

# An Introduction to Shiny

A hands-on workshop

Andreas M. Brandmaier and Leonie Hagitte

2024-03-14

## Table of contents

<b>1</b>	<b>Herzlich Willkommen!</b>	<b>3</b>
1.1	Ask questions anytime . . . . .	3
1.2	Shiny . . . . .	3
1.3	How do Shiny apps look like? . . . . .	4
1.4	Required software . . . . .	4
1.5	Workshop materials . . . . .	5
1.6	What to expect . . . . .	5
<b>2</b>	<b>Goals</b>	<b>6</b>
2.1	Objectives of today . . . . .	6
2.2	Content . . . . .	6
2.3	Anatomy of a Shiny app . . . . .	7
2.4	Anatomy of a Shiny app . . . . .	7
2.5	Anatomy of a Shiny app . . . . .	7
<b>3</b>	<b>User-interface</b>	<b>8</b>
3.1	Shiny Widgets Gallery . . . . .	8
3.2	Example . . . . .	8
<b>4</b>	<b>Layout</b>	<b>8</b>
4.1	Sidebar layout . . . . .	8
4.2	Multi-row layout . . . . .	9
4.3	Other layouts . . . . .	9
4.4	Outputs . . . . .	9
4.5	Outputs and Renderers . . . . .	10
4.6	Server logic: Accessing inputs . . . . .	10

<b>5</b>	<b>Example: A pocket calculator</b>	<b>11</b>
5.1	Demo 1 - Plus One . . . . .	11
5.2	Demo 1 - Plus One . . . . .	11
5.3	Seeking AI help . . . . .	12
5.4	Your turn - Exercise 1 . . . . .	13
5.5	Reactive diagram . . . . .	13
5.6	Solution . . . . .	14
5.7	Your Turn - Exercise 2 . . . . .	15
5.8	Reactive diagram . . . . .	16
5.9	Solution . . . . .	16
5.10	Formatting text . . . . .	17
5.11	Solution . . . . .	18
<b>6</b>	<b>Who doesn't like penguins?</b>	<b>19</b>
6.1	Palmer Penguins . . . . .	20
6.2	Reactive expression . . . . .	20
6.3	Your Turn - Exercise 3 . . . . .	21
6.4	DRY - Don't repeat yourself . . . . .	22
6.5	DRY - Don't repeat yourself . . . . .	23
6.6	Reactives . . . . .	23
6.7	DRY - Solution . . . . .	23
6.8	Deployment . . . . .	25
6.9	Dashboards . . . . .	26
6.10	Demo Dashboard . . . . .	26
6.11	Simulation . . . . .	27
6.12	Simulation Stub . . . . .	28
6.13	Your Turn - Exercise 4 . . . . .	29
6.14	Simulation Solution . . . . .	29
6.15	Inspiration . . . . .	31
6.16	Your turn - go wild! . . . . .	32
6.17	License . . . . .	32
6.18	Thanks . . . . .	32

# 1 Herzlich Willkommen!

## 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](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

```
1 library(shiny)
2
3 shinyApp(
4   ui = list(),
5   server = function(input, output, session) { }
6 )
```

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

```
1 library(shiny)
2
3 shinyApp(
4   ui = list(),
5   server = function(input, output, session) { }
6 )
```

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

```
1 library(shiny)
2
3 shinyApp(
4   ui = list(),
5   server = function(input, output, session) { }
6 )
```

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](https://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" )
```

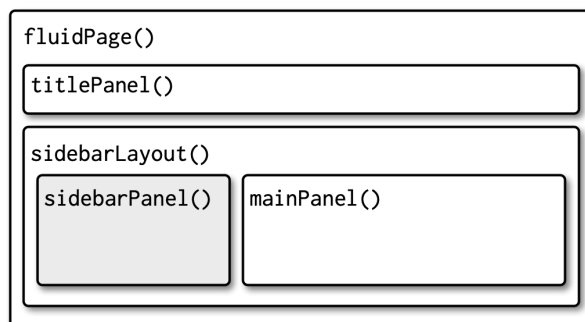
or

