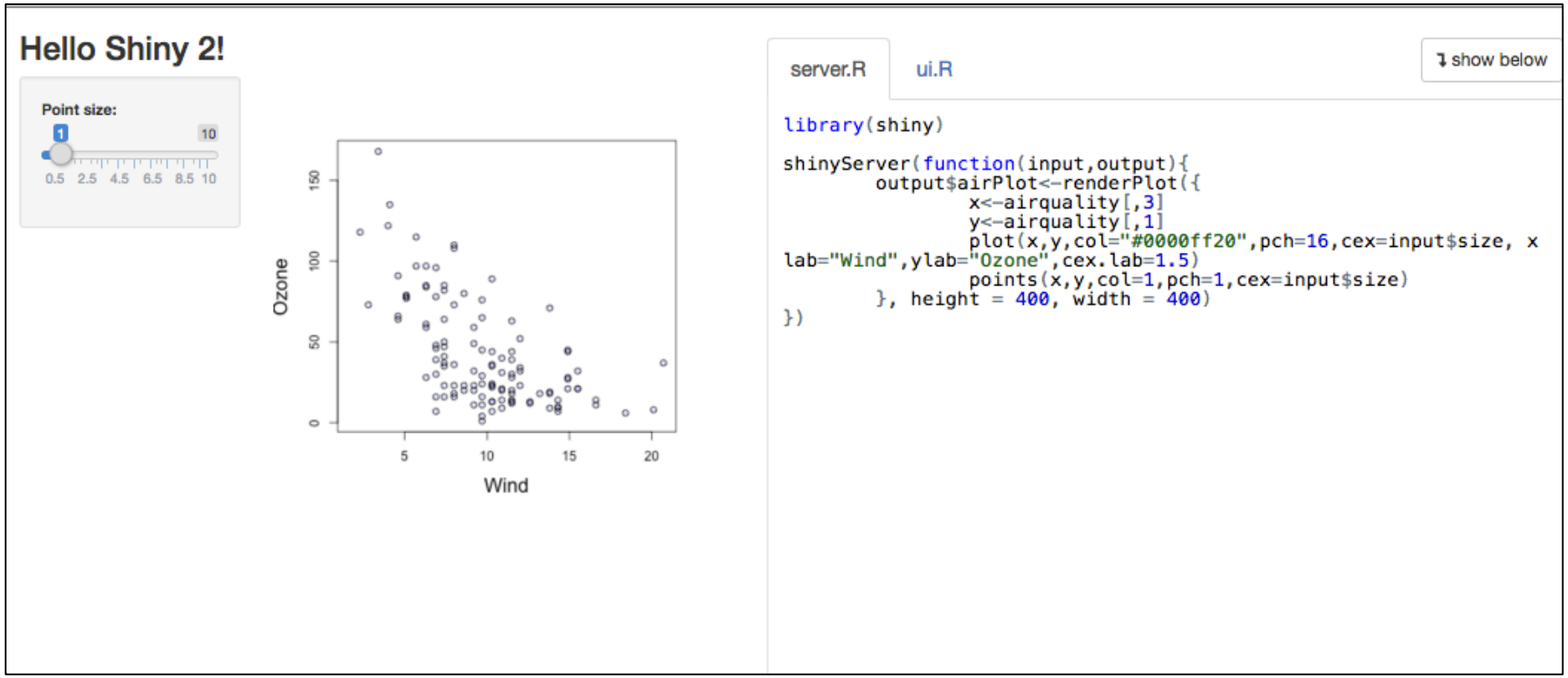
NOTE: this color gives transparent coloring for points.

Generate the plot based on user-input "size" from web app.

Set the plot dimensions.

