semana 1

Luis Ambrocio

10/8/2021

Contents

shiny

¿Qué es Shiny?

- Shiny es un marco de aplicación web para R.
- Shiny te permite crear una interfaz gráfica para que los usuarios pueden interactuar con sus visualizaciones, modelos y algoritmos sin necesidad de conocer R por sí mismos.
- Usando Shiny, el momento de crear simples, pero poderosos, Los productos de datos interactivos basados en web en R se minimizan.
- Shiny está hecho por la buena gente de R Studio.

Algunos requisitos previos leves

Shiny realmente no lo requiere, pero como con todas las web programación, un poco de conocimiento de HTML, CSS y Javascript es muy útil.

- HTML proporciona una estructura y secciones de la página web, así como instrucciones de marcado
- CSS da el estilo

- Javscript para interactividad

Shiny usa Bootstrap(no relación con el estilo bootstrap de estadísticas), que (para mí) parece verse bien y se reproduce bien en plataformas móviles.

Empezando

- Asegúrese de tener instalada la última versión de R
- Si está en Windows, asegúrese de tener Rtools instalado
- `install.packages ("shiny")`
- `biblioteca (shiny)`
- Gran tutorial en http://shiny.rstudio.com/tutorial/
- Básicamente, esta conferencia es un recorrido por ese tutorial. ofreciendo algunas de nuestras ideas

archivos

Un proyecto shiny es un directorio que contiene al menos dos archivos:

- `ui.R` (para la interfaz de usuario) controla el aspecto de su aplicación.
- `server.R` que controla lo que hace tu aplicación.

ejemplo 1

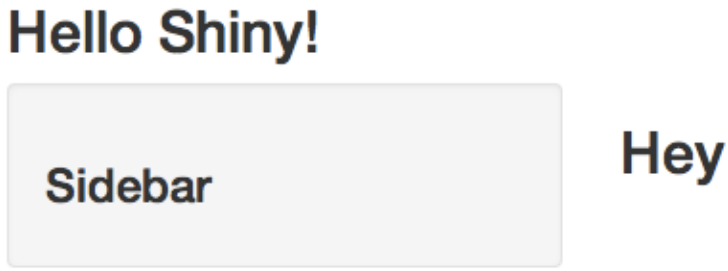
ui.R

```
library(shiny)
shinyUI(fluidPage(
  titlePanel("Data science FTW!"),
  sidebarLayout(
    sidebarPanel(
      h3("Sidebar Text")
    ),
    mainPanel(
      h3("Main Panel Text")
    )
  )
))
```

server.R

```
library(shiny)
shinyServer(function(input, output) {

})
```

Para ejecutarlo - En R, cambie a los directorios con estos archivos y escriba `runApp ()` - o poner la ruta al directorio como argumento - Debería abrir una ventana del navegador con la aplicación en ejecución.



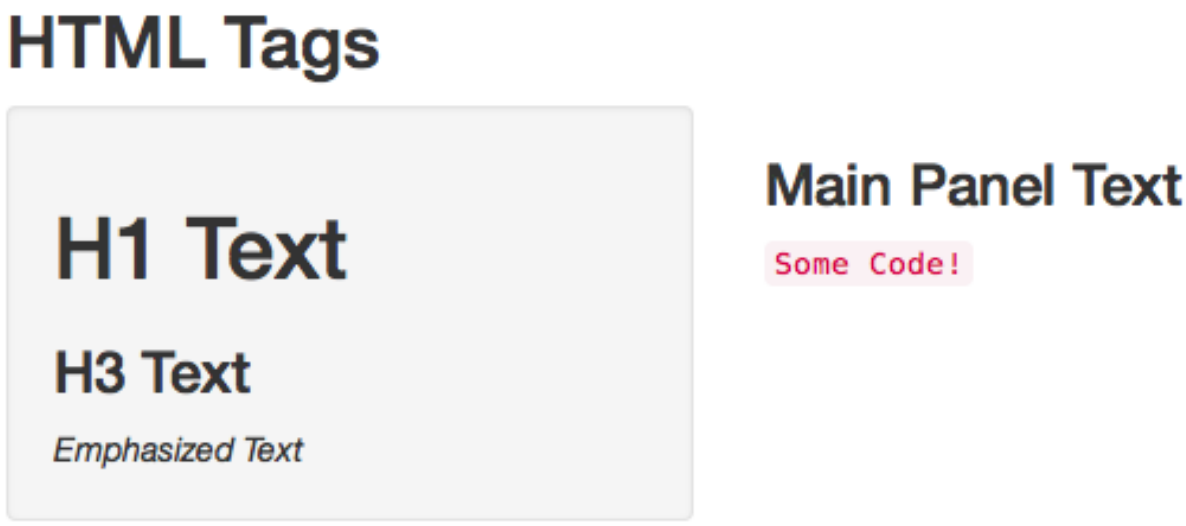
ejemplo 2: app con HTML

Shiny proporciona varias funciones de envoltura para usar etiquetas HTML estándar en su `ui.R`, incluyendo`h1()`a `h6()`, `p ()`, `a ()`, `div()` y `span()`.

- Consulte "?builder" para obtener más detalles.

