

Lecture 4. Introduction to R Shiny

PUBH 6199: Visualizing Data with R, Summer 2026

Xindi (Cindy) Hu, ScD

2026-06-09



Outline for today

- **What is R Shiny**
- Write a basic R shiny app
- Deploy your Shiny apps with shinyapps.io
- AI-powered help: using Shiny Assistant



About R Shiny

- Shiny is an R package that makes it easy to build interactive web applications (apps) straight from R
- Shiny allows users to build dynamic, data-driven web apps without requiring extensive knowledge of web development
- To install **R Shiny** run the following command in the console of your RStudio
`install.packages("shiny")`



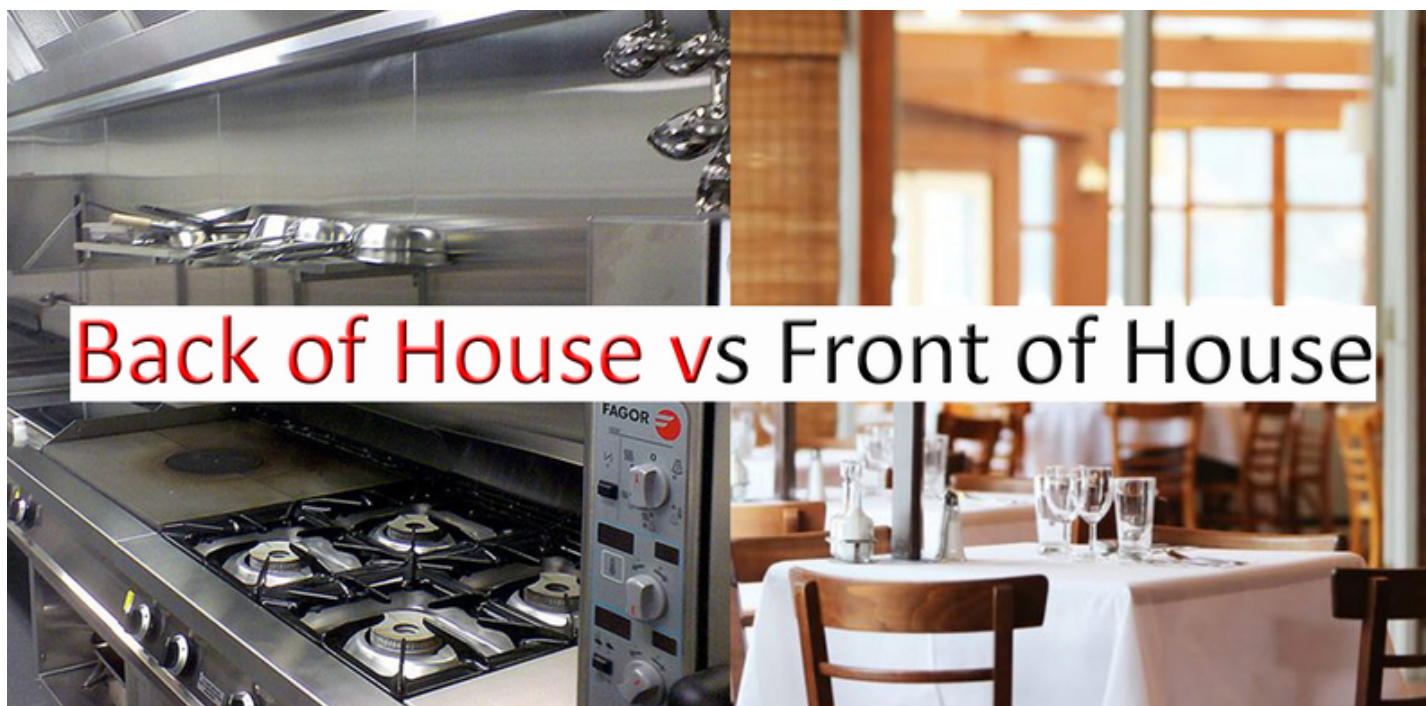
Key Features

- **Interactive Web Applications:** Shiny makes web apps that respond to user inputs
- **Seamless Integration with R:** Users can leverage the full power of R, like data manipulation, statistical modeling, and plotting
- **Easy Deployment** using Shiny Cloud
- **Extensibility:** Shiny apps can be enhanced with custom HTML, CSS, and JavaScript



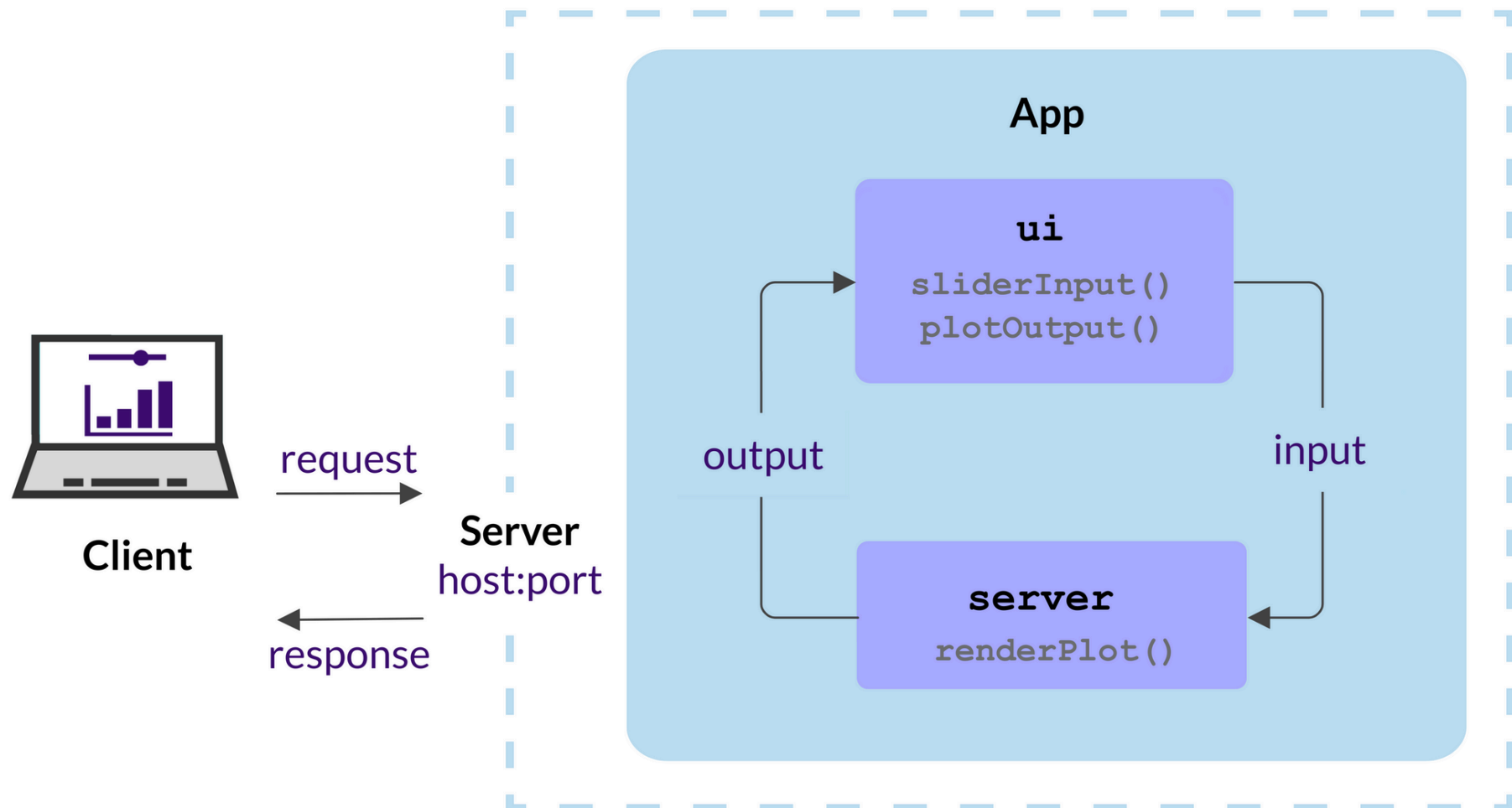
Basic Structure Shiny Application

- A typical Shiny application has two main components:
 - **User Interface (UI):** Defines the layout and appearance of the app, including input controls (like sliders and text boxes) and output displays (like plots and tables).
 - **Server Function:** Contains the logic that processes inputs and generates outputs. It reacts to user interactions



Visual Logic

- Here we can see the visual logic behind a Shiny application



*Input(), *Output() and render*()

These three groups of functions are the core of Shiny's reactive programming model:

- ***Input()**: Represents user inputs, such as text input, slider input, or checkbox input. For example, `input$sliderValue` retrieves the value of a slider input with the ID "sliderValue".
- ***Output()**: Represents outputs that are displayed in the UI, such as plots, tables, or text. For example, `output$histPlot` refers to a plot output with the ID "histPlot".
- **render*()**: Functions that generate outputs based on inputs. For example, `renderPlot()` creates a plot output, and `renderText()` generates text output. These functions are reactive and will automatically update when the inputs change.