Including R code in web app

```
> runApp("hello_shiny_2", display.mode="showcase")
```



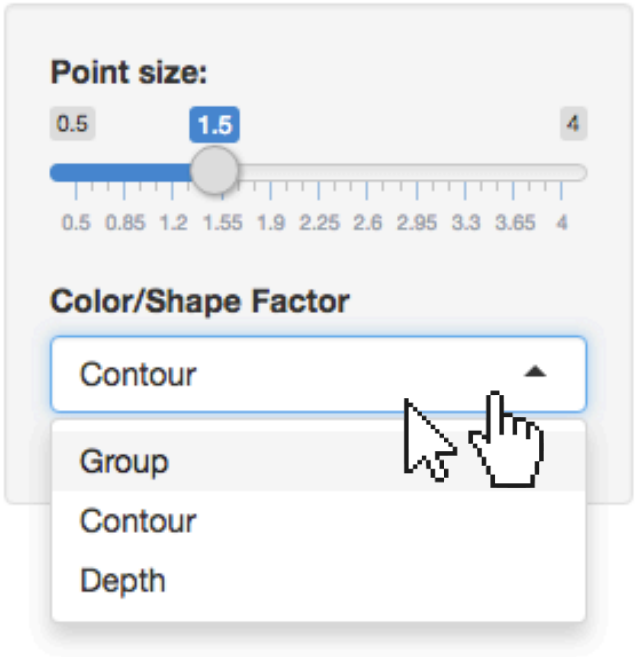
The viewable tabs let the user toggle between the ui.R and server.R source files.

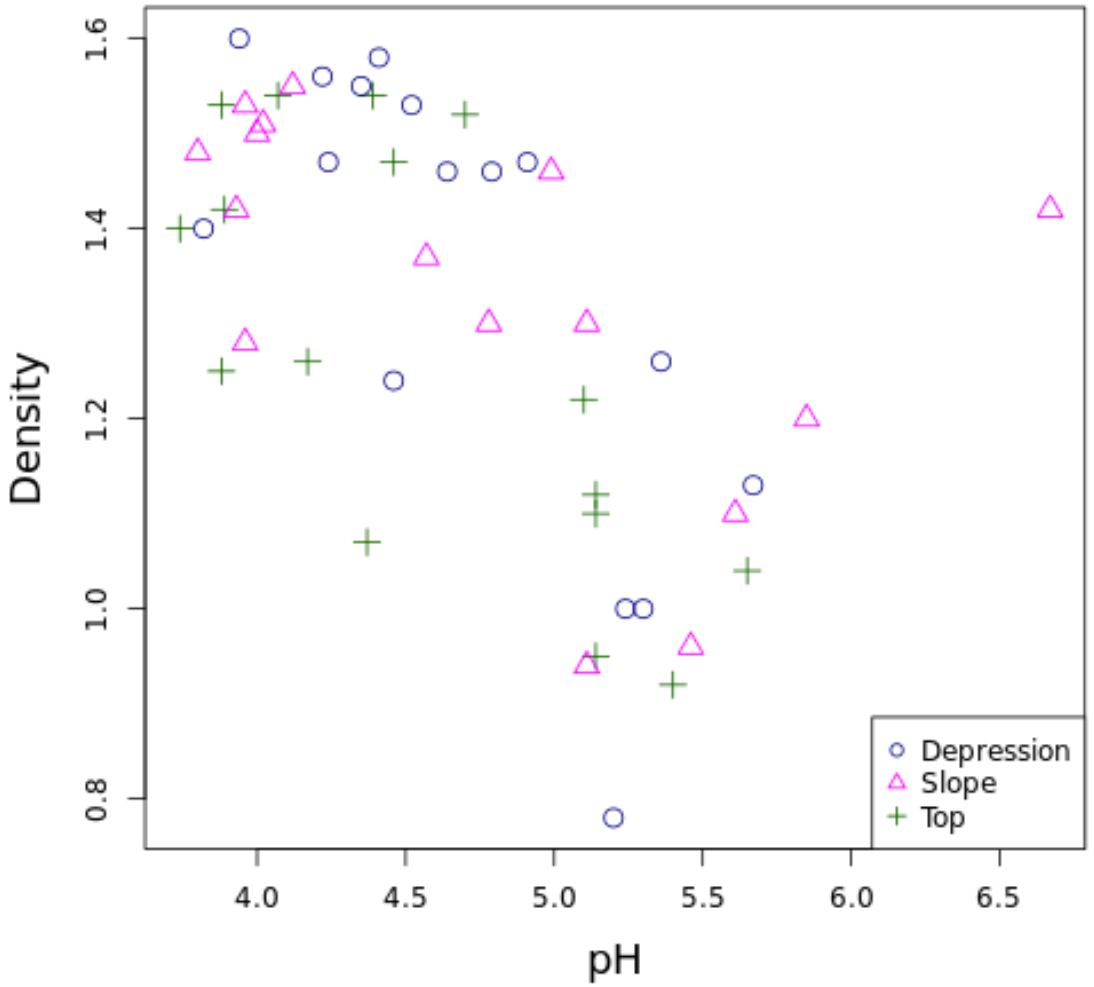
Drop-down lists (i.e. `selectInput`)