Ui.R

```
library(shiny)
shinyUI(fluidPage(
  titlePanel("HTML Tags"),
  sidebarLayout(
    sidebarPanel(
      h1("H1 Text"),
      h3("H3 Text"),
      em("Emphasized Text")
    ),
    mainPanel(
      h3("Main Panel Text"),
      code("Some Code!")
    )
  )
))
```



ejemplo 3: Aplicación con entradas y salidas

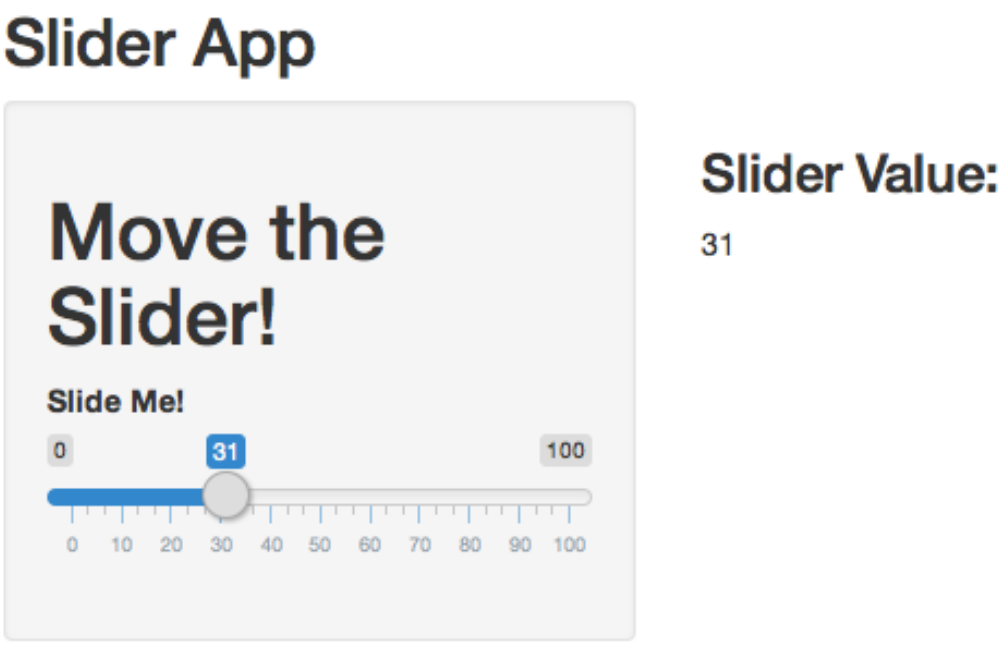
Ahora que ha jugado con la personalización del aspecto de su aplicación, ¡vamos a darle algunas funciones! Shiny proporciona varios tipos de entradas, incluidos botones, casillas de verificación, texto cajas y calendarios. Primero experimentemos con el control deslizante. aporte. Esta sencilla aplicación mostrará el número que es seleccionado con un control deslizante.

ui.R

```
library(shiny)
shinyUI(fluidPage(
  titlePanel("Slider App"),
  sidebarLayout(
    sidebarPanel(
      h1("Move the Slider!"),
      sliderInput("slider1", "Slide Me!", 0, 100, 0)
    ),
    mainPanel(
      h3("Slider Value:"),
      textOutput("text")
    )
  )
))
```

server.R

```
library(shiny)
shinyServer(function(input, output) {
  output$text <- renderText(input$slider1)
})
```



- `sliderInput ()` especifica un control deslizante que un usuario puede manipular
- `testOutput ()` muestra el texto que se representa en `server.R`
- `renderText ()` transforma la entrada de la interfaz de usuario en texto que se puede mostrar.

Observe que en `ui.R` se asignan nombres de entrada y salida con cadenas (`sliderInput (" slider1 ", ..., textOutput (" texto "))` y en `server.R` se puede acceder a esos componentes con el Operador `$`: `salida $ texto <- renderText (entrada $ slider1)`.

ejemplo 4:Aplicaciones con gráficos

Permitir a los usuarios manipular datos y ver los resultados de sus manipulaciones como trama pueden resultar muy útiles. shiny proporciona la función `plotOutput()` para `ui.R` y la Función `renderPlot()` para `sever.R` para tomar la entrada del usuario y creando parcelas.

