Visualizacion de datos con R: Shiny

Diplomado en Data Science, MatPUC

Joshua Kunst

Antes de Partir

Antes de Partir

Asumimos que tenemos conocimiento de como funciona R, paquetes, funciones, etc.

No es necesario en `shiny` pero usaremos los paquetes `dplyr` y `ggplot` principalmente para hacer manipulación y visualización de los datos

Necesitaremos algunos paquetes:

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

La prestación podrá acceder desde https://github.com/datosuc/Visualizacion-de-datos-con-R y el código fuente, apps, ejemplos en https://github.com/datosuc/Visualizacion-de-datos-con-R/tree/master/apps

Ayuda

No olvidar que una buena forma de aprender es con la documentación oficial:

- https://shiny.rstudio.com/tutorial/
- https://shiny.rstudio.com/tutorial/written-tutorial/lesson1/
- https://github.com/rstudio/cheatsheets/raw/master/shiny.pdf
- https://github.com/rstudio/cheatsheets/raw/master/translations/spanish/shiny_Spanish.pdf

La infaltable chuleta:

https://content.cdntwrk.com/files/aT0xMTI0NDEzJnY9MSZpc3N1ZU5hbWU9c2hpbnktc3BhbmlzaCZjbWQ9ZCZzaWc9ZTdhMThkY2VjYzM1Y

Temas a tratar

- Aplicación (web)
- Shiny
- Layouts
- HTMLWidgets
- Templates y disenio
- Expresiones reactivas, memoización

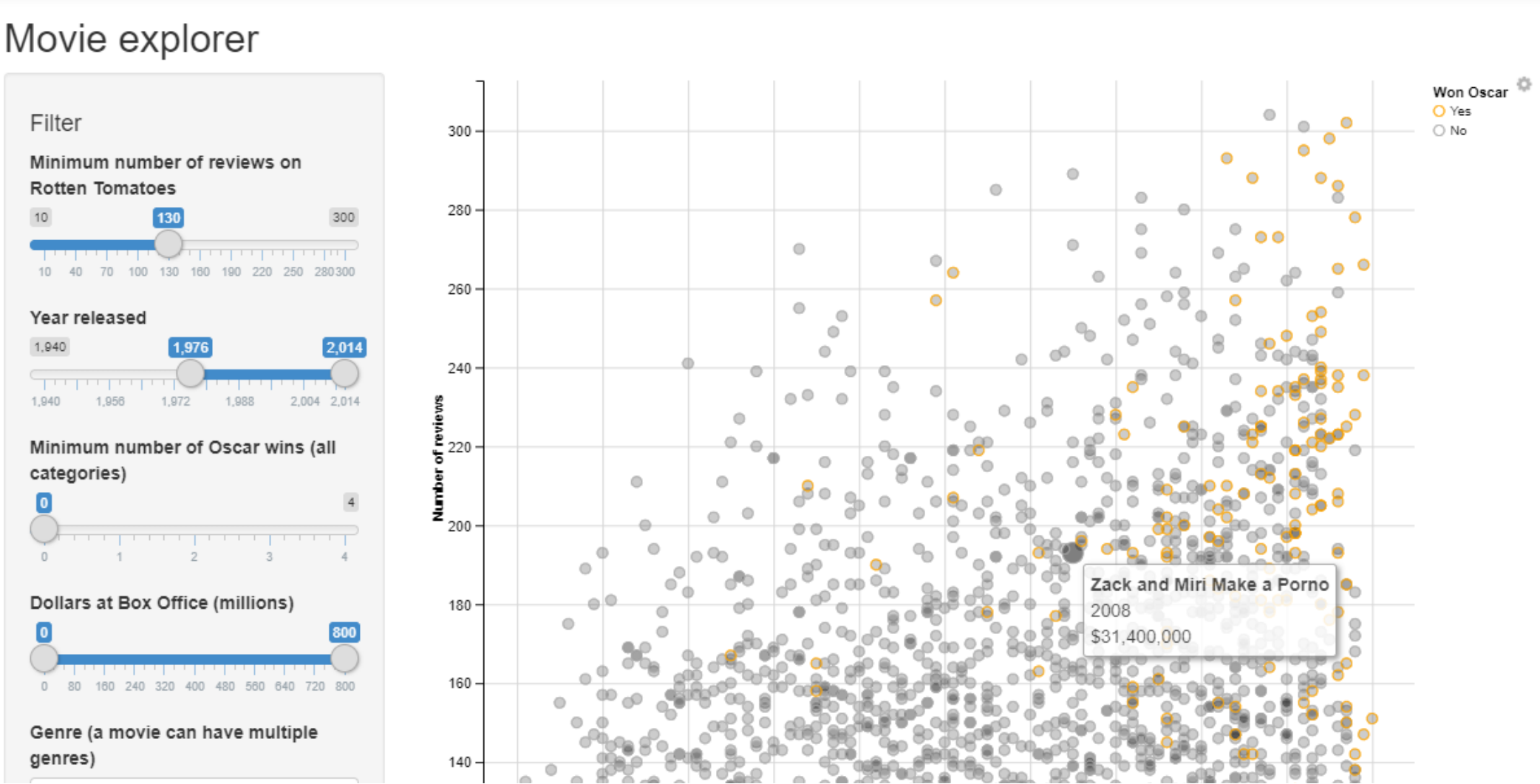
¿Qué es una **app**(licación) web?

