Shiny: Visualizacion de datos con R

Diplomado en Data Science 22, MatPUC

Joshua Kunst Fuentes jbkunst@gmail.com

Version 2022

Johnson

Jue 21/7 - Sáb 23/7 - Jue 28/7 - Jue 4/8

Touring

Mie 27/7 - Mie 02/8 - Mie 10/8 - Mie 17/8

Programa

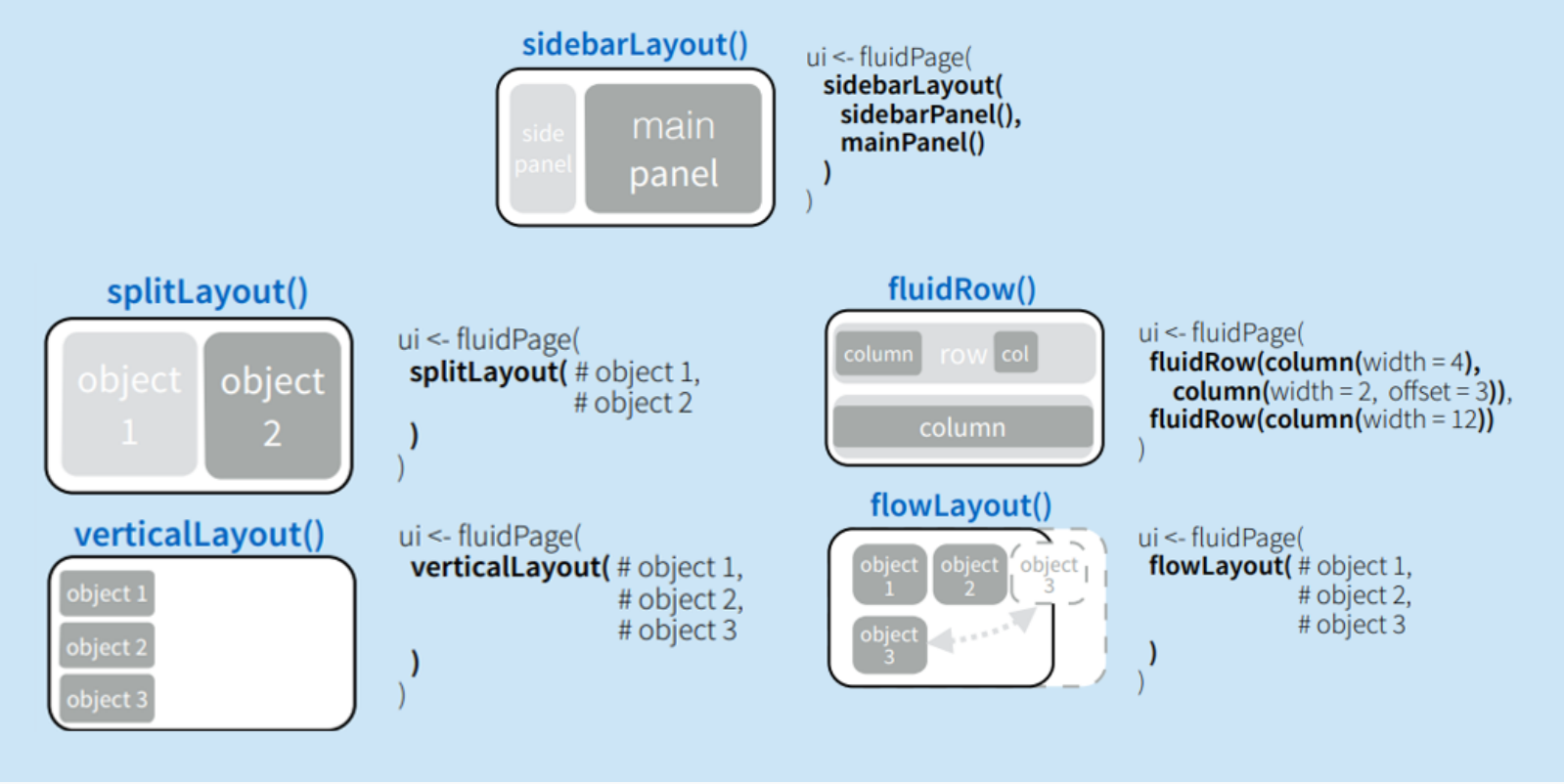
- Clase 1
  - Aplicación (web), ejemplos.
  - Introducción a shiny: Interfaz usuario y servidor
  - Reactividad
- Clase 2
  - Layouts
  - Integración HTMLWidgets
- Clase 3
  - Repaso/Ejercicios
  - Templates y diseño
  - Compartir una app
- Clase 4
  - Expresiones reactivas
  - Extensiones shiny

Layouts

Tipos de Layouts

Laytous se refiere a la disposición de elementos -como inputs, textos, outputs- en nuestra app.