```
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

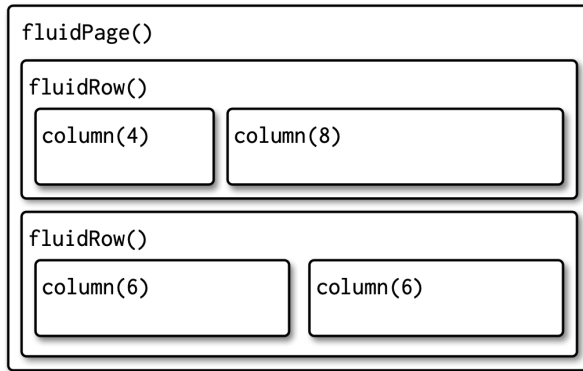
### 4.1 Sidebar layout





From [Mastering Shiny](#)

## 4.2 Multi-row layout



From [Mastering Shiny](#)

## 4.3 Other layouts

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

The screenshot shows a Shiny web application interface. At the top, there are three tabs: 'Import data', 'Set parameters', and 'Visualise results'. The 'Set parameters' tab is currently selected. Below the tabs, there is a 'Data' section with an 'Upload...' button and a message 'No file selected'. Below this, there are three input fields: 'Delimiter (leave blank to guess)', 'Rows to skip' (with a value of 0), and 'Rows to preview' (with a value of 10).

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

```
1 shinyApp(  
2   ui = list(),  
3   server = function(input, output, session) { }  
4 )
```

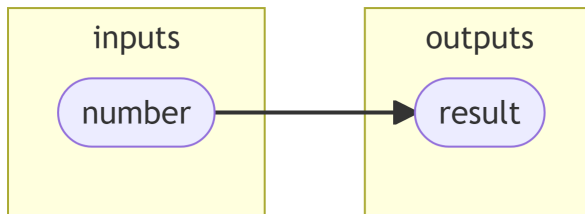
- Inputs are accessed in the *server* function via the *input* argument.
- Inputs are *reactive*, meaning that changes trigger updates to outputs.

## 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

```
1 library(shiny)
2
3 # Define UI for application that draws a histogram
4 ui <- fluidPage(
5
6   # Application title
7   titlePanel("Calculator"),
8
9   # Sidebar with a slider input for number of bins
10  sidebarLayout(
11    sidebarPanel(
12      numericInput("number",
13                  "Number", value=0)
14    ),
15
16    # Show a plot of the generated distribution
17    mainPanel(
18      h3("Result"),
```

```

19     textOutput("result")
20   )
21 )
22 )
23
24 # Define server logic required to draw a histogram
25 server <- function(input, output) {
26
27   output$result <- renderText({
28     return(input$number + 1)
29   })
30 }
31
32 # Run the application
33 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](#)



What is the name of the input element, where I can select from different options?



The input element you are referring to is called a "select input" in Shiny. It allows users to choose from a predefined list of options.



In Shiny, you can create a select input using the ``selectInput()`` function. It takes several arguments, including the input ID, label, choices, and selected option. Here's an example:

```
R Copy code

library(shiny)

ui <- fluidPage(
  selectInput(inputId = "fruit",
    label = "Select a fruit:",
    choices = c("Apple", "Banana", "Orange"),
    selected = "Apple")
)

server <- function(input, output) {
  # Access the selected value using input$fruit
}

shinyApp(ui, server)
```

## 5.4 Your turn - Exercise 1

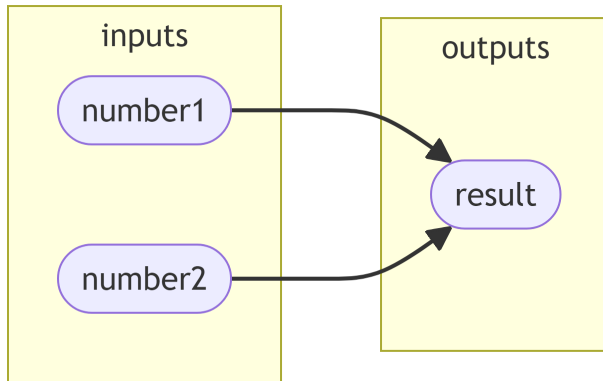
Copy the code from the previous slide (or open `R/demo1.R`) and run it in R

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

