

Shiny Apps

Juan G. Rubalcaba

Shiny apps

https://jrubalcaba.github.io/posts/shiny_seminar/

Shiny apps seminar



Presentation

[Presentation PDF](#)

R Codes

- 1 - Making a scatterplot with Shiny - [1_scatterplot.R](#)
- 2 - Scatterplot editor - [2_scatterplot_edit.R](#)
- 3 - Adding more widgets - [3_scatterplot_widgets.R](#)
- 4 - Playing with model - parameters [4_nonlinear_model.R](#)
- 5 - Professional-looking apps with Shinydashboard - [5_shinydashboard.R](#)
- 6 - Using observers - [6_Observe_event.R](#)
- 7 - Using observers to fit nonlinear least squares - [7_Fitting_nls.R](#)

Resources

Introductory tutorial - <https://shiny.rstudio.com/tutorial/>

COVID19 Tracker - <https://shiny.rstudio.com/gallery/covid19-tracker.html>

NicheMapR app - [Kearney et al. \(2020, MEE\)](#)

Shiny apps

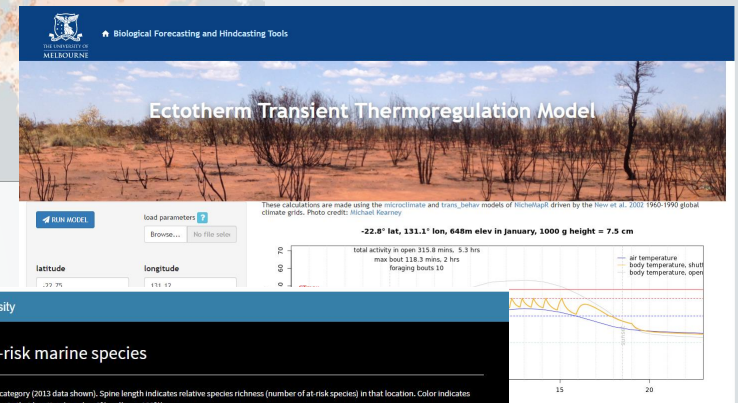
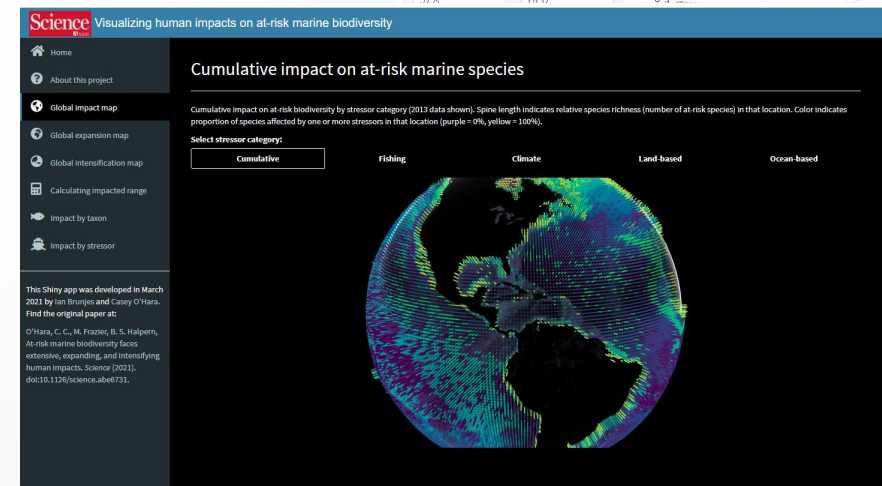
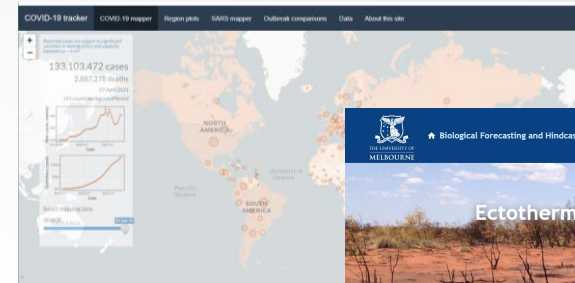
What is a Shiny app? **Gallery**

