An Introduction to Shiny

A hands-on workshop

Andreas M. Brandmaier and Leonie Hagitte

2024-03-14

Table of contents

1 Herzlich Wilkommen!

1.1 Ask questions anytime

1.2 Shiny

> Shiny is an R package that makes it easy to build interactive web apps in R

Apps can be

- standalone,
- deployed to a website,
- or be part of an interactive (Markdown) document

1.3 How do Shiny apps look like?

Some examples:

- A word cloud: https://shiny.posit.co/r/gallery/start-simple/word-cloud/
- Latent change score simulation: https://lcdlab.shinyapps.io/LCS-simulation-app/

1.4 Required software

You need to install these software packages in order to follow along with the examples of today:

- **R**: https://cran.r-project.org
- **RStudio**: https://posit.co/download/rstudio-desktop/

And a couple of R packages:

- **shiny, tidyverse packages, palmerpenguins, ...**

```
install.packages(c("shiny","tidyverse", "shinydashboard","palmerpenguins"))
```

1.5 Workshop materials

Please find the slides and code snippets here:

https://github.com/brandmaier/shiny_workshop_2024

1.6 What to expect

- This is a hands-on workshop; you'll get the most out of it if you download the materials and actively participate
- Introductory R coding skills are OK! We have exercises at varying levels of proficiency
- The workshop materials remain open and accessible after the workshop
- Feel free to team up!

2 Goals

2.1 Objectives of today

- Learn about the structure of a shiny application.
- Learn how to create shiny apps from a template.
- Learn how to think in terms of *inputs* and *outputs*.
- Write your own apps (using simulated data, real data or *your* data)

2.2 Content

Let's talk about...

- User-interface / Layout
- Reactivity / Logic
- Awesome visualizations

2.3 Anatomy of a Shiny app

```
library(shiny)

shinyApp(
  ui = list(),
  server = function(input, output, session) {  }
)
```

We first load the `shiny` package and define a `shinyApp`, which really is only a function call with two arguments.

2.4 Anatomy of a Shiny app

```
library(shiny)

shinyApp(
  ui = list(),
  server = function(input, output, session) {  }
)
```

The `ui` specifies the *visible* user interface

- Dynamic elements *inputs* and *outputs*
- Static elements like headings, text, static images
- A layout how to arrange these things

2.5 Anatomy of a Shiny app

```
library(shiny)

shinyApp(
  ui = list(),
  server = function(input, output, session) {  }
)
```

The `server` is *invisible* and is responsible for all computations

- The `server` monitors *inputs*
- When inputs change, *outputs* are updated (*reactivity*)

3 User-interface

3.1 Shiny Widgets Gallery

shiny.rstudio.com/gallery/widget-gallery.html

3.2 Example

Inputs have unique ids that correspond to server-side variables, a label, a starting value and extra options (e.g., range restrictions, etc.)

```
textInput(inputId="familyname", label="Family
name:",  value="Steve Miller"  )
```

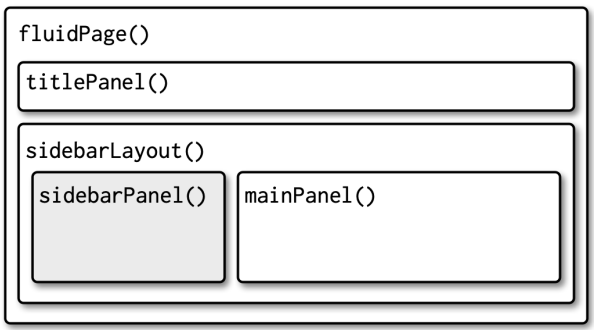
or

```
numericInput(inputId="age", label="Age (in years):",
value=1, min=0, max=150 )
```

On the server, we will be able to access variables `input$familyname` and `input$age`

