Shiny application development

Migale facility - 2022/03/18

Sandra Dérozier - Mahendra Mariadassou - Cédric Midoux

(*Orignal slides from Amandine Velt and Cédric Midoux*)

RÉPUBLIQUE FRANÇAISE
Liberté Égalité Fraternité

INRAE migale

Practical informations

- 9h30 - 17h00
- 2 breaks (morning and afternoon)
- Lunch break of 1 hour
- Remote session … please be comprehensive!

Remote sessions rules

- Even remotely, it should be interactive!
- Please interrupt us:
  - raise (virtually) your hand
  - ask questions in the chat
- Practical session will be different:
  - tutorial support with quick exercises to do on your own (a few minutes each)
  - group synchronization to be sure everyone follows
  - Inform us of your progress:
    - Ask (any) questions*
    - Use green / red reactions

*except Cédric's weight and age, he's very protective about those ;-)

Ice breaking session

- Who are you?
  - Institution, laboratory, position…
- Why are you here?
  - What are your goals with Shiny?
  - Do you already have developments in progress?
- Have you ever used *Shiny*?

Objectives

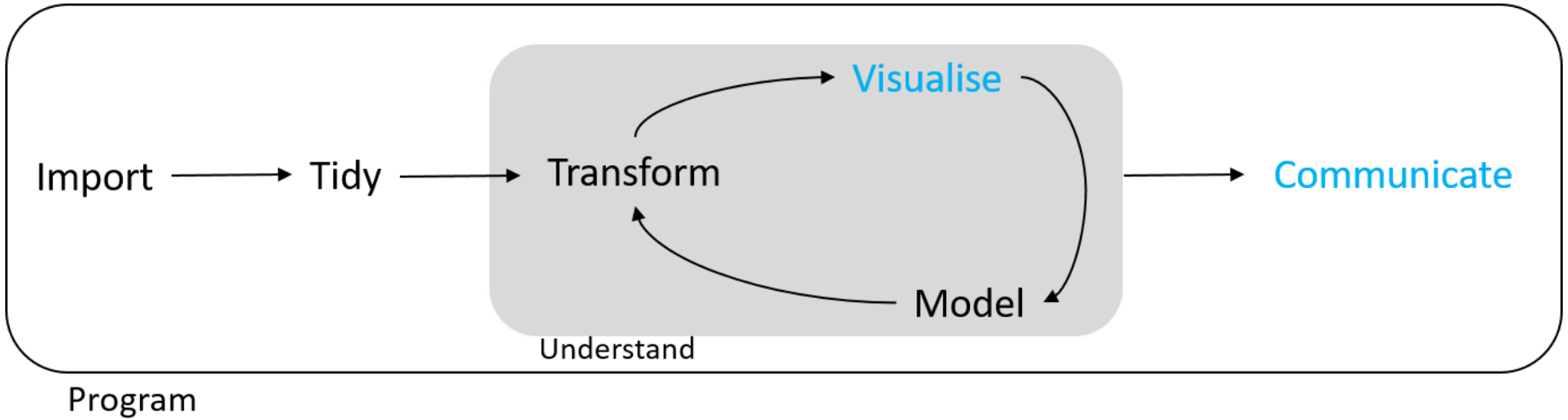
- After this training day, you will know:
  - General concepts of a Shiny app
  - Shiny application development
  - Shiny application deployment solutions

Schedule

- Morning
  - 9h30 : Opening
  - 10h : Why Shiny?
  - 10h30 : Server/UI + input/output + Widgets
- Afternoon
  - 13h30 : Layout
  - 14h : Reactivity
  - 15h : Deployment
  - 16h30 : Conclusion + Q&A

Why Shiny?

How to communicate one's results with R?



R for data analysis
Garrett Grolemund & Hadley Wickham

How to communicate its results with R?

With R, to change a parameter, you have to **change the code** manually.

```r
x <- faithful$waiting

# number of bins = 10
nbins <- 10
hist(x, breaks = nbins, col = "#75AADB",
     border = "white",
     xlab = "Waiting time to next eruption (in mins)",
     main = "Histogram of waiting times")

# number of bins = 50
nbins <- 50
hist(x, breaks = nbins, col = "#75AADB",
     border = "white",
     xlab = "Waiting time to next eruption (in mins)",
     main = "Histogram of waiting times")
```

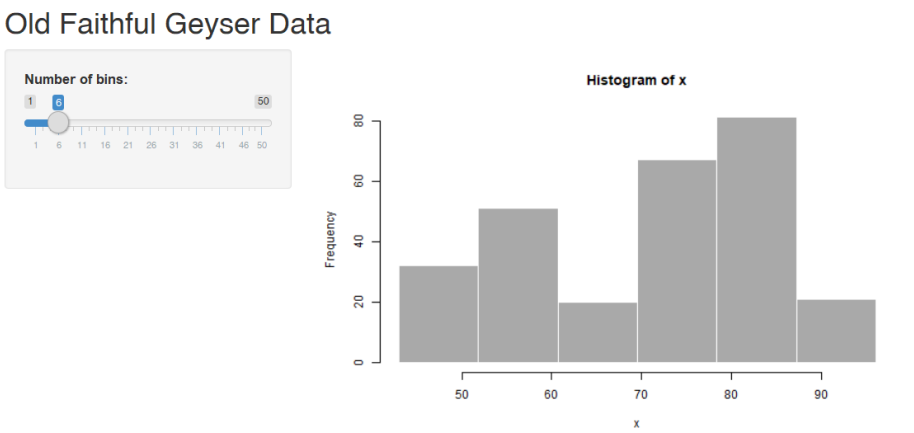
We would like to have a **slider** to modify it **interactively**!

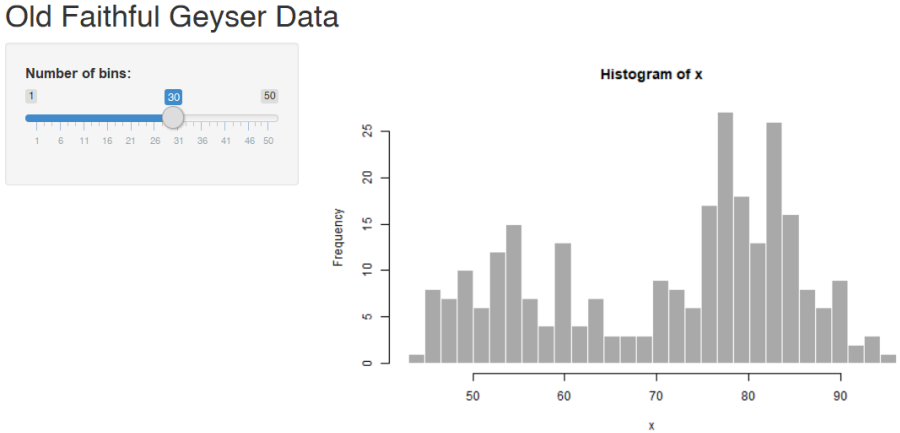
➔ Shiny solution: https://gallery.shinyapps.io/001-hello/

How to communicate one's results with R?

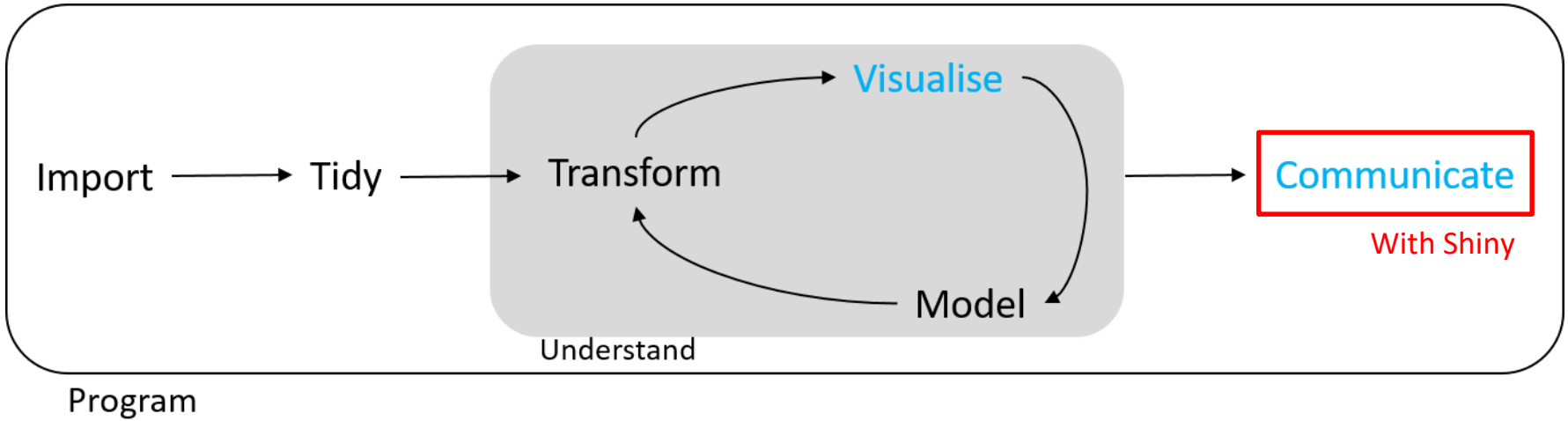
Shiny brings **reactivity**

```
shiny::runExample("01_hello")
```





How to communicate one's results with R?



