Apoyo Ayudantía Shiny

Natalie Julian

La estructura de una ShinyApp

```
library(shiny)

ui ← fluidPage()

server ← function(input, output) {}

runApp(list(ui = ui, server = server))
```

En `shiny`, una aplicación constará de **2** partes:

- La interfaz de usuario, `ui` (user interface), donde definiremos el look de nuestra aplicación, y lugar de `inputs` y `outputs`.
- El `server`, en donde especificaremos como interactuan los `outputs` en función de los `inputs`.

La estructura de una ShinyApp

```
library(shiny)

ui ← fluidPage()

server ← function(input, output) {}

runApp(list(ui = ui, server = server))
```

- Se define una interfaz de usuario (user interface). En adelante `ui`.
- En este caso es una página fluida vacía `fluidPage()`.

- En el futuro acá definiremos diseño/estructura de nuestra aplicación (*layout*). Que se refiere la disposición de nuestros `inputs` y `outputs`.

La estructura de una ShinyApp

```
library(shiny)

ui ← fluidPage()

server ← function(input, output) {}

runApp(list(ui = ui, server = server))
```

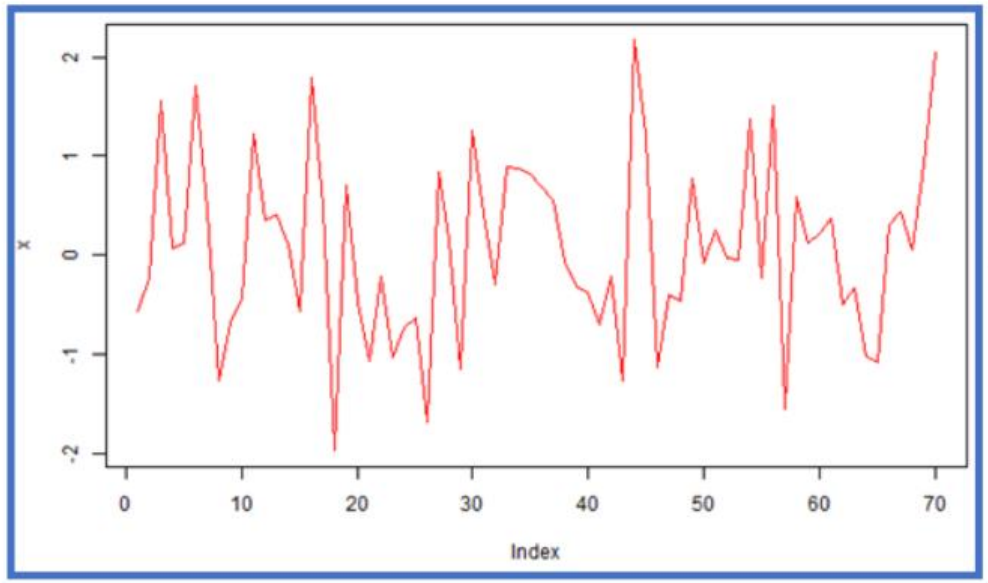
De formar general la aplicación será:

```
library(shiny)
# acá se cargarán paquetes y posiblemente también datos
# necesarios para ui (como definir opciones de inputs)

ui ← fluidPage(
  # código que da forma a nuestrá aplicación: títulos, secciones, textos, inputs
)

server ← function(input, output) {
  # toooda la lógica de como interactuan los outputs en función de los inputs
}
```

Outputs




```r
library(shiny)

ui <- fluidPage(
   sidebarLayout(
      sidebarPanel(
        sliderInput("nrand", "Simulaciones",
                    min = 50, max = 100, value = 70),
        selectInput("col", "Color", c("red", "blue", "black")),
        checkboxInput("punto", "Puntos:", value = FALSE)
      ),
      mainPanel(plotOutput("outplot"))
   )
)

server <- function(input, output) {
   output$outplot <- renderPlot({
     set.seed(123)
     x <- rnorm(input$nrand)
     t <- ifelse(input$punto, "b", "l")
     plot(x, type = t, col = input$col)
   })
}
```

