

An Introduction to R Shiny



A Tool Demonstration by Group 6

What is Shiny?

- Code-based tool for data visualization.
- R Shiny is a web application framework for R.
- Allows creation of interactive web applications directly from R code.
- Build web apps with R Shiny without knowing HTML, CSS, or JavaScript.
- Open source and free to use.
- Has a large and supportive community of users and developers.
- Widely used in data science, finance, healthcare, and other fields for creating data-driven web applications.

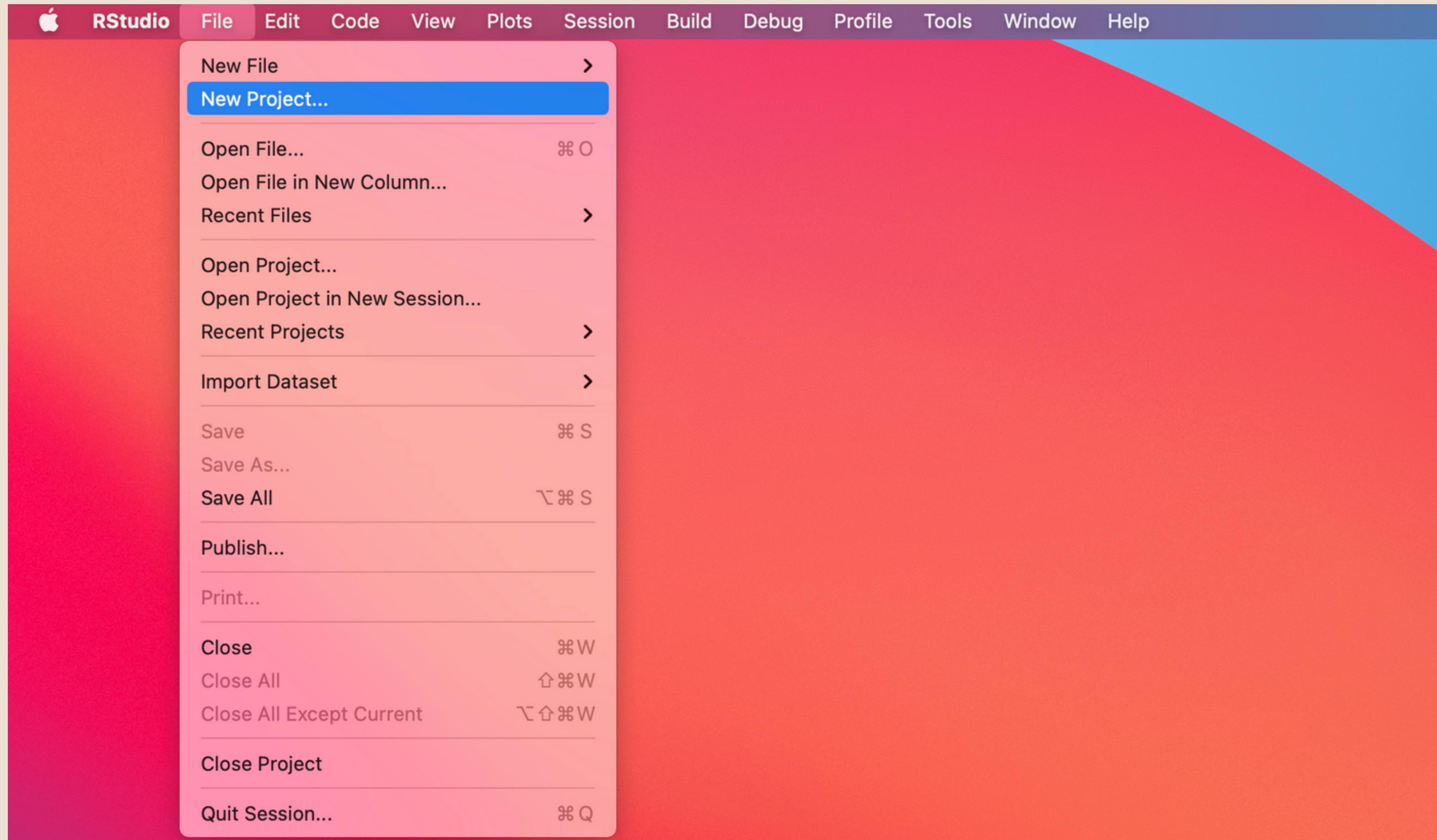
Features

- Supports reactive programming, which means that your application will update in real-time as the user interacts with it.
- Wide range of input widgets, such as sliders, dropdowns, text boxes, and buttons, that can be used to build interesting UIs.
- Built-in support for data processing and manipulation, so you can perform calculations and analysis within your application.
- Create complex layouts and data visualizations, including interactive maps and charts.
- Integrates with other web technologies, such as databases, APIs, and web services.

Demonstration

Setting up a new project in Rstudio

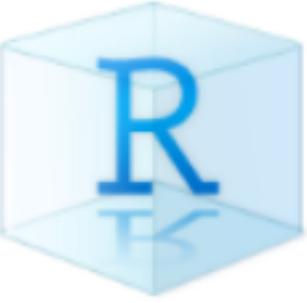
Step 1- Select File > New Project.. from the menu bar.



Step 2- Select New Directory from the pop-up window

New Project Wizard

Create Project

-  **New Directory**
Start a project in a brand new working directory >
-  **Existing Directory**
Associate a project with an existing working directory >
-  **Version Control**
Checkout a project from a version control repository >

Cancel

Step 3- Select Shiny Application

New Project Wizard

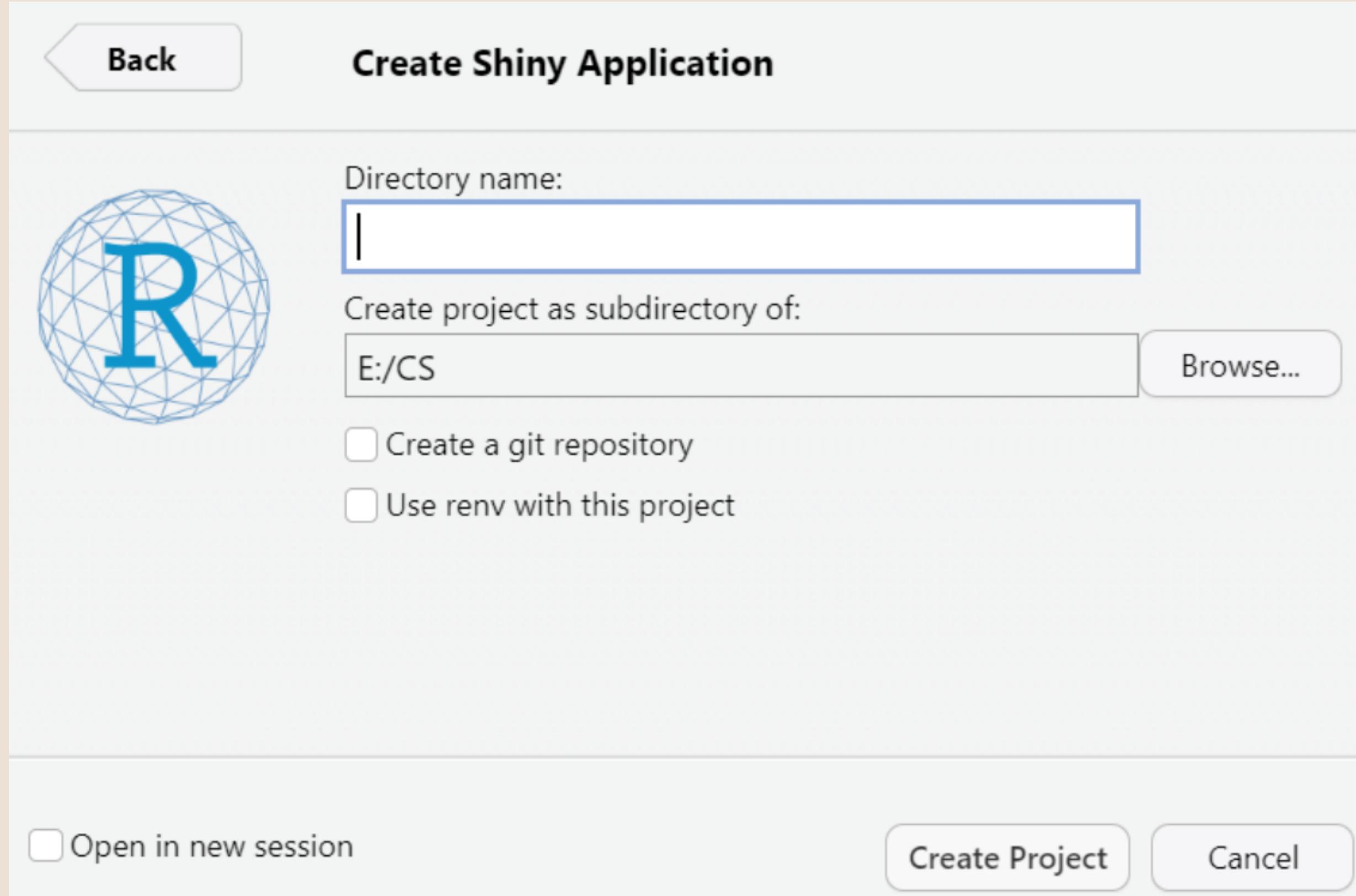
[Back](#)

