U.S. ARMY EVALUATION CENTER

Creating Shiny Apps in R for Sharing Automated Statistical Products

Randy Griffiths

Goal

1. Understand basic structure of Shiny app code
2. Produce simple apps
3. Feel confident that you can create more complicated apps

Agenda

Assumptions & Disclaimers

What is a Shiny app?

How to make a Shiny app

Your turn

Assumptions

R and R Studio are installed on your computer

Working knowledge of R language

Never made a Shiny application

Miscellaneous

`install.packages(“shiny”)` if you have not already done so

Get to know your neighbor

- You may need a friend as you work problems

Disclaimers

- I am not a computer scientist
  - I don’t write elegant or efficient code
- I am not an expert with web-page design
  - My knowledge of HTML and using CSS themes is embarrassing
- I have not worked with Shiny for very long
  - I made my first app in December 2016
    - https://aec-doe.shinyapps.io/Reliability_Quick_Calculations/
    - https://aec-doe.shinyapps.io/Likert_plots/

My first Shiny apps are still being used in AEC
If I can do it, so can you

What is a Shiny app?

"Shiny is an R package that makes it easy to build interactive web apps straight from R. You can host standalone apps on a webpage or embed them in R Markdown documents or build dashboards. You can also extend your Shiny apps with CSS themes, htmlwidgets, and JavaScript actions." (https://shiny.rstudio.com/)

When is a Shiny app useful?

You want to share a specific type of analysis with many users with different data or other user input. You can do it in R but don’t trust them to use your code.

How to make a Shiny app
Where to begin

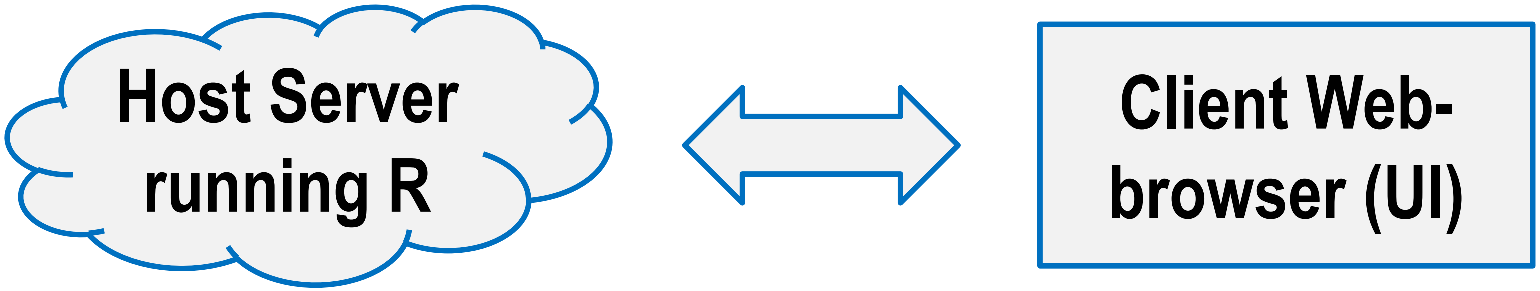
Shiny website has a lot of free training material and a lot of code.

- https://shiny.rstudio.com/
- Many examples with executable code
- Widget Gallery (under Widgets) is extremely useful

Understand

- User interface (UI) and server functions

What is a Shiny app?



Host Server running R

Client Web-browser (UI)

Receives input from UI for computing (math, plots, etc.)