```
1 library(shiny)
2
3 # Define UI for application that draws a histogram
4 ui <- fluidPage(
5
6   # Application title
7   titlePanel("Calculator"),
8
9   # Sidebar with a slider input for number of bins
10  sidebarLayout(
11    sidebarPanel(
12      numericInput("n1",
13                    "Number", value=0),
14      numericInput("n2",
15                    "Number", value=0)
16    ),
17
18    # Show a plot of the generated distribution
19    mainPanel(
20      textOutput("result")
21    )
22  )
23 )
```

```

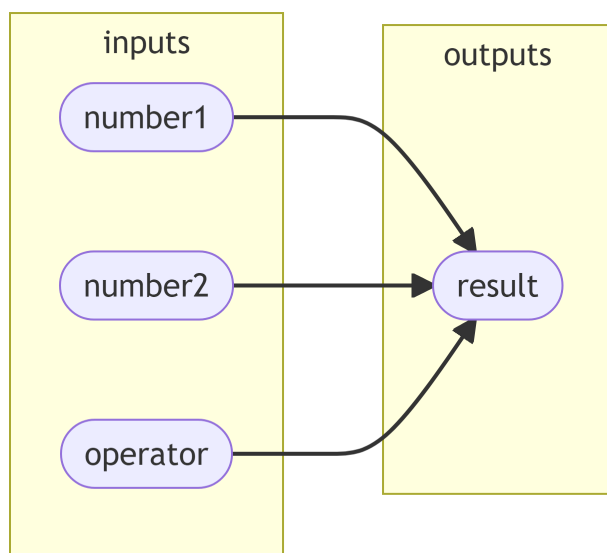
24
25 # Define server logic required to draw a histogram
26 server <- function(input, output) {
27
28     output$result <- renderText({
29         return(input$n1+input$n2)
30     })
31 }
32
33 # Run the application
34 shinyApp(ui = ui, server = server)

```

## 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

```
1 library(shiny)
2
3 # Define UI for application that draws a histogram
4 ui <- fluidPage(
5
6   # Application title
7   titlePanel("Calculator"),
8
9   # Sidebar with a slider input for number of bins
10  sidebarLayout(
11    sidebarPanel(
12      numericInput("n1",
13                  "Number", value=0),
14      numericInput("n2",
15                  "Number", value=0),
16      selectInput("operator", "Operator", c("+", "-", "/", "*"))
```



```

17         ),
18
19         # Show a plot of the generated distribution
20         mainPanel(
21             textOutput("result")
22         )
23     )
24 )
25
26 # Define server logic required to draw a histogram
27 server <- function(input, output) {
28
29     output$result <- renderText({
30         result <- switch (input$operator,
31             "+" = input$n1+input$n2,
32             "-" = input$n1-input$n2,
33             "/" = input$n1/input$n2,
34             "*" = input$n1*input$n2
35         )
36         return(result)
37     })
38 }
39
40 # Run the application
41 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

