

# R shiny: building interactive graphical applications

---

Ghislain Durif and Jean-Michel Marin

February 19, 2024

Laboratory of Biology and Modeling of the Cell (LBMC), ENS Lyon, France and CNRS – Institut Montpellierain Alexander Grothendieck (IMAG), University of Montpellier

# Introduction

---

# Resources

- Official shiny website <https://shiny.posit.co/>
- App example gallery <https://shiny.posit.co/r/gallery/>
- Articles <https://shiny.posit.co/r/articles/>
- Video and written tutorials  
<https://shiny.posit.co/r/getstarted/shiny-basics/lesson1/index.html>
- Mastering shiny by Hadley Wickham  
<https://mastering-shiny.org/>

How a software interact with its users?

- command line interface (CLI)
- graphical user interface (GUI)

# Command line tools

- Shell/Terminal command line interface (e.g. bash)

```
user@host $ ls
file.raw hello_world.R README.md shiny_training.Rproj slides
user@host $ Rscript hello_world.R
[1] "doing something"
|+++++| 100% elapsed=01s
```

- R console

```
> library(pbapply)
> print("HelloWorld")
[1] "HelloWorld"
> res <- pblapply(1:1000, function(i) sum(i * seq(1,1E5)))
|+++++| 100% elapsed=01s
```

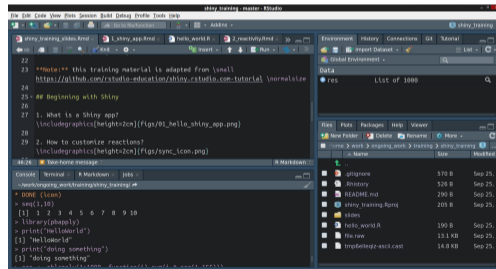
- Python console, etc.

# Graphical user interface

- Graphical display and visual effects/interactions (e.g. buttons to click)

Examples:

- RStudio = GUI to edit and run R code
- Spyder = GUI to edit and run Python code
- OS graphical environment ("super GUI")
- Web browser
- ...



# shiny?

A tool to develop applications<sup>1</sup> with a graphical user interface in R

- Design the **graphical interface** (display and interactions)
- Manage the **reactions to user input** and process data



---

<sup>1</sup>≈softwares

# Client/server model

## shiny app = web application

- a **client** (*frontend*) = a web browser managing the graphical rendering and interactions with the user
- a **server**<sup>2</sup> (*backend*) to process user input or data, and produce output (e.g. run R codes)

---

<sup>2</sup>local or remote

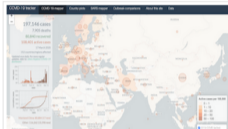


shiny

---

# shiny user showcase gallery

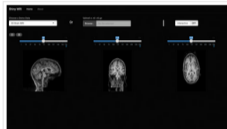
## Life sciences



COVID-19 tracker



Exploring large hospital data for better use of antimicrobials



ShinyMRI - View MRI images in Shiny



Nutrition Calculator - calculate nutrition for recipes



A/B Testing Sample Size Calculator



ctmweb, a web app to analysis Animal tracking data



Visualizing Biodiversity in National Parks data



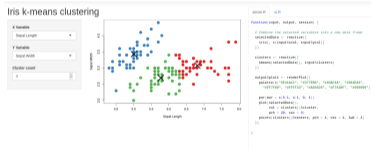
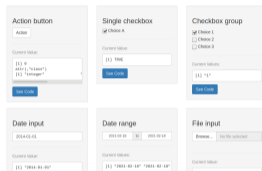
ExPanD: Explore Your Data Interactively

<https://shiny.posit.co/r/gallery/>

# shiny demo gallery

## Examples including code

- widget<sup>3</sup> use cases
- simple app
- data visualization app



<https://shiny.posit.co/r/gallery/>

---

<sup>3</sup>"window gadget", GUI components generally able to interact with users

# shiny app template

```
library(shiny)

ui <- fluidPage()

server <- function(input, output) {}

shinyApp(ui = ui, server = server)

## Listening on http://127.0.0.1:5138
```

# Frontend: user interface (UI) management

## Under the hood:

- appearance: HTML<sup>4</sup>  and CSS<sup>5</sup> 
- reactivity: javascript<sup>6</sup> 