Aplicación 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

App con mas `input`s y `output`s



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))
```

- Se define el `server` en donde estará toda la lógica de nuestra aplicación.
- Principalmente serán instrucciones que dependeran de `inputs` y reflejaremos `outputs`: como tablas, gráficos.

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: Nuestra primer App andando

Hacer funcionar el siguiente `código` en R Rstudio: (hint: sí, copy + paste + run)

```
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)
   })
}

shinyApp(ui, server)
```

Funcionamiento de una app de Shiny

App

Contenedor

Otros contenedores

Inputs

Outputs

Interacción

Resultado

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)
   })
}
```

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)
   })
}
```

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

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)
   })
}
```

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

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)
   })
}
```

- 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?)
- Hay muchas parejas como `renderText` / `textOutput` o `renderTable` / `tableOutput` entre otras (revisar la sección de outputs en el cheat sheet)

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`

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` 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 http://shinyapps.dreamrs.fr/shinyWidgets/

Ejercicio: Inputs y outputs vengan a mi!

Haga click en:

- *File*, luego *New File* y *Shiny Web App*, seleccione el nombre
- Ejecutela con *Run App* e intearctúe
- 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

Hints importantes:

- No tema a escribir, ni preguntar!
- Está totalmente permitido equivocarse, de hecho se pondrán puntos extras
- Posible solución estará en https://github.com/datosuc/Visualizacion-de-datos-con-R/blob/master/apps/02-ejercicio-2/app.R

Layouts

Tipos de Layouts

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

Recorreremos algunos más comunes cono el sidebarLayout y tabsetPanel.

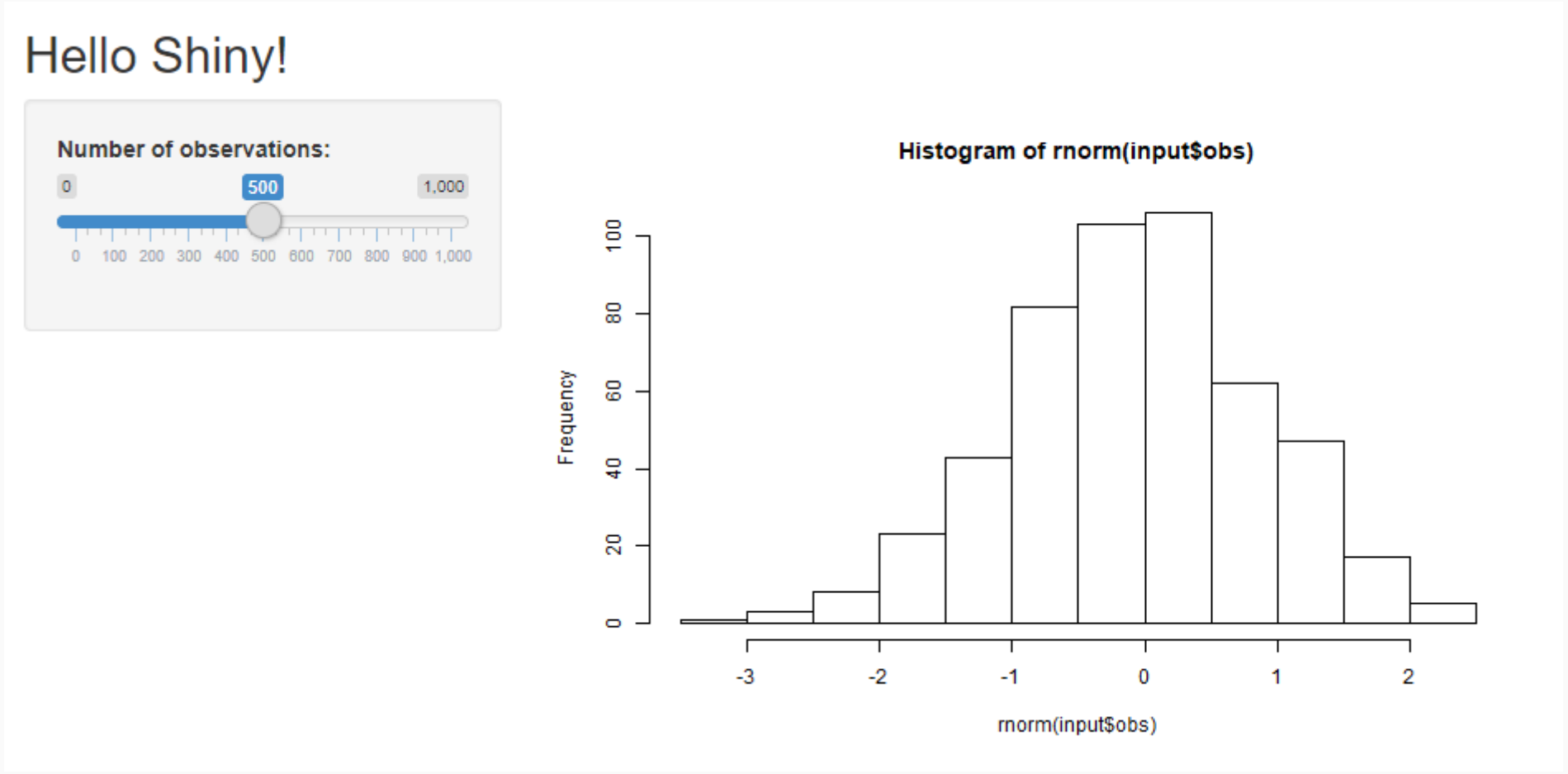
sidebarLayout

El más usuado, generalmente los inputs están agrupados a mano izquierda y