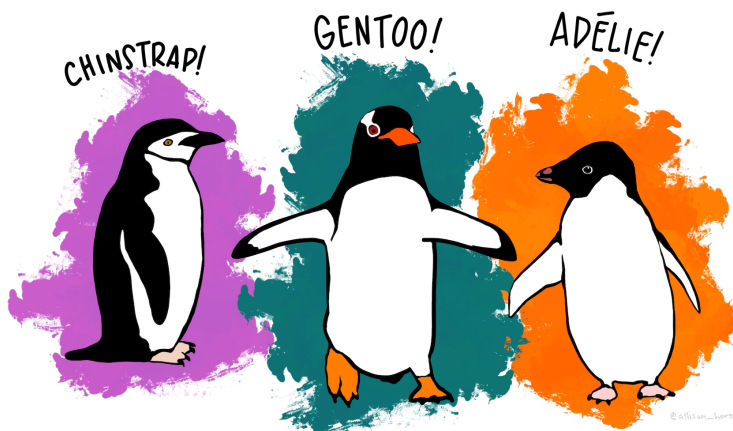
```
1 library(shiny)
2
3 # Define UI for application that draws a histogram
4 ui <- fluidPage(
5
6   # Application title
7   titlePanel("Calculator"),
8
9   # Sidebar with a slider input for number of bins
10  sidebarLayout(
11    sidebarPanel(
12      numericInput("n1",
13                  "Number", value=0),
14      selectInput("operator", "Operator", c("+", "-", "/", "*")),
15      numericInput("n2",
16                  "Number", value=0)
17    ),
18
19    # Show a plot of the generated distribution
20    mainPanel(
21      shiny::h2("Result:"),
22      htmlOutput("result")
23    )
24  )
25 )
26
27 # Define server logic required to draw a histogram
```

```

28 server <- function(input, output) {
29
30   output$result <- renderText({
31     result <- switch (input$operator,
32       "+" = input$n1+input$n2,
33       "-" = input$n1-input$n2,
34       "/" = input$n1/input$n2,
35       "*" = input$n1*input$n2
36     )
37
38     result <- paste0("<h2>",result,"</h2>")
39
40     return(result)
41   })
42 }
43
44 # Run the application
45 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	bill_depth	flipper_length	body_mass	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

```
1 library(shiny)
2 library(tidyverse)
3 library(palmerpenguins)
4
5 # Define UI for application that draws a histogram
6 ui <- fluidPage(
7
8   # Application title
9   titlePanel("Penguins"),
10
11   # Sidebar with a slider input for number of bins
12   sidebarLayout(
13     sidebarPanel(
14       # <----- here go input elements
15     ),
```

```

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

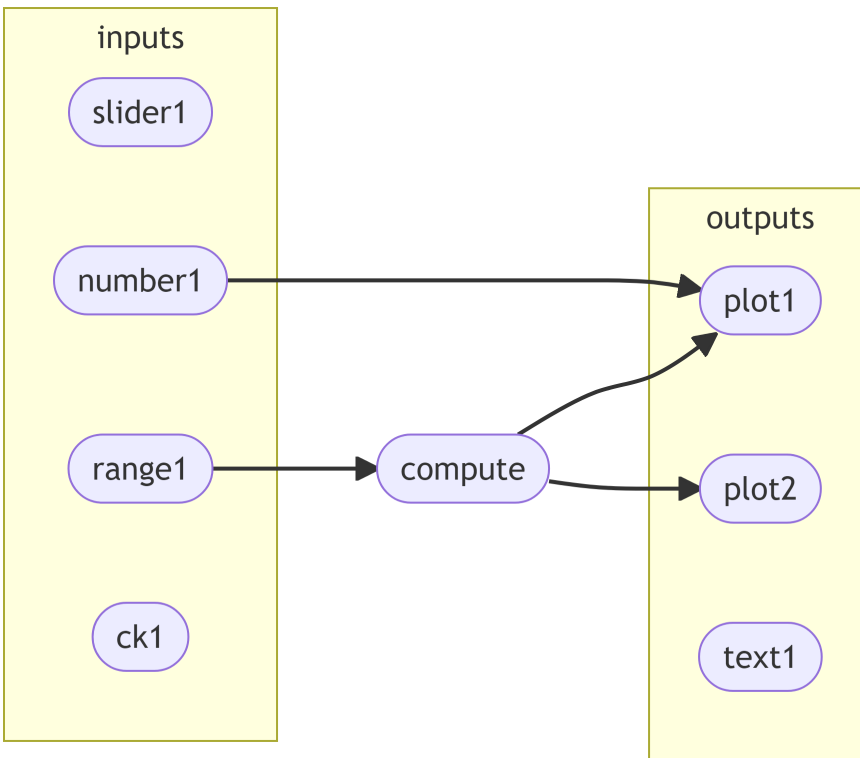
- Copy the code from the previous slide (or open R/challenge2.R) and run it in R
- Add logic to create a second plot as output `plot2` on the server

