R shiny: building interactive graphical applications - Slides totaly inspired from those of Ghislain Durif ; `https://plmlab.math.cnrs.fr/gdurif/shiny-training`

R Programming - HAX815X

Jean-Michel Marin

February 2026

Faculty of Sciences, University of Montpellier

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/`

User interface

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
```

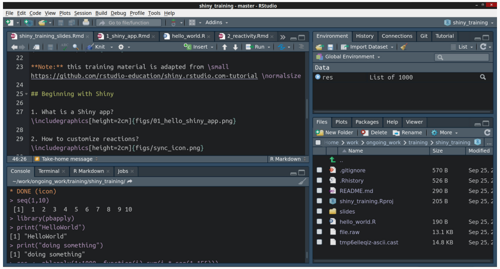
Python console, etc.

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
- ...



```
**Note:** this training material is adapted from \small
https://github.com/rstudio-education/shiny.rstudio.com-tutorial \normalsize

## Beginning with Shiny

1. What is a Shiny app?
\includegraphics[height=2cm]{figs/01_hello_shiny_app.png}

2. How to customize reactions?
\includegraphics[height=2cm]{figs/sync_icon.png}
```

```
~/work/ongoing_work/training/shiny_training/
* DONE (icon)
> seq(1,10)
 [1]  1  2  3  4  5  6  7  8  9 10
> library(pbapply)
> print("HelloWorld")
[1] "HelloWorld"
> print("doing something")
[1] "doing something"
```

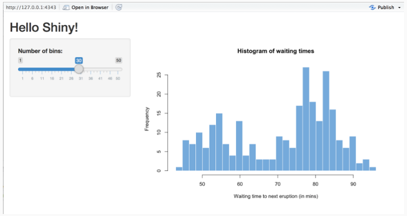
```
Data
res            List of 1000
```

Name	Size	Modified
..		
.gitignore	570 B	Sep 25, 2
.Rhistory	526 B	Sep 25, 2
README.md	290 B	Sep 25, 2
shiny_training.Rproj	205 B	Sep 25, 2
slides		
hello_world.R	190 B	Sep 25, 2
file.raw	13.1 KB	Sep 25, 2
tmp6elleqiz-ascii.cast	14.8 KB	Sep 25, 2

shiny?

A tool to develop applications[1] with a graphical user interface in R

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



[1]≈softwares

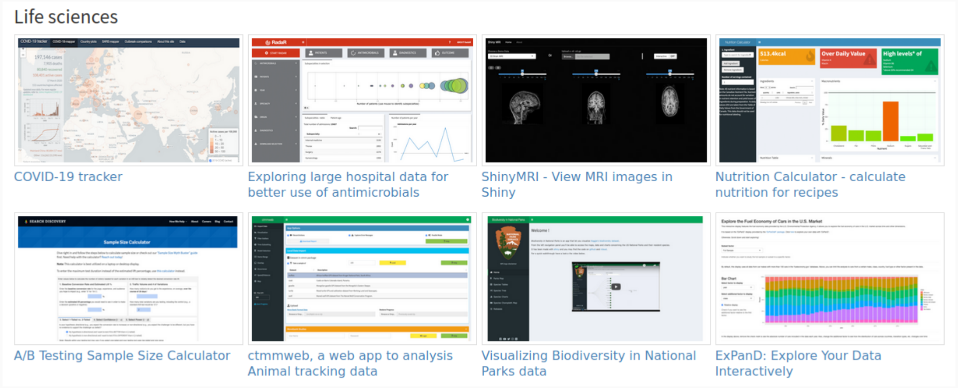
Client/server model

shiny app = web application

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

[2]local or remote

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

ctmmweb, 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

- simple app
- widget[3] use cases
- data visualization app

[3]"*window gadget*", GUI components generally able to interact with users

shiny app template

```r
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

- **appearence**: HTML[4] and CSS[5]
- **reactivity**: javascript[6]

[4] standard markup language for web document design

[5] style sheet language used to describe the presentation of a document written in a markup language

[6] 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

```r
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'))
)
```

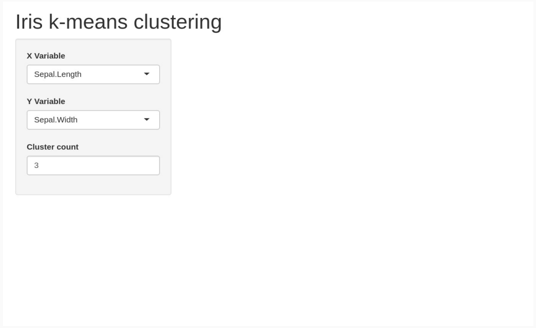
Hidden step (HTML conversion)

```html
<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>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-a11y"]}</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>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-a11y"]}</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

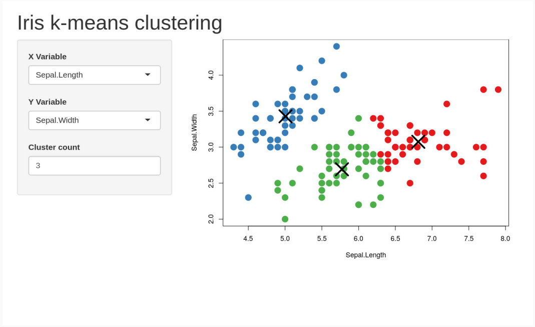
Cluster count: 3

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 = ”color:red;”`)

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

Reactivity = reaction to user input or to events

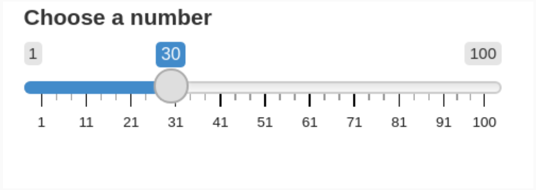
- Data/information stored in reactive values
  - information provided by **user input**
  - intermediate or final **processing results**
- Modification of a reactive value triggers a server-side chain reaction
- Web server implementation managed by shiny

User input

Display

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))
}
```