How to make shiny apps (**User Interface**)

How do they work? (**Server**)

Interactive **plots and maps**

Launching Shiny apps



Shiny apps

Interactive **web applications** for R

Shiny apps can do **whatever R can do**

- manage datasets / perform statistics
- data visualization (graphs, maps...)
- simulations



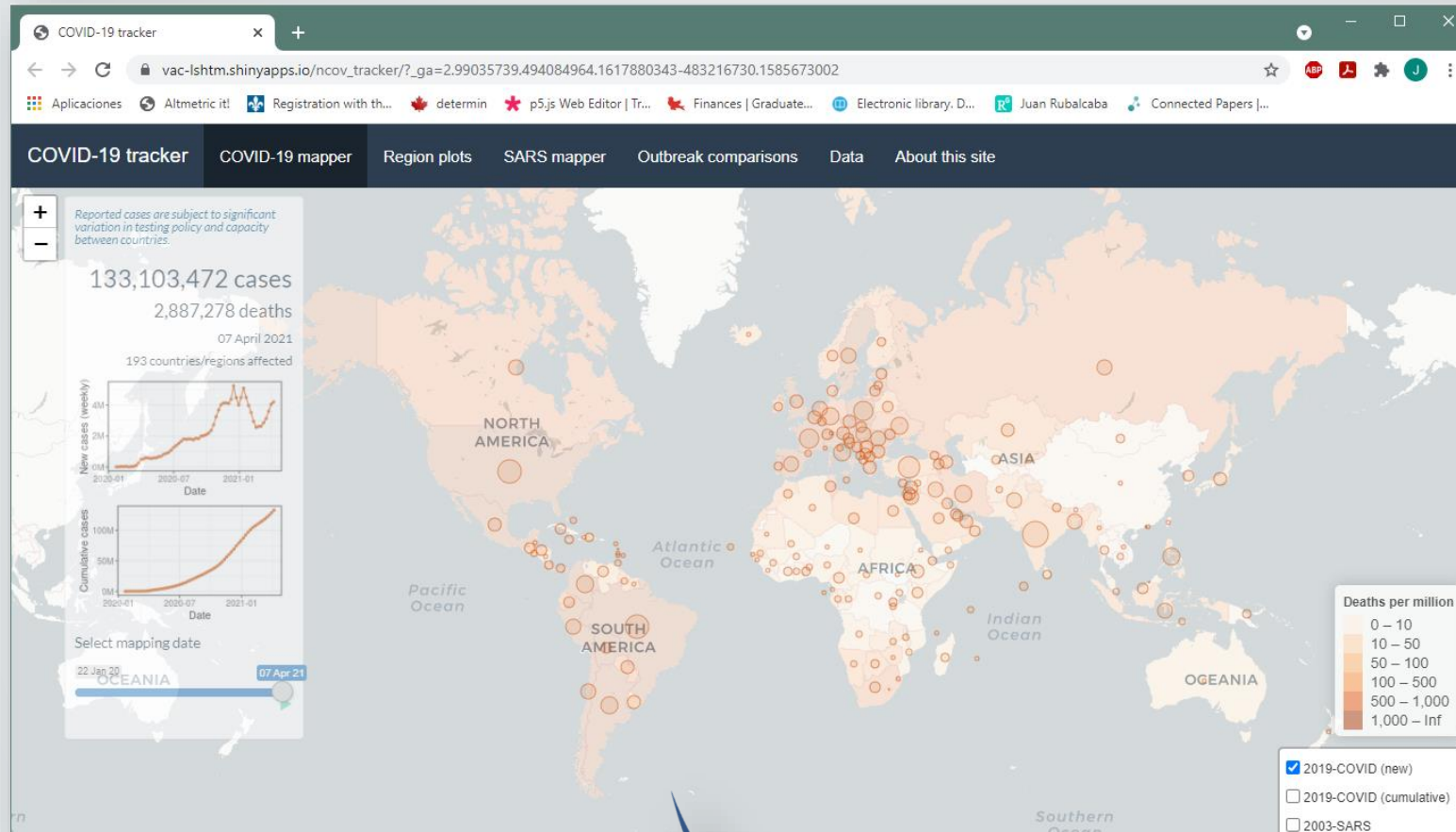
Interactively!