Dependiendo de las necesidaes puede ser convenientes algunos tipos de layuts sobre otros



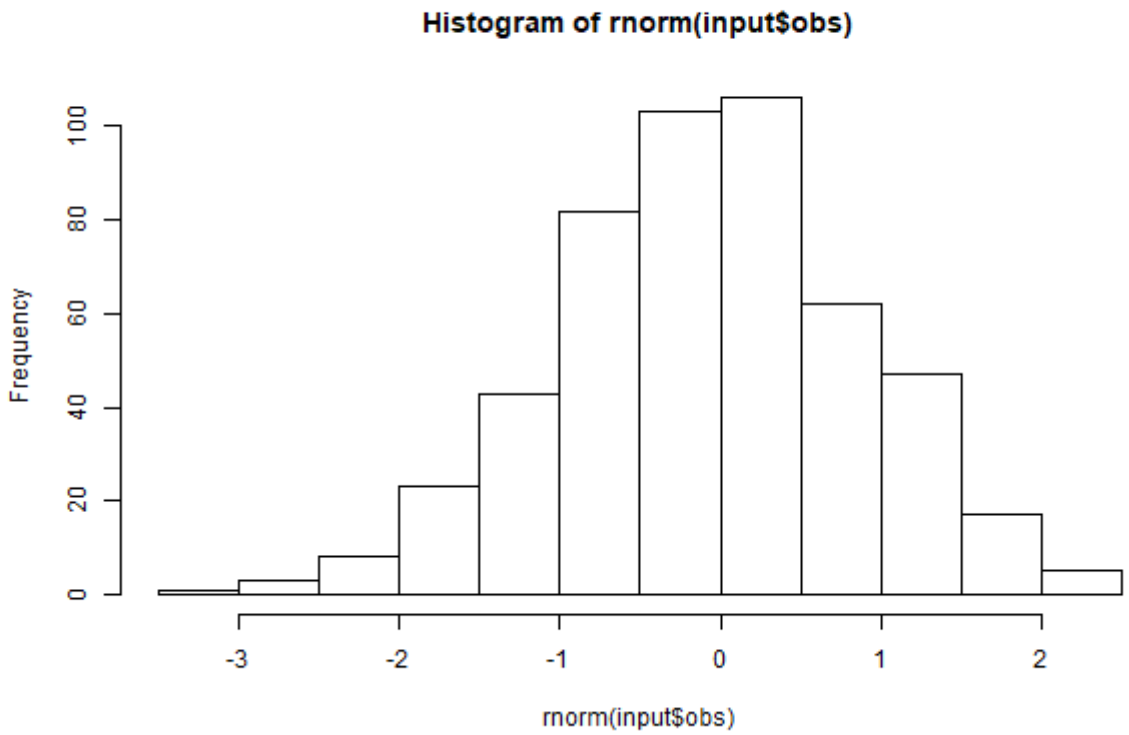
sidebarLayout

Hello Shiny!

Number of observations:

0 500 1,000

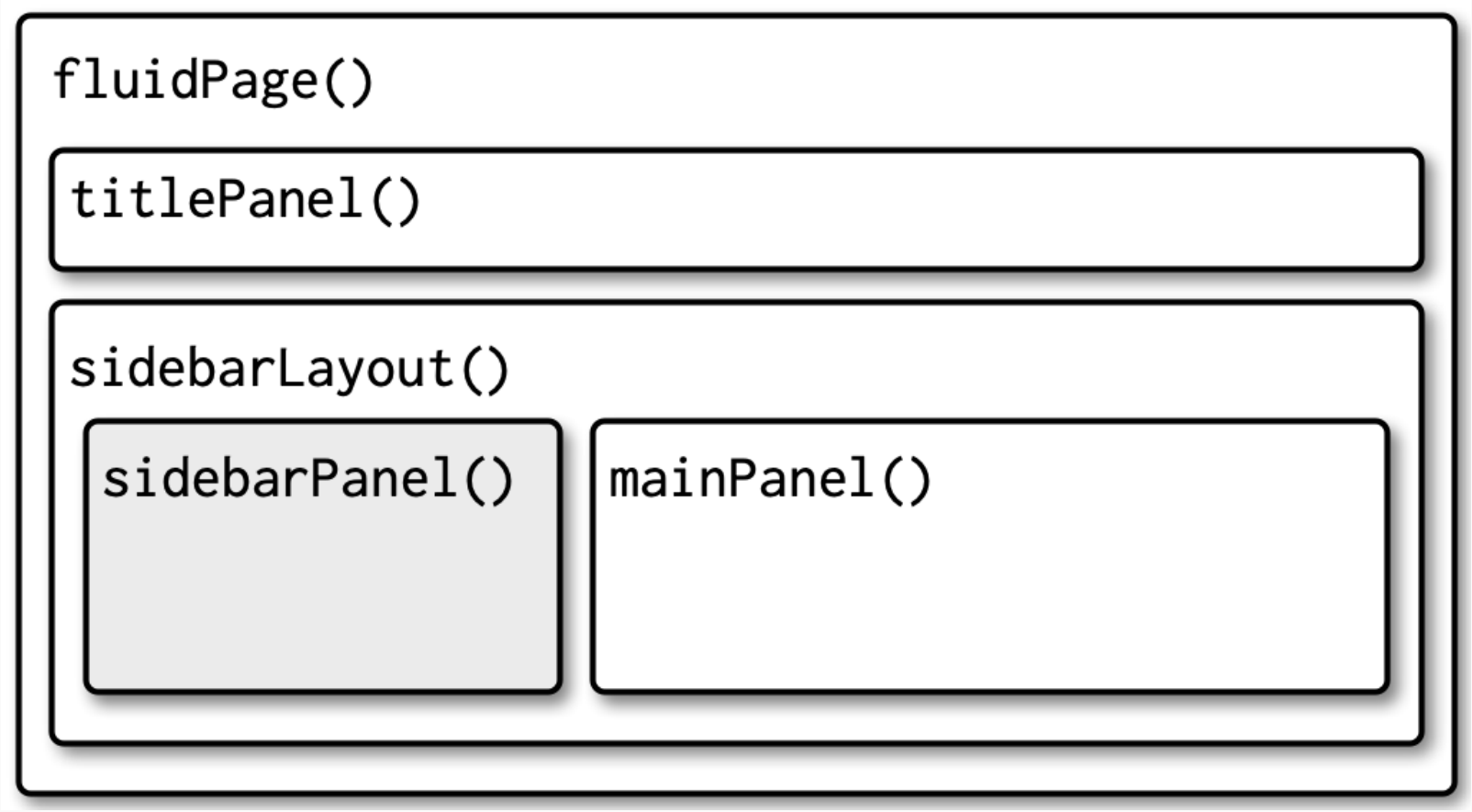
0 100 200 300 400 500 600 700 800 900 1,000



sidebarLayout: Estructura

Es una distribución simple donde lo primero en aparecer es los controles o inputs.

Luego viene el panel principal el cual posee mayor espacio.



sidebarLayout: Ejemplo código

```
library(shiny)

ui ← fluidPage(
  titlePanel("Hello Shiny!"),
  sidebarLayout(
    sidebarPanel(
      sliderInput("obs", "Number of observations:", min = 0, max = 1000, value = 500)
    ),
    mainPanel(plotOutput("distPlot"))
  )
)

server ← function(input, output) {
  output$distPlot ← renderPlot({ hist(rnorm(input$obs)) })
}

shinyApp(ui, server)
```

Paneles de Navegación

Tipos de Paneles de Navegación

Es posible que nuestra aplicación vaya creciendo, en dicho caso, Los paneles nos permiten organizar distintas secciones de la aplicación.