R for data analysis
Garrett Grolemund & Hadley Wickham

Shiny to ease access to data

- Defining features:
  - Facilitates dataviz and data mining...
  - ...through simple, readable and user-friendly interfaces
  - ...and dynamic outputs: graphs, tables, etc
  - Reduces the required amount of code to learn / write (if you already know R)

How to communicate its results with R?

Share your raw code
- No context or comments
- Can be difficult to interpret
- No dependency management

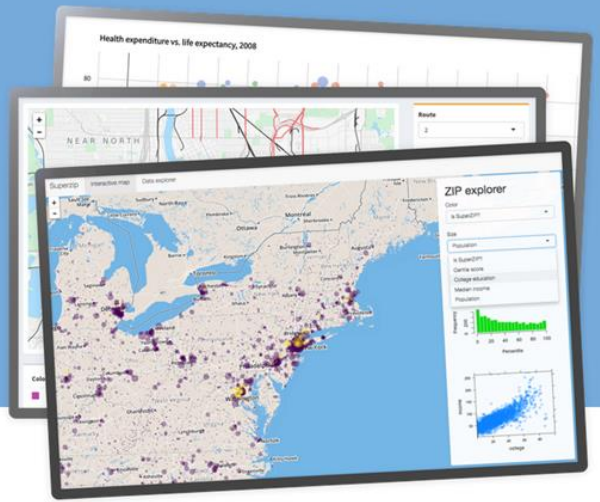
Use Rmarkdown
- Static report retracing analysis
- No re-execution

Implement a Shiny application
- User-frienldy graphical interface
- Editable parameters and re-executable code
- Ability to export plots and reports
- User should install R and packages

Deploy the Shiny app on a server
- Applications run on a dedicated serve

Shiny from RStudio

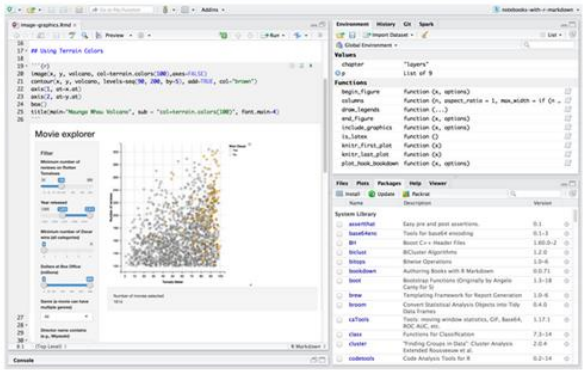
Get Started | Gallery | Articles | Reference | Deploy | Help | Contribute

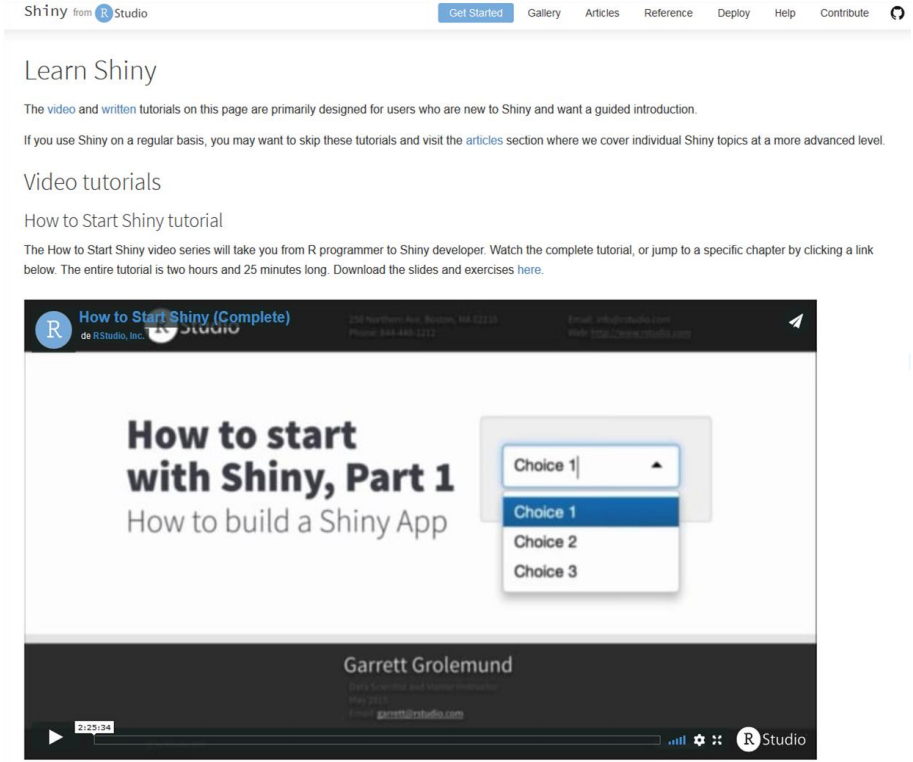


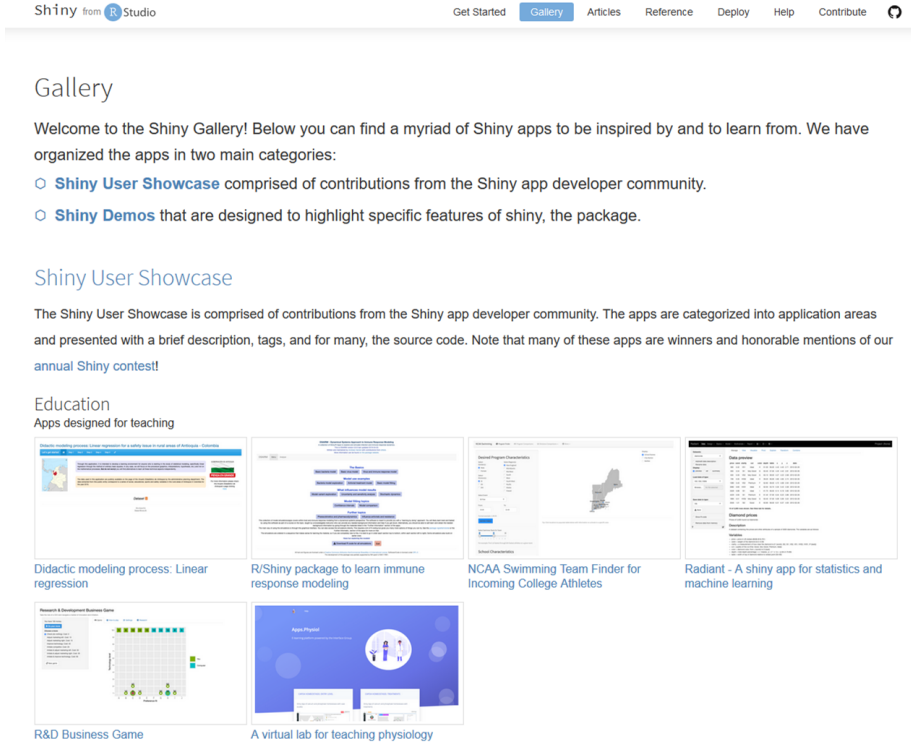
Interact. Analyze. Communicate.

Take a fresh, interactive approach to telling your data story with Shiny. Let users interact with your data and your analysis. And do it all with R.

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







Tutorials
https://shiny.rstudio.com/tutorial/

Gallery
https://shiny.rstudio.com/gallery/

Let's play with Shiny!

- Pick one of these Shiny apps and interact with it
- Try to see how the app behaves and what can be done with it
  - ECDC-vaccination et dataCoViz de @flodebarre
  - Covid19-tracker de Edward Parker
  - Movie explorer
  - Legislatives2017
  - Any one from the gallery

How does it work?

How does it work?