Extensions with HTML, CSS, JavaScript



```
dens <- density(data, n = npts)
dx <- dens$x
dy <- dens$y
if(add == TRUE)
  plot(0., 0., main = "Density Plot",
       xlab = "x", ylab = "Density",
       if(orientation == "v")
         dx2 <- (dx - min(dx)) / (max(dx) - min(dx))
         x[1.]
         dy2 <- (dy - min(dy)) / (max(dy) - min(dy))
         y[1.]
         seqbelow <- rep(y[1.], length(dx))
         if(Fill == T)
           confshade(dx2, seqbelow, dy2)
```

Shiny apps



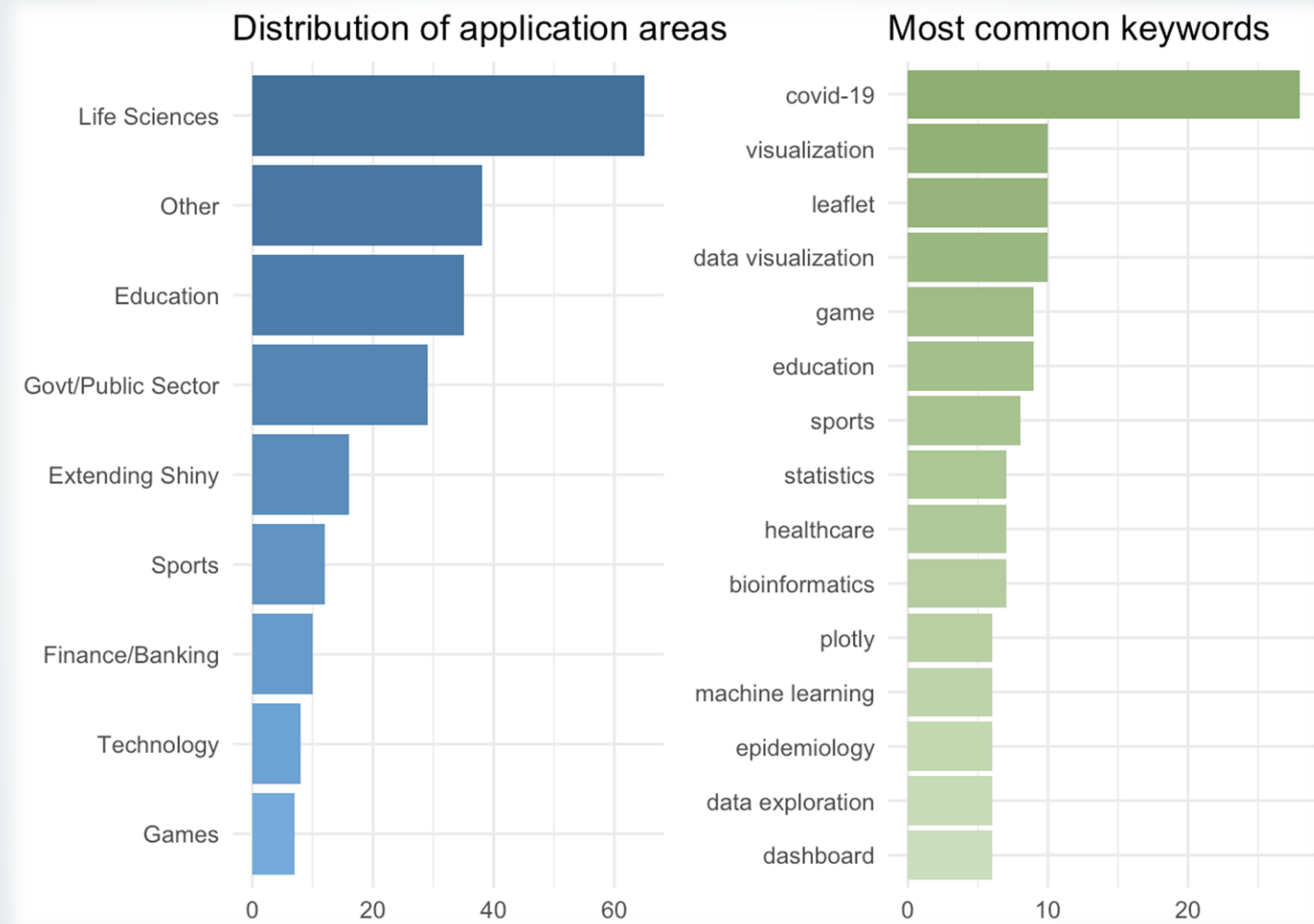
Shiny

```
..., n = npts)

main
= (ys)
in (x) ax(dx)
in (y) ax(dy)

seqbelow <- rep(y[1.], length(dx))
if(Fill == T)
  confshade(dx2, seqbelow, dy2
```