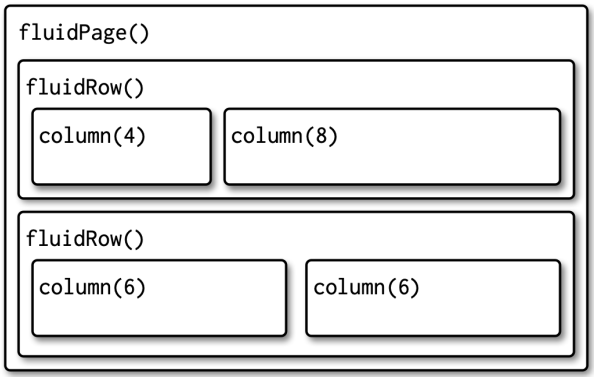
4 Layout

4.1 Sidebar layout



From Mastering Shiny

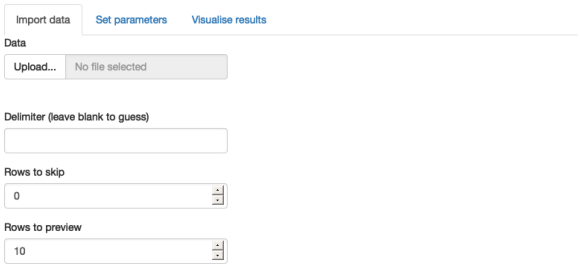
4.2 Multi-row layout



From Mastering Shiny

4.3 Other layouts

Many more, e.g. Tabsets - see `tabsetPanel()`



From Mastering Shiny

4.4 Outputs

Example output elements (placeholders for dynamic content):

- `textOutput()` or `htmlOutput()`
- `plotOutput()`
- `tableOutput()`

You can use

```
help.search("Output", package = "shiny")
```

`starte den http Server für die Hilfe fertig`

to find other output functions in shiny.

4.5 Outputs and Renderers

Each `*Output()` function has a corresponding `render*()` server-side function. For example:

- `textOutput()` → `renderText()`
- `plotOutput()` → `renderPlot()`
- `tableOutput()` → `renderTable()`

4.6 Server logic: Accessing inputs

```
shinyApp(
  ui = list(),
  server = function(input, output, session) {  }
)
```

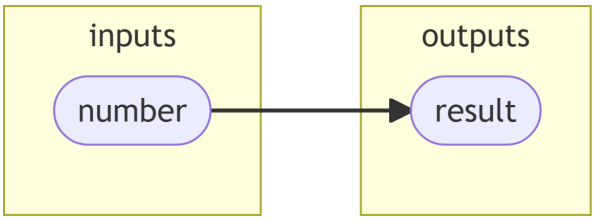
- Inputs are accessed in the *server* function via the *input* argument.
- Inputs are *reactive*, meaning that changes trigger updates to outputs.

5 Example: A pocket calculator

5.1 Demo 1 - Plus One

Demo: We write a simple calculator that adds +1 to a number we enter.

The simplest structure of a reactive program involves just a source and an endpoint:



5.2 Demo 1 - Plus One

R/demo1.R

```
library(shiny)

# Define UI for application that draws a histogram
ui <- fluidPage(

  # Application title
  titlePanel("Calculator"),

  # Sidebar with a slider input for number of bins
  sidebarLayout(
    sidebarPanel(
      numericInput("number",
                   "Number", value=0)
    ),

    # Show a plot of the generated distribution
    mainPanel(
      h3("Result"),
```

```
      textOutput("result")
    )
  )
)

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

  output$result <- renderText({
    return(input$number + 1)
  })
}

# Run the application
shinyApp(ui = ui, server = server)
```

5.3 Seeking AI help

Large language models are great companions for programming

Here is a ChatGPT link (requires Microsoft or Google account) to answer your questions (but please ask us as well any time)