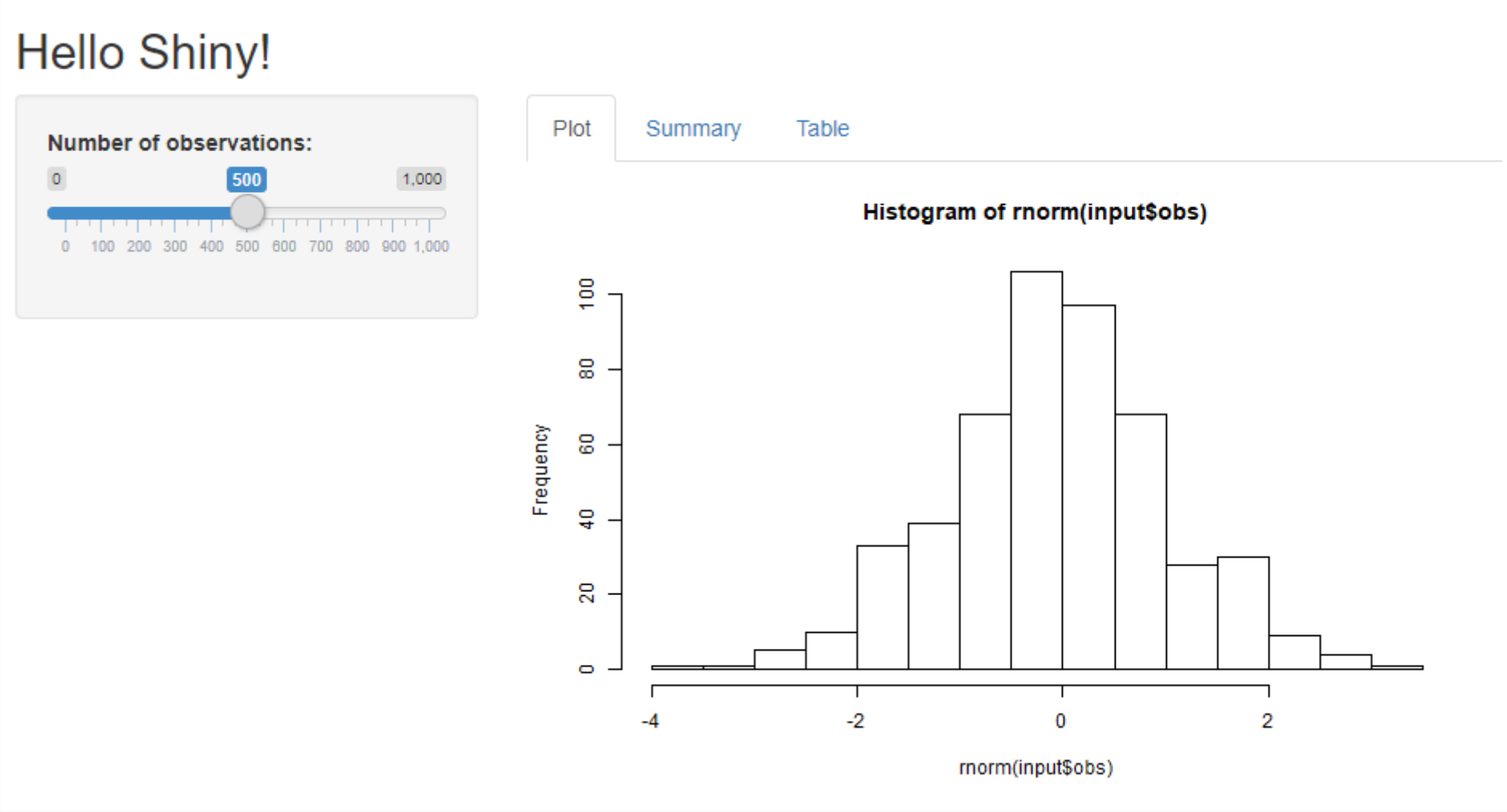
tabsetPanel

Los tabs (`tabsetPanel`) son útiles para separar secciones *similares* en nuestra app. A diferencia de por ejemplo `navbarPage`.

```
library(shiny)

ui ← fluidPage(
  titlePanel("Hello Shiny!"),
  sidebarLayout(
    sidebarPanel(
      sliderInput("obs", "Number of observations:", min = 0, max = 1000, value = 500)
    ),
    mainPanel(
      tabsetPanel(
        tabPanel("Plot", plotOutput("plot")),
        tabPanel("Summary", verbatimTextOutput("summary")),
        tabPanel("Table", tableOutput("tabla"))
      )
    )
  )
)

server ← function(input, output) {
  output$plot ← renderPlot({ hist(rnorm(input$obs)) })
  output$summary ← renderText({ input$obs })
  output$tabla ← renderTable({ data.frame(input$obs) })
}

shinyApp(ui, server)
```

tabsetPanel

Los tabs (`tabsetPanel`) son útiles para separar secciones *similares* en nuestra app. A diferencia de por ejemplo `navbarPage`.



Ejercicio: Modificar la estructura de la aplicación

Nivel *fácil, para calentar las manos*: Para el ejemplo de `sidebarLayout`: Modifique la aplicación para que posea un layout de tipo `flowLayout`.

Nivel *entretenido*: Para el ejemplo de `tabsetPanel` agregue:

- Un tab más con un gráfico de líneas.
- Un tab que utilice la función `summary` para mostra resúmen de los datos.
- Modifique la applicación para utilizar la función `navbarPage`.