Shiny apps



Shiny examples

Global **data visualization**

Run **simulations** and download data

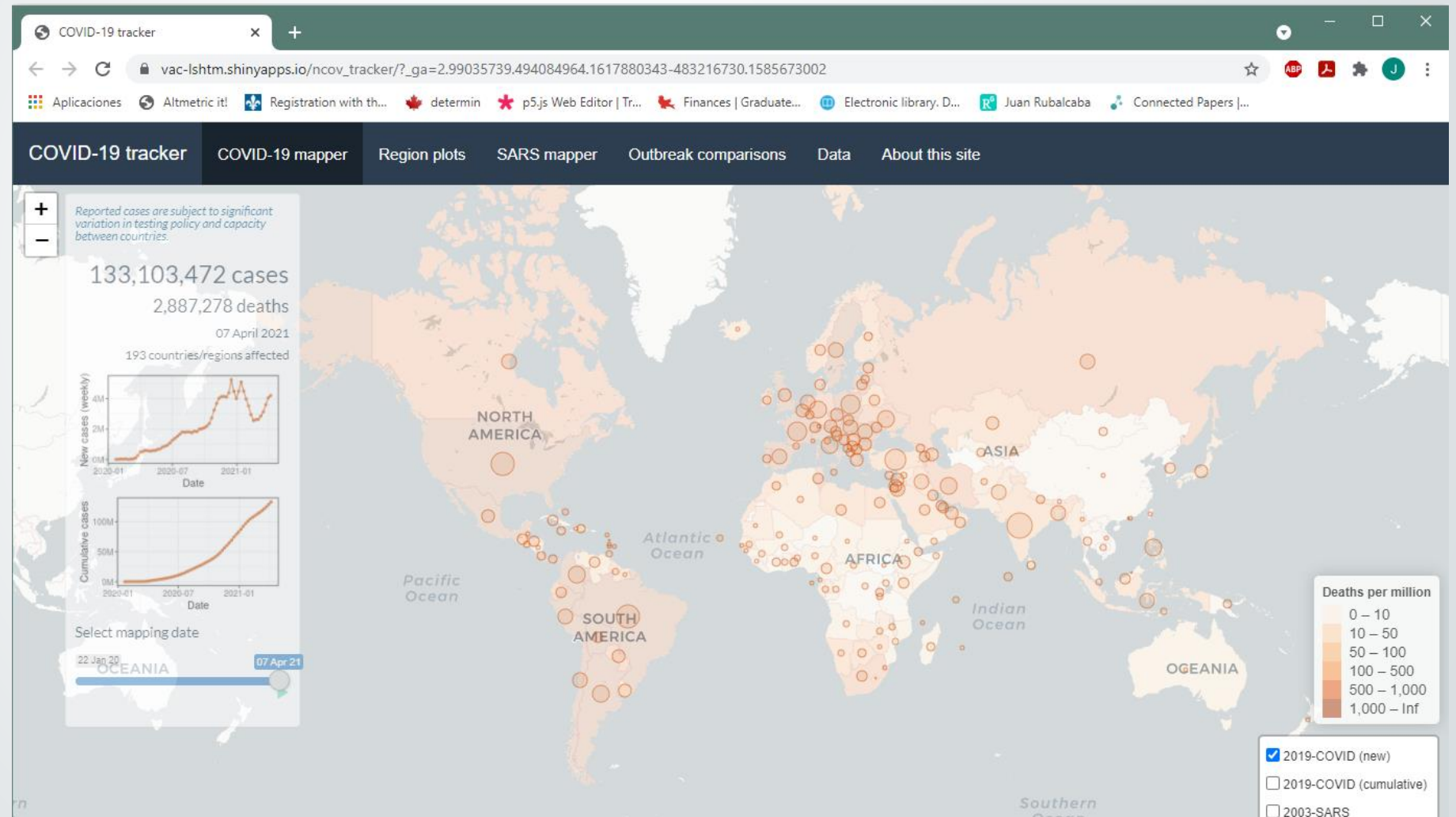