La estructura de una ShinyApp 2

```
ui ← fluidPage(
   sidebarLayout(
      sidebarPanel(
        sliderInput("nrand", "Simulaciones", min = 50, max = 100, value = 70),
        selectInput("col", "Color", c("red", "blue", "black")),
        checkboxInput("punto", "Puntos:", value = FALSE)
      ),
      mainPanel(plotOutput("outplot"))
   )
)

server ← function(input, output) {
   output$outplot ← renderPlot({
     set.seed(123)
     x ← rnorm(input$nrand)
     t ← ifelse(input$punto, "b", "l")
     plot(x, type = t, col = input$col)
   })
}
```

- `fluidPage`, `sidebarLayout`, `sidebarPanel`, `mainPanel` definen el diseño/*layout* de nuestra app.
- Existen muchas más formas de organizar una app: Por ejemplo uso de *tabs* de *menus*, o páginas con navegación. Más detalles http://shiny.rstudio.com/articles/layout-guide.html.

La estructura de una ShinyApp 2

```
ui ← fluidPage(
   sidebarLayout(
      sidebarPanel(
        sliderInput("nrand", "Simulaciones", min = 50, max = 100, value = 70),
        selectInput("col", "Color", c("red", "blue", "black")),
        checkboxInput("punto", "Puntos:", value = FALSE)
      ),
      mainPanel(plotOutput("outplot"))
   )
)

server ← function(input, output) {
   output$outplot ← renderPlot({
     set.seed(123)
     x ← rnorm(input$nrand)
     t ← ifelse(input$punto, "b", "l")
     plot(x, type = t, col = input$col)
   })
}
```

- `sliderInput`, `selectInput`, `checkboxInput` son los inputs de nuestra app, con esto el usuario puede interactuar con nuestra aplicación (https://shiny.rstudio.com/gallery/widget-gallery.html).
- Estas funciones generan el input deseado en la app y shiny perminte que los valores de estos inputs sean usados como valores usuales en R en la parte del server (numericos, strings, booleanos, fechas).

La estructura de una ShinyApp 2

```
ui ← fluidPage(
   sidebarLayout(
      sidebarPanel(
        sliderInput("nrand", "Simulaciones", min = 50, max = 100, value = 70),
        selectInput("col", "Color", c("red", "blue", "black")),
        checkboxInput("punto", "Puntos:", value = FALSE)
      ),
      mainPanel(plotOutput("outplot"))
   )
)

server ← function(input, output) {
   output$outplot ← renderPlot({
     set.seed(123)
     x ← rnorm(input$nrand)
     t ← ifelse(input$punto, "b", "l")
     plot(x, type = t, col = input$col)
   })
}
```

- `renderPlot` define un tipo de salida gráfica.
- Existen otros tipos de salidas, como tablas `tableOutput` o tablas más interactivas como `DT::DTOutput`.

La estructura de una ShinyApp 2

```
ui ← fluidPage(
   sidebarLayout(
      sidebarPanel(
        sliderInput("nrand", "Simulaciones", min = 50, max = 100, value = 70),
        selectInput("col", "Color", c("red", "blue", "black")),
        checkboxInput("punto", "Puntos:", value = FALSE)
      ),
      mainPanel(plotOutput("outplot"))
   )
)

server ← function(input, output) {
   output$outplot ← renderPlot({
     set.seed(123)
     x ← rnorm(input$nrand)
     t ← ifelse(input$punto, "b", "l")
     plot(x, type = t, col = input$col)
   })
}
```

- Cada `*Output()` y `render*()` se asocian con un **id** definido por nosotros
- Este **id** debe ser único en la application
- En el ejemplo `renderPlot` esta asociado con `plotOutput` vía el id `outplot`

La estructura de una ShinyApp 2

```
ui ← fluidPage(
   sidebarLayout(
      sidebarPanel(
        sliderInput("nrand", "Simulaciones", min = 50, max = 100, value = 70),
        selectInput("col", "Color", c("red", "blue", "black")),
        checkboxInput("punto", "Puntos:", value = FALSE)
      ),
      mainPanel(plotOutput("outplot"))
   )
)

server ← function(input, output) {
   output$outplot ← renderPlot({
     set.seed(123)
     x ← rnorm(input$nrand)
     t ← ifelse(input$punto, "b", "l")
     plot(x, type = t, col = input$col)
   })
}
```