Solución 1

```
library(shiny)

ui ← fluidPage(
  titlePanel("Hello Shiny!"),
  flowLayout(
    sliderInput("obs",
                "Number of observations:",
                min = 0,
                max = 1000,
                value = 500),
    plotOutput("distPlot")
    )
  )

server ← function(input, output) {
  output$distPlot ← renderPlot({
    hist(rnorm(input$obs))
    })
}

shinyApp(ui, server)
```

Solución 2

```
library(shiny)
library(ggplot2)

ui ← navbarPage(
  title = "Hello Shiny!",
  tabPanel("Plot",
           sliderInput("obs",
           "Number of observations:",
           min = 0, max = 1000, value = 500),
           plotOutput("plot")
           ),
  tabPanel("Línea", plotOutput("linea")),
  tabPanel("Summary", verbatimTextOutput("summary")),
  tabPanel("Table", tableOutput("tabla"))
)
```

```
server ← function(input, output) {
  output$plot ← renderPlot({
    hist(rnorm(input$obs))
    })
  output$summary ← renderPrint({
    summary(rnorm(input$obs))
    })
  output$tabla ← renderTable({
    data.frame(rnorm(input$obs))
    })
  output$linea ← renderPlot({
    x ← rnorm(input$obs)
    qplot(x = 1:input$obs, x, geom = "line")
  })


}

shinyApp(ui, server)
```

HTMLWidgets

HTMLWidgets

HTMLWidgets son un tipo de paquetes que nos permiten realizar visualizaciones en HTML las cuales se pueden usar en (1) consola, (integrar) integrar con shiny y también (3) rmarkdown.

Existen una gran cantida de paquetes https://gallery.htmlwidgets.org/, y nos sirven para complementar nuestra aplicación.

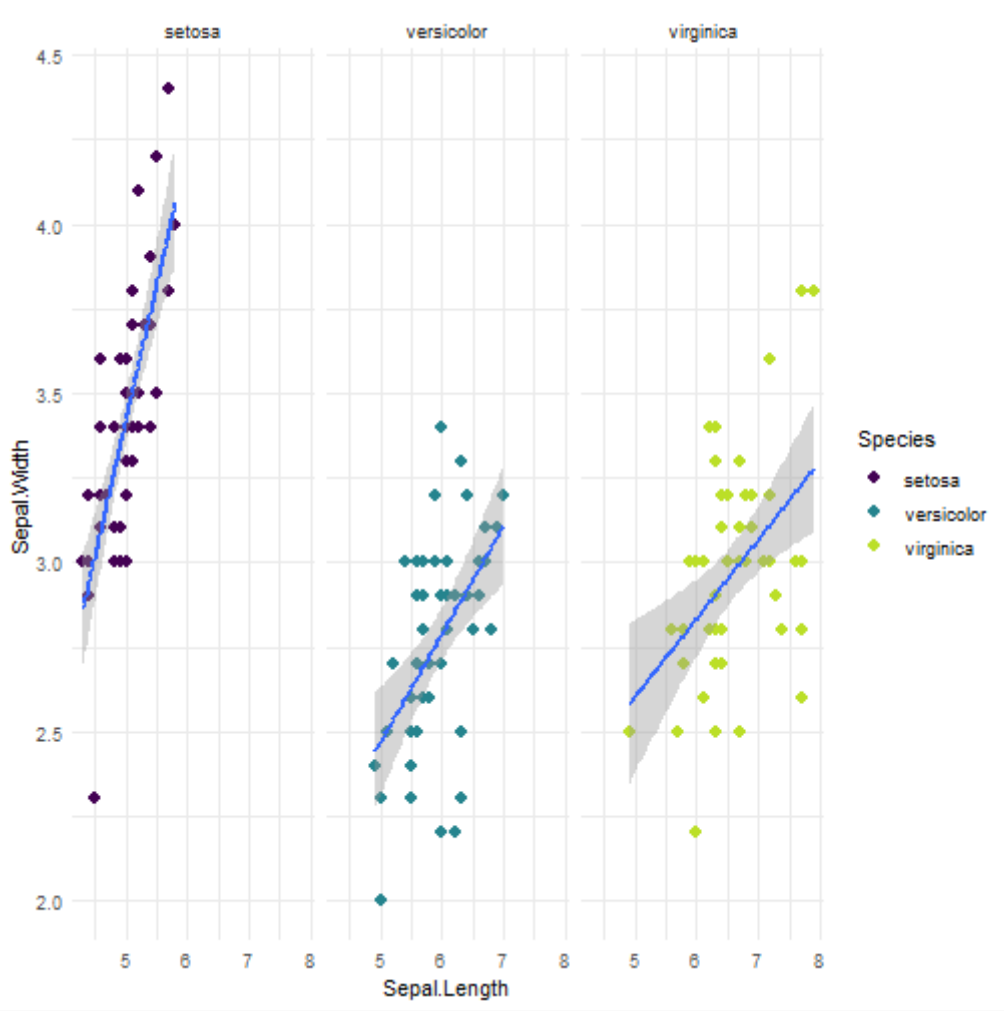
Cada paquete HTMLWidget tiene su propio set de funciones, el código utilizado para hacer un gráfico en plotly no es el mismo (pero generalmente muy similar) al utilizado en highcharter, echarts4r:

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

Ejemplo de uso de script https://github.com/jbkunst/shiny-visualizacion-de-datos-con-R/blob/master/R/script-htmlwidgets.R

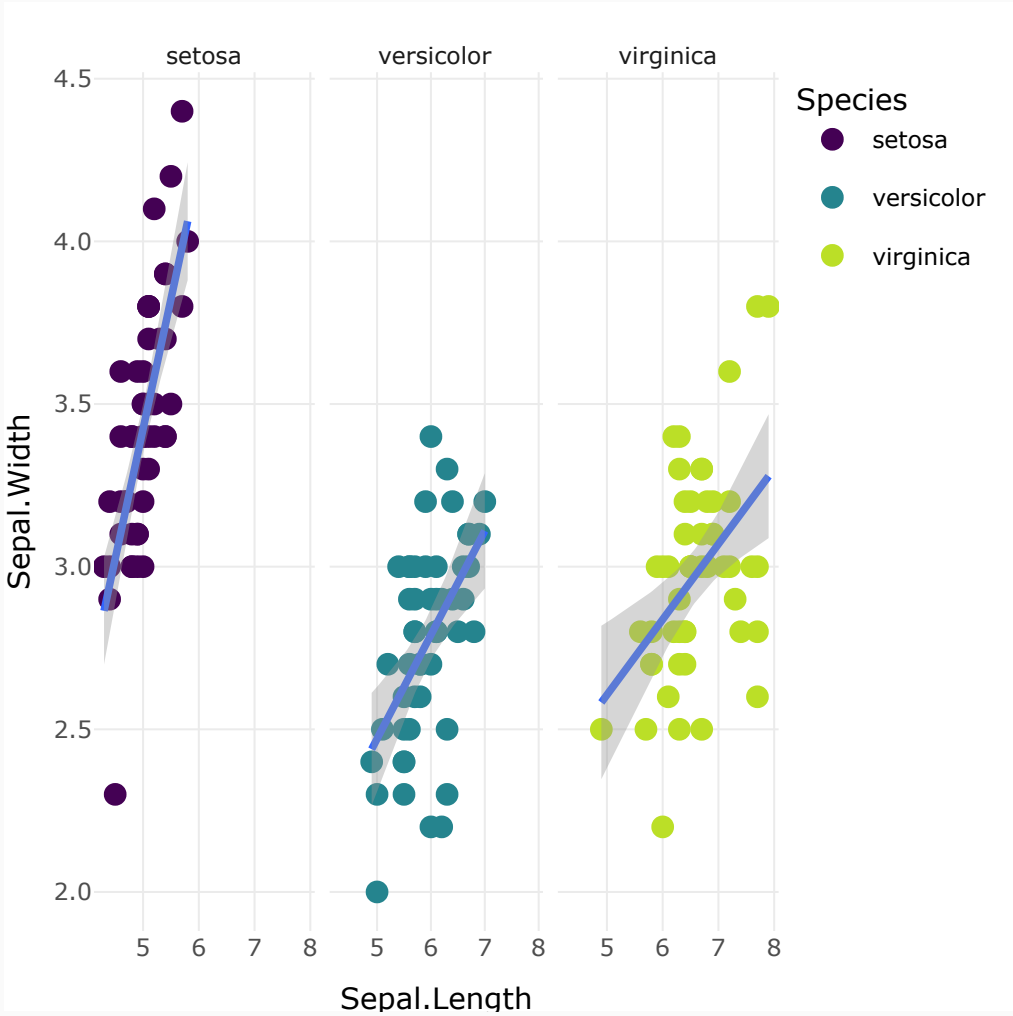
Antes de Plotly, ggplot2

```r
library(ggplot2)

data(iris)

ggplot(iris, aes(Sepal.Length, Sepal.Width)) +
  geom_point(aes(color = Species), size = 2.5) +
  scale_color_viridis_d(end = .9) +
  geom_smooth(method = "lm") +
  facet_wrap(vars(Species)) +
  theme_minimal()
```