Data / code **transparency**

Understanding **models**

Educational apps

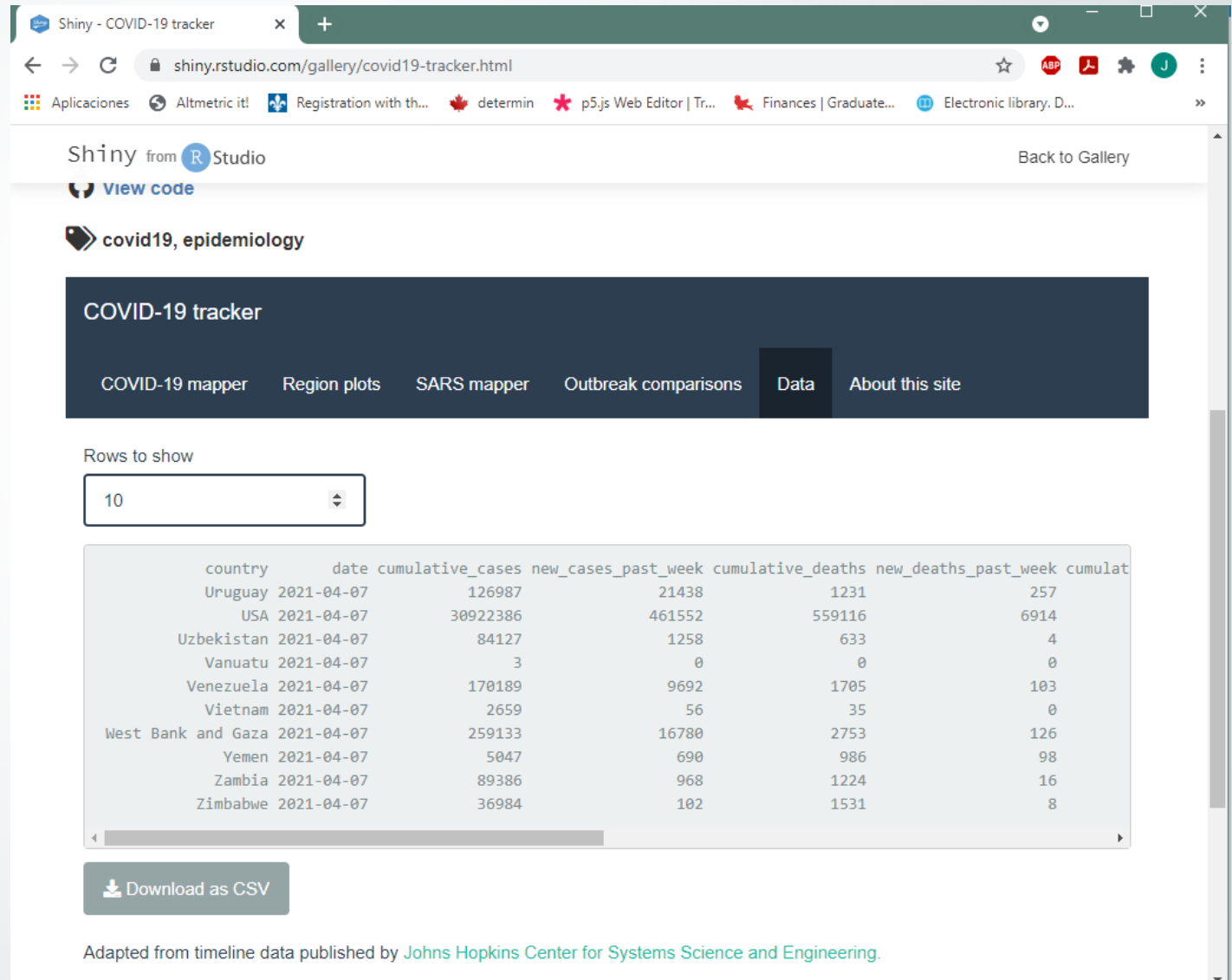
Data visualization – manage large datasets