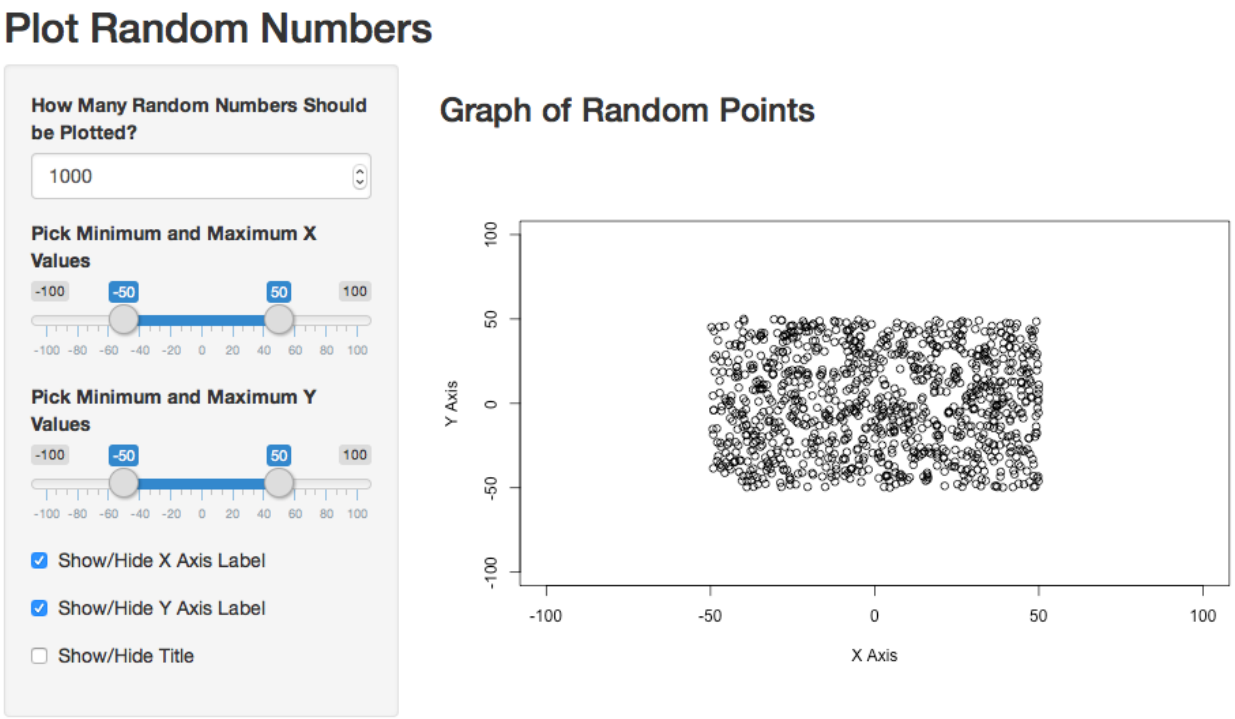
ui.R

```
library(shiny)
shinyUI(fluidPage(
  titlePanel("Plot Random Numbers"),
  sidebarLayout(
    sidebarPanel(
      numericInput("numeric", "How Many Random Numbers Should be Plotted?",
                   value = 1000, min = 1, max = 1000, step = 1),
      sliderInput("sliderX", "Pick Minimum and Maximum X Values",
                  -100, 100, value = c(-50, 50)),
      checkboxInput("show_xlab", "Show/Hide X Axis Label", value = TRUE),
      checkboxInput("show_ylab", "Show/Hide Y Axis Label", value = TRUE),
      checkboxInput("show_title", "Show/Hide Title")
    ),
    mainPanel(
      h3("Graph of Random Points"),
      plotOutput("plot1")
    )
```

```
  )
))
```

server.R

```
library(shiny)
shinyServer(function(input, output) {
  output$plot1 <- renderPlot({
    set.seed(2016-05-25)
    number_of_points <- input$numeric
    minX <- input$sliderX[1]
    maxX <- input$sliderX[2]
    minY <- input$sliderY[1]
    maxY <- input$sliderY[2]
    dataX <- runif(number_of_points, minX, maxX)
    dataY <- runif(number_of_points, minY, maxY)
    xlab <- ifelse(input$show_xlab, "X Axis", "")
    ylab <- ifelse(input$show_ylab, "Y Axis", "")
    main <- ifelse(input$show_title, "Title", "")
    plot(dataX, dataY, xlab = xlab, ylab = ylab, main = main,
         xlim = c(-100, 100), ylim = c(-100, 100))
  })
})
```



ui.R

- `numericInput ()` permite al usuario ingresar cualquier número
- `checkboxInput ()` crea casillas que se pueden marcar
- `plotOutput ()` muestra un gráfico

servidor.R

- `renderPlot ()` envuelve la creación de un gráfico para que se pueda mostrar

ejemplo 5: reactividad

Una expresión reactiva es como una receta que manipula las entradas de Shiny y luego devuelve un valor. La reactividad proporciona una forma para que su aplicación responda, ya que las entradas cambiarán dependiendo de cómo los usuarios interactúen con su interfaz de usuario. Las expresiones envueltas por `reactive ()` deben ser expresiones sujetas a cambios.

Es como crear una funcion:

```
calc_sum <- reactive({
  input$box1 + input$box2
})

# ...

calc_sum()
```

Esta aplicación predice la potencia de un automóvil dada la eficiencia de combustible en millas por galón para el automóvil.

