# Computación Estadística con R

## Clase 4

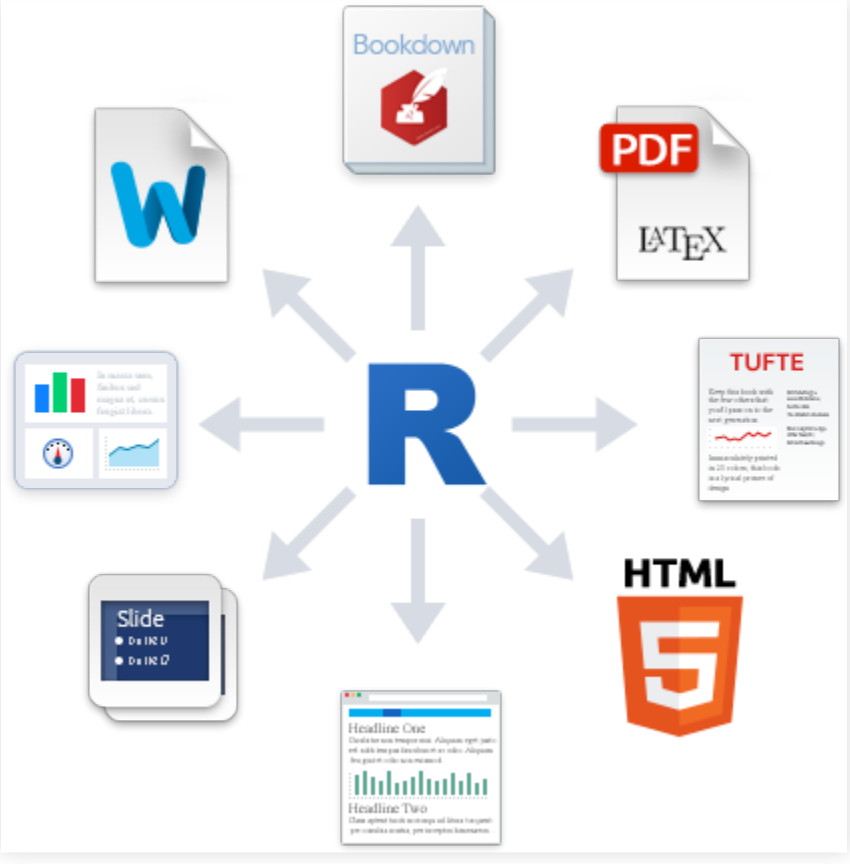
Rubén Soza

# Introducción a R Markdown

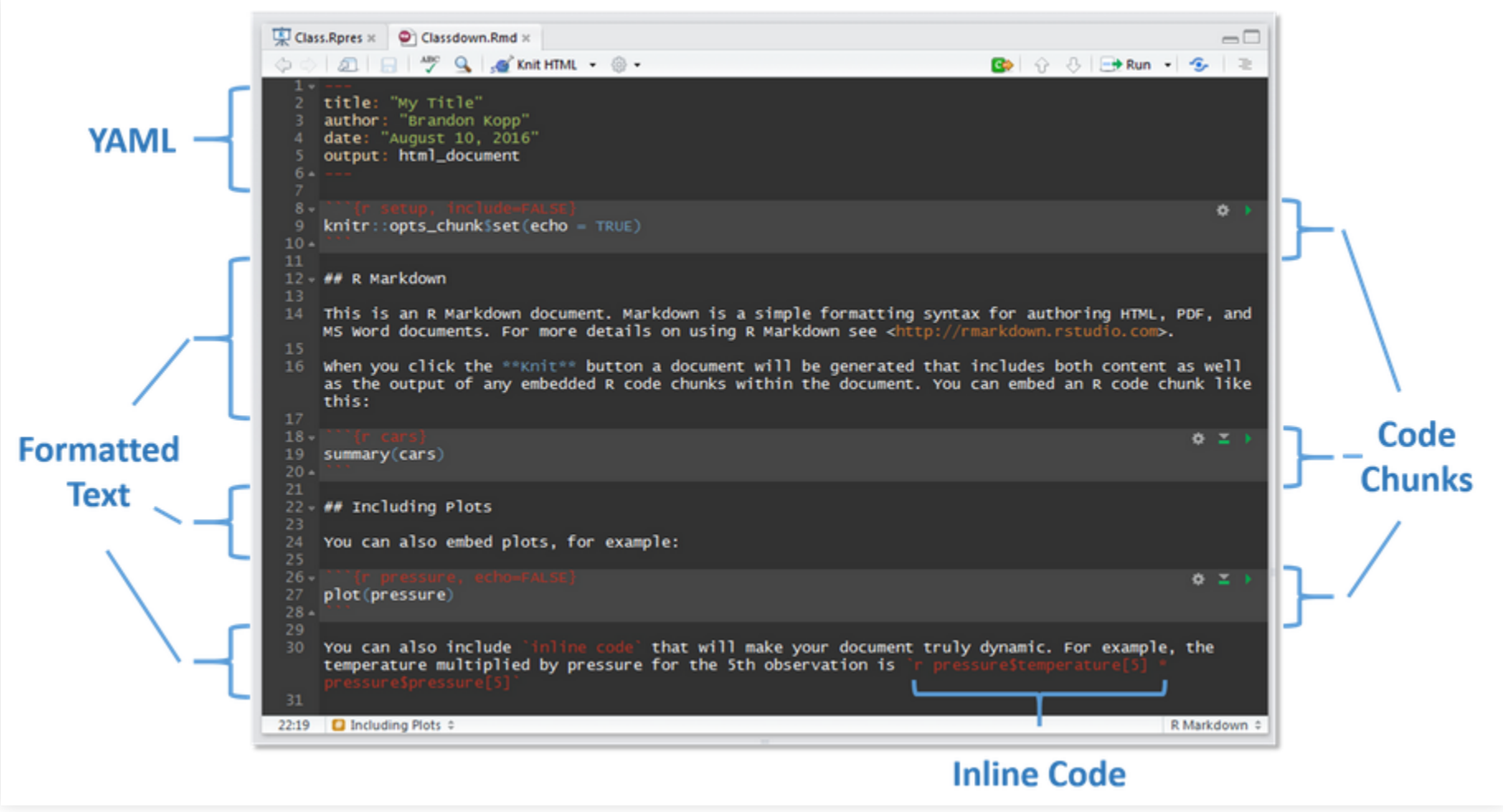
# ¿Qué es R Markdown?

- Marco unificado para ciencia de datos
- Combina:
  - Código
  - Resultados
- Los documentos R Markdown son totalmente reproducibles y automatizables.

# Diferentes formatos de salida



# Primer vistazo




```
---
title: "My Title"
author: "Brandon Kopp"
date: "August 10, 2016"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

## R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```{r cars}
summary(cars)
```

## Including Plots

You can also embed plots, for example:

```{r pressure, echo=FALSE}
plot(pressure)
```

You can also include `inline code` that will make your document truly dynamic. For example, the temperature multiplied by pressure for the 5th observation is `r pressure$temperature[5] * pressure$pressure[5]`
```

# Detrás de escenas

`Ventaja flujo de trabajo de dos pasos:` ¡Se puede crear una amplia gama de formatos de salida!


- Word: Requiere Microsoft Word instalado.
- PDF: Requiere un compilador de LaTeX instalado.
- HTML.

# Sintaxis

- `*cursiva*` y `_cursiva_` -> *cursiva* y *cursiva*
- `**negrita**` y `__negrita__` -> **negrita** y **negrita**
- `[link](www.rstudio.com)` -> link
- `# Encabezado 1`
- `## Encabezado 2`
- `### Encabezado 3`
- imagen: `![](camino/a/imagen.png)`
- `- lista`

# Código

Podemos ingresar código de R en nuestros documentos utilizando los **chunks**(ctrl + alt + I). Existen opciones que permiten manipular la acción de un chunk en específico en nuestro documento. Algunas opciones son:

| **Opción** | **Efecto** |
|---|---|
| include | ¿Muestra el fragmento de código de R y su resultado? |
| echo | ¿Muestra el fragmento de código de R? |
| message | ¿Muestra los mensajes de salida? |
| warning | ¿Muestra las advertencias? |
| eval | ¿Evalúa el fragmento de código? |

Para más información ver el siguiente link

# YAML

Aquí se escriben opciones generales del documento. Se pueden configurar, entre muchas otras cosas:

- Fuente y formato
- Tamaños de figuras o gráficos
- Agregar un CSS

# Código entre texto

Podemos realizar código r en cualquier oración. Para ello basta escribir