COVID19 Tracker <https://shiny.rstudio.com/gallery/covid19-tracker.html>



Data visualization – manage large datasets

COVID19 Tracker <https://shiny.rstudio.com/gallery/covid19-tracker.html>

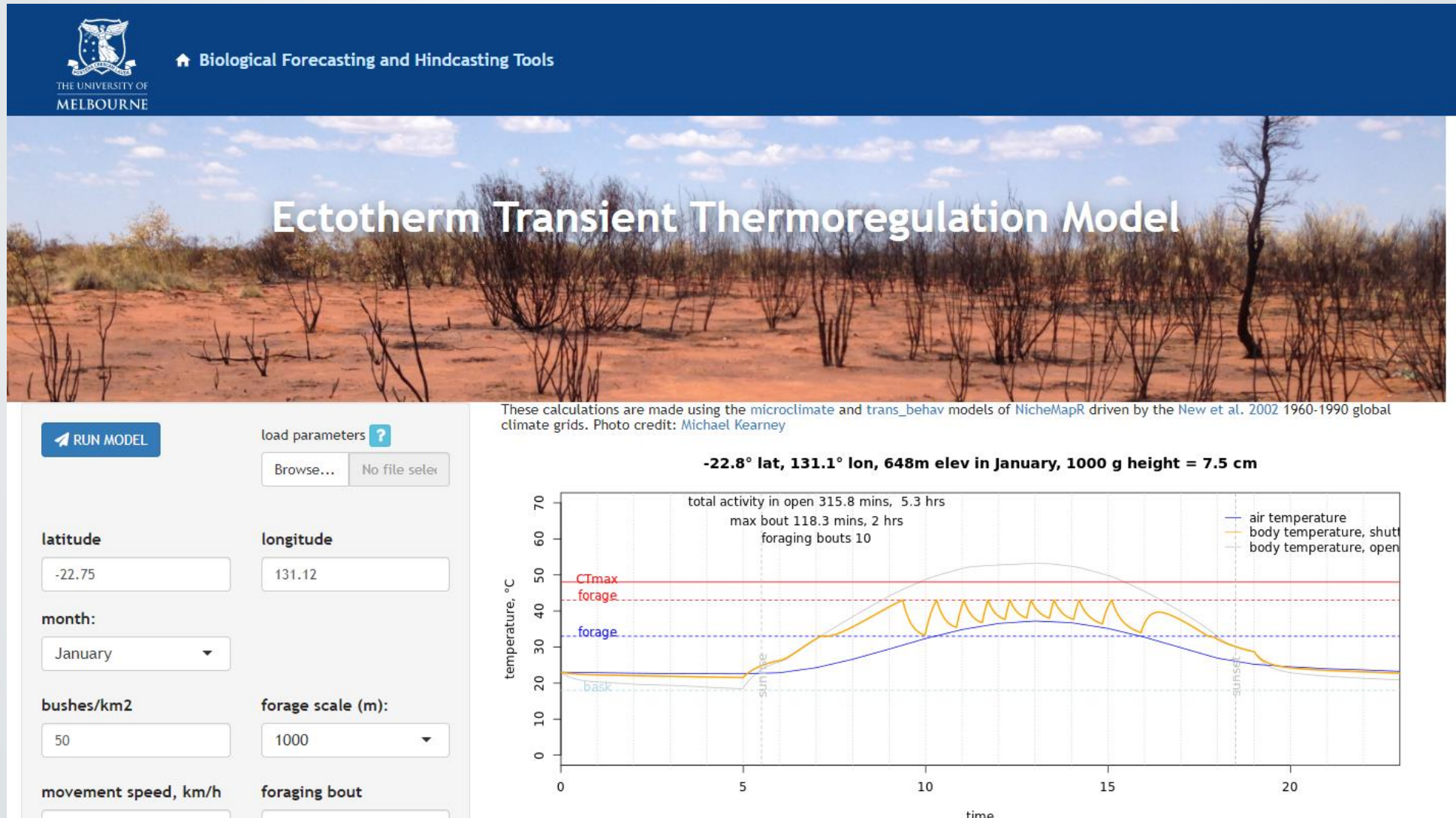


The screenshot shows the COVID-19 Tracker Shiny app interface. At the top, there's a navigation bar with the title "COVID-19 tracker" and several tabs: "COVID-19 mapper", "Region plots", "SARS mapper", "Outbreak comparisons", "Data" (which is selected), and "About this site". Below the tabs, there's a section titled "Rows to show" with a dropdown menu set to "10". The main content area displays a table of COVID-19 data. The table has columns for country, date, cumulative_cases, new_cases_past_week, cumulative_deaths, new_deaths_past_week, and cumulative_deaths. The data is sorted by date, showing records for Uruguay, USA, Uzbekistan, Vanuatu, Venezuela, Vietnam, West Bank and Gaza, Yemen, Zambia, and Zimbabwe, all dated 2021-04-07. At the bottom of the table, there's a "Download as CSV" button. The footer text states: "Adapted from timeline data published by Johns Hopkins Center for Systems Science and Engineering."

country	date	cumulative_cases	new_cases_past_week	cumulative_deaths	new_deaths_past_week	cumulative_deaths
Uruguay	2021-04-07	126987	21438	1231	257	
USA	2021-04-07	30922386	461552	559116	6914	
Uzbekistan	2021-04-07	84127	1258	633	4	
Vanuatu	2021-04-07	3	0	0	0	
Venezuela	2021-04-07	170189	9692	1705	103	
Vietnam	2021-04-07	2659	56	35	0	
West Bank and Gaza	2021-04-07	259133	16780	2753	126	
Yemen	2021-04-07	5047	690	986	98	