---

<sup>4</sup>standard markup language for web document design

<sup>5</sup>style sheet language used to describe the presentation of a document written in a markup language

<sup>6</sup>scripting language managed client-side by web browsers

# UI design in practice with shiny

Forget about HTML/CSS/JS? (not true for advanced customization)

Intuitive UI design with **R wrapper functions** to manage:

- **graphical components** and **layout organisation** (wrapping HTML) and style management (wrapping CSS)
- **reactivity** to user input (wrapping JS)

# UI design in shiny: an example

```
ui <- fluidPage(  
  headerPanel('Iris k-means clustering'),  
  sidebarPanel(  
    selectInput('xcol', 'X Variable', names(iris)),  
    selectInput('ycol', 'Y Variable', names(iris),  
      selected = names(iris)[2]),  
    numericInput('clusters', 'Cluster count',  
      3, min = 1, max = 9)),  
  mainPanel(plotOutput('plot1'))  
)
```

## Hidden step (HTML conversion)

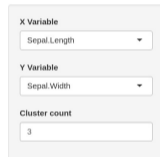
```
<div class="container-fluid">  
  <div class="col-sm-12">  
    <h1>Iris k-means clustering</h1>  
  </div>  
  <div class="col-sm-4">  
    <form class="well" role="complementary">  
      <div class="form-group shiny-input-container">  
        <label class="control-label" id="xcol-label" for="xcol">X Variable</label>  
        <div>  
          <select id="xcol"><option value="Sepal.Length" selected="selected">Sepal.Length</option>  
<option value="Sepal.Width">Sepal.Width</option>  
<option value="Petal.Length">Petal.Length</option>  
<option value="Petal.Width">Petal.Width</option>  
<option value="Species">Species</option></select>  
          <script type="application/json" data-for="xcol" data-nonempty="">{"plugins":{"selectize-plugin-ally"}}</script>  
        </div>  
      </div>  
      <div class="form-group shiny-input-container">  
        <label class="control-label" id="ycol-label" for="ycol">Y Variable</label>  
        <div>  
          <select id="ycol"><option value="Sepal.Length">Sepal.Length</option>  
<option value="Sepal.Width" selected="selected">Sepal.Width</option>  
<option value="Petal.Length">Petal.Length</option>  
<option value="Petal.Width">Petal.Width</option>  
<option value="Species">Species</option></select>  
          <script type="application/json" data-for="ycol" data-nonempty="">{"plugins":{"selectize-plugin-ally"}}</script>  
        </div>  
      </div>  
      <div class="form-group shiny-input-container">  
        <label class="control-label" id="clusters-label" for="clusters">Cluster count</label>  
        <input id="clusters" type="number" class="form-control" value="3" min="1" max="9"/>  
      </div>  
    </form>  
  </div>  
  <div class="col-sm-8" role="main">  
    <div id="plot1" class="shiny-plot-output" style="width:100%;height:400px;"></div>  
  </div>  
</div>
```

# UI design in shiny: an example

```
ui <- fluidPage(  
  headerPanel('Iris k-means clustering'),  
  sidebarPanel(  
    selectInput('xcol', 'X Variable', names(iris)),  
    selectInput('ycol', 'Y Variable', names(iris),  
      selected = names(iris)[2]),  
    numericInput('clusters', 'Cluster count',  
      3, min = 1, max = 9)),  
  mainPanel(plotOutput('plot1'))  
)
```

Display

Iris k-means clustering



X Variable  
Sepal.Length

Y Variable  
Sepal.Width

Cluster count  
3

Layout only (before server processing)



# UI design in shiny: an example

```
ui <- fluidPage(  
  headerPanel('Iris k-means clustering'),  
  sidebarPanel(  
    selectInput('xcol', 'X Variable', names(iris)),  
    selectInput('ycol', 'Y Variable', names(iris),  
      selected = names(iris)[2]),  
    numericInput('clusters', 'Cluster count',  
      3, min = 1, max = 9)),  
  mainPanel(plotOutput('plot1'))  
)
```