```
Some text 'r CODE GOES HERE' some more text.
```

Para mayor información de todo esto, ver el siguiente link

# Veamos un ejemplo en R

# Utilizar Rmarkdown para reportar tablas

# Paquetes útiles

Existen diferentes paquetes para generar tablas en tus reportes. Veamos 2 de ellos: `kableExtra` y `DT`. Primero cargamos la base de datos y los paquetes a utilizar:

```
library(tidyverse)
library(kableExtra)
library(DT)
data(iris)
```

# kableExtra

La función `kable()` genera una tabla que puede ser editada utilizando funciones de la librería `kableExtra`. Veamos un ejemplo sencillo:

```
kable(iris, "html") %>%
  kable_styling("striped",position = "center", full_width = F) %>%
  column_spec(2, bold = T, color = "red") %>%
  add_header_above(c("Sepal" = 2, "Petal" = 2, " " = 1)) %>%
  pack_rows("Grupo 1", 1,4) %>%
  pack_rows("Grupo 2", 5,8) %>%
  scroll_box(width = "100%", height = "500px") %>%
  footnote("Tabla de Iris")
```

| Sepal | | Petal | | |
|---|---|---|---|---|
| **Sepal.Length** | **Sepal.Width** | **Petal.Length** | **Petal.Width** | **Species** |
| **Grupo 1** | | | | |
| 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| 4.9 | 3.0 | 1.4 | 0.2 | setosa |
| 4.7 | 3.2 | 1.3 | 0.2 | setosa |
| 4.6 | 3.1 | 1.5 | 0.2 | setosa |
| **Grupo 2** | | | | |
| 5.0 | 3.6 | 1.4 | 0.2 | setosa |
| 5.4 | 3.9 | 1.7 | 0.4 | setosa |
| 4.6 | 3.4 | 1.4 | 0.3 | setosa |

Para mayor información ver este link

# DT

La librería `DT` permite realizar tablas más customizables.