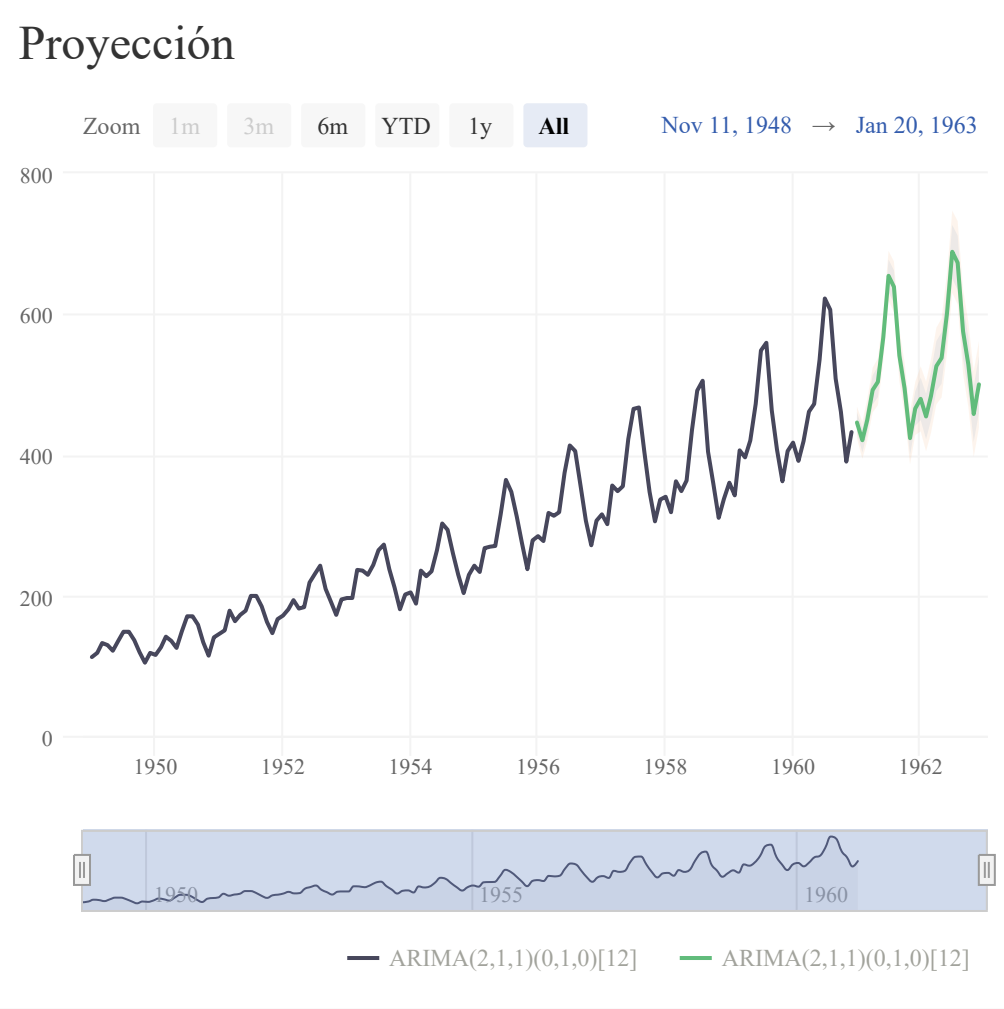
Plotly

```r
library(ggplot2)
library(plotly)

data(iris)

p ← ggplot(iris, aes(Sepal.Length, Sepal.Width)) +
  geom_point(aes(color = Species), size = 2.5) +
  scale_color_viridis_d(end = .9) +
  geom_smooth(method = "lm") +
  facet_wrap(vars(Species)) +
  theme_minimal()

ggplotly(p)
```



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) ▷
  hc_title(text = "Proyección")
```



Antes de seguir, descarguemos datos!

```
library(rvest)  # descargar datos d

url ← "https://www.sismologia.cl/s

read_html(url) ▷
  html_table() ▷
  dplyr::nth(2) ▷
  janitor::clean_names() ▷
  tidyr::separate(
    latitud_longitud,
    into = c("latitud", "longitud")
    sep = " ", convert = TRUE
  )

datos ← read_html(url) ▷
  html_table() ▷
  dplyr::nth(2) ▷
  janitor::clean_names() ▷
  tidyr::separate(
    latitud_longitud,
    into = c("latitud", "longitud")
    sep = " ", convert = TRUE
  )
```