- Here, we will allow the user to color the points in an X-Y scatterplot based on a 3rd factor variable.
- We will also use the same 3rd factor variable to designate the shape of the point.
- We will carry forward the slider for choosing the point size of the points as well.

```
> runApp("hello_shiny_3")
```

Hello Shiny 3!





Web accessible version: http://shiny.rhondadecook.com/hello_shiny_3/

Example `ui.r` file from tutorial "Hello Shiny 3!"

(setting-up the structure of the web page)

```
library(shiny)
library(car)  ## To get access to 'Soils' data set.

shinyUI(fluidPage(
    titlePanel("Hello Shiny 3!"),
    sidebarLayout(
        sidebarPanel(
            sliderInput("size", "Point size:",
                min=0.5, max=4, value=2),
            selectInput("chosen", label="Color/Shape
                Factor", choices = c("Group","Contour",
                "Depth"), selected="Contour",
                multiple = FALSE)
        ),
        mainPanel(
            plotOutput("soilPlot")
        )
    )
))
```

Create a layout with a sidebar (1/3 space) and main area (2/3 space).

Put the slider for inputs over in the sidebar panel.

Put the drop-down choices (selectInput) for inputs in the sidebar panel, define choices, and set a default choice.

Put the generated plot in the main panel.

Give your output plot a name, such as "`soilPlot`". This name will also be used in the `server.r` file.

Example `server.r` file from tutorial "Hello Shiny 3!"

(the "under the hood" computations)

```
library(shiny)
library(car) ## To get access to 'Soils' data set.

shinyServer(function(input,output){
    output$soilPlot<-renderPlot({
        ## The chosen color/shape factor data:
        pchInput<-Soils[,input$chosen]
        x<-Soils$pH
        y<-Soils$Dens
        ## Choose enough colors for max levels of any plotting factor:
        colorVec<-colors()[c(490,450,81,52,67,83,84,47,142,121,373,530)]
        ## Create your numeric identifier for color/shape:
        numericInput<-as.numeric(pchInput)
        ## Assign each observation the appropriate color:
        colorForPoints<-colorVec[numericInput]
        plot(x,y,col=colorForPoints,pch=numericInput,xlab="pH",
            ylab="Density",cex.lab=1.5,cex=input$size)
        legend("bottomright",levels(pchInput),col=colorVec,
            pch=sort(unique(numericInput)))
    }, height = 500, width = 500)
})
```

Name of plot stated in the `ui.r` file, or "`soilPlot`".