*Input () function syntax

Shiny input functions: `sliderInput()`, `selectInput()`, `textInput()`
`numericInput()`, etc.

General format:

```
1 inputFunction(inputId, label, value, ...)
```

- **inputId**: connects UI and server via `input$inputId`, must be a simple string without spaces, must be unique within the app
- **label**: text label for the input control, displayed in the UI
- **value**: initial value of the input control, can be a number, character, or logical value. For example:

```
1 # use position for inputId and label
2 # use named args for the rest
3 sliderInput("min", "Limit (minimum)", value = 50, min = 0, max = 100)
```



Different types of UI Inputs

Inputs

collect values from the user

Access the current value of an input object with **input\$<inputId>**. Input values are **reactive**.

Action

actionButton(inputId, label, icon, ...)

Link

actionLink(inputId, label, icon, ...)

Choice 1

checkboxGroupInput(inputId, label, choices, selected, inline)

Choice 2

Choice 3

Check me

checkboxInput(inputId, label, value)



dateInput(inputId, label, value, min, max, format, startview, weekstart, language)



dateRangeInput(inputId, label, start, end, min, max, format, startview, weekstart, language, separator)

Choose File

fileInput(inputId, label, multiple, accept)

1

numericInput(inputId, label, value, min, max, step)

.....

passwordInput(inputId, label, value)

Choice A

radioButtons(inputId, label, choices, selected, inline)

Choice B

Choice C

Choice 1

selectInput(inputId, label, choices, selected, multiple, selectize, width, size) (also [selectizeInput\(\)](#))

Choice 1

Choice 2

0 5 10

sliderInput(inputId, label, min, max, value, step, round, format, locale, ticks, animate, width, sep, pre, post)

0 2 4 6 8 10

Apply Changes

submitButton(text, icon)
(Prevents reactions across entire app)

Enter text

textInput(inputId, label, value)

Source: [Shiny for R cheatsheet](#)

PUBH 6199: Visualizing Data with R



*Output() function syntax

Shiny output functions: `plotOutput()`, `tableOutput()`, `textOutput()`, etc.

What *Output() function does:

- Output functions in the UI create placeholders.
- These placeholders are filled by the server using a matching `render*()` function.

General format:

```
1 outputFunction(outputId)
```

- `outputId`: connects UI and server via `output$outputId`, must be a simple string without spaces, must be unique within the app. For example:

```
1 plotOutput("scatterplot")
```

- Accessed in server as:

```
1 output$outputId <- renderFunction({...})
```



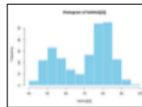
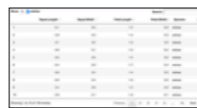
Output <-> Render pairing

```

1 # UI
2 plotOutput("my_plot")
3 # Server
4 output$my_plot <- renderPlot({
5   ggplot(data, aes(...)) + geom_point()
6 })

```

Outputs - render*() and *Output() functions work together to add R output to the UI



```

"data.frame": 3 obs. of 2 variables:
 $ Sepal.Length: num  5.1 4.9 4.7
 $ Petal.Length: num  1.6 1.4 1.5

```

Species	Sepal.Length	Petal.Length	Species
setosa	4.6	1.2	setosa
setosa	5.0	1.6	setosa
setosa	5.4	1.7	setosa
setosa	4.7	1.2	setosa
setosa	4.7	1.3	setosa
setosa	4.8	1.6	setosa
setosa	4.9	1.4	setosa
setosa	4.8	1.5	setosa
setosa	4.7	1.4	setosa
setosa	4.9	1.5	setosa

foo



dataTableOutput(outputId, icon, ...)

DT::renderDataTable(expr, options, callback, escape, env, quoted)

renderImage(expr, env, quoted, deleteFile)

renderPlot(expr, width, height, res, ..., env, quoted, func)

renderPrint(expr, env, quoted, func, width)

renderTable(expr, ..., env, quoted, func)

renderText(expr, env, quoted, func)

renderUI(expr, env, quoted, func)

imageOutput(outputId, width, height, click, dblclick, hover, hoverDelay, inline, hoverDelayType, brush, clickId, hoverId)

plotOutput(outputId, width, height, click, dblclick, hover, hoverDelay, inline, hoverDelayType, brush, clickId, hoverId)

verbatimTextOutput(outputId)

tableOutput(outputId)

textOutput(outputId, container, inline)

& uiOutput(outputId, inline, container, ...) & htmlOutput(outputId, inline, container, ...)

Source: Shiny for R cheatsheet

PUBH 6199: Visualizing Data with R



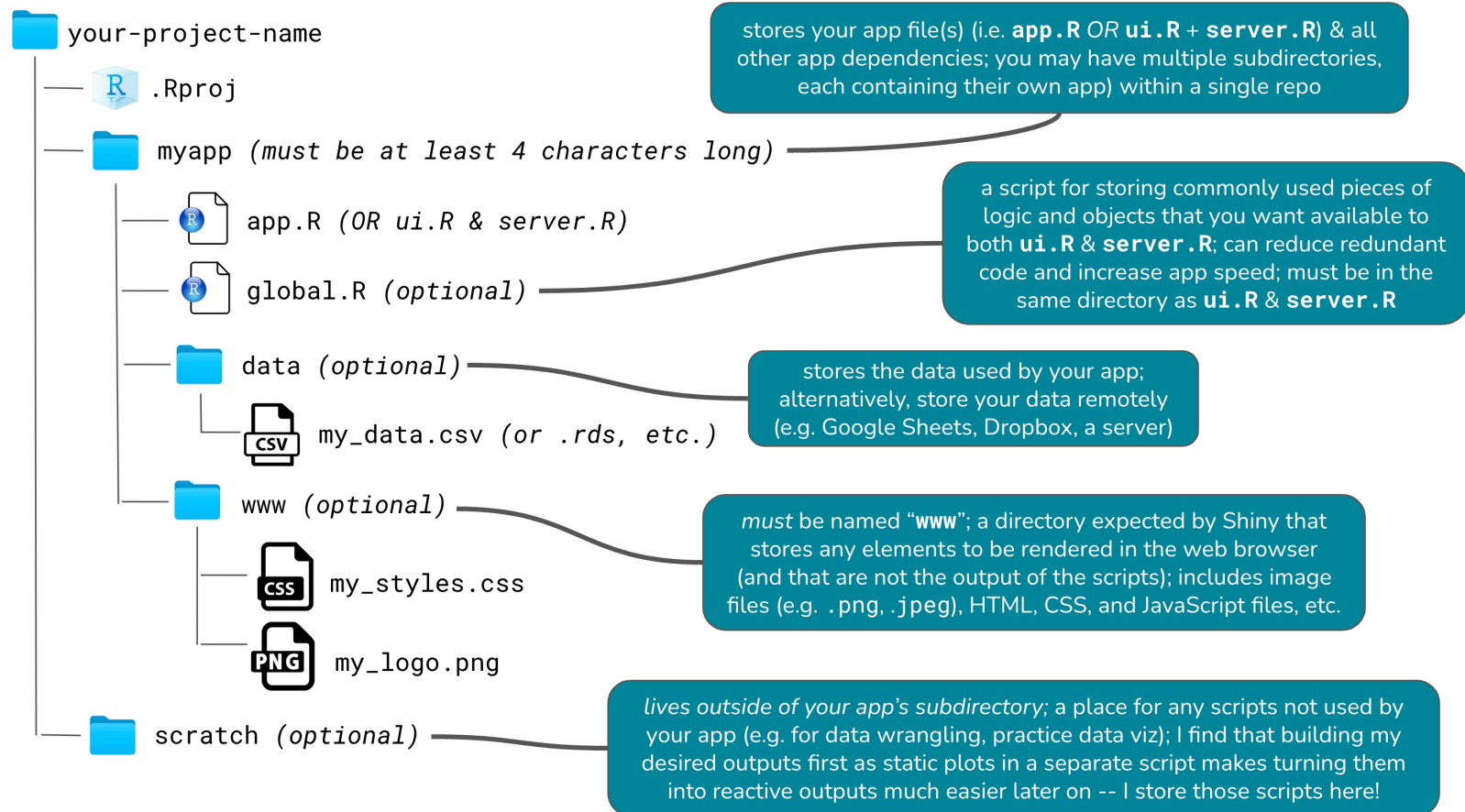
Outline for today

- What is R Shiny
- **Write a basic R shiny app**
- Deploy your Shiny apps with shinyapps.io
- AI-powered help: using Shiny Assistant