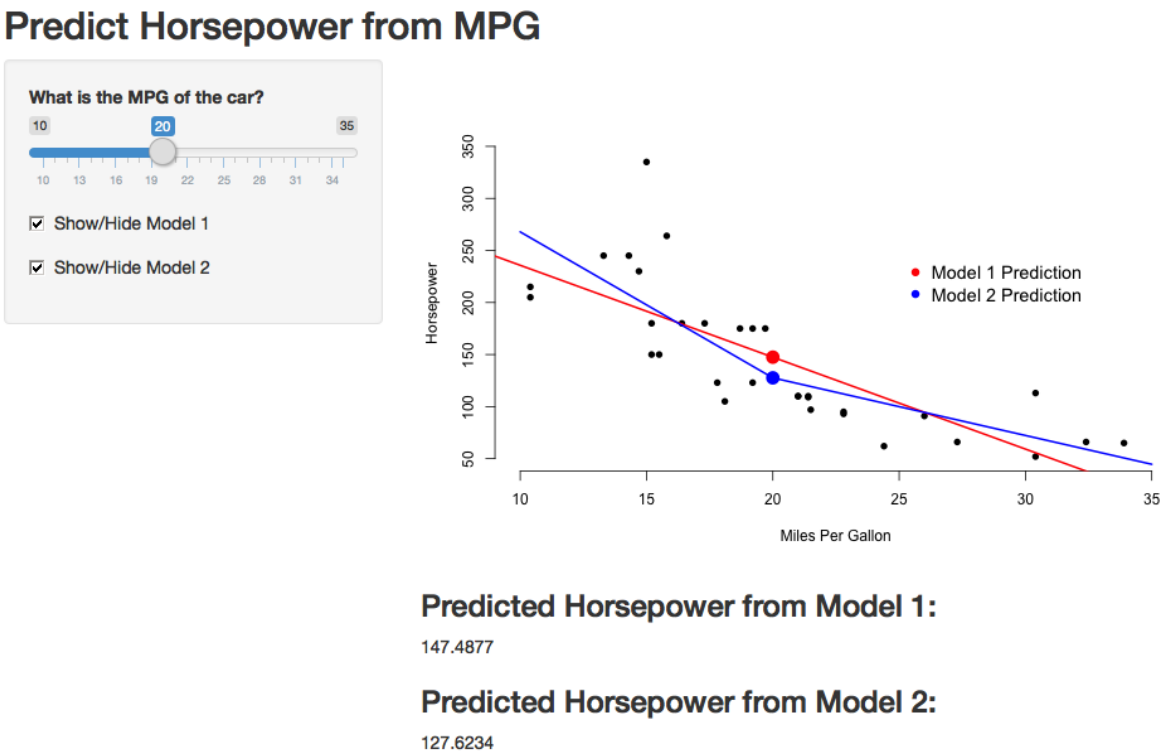
ui.R

```
library(shiny)
shinyUI(fluidPage(
  titlePanel("Predict Horsepower from MPG"),
  sidebarLayout(
    sidebarPanel(
      sliderInput("sliderMPG", "What is the MPG of the car?", 10, 35, value = 20),
      checkboxInput("showModel1", "Show/Hide Model 1", value = TRUE),
      checkboxInput("showModel2", "Show/Hide Model 2", value = TRUE)
    ),

    mainPanel(
      plotOutput("plot1"),
      h3("Predicted Horsepower from Model 1:"),
      textOutput("pred1"),
      h3("Predicted Horsepower from Model 2:"),
      textOutput("pred2")
    )
  )
))
```

server.R

```
library(shiny)
shinyServer(function(input, output) {
  mtcars$mpgsp <- ifelse(mtcars$mpg - 20 > 0, mtcars$mpg - 20, 0)
  model1 <- lm(hp ~ mpg, data = mtcars)
  model2 <- lm(hp ~ mpgsp + mpg, data = mtcars)

  model1pred <- reactive({
    mpgInput <- input$sliderMPG
    predict(model1, newdata = data.frame(mpg = mpgInput))
  })
```

```
  model2pred <- reactive({
    mpgInput <- input$sliderMPG
    predict(model2, newdata =
              data.frame(mpg = mpgInput,
                         mpgsp = ifelse(mpgInput - 20 > 0,
                                        mpgInput - 20, 0)))
  })

  output$plot1 <- renderPlot({
    mpgInput <- input$sliderMPG

    plot(mtcars$mpg, mtcars$hp, xlab = "Miles Per Gallon",
         ylab = "Horsepower", bty = "n", pch = 16,
         xlim = c(10, 35), ylim = c(50, 350))
    if(input$showModel1){
      abline(model1, col = "red", lwd = 2)
    }
    if(input$showModel2){
      model2lines <- predict(model2, newdata = data.frame(
        mpg = 10:35, mpgsp = ifelse(10:35 - 20 > 0, 10:35 - 20, 0)
      ))
      lines(10:35, model2lines, col = "blue", lwd = 2)
    }

  legend(25, 250, c("Model 1 Prediction", "Model 2 Prediction"), pch = 16,
           col = c("red", "blue"), bty = "n", cex = 1.2)
    points(mpgInput, model1pred(), col = "red", pch = 16, cex = 2)
    points(mpgInput, model2pred(), col = "blue", pch = 16, cex = 2)
  })

  output$pred1 <- renderText({
    model1pred()
  })

  output$pred2 <- renderText({
    model2pred()
  })
})
```