ChatGPT companion for Shiny

5.4 Your turn - Exercise 1

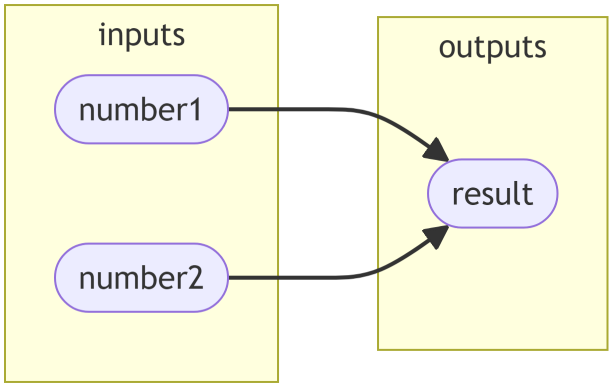
Copy the code from the previous slide (or open `R/demo1.R`) and run it in R

Check that you are able successfully run the shiny app and are able to interact with it.

- If everything is working try modifying the code (e.g. try adding a second number input and change the logic so that both numbers are added).

5.5 Reactive diagram

The reactive diagram of this solution shows two inputs and one output:



5.6 Solution

R/solution1_1.R

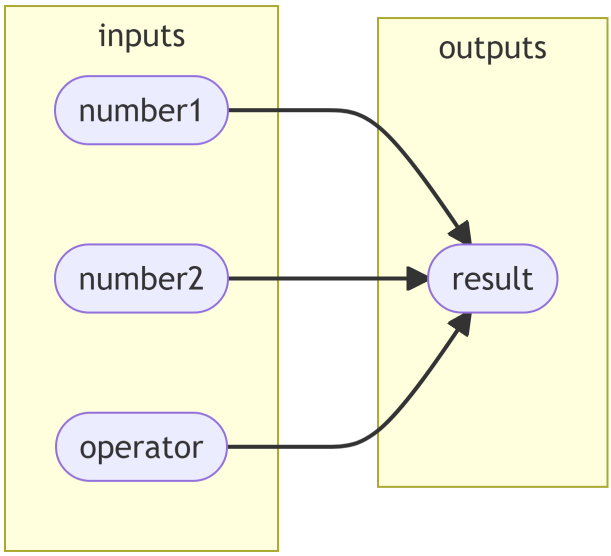
```
library(shiny)

# Define UI for application that draws a histogram
ui <- fluidPage(

    # Application title
    titlePanel("Calculator"),

    # Sidebar with a slider input for number of bins
    sidebarLayout(
        sidebarPanel(
            numericInput("n1",
                         "Number", value=0),
            numericInput("n2",
                          "Number", value=0)
        ),

        # Show a plot of the generated distribution
        mainPanel(
           textOutput("result")
        )
    )
)
```

```r

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

    output$result <- renderText({
        return(input$n1+input$n2)
    })
}

# Run the application
shinyApp(ui = ui, server = server)
```

5.7 Your Turn - Exercise 2

- Continue with your code (or from `R/solution1_1.R`) and add a menu to choose different operators (e.g., plus, minus, ...)
- For example, add a `selectInput(inputId, label, choices)`
- Add server-side logic to implement the different operators

5.8 Reactive diagram



5.9 Solution

R/solution1_2.R

```
library(shiny)

# Define UI for application that draws a histogram
ui <- fluidPage(

    # Application title
    titlePanel("Calculator"),

    # Sidebar with a slider input for number of bins
    sidebarLayout(
        sidebarPanel(
            numericInput("n1",
                         "Number", value=0),
            numericInput("n2",
                          "Number", value=0),
            selectInput("operator","Operator",c("+","-","/","*"))
```

```
        ),

        # Show a plot of the generated distribution
        mainPanel(
           textOutput("result")
        )
    )
)

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

    output$result <- renderText({
       result <- switch (input$operator,
          "+" = input$n1+input$n2,
          "-" = input$n1-input$n2,
          "/" = input$n1/input$n2,
          "*" = input$n1*input$n2
        )
        return(result)
    })
}

# Run the application
shinyApp(ui = ui, server = server)
```