```
library(DT)
datatable(iris, filter = "top")
```

Para más información ver este link

# Ejercicios: parte I (Basados en DataCamp )

Convertir los títulos de las secciones de estos párrafos en encabezados utilizando la sintaxis apropiada, usando diferentes niveles de encabezado para las secciones y subsecciones:

- Secciones (encabezados de primer nivel): Introducción, La base de datos, Computando el nivel de dificultad, Computando la incertidumbre, Una métrica final.
- Subsecciones (encabezados de segundo nivel): Chequeando la base de datos, Graficando el perfil de dificultad, Detectando niveles difíciles, Mostrando incertidumbre
- Genere un índice para su archivo

# Ejercicios: parte II

- En la línea 10, convertir “Candy Crush Saga” en negrita
- Convertir “King” (línea 10) en un link hacia: https://es.wikipedia.org/wiki/King_(empresa)
- Enfatice el texto en la línea 141 convirtiéndola en itálica
- Incluir la imagen del siguiente link en la Introducción: http://www.garotasgeeks.com/wp-content/uploads/2014/05/candy-crush1-610x240.png

# Ejercicios: parte III

- Quitar los mensajes que genera el cargar la librería en la línea 18
- Cargar los datos y mostrar las primeras filas de la base de datos (línea 36) sin que me muestre el código ni los mensajes
- Sabiendo que el código para calcular el número de jugadores y el período que abarcan los datos es length(unique(data*player¡d*))*yrange*(*data*dt), respectivamente, completar con un código incrustado la línea 44

# Ejercicios: parte IV

- Agregar el theme “paper”
- ¿Qué pasa si agrego la opción ‘code_folding: show’?

# Ejercicios: parte V

- La línea 40 muestra un preview de la base de datos utilizada. Modifique este chunk para mostrar una tabla interactiva con filtros.

# Introducción a ShinyR

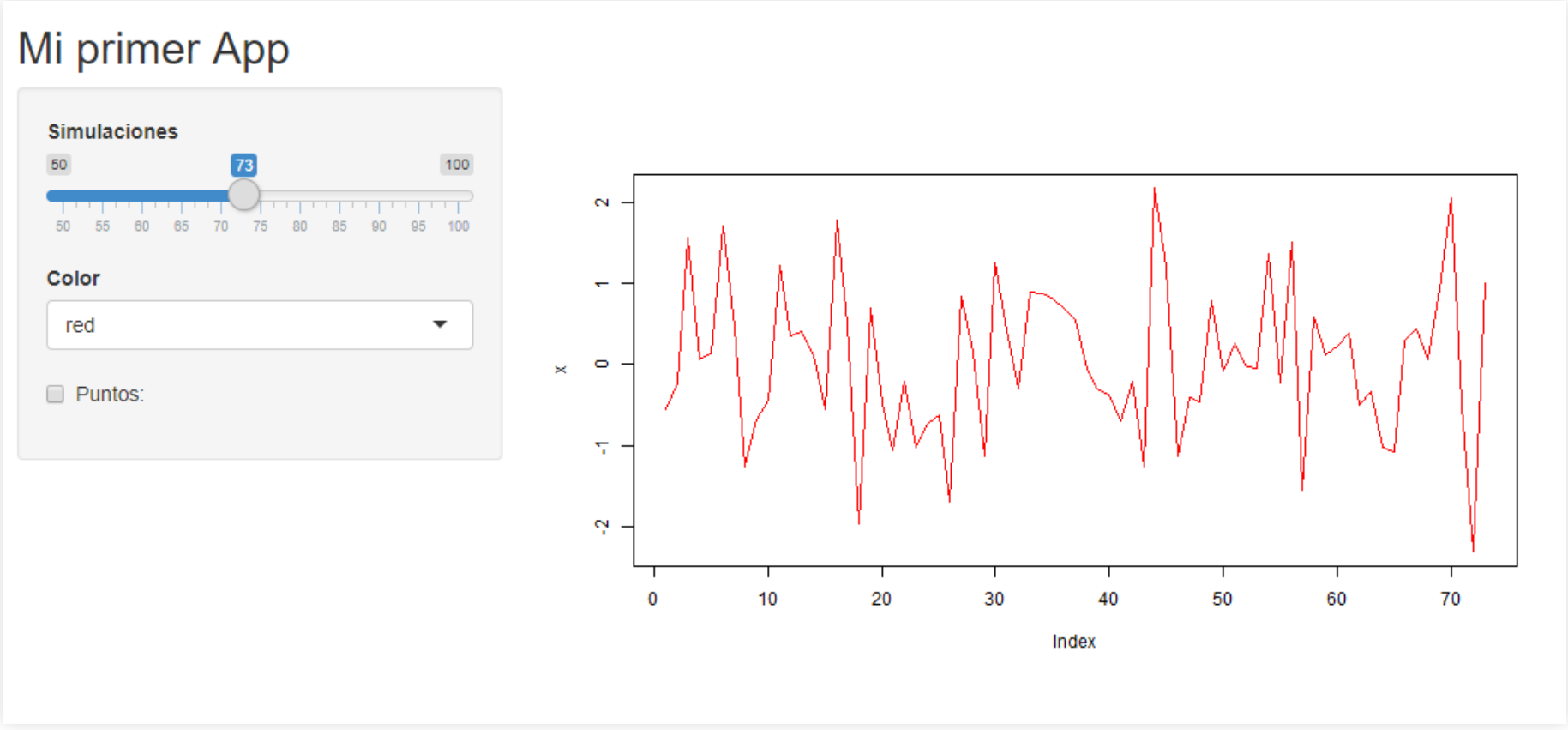
# Antes de Partir

Necesitaremos algunos paquetes:

```
install.packages(
  c("tidyverse", "shiny", "shinythemes", "shinyWidgets",
    "shinydashboard", "DT", "leaflet", "plotly")
  )
```

# ¿Qué es una app(licación) web?

(Wikipedia:) Herramientas que los usuarios pueden utilizar accediendo a un servidor web a través de internet o de una intranet mediante un navegador.



# Aplicación Web

Las apps poseen **inputs** y **outputs**:

Veamos el siguiente link: https://ruben-soza.shinyapps.io/actividades/

# La estructura de una ShinyApp