Project Type

-  New Project >
-  R Package >
-  Shiny Application >
-  Quarto Project >
-  Quarto Website >
-  Quarto Blog >
-  Quarto Book >

[Cancel](#)

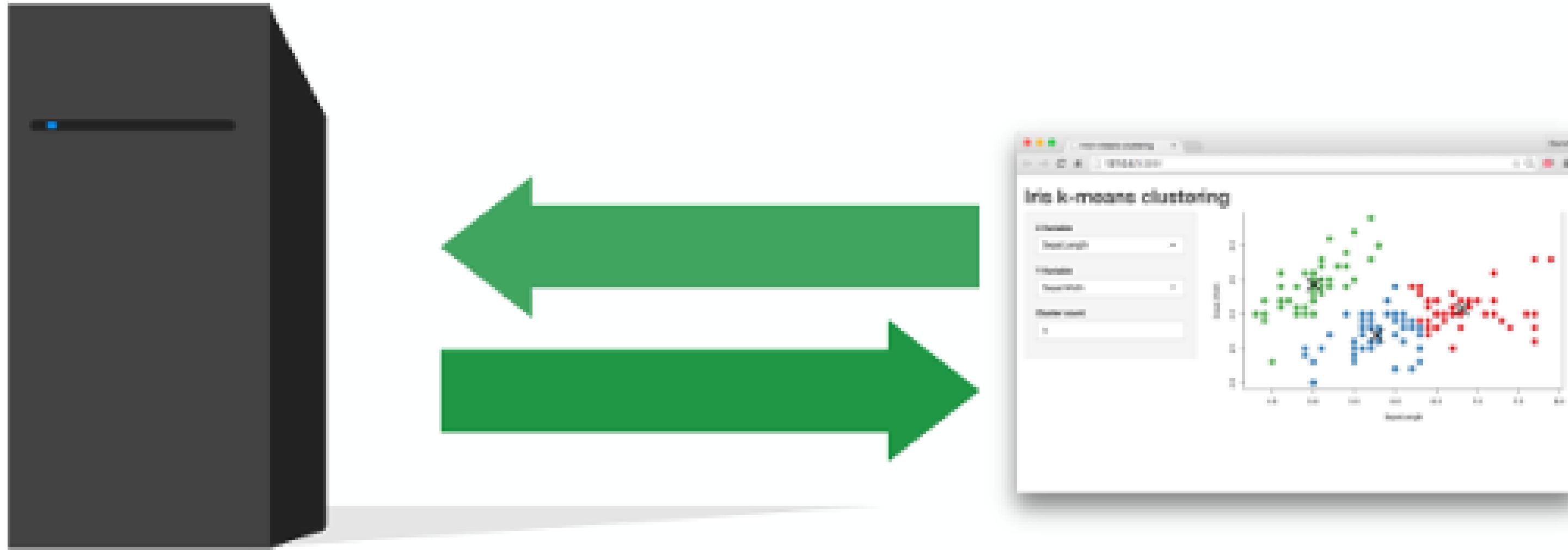
Step 4- Pick a name for your Project Directory, and select a subdirectory to store your project.



Step 5- Tick the **open in new session**, and click **Create Project** to open your R project in a new R studio window.

Creating an App with R Shiny

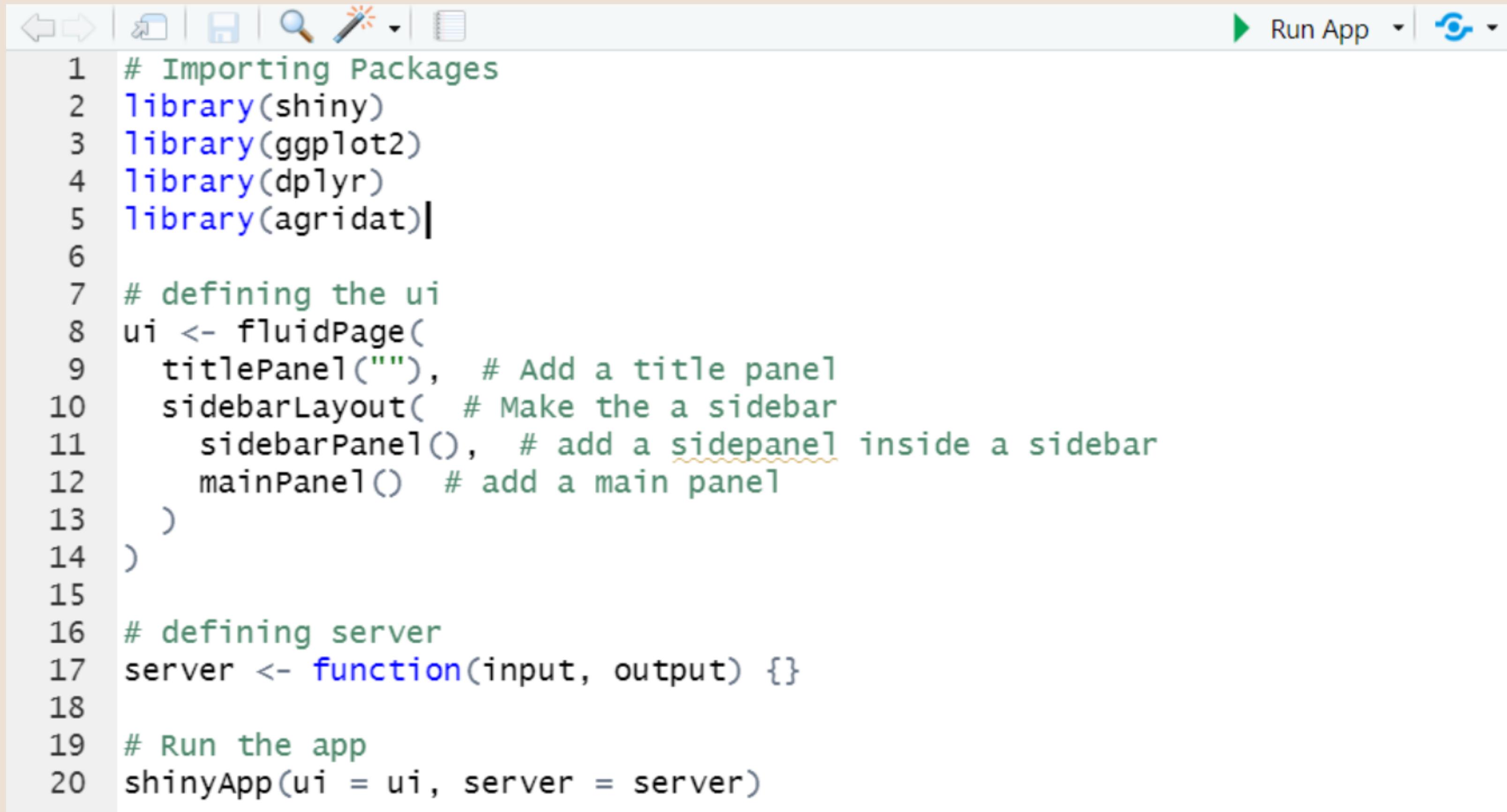
A shiny web applications requires a computer/server with R



Server Instructions

User Interface (UI)

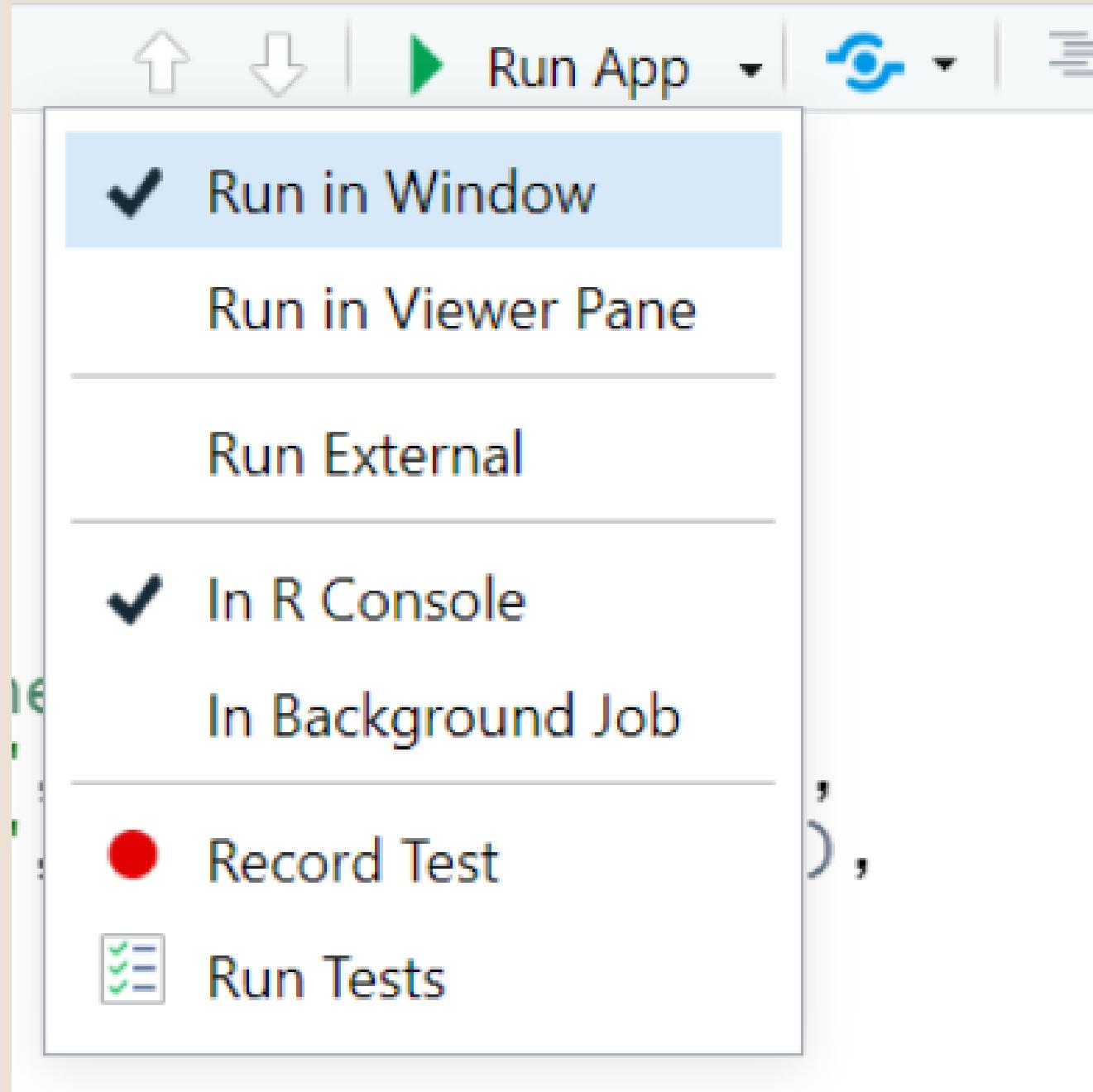
1. Basic R script for structure of a shiny app:



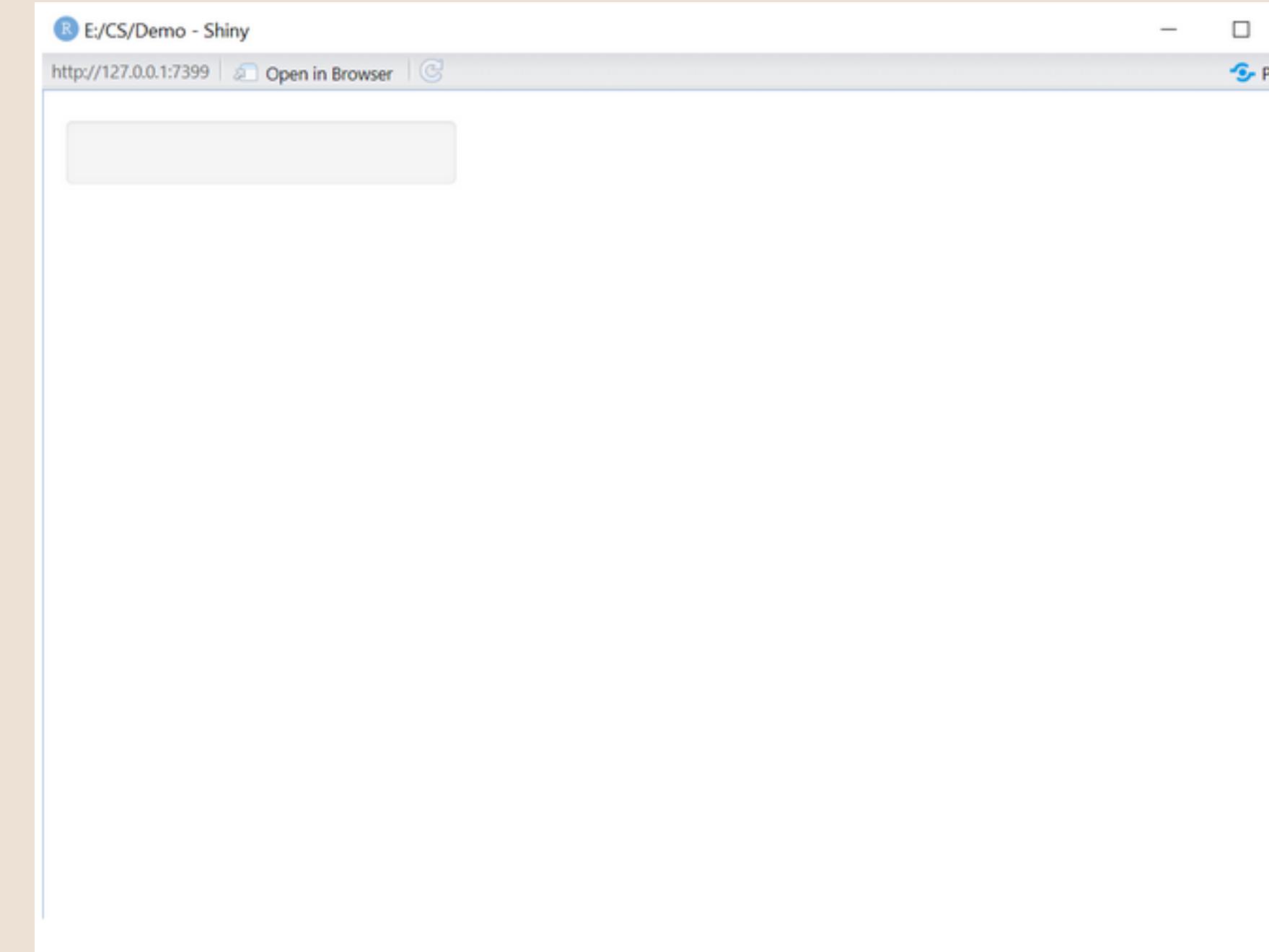
The screenshot shows an RStudio interface with a code editor containing an R script. The script defines the structure of a Shiny application, including importing packages, defining the user interface (ui), defining the server logic, and running the app. The RStudio toolbar at the top includes icons for file operations, search, and help, along with a 'Run App' button.

```
1 # Importing Packages
2 library(shiny)
3 library(ggplot2)
4 library(dplyr)
5 library(agridat)
6
7 # defining the ui
8 ui <- fluidPage(
9   titlePanel("", # Add a title panel
10  sidebarLayout( # Make the a sidebar
11    sidebarPanel(), # add a sidepanel inside a sidebar
12    mainPanel() # add a main panel
13  )
14 )
15
16 # defining server
17 server <- function(input, output) {}
18
19 # Run the app
20 shinyApp(ui = ui, server = server)
```

2. To run the app, click **Run App** at the top right of the script window.



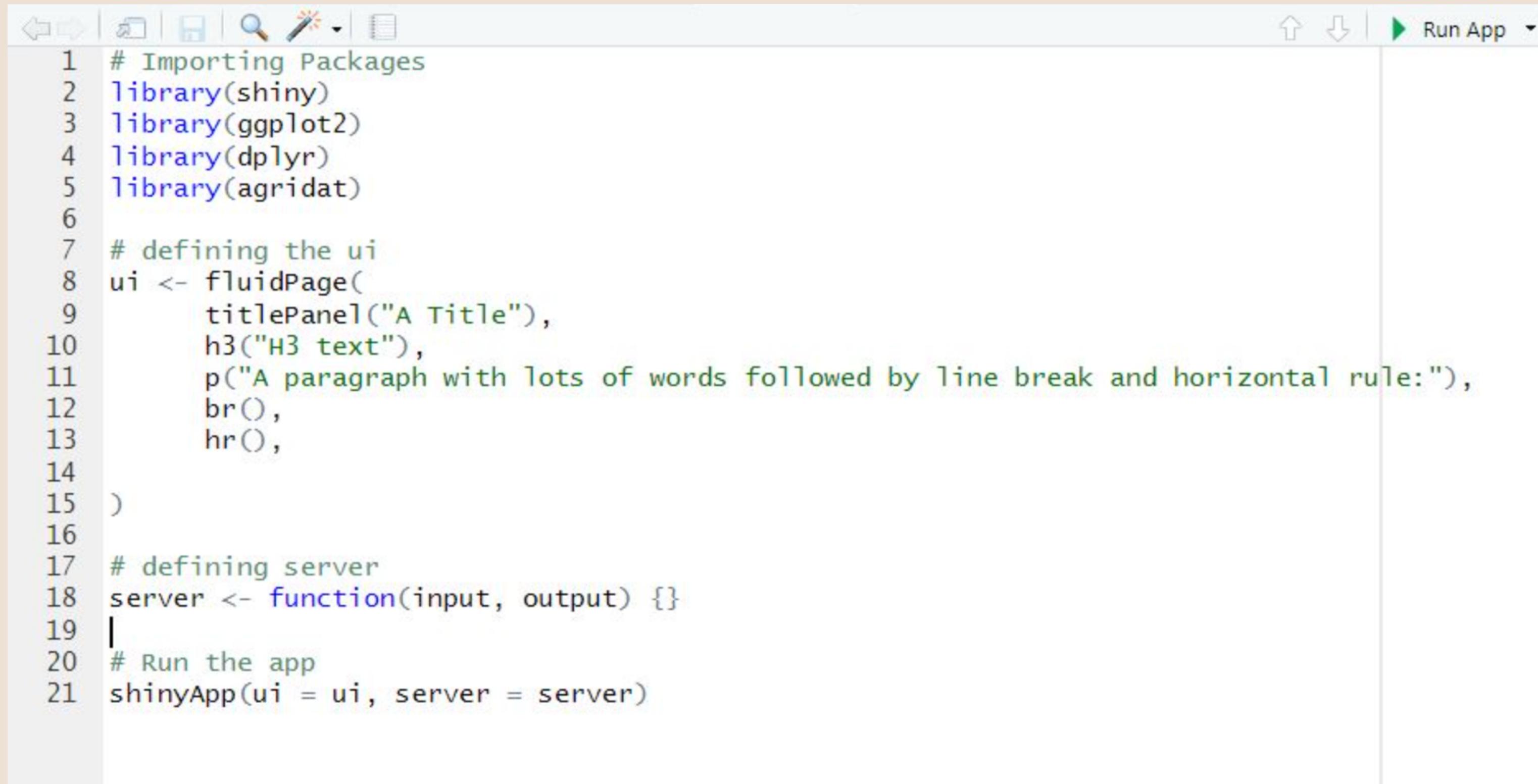
3. Output after running the script from the previous slide.