```
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)
```

sidebarLayout

El más usuado, generalmente los inputs están agrupados a mano izquierda y



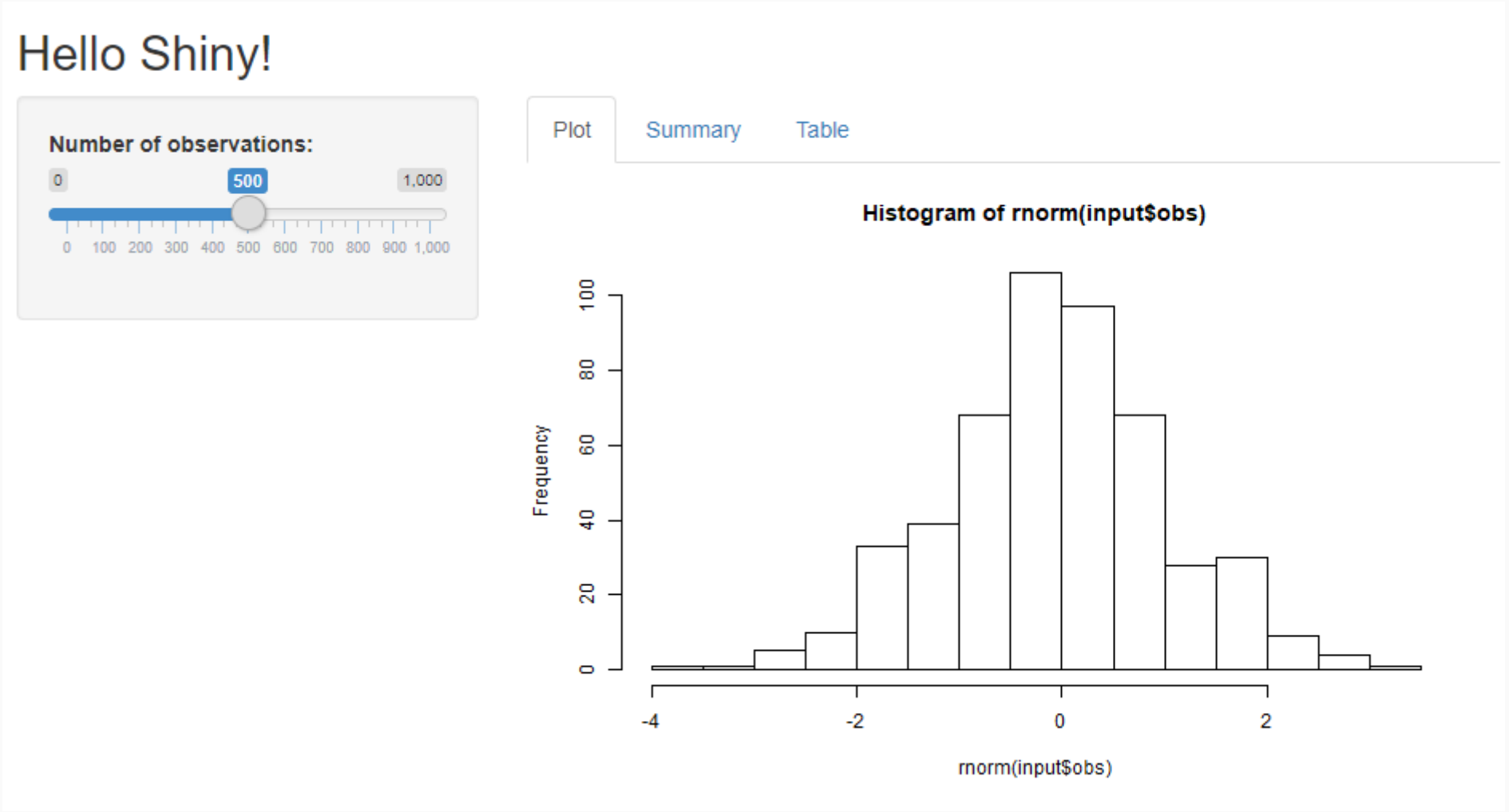
tabsetPanel

Los tabs son útiles para separar secciones en nuestra app

```
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"))
      )
    )
  )
)
inf
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 son útiles para separar secciones en nuestra app



HTMLWidgets

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.

Ejercicio: Transformando script R en una App

- Inspeccionar, ejecutar y modificar el script https://github.com/datosuc/Visualizacion-de-datos-con-R/blob/master/apps/script-export.R (la siguiente slide tambien lo tiene).
- Generar una app que tenga como input una lista de países y muestre el forecast de las exportaciones de dicho país.