→ TP1

- Create a Shiny application with RStudio
  - File > New File > Shiny Web App
- Give a name to your first app and save it in the folder of your choice with the option "Single File"
- Browse, explore and execute the code
- Identify each element and their interactions

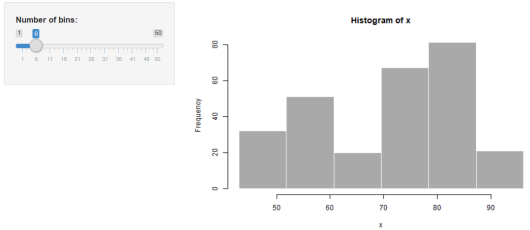
ui.R

```r
# This is the user-interface definition of a Shiny web application. You can
# run the application by clicking 'Run App' above.
#
# Find out more about building applications with Shiny here:
#
#    http://shiny.rstudio.com/
#

library(shiny)
library(shinythemes)

# Define UI for application that draws a histogram
ui <- fluidPage(theme=shinytheme("flatly"),

  # Application title
  titlePanel("Old Faithful Geyser Data"),

  # Sidebar with a slider input for number of bins
  sidebarLayout(
    sidebarPanel(
       sliderInput("bins",
                   "Number of bins:",
                   min = 1,
                   max = 50,
                   value = 30)
    ),

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

server.R

```r
#
# This is the server logic of a Shiny web application. You can run the
# application by clicking 'Run App' above.
#
# Find out more about building applications with Shiny here:
#
#    http://shiny.rstudio.com/
#

library(shiny)

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

  output$distPlot <- renderPlot({

    # generate bins based on input$bins from ui.R
    x    <- faithful[, 2]
    bins <- seq(min(x), max(x), length.out = input$bins + 1)

    # draw the histogram with the specified number of bins
    hist(x, breaks = bins, col = 'darkgray', border = 'white')

  })

}
```

Old Faithful Geyser Data



ui.R

server.R

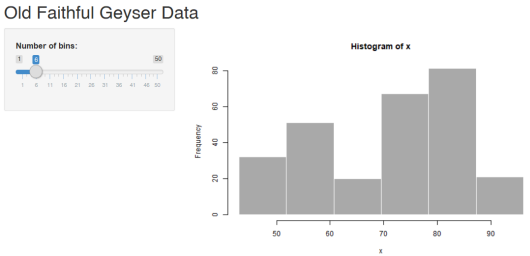
Inputs widgets

Load packages and data

Get values

Creation
Update
Output computation

Output display



ui.R

server.R

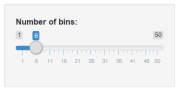
```
*Input(inputId = « in1 »,
       label = «lab»),
```

```
library(...)
```

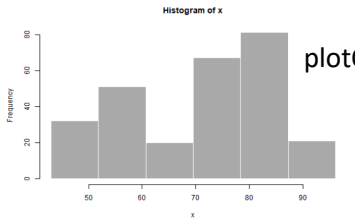
```
output$out1 <-
    render*(input$in1)
```

```
*Output(outputId = « out1 »)
```

Old Faithful Geyser Data



sliderInput



plotOutput

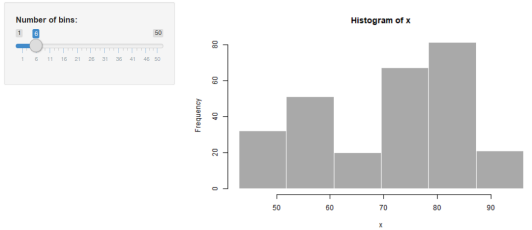
ui.R

```r
# This is the user-interface definition of a Shiny web application. You can
# run the application by clicking 'Run App' above.
#
# Find out more about building applications with Shiny here:
#
#    http://shiny.rstudio.com/
#

library(shiny)
library(shinythemes)

# Define UI for application that draws a histogram
ui <- fluidPage(theme=shinytheme("flatly"),

  # Application title
  titlePanel("Old Faithful Geyser Data"),

  # Sidebar with a slider input for number of bins
  sidebarLayout(
    sidebarPanel(
      sliderInput("bins",
                  "Number of bins:",
                  min = 1,
                  max = 50,
                  value = 30)
    ),

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

server.R

```r
#
# This is the server logic of a Shiny web application. You can run the
# application by clicking 'Run App' above.
#
# Find out more about building applications with Shiny here:
#
#    http://shiny.rstudio.com/
#

library(shiny)

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

  output$distPlot <- renderPlot({

    # generate bins based on input$bins from ui.R
    x    <- faithful[, 2]
    bins <- seq(min(x), max(x), length.out = input$bins + 1)

    # draw the histogram with the specified number of bins
    hist(x, breaks = bins, col = 'darkgray', border = 'white')

  })

}
```

Old Faithful Geyser Data



Various types of inputs widgets

ui.R

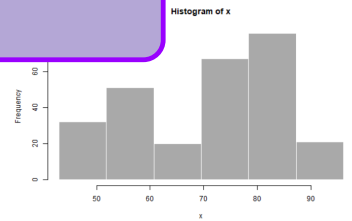
```
# This is the user-interface definition of a Shiny web application. You can
# run the application by clicking 'Run App' above
#
# Find out more about building applications with
#
#    http://shiny.rstudio.com/
#

library(shiny)
library(shinythemes)

# Define UI for application that draws a histogr
ui <- fluidPage(theme=shinytheme("flatly"),

  # Application title
  titlePanel("Old Faithful Geyser Data"),

  # Sidebar with a slider input for number of b
  sidebarLayout(
    sidebarPanel(
      sliderInput("bins",
                  "Number of bins:",
                  min = 1,
                  max = 50,
                  value = 30)
    ),

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

Inputs widgets

Which ones would you need?

server.R

```
pplication. You can run the

 with Shiny here:

stogram
{

m ui.R
= input$bins + 1)

 number of bins
, border = 'white')
```



Various types of inputs widgets

ui.R

```r
# This is the user-interface definition of a Shiny web application. You can
# run the application by clicking 'Run App' above
#
# Find out more about building applications with
#
#    http://shiny.rstudio.com/
#

library(shiny)
library(shinythemes)

# Define UI for application that draws a histogr
ui <- fluidPage(theme=shinytheme("flatly"),

  # Application title
  titlePanel("Old Faithful Geyser Data"),

  # Sidebar with a slider input for number of b
  sidebarLayout(
    sidebarPanel(
      sliderInput("bins",
                  "Number of bins:",
                  min = 1,
                  max = 50,
                  value = 30)
    ),

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

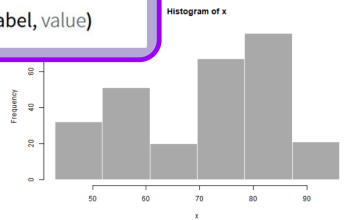
server.R

```r
pplication. You can run the

 with Shiny here:

stogram
{

m ui.R
= input$bins + 1)

 number of bins
, border = 'white')
```

Inputs widgets https://shiny.rstudio.com/gallery/widget-gallery.html

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

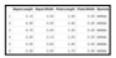
- Action — **actionButton**(inputId, label, icon, …)
- Link — **actionLink**(inputId, label, icon, …)
- Choice 1, Choice 2, Choice 3 — **checkboxGroupInput**(inputId, label, choices, selected, inline)
- Check me — **checkboxInput**(inputId, label, value)
- **dateInput**(inputId, label, value, min, max, 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, Choice B, Choice C — **radioButtons**(inputId, label, choices, selected, inline)
- Choice 1 / Choice 1, Choice 2 — **selectInput**(inputId, label, choices, selected, multiple, selectize, width, size) (also **selectizeInput()**)
- 0 5 10 — **sliderInput**(inputId, label, min, max, value, step, round, format, locale, ticks, animate, width, sep, pre, post)
- Apply Changes — **submitButton**(text, icon) (Prevents reactions across entire app)
- Enter text — **textInput**(inputId, label, value)



Various types of outputs

Outputs

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

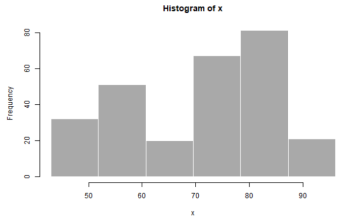
		works with	
	DT::**renderDataTable**(expr, options, callback, escape, env, quoted)		**dataTableOutput**(outputId, icon, …)
	**renderImage**(expr, env, quoted, deleteFile)		**imageOutput**(outputId, width, height, click, dblclick, hover, hoverDelay, hoverDelayType, brush, clickId, hoverId, inline)
	**renderPlot**(expr, width, height, res, …, env, quoted, func)		**plotOutput**(outputId, width, height, click, dblclick, hover, hoverDelay, hoverDelayType, brush, clickId, hoverId, inline)
	**renderPrint**(expr, env, quoted, func, width)		**verbatimTextOutput**(outputId)
	**renderTable**(expr,…, env, quoted, func)		**tableOutput**(outputId)
foo	**renderText**(expr, env, quoted, func)		**textOutput**(outputId, container, inline)
	**renderUI**(expr, env, quoted, func)		**uiOutput**(outputId, inline, container, …) & **htmlOutput**(outputId, inline, container, …)

```
29      mainPanel(
30         plotOutput("distPlot")
31      )
32   )
33)
```

server.R

```
1  #
2  # This is the server logic of a Shiny web application. You can run the
3  # application by clicking 'Run App' above.
4  #
5  # Find out more about building applications with Shiny here:
6  #
7  #    http://shiny.rstudio.com/
8  #
9
10 library(shiny)
11
12 # Define server logic required to draw a histogram
13 server <- function(input, output, session) {
14
15   output$distPlot <- renderPlot({
16
17     # generate bins based on input$bins from ui.R
18     x    <- faithful[, 2]
19     bins <- seq(min(x), max(x), length.out = input$bins + 1)
20
21     # draw the histogram with the specified number of bins
22     hist(x, breaks = bins, col = 'darkgray', border = 'white')
23
24   })
25
26 }
27
```

