Data Visualization with R
Shiny tutorial

Ariane Ducellier

University of Washington - Fall 2023

What is Shiny?

Examples

Let us run several examples:

```
library(shiny)
```

```
runExample("08_html")
```

```
runExample("01_hello")
```

Examples

UI part:

```
ui <- fluidPage(
  titlePanel(...),
  sidebarLayout(
    sidebarPanel(
      sliderInput(
        ...
      )
    ),
    mainPanel(
      plotOutput(outputId="distplot")
    )
  )
)
```

Server part:

```
server <- function(input, output) {
  output$distplot <- renderPlot({
    ...
  })
}
```

Creation of Shiny app:

```
shinyApp(ui=ui, server=server)
```

R Markdown with interactive Shiny elements

```
Go to File >
      New File >
      R Markdown >
      Shiny
```

Fill the document with the code from `tutorial_shiny_1.Rmd`.

Click on `Run Document`.

Minimal example

We need the files ui.R and server.R that are kept within the same folder. ui.R describe the user interface.

```
fluidPage(...,
    title = NULL, theme = NULL, lang = NULL)
```

indicates that we are going to use a fluid page layout with rows containing columns.

```
titlePanel(title, windowTitle = title)
```

describes the title of the application.

Minimal example

```
sidebarLayout(sidebarPanel,
              mainPanel,
              position = c("left", "right"),
              fluid = TRUE)
```

describe the general layout of the page, with:

- Inputs on the side (sidebarPanel),
- Outputs in the middle (mainPanel).

Minimal example

The panels contain input and output widgets:

```
textInput(inputId = "comment",
          label,
          value = "",
          width = NULL,
          placeholder = NULL)
```

```
textOutput(outputId = "textDisplay",
           container = if (inline) span else div,
           inline = FALSE)
```

server.R contains functions which use inputid as an input, and
produce outputId as an output.

Minimal example

server.R contains a function describing how to use the input
from ui.R to produce the outputs from ui.R.

```
function(input, output){
  output$textDisplay = renderText({...input$comment...
  })
 )
```

The function(input, output) contains the reactive
components of the application. For example:

```
renderText(expr,
           env = parent.frame(),
           quoted = FALSE,
           func = NULL)
```

Run the minimal example

Set the working directory to the folder that contains `ui.R` and `server.R`,

```
setwd("/Users/my_name/Documents/my_folder/")
```

load the Shiny package:

```
library(shiny)
```

and run the application:

```
runApp()
```

Various widgets

```
checkboxGroupInput(inputId, label, choices=NULL, ...)
```

```
checkboxInput(inputId, label, value=FALSE, ...)
```

```
dateInput(inputId, label, ...)
```

```
dateRangeInput(inputId, label, ...)
```

```
numericInput(inputId, label, value, ...)
```

```
radioButtons(inputId, label, choices=NULL, ...)
```

Various widgets

```
selectInput(inputId, label, choices, ...)
```

```
sliderInput(inputId, label, min, max, value, ...)
```

```
textInput(inputId, label, ...)
```

To see an example of how the widgets look like, type:

```
library(shiny)
runGist(6571951)
```

Panels

We can show multiple frames in screen and let the user select one.
The processing of the data is only carried out for the currently
selected tab.

```
tabsetPanel(
  tabPanel("title_text", textOutput("name_text")),
  tabPanel("title_plot", plotoutput("name_plot")),
  tabPanel("title_map", leafletOutput("name_map"))
)
```

The leaflet package allows us to produce maps shown with
leafletOutput in the ui.R and created with renderLeaflet in
the server.R file.

Reactive objects

In the server.r file, we filter the data using a reactive object:

```
theData = reactive({
  mapData %>%
    filter(year >= input$year
})
```

- A reactive object changes when its input changes.
- When it runs, the output is cached.
- If it is called several times in an application, it will not run again if the inputs are unchanged.

Simple layouts

Left-to-right and top-to-bottom. The elements reorder themselves when resizing the window.

```
flowLayout( ... )
```

Top-to-bottom

```
verticalLayout( ... )
```

Left-to-right with manually set widths

```
splitLayout(cellWidths = c("25%", "75%"),
            ... ),
```

Complete layouts

Side bar and main panel

```
fluidpage(
  sidebarLayout(sidebarPanel, mainPanel, position))
)
```

Top level navigation bar and several tabs

```
navbarPage(title, tabPanel)
```

Left navigation bar and several tabs

```
fluidpage(
  navlistPanel(title, tabPanel)
)
```

Complete layouts

Rows and columns. The sum of the widths of the columns must be equal to 12.

```
fluidpage(
  fluidrow(
    column(width=4, ...),
    column(width=4, ...), ... ))
```

Combination of layouts

```
fluidPage(
  fluidRow(
    column(width=4, ...), column(width=8, ...)),
  splitLayout( ... ),
  verticalLayout( ... )
)
```

Hiding elements

Name the panels:

```
tabsetPanel(id = "theTabs",
  tabPanel( ... , value = "trend"),
  ...
)
```

Add a condition to show an UI element only if a tab is selected:

```
conditionalpanel(
  condition = "input.theTabs == trend",
  checkboxInput( ... )
)
```

Tables - Basic Shiny

In `ui.R`:

```
tableOutput("textDisplay")
```

In `server.R`

```
output$textDisplay = renderTable({
  getMat = matrix(c( ... ), ncol = 2, byrow = TRUE)
  colnames(getMat) = c("Value", "Class")
  getMat
})
```

Tables - With package DT (DataTable)

In `ui.R`:

```
dataTableOutput("countryTable")
```

In `server.R`

```
output$countryTable = renderDataTable({
  dataTable(
    ... ,
    colnames = ... ,
    caption = ... ,
    filter = "top",
    options = list(pageLength = 15,
                   lengthMenu = c(10, 20, 50))
})
```

Reactive user interfaces

In `ui.R`:

```
uiOutput("yearSelectorUI")
```

In `server.R`

```
output$yearSelectorUI = renderUI(
  selectedYears = ...
  selectInput( ... , selectedYears)
})
```

When the value in `selectedYears` change, the choice of years in the widget will also change.

Progress bar

If some computation in server.R can take a long time, it is useful to wrap the corresponding code inside the Shiny withProgress() function.

In server.R

```
withProgress(message = ... ,
  detail =  ... , value = 0,
  ... function code ...
  incProgress(1/3)
  ... function code ...
  incProgress(1/3)
  ... function code ...
  incProgress(1/3)
  ... function code ...
})
```