4. Without the connection to the server and app, UI is just HTML content

```
> fluidPage(  
+   titlePanel("", # Add a title panel  
+   sidebarLayout( # Make the a sidebar  
+     sidebarPanel(), # add a sidepanel inside a sidebar  
+     mainPanel() # add a main panel  
+   )  
+ )  
<div class="container-fluid">  
  <h2></h2>  
  <div class="row">  
    <div class="col-sm-4">  
      <form class="well" role="complementary"></form>  
    </div>  
    <div class="col-sm-8" role="main"></div>  
  </div>  
</div>  
> |
```

5. We can add HTML tags in the ui.



The screenshot shows the RStudio interface with the code editor tab selected. The code is written in R and defines a Shiny application. It includes sections for importing packages, defining the user interface (ui), defining the server logic, and running the app. The ui section contains HTML tags like titlePanel, h3, p, br, and hr.

```
1 # Importing Packages
2 library(shiny)
3 library(ggplot2)
4 library(dplyr)
5 library(agridat)
6
7 # defining the ui
8 ui <- fluidPage(
9   titlePanel("A Title"),
10  h3("H3 text"),
11  p("A paragraph with lots of words followed by line break and horizontal rule:"),
12  br(),
13  hr(),
14 )
15
16
17 # defining server
18 server <- function(input, output) {}
19
20 # Run the app
21 shinyApp(ui = ui, server = server)
```

6. The very basic shiny app you'll see when you run the code in previous slide

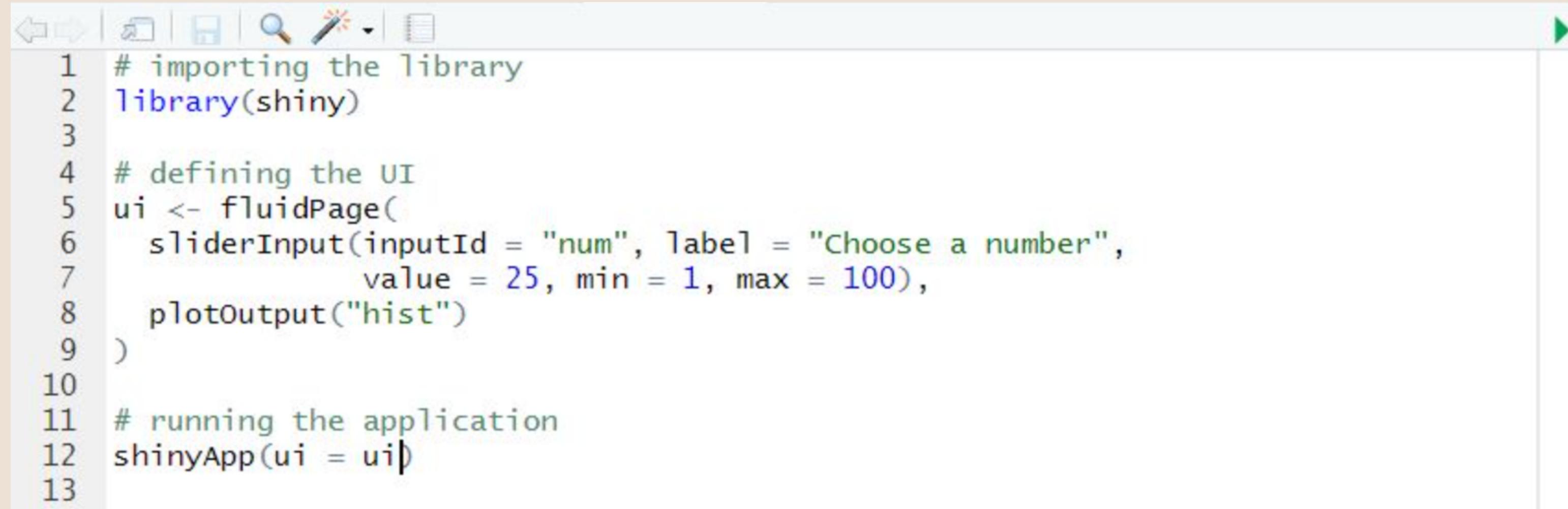
The screenshot shows a window titled "E:/CS/Demo - Shiny". The window has standard operating system controls (minimize, maximize, close) at the top right. Below the title bar is a toolbar with a blue "R" icon, a "Publish" button, and other icons. The main content area contains the following text:

A Title

H3 text

A paragraph with lots of words followed by line break and horizontal rule:

7. R script and Error on running the app without defining the server



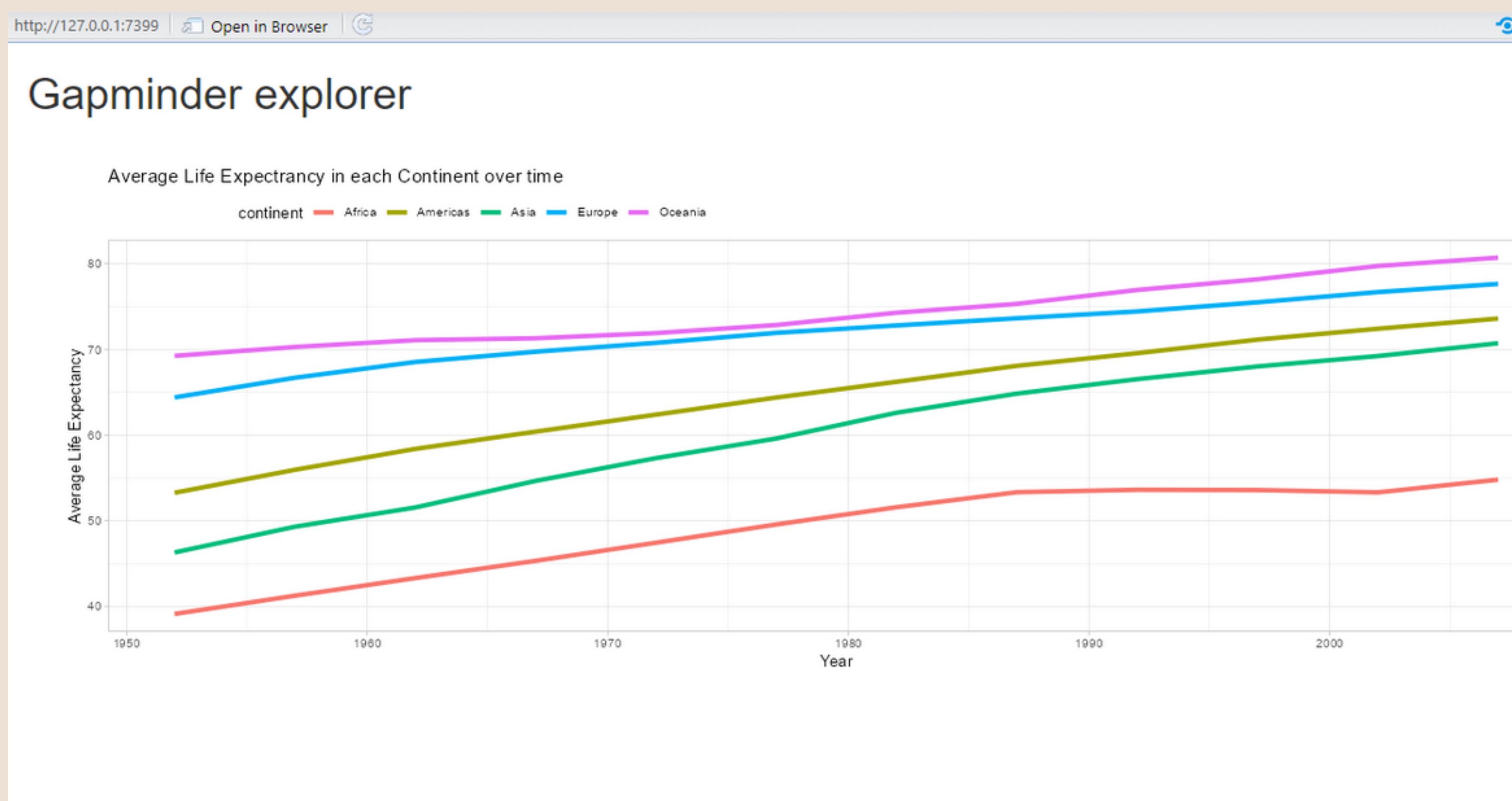
```
1 # importing the library
2 library(shiny)
3
4 # defining the UI
5 ui <- fluidPage(
6   sliderInput(inputId = "num", label = "Choose a number",
7               value = 25, min = 1, max = 100),
8   plotOutput("hist")
9 )
10
11 # running the application
12 shinyApp(ui = ui)
13
```

```
> runApp('app2.R')
Error in shinyApp(ui = ui) :
  argument "server" is missing, with no default
>
```