Old Faithful Geyser Data



Practical work

→ **TP2**: Add text

→ **TP3**: Pick a title

→ **TP4**: Add color

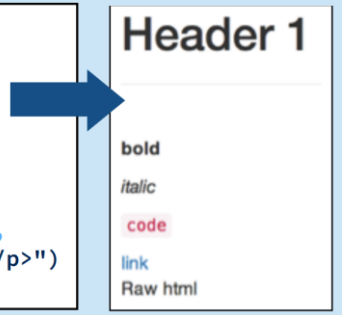
Layout

Layout

Add CSS / HTML / Javascript

- UI returns HTML document
- HTML tags can be added to the ui

```
ui <- fluidPage(
  h1("Header 1"),
  hr(),
  br(),
  p(strong("bold")),
  p(em("italic")),
  p(code("code")),
  a(href="", "link"),
  HTML("<p>Raw html</p>")
)
```




```
fluidPage(
  textInput("a","")
)
## <div class="container-fluid">
##  <div class="form-group shiny-input-container">
##     <label for="a"></label>
##     <input id="a" type="text"
##        class="form-control" value=""/>
##   </div>
## </div>
```

Returns HTML

- To include a CSS file, use the `includeCSS()` function and place the file in the `www` folder

```
includeCSS("styles.css"),
```

Layout

ui.R

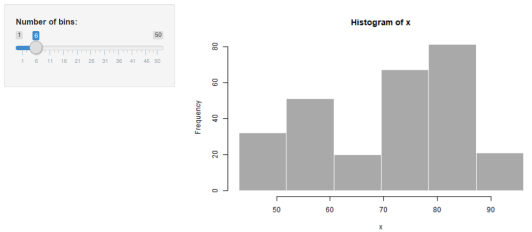
```
# This is the user-interface definition of a Shiny web application. You can
# run the applicati
#
# Find out more ab
#
#    http://shiny.
#

library(shiny)
library(shinythemes)

# Define UI for application that draws a histogram
ui <- fluidPage(theme=shinytheme("flatly"),

  # Application title
  titlePanel("Old Faithful Geyser Data"),

  # Sidebar with a slider input for number of bins
  sidebarLayout(
    sidebarPanel(
       sliderInput("bins",
                   "Number of bins:",
                   min = 1,
                   max = 50,
                   value = 30)
    ),

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

`fluidPage()` fits the width of the browser

server.R

```
#
# This is the server logic of a Shiny web application. You can run the
# application by clicking 'Run App' above.
#
# Find out more about building applications with Shiny here:
#
#    http://shiny.rstudio.com/
#

library(shiny)

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

  output$distPlot <- renderPlot({

    # generate bins based on input$bins from ui.R
    x    <- faithful[, 2]
    bins <- seq(min(x), max(x), length.out = input$bins + 1)

    # draw the histogram with the specified number of bins
    hist(x, breaks = bins, col = 'darkgray', border = 'white')

  })

}
```

Old Faithful Geyser Data



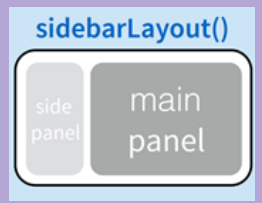
Layout

ui.R

```
# This is the user-interface definition of a Shiny web application. You can
# run the applicati
#
# Find out more ab
#
#    http://shiny.
#

library(shiny)
library(shinythemes)

# Define UI for application that draws a histogram
ui <- fluidPage(theme=shinytheme("flatly"),

  # Application title
  titlePanel("Old Faithful Geyser Data"

  # Sidebar with a slider input for nu
  sidebarLayout(
    sidebarPanel(
      sliderInput("bins"
                  "Number of bins:",
                  min = 1,
                  max = 50,
                  value = 30)
    ),

    # Show a plot of the generated dis
    mainPanel(
      plotOutput("distPlot")
    )
  )
)
```

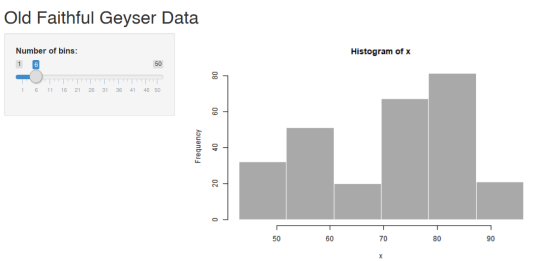
`fluidPage()` fits the width of the browser

Many layouts are available



server.R

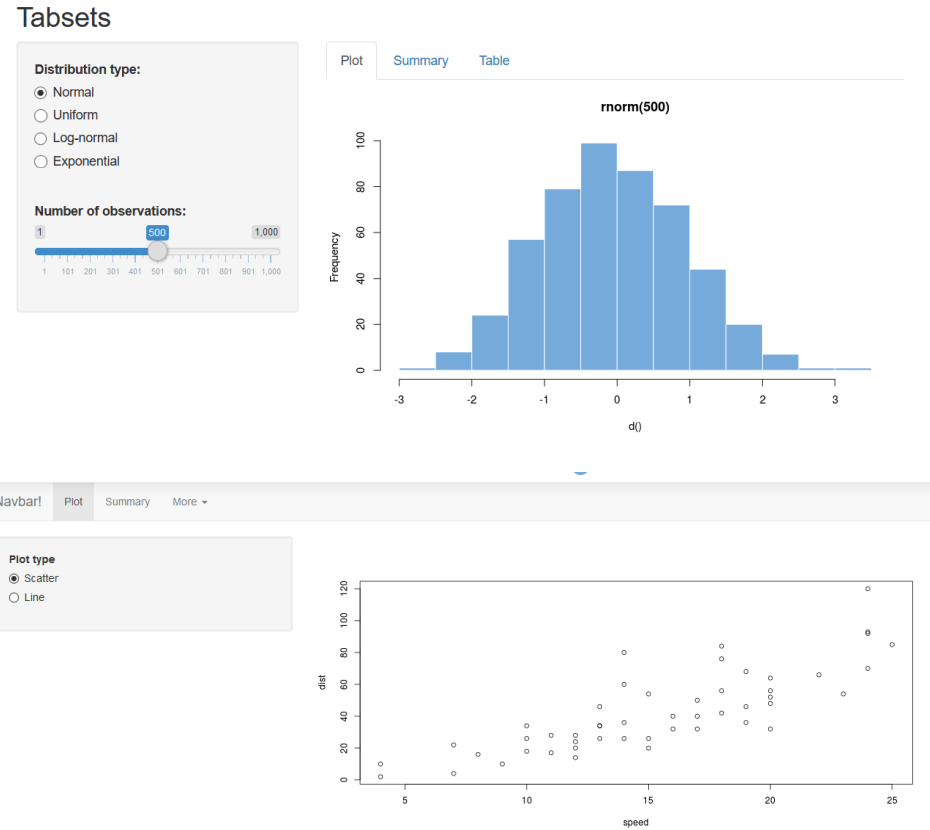
```
#
# This is the server logic of a Shiny web application. You can run the
# application by clicking 'Run App' above.
#
# Find out more about building applications with Shiny here:
#
#    http://shiny.rstudio.com/
#

library(shiny)

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

  output$distPlot <- renderPlot({

    # generate bins based on input$bins from ui.R
    x    <- faithful[, 2]
    bins <- seq(min(x), max(x), length.out = input$bins + 1)

    # draw the histogram with the specified number of bins
    hist(x, breaks = bins, col = 'darkgray', border = 'white')

  })