Código

```r
if(!require(forecast)) install.packages("forecast")
if(!require(xts)) install.packages("xts")
if(!require(tradestatistics)) install.packages("tradestatistics")
if(!require(ggplot2)) install.packages("ggplot2")

library(forecast)
library(xts)
library(tradestatistics)
library(ggplot2)

pais <- "chn"  # seteo pais

data <- ots_create_tidy_data(years = 1990:2018, reporters = pais, table = "yr")

valores <- data$export_value_usd
fechas <- as.Date(paste0(data$year, "0101"), format = "%Y%m%d",)

serie <- xts(valores, order.by = fechas) # creo la serie de tiempo para la fucion forecast

prediccion <- forecast(serie, h = 5) # realizo automágicamente una predicción

# IMPORTANTE: estas predicciones no *son las mas mejores* la idea
# del ejercicio es imaginar que tenemos un proceso el
# cual transformaremos en una app
autoplot(prediccion)
```

Repaso resumido

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_inpout)`, por ejemplo `sliderInput("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") })
```

Repaso resumido

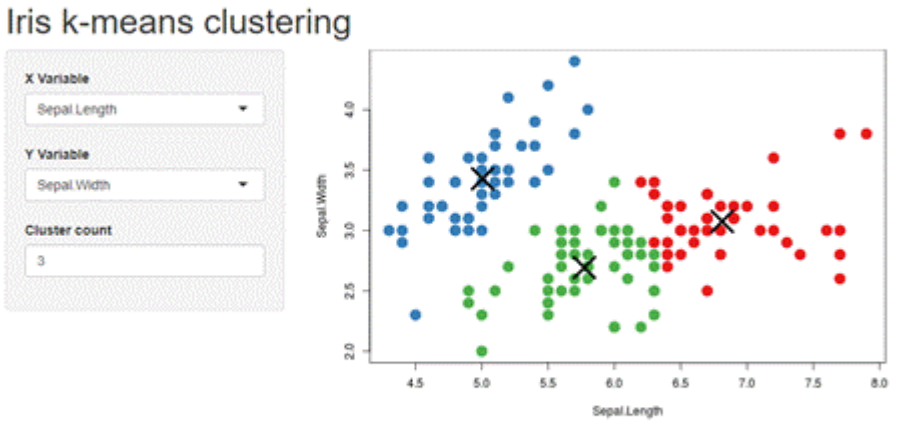
Así nuestra app de repaso quedaría:

```
library(shiny)

# Antes del ui y server podemos cargar paquetes
# o datos que nuestra app usará. No tiene por que ser todo
# tan simple

ui ← fluidPage(
  sliderInput("valor", label = "Valor", min = 1, max = 10, value = 1),
  plotOutput("grafico")
  )

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

shinyApp(ui, server)
```

Temas & Estilos

Temas & Estilos

- Al principio todas nuestras app son similares.
- Existen extensiones/paquetes que permiten cambiar el estilo/look de la aplicación.



shinythemes

Los más fácil de implementar, sin tan alto impacto en código ni imagen. Opciones en http://bootswatch.com/

shinythemes

Antes:

```
library(shiny)