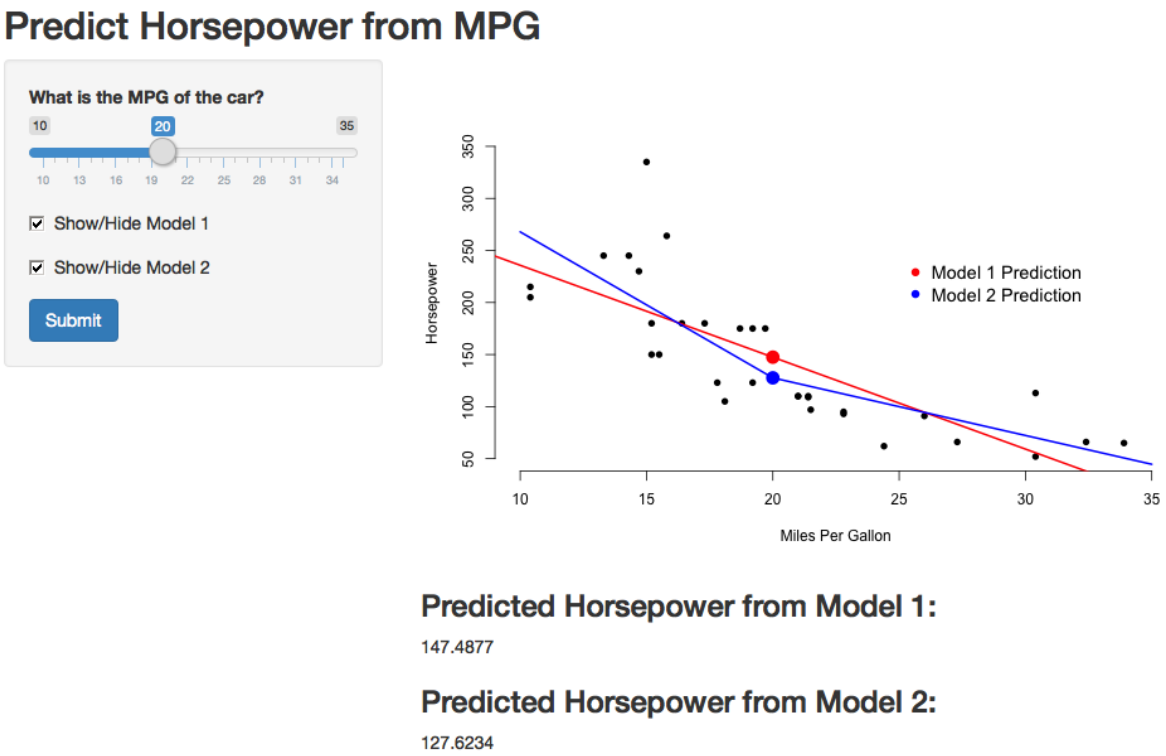


ejemplo 6: reactividad retardada

Es posible que no desee que su aplicación reaccione de inmediato a los cambios en la entrada del usuario debido a algo como un cálculo de larga duración. Para evitar que las expresiones reactivas reaccionen, puede usar un botón de envío en su aplicación. Echemos un vistazo a la última aplicación que creamos, pero con un botón de envío agregado a la aplicación.

ui.R

```
shinyUI(fluidPage(
  titlePanel("Predict Horsepower from MPG"),
  sidebarLayout(
    sidebarPanel(
      sliderInput("sliderMPG", "What is the MPG of the car?", 10, 35, value = 20),
      checkboxInput("showModel1", "Show/Hide Model 1", value = TRUE),
      checkboxInput("showModel2", "Show/Hide Model 2", value = TRUE),
      submitButton("Submit") # New!
    ),
```



ejemplo 7: tabs

Hay varios otros tipos de componentes de la interfaz de usuario que puede mezclar en su aplicación, incluidas pestañas, barras de navegación y barras laterales. Le mostraremos un ejemplo de cómo usar las pestañas para darle a su aplicación múltiples vistas. La función `tabsetPanel ()` especifica un grupo de pestañas, y luego la función `tabPanel ()` especifica el contenido de una pestaña individual.

ui.R

```
library(shiny)
shinyUI(fluidPage(
  titlePanel("Tabs!"),
  sidebarLayout(
    sidebarPanel(
      textInput("box1", "Enter Tab 1 Text:", value = "Tab 1!"),
      textInput("box2", "Enter Tab 2 Text:", value = "Tab 2!"),
      textInput("box3", "Enter Tab 3 Text:", value = "Tab 3!")
    ),
    mainPanel(
      tabsetPanel(type = "tabs",
                  tabPanel("Tab 1", br(), textOutput("out1")),
                  tabPanel("Tab 2", br(), textOutput("out2")),
                  tabPanel("Tab 2", br(), textOutput("out3"))
      )
```

```
    )
  )
))
```

server.R

```
library(shiny)
shinyServer(function(input, output) {
  output$out1 <- renderText(input$box1)
  output$out2 <- renderText(input$box2)
  output$out3 <- renderText(input$box3)
})
```

ejemplo 8: Graficos interactivos

Una de mis características favoritas de Shiny es la capacidad de crear gráficos con los que el usuario puede interactuar. Un método que puede usar para seleccionar múltiples puntos de datos en un gráfico es especificando el argumento `brush` en`plotOutput ()`en el lado`ui.R`, y luego usando la función `brushedPoints ()` en el lado `server.R`. La siguiente aplicación de ejemplo se ajusta a un modelo lineal para los puntos seleccionados y luego dibuja una línea de mejor ajuste para el modelo resultante.

ui.R

```
library(shiny)
shinyUI(fluidPage(
  titlePanel("Visualize Many Models"),
  sidebarLayout(
    sidebarPanel(
      h3("Slope"),
      textOutput("slopeOut"),
      h3("Intercept"),
      textOutput("intOut")
    ),
    mainPanel(
      plotOutput("plot1", brush = brushOpts(
        id = "brush1"
      ))
    )
  )
))
```

server.R

```
library(shiny)
shinyServer(function(input, output) {
  model <- reactive({
    brushed_data <- brushedPoints(trees, input$brush1,
                                  xvar = "Girth", yvar = "Volume")
    if(nrow(brushed_data) < 2){
      return(NULL)
    }
    lm(Volume ~ Girth, data = brushed_data)
  })
  output$slopeOut <- renderText({
    if(is.null(model())){
      "No Model Found"
    } else {
      model()[[1]][2]
    }
  })

output$intOut <- renderText({
    if(is.null(model())){
      "No Model Found"
    } else {
      model()[[1]][1]
    }
  })
  output$plot1 <- renderPlot({
    plot(trees$Girth, trees$Volume, xlab = "Girth",
         ylab = "Volume", main = "Tree Measurements",
         cex = 1.5, pch = 16, bty = "n")
    if(!is.null(model())){
      abline(model(), col = "blue", lwd = 2)
    }
  })
})
```

Compartir aplicaciones brillantes

Una vez que haya creado una aplicación Shiny, hay varias formas de compartir su aplicación. El uso de shinyapps.io permite a los usuarios interactuar con su aplicación a través de un navegador web sin necesidad de tener R o Shiny instalados. Shinyapps.io tiene un nivel gratuito, pero si desea utilizar una aplicación Shiny en su negocio, es posible que le interese pagar por el servicio. Si está compartiendo su aplicación con un usuario de R, puede publicar su aplicación en GitHub e indicarle al usuario que use la función `runGist ()` o `runGitHub ()` para iniciar su aplicación.