5.10 Formatting text

We can use HTML elements to style text. E.g.,

```
cat("<b>Bold</b> or <i>Italics</i>,h1>First-level heading</h> <h2>Second-level heading</h2>,
```

```
<b>Bold</b> or <i>Italics</i>,h1>First-level heading</h> <h2>Second-level heading</h2>, ...
```

In UI as static or dynamic elements:

```
    h2("Title"),
    htmlOutput(outputId = "result")
```

On the server:

```
output$result <- renderText({ "<h2>Headline</h2>" })
```

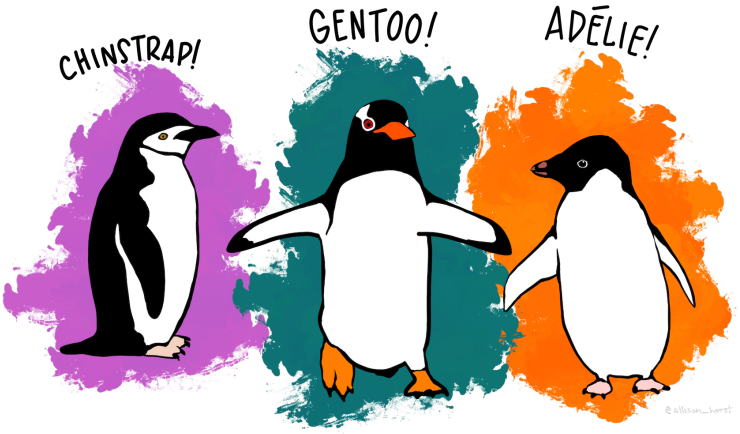
5.11 Solution

R/solution1_3.R

```
library(shiny)

# Define UI for application that draws a histogram
ui <- fluidPage(

    # Application title
    titlePanel("Calculator"),

    # Sidebar with a slider input for number of bins
    sidebarLayout(
        sidebarPanel(
            numericInput("n1",
                         "Number", value=0),
            selectInput("operator","Operator",c("+","-","/","*")),
            numericInput("n2",
                          "Number", value=0)
        ),

        # Show a plot of the generated distribution
        mainPanel(
           shiny::h2("Result:"),
           htmlOutput("result")
        )
    )
)

# Define server logic required to draw a histogram
```

```
server <- function(input, output) {

    output$result <- renderText({
       result <- switch (input$operator,
          "+" = input$n1+input$n2,
          "-" = input$n1-input$n2,
          "/" = input$n1/input$n2,
          "*" = input$n1*input$n2
        )

       result <- paste0("<h2>",result,"</h2>")

        return(result)
    })
}

# Run the application
shinyApp(ui = ui, server = server)
```

6 Who doesn't like penguins?



Artwork by @allison_horst

6.1 Palmer Penguins

We are going to use the `penguins` dataset from `palmerpenguins`

```
library(palmerpenguins)
```

```
Warning: Paket 'palmerpenguins' wurde unter R Version 4.2.3 erstellt
```

```
knitr::kable(head(penguins))
```

species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex	year
Adelie	Torgersen	39.1	18.7	181	3750	male	2007
Adelie	Torgersen	39.5	17.4	186	3800	female	2007
Adelie	Torgersen	40.3	18.0	195	3250	female	2007
Adelie	Torgersen	NA	NA	NA	NA	NA	2007
Adelie	Torgersen	36.7	19.3	193	3450	female	2007
Adelie	Torgersen	39.3	20.6	190	3650	male	2007