ui ← fluidPage(
   sidebarLayout( ...
```

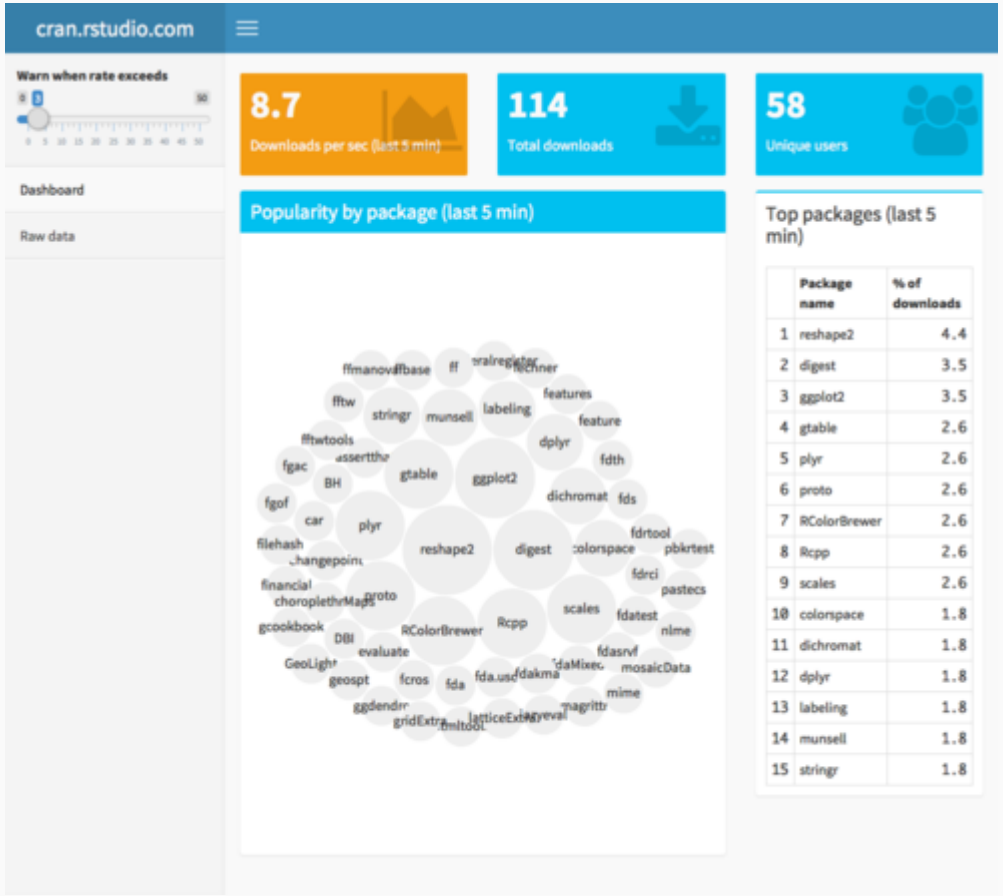
Ahora:

```
library(shiny)
library(shinythemes)

ui ← fluidPage(
   theme = shinytheme("superhero"),
   sidebarLayout( ...
```

NOTAR que este cambio es solo en la parte ui. La parte del server no cambia.

shinydashboard

Orientados a **dashboards** agrega más funcionalidades



shinydashboard

```
library(shinydashboard)

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

Más y más templates y diseños

- shinydashboardPlus: https://rinterface.com/shiny/shinydashboardPlus/
- bs4Dash: https://rinterface.com/shiny/bs4Dash/classic/
- miniUI2Demo: https://dgranjon.shinyapps.io/miniUI2Demo
- tablerDash: https://rinterface.com/shiny/tablerDash/

Ejercicio: Aplicar temas

- Tomar la última app de y probar por al menos 2 temas (recomiendo *paper*) de `shinythemes`:
- Modifcar el ui utilizando el ejemplo de `shinydashboard` (copy+paste).

```
library(shiny)

ui ← fluidPage(
  sliderInput("valor", label = "Valor", min = 1, max = 10, value = 1),
  plotOutput("grafico")
  )

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

shinyApp(ui, server)
```

Publicar/Compartir tu app

Publicar/Compartir tu app

La forma más sencilla de compartir una app es:

- Crear una cuenta en https://www.shinyapps.io/ (puedes asociar tu correo gmail)
- Luego crear una app.
- `rsconnect::deployApp("<ruta_a_la_aplicacion>")`, o:

Ejercicio: Intentado usar todo

- Cree un proyecto en RStudio y una shinyapp.
- Descargue el archivo http://datos.gob.cl/dataset/28198 (puntos bip, si es que el link se rompe)
- Procese el archivo anterior para obtener un .rds o un csv más limpio.
- Ahora que su aplicación considere tener un selector de comunas, y que su aplicación retorne un mapa con las ubicaciones de los lugares de cargasbip de la comuna seleccionada usando como template el primer ejemplo de https://rstudio.github.io/leaflet/markers.html y https://rstudio.github.io/leaflet/shiny.html
- Subir una aplicación a shinyapps.io