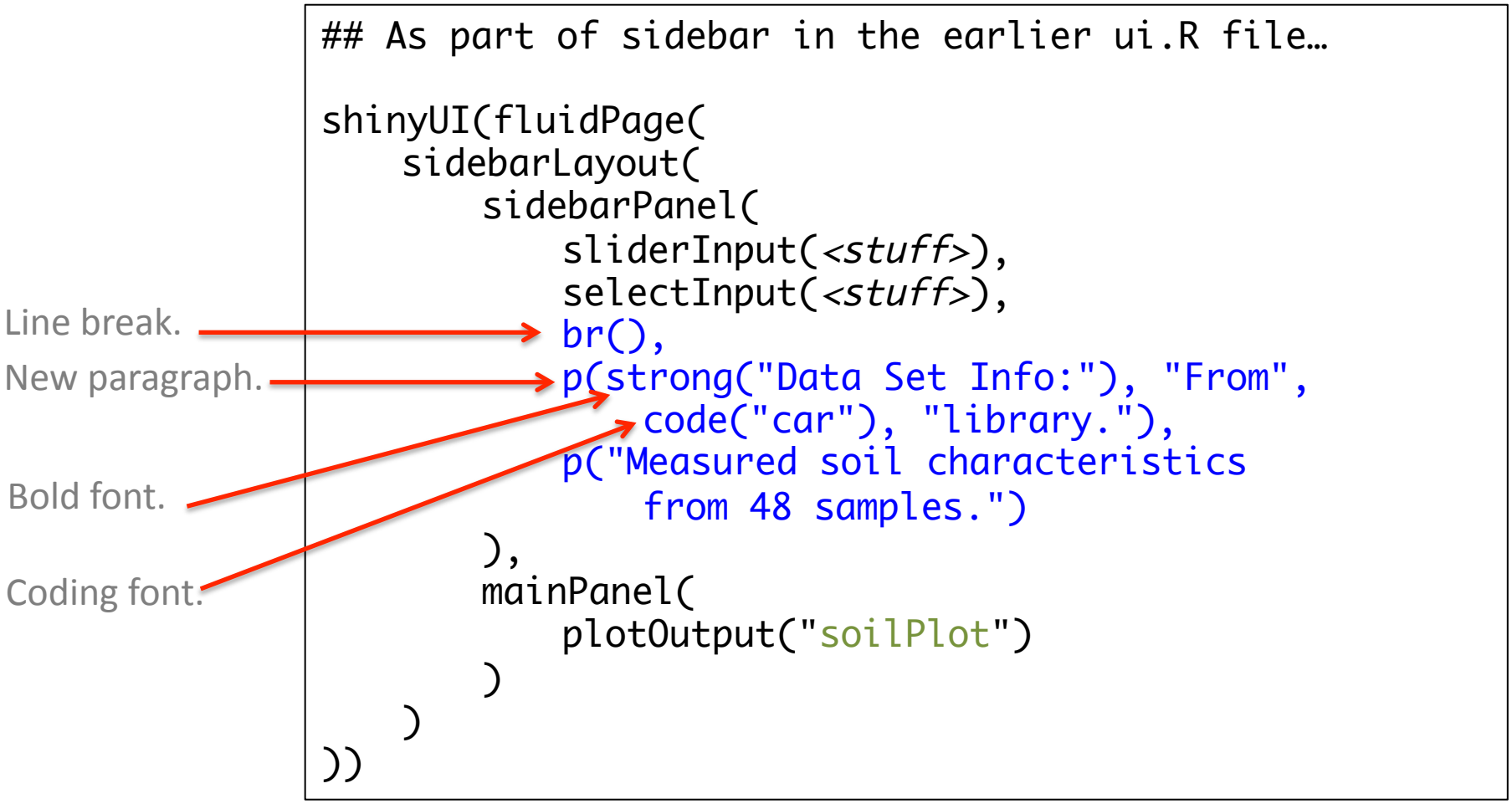
Set user-defined input color/shape in plot (these are all **factors** in Soils).

Set the plot dimensions.

Generate the plot based on user-inputs "chosen" and "size" from web app.

Providing extra help text in your Shiny app, use the `ui.r` file.

(If you want to include information for the user)




```
## As part of sidebar in the earlier ui.R file…

shinyUI(fluidPage(
    sidebarLayout(
        sidebarPanel(
            sliderInput(<stuff>),
            selectInput(<stuff>),
            br(),
            p(strong("Data Set Info:"), "From",
              code("car"), "library."),
            p("Measured soil characteristics
                from 48 samples.")
        ),
        mainPanel(
            plotOutput("soilPlot")
        )
    )
))
```

Hello Shiny 3!