}
```



Layer tabPanels

```
ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(…)
    mainPanel(
          tabsetPanel(
        tabPanel("Plot", "content"),
        tabPanel("Summary", "content"),
        tabPanel("Table", "content")
          )
    )
  )
)
```

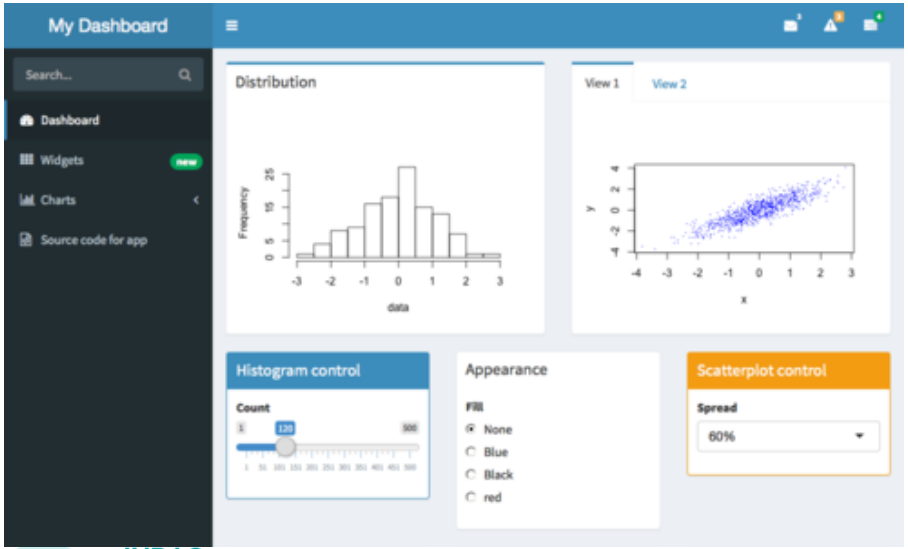
```
ui <- fluidPage(
navbarPage(title="Navbar!",
  tabPanel("Plot", "content"),
  tabPanel("Summary", "content"),
  tabPanel("More", "content")
))
```

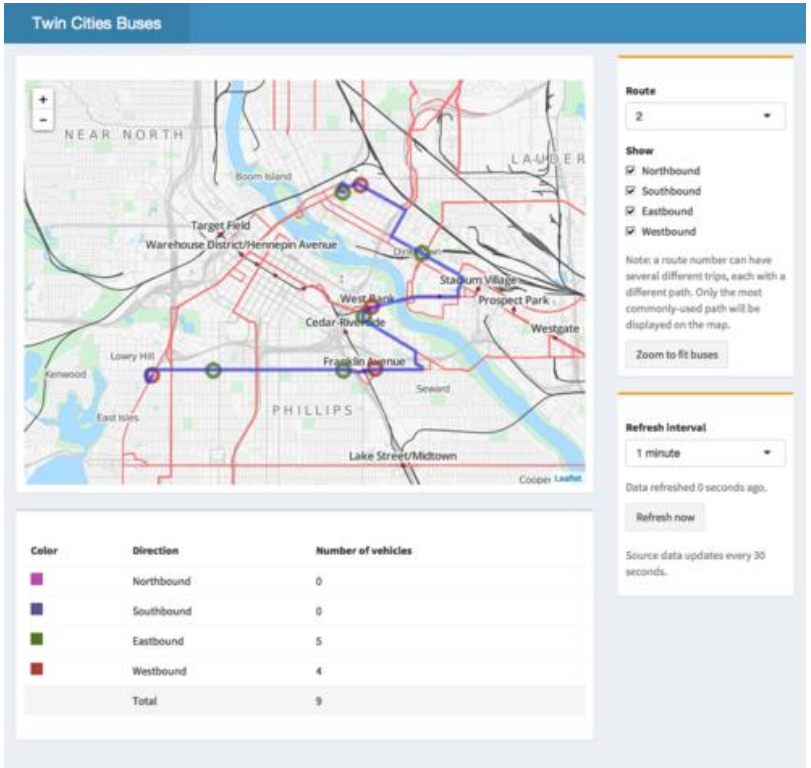


Shiny Dashboard

Using the `shinydashboard` package, you can easily create a dashboard type applications like:

```
library(shinydashboard)

dashboardPage(
  dashboardHeader(),
  dashboardSidebar(),
  dashboardBody()
)
```





CSS

3 ways to include CSS:

- insert the style in the **head**
- insert the style at the tag level, through the **style** attribute, also known as inline CSS
- point to an external file with **includeCSS(”**style.css**”)**

CSS

Insert the style in the **head**

ui.R

```
ui <- fluidPage(
  tags$head(
    tags$style(HTML("
      /* Change font of header text */
      h2 {
        font-family: 'monospace';
      }
      /* Make text visible on inputs */
      .shiny-input-container {
        color: #474747;
      }"))
  ),
  titlePanel("Old Faithful Geyser Data"),
  ...
)
```

CSS

Insert the style at the tag level, through the **style** attribute, also known as inline CSS

ui.R

```
ui <- fluidPage(

  # titlePanel("Old Faithful Geyser Data"),
  h2("Old Faithful Geyser Data", style = "font-family: monospace;"),

  ...
)
```

CSS

Point to an external file with **includeCSS(**“style.css”**)**

ui.R

```
ui <- fluidPage(
  includeCSS(“www/style.css”),
  titlePanel(“Old Faithful Geyser Data”),
  ...
)
```

style.css

```
/* Change font of header text */
h2 {
   font-family: ‘monospace’;
}

/* Make text visible on inputs */
.shiny-input-container {
   color: #474747;
}
```

Practical work

→ **TP5**: Multi-tab applications and customization with CSS

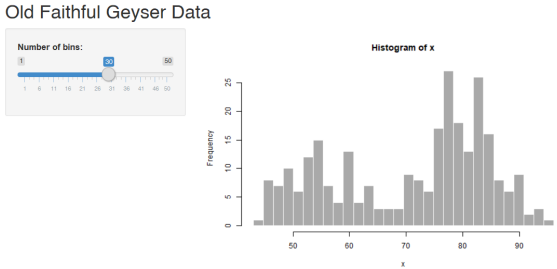
→ **TP6**: Simulations

Reactivity

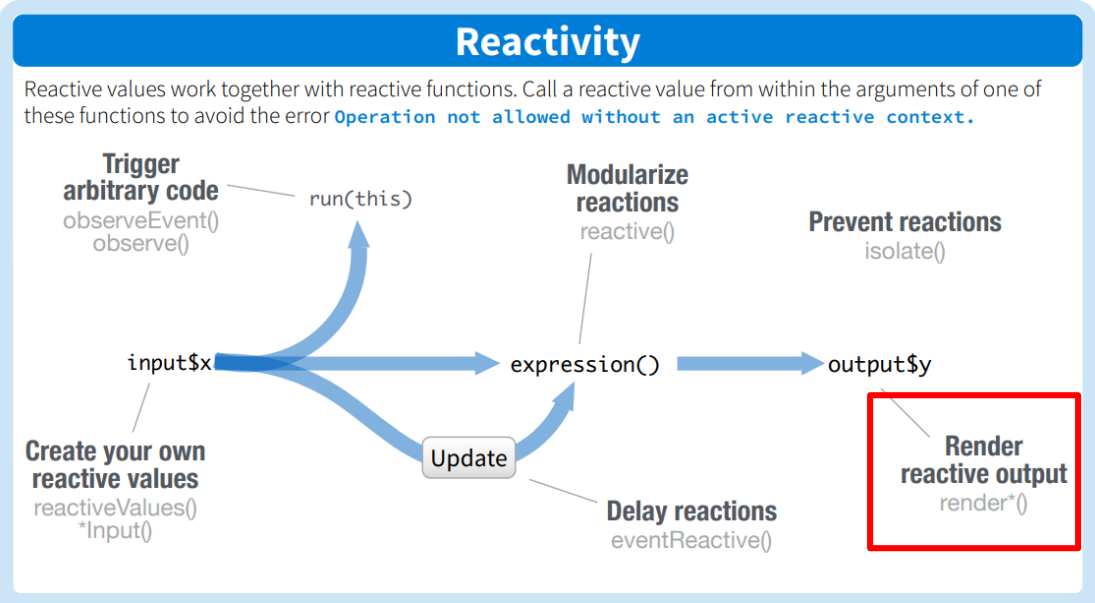
The reactivity of Shiny app

- Each change of `input$x` (in the user interface) induces `output$y` to be re-evaluated
  - The render* functions family will re-execute the code as soon as one of the inputs it depends on changes
  - In the initial example, when `input$bins` changes, the plot function is re-computed