8. Script for Plotting a Static Line Graph.

```
1 library(gapminder)
2 library(ggplot2)
3 library(dplyr)
4 library(shiny)
5
6 ui <- fluidPage(
7   titlePanel("Gapminder explorer"),
8   plotOutput("line")
9 )
10
11 server <- function(input, output) {
12   output$line <- renderPlot({
13     data <- gapminder %>%
14       group_by(year, continent) %>%
15       summarise(avgLifeExp = mean(lifeExp))
16
17     ggplot(data, aes(x = year, y = avgLifeExp, color = continent)) +
18       geom_line(size = 1.5) +
19       ggtile("Average Life Expectancy in each Continent over time") +
20       labs(x = "Year", y = "Average Life Expectancy") +
21       theme_light() +
22       theme(
23         plot.margin = unit(c(2, 1, 1, 1), "cm"),
24         plot.title = element_text(vjust = 13),
25         legend.position = c(0.25, 1.07), legend.direction = "horizontal"
26       )
27   })
28 }
29
30 shinyApp(ui = ui, server = server)
```

9. Visuals of graph plotted by script from the previous page.



Some basic interactive visualizations using R Shiny

Example 1. R script showing how to take inputs for visualizing data.

```
1 library(shiny)
2
3 ui <- fluidPage(
4   titlePanel("Title"),
5   sidebarLayout(
6     sidebarPanel(
7       selectInput(inputId = "one",
8                  label = "Drop Down", # Give the input a label
9                  choices = c("A" = "a", "B" = "b", "C" = "c", "D" = "d",
10                 "E" = "e", "F" = "f", "G" = "g", "H" = "h"),
11                 selected = "a"),
12       sliderInput(inputId = "two",
13                  label = "Slider",
14                  min=1, max=25, value= c(10)),
15      textInput(inputId = "text",
16                  label = "Text Input", ""),
17       actionButton(inputId = "action", label = "Go!"),
18       radioButtons(inputId = "radio", label = "Radio Buttons",
19                  choices = c("A", "B")),
20
21     ),
22     mainPanel()
23   )
24 )
25
26 server <- function(input, output) {}
27 shinyApp(ui = ui, server = server)
28
```

Example 1. Output Visual for the R script from the previous slide.

http://127.0.0.1:7399 | [Open in Browser](#) |

[Publish](#) ▾

Title

Drop Down

A ▾

Slider

1 10 25

1 4 7 10 13 16 19 22 25

Text Input

Go!

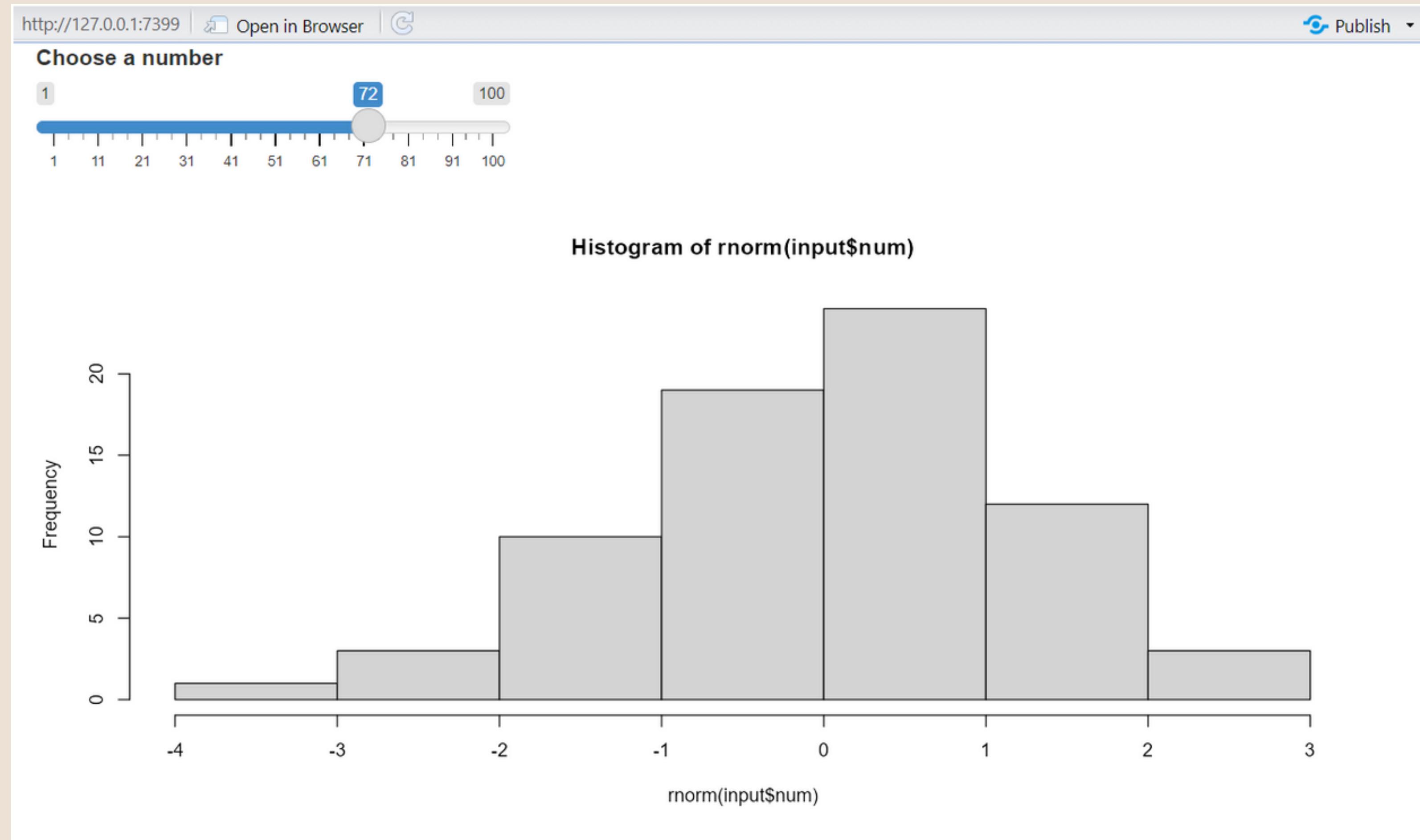
Radio Buttons

A
 B

Example 2. R script to create a simple normal distribution histogram.

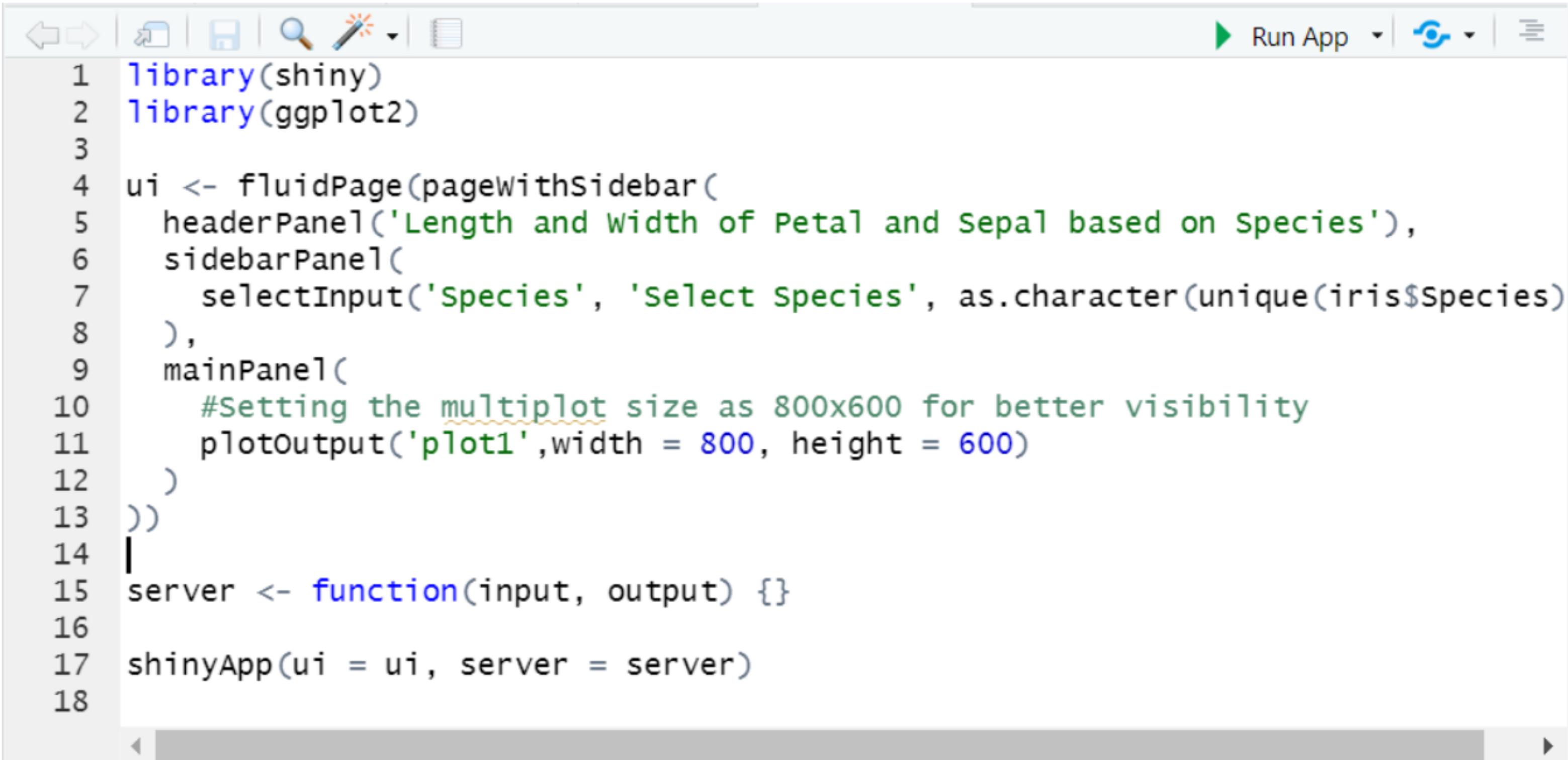
```
1 # importing the library
2 library(shiny)
3
4 # defining the UI
5 ui <- fluidPage(
6   sliderInput(inputId = "num", label = "Choose a number",
7               value = 25, min = 1, max = 100),
8   plotOutput("hist")
9 )
10
11 # defining server
12 server <- function(input, output) {
13   output$hist <- renderPlot({
14     hist(rnorm(input$num)) #n random samples distr. with mean=0 sd=1
15   })
16 }
17
18 # running the application
19 shinyApp(ui = ui, server = server)
20
21
```