Más recursos brillantes

- El tutorial oficial de Shiny
- Galería de aplicaciones brillantes
- Muéstrame Shiny: Galería de aplicaciones web R
- Integración de Shiny y Plotly
- Shiny on Stack Overflow
- OpenCPU de Jerome Ooms, es un proyecto realmente genial que proporciona una API para llamar a R desde documentos web

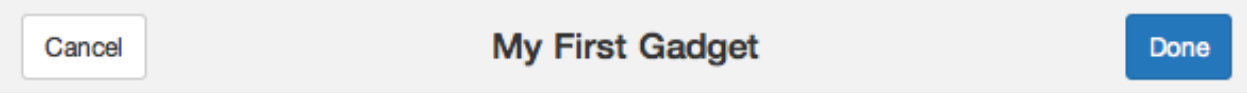
shiny gadgets

Shiny Gadgets proporciona una forma de utilizar la interactividad de Shiny y las herramientas de interfaz de usuario como parte de un análisis de datos. Con Shiny Gadgets puedes crear una función que abre una pequeña aplicación Shiny. Dado que estas aplicaciones son más pequeñas, usaremos el paquete miniUI para crear interfaces de usuario.

En esencia, un Shiny Gadget es una función que lanza una pequeña aplicación Shiny (de una sola página). Dado que Shiny Gadgets está destinado a mostrarse en el panel del visor de RStudio, el paquete miniUI es útil para sus elementos gráficos más pequeños. Construyamos un Shiny Gadget muy simple.

ejemplo 1:

```
library(shiny)
library(miniUI)

myFirstGadget <- function() {
  ui <- miniPage(
    gadgetTitleBar("My First Gadget")
  )
  server <- function(input, output, session) {
    # The Done button closes the app
    observeEvent(input$done, {
      stopApp()
    })
  }
  runGadget(ui, server)
}
```



Obtenga el código anterior y ejecute `myFirstGadget ()` para ver un Shiny Gadget muy básico. Solo para revisar algunos de los nuevos funciones en este gadget:

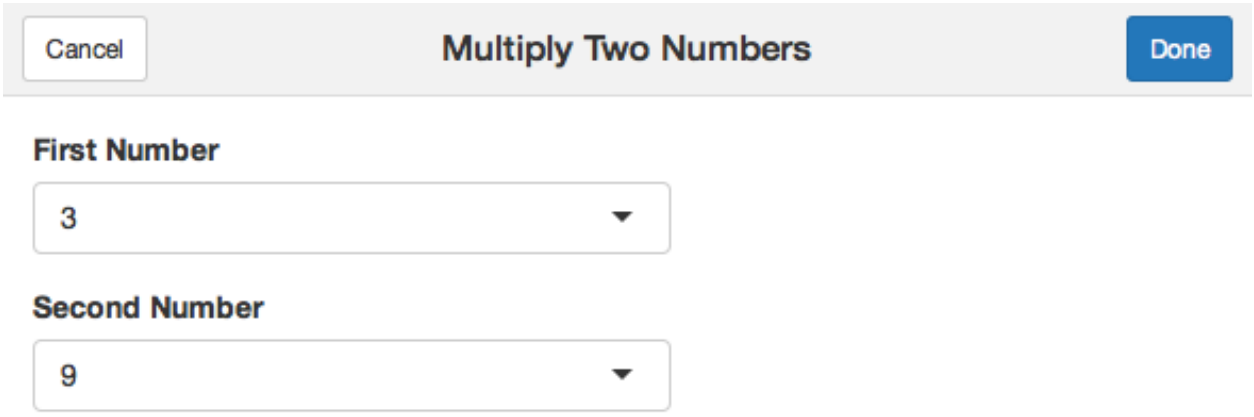
- `miniPage ()` crea un pequeño diseño.
- `gadgetTitleBar ()` proporciona una barra de título con Listo y Cancelar botones.
- `runGadget ()` ejecuta Shiny Gadget, tomando `ui` y `servidor` como argumentos.

ejemplo 2: gadgets con argumentos

Una de las ventajas de Shiny Gadgets es que, dado que Shiny Gadgets son funciones, pueden tomar valores como argumentos y pueden devolver valores. Echemos un vistazo a un ejemplo simple de un gadget brillante que toma argumentos y devuelve un valor. El siguiente Shiny Gadget toma dos vectores diferentes de números como argumentos y usa esos vectores para completar dos `selectInputs`. A continuación, el usuario puede elegir dos números dentro del gadget y se devolverá el producto de esos dos números.

```
library(shiny)
library(miniUI)

multiplyNumbers <- function(numbers1, numbers2) {
  ui <- miniPage(
    gadgetTitleBar("Multiply Two Numbers"),
    miniContentPanel(
      selectInput("num1", "First Number", choices=numbers1),
      selectInput("num2", "Second Number", choices=numbers2)
    )
  )
```

```
  server <- function(input, output, session) {
    observeEvent(input$done, {
      num1 <- as.numeric(input$num1)
      num2 <- as.numeric(input$num2)
      stopApp(num1 * num2)
    })
  }
  runGadget(ui, server)
}
```