```
14
15  output$distPlot <- renderPlot({
16
17    # generate bins based on input$bins from ui.R
18    x    <- faithful[, 2]
19    bins <- seq(min(x), max(x), length.out = input$bins + 1)
20
21    # draw the histogram with the specified number of bins
22    hist(x, breaks = bins, col = 'darkgray', border = 'white')
23
24  })
```

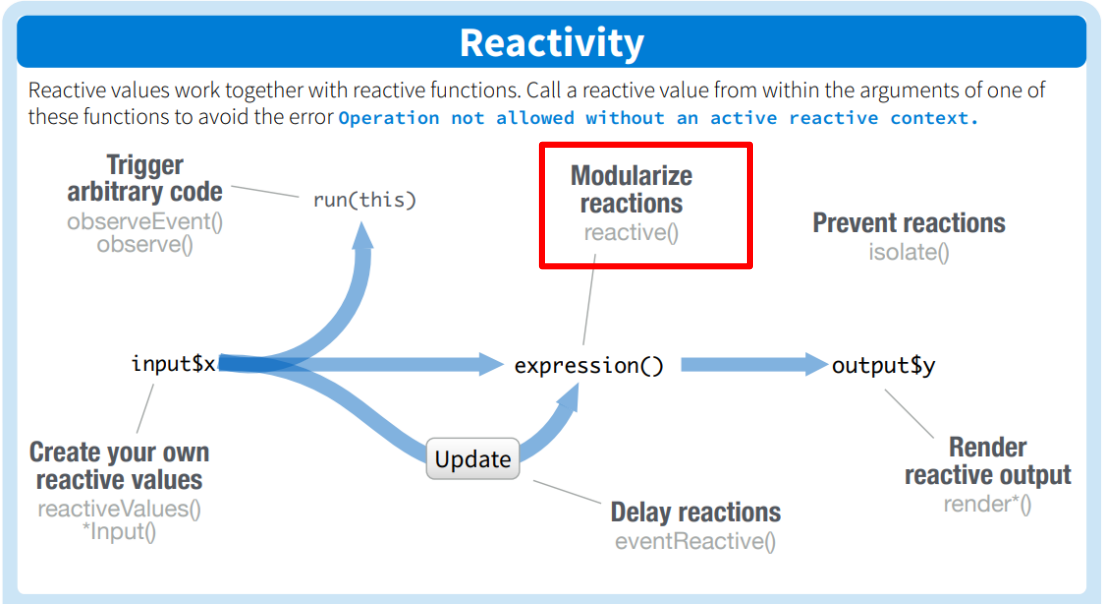


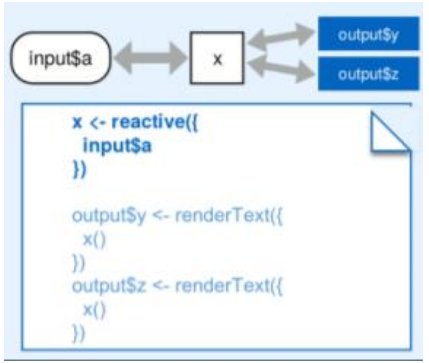
The reactivity graph of Shiny applications



At each change of input$x (in the user interface), output$y is re-evaluated.

Reactivity in Shiny applications

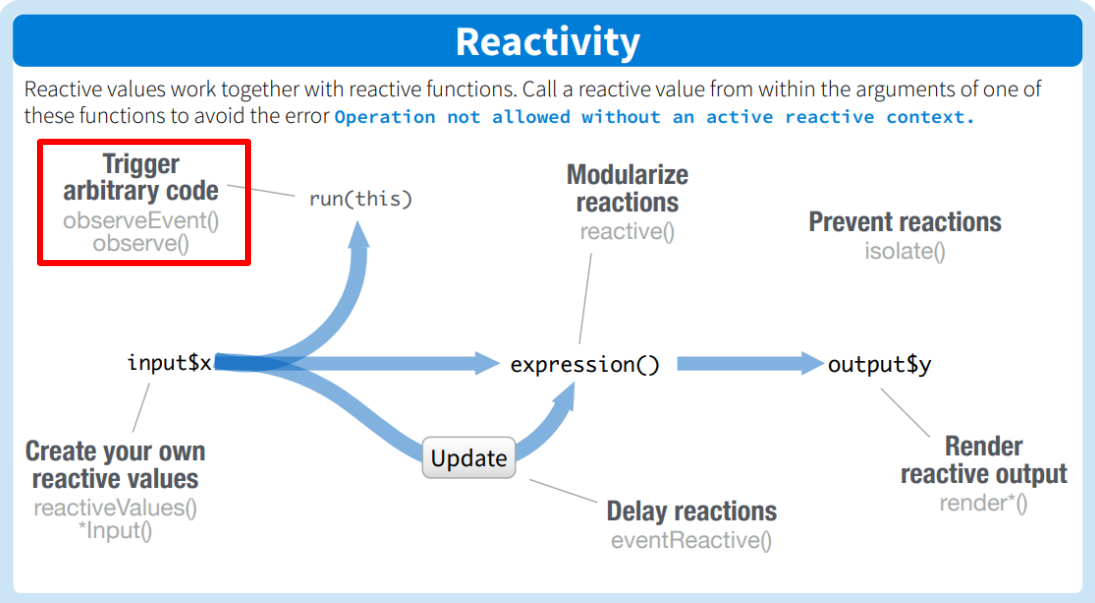





```
ui <- fluidPage(
textInput("a","","A"),
textInput("z","","Z"),
textOutput("b"))

server <- function(input,output){
  re <- reactive({
    paste(input$a,input$z)})
  output$b <- renderText({
    re()
  })
}
shinyApp(ui, server)
```

We can add *reactive* expression() for **caching values**.
Call the expression with function syntax, like `re()`

Reactivity in Shiny applications

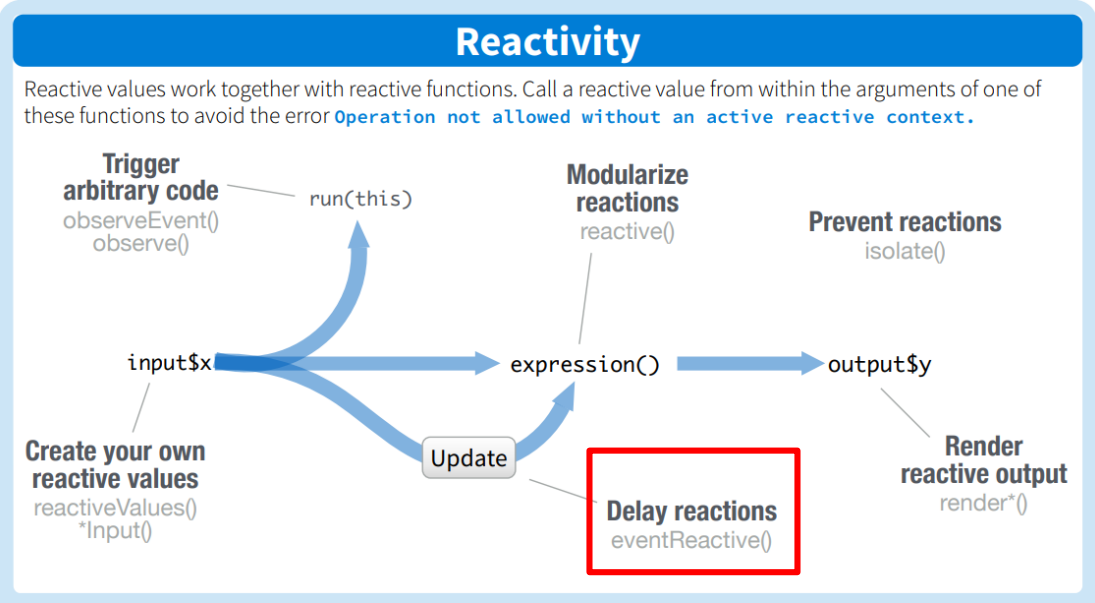


Action `actionButton(inputId, label, icon, ...)`

We run some only after an observe event like **press a button**.

```
library(shiny)
ui <- fluidPage(
  textInput("a","","A"),
  actionButton("go","Go")
)

server <- function(input,output){
  observeEvent(input$go,{
    print(input$a)
  })
}

shinyApp(ui, server)
```

Reactivity in Shiny applications



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

In the same way, you can update an output when we obverse an event.

```
library(shiny)
ui <- fluidPage(
  textInput("a","","A"),
  actionButton("go","Go"),
  textOutput("b")
)

server <- function(input,output){
  re <- eventReactive(
    input$go,{input$a})
  output$b <- renderText({
    re()
  })
}

shinyApp(ui, server)
```

Practical work

→ **TP7**: Add new data

→ **TP8**: Work with randomly generated data

→ **TP9**: Specify the generating distribution

→ **TP10**: Upload your data

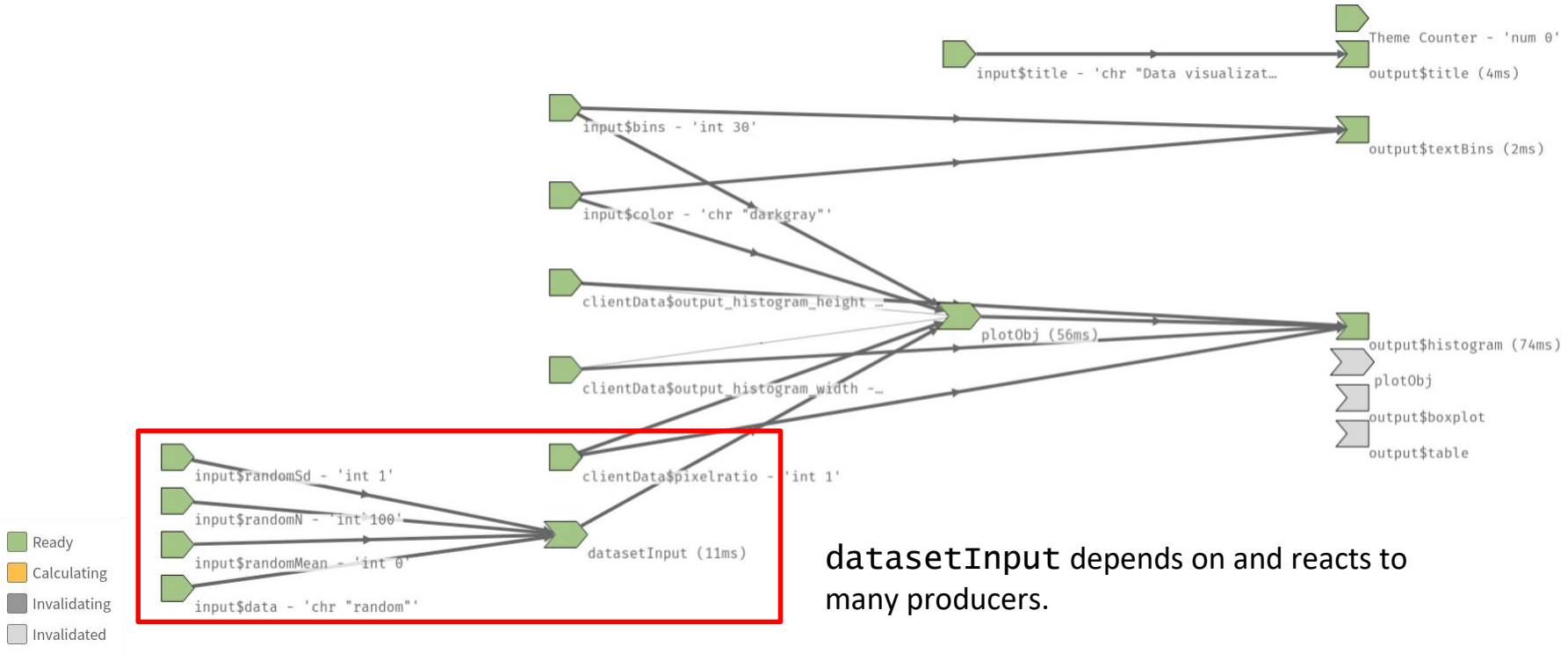
→ …

Exploring reactivity with reactlog