Input processing done server-side

R console (server-side)

```
## Listening on http://127.0.0.1:5138
## [1] 25
## [1] 30
```

Output rendering

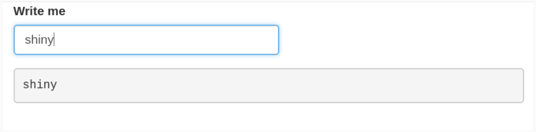
UI-side

```
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



```
shiny
```

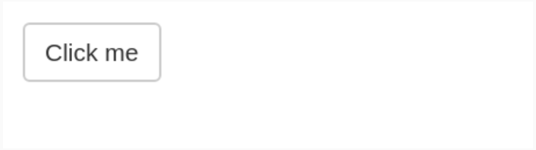
Input processing done server-side

Events

Display

UI-side

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



Server-side

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

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

- Modification of reactive value(s): input given by user in UI, or local data modified by server (in a previous reaction chain)
- Invalidation of all events and outputs depending on the modified reactive value(s)
- 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`
- shinyWidgets `https://github.com/dreamRs/shinyWidgets`
- shinybusy `https://github.com/dreamRs/shinybusy`
- shinydashboard `https://rstudio.github.io/shinydashboard/`
- shinyjs `https://github.com/daattali/shinyjs`

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[7] or `https://www.shinyapps.io/`)
- Develop and release your shiny app as an R package

[7] `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)

Shiny for python now available

`https://shiny.posit.co/py/`

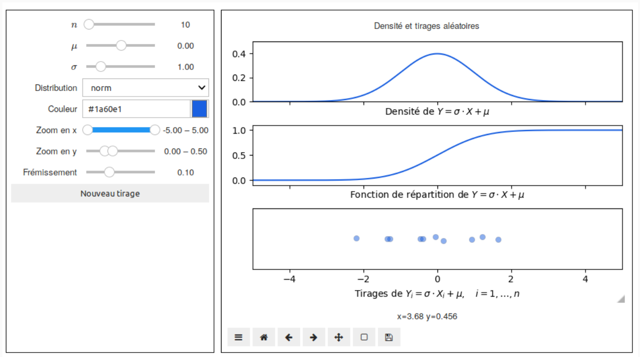
Shiny express: A simpler way to write and learn Shiny

`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?



n 10
μ 0.00
σ 1.00
Distribution norm
Couleur #1a60e1
Zoom en x -5.00 – 5.00
Zoom en y 0.00 – 0.50
Frémissement 0.10

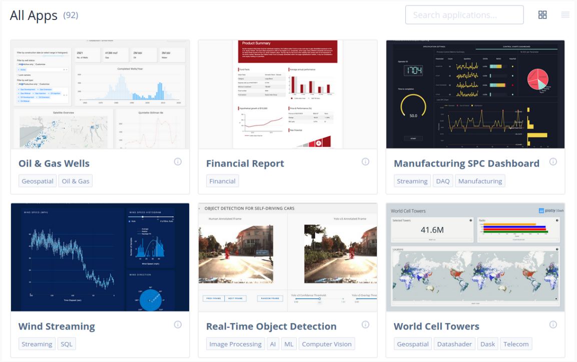
Nouveau tirage
Densité et tirages aléatoires
0.4
0.2
0.0
Densité de $Y = \sigma \cdot X + \mu$
1.0
0.5
0.0
Fonction de répartition de $Y = \sigma \cdot X + \mu$
−4
−2
0
2
4
Tirages de $Y_i = \sigma \cdot X_i + \mu, \quad i = 1, \ldots, n$
x=3.68 y=0.456

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?


All Apps (92)

Oil & Gas Wells — Geospatial, Oil & Gas

Financial Report — Financial

Manufacturing SPC Dashboard — Streaming, DAQ, Manufacturing

Wind Streaming — Streaming, SQL

Real-Time Object Detection — Image Processing, AI, ML, Computer Vision

World Cell Towers — Geospatial, Datashader, Dask, Telecom

Dash gallery: `https://dash-gallery.plotly.host/Portal/`

reticulate R package

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

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

reticulate R package

```r
library(reticulate)
use_python("/Users/marin/miniforge3/bin/python")
use_condaenv(condaenv = "base", conda = "/Users/marin/miniforge3/bin/conda")
skl_lr <- import("sklearn.linear_model")
x <- as.matrix(rnorm(100, sd = 2))
y <- 2 * x + as.matrix(rnorm(100))
lr <- skl_lr$LinearRegression()
lr$fit(r_to_py(x), r_to_py(y))
```

```
## LinearRegression()
```

```r
lr$coef_
```

```
##          [,1]
## [1,] 2.01037
```

Rcpp R package

- 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`[8], `BH` for `Boost`[9])

`http://rcpp.org/`

[8] linear algebra library

[9] collection of C++ libraries, including maths libraries, etc.

Rcpp R package

In `my_file.cpp`

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

In R

```
sourceCpp(”my_file.cpp”)
x <- rnorm(100)
y <- timesTwo(x)
```

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