- Cada función `*Input` requiere un **id** para ser identificado en el server
- Cada `*Input` requiere argumentos especificos a cada tipo de input, valor por defecto, etiquetas, opciones, rangos, etc
- Acá, el valor númerico ingresado/modifcado por el usuario se puede acceder en el server bajo `input$nrand`

HTMLWidgets

- HTMLWidgets son un tipo de paquetes que nos permiten realizar visualizaciones en HTML las cuales son fácil de integrar con shiny y también rmarkdown.
- Existen una gran cantida de paquetes https://gallery.htmlwidgets.org/
- Son -entonces- paquetes para complementar nuestra aplicación.

Cada paquete HTMLWidget tiene su propio set de funciones, el código utilizado para hacer un gráfico en plotly no es el mismo (pero generalmente muy similar) al utilizado en highcharter, echarts4r:

- https://plotly.com/r/
- https://echarts4r.john-coene.com/
- https://jkunst.com/highcharter/
- https://rstudio.github.io/leaflet/
- https://rstudio.github.io/DT/

Ejemplo de uso de script https://github.com/datosuc/Visualizacion-de-datos-con-R/blob/master/R/script-htmlwidgets.R

Highcharter

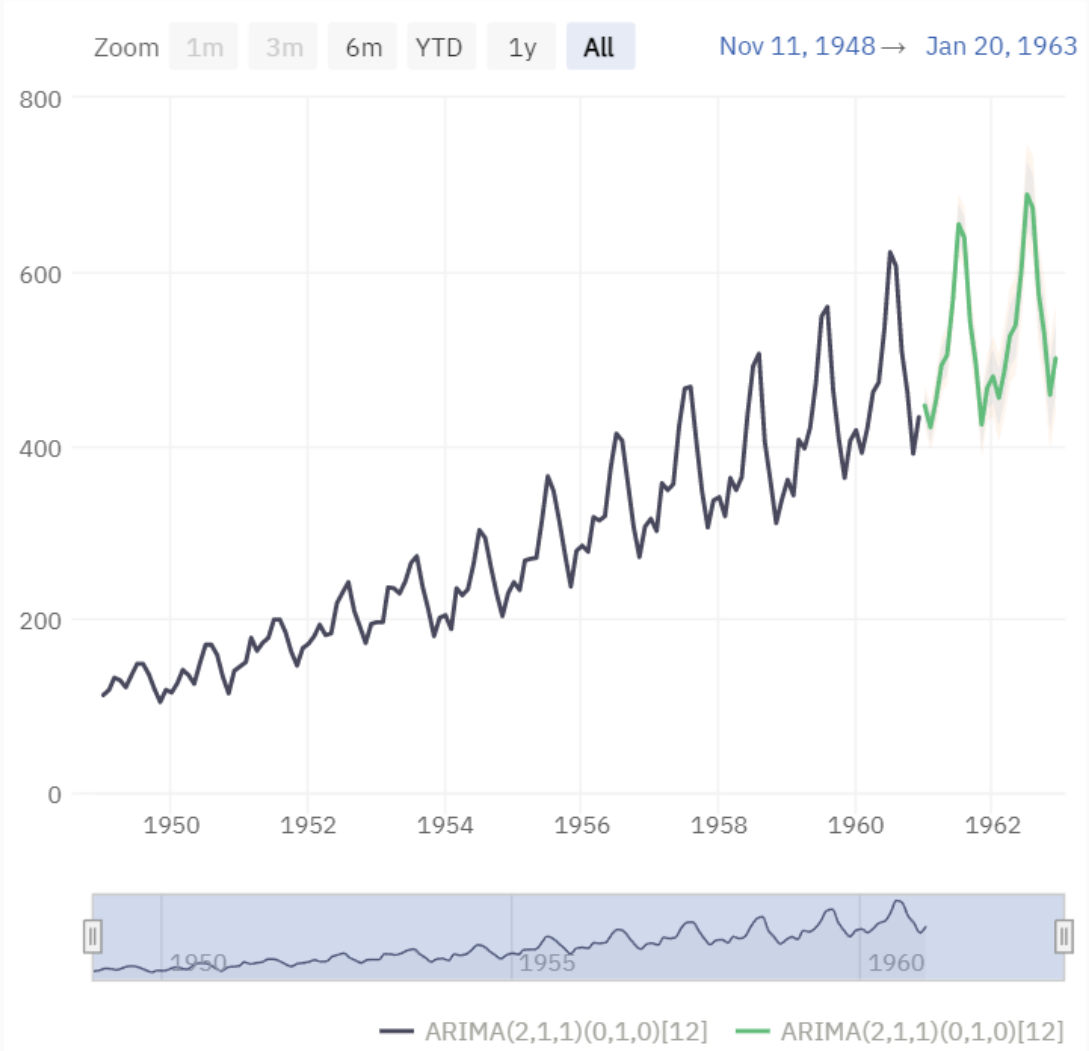
https://jkunst.com/highcharter/

```
library(highcharter)
library(forecast)

data("AirPassengers")

modelo ← forecast(auto.arima(AirPassengers))

hchart(modelo) %>%
  hc_add_theme(hc_theme_hcrt()) %>%
  hc_navigator(enabled = TRUE) %>%
  hc_rangeSelector(enabled = TRUE)
```