```
library(shiny)

ui <- fluidPage()

server <- function(input, output) {}

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

# 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))
```

- Se define el `server` en donde estará toda la lógica de nuestra aplicación.
- Principalmente serán instrucciones que dependeran de `inputs`

# La estructura de una ShinyApp

```
library(shiny)

ui <- fluidPage()

server <- function(input, output) {}

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

- `runApp` es la funcion que crea y deja corriendo la app con los parámetros otorgados.
- **No siempre** tendremos que escribirla pues veremos que RStudio al crear una shinyApp nos pondrá un botón para *servir* la aplicación

# Ejercicio:

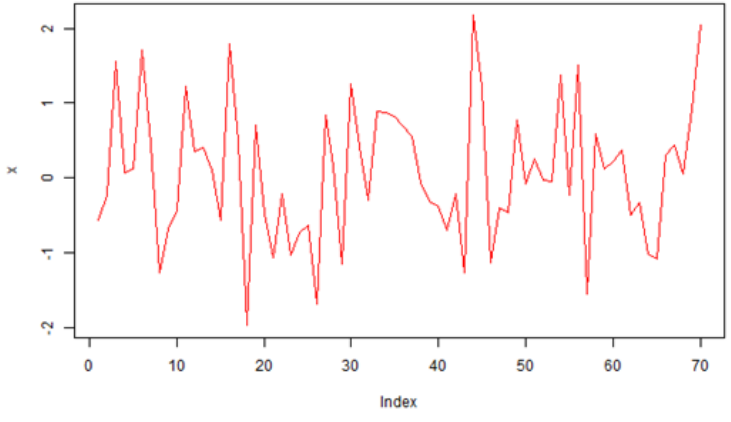
Hacer funcionar el código `mi_primer_shiny.R` en R Rstudio:

```
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)
```

# Funcionamiento de una app de Shiny

# App

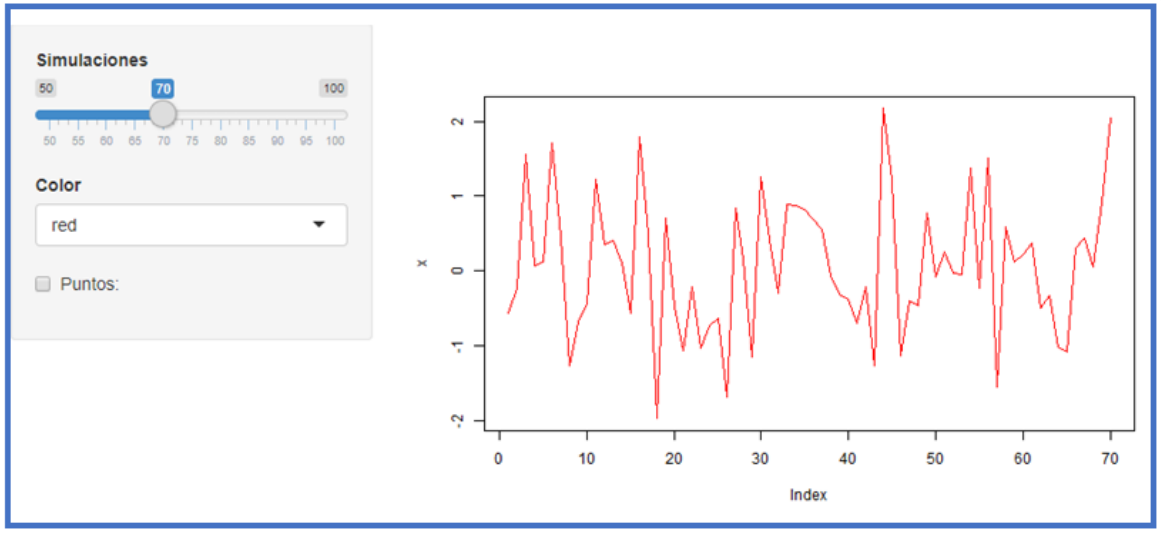




```r
library(shiny)

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)
   })
}
```