Creates objects and sends specified output to the UI

Collects input from UI and sends to server for computing

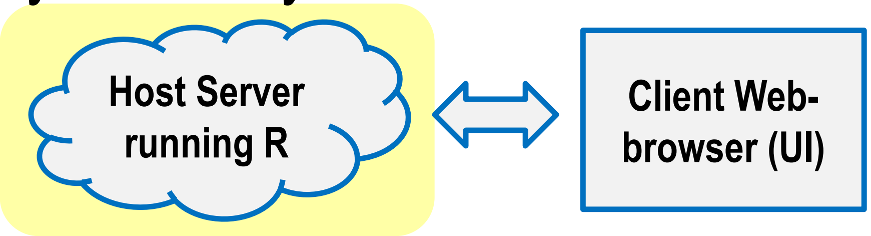
Receives output from server for display

How to make a Shiny app
UI Page Layout

`shinyUI()` wraps the UI content

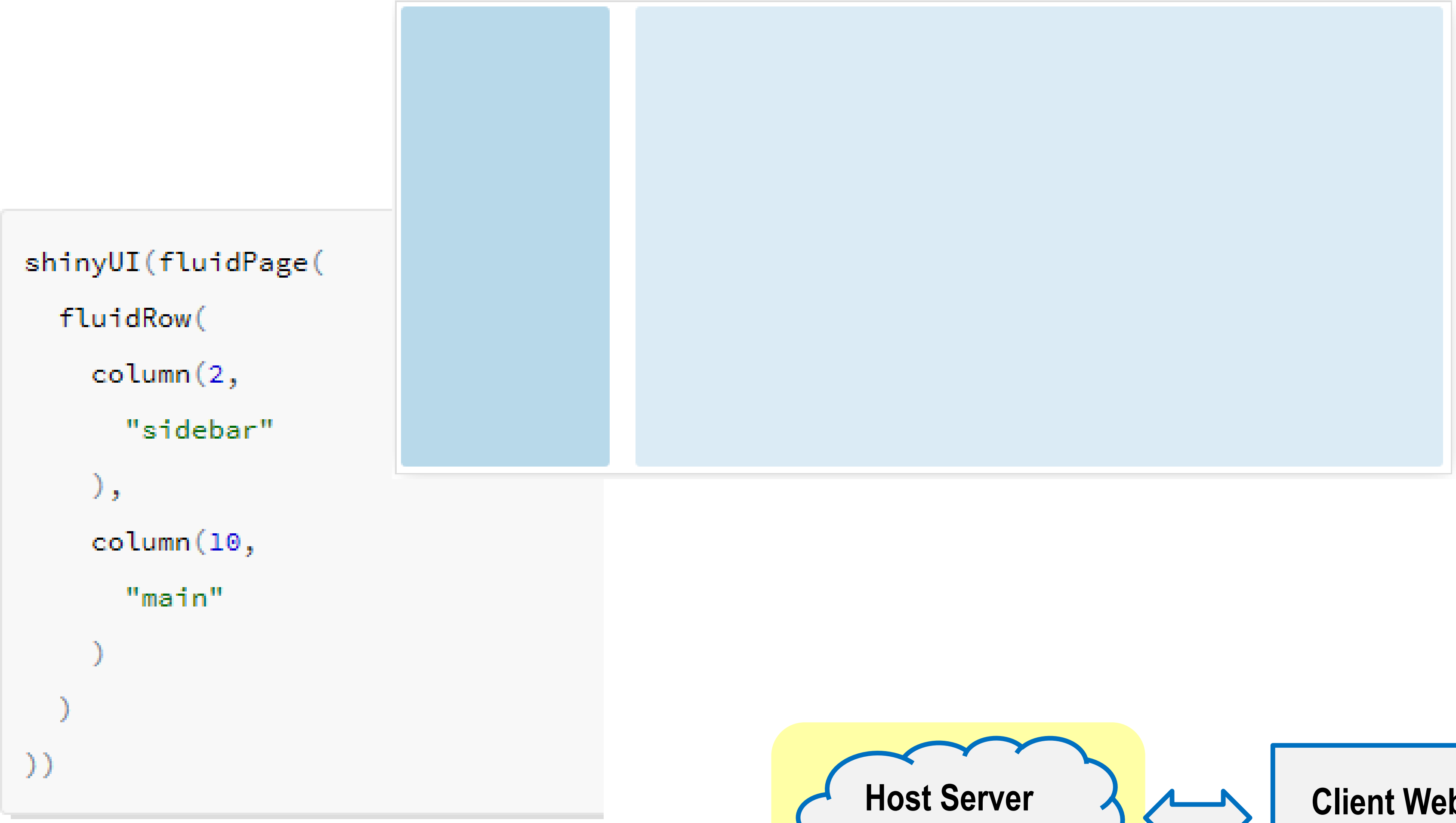
`fluidPage()` is the most flexible page layout