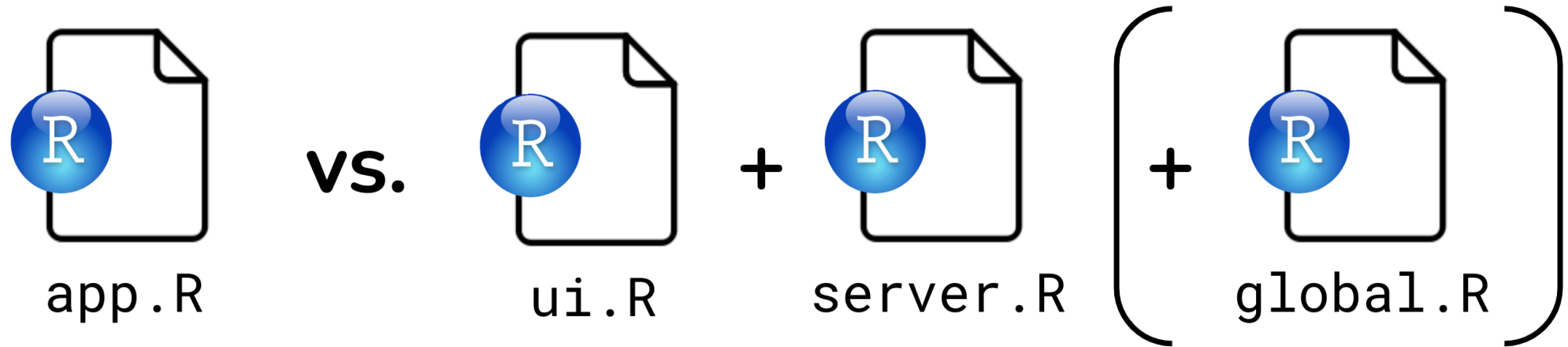


Setting up a shiny app

- Create a GitHub repo and clone the repo to our computer
- Install R Shiny in your R environment : `install.packages("shiny")`
- Organize repo structure



Two options for shiny apps



- **Single-File App:** All code is in one file, typically named `app.R`. This is suitable for simple applications or creating a [reprex](#) example.
- **Two-File App:** Code is split into multiple files, usually `ui.R` for the user interface and `server.R` for the server logic. Optionally, `global.R` can be used for data ingestion and wrangling. This is better for larger applications.

Create a single-file Shiny app

- Create a new R script file named `app.R`
- The file should contain both the UI and server components
- The basic structure of a single-file Shiny app is as follows:

```
1 # load packages ----
2 library(shiny)
3
4 # user interface ----
5 ui <- fluidPage()
6
7 # server instructions ----
8 server <- function(input, output) {}
9
10 # combine UI & server into an app ----
11 shinyApp(ui = ui, server = server)
```

Once you have this structure, RStudio recognizes this as a R shiny app and you can run the app by clicking the “Run App” button in RStudio.

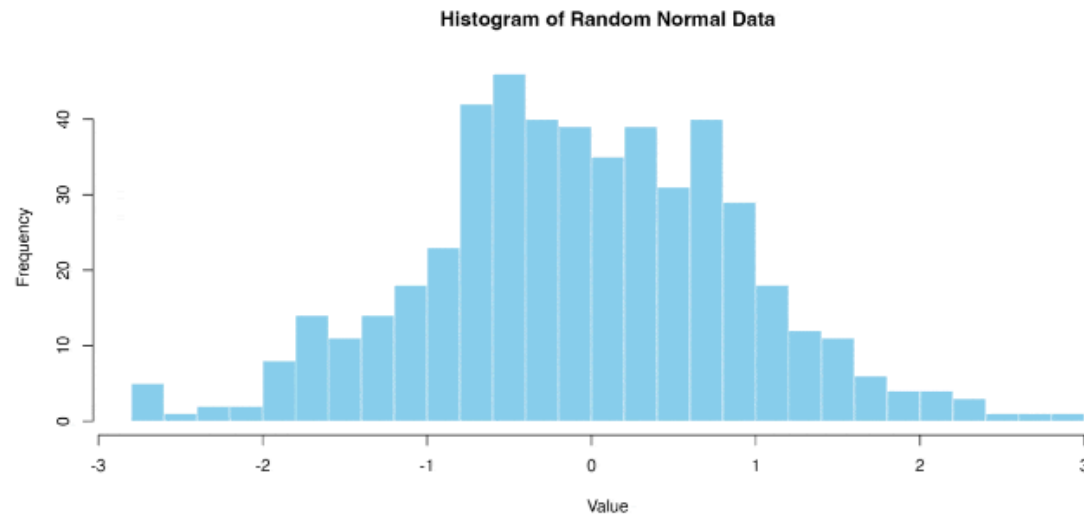
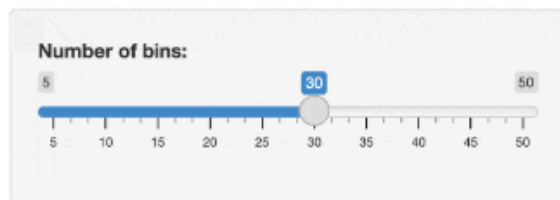


Let's build the app step-by-step

- We will create a simple Shiny app that displays a histogram of random normal data
- The app has a title and subtitle
- Users can adjust the number of bins in the histogram using a slider input
- The histogram will react to the slider input and update accordingly

Histogram Example

Choose a number of bins to update the histogram



Code Breakdown UI

Calling the library

```
1 library(shiny)
```

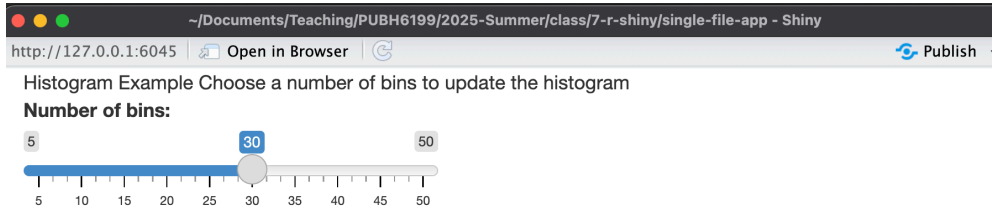
Define the User Interface (UI) of the app, consisting of a title, a subtitle, a slider input, and a plot output

```
1 ui <- fluidPage(  
2   # app title  
3   "Histogram Example",  
4   # app subtitle  
5   "Choose a number of bins to update the histogram",  
6   # slider input  
7   sliderInput("bins",  
8               "Number of bins:",  
9               min = 5, max = 50, value = 30),  
10  # histogram output  
11  plotOutput("histPlot")  
12 )
```

Click [Run App](#), what do you see? what is different from what you imagined?