Zambia	2021-04-07	89386	968	1224	16	
Zimbabwe	2021-04-07	36984	102	1531	8	

Run simulations - download data

http://bioforecasts.science.unimelb.edu.au/app_direct/ectotherm_transient/



Are Shiny apps **substituting other R packages?**

Are Shiny apps **substituting** other R packages?

ECOGRAPHY

A JOURNAL OF SPACE
AND TIME IN ECOLOGY

Software note | [Free Access](#)

NicheMapR – an R package for biophysical modelling: the microclimate model

Michael R. Kearney✉, Warren P. Porter

First published: 25 August 2016 | <https://doi.org/10.1111/ecog.02360> | Citations: 80

Are Shiny apps **substituting** other R packages?

Methods in Ecology and Evolution



APPLICATION |  Free Access |

A Shiny R app to solve the problem of when to stop managing or surveying species under imperfect detection

Luz Pascal , Milad Memarzadeh, Carl Boettiger, Hannah Lloyd, Iadine Chadès

First published: 02 October 2020 | <https://doi.org/10.1111/2041-210X.13501>

ranacapa: An R package and Shiny web app to explore environmental DNA data with exploratory statistics and interactive visualizations

[Gaurav S. Kandlikar](#), Conceptualization, Data Curation, Methodology, Project Administration, Software, Validation,

Data / Code transparency

Data / Code transparency

Climate_data.zip

- Climate_data
 - BCM_Tmax.zip 11.3 GB
 - BCM_Tmin.zip 11.1 GB
 - BCM_ppt.zip 7.6 GB
 - CA_Soil_Thickness.asc 156.1 MB
 - CA_Vegetation.asc 72.7 MB
- MACOSX
 - Climate_data
 - ._BCM_Tmax.zip 220 Bytes
 - ._BCM_Tmin.zip 220 Bytes
 - ._BCM_ppt.zip 220 Bytes
 - ._CA_Soil_Thickness.asc 314 Bytes
 - ._CA_Vegetation.asc 314 Bytes