Example 2. Snapshot of Resulting visualization for the script in the previous slide.



Visualizing a Dashboard with R Shiny: Iris Dataset

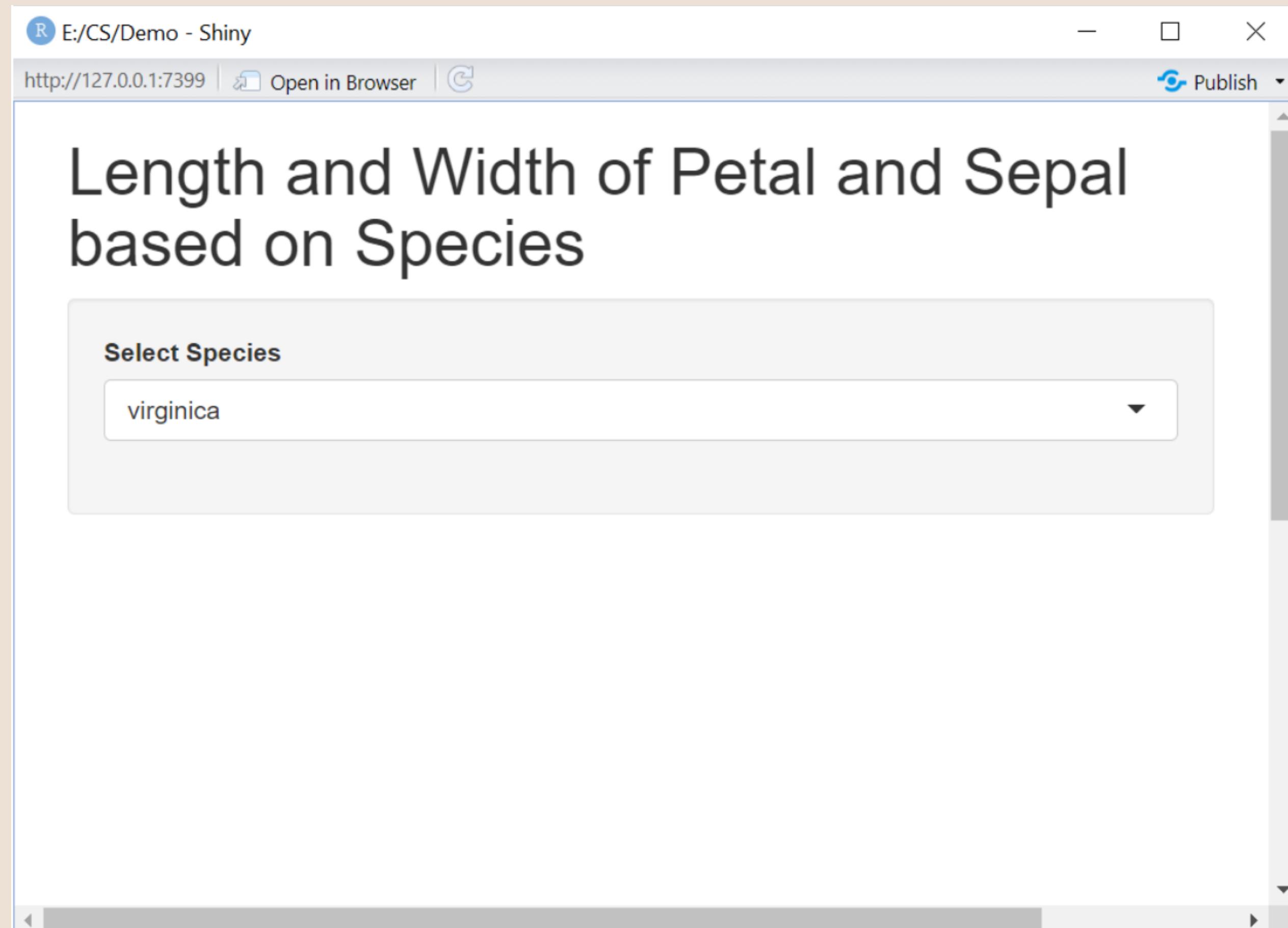
- The dataset used for this project is IRIS which is available in R.
- Creating the UI:



The screenshot shows the RStudio interface with the code editor tab selected. The code is written in R and defines a Shiny application. The code includes imports for shiny and ggplot2, sets up a fluid page with a sidebar, and creates a select input for species. It also includes a main panel with a plot output and a server function that runs the shinyApp.

```
1 library(shiny)
2 library(ggplot2)
3
4 ui <- fluidPage(pageWithSidebar(
5   headerPanel('Length and Width of Petal and Sepal based on Species'),
6   sidebarPanel(
7     selectInput('Species', 'Select Species', as.character(unique(iris$Species)))
8   ),
9   mainPanel(
10    #Setting the multiplot size as 800x600 for better visibility
11    plotOutput('plot1', width = 800, height = 600)
12  )
13 ))
14
15 server <- function(input, output) {}
16
17 shinyApp(ui = ui, server = server)
18
```

- Resulting Visual of Step 1.



- Creating the Server Logic (visualizations are made in ggplot):