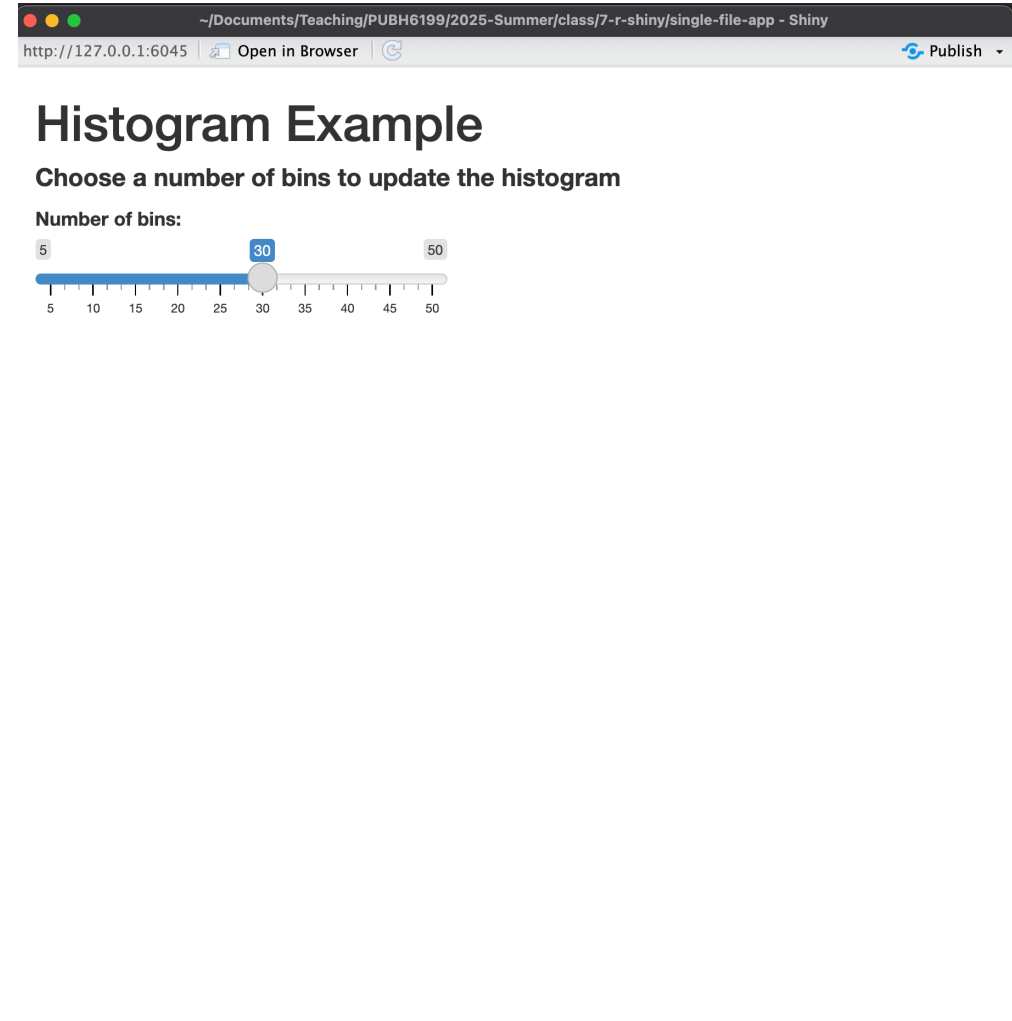
UI Work-in-Progress



1. There is no distinction between title and subtitle
2. There is no plot

Adding hierarchy to the text

```
1 ui <- fluidPage(  
2   # app title  
3   h1("Histogram Example"),  
4   # app subtitle  
5   h4(strong("Choose a number of bins to update the h  
6   # slider input  
7   sliderInput("bins",  
8     "Number of bins:",  
9     min = 5, max = 50, value = 30),  
10  # histogram output  
11  plotOutput("histPlot")  
12 )
```



Code Breakdown: Server

Server takes the inputs and create a reactive chart

```
1 server <- function(input, output) {  
2  
3   output$histPlot <- renderPlot({  
4     # Generate random data  
5     data <- rnorm(500)  
6  
7     # Create histogram with user-specified bins  
8     hist(data, breaks = 30, col = "skyblue",  
9           border = "white", main = "Histogram of Random Normal Data",  
10            xlab = "Value", ylab = "Frequency")  
11   })  
12 }  
13  
14 shinyApp(ui = ui, server = server)
```



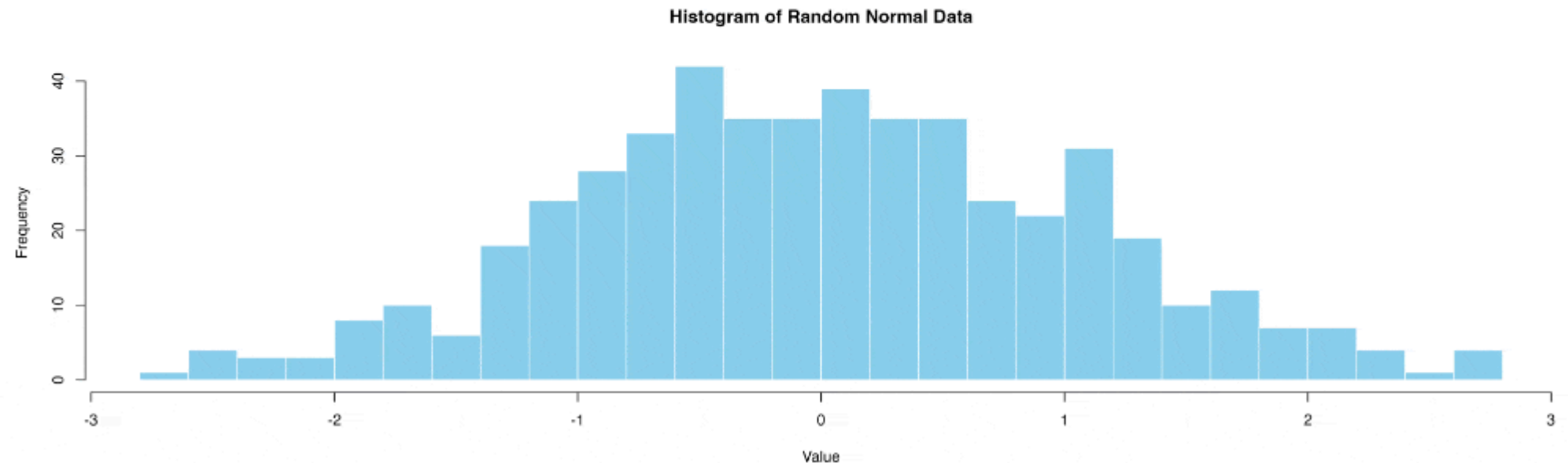
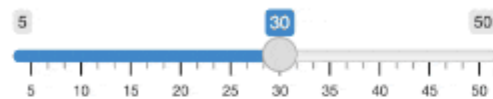
Running the App

- Save the code to an R script file, e.g., app.R.
- In your R console, run `shiny::runApp("app.R")`

Histogram Example

Choose a number of bins to update the histogram

Number of bins:



How to make it reactive?

This will not react to user input because the number of bins is always 30.

```
1 server <- function(input, output) {
2   output$histPlot <- renderPlot({
3     # Generate random data
4     data <- rnorm(500)
5     # Create histogram with user-specified bin
6     hist(data, breaks = 30, col = "skyblue",
7           border = "white", main = "Histogram of",
8           xlab = "Value", ylab = "Frequency")
9   })
10 }
11
12 shinyApp(ui = ui, server = server)
```

This takes in user input and uses it to create a histogram with the specified number of bins.

```
1 server <- function(input, output) {
2   output$histPlot <- renderPlot({
3     # Generate random data
4     data <- rnorm(500)
5     # Create histogram with user-specified bin
6     hist(data, breaks = input$bins, col = "sky
7           border = "white", main = "Histogram of
8           xlab = "Value", ylab = "Frequency")
9   })
10 }
11
12 shinyApp(ui = ui, server = server)
```