- You can split your “page”, or UI, into rows which can have columns
- I’ve never needed more rows than I was allowed
- Column widths are defined on a 12 point scale within a row
- Actual width will be scaled dynamically to fill the browser’s width

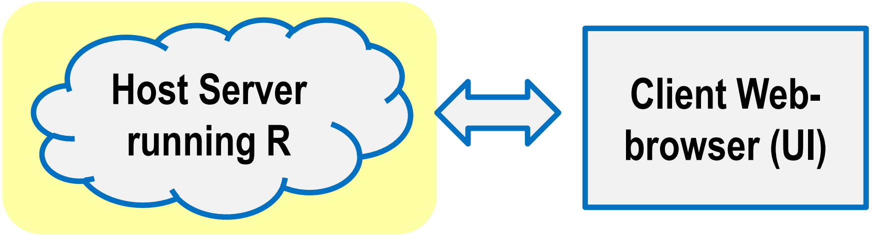


How to make a Shiny app

UI Page Layout



```
shinyUI(fluidPage(
  fluidRow(
    column(2,
      "sidebar"
    ),
    column(10,
      "main"
    )
  )
))
```



How to make a Shiny app

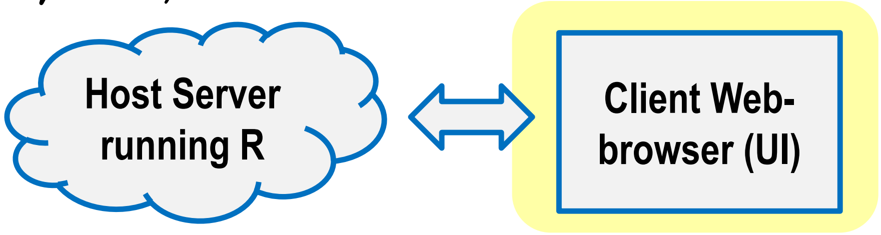
UI Page Layout

UI sends objects to the server via a list called “input”

Shiny has many functions that create objects automatically and adds them to the input list which can be called on by the server as `input$name`

The widgets page has many useful examples

- `sliderInput(“name”, …)`
- `radioButtons(“name”, …)`
- `fileInput(“name”, …)`
- `numericInput(“name”, …)`



Host Server running R

Client Web-browser (UI)

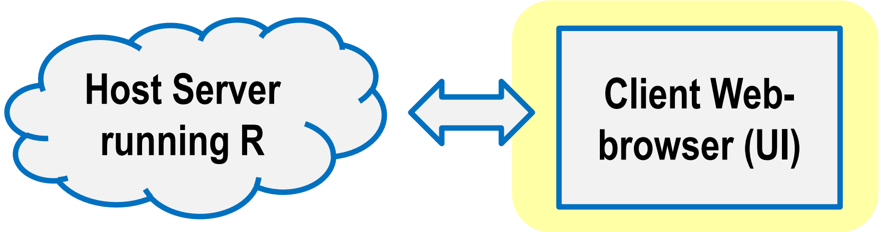
How to make a Shiny app
Server receives UI input

`shinyServer()` includes the content sent to the UI and accepts a function as the argument. It is typically used with a function which takes two parameters: lists named input and output

```
shinyServer(function(input, output){
   code
})
```