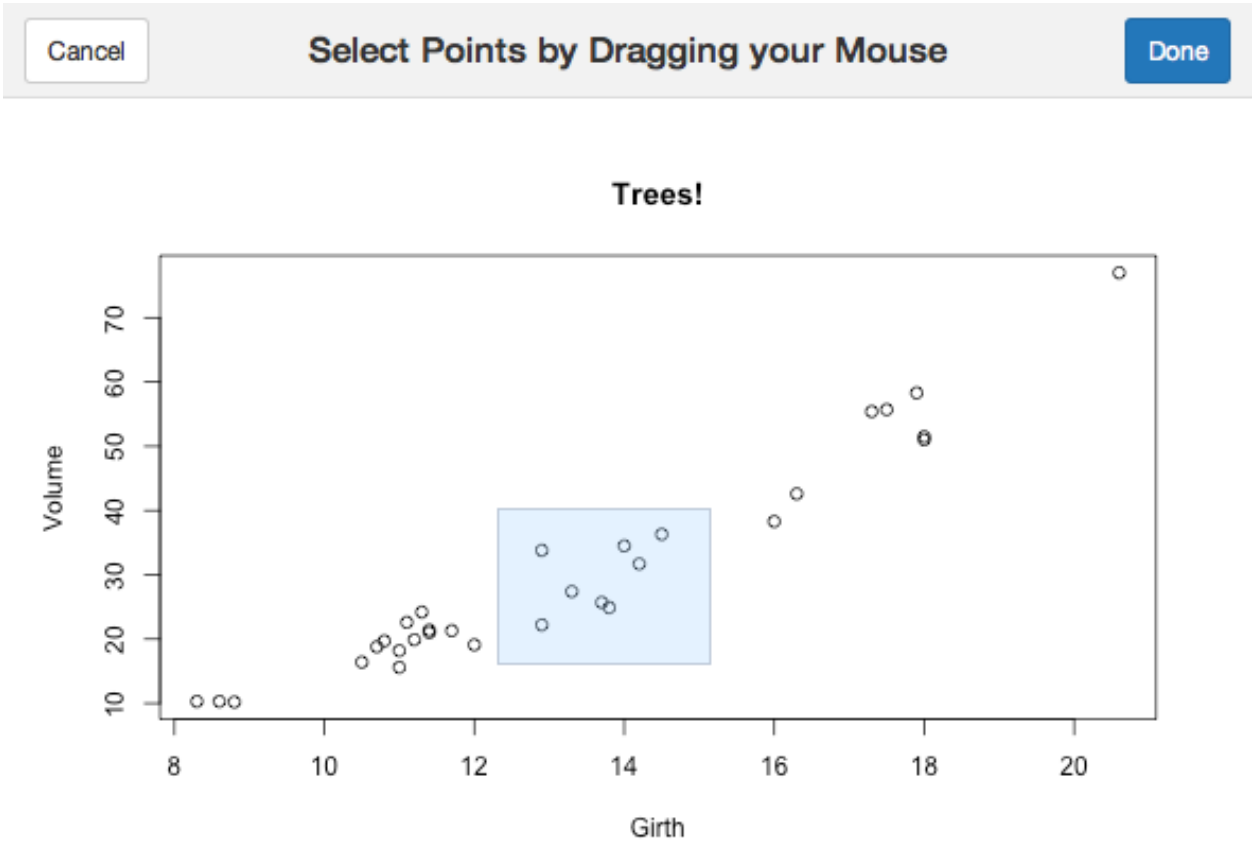
Obtenga el código anterior y ejecute `multiplyNumbers (1: 5, 6:10)` para que puedas hacerte una idea de cómo funciona este gadget. Como puede ver, este Gadget usa `selectInput()` para que el usuario pueda seleccionar dos números diferentes. Al hacer clic en el botón Listo, se multiplican los números de remolque, que se devuelven como resultado de la función `multiplyNumbers()`.

ejemplo 3: gadgets con graficos interactivos

Los Shiny Gadgets son particularmente útiles cuando un análisis de datos necesita un toque de intervención humana. Puede imaginar la presentación de una visualización de datos interactiva a través de un gadget, donde un analista podría manipular datos en el gadget, y luego el gadget devolvería los datos manipulados. Analicemos un ejemplo de cómo se podría hacer esto.

```
library(shiny)
library(miniUI)
```

```
pickTrees <- function() {
  ui <- miniPage(
    gadgetTitleBar("Select Points by Dragging your Mouse"),
    miniContentPanel(
      plotOutput("plot", height = "100%", brush = "brush")
    )
  )

  server <- function(input, output, session) {
    output$plot <- renderPlot({
      plot(trees$Girth, trees$Volume, main = "Trees!",
        xlab = "Girth", ylab = "Volume")
    })
    observeEvent(input$done, {
      stopApp(brushedPoints(trees, input$brush,
        xvar = "Girth", yvar = "Volume"))
    })
  }

  runGadget(ui, server)
}
```



Obtenga el código anterior y ejecute `pickTrees()`. Haga clic y arrastre un cuadro sobre el gráfico. Una vez

que haya dibujado un cuadro con el que esté satisfecho, haga clic en el botón Listo y los puntos que seleccionó se le devolverán como un marco de datos. Esta funcionalidad es posible gracias a la función `brushedPoint ()`, que es parte del paquete `shiny`. (Consulte `?Shiny :: brushedPoints` para obtener más información).