This is what we have now

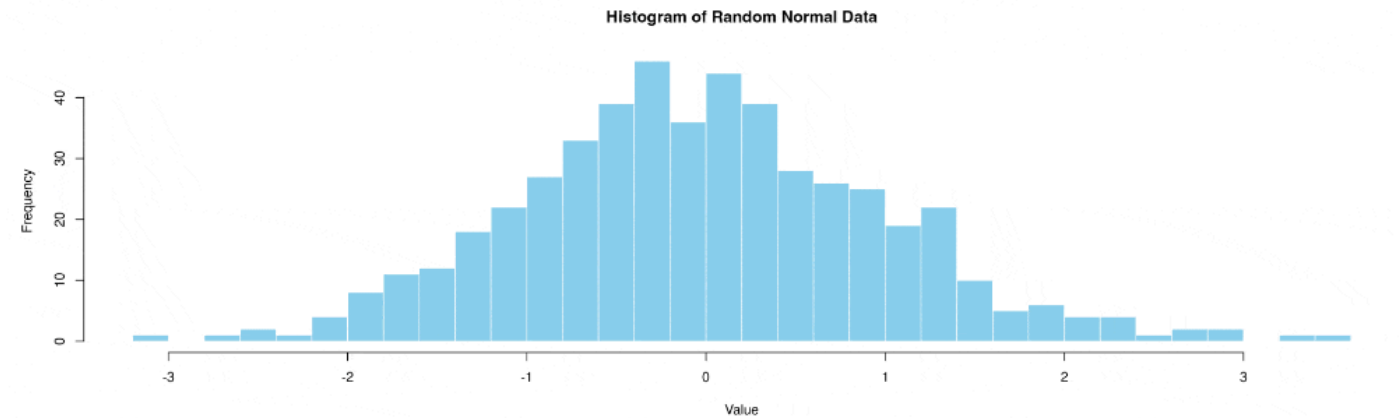
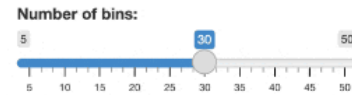
```

1 ui <- fluidPage(
2   # app title
3   h1("Histogram Example"),
4   # app subtitle
5   h4(strong("Choose a number
6   # slider input
7   sliderInput("bins",
8     "Number of bins
9     min = 5, max =
10  # histogram output
11  plotOutput("histPlot")
12 )
13 # Server
14 server <- function(input, out
15
16
17   output$histPlot <- renderPl
18   # Generate random data
19   data <- rnorm(500)
20
21   # Create histogram with u
22   hist(data, breaks = input
23     border = "white", ma
24     xlab = "Value", ylab
25   })

```

Histogram Example

Choose a number of bins to update the histogram



Adding visual structures

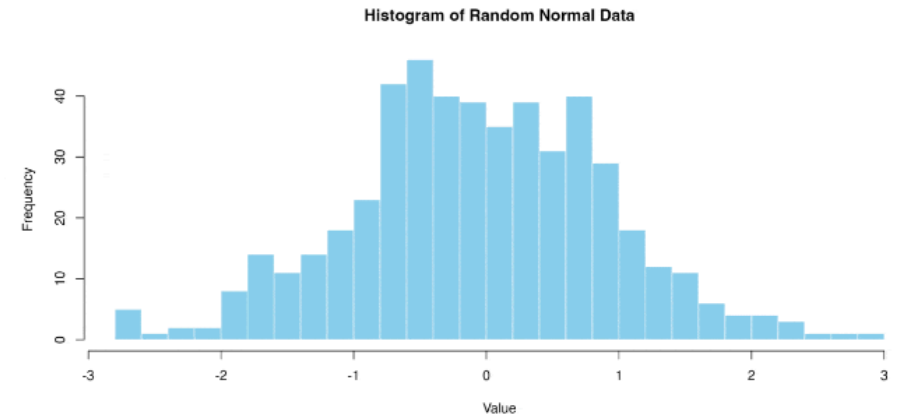
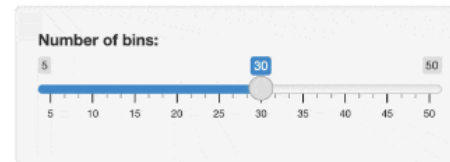
```

1 ui <- fluidPage(
2   titlePanel("Histogram Examp
3   h4(strong("Choose a number
4
5   sidebarLayout(
6     sidebarPanel(
7       sliderInput("bins",
8         "Number of
9         min = 5, ma
10    ),
11
12    mainPanel(
13      plotOutput("histPlot")
14    )
15  )
16 )
17
18 # Server
19 server <- function(input, out
20
21   output$histPlot <- renderPl
22   # Generate random data
23   data <- rnorm(500)
24
25   # Create histogram with u

```

Histogram Example

Choose a number of bins to update the histogram



A top-down view of a desk with a white laptop, a grid-patterned notebook with a silver pen, a white cup of coffee, and a lined notebook with a green pen. The text is overlaid on a semi-transparent white background.

In-Class Activity:

Add a second slider input so that the app reacts to the number of random data points generated for the histogram.

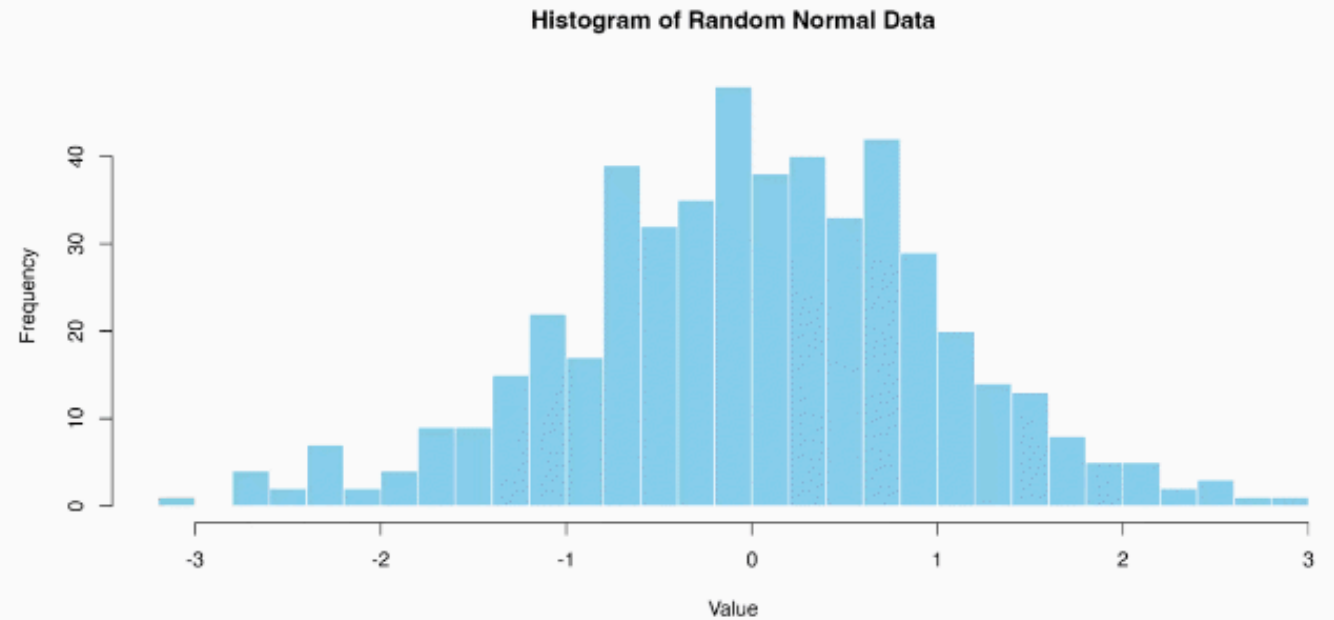
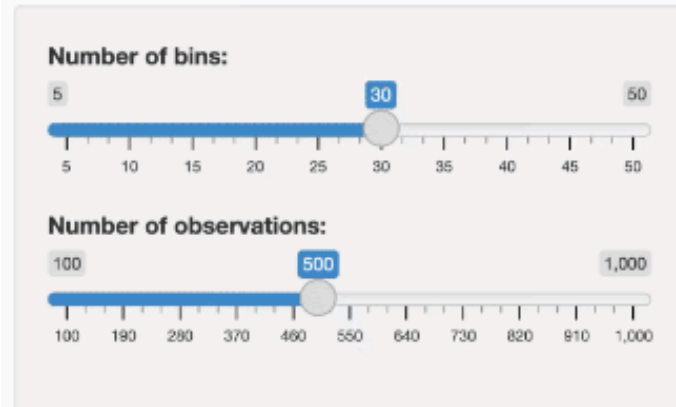
Download starter code from [lecture notes zip file](#)