# Contenedor

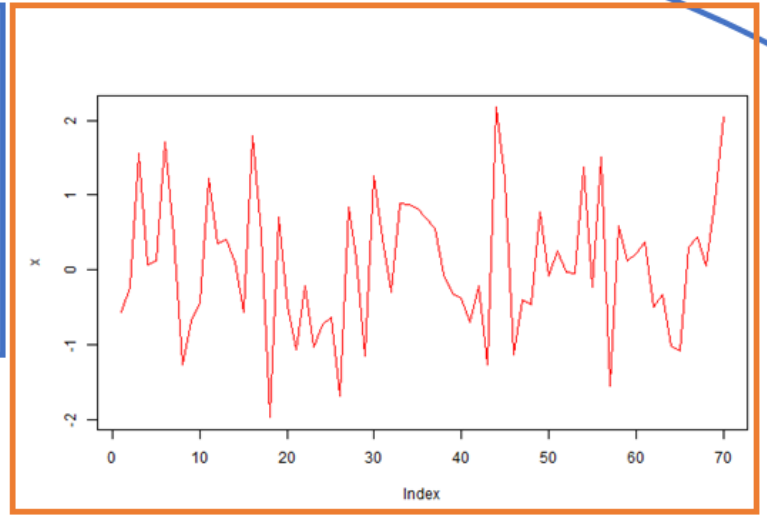



```
library(shiny)

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)
   })
}
```

# Otros contenedores

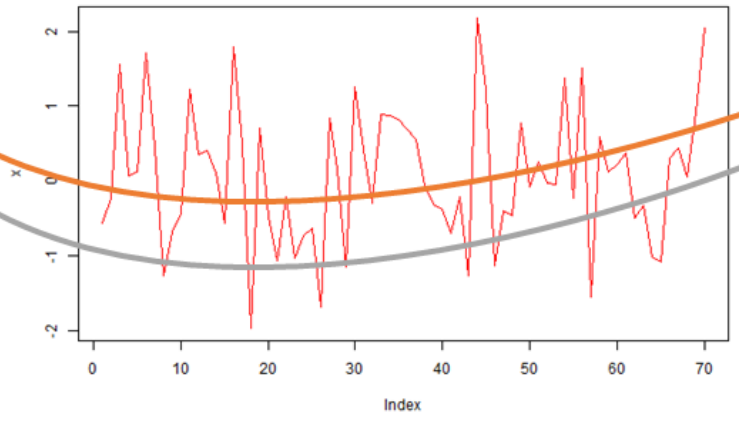




```
library(shiny)

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)
  })
}
```

# Inputs

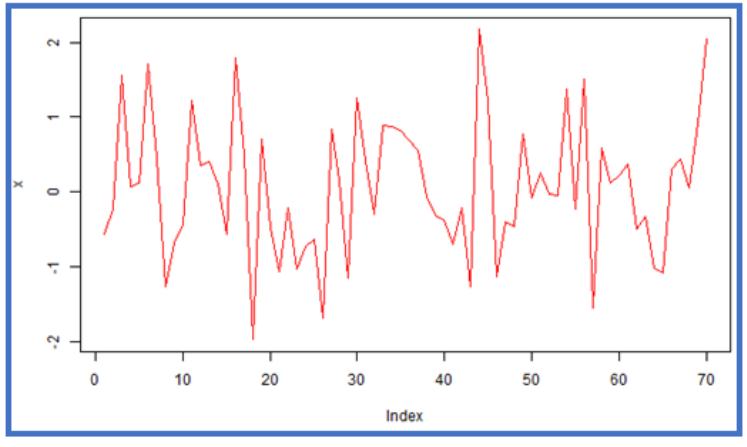




```r
library(shiny)

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)
  })
}
```