- 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

```

1 library(shiny)
2 library(tidyverse)
3 library(palmerpenguins)
4
5 # Define UI for application that draws a histogram
6 ui <- fluidPage(
7
8   # Application title
9   titlePanel("Penguins"),
10
11   # Sidebar with a slider input for number of bins
12   sidebarLayout(
13     sidebarPanel(
14       sliderInput("rng", "Range ",value=c(3000,5000),min=2700, max=6300),
15       selectInput("size", label="Size", choices=c("flipper_length_mm","bill_length_mm"),
16       checkboxInput("grp",label="Subgroups", value=TRUE)
17     ),
18
19     # Show a plot of the generated distribution
20     mainPanel(
21       plotOutput("plot1"),
22       plotOutput("plot2"),
23       textOutput("text1")
24     )
25   )
26 )
27
28 # Define server logic required to draw a histogram
29 server <- function(input, output) {
30
31   penguins_filtered <- reactive({
32     penguins %>% filter(body_mass_g >= input$rng[1] & body_mass_g <= input$rng[2])
33   })
34
35   output$plot1 <- renderPlot({
36
37     wf <- NULL
38     if (input$grp) {
39       wf <- facet_wrap(~species)
40     }

```



```

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)

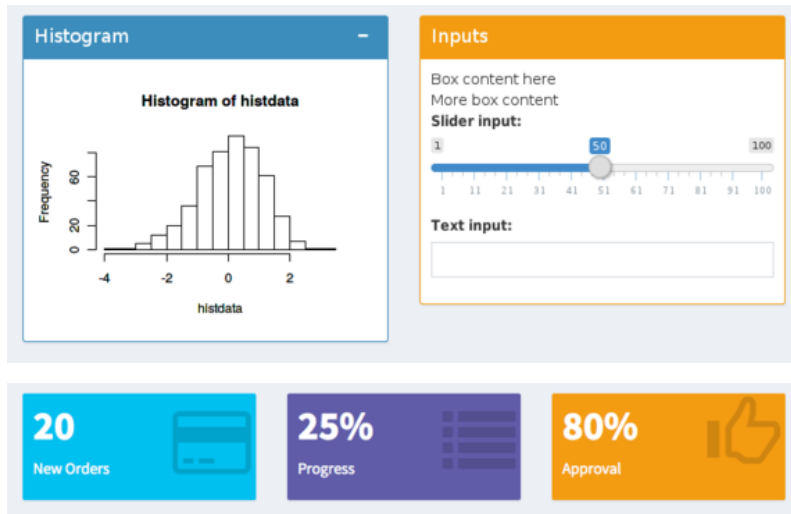
```

## 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`

```
1 library(shinydashboard)
2
3 ui <- dashboardPage(
4   dashboardHeader(title = "Value boxes"),
5   dashboardSidebar(),
6   dashboardBody(
7     fluidRow(
8       # A static valueBox
9       valueBox(20, "New Orders", icon = icon("credit-card")),
10
11       # Dynamic valueBox
12       valueBoxOutput("progressBox"),
13
14     ),
15     fluidRow(
16       # Clicking this will increment the progress amount
```

```

17     box(width = 4, actionButton("count", "Do some work"))
18   )
19 )
20 )
21
22 server <- function(input, output) {
23   output$progressBox <- renderValueBox({
24
25     if (input$count < 10) {
26       ic <- icon("thumbs-down")
27       col <- "red"
28     } else {
29       ic <- icon("thumbs-up")
30       col <- "green"
31     }
32
33     valueBox(
34       paste0(input$count, "%"), "Progress", icon = ic,
35       color = col
36     )
37   })
38
39 }
40
41
42 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