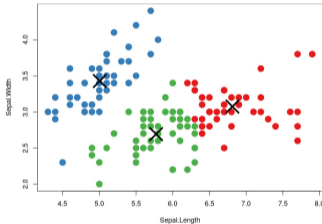
Display

Iris k-means clustering

X Variable  
Sepal.Length

Y Variable  
Sepal.Width

Cluster count  
3



Layout + reactivity (with server processing)

# UI elements

HTML-wrapping elements:

- all standard **tags** (headers, hyperlink, etc.)
- pre-packaged **layouts** (grid with rows and columns, panels, tabs, etc.)
- **widgets** for **user input** (sliders, numeric input, text input, etc.)
- **output display** elements (to render display/visualization of data/result)

Possible to add CSS styling (with optional arguments, e.g. `style =`

# Backend (server-side): management of input and events

**Reactivity** = reaction to user input or to events

- Data/information stored in **reactive values**
  - information provided by **user input**
  - intermediate or final **processing results**
- **Modification** of a reactive value triggers a server-side **chain reaction**
- Web server implementation managed by shiny

# User input

## UI-side

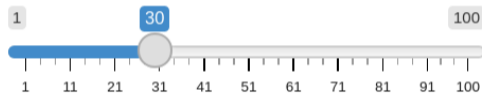
```
ui <- fluidPage(  
  sliderInput(inputId = "num", label = "Choose a number",  
    value = 25, min = 1, max = 100)  
)
```

## Server-side

```
server <- function(input, output){  
  observe(print(input$num))  
}
```

## Display

### Choose a number



## Input processing done server-side

R console (server-side)

```
## Listening on http://127.0.0.1:5138
```

```
## [1] 25
```

```
## [1] 30
```

# Output rendering

## UI-side

```
ui <- fluidPage(  
  textInput(inputId = "input_txt", label = "Write me"),  
  verbatimTextOutput(outputId = "output_txt", placeholder = TRUE)  
)
```

## Server-side

```
server <- function(input, output){  
  output$output_txt <- renderText(input$input_txt)  
}
```

## Display

Write me

shiny

Input processing done server-side

# Events

## UI-side

```
ui <- fluidPage(  
  actionButton(inputId = "click", label = "Click me")  
)
```

## Server-side

```
server <- function(input, output){  
  observeEvent(input$click, print(as.numeric(input$click)))  
}
```

## Display



## Input processing done server-side

R console (server-side)

```
## Listening on http://127.0.0.1:5138  
## [1] 1  
## [1] 2
```

# Reactivity

Server-side: **reactive values** including all UI inputs and local data

1. **Modification** of reactive value(s): input given by user in UI, or local data modified by server (in a previous reaction chain)
2. **Invalidation** of all events and outputs depending on the modified reactive value(s)
3. **Processing** code chunks corresponding to all invalidated events (data processing) and outputs (graphical rendering)

# Complete shiny app

- UI-side = combination of layouts, HTML-wrapped elements, widgets, UI input and output elements
- server-side = R codes orchestrating input/data processing and output rendering



# Create interface modules

- Complete implementation of complex UI elements and corresponding **server-side logic**
- Modules are reusable “*autonomous*” units in a shiny app

Tutorials:

- Modularizing shiny app code (<https://shiny.posit.co/r/articles/improve/modules/>)
- Communication between modules  
(<https://shiny.posit.co/r/articles/improve/communicate-bet-modules/>)

# Additional shiny features

- shinyFiles: <https://github.com/thomasp85/shinyFiles> (manage files/directory)
- shinyWidgets: <https://github.com/dreamRs/shinyWidgets> (additional components)
- shinybusy: <https://github.com/dreamRs/shinybusy> (busy indicator)
- shinydashboard: <https://rstudio.github.io/shinydashboard/> (dashboard interface)
- shinyjs: <https://github.com/daattali/shinyjs> (javascript-based reactivity)

# Releasing and sharing your shiny app

- Publish the R code for people to run on their machine/server
- Host the app on a shiny server (yours<sup>7</sup> or <https://www.shinyapps.io/>)
- Develop and release your shiny app as an R package

---

<sup>7</sup><https://docs.posit.co/shiny-server/>

# Limits

- Out-of-the-box style is nice but recognizable
- UI advanced customization requires knowledge of HTML/CSS/JS
- All server-side processing (computations) done in R, potential performance limitation (may be overcome by language interfacing, c.f. later)