Point size:

0.5 2 4

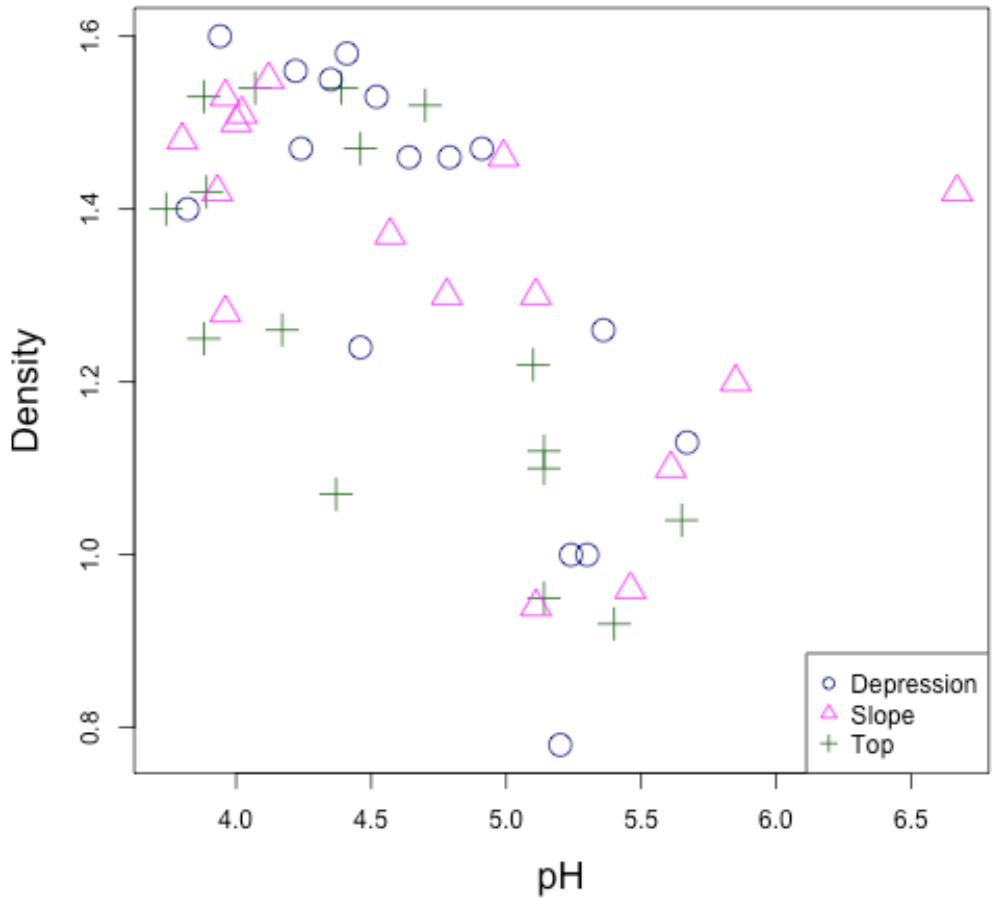
0.5 0.85 1.2 1.55 1.9 2.25 2.6 2.95 3.3 3.65 4

Color/Shape Factor

Contour

Data Set Info: From `car` library.

Measured soil characteristics from 48 samples.

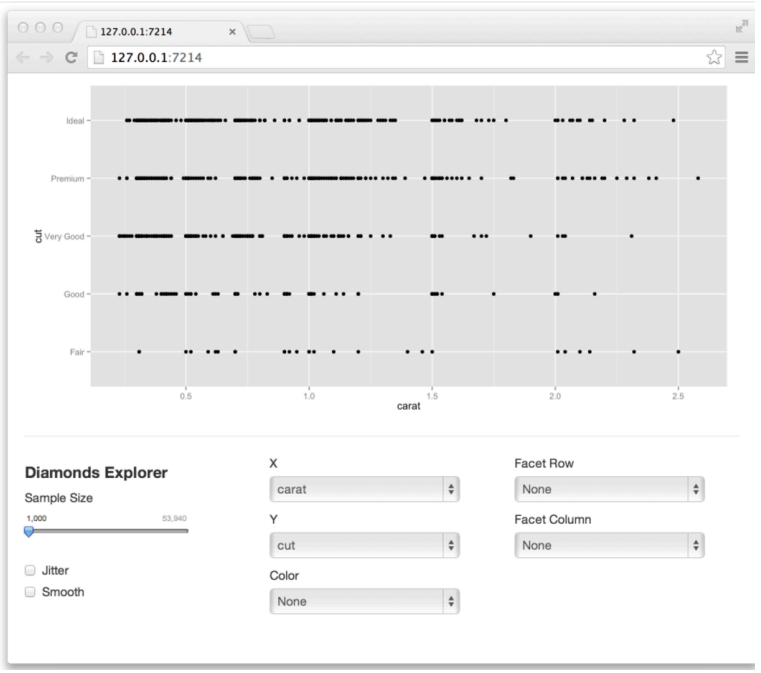


A few other options

R shiny provides 11 specific examples each highlighting a certain ability:

```
> runExample()

Valid examples are "01_hello", "02_text",
"03_reactivity", "04_mpg", "05_sliders", "06_tabsets",
"07_widgets", "08_html", "09_upload", "10_download",
"11_timer"
```