```
1  library(shiny)
2
3  # Define UI for application that draws a histogram
4  ui <- fluidPage(
5
6      # Application title
7      titlePanel("Simulation"),
8
9      # Sidebar with a slider input for number of bins
10     sidebarLayout(
11         sidebarPanel(
12             numericInput("N",
13                           "Sample Size", value=100),
14             downloadButton("download")
15         ),
16
17         # Show a plot of the generated distribution
18         mainPanel(
19             plotOutput("graph")
20         )
21     )
22 )
23
24
25 # Define server logic
26 server <- function(input, output) {
27
28     sim <- reactive({
29         # ----- create a simulated dataset here
30     })
31
32     # return the dataset as file
33     output$download = downloadHandler(
34         filename = function() {
35             "simulation.csv"
36         },
```

```

37     content = function(file) {
38       readr::write_csv(sim(), file)
39     }
40   )
41
42   output$graph <- renderPlot({
43
44     # <----- do some plotting here
45
46   })
47
48 }
49
50
51 # Run the application
52 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 scatter-plot)
- Add extra features to make the simulation interactive

### 6.14 Simulation Solution

📄 `R/solution6.R`

```

1  library(shiny)
2
3  # Define UI for application
4  ui <- fluidPage(
5
6    # Application title

```

```

7     titlePanel("Simulation"),
8
9     # Sidebar with a slider input for number of bins
10    sidebarLayout(
11      sidebarPanel(
12        numericInput("N",
13                     "Sample Size", value=100),
14        numericInput("r",
15                     "Correlation", value=0),
16        downloadButton("download")
17      ),
18    ),
19
20    # Show a plot of the generated distribution
21    mainPanel(
22      plotOutput("graph")
23    )
24  )
25 )
26
27 # Define server logic
28 server <- function(input, output) {
29
30   sim <- reactive({
31     r = input$r
32     N = input$N
33
34     df <- MASS::mvrnorm(n=N, mu=c(0,0),
35                        Sigma=matrix(c(1,r,
36                                     r,1),
37                                    nrow=2))
38
39     df <- data.frame(df)
40     names(df) <- c("x","y")
41
42     return(df)
43   })
44
45   output$download = downloadHandler(
46     filename = function() {

```

```

47     "simulation.csv"
48   },
49   content = function(file) {
50     readr::write_csv(sim(), file)
51   }
52 )
53
54 output$graph <- renderPlot({
55
56   sim() %>% ggplot(aes(x=x,y=y))+ geom_point()+geom_smooth(method = "lm")
57
58 })
59
60
61 }
62
63 # Run the application
64 shinyApp(ui = ui, server = server)

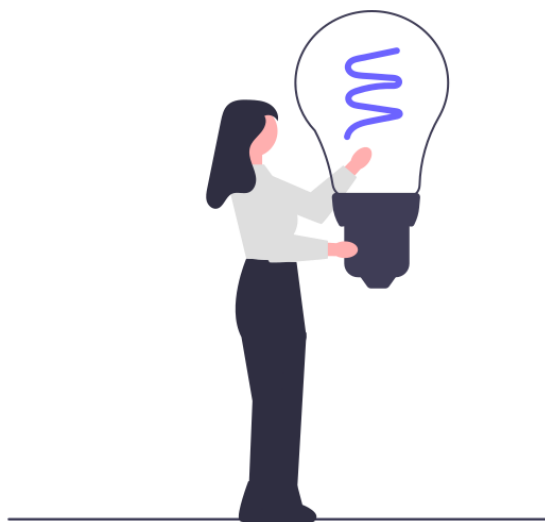
```

## 6.15 Inspiration

[shiny.rstudio.com/gallery/](https://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

To the extent possible under law and unless otherwise noted, Andreas and Leonie have waived all copyright and related or neighboring rights to this workshop document and the accompanying R source codes. This work is published from: Deutschland/Germany.

Some parts of this workshop are inspired by work by Colin Rundel (<https://github.com/rstudio-conf-2022/get-started-shiny/>), which is provided under <https://creativecommons.org/licenses/by/4.0/>.

Illustrations by undraw <https://undraw.co> (see their license <https://undraw.co/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)