DT

- https://rstudio.github.io/DT/

```
library(DT)
library(rvest)    # descargar datos de paginas web

url ← "http://www.sismologia.cl/ultimos_sismos.html"

datos ← read_html(url) %>%
  html_table() %>%
  dplyr::first()

datatable(datos)
```

Show 10 entries Search:

	Fecha Local	Fecha UTC	Latitud	Longitud	Profundidad [Km]
1	2021-11-17 21:37:21	2021-11-18 00:37:21	-21.838	-68.623	125
2	2021-11-17 21:29:15	2021-11-18 00:29:15	-30.127	-72.032	32.8
3	2021-11-17 20:32:18	2021-11-17 23:32:18	-23.851	-69.361	85

Repaso resumido

- Una shiny app consta de dos partes:
  - `ui` (**u**ser **i**nterface) donde definiremos el lugar de los los `input`s que el usuario podrá controlar, como también el lugar de donte estarán los `output`s que retornemos.
  - `server` (**server** XD), donde definiremos que retornaremos en cada output dependiendo de los inputs.

- Los inputs de forma general son de la forma `tipoInput("nombreinput", parametros_del_input)`, por ejemplo `sliderInput("valor", label = "Valor", min = 1, max = 10, value = 1)`.
- En el server accedo al valor del input como `input$nombreinput`.

- Un output se define en la interfaz (gráfico, tabla, mapa, texto) con la forma `tipoOutput("nombreoutput")`, por ejemplo si quiero una salida/output tipo gráfico se usa `plotOutput("grafico")`
- Para enviar un grafico en el server se usa: `output$nombreoutput ← renderTipo({ codigo })`, por ejemplo:

```
output$grafico ← renderPlot({ plot(rnorm(input$valor), type = "l") })
```

shinydashboard

```
library(shinydashboard)

ui ← dashboardPage(
  dashboardHeader(),
  dashboardSidebar(
     sliderInput("valor", label = "Valor", min = 1, max = 10, value = 1)
  ),
  dashboardBody(
    fluidRow(box(width = 12, plotOutput("grafico")))
  )
)
```

Publicar/Compartir tu app

Existen dos formas simples/sencillas para compartir una aplicación. Es decir que sea visible en otros dispositivos pc/móviles. Luego existen otras que requieren de mayor conocimiento técnico.

Vía IP local

Permite que tu pc sirva la aplicación y pueda ser visible

Servicio shinyapps.io

Permite que otro pc (una máquina virtual) sirva tu apliación y la exponga con una url pública. Ejemplo https://usuario.shinyapps.io/nombreapp.

Otras

Como por ejemplo arrendar una máquina virtual linode/digitalocean/aws e instalar shinyserver.