```
## # A tibble: 11 × 6
##    fecha_local_lugar                    fecha…¹ latitud longi…² profu…³ magni…⁴
##    <chr>                                <chr>     <dbl>   <dbl> <chr>   <chr>
##  1 2022-07-21 18:36:3864 km al NO de Va… 2022-0…   -32.6   -72.1 28 km   3.9 Ml
##  2 2022-07-21 17:56:3118 km al SE de Qu… 2022-0…   -33.0   -71.1 63 km   2.7 Ml
##  3 2022-07-21 12:43:2362 km al S de Soc… 2022-0…   -24.1   -67.7 264 km  2.8 Ml
##  4 2022-07-21 12:29:3153 km al SE de So… 2022-0…   -23.9   -67.5 264 km  2.6 Ml
##  5 2022-07-21 10:06:0364 km al SO de Ol… 2022-0…   -21.7   -68.5 133 km  3.5 Ml
##  6 2022-07-21 09:49:0626 km al NE de Si… 2022-0…   -22.7   -69.2 60 km   3.2 Ml
##  7 2022-07-21 05:57:0717 km al SO de La… 2022-0…   -34.4   -71.4 54 km   3.0 Ml
##  8 2022-07-21 05:27:5667 km al SE de So… 2022-0…   -24.1   -67.6 241 km  3.7 Ml
##  9 2022-07-21 04:35:47216 km al SO de M… 2022-0…   -45.2   -75.8 33 km   3.5 Ml
## 10 2022-07-20 22:48:5254 km al NE de Sa… 2022-0…   -22.5   -68.0 197 km  3.1 Ml
## 11 2022-07-20 21:17:0326 km al S de Cal… 2022-0…   -22.7   -68.8 108 km  3.4 Ml
## # … with abbreviated variable names ¹fecha_utc, ²longitud, ³profundidad,
## #   ⁴magnitud_2
```

DT

```
library(DT)

datatable(datos)
```

Show 10 entries Search:

	fecha_local_lugar	fecha_utc	latitud	longitud	profundidad	magnitud
1	2022-07-21 18:36:3864 km al NO de Valparaíso	2022-07-21 22:36:38	-32.637	-72.094	28 km	3.9 Ml
2	2022-07-21 17:56:3118 km al SE de Quillota	2022-07-21 21:56:31	-33.022	-71.149	63 km	2.7 Ml
3	2022-07-21 12:43:2362 km al S de Socaire	2022-07-21 16:43:23	-24.122	-67.718	264 km	2.8 Ml
4	2022-07-21 12:29:3153 km al SE de Socaire	2022-07-21 16:29:31	-23.903	-67.5	264 km	2.6 Ml
5	2022-07-21 10:06:0364 km al SO de Ollagüe	2022-07-21 14:06:03	-21.749	-68.516	133 km	3.5 Ml
6	2022-07-21 09:49:0626 km al NE de Sierra Gorda.	2022-07-21 13:49:06	-22.721	-69.158	60 km	3.2 Ml

Leaflet

```
library(leaflet)

leaflet(datos) ▷
  addTiles() ▷
  addMarkers(
    lng = ~longitud,
    lat = ~latitud,
    popup = ~as.character(magnitud_2),
    label = ~as.character(`fecha_local_lugar`)
  ) ▷
  addProviderTiles("Esri.WorldImagery")
```

Pero como usar HTMLWidgets en nuestra App?

Cada uno de los HTMLWidgets presentados, en su documentación detallan como usarlos en una aplicación shiny.

De forma general en cada paquete existirá una función tipo:

- `*Output` para colocarla en el `ui`.
- `render*` para definirla en el `server`

A modo de ejemplo, el paquete `leaflet` tiene `leafletOutput()` y `renderLeaflet()`.

Los anterior está documentado en https://rstudio.github.io/leaflet/shiny.html.

Ejercicio: Agregar HTMLWidgets a la app

A la aplicación obtenida de modificar la de `tabsetPanel` (Nivel *entretenido*), modificar gráficos y tablas para incluir algunos HTMLWidgets.

Solución

```
library(shiny)
library(ggplot2)
library(plotly)
library(DT)

ui ← navbarPage(
  title = "Hello Shiny!",
  tabPanel("Plot",
           sliderInput("obs",
           "Number of observations:",
           min = 0, max = 1000, value = 500),
           plotOutput("plot")
           ),
  tabPanel("Línea", plotlyOutput("linea")),
  tabPanel("Summary", verbatimTextOutput("summary")),
  tabPanel("Table", DTOutput("tabla"))
)
```

```
server ← function(input, output) {
  output$plot ← renderPlot({
    hist(rnorm(input$obs))
    })
  output$summary ← renderPrint({
    summary(rnorm(input$obs))
    })
  output$tabla ← renderDT({
    d ← data.frame(rnorm(input$obs))
    datatable(d)
    })
  output$linea ← renderPlotly({
    y ← rnorm(input$obs)
    p ← qplot(x = 1:input$obs, y = y, geom = "line")
    ggplotly(p)
  })

}

shinyApp(ui, server)
```