# Outputs

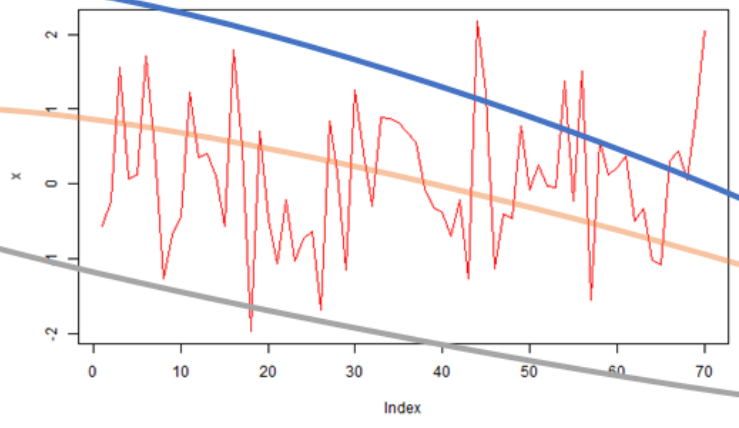




```r
library(shiny)

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)
   })
}
```

# Interacción

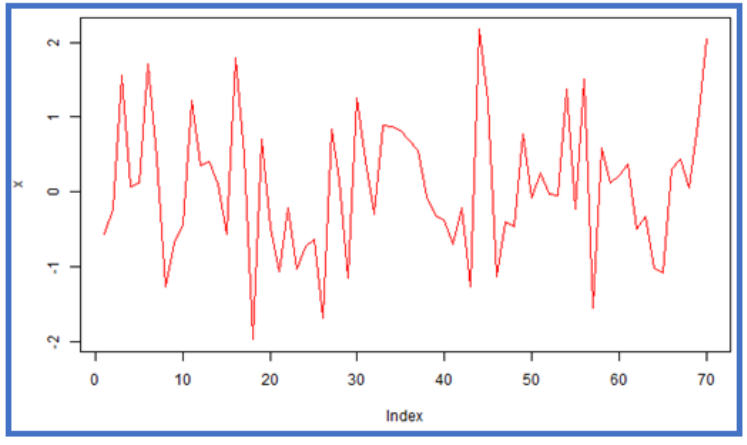




```r
library(shiny)

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)
   })
}
```

# Resultado


```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. Más detalles en http://shiny.rstudio.com/articles/layout-guide.html

```r
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.

- Estas funciones generan el input deseado en la app y shiny permite que los valores de estos inputs sean usados como valores usuales en R en la parte del server (numericos, strings, booleanos, fechas).

```
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)
   })
}
```

- `plotOutput` define el lugar donde la salida estará
- Como mencionamos, nuestras app pueden tener muchos outputs: tablas, texto, imágenes

```r
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`

```
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)
   })
}
```

- Este espacio determina la lógica de nuestra salida
- Acá haremos uso de los inputs para entregar lo que deseamos

# Interacción entre Inputs y Outputs

```
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)
   })
}
```

- Las funciones `Output()` y `render()` trabajan juntas para agregar salidas de R a la interfaz de usuario
- En este caso `renderPlot` esta asociado con `plotOutput` (¿cómo?)
- Existen más parejas: `renderText`/`textOutput` o `renderTable`/`tableOutput`.

```
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**
- Este **id** debe ser único en la application
- En el ejemplo `renderPlot` esta asociado con `plotOutput` vía el `outplot`

```
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/modificado por el usuario se puede acceder en el server bajo `input$nrand`

```
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` se usa para seleccionar un valor numérico entre un rango
- `selectInput` otorga la posibildad que el usuario escoge entre un conjunto de valores
- `checkboxInput` en el server es un valor lógico `TRUE`/`FALSE`

- ¿Necesitas más? https://gallery.shinyapps.io/065-update-input-demo/ y `shinyWidgets::shinyWidgetsGallery()`
- Si deseas compartir tu ShinyApp puede ser útil https://shiny.rstudio.com/tutorial/written-tutorial/lesson7/

# Ejercicio 2

- Haga click en *File*, luego *New File* y *Shiny Web App*, seleccione el nombre
- Ejecutela con *Run App* e interactúe
- Luego modifique y cree una app que contenga:
- 2 inputs, un `sliderInput` y un `textInput`
- 3 output de tipo texto `textOutput` donde el primer contenga el valor del primer input, el segundo el valor del segundo input, y el tercero la suma de los dos inputs