Optimizando código

Optimizando código

Tomemos el ejemplo anterior de graficar y consideremos una opción para mostrar el eje en escala logaritmica

```
library(shiny)
library(forecast)
library(xts)
library(tradestatistics)
library(ggplot2)
library(scales)
library(plotly)
library(shinythemes)

formatear_monto ← function(monto){
    paste("$", comma(monto/1e6, accuracy = .01), "MM")
}

lista_paises ← setNames(ots_countries$country_iso, ots_countries$country_name_english)

ui ← fluidPage(
    theme = shinytheme("cyborg"),
    titlePanel("Ahora si que sí"),
    sidebarLayout(
        sidebarPanel(
            selectInput("pais", "Seleccionar un país:", choices = lista_paises, selected = "chl"),
            checkboxInput("log", label = "Escala en log")    #<<
        ),
        mainPanel(
          plotlyOutput("grafico")
        )
    )
)
```

Optimizando código (cont.)

```
server ← function(input, output) {
    output$grafico ← renderPlotly({
        pais ← input$pais
        data ← ots_create_tidy_data(years = 1990:2018, reporters = pais, table = "yr")
        valores ← data$export_value_usd
        fechas ← as.Date(paste0(data$year, "0101"), format = "%Y%m%d",)
        serie ← xts(valores, order.by = fechas)
        prediccion ← forecast(serie, h = 5)
        dfpred ← as.data.frame(prediccion)
        dfpred ← dfpred %>% mutate(anio = 2018 + 1:5)

        plt ← ggplot(data) +
          geom_line(aes(x = year, y = export_value_usd)) +
          geom_line(aes(x = anio, y = `Point Forecast`), data = dfpred, color = "darkred", size  = 1.2) +
          geom_ribbon(aes(x = anio, ymin = `Lo 95`, ymax = `Hi 95`), data = dfpred, alpha = 0.25) +\
          scale_y_continuous(labels = formatear_monto) +
          labs(x = "Año", y = NULL, title = pais, subtitle = "Acá va un subtitulo",
                caption = "Datos provenientes del paquete {tradestatistics}.")

        if(input$log){                    #<<
          plt ← plt + scale_y_log10()     #<<
        }

        ggplotly(plt)

    })

}

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

Expresiones reactivas (*reactive expressions*)

La idea de expresiones reactiva es que podemos limitar que es lo que se (re)ejecuta al cambiar un input.

Una expresión reactiva es código R que usa un widget/input y retorna un valor, la expresion se actualizará cuando el valor del (de los) widgets cambien.

Se crea una expresion con la función `reactive` la que toma una expresión/código R entre `{}`, de la misma forma que las funciones `render` (`renderPlot`, `renderTable`)

Así, por ejemplo ejemplo, para evitar correr código si solamente queremos cambiar elementos del gráfico una solución usando expresiones reactivas sería.

```
server ← function(input, output) {

    dataExport ← reactive({

      pais ← input$pais
      data ← ots_create_tidy_data(years = 1990:2018, reporters = pais, table = "yr")
      data

    })

    output$grafico ← renderPlotly({

        data ← dataExport()


    output
```

Expresiones reactivas (*reactive expressions*) (cont.)

Recordar que no necesariamente se puedes utilizar en **una** expresion, por ejemplo si queremos usar el data frame para realizar una tabla podríamos hacer:

```
server ← function(input, output) {

    dataExport ← reactive({

      pais ← input$pais
      data ← ots_create_tidy_data(years = 1990:2018, reporters = pais, table = "yr")
      data