Finished app with two slider inputs

Histogram Example

Choose a number of bins to update the histogram

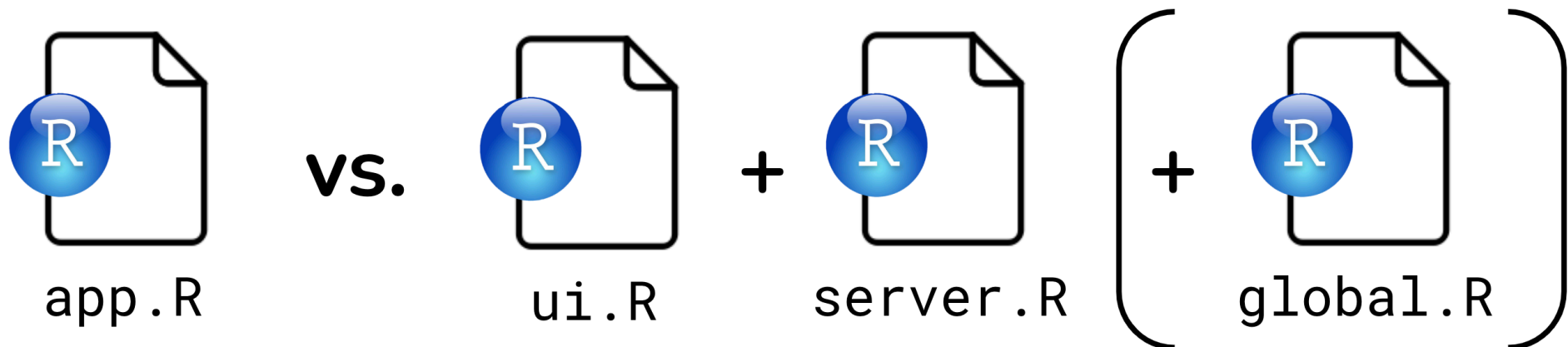


Two-file shiny app with a dataset

In this tutorial, we will build an interactive R Shiny application step-by-step.

Our app will:

- Display a scatter plot using the `mtcars` dataset
- Allow users to choose x-axis and y-axis variables
- Enable plot color customization by a third variable
- Label points with car names



global.R: load libraries, clean data

```

1 library(shiny)
2 library(ggplot2)
3 library(dplyr)
4
5 # Data wrangling: Add a car name column and factor variables
6 mtcars_clean <- mtcars %>%
7   tibble::rownames_to_column("car") %>%
8   mutate(
9     cyl = as.factor(cyl),
10    gear = as.factor(gear)
11  )
12 glimpse(mtcars_clean)

```

Rows: 32

Columns: 12

```

$ car <chr> "Mazda RX4", "Mazda RX4 Wag", "Datsun 710", "Hornet 4 Drive", "Ho...
$ mpg <dbl> 21.0, 21.0, 22.8, 21.4, 18.7, 18.1, 14.3, 24.4, 22.8, 19.2, 17.8,...
$ cyl <fct> 6, 6, 4, 6, 8, 6, 8, 4, 4, 6, 6, 8, 8, 8, 8, 8, 8, 4, 4, 4, 4, 8,...
$ disp <dbl> 160.0, 160.0, 108.0, 258.0, 360.0, 225.0, 360.0, 146.7, 140.8, 16...
$ hp <dbl> 110, 110, 93, 110, 175, 105, 245, 62, 95, 123, 123, 180, 180, 180...
$ drat <dbl> 3.90, 3.90, 3.85, 3.08, 3.15, 2.76, 3.21, 3.69, 3.92, 3.92, 3.92,...
$ wt <dbl> 2.620, 2.875, 2.320, 3.215, 3.440, 3.460, 3.570, 3.190, 3.150, 3...
$ qsec <dbl> 16.46, 17.02, 18.61, 19.44, 17.02, 20.22, 15.84, 20.00, 22.90, 18...
$ vs <dbl> 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0,...
$ am <dbl> 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0,...
$ gear <fct> 4, 4, 4, 3, 3, 3, 3, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 4, 4, 4, 3, 3,...
$ carb <dbl> 4, 4, 1, 1, 2, 1, 4, 2, 2, 4, 4, 3, 3, 3, 4, 4, 4, 1, 2, 1, 1, 2,...

```



ui.R: define user interface

```

1 ui <- fluidPage(
2   titlePanel("Interactive Scatter Plot - mtcars"),
3
4   sidebarLayout(
5     sidebarPanel(
6       selectInput("xvar", "Choose X-axis variable:", choices = names(mtcars), selected = "wt"),
7       selectInput("yvar", "Choose Y-axis variable:", choices = names(mtcars), selected = "mpg"),
8       selectInput("colorvar", "Choose color variable:", choices = names(mtcars), selected = "cyl"),
9     ),
10    mainPanel(
11      plotOutput("scatterPlot")
12    )
13  )
14 )

```

Interactive Scatter Plot - mtcars

Choose X-axis variable:

wt

Choose Y-axis variable:

mpg

Choose color variable:

cyl

```

Elements Console Sources Network >>
<!DOCTYPE html>
<html>
  <head>
    <script>
  </script>
  </head>
  <body data-new-gr-c-s-check-loaded="14.1238.0" data-gr-ext-installed>
    <div class="container-fluid">
      <div class="row">
        <div class="col-sm-4">
          <div class="col-sm-8" role="main">
            <div class="shiny-plot-output html-fill-item shiny-bound-output recalculating" id="scatterPlot" style="width:100%;height:400px;" aria-live="polite">
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
  </body>
</html>

```

Styles Computed Layout Event Listeners DOM Breakpoints Properties >>

Filter show .cls +

```

element.style {
  width: 100%;
  height: 400px;
}
.recalculating {
  --shiny-fade-opacity: var(--shiny-fade-opacity, .3);
  opacity: var(--shiny-fade-opacity);
  transition: opacity .25s ease .5s;
}

```

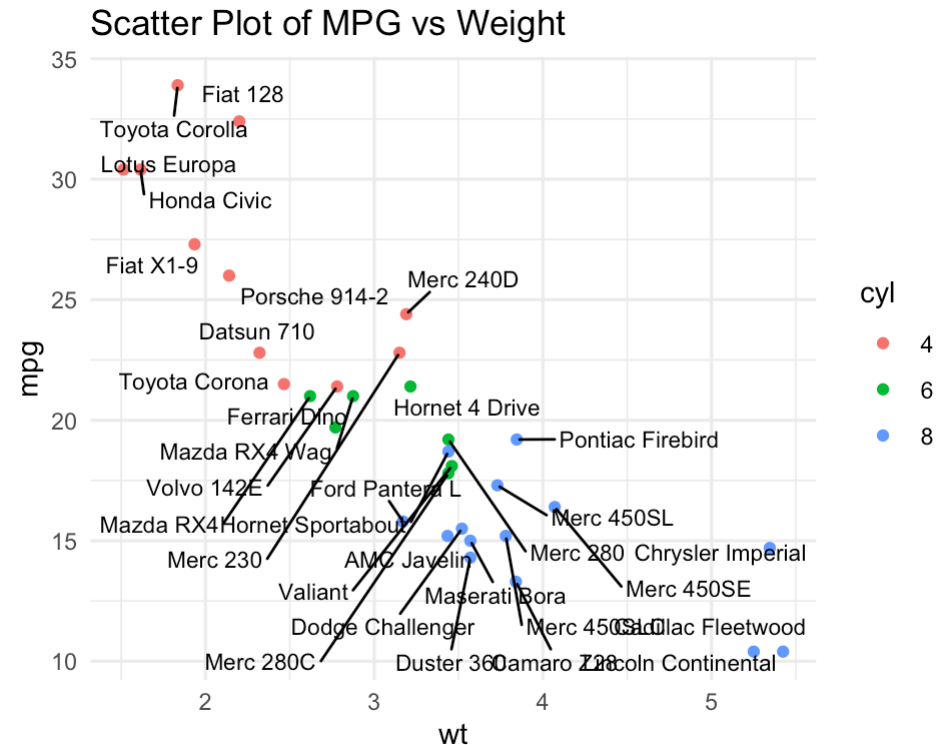
vendor-prefixes, less:77



Intermediate step: make ggplot in `scratch.R`

This is not a must but highly recommend, you can test out your ggplot code in a separate R script file, e.g., `scratch.R`, before integrating it into the Shiny app.

```
1 ggplot(mtcars_clean, aes(x = wt, y = mpg)) +
2   geom_point(aes(color = cyl)) +
3   ggrepel::geom_text_repel(aes(label = car), vjust =
4   labs(
5     title = "Scatter Plot of MPG vs Weight"
6   ) +
7   theme_minimal()
```



server.R: adapt regular ggplot code

scratch.R

```

1 ggplot(mtcars_clean, aes(x = wt, y = mpg)) +
2   geom_point(aes(color = cyl)) +
3   geom_text(aes(label = car), vjust = -1, size = 10) +
4   labs(
5     title = "Scatter Plot of MPG vs Weight"
6   ) +
7   theme_minimal()