## Examples of ML related apps

- <https://github.com/davesteps/machLearn> (local app) or <https://davesteps.shinyapps.io/machLearn/> (remote app)
- <https://github.com/RamiKrispin/MLstudio> (packaged app)

## Some alternatives

---

# Shiny for python now available

<https://shiny.posit.co/py/>

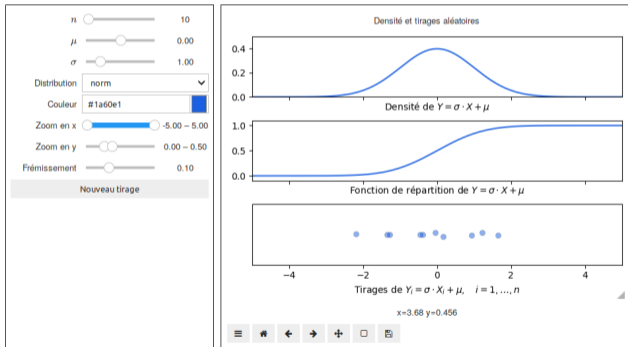
Shiny express: A simpler way to write and learn Shiny.

<https://shiny.posit.co/blog/posts/shiny-express/>

# Python ipywidget

<https://ipywidgets.readthedocs.io/>

- Widgets in Jupyter notebooks
- Interactive notebook
- Development of complete graphical application?



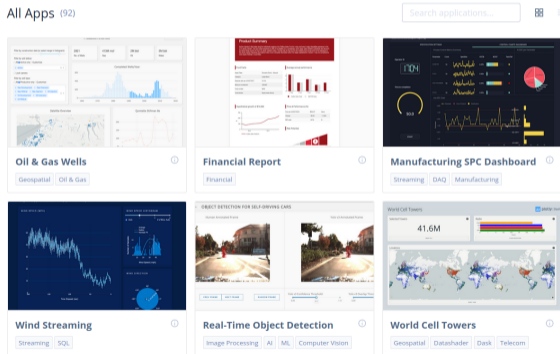
Example: <https://github.com/josephsalmon/Random-Widgets>



# Python Dash

<https://dash.plotly.com>

- Client/server logic
- Design display and manage reactivity
- Less intuitive server-side implementation?



*Dash gallery:* <https://dash-gallery.plotly.host/Portal/>

# reticulate R package

`https://rstudio.github.io/reticulate/`

- Call Python code directly from R (e.g. in your shiny app)
- Direct import of Python packages
- Support Python virtual environments or Conda environments

# reticulate R package

```
# setup
library(reticulate)
use_python("~/anaconda3/bin/python")
use_condaenv(condaenv = "base", conda = "~/anaconda3/bin/conda")

# import
skl_lr <- import("sklearn.linear_model")

# data
x <- as.matrix(rnorm(100, sd = 2))
y <- 2 * x + as.matrix(rnorm(100))

# model
lr <- skl_lr$LinearRegression()

# training
lr$fit(r_to_py(x), r_to_py(y))
```

```
## LinearRegression()
```

```
lr$coef_
```

```
##           [,1]
## [1,] 1.977388
```

# Rcpp R package

<http://rcpp.org/>

- Seamless interfacing of C++ code in R
- Binder automatic generation
- C++ code compilation on the fly or smooth integration in R package installation
- Easy integration of header C++ libraries (RcppEigen for Eigen<sup>8</sup>, BH for Boost<sup>9</sup>)

---

<sup>8</sup>linear algebra library

<sup>9</sup>collection of C++ libraries, including maths libraries, etc.

# Rcpp R package

In my\_file.cpp

```
#include <Rcpp.h>
using namespace Rcpp;
// [[Rcpp::export]]
NumericVector timesTwo(NumericVector x) {return x *2;}
```

In R:

```
sourceCpp("my_file.cpp")

x <- rnorm(100)

y <- timesTwo(x)
```

To conclude

---

# Take-home message

R shiny: develop graphical application as web app

Client-side (frontend)

- Simple **out-of-the-box** webdesign with user interaction
- Possible **customization** (HTML, CSS, JavaScript)

Server-side (backend)

- **Reactivity** to user input
- User input and data **processing**