Different page layout other than sidebar/mainplot setting rows and columns:

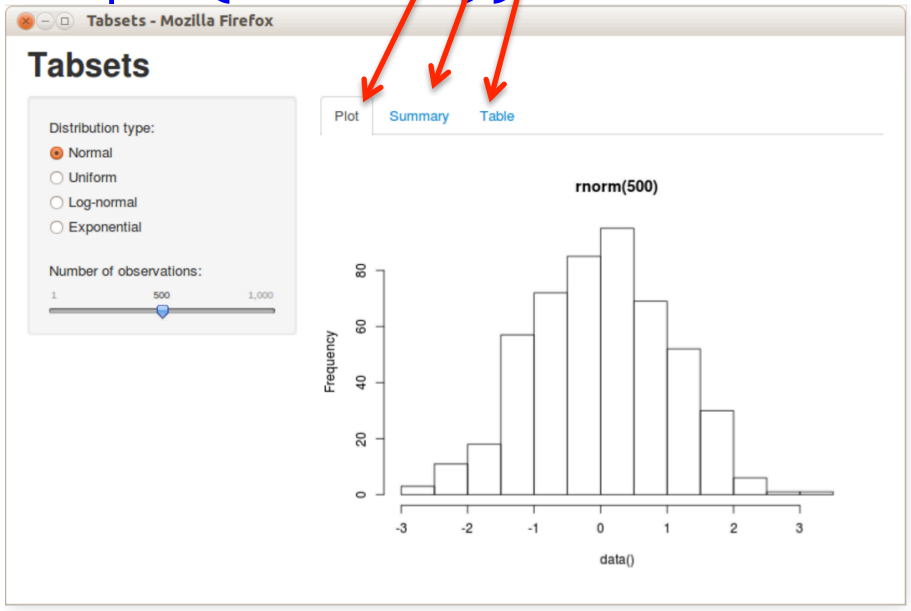


See R studio website gallery:
https://shiny.rstudio.com/gallery/

A few other options

`tabsetPanel(…)` and `tabPanel(…)` can be used to allow multiple pages within the same app.

```
mainPanel(
    tabsetPanel(
      tabPanel("Plot", plotOutput("plot")),
      tabPanel("Summary",
         verbatimTextOutput("summary")),
      tabPanel("Table", tableOutput("table"))
    )
  )
```