```
> reactlog::reactlog_enable()
> runApp('app9.R')
> shiny::reactlogShow()
```

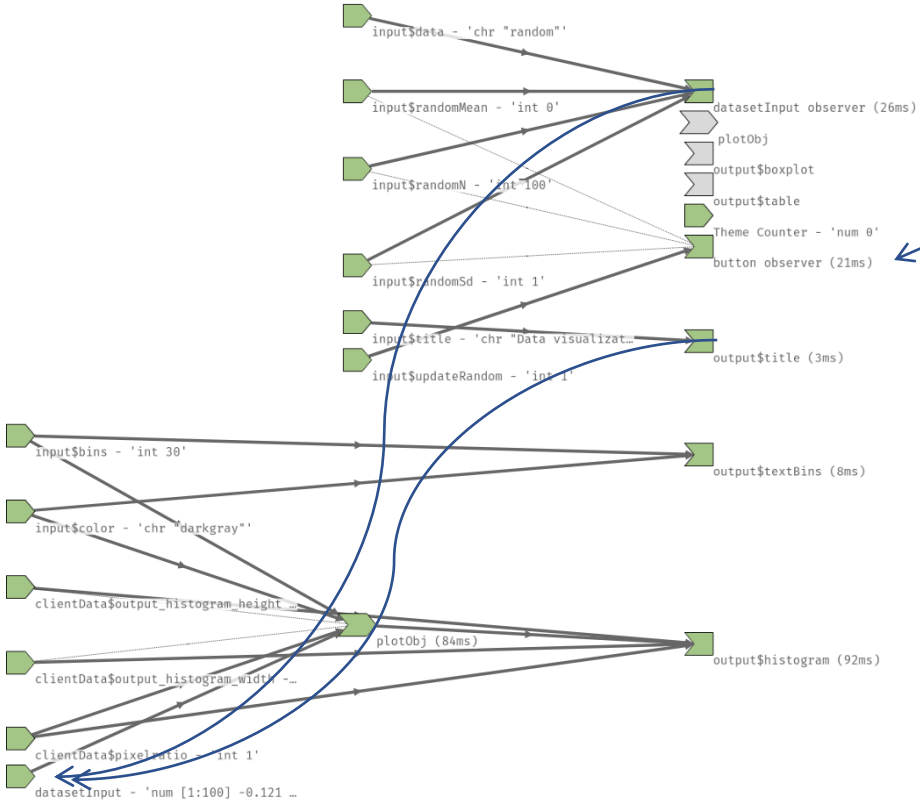
Allows you to play the app "step-by-step" and examine what's going on under the hood.

Exploring reactivity: app9



`datasetInput` depends on and reacts to many producers.

App9_bis: ugly dependency graph

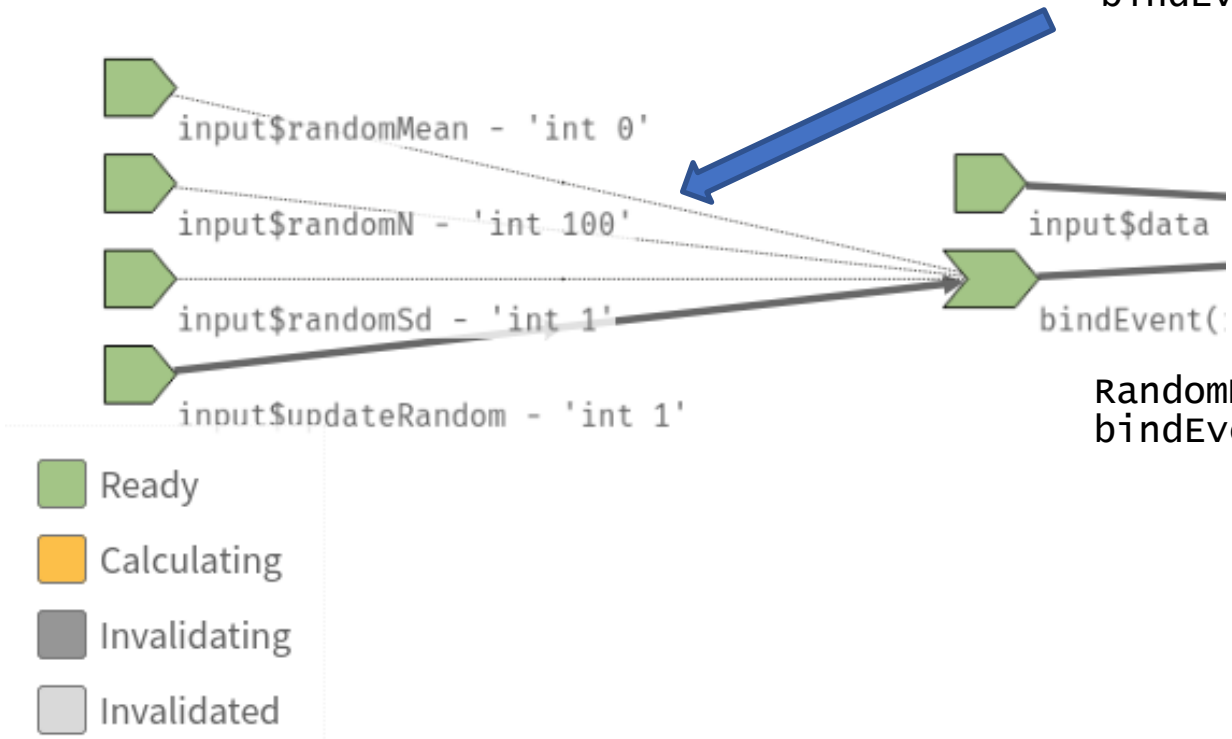


Both observers update `datasetInput()`

Note: observers **break** the natural dependency / reactivity graph

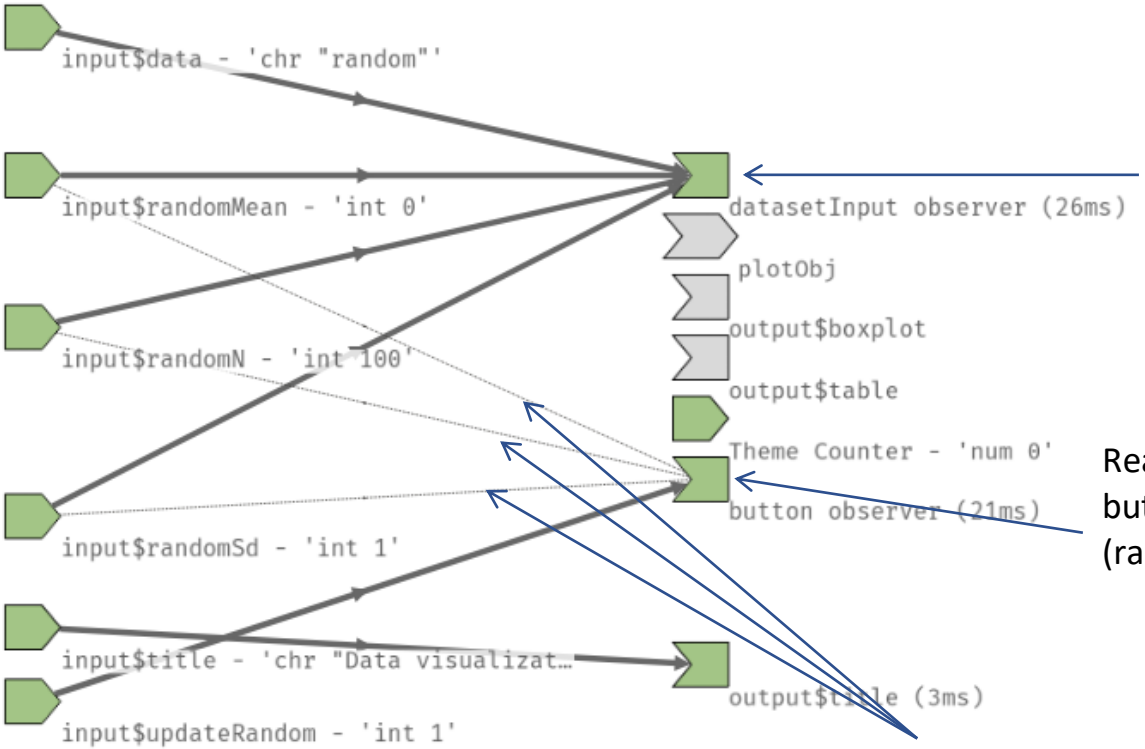
Dependency without reactivity

Reactivity broken by `bindEvent()`



`RandomData()` updated in `bindEvent()`

Reactivity versus dependencies

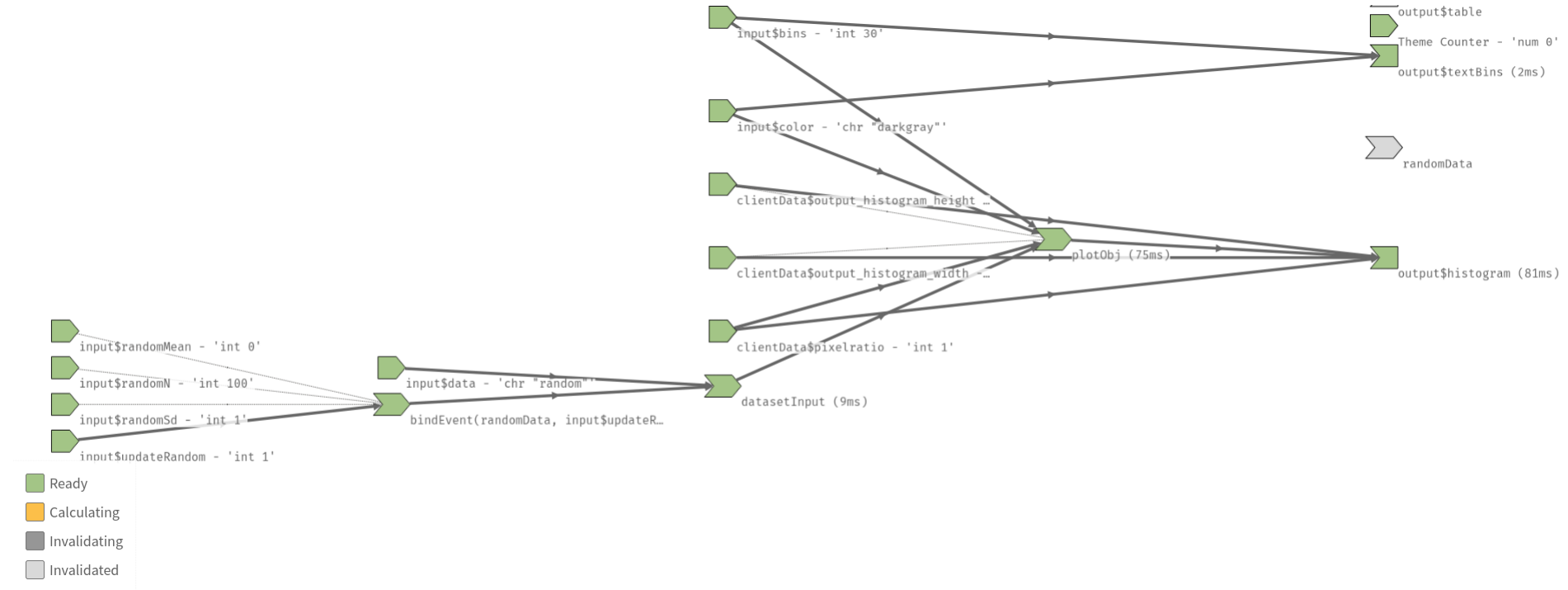


Reacts to (and depends on) any parameter change

Reacts only to clicks on update button but depends on more parameters (randomN, randomMean, randomSd).

Dependency without reactivity,thanks to bindEvent()

App9_ter: straighforward dependencies



Share a Shiny app

Share a Shiny app

Local deployment

- Sharing the code with the user
  - GitHub repository (`shiny::runGitHub` and `shiny::runURL`)
  - R package (build with golem) that the user will install locally

- Disadvantage: requires to install R and manage the packages installation

Share a Shiny app

shinyapps.io

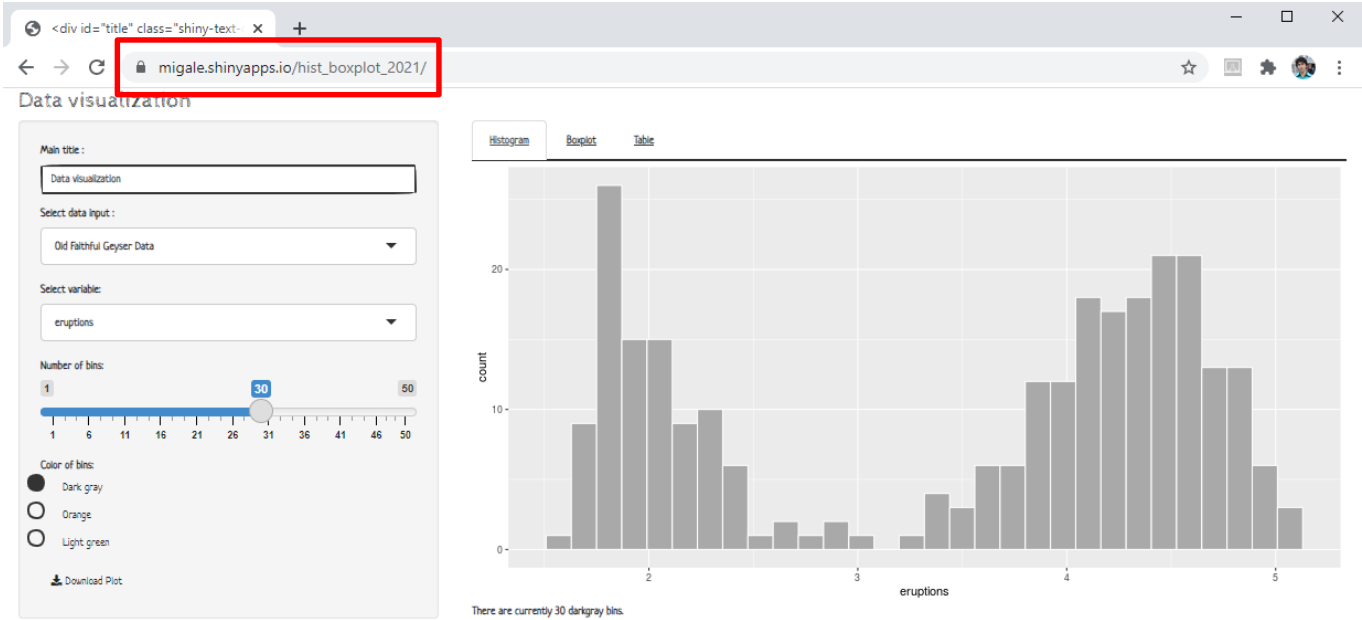
- App is hosted on RStudio servers
  - *Free formula* : 5 Shiny app, 25 actives hours/month
  - *Scalable formula* : since $9/month to $300/month
- 1-click deployment
- User-friendly tracking dashboard
- Disadvantage: *Freemium* and RStudio dependency

➔ https://docs.rstudio.com/shinyapps.io/

Share a Shiny app

shinyapps.io

→ **TP14**: Deploy an app with shinyapps.io



Building with {golem}

- {golem} is a package to ease the development and deployment of shiny app (like {usethis})
    - Description, automatic dependencies, unit test, inline documentation, etc