The screenshot shows the RStudio interface with the code editor tab active. The code is written in R and defines a server function to generate four histograms for the Sepal and Petal dimensions of the Iris dataset.

```
14
15 library(gridExtra)
16
17 server <- function(input, output) {
18
19   data <- reactive({iris[iris$Species == input$Species,]})  

20
21   output$plot1 <- renderPlot({  

22
23     # Sepal Length  

24     g1 <- ggplot(data(), aes(Sepal.Length))  

25     g1 <- g1 + geom_histogram(binwidth = .5, fill="#477744", color = "#477744", alpha = .2)  

26     g1 <- g1 + geom_vline( aes(xintercept = mean(data()$Sepal.Length)), colour="black", size=2, alpha=.6)  

27     g1 <- g1 + labs(x = "Sepal Length")  

28     g1 <- g1 + labs(y = "Frequency")  

29     g1 <- g1 + labs(title = paste("Distribution of Sepal Length, mu =", round(mean(data()$Sepal.Length),2)))  

30
31     # Sepal width  

32     g2 <- ggplot(data(), aes(Sepal.Width))  

33     g2 <- g2 + geom_histogram(binwidth = .5, fill="#477744", color = "#477744", alpha = .2)  

34     g2 <- g2 + geom_vline( aes(xintercept = mean(data()$Sepal.Width)), colour="black", size=2, alpha=.6)  

35     g2 <- g2 + labs(x = "Sepal Width")  

36     g2 <- g2 + labs(y = "Frequency")  

37     g2 <- g2 + labs(title = paste("Distribution of Sepal Width, mu =", round(mean(data()$Sepal.Width),2)))  

38
39     # Petal Length  

40     g3 <- ggplot(data(), aes(Petal.Length))  

41     g3 <- g3 + geom_histogram(binwidth = .5, fill="#f55c7a", color = "#f55c7a", alpha = .2)  

42     g3 <- g3 + geom_vline( aes(xintercept = mean(data()$Petal.Length)), colour="black", size=2, alpha=.6)  

43     g3 <- g3 + labs(x = "Petal Length")  

44     g3 <- g3 + labs(y = "Frequency")  

45     g3 <- g3 + labs(title = paste("Distribution of Petal Length, mu =", round(mean(data()$Petal.Length),2)))  

46
47     # Petal width  

48     g4 <- ggplot(data(), aes(Petal.Width))  

49     g4 <- g4 + geom_histogram(binwidth = .5, fill="#f55c7a", color = "#f55c7a", alpha = .2)  

50     g4 <- g4 + geom_vline( aes(xintercept = mean(data()$Petal.Width)), colour="black", size=2, alpha=.6)  

51     g4 <- g4 + labs(x = "Petal width")  

52     g4 <- g4 + labs(y = "Frequency")  

53     g4 <- g4 + labs(title = paste("Distribution of Petal width, mu =", round(mean(data()$Petal.width),2)))  

54
55     #Plot 4 graphs together|  

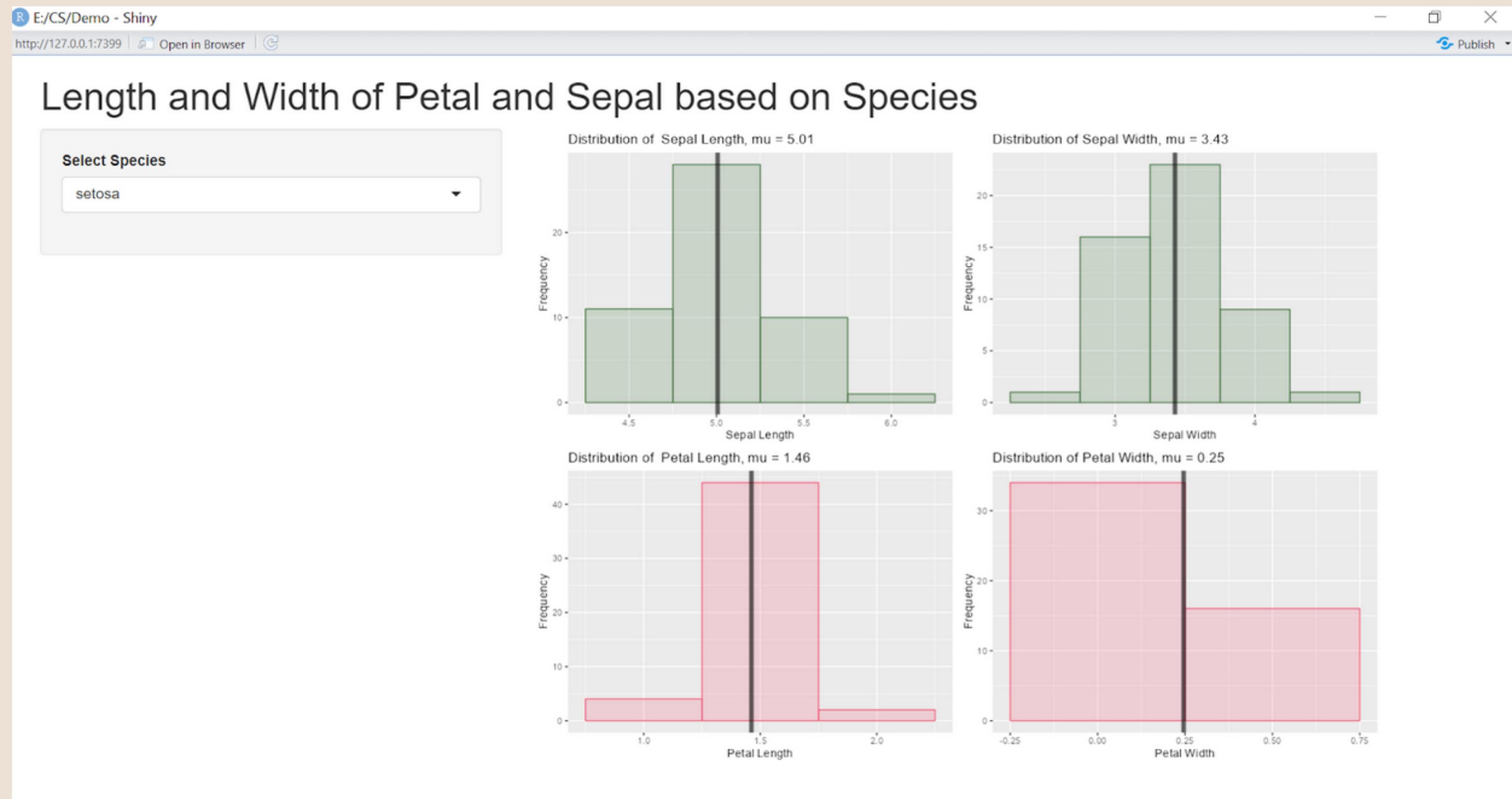
56     grid.arrange(g1,g2,g3,g4,nrow=2, ncol=2)  

57   })  

58 }  

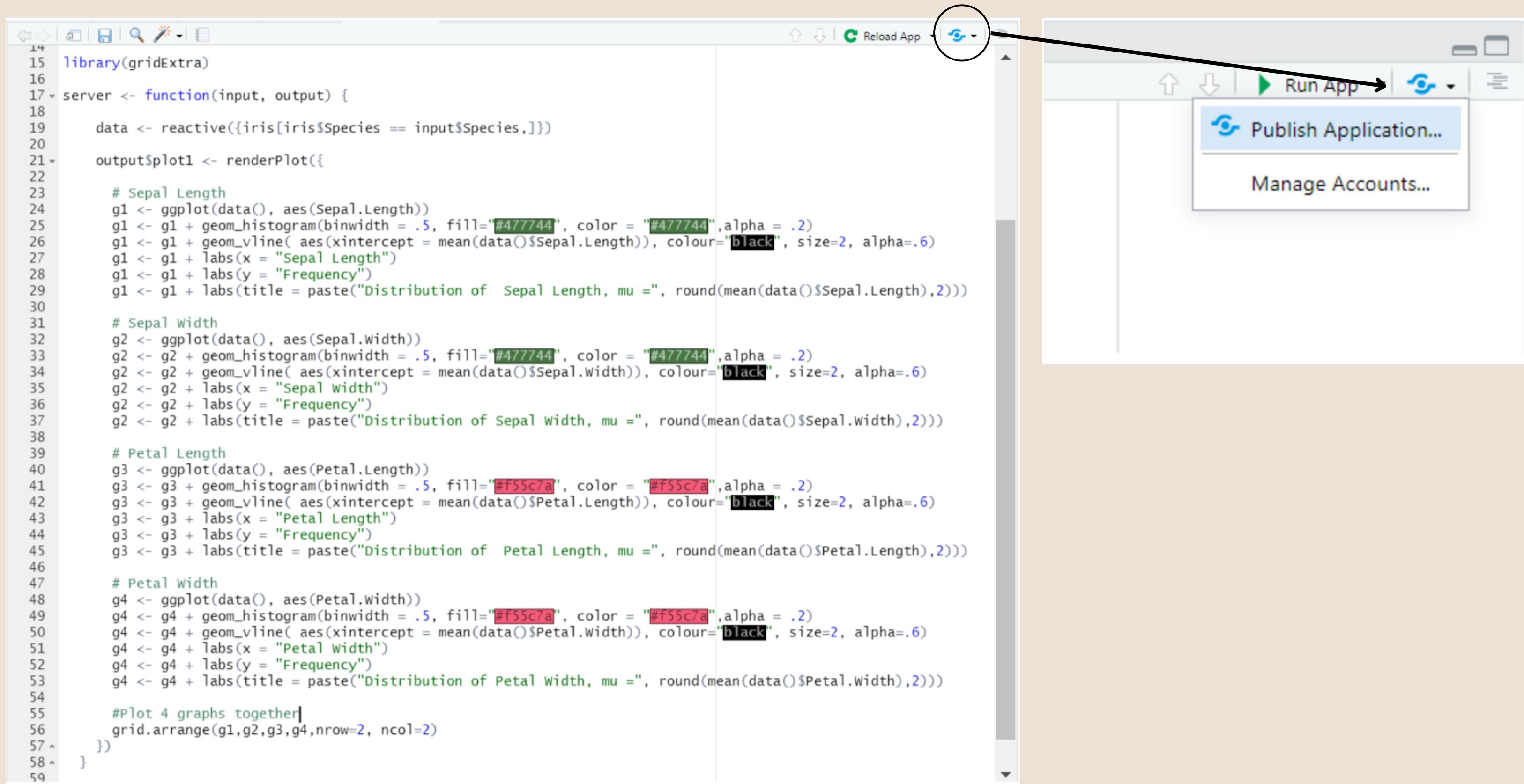
59 }
```

- Final Visualization



Publishing an R Shiny App

Step 1- Click the button shown in the figure below in the right corner of the window



Step 2- If it is your first Shiny app, the box “Publish From Account” should be empty. Click on “Add New Account” to link the shinyapps.io account you just created, else connect to your existing Publishing Account.

The image shows two overlapping windows from RStudio. The left window is titled "Publish to Server" and displays the "Publish Files From" path as ".../GitHub/deploy-a-shiny-app". It includes a checkbox for "app.R" and a "Launch browser" checkbox at the bottom. The right window is titled "Connect Account" and has a heading "Connect Publishing Account". It features a "shinyapps.io" logo and instructions: "To publish content, you first need to connect RStudio to an account on the service you want to publish to." Below this, it says "Once you've authorized this computer to publish content to an account, you can publish any time without re-entering your credentials." A red arrow points from the "Add New Account" button in the Publish dialog to the "shinyapps.io" logo in the Connect dialog.

Publish to Server

Publish Files From: .../GitHub/deploy-a-shiny-app

app.R

Launch browser

Publish From Account:

[antoinesoetewey: shinyapps.io](#) [Add New Account](#)

Title:
deploy-a-shiny-app

Connect Account

Connect Publishing Account



To publish content, you first need to connect RStudio to an account on the service you want to publish to.

Once you've authorized this computer to publish content to an account, you can publish any time without re-entering your credentials.