```

- Regular ggplot code for scatter plot
- x-axis is wt
- y-axis is mpg
- color by cyl
- label points with car names

server.R

```

1 server <- function(input, output) {
2   # Render the plot
3   output$scatterPlot <- renderPlot({
4     ggplot(mtcars_clean, aes_string(x = input$xvar, y = input$yvar)) +
5       geom_point(aes_string(color = input$colorvar)) +
6       geom_text(aes(label = car), vjust = -1, size = 10) +
7       theme_minimal() +
8       labs(title = paste("Scatter Plot of", input$xvar, "vs", input$yvar))
9   })
10 }
11 }

```

- Adapt the ggplot code to use `aes_string()` for dynamic variable selection
- Use `input$xvar`, `input$yvar`, and `input$colorvar` to make the plot reactive



Run the two-file Shiny app

No need for `shinyApp(ui = ui, server = server)` when you have two-file app structure:

- Save the `ui.R` and `server.R` files **in the same directory**
- Shiny automatically stitch them together, and optionally look for `global.R`.
- In RStudio, click the **Run App** button

Interactive Scatter Plot - mtcars

Choose X-axis variable:

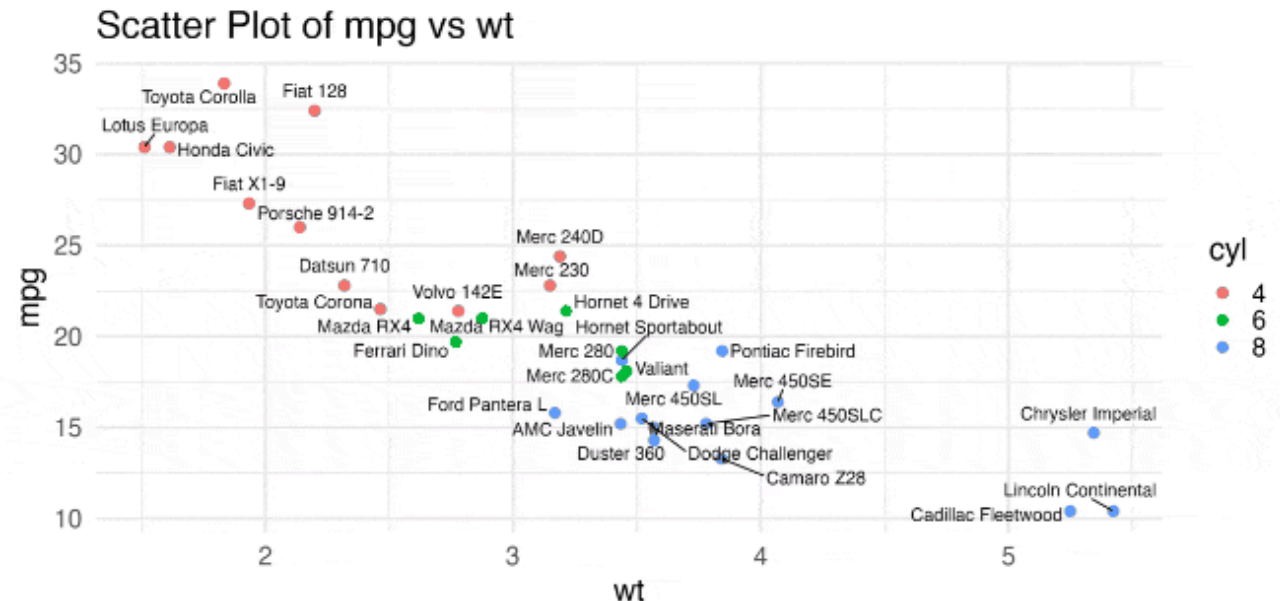
wt

Choose Y-axis variable:

mpg

Choose color variable:

cyl



Outline for today

- What is R Shiny
- Write a basic R shiny app
- **Deploy your Shiny apps with shinyapps.io**
- AI-powered help: using Shiny Assistant



Why shinyapps.io?

- **Free:** Free tier available for up to 5 small apps
- **Easy to use:** No need to set up a server or manage infrastructure
- **Scalable:** Can handle apps of varying sizes and complexities
- **Integration with RStudio:** Seamless deployment from RStudio IDE



Getting started with **shinyapps.io**

- Create an account on shinyapps.io
- **Recommended** to log in using GitHub
- Install the `rsconnect` package in R: `install.packages("rsconnect")`
- Following the instructions on the shinyapps.io to authorize your account

STEP 1 - INSTALL RSCONNECT

The `rsconnect` package can be installed directly from CRAN. To make sure you have the latest version run following code in your R console:

```
install.packages('rsconnect')
```

STEP 2 - AUTHORIZE ACCOUNT

The `rsconnect` package must be authorized to your account using a token and secret. To do this, click the copy button below and we'll copy the whole command you need to your clipboard. Just paste it into your console to authorize your account. Once you've entered the command successfully in R, that computer is now authorized to deploy applications to your shinyapps.io account.

```
rsconnect::setAccountInfo(name='[REDACTED]',  
  token='[REDACTED]',  
  secret='<SECRET>')
```

[Show secret](#)

[Copy to clipboard](#)

In the future, you can manage your tokens from the [Tokens](#) page the settings menu.

STEP 3 - DEPLOY

Once the `rsconnect` package has been configured, you're ready to deploy your first application. If you haven't written any applications yet, you can also checkout the [Getting Started Guide](#) for instructions on how to deploy our demo application. Run the following code in your R console.

```
library(rsconnect)  
rsconnect::deployApp('path/to/your/app')
```



Deploy your app to **shinyapps.io**

- In RStudio, open the app you want to deploy
- Click on the **Publish** button in the top right corner of the RStudio IDE
- Select the shinyapps.io account you just connected
- Fill in the app name and description
- Click **Publish** to deploy your app
- Once deployed, a browser will open to your application. <https://{your-username}.shinyapps.io/{your-app-directory-name}>
- You should see a `rsconnect/` folder within your app directory, which contains the deployment information. **This should be added and committed into version control (i.e. push it to GitHub)**



shinyapps.io dashboard

Dashboard provides an overview of your deployed apps, including: app name and URL, deployment status, usage statistics (e.g., number of active users, CPU usage)

The screenshot shows the shinyapps.io dashboard for an application named "two-files-app". The dashboard is organized into several sections:

- Header:** "shinyapps.io" logo, a menu icon, and user information: "Help", "Account: cindyhu", and a profile icon for "Cindy Hu".
- Left Sidebar:** Navigation menu with "Dashboard", "Applications" (expanded), and "Account". Under "Applications", there are sub-items: "All", "Running", "Sleeping", and "Archived".
- Application Header:** "APPLICATION 14880501 - TWO-FILES-APP". Below this is a row of icons for "Overview", "Metrics", "URLs", "Settings", "Users", "Logs", "Restart", "Archive", and "Delete".
- Overview Panel:** A table-like view showing application details:

Id	14880501
Name	two-files-app
URL	https://cindyhu.shinyapps.io/two-files-app/
Status	Running
Size	large
Deployed	Jun 4, 2025
Updated	Jun 4, 2025
Created	Jun 4, 2025
Bundle	Download
- Instances Panel:** Shows "INSTANCES" with a "Max" button and a single instance with "Id: 11713451" and control icons.
- Application Usage Panel:** Shows "APPLICATION USAGE" with a bar chart. The chart title is "Total: 0.17 hours". The y-axis is labeled "hours" and ranges from 0.00 to 0.20. The x-axis shows dates from May 29 to Jun 04. A single bar for Jun 04 reaches approximately 0.17 hours.
- Footer:** "© 2022 Posit Software, PBC | All Rights Reserved | Terms Of Use"

Check out [shinyapps.io user guide](#) for more information on hosting your app!