- Install with `install.packages("golem")`
- Create a shiny app as a package with golem
    - File > New Project > Package for Shiny App Using golem
    - Fill in and execute `dev/01_start.R`
    - Add modules `dev/02_dev.R` and populate ui / server
    - Launch app using `dev/run_dev.R`
    - Deploy using `dev/03_deploy.R`

Host the app on Migale

https://shiny.migale.inrae.fr/

- If your application is stable and packaged, we can host it
  - Scientific field of interest (life sciences)
  - Packaged application (like golem)
  - Using docker (we can help you)
  - GitHub / GitLab versioning
  - No huge computations

Take Home Message

- Shiny allows you to make an R script interactive, easily
- Shiny is the ideal solution for sharing and exploring results
- Several deployment solutions exist and work well, depending what is needed
- Need to optimize your application because latency and memory problems can quickly occur

Sources

- References :
  - **Mastering Shiny : https://mastering-shiny.org/**
  - Gallery: https://shiny.rstudio.com/gallery/
  - Gallery widget: https://shiny.rstudio.com/gallery/widget-gallery.html
  - Show me Shiny :http://www.showmeshiny.com/
  - CheatSheet : https://shiny.rstudio.com/images/shiny-cheatsheet.pdf
  - CheatSheet VF : https://thinkr.fr/pdf/shiny-french-cheatsheet.pdf

- Tuto :
  - Official : https://shiny.rstudio.com/tutorial/
  - Introduction à Shiny (VF) :http://perso.ens-lyon.fr/lise.vaudor/Tuto_Shiny/
  - Formation IFB (VF) : https://github.com/IFB-ElixirFr/IFB_Shiny_training
  - UI organisation : https://speakerdeck.com/jcheng5/styling-shiny

- Blog :
  - **https://blog.rstudio.com/categories/shiny/**
  - https://bioinfo-fr.net/rendre-ses-projets-r-plus-accessibles-grace-a-shiny
  - https://superstatisticienne.fr/r-sur-le-web-le-package-shiny/
  - https://thinkr.fr/a-decouverte-de-shiny/