Defining new output elements to be sent to UI is done by

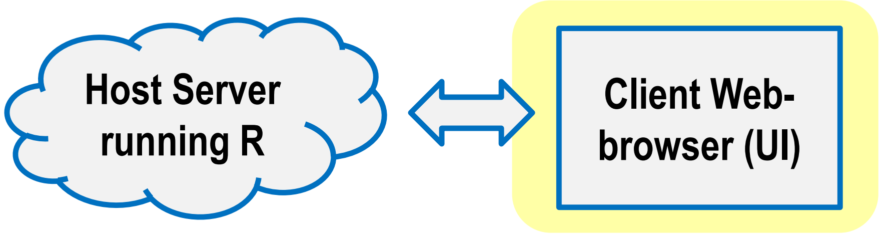
- `output$name <- stuff`



How to make a Shiny app
Server using input from UI

Functions we will use to output results to UI include

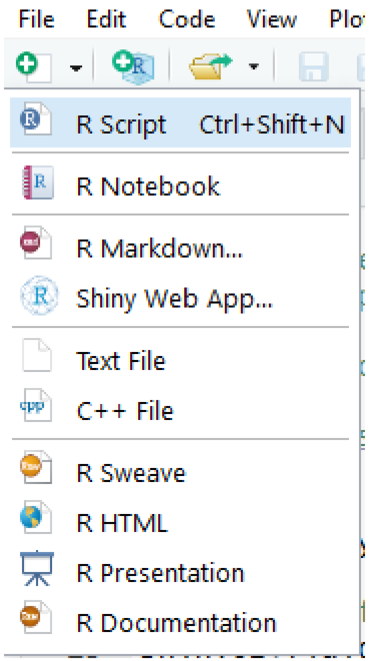
- `ouput$plotName <- renderPlot({})`
- `output$tableName <- renderTable({})`

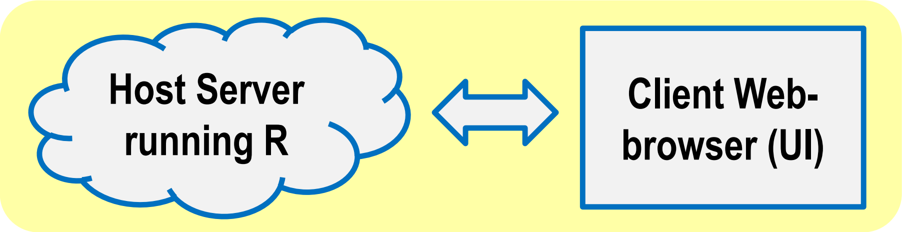


How to make a Shiny app
Getting started

Using R-Studio

- Open new shiny Web App file to see an app structure
- Easier to develop apps using separate ui and server files (choose that option)
- ui.R structures the html file based on layout functions and receives objects from server.R
- server.R does the "R stuff" and receives objects from ui.R (user inputs)
- Passing objects between server and ui creates an interactive web app



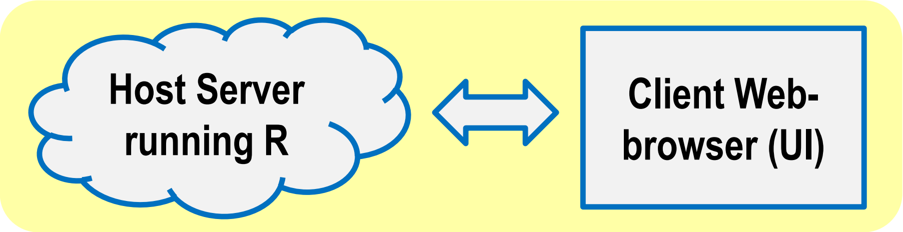


How to make a Shiny app

Getting started

Add user options to the first column

- What are the histogram inputs that you typically manipulate?
- Look at the Widget code on the shiny app site for ideas
  - https://shiny.rstudio.com/gallery/widget-gallery.html
- Recommendation: check box for auto-saving your files before each build



How to make a Shiny app
Make a plot from user input

Practice

Provide a plot of random data from a distribution chosen by the user.