Outline for today

- What is R Shiny
- Write a basic R shiny app
- Deploy your Shiny apps with shinyapps.io
- **AI-powered help: using Shiny Assistant**



What is Shiny Assistant?

<https://gallery.shinyapps.io/assistant/>





Hello, I'm Shiny Assistant! I'm here to help you with [Shiny](#), a web framework for data driven apps. You can ask me questions about how to use Shiny, to explain how certain things work in Shiny, or even ask me to build a Shiny app for you.

Here are some examples:

- "How do I add a plot to an application?"
- "Create an app that shows a normal distribution."
- "Show me how make it so a table will update only after a button is clicked."
- Ask me, "Open the editor", then copy and paste your existing Shiny code into the editor, and then ask me to make changes to it.

Let's get started! 🚀

Privacy Notice 

Enter a message...



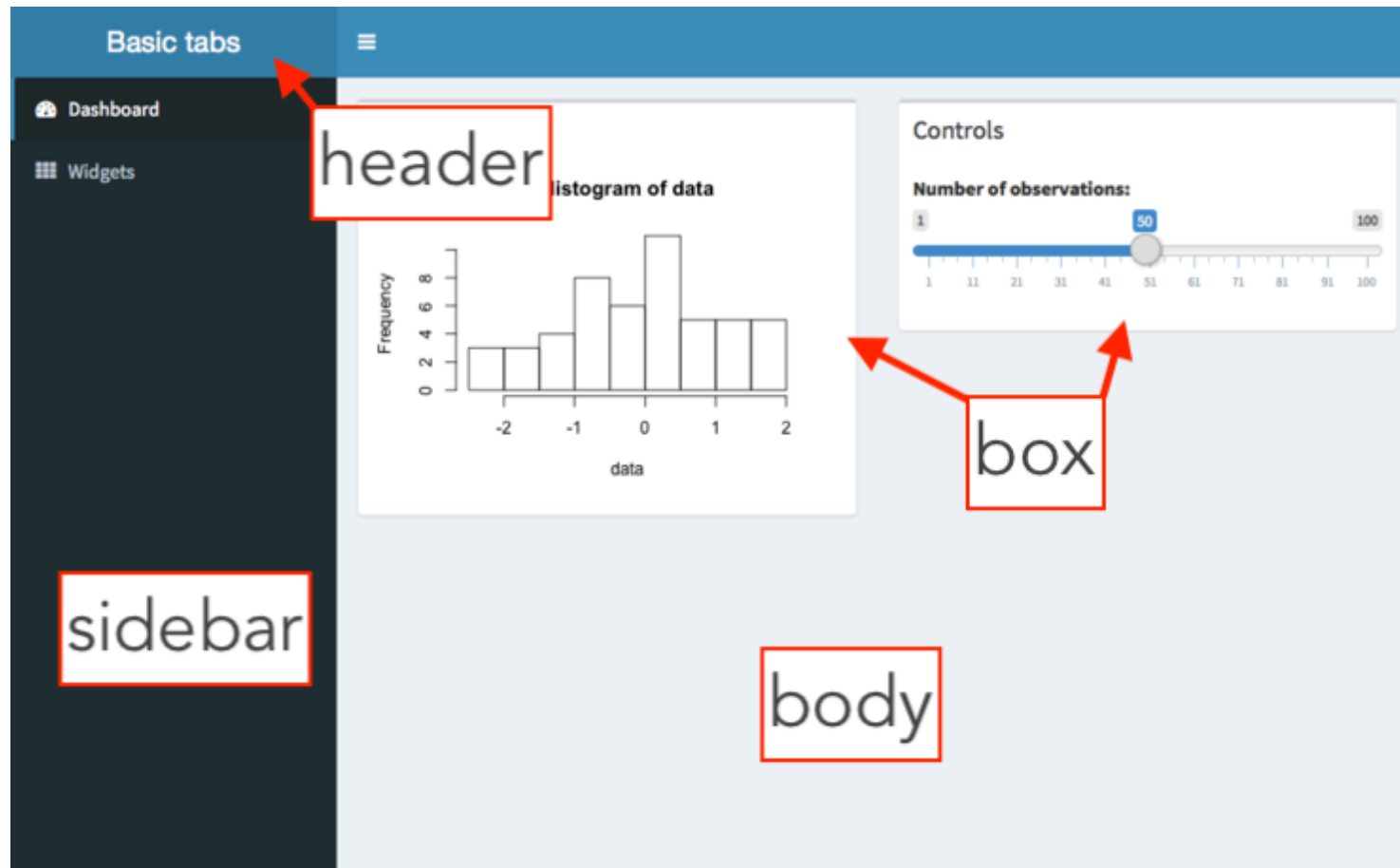
Use Shiny Assistant to recreate the two-file shiny app

Prompt: Create a shiny app using the mtcars dataset. Use three files global.R, ui.R, server.R. The app displays an interactive scatter plot that reacts to user input of x-axis, y-axis, and a third variable for color of the points. Show car name as label text next to the points.



Next, let's build a **shinydashboard**

The `{shinydashboard}` package provides a framework for building dashboards in R Shiny. It allows you to create visually appealing and interactive apps with a more classic “dashboard” layout, including sidebars, tabs, and boxes. You need both packages: `{shiny}` and `{shinydashboard}`.



Use Shiny Assistant to create a **shinydashboard**

Prompt: Create a shinydashboard using the mtcars dataset. Use three files global.R, ui.R, server.R. The app allows users to filter by mpg and displays three output: an interactive scatter plot that reacts to user input of x-axis, y-axis, and color variable. A heatmap of car performance. And a data table of the filtered dataset. Show car name as label text next to the points in the scatter plot.

R Python

Concise



Enter a message...



Refine the app locally

- Download the code from Shiny Assistant



- Save the files in your local R project directory, remember to organize your repo structure well
- Open the files in RStudio and run the app locally
- Make any necessary adjustments to the code



Your turn in hw4

Try one of these built-in datasets to explore with Shiny Assistant

Dataset	Description
mtcars 🚗	Car performance data (mpg, cylinders, hp)
iris 🌸	Iris flower measurements (sepal, petal, species)
diamonds 💎	Diamond pricing (carat, cut, price, etc.)
faithful 🌋	Old Faithful geyser eruptions (duration, waiting time)
airquality 🌍	New York air quality data (Ozone, Temp, Wind)
ToothGrowth 🦷	Vitamin C & tooth growth in guinea pigs

Or upload your own dataset to GitHub and give Shiny Assistant a public URL!

Write good prompt, and you will get good code!

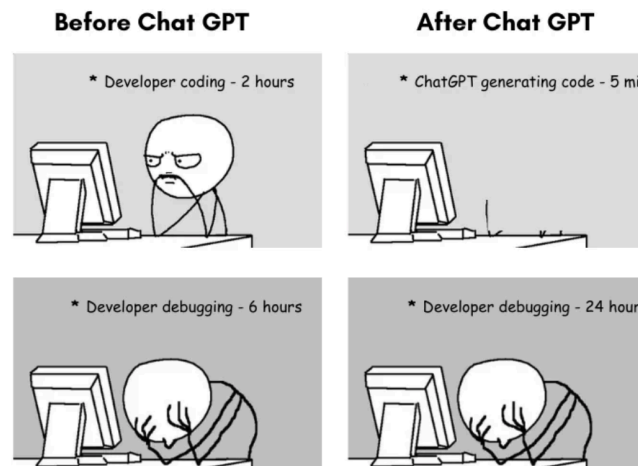
Describe what change you made to the app on top of what Shiny Assistant generated.



Further resources

How much time do you have?

- **10 min:** Print out this [Shiny for R cheatsheet](#)
- **2.5 hrs:** Follow this [Posit tutorial](#)
- **Lifetime:** Check out resources like the [Shiny Gallery](#), [TidyTuesday](#), and [Mastering Shiny](#) book
- **Unknown:** chatGPT, Gemini, Shiny Assistant (powered by Anthropic), and other AI tools can help you build Shiny apps



End-of-Class Survey

 Fill out the end-of-class survey

~ This is the end of Lecture 4 ~