6.2 Reactive expression

R/challenge2.R

```
library(shiny)
library(tidyverse)
library(palmerpenguins)

# Define UI for application that draws a histogram
ui <- fluidPage(

    # Application title
    titlePanel("Penguins"),

    # Sidebar with a slider input for number of bins
    sidebarLayout(
        sidebarPanel(
            # <-------   here go input elements
        ),
```

```
16
17          # Show a plot of the generated distribution
18          mainPanel(
19             plotOutput("plot1"),
20             plotOutput("plot2"),
21             textOutput("text1")
22             # <-------    add more outputs here if needed
23          )
24      )
25  )
26
27  # Define server logic required to draw a histogram
28  server <- function(input, output) {
29
30      output$plot1 <- renderPlot({
31         penguins %>% ggplot(aes(x=body_mass_g,y=bill_length_mm))+
32          geom_point()+
33          geom_smooth(method = "lm")
34      })
35
36      output$plot2 <- renderPlot({
37        # <------ generate plot here (ggplot, or base R)
38      })
39
40      output$text1 <- renderText({
41        # <------- generate some text here
42      })
43  }
44
45  # Run the application
46  shinyApp(ui = ui, server = server)
```

6.3 Your Turn - Exercise 3

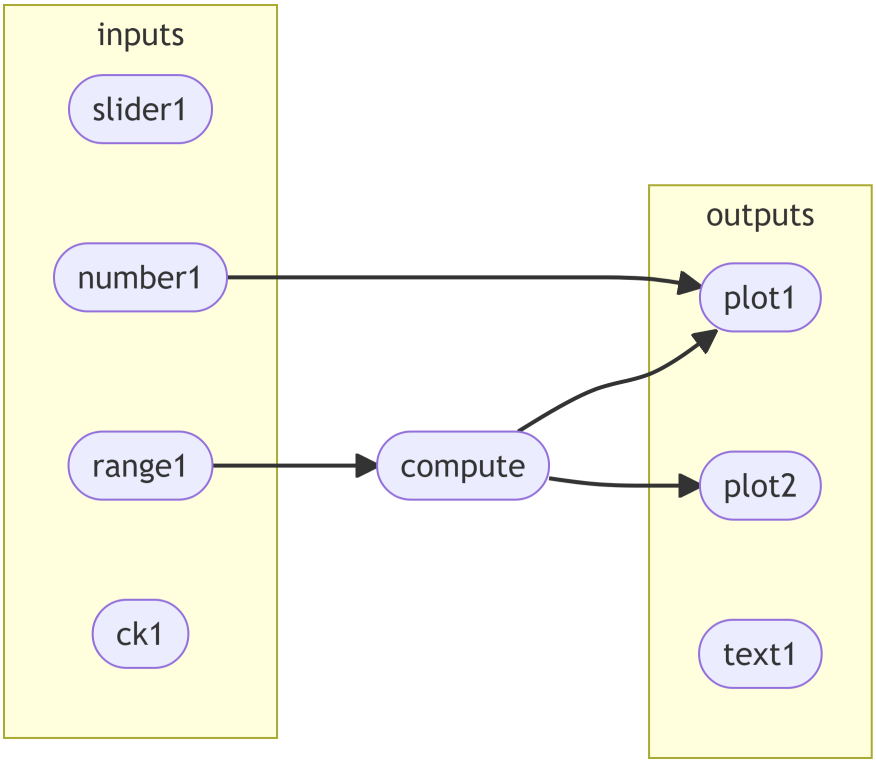
- Copy the code from the previous slide (or open `R/challenge2.R`) and run it in R
- Add logic to create a second plot as output `plot2` on the server

- Add extra inputs (e.g., add a `selectInput` for subgroup selection of penguin species) or add a `rangeInput` to display only certain ranges of years, or make point size adjustable by a given variable (`selectInput` or a `checkboxInput`).