Conclusión

Para obtener más detalles sobre Shiny Gadgets, visite el sitio web de Shiny Gadgets:

- http://shiny.rstudio.com/articles/gadgets.html
- http://shiny.rstudio.com/articles/gadget-ui.html

GoogleVis

Idea básica

- La función R crea una página HTML
- La página HTML llama a Google Charts
- El resultado es un gráfico HTML interactivo

ejemplo

```
suppressPackageStartupMessages(library(googleVis))
M <- gvisMotionChart(Fruits, "Fruit", "Year",
                     options=list(width=600, height=400))
# plot(M)
```

Gráficos en googleVis

"gvis + ChartType"

- Gráficos dinámicos: `gvisMotionChart`
- Mapas interactivos: `gvisGeoChart`
- Tablas interactivas: `gvisTable`
- Gráficos de líneas: `gvisLineChart`
- Gráficos de barras: `gvisColumnChart`
- Mapas de árboles: `gvisTreeMap`

http://cran.r-project.org/web/packages/googleVis/googleVis.pdf

Mapas interactivos

```
G <- gvisGeoChart(Exports, locationvar="Country",
                  colorvar="Profit",options=list(width=600, height=400))
print(G,"chart")
```

```
# plot(G)
```

Grafico: http://127.0.0.1:27434/custom/googleVis/GeoChartID46c4fa0903.html

especificando region

```
G2 <- gvisGeoChart(Exports, locationvar="Country",
                  colorvar="Profit",options=list(width=600, height=400,region="150"))
 print(G2,"chart")
```

```
# plot(G2)
```

Grafico: http://127.0.0.1:27434/custom/googleVis/GeoChartID46c4bc436cf.html

Encontrar parámetros para establecer en `opciones`

https://developers.google.com/chart/interactive/docs/gallery/geochart

estableciendo mas opciones

```
df <- data.frame(label=c("US", "GB", "BR"), val1=c(1,3,4), val2=c(23,12,32))
Line <- gvisLineChart(df, xvar="label", yvar=c("val1","val2"),
        options=list(title="Hello World", legend="bottom",
                titleTextStyle="{color:'red', fontSize:18}",
                vAxis="{gridlines:{color:'red', count:3}}",
                hAxis="{title:'My Label', titleTextStyle:{color:'blue'}}",
                series="[{color:'green', targetAxisIndex: 0},
                         {color: 'blue',targetAxisIndex:1}]",
                vAxes="[{title:'Value 1 (%)', format:'##,######%'},
                                {title:'Value 2 (\U00A3)'}]",
                curveType="function", width=500, height=300
                ))

print(Line,"chart")
```

Grafico: http://127.0.0.1:27434/custom/googleVis/LineChartID46c440395ee8.html

Combinando multiples graficas juntas

```
G <- gvisGeoChart(Exports, "Country", "Profit",options=list(width=200, height=100))
T1 <- gvisTable(Exports,options=list(width=200, height=270))
M <- gvisMotionChart(Fruits, "Fruit", "Year", options=list(width=400, height=370))
GT <- gvisMerge(G,T1, horizontal=FALSE)
GTM <- gvisMerge(GT, M, horizontal=TRUE,tableOptions="bgcolor=\"#CCCCCC\" cellspacing=10")

# plot(GTM)
```

Grafico: http://127.0.0.1:27434/custom/googleVis/MergedID46c4f45265b.html

Viendo el codigo HTML

```
M <- gvisMotionChart(Fruits, "Fruit", "Year", options=list(width=600, height=400))
# print(M)
print(M, 'chart', file='myfilename.html')
```

Cosas que puede hacer con Google Vis

- Las visualizaciones se pueden incrustar en sitios web con código HTML
- Las visualizaciones dinámicas se pueden construir con Shiny, Rook y R.rsp
- Incrustarlos en documentos basados en R markdown
  - Establecer `results =" asis "` en las opciones del fragmento
  - Se puede usar con knitr y slidify

Para mas informacion

```
demo(googleVis)
```

- http://cran.r-project.org/web/packages/googleVis/vignettes/googleVis.pdf
- http://cran.r-project.org/web/packages/googleVis/googleVis.pdf
- https://developers.google.com/chart/interactive/docs/gallery
- https://developers.google.com/chart/interactive/faq
- https://github.com/mages/Introduction_to_googleVis/blob/gh-pages/index.Rmd

Plotly

¿Qué es Plotly?

Plotly es una aplicación web para crear y compartir visualizaciones de datos. Plotly puede trabajar con varios lenguajes de programación y aplicaciones, incluidos R, Python y Microsoft Excel. Nos concentraremos en crear diferentes gráficos con Plotly y compartir esos gráficos.

en https://plot.ly/ puedes compratir tus graficos, solo tienes que crear una cuenta

Diagrama de dispersión básico

Un diagrama de dispersión básico es fácil de hacer, con el beneficio adicional de la información sobre herramientas que aparece cuando el mouse pasa por encima de cada punto. Especifique un diagrama de dispersión indicando `type =" scatter "`. Observe que los argumentos para las variables `x` e`y` se especifican como fórmulas, con el operador de tilde (`~`) precediendo a la variable que está trazando.

```
library(plotly)
plot_ly(mtcars, x = ~wt, y = ~mpg, type = "scatter")
```