https://shiny.rstudio.com/articles/tabsets.html

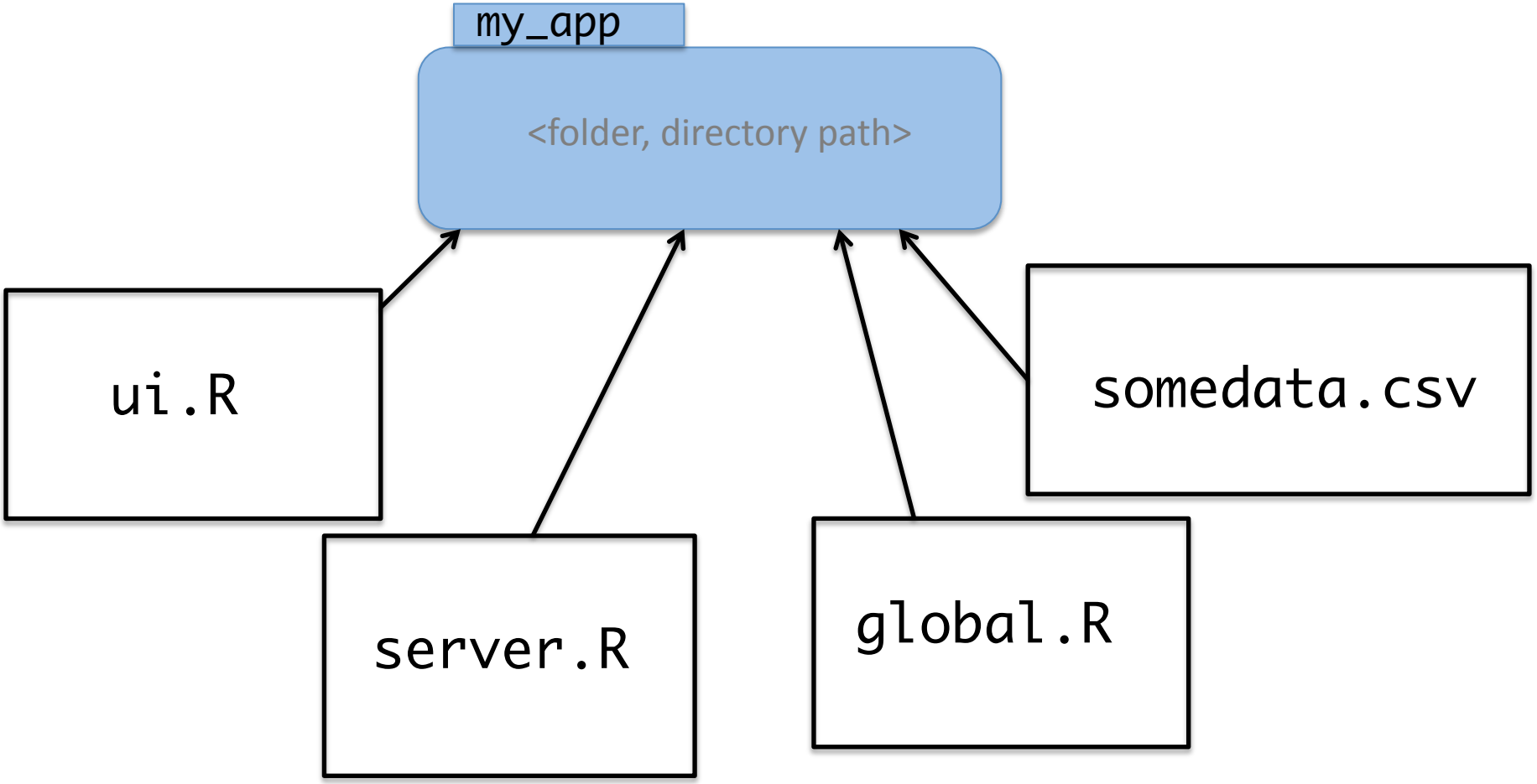
Exiting your Shiny app

NOTE: Your R session will be busy while running a Shiny app, so you will not be able to run any R commands while the Shiny app is running. R is monitoring the app and executing the app's reactions.

```
> runExample("01_hello")

Listening on http://128.255.147.7
```

To get your R session back, **hit escape** or, if using RStudio, click the stop sign icon (found in the upper right corner of the RStudio console panel).

Folder/File structure for R shiny app if you have a data set to read-in and/or manipulate prior to use.



If you have a data file to be used for the shiny app, put it in the app folder. To read the data, we will include the `read.csv()` command in a file called `global.R`, which will also be in the app folder. Shiny automatically knows to run `global.R` once upon launch of the app. See the next slide for a `global.R` example file.

An example of a `global.R` file

```
data <- read.csv("somedata.csv")

library(plyr)  ## For mapvalues() function below.

## Original factor levels were not in right order:
data$nPeople<-mapvalues(data$nPeople,
    from=levels(data$nPeople),
    to=c("1","2","3","4","5","6","7", "8", "9","10","15+"))

## Change var type of numerically coded factors:
data$Subject <- as.factor(data$Subject)


## Define global variables:
factnames <- names( Filter( is.factor, data ) )
numnames <- names( Filter( is.numeric, data ) )
```

After launching the app, both the `ui.R` and `server.R` will have access to the object 'data' and any of the global variables that were defined here.

NOTE: it is possible to include the `read.csv()` command at the top of the `ui.R` and `server.R` files, but I find using the `global.R` file better practice.