    })

    output$tabla ← renderTable({ dataExport()  })

    output$grafico ← renderPlotly({

        data ← dataExport()


    output
```

Pueden revisar el código en:

- https://github.com/datosuc/Visualizacion-de-datos-con-R/blob/master/app-exportaciones/app.R
- https://github.com/datosuc/Visualizacion-de-datos-con-R/blob/master/app-exportaciones/app-reactive-expression.R

Memoización

(*Wikipedia*) En Informática, el término memoización (del inglés memoization) es una técnica de optimización que se usa principalmente para acelerar los tiempos de cálculo, almacenando los resultados de la llamada a una subrutina en una memoria intermedia o búfer y devolviendo esos mismos valores cuando se llame de nuevo a la subrutina o función con los mismos parámetros de entrada.

Supongamos la eterna función:

```
gran_funcion ← function(valor){
  Sys.sleep(sample(5:10, size = 1))
  2 * valor
}

gran_funcion(1)
```

```
## [1] 2
```

```
system.time({gran_funcion(1)})
```

```
##    user  system elapsed
##    0.02    0.00   10.03
```

Memoización (cont)

En R existe el paquete `memoise` el cual ayuda a *memoizar* funciones:

```
library(memoise)

gran_funcion_memoizada ← memoise(gran_funcion)

gran_funcion_memoizada(1)
```

```
## [1] 2
```

```
gran_funcion_memoizada(1) == gran_funcion(1)
```

```
## [1] TRUE
```

```
system.time({gran_funcion_memoizada(1)})
```

```
##    user  system elapsed
##       0       0       0
```

```
gran_funcion_memoizada
```

```
## Memoised Function:
## function(valor){
```

Memoización (cont 2)

En R existe el paquete `memoise` el cual ayuda a *memoizar* funciones:

Internamente lo que hace es:

- En la primera vez que se ejecuta la funcion memoizada debe utilizar la función original para conocer su valor.
- Luego guarda el valor del output (para una próxima ejecución de la función) asociándolo a los valores/parámetros.
- En una futura ocación al ejecutar la función memoizada con un parámetro ya ejecutado, en lugar de ejecutar la función original recupera anteriormente guardado.

Comentarios y cuidados:

- La función debe ser determinística, debe retornar siempre un valor fijo, ya sea un modelo, vector, etc (piense en memoizar `runif`).
- Existen funciones que la memoización dura cierto
- Recordar que esto no es exclusivo para shiny apps, ni para R.

Evaluación

Evaluación

Con el siguiente script:

```
if(!require(tidyverse)) install.packages("tidyverse")
if(!require(mindicador)) install.packages("mindicador")
if(!require(highcharter)) install.packages("highcharter")
if(!require(DT)) install.packages("DT")

library(tidyverse); library(lubridate); library(ggplot2)

d ← mindicador::mindicador_importar_datos("uf", anios = 2015:2020)

hchart(d, "line", hcaes(fecha, valor))

ggplot(d) +
  geom_line(aes(fecha, valor))

dres ← d %>%
  group_by(year(fecha)) %>%
  summarise(valor_mean = mean(valor))

d

DT::datatable(d)
```

Cree una shinyapp que:

- Le permita al usuario escoger que indicador a estudiar.
- La aplicación debe mostrar a travéz de un gráfico la variación del indicador respecto al tiempo.
- Además le debe permitir al usuario la posibilidad de escoger las fechas a visualizar.
- Mostrar una tabla con los datos utlizados para el gráficos.
- Utilizar algun paquete para cambiar el look de la aplicación.
- Finalmente utilice el servicio gratuiro de shinyapps.io para publicar la aplicación.

Importante/hints:

- No se pide ningún paquete en particular, pero el ir utilizando más paquetes *pomposos* otorgará puntos los que pueden ayudar a compensar puntos no terminados.
- Puede revisar la lista de indicadores con `mindicador::mindicador_indicadores$codigo`.
- recuerde que existe `dateInput` y `dateRangeInput`.

Enviar a más tardar el domingo 15 de noviembre el script y el link de la aplicación al correo **jbkunst@gmail.com** con el asunto **Evaluación shiny Sección B** Indicando en el cuerpo del mail los integrantes (a lo más 2).