Next [Cancel](#)

Step-3. Click on the link to your ShinyApps account:

Connect Account

Back Connect ShinyApps.io Account



Go to [your account on ShinyApps](#) and log in.

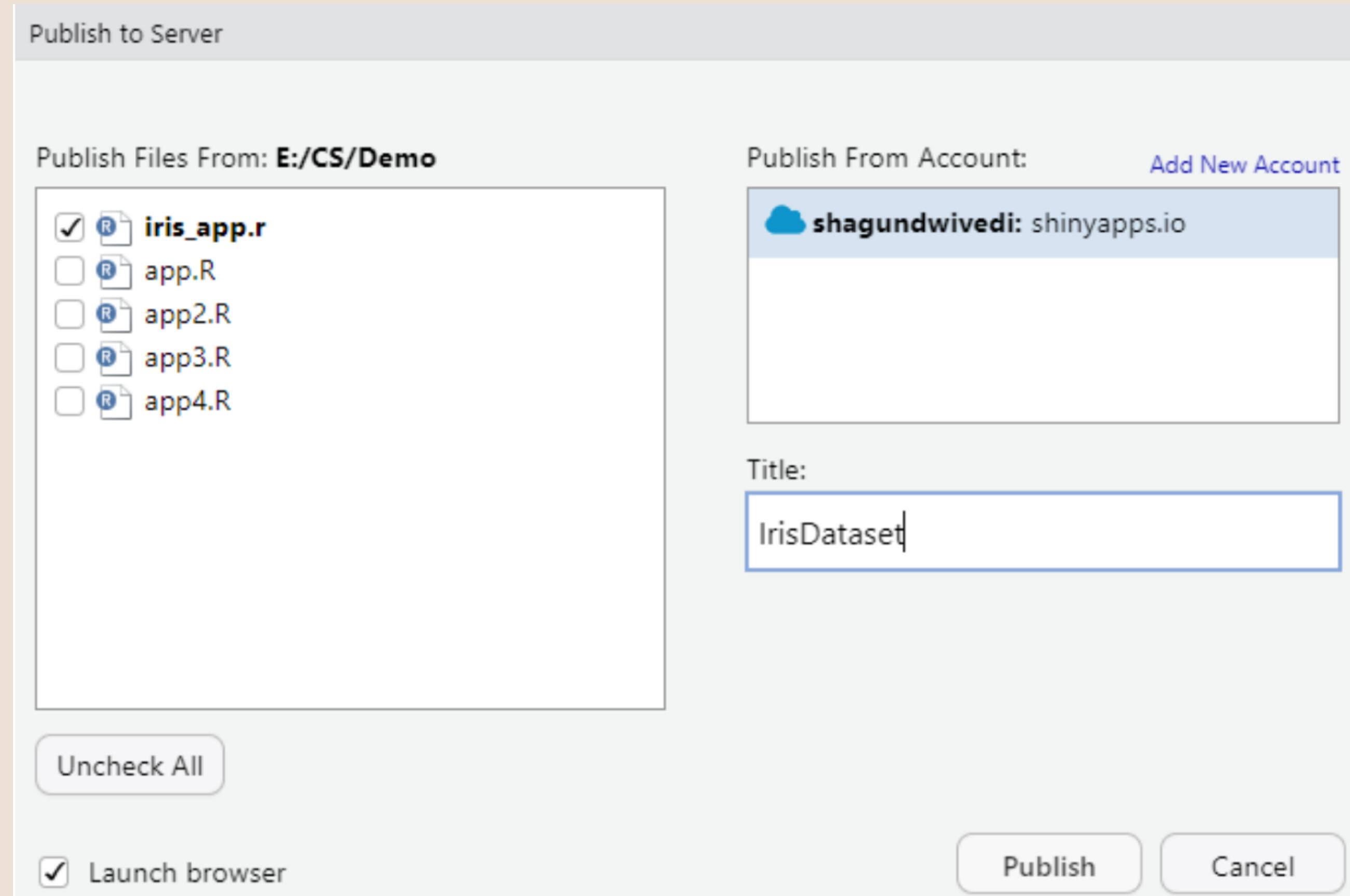
Click your name, then choose **Tokens** from your account menu.

Click **Show** on the token you want to use, then **Show Secret** and **Copy to Clipboard**. Paste the result here:

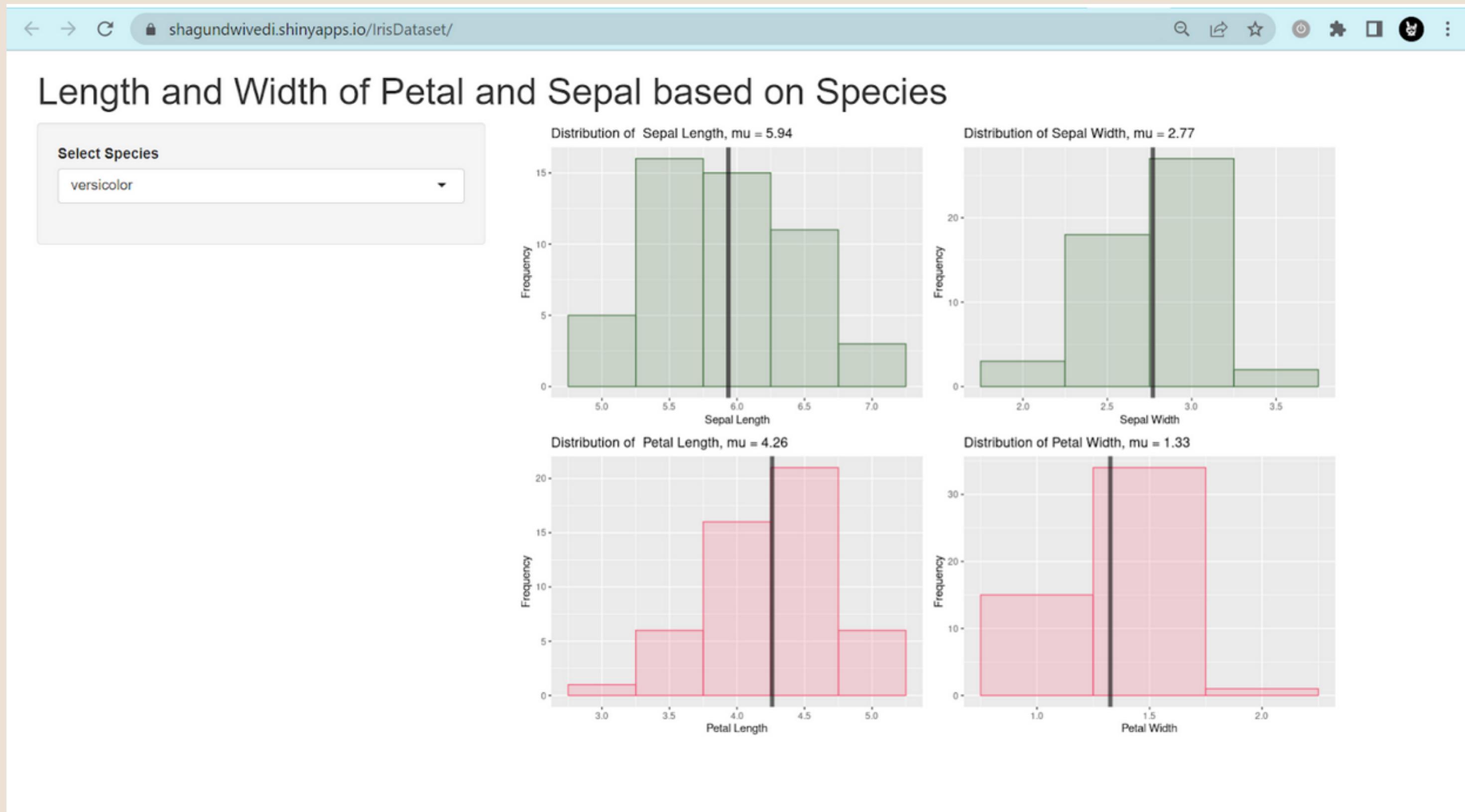
Need a ShinyApps.io account? [Get started here.](#)

Connect Account Cancel

Step-4. Choose a title (without space) and click on the Publish button:



Step-5. After a while, we get redirected to the shinyapps.io webpage:



Personal insights about R Shiny

Strengths and Weaknesses

Insights

- Relatively easy to learn if you know R
- Results can be published in a straightforward manner
- Using RStudio makes work smoother

PROS (Being Code Based):

- highly customizable
- good for creating dashboards
- can use any R library like ggplot, tidyverse to create visualizations
- can carry out data manipulation if needed

CONS (Being Code Based):

- requires knowledge of the R language
- more time and effort taking than UI-based tools like Tableau
- functionalities not present in R packages are hard to implement
- user needs to understand UI and server logic and how reactivity applies to the app

How can you learn R Shiny?

A list of resources for you to get started with R Shiny!

- https://lrouviere.github.io/VISU/pres_shiny.pdf
- <https://cl.indiana.edu/~obscrivn/Shiny-Introduction.pdf>
- <https://deanattali.com/blog/building-shiny-apps-tutorial/>
- <https://ibiostat.be/seminar/uploads/introdcution-r-shiny-package-20160330.pdf>

Contributors: Group 6

- Shagun Dwivedi
- Deeksha Chutani
- Param Chordiya
- Shaifali Vashistha

Thank You