6.4 DRY - Don't repeat yourself

- Assume a range input (`sliderInput(value=c(0,10))`) that filters data
- Filter logic should be executed only once for every relevant output
- Never copy&paste server logic, instead use a `reactive` element

6.5 DRY - Don't repeat yourself



6.6 Reactives

Their primary use is similar to a function in an R script, they help to

- avoid repeating yourself
- decompose complex computations into smaller / more modular steps
- can improve computational efficiency by breaking up / simplifying reactive dependencies

6.7 DRY - Solution

`R/demo3.R`

```r
library(shiny)
library(tidyverse)
library(palmerpenguins)

# Define UI for application that draws a histogram
ui <- fluidPage(

    # Application title
    titlePanel("Penguins"),

    # Sidebar with a slider input for number of bins
    sidebarLayout(
        sidebarPanel(
            sliderInput("rng", "Range ",value=c(3000,5000),min=2700, max=6300),
            selectInput("size", label="Size", choices=c("flipper_length_mm","bill_length_mm"
            checkboxInput("grp",label="Subgroups", value=TRUE)
        ),

        # Show a plot of the generated distribution
        mainPanel(
           plotOutput("plot1"),
           plotOutput("plot2"),
           textOutput("text1")
        )
    )
)

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

    penguins_filtered <- reactive({
      penguins %>% filter(body_mass_g >= input$rng[1] & body_mass_g <= input$rng[2])
    })

    output$plot1 <- renderPlot({

      wf <- NULL
      if (input$grp) {
        wf <- facet_wrap(~species)
      }
```

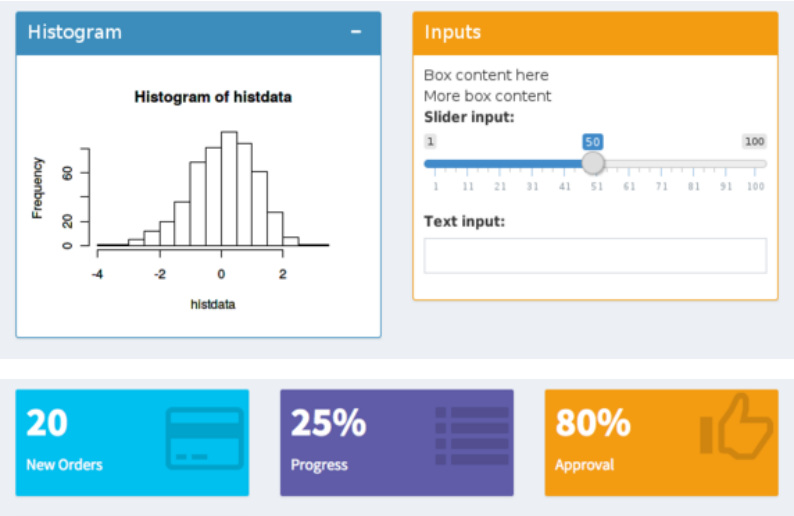
```
41
42          penguins_filtered() %>% ggplot(aes(x=body_mass_g,y=bill_length_mm))+
43           geom_point(aes_string(size=input$size))+
44           geom_smooth(method = "lm")+
45           wf
46       })
47
48       output$plot2 <- renderPlot({
49         if (input$grp) {
50           penguins_filtered() %>% ggplot(aes(x=flipper_length_mm,fill=species))+geom_histogram
51         } else {
52           penguins_filtered() %>% ggplot(aes(x=flipper_length_mm))+geom_histogram()
53         }
54
55       })
56
57       output$text1 <- renderText({
58         paste0("<b>There</b> are ",nrow(penguins_filtered()), " penguins in the data set")
59       })
60   }
61
62   # Run the application
63   shinyApp(ui = ui, server = server)
```

6.8 Deployment

- Free online deployment at https://www.shinyapps.io/ after registration
- Free account limited (e.g., 25h operating hours, 5 apps; more plans available)
- Sharing your app for others to run it locally (e.g., via OSF)
- Reproducibility! Make sure that everything is contained, no absolute file paths were used (see `here` package) and that all dependencies are loaded

6.9 Dashboards

Package `shinydashboard` has some nice GUI elements for dashboards:



6.10 Demo Dashboard

R/demo7.R

```
library(shinydashboard)

ui <- dashboardPage(
  dashboardHeader(title = "Value boxes"),
  dashboardSidebar(),
  dashboardBody(
    fluidRow(
      # A static valueBox
      valueBox(20, "New Orders", icon = icon("credit-card")),

      # Dynamic valueBox
      valueBoxOutput("progressBox"),

    ),
    fluidRow(
      # Clicking this will increment the progress amount
```

```
      box(width = 4, actionButton("count", "Do some work"))
    )
  )
)

server <- function(input, output) {
  output$progressBox <- renderValueBox({

    if (input$count < 10) {
      ic <- icon("thumbs-down")
      col <- "red"
    } else {
      ic <- icon("thumbs-up")
      col <- "green"
    }

    valueBox(
      paste0(input$count, "%"), "Progress", icon = ic,
      color = col
    )
  })


}

shinyApp(ui, server)
```

6.11 Simulation

Shiny is useful for simulating data (multivariate distributions, network graphs, agents, …)

- Inputs allow us to vary simulation parameters
- Outputs display simulation results
- We use a `reactive()` to generate our dataset, so that it can be reused in different places
- `downloadButton` and `downloadHandler` allow us to download the simulated data files for later analyses

6.12 Simulation Stub

R/demo6.R

```
library(shiny)

# Define UI for application that draws a histogram
ui <- fluidPage(

    # Application title
    titlePanel("Simulation"),

    # Sidebar with a slider input for number of bins
    sidebarLayout(
        sidebarPanel(
            numericInput("N",
                         "Sample Size", value=100),
            downloadButton("download")

        ),

        # Show a plot of the generated distribution
        mainPanel(
           plotOutput("graph")
        )
    )
)

# Define server logic
server <- function(input, output) {

  sim <- reactive({
   # <----- create a simulated dataset here
  })

  # return the dataset as file
  output$download = downloadHandler(
    filename = function() {
      "simulation.csv"
    },
```

```
      content = function(file) {
        readr::write_csv(sim(), file)
      }
    )

    output$graph <- renderPlot({

      # <------ do some plotting here

    })


}

# Run the application
shinyApp(ui = ui, server = server)
```

6.13 Your Turn - Exercise 4

Copy the code from the previous slide (or open `R/demo6.R`) and run it in R

- Add logic to simulate data (e.g., using `rnorm` or `MASS::mvrnorm`)
- Add a plot to show the simulation results (e.g., a scatterplot)
- Add extra features to make the simulation interactive

6.14 Simulation Solution

`R/solution6.R`

```
library(shiny)

# Define UI for application
ui <- fluidPage(

    # Application title
```

```
    titlePanel("Simulation"),

    # Sidebar with a slider input for number of bins
    sidebarLayout(
        sidebarPanel(
            numericInput("N",
                         "Sample Size", value=100),
            numericInput("r",
                         "Correlation", value=0),
            downloadButton("download")

        ),

        # Show a plot of the generated distribution
        mainPanel(
           plotOutput("graph")
        )
    )
)

# Define server logic
server <- function(input, output) {

  sim <- reactive({
    r = input$r
    N = input$N

    df <- MASS::mvrnorm(n=N, mu=c(0,0),
                        Sigma=matrix(c(1,r,
                                       r,1),
                                     nrow=2))

    df <- data.frame(df)
    names(df) <- c("x","y")

    return(df)
  })

  output$download = downloadHandler(
    filename = function() {
```

```
      "simulation.csv"
    },
    content = function(file) {
      readr::write_csv(sim(), file)
    }
  )

    output$graph <- renderPlot({

       sim() %>% ggplot(aes(x=x,y=y))+ geom_point()+geom_smooth(method = "lm")

    })


}

# Run the application
shinyApp(ui = ui, server = server)
```

6.15 Inspiration

shiny.rstudio.com/gallery/

> The Shiny User Showcase is comprised of contributions from the Shiny app developer community.

6.16 Your turn - go wild!

6.17 License

6.18 Thanks

Thank you for being on this journey with us!

Andreas (https://www.brandmaier.de; also find me on Twitter), Bluesky, Linkedin)