Useful code:

`fluidPage()` divide your page into two sections: input from user and the plot

`fluidRow(column(2, …), column(10,…))` defines a row with 2 columns of relative widths 2 and 10

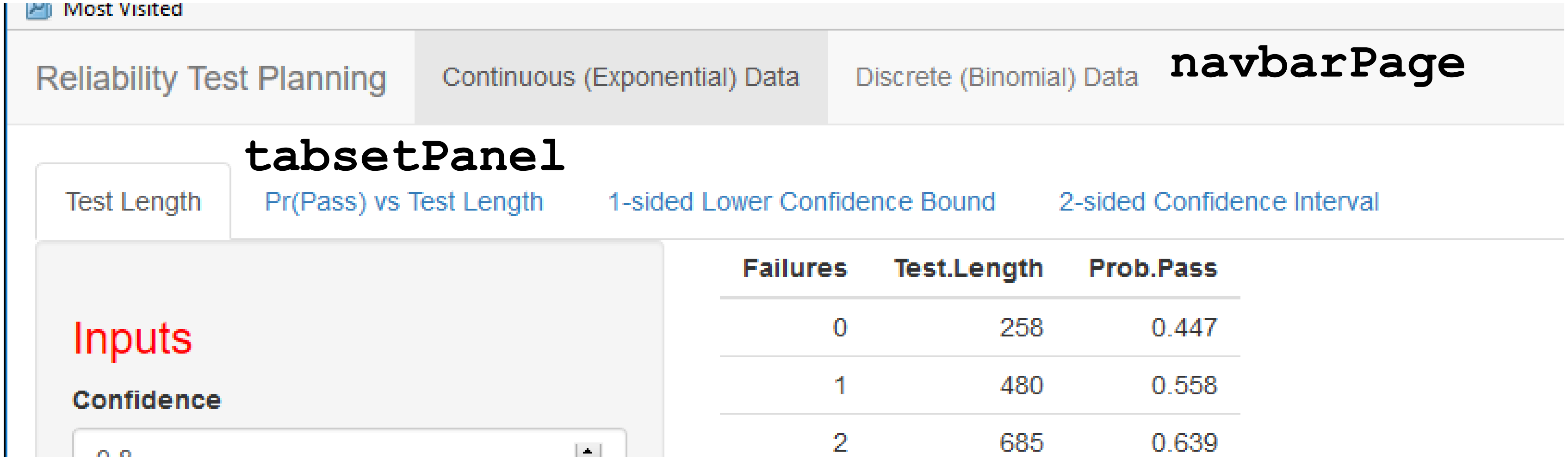
`br()` is a vertical break

```
if(input$dist == 'norm'){
      x <- rnorm(100)
    } else if(input$dist
== 'exp'){
      x <- rexp(100)
    } else {
      x <- rlnorm(100)
    }
```

Adding Complexity

Before long you notice your app is getting very cluttered. What can you do?

- Add tabs along the top of page with `navbarPage()`
- Add tabs within a page with `tabsetPanel()`
- Look into `shinydashboard` for more complicated structure


Most Visited
Reliability Test Planning
Continuous (Exponential) Data
Discrete (Binomial) Data
Test Length
Pr(Pass) vs Test Length
1-sided Lower Confidence Bound
2-sided Confidence Interval

navbarPage

tabsetPanel

Inputs

Confidence

Failures	Test.Length	Prob.Pass
0	258	0.447
1	480	0.558
2	685	0.639

Adding Complexity

Using your existing app, add 3 tabs

1. Plot (already coded)
2. Data table
   - `output$table <- renderTable()` with `tableOutput(“table”)`
3. Table of summary statistics
   - `renderPrint(summary(data))` with `verbatimTextOutput()`

```
tabsetPanel(
  tabPanel(“Tab 1”, …),
  tabPanel(“Tab 2”, …),
  tabPanel(“Tab 3”, …)
)
```

Extra Credit

```
reactive({})
```

Advice

Develop R code for a specific product first

- You don’t want to be troubleshooting shiny code and complex code for output at the same time
- Better to ensure code works well for specific output outside of Shiny, then wrap that code in a Shiny app

“Run App” as often as possible

- Don’t write a lot of code then try to run the app. It will likely be painful to debug