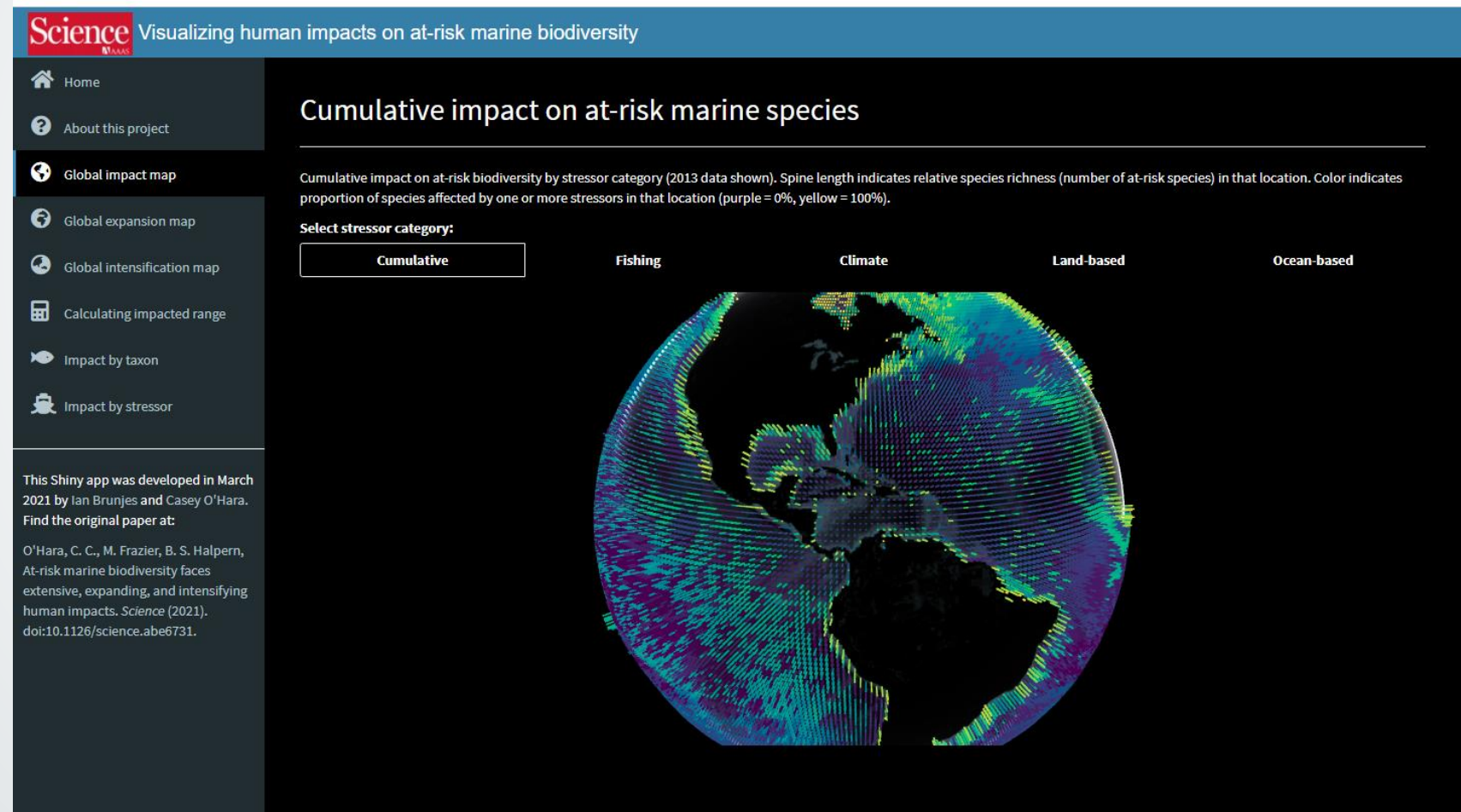
Files (30.1 GB)		
Name	Size	
Climate_data.zip	30.0 GB	Preview Download
md5:d8065e911a37f66198464238eeb726d2 ?		
Endoscape.zip	45.7 MB	Preview Download
md5:908550656a3853295b9cf07dc8450ab8 ?		
msom_jags_code.R	4.8 kB	Download
md5:cceff56e26dd1b15c9a69638dbb6983c ?		
NicheMapR_code_and_files.zip	7.2 MB	Preview Download
md5:7897e05be054d8c2660040804a76f798 ?		
read_me.rtf	7.3 kB	Download
md5:ee5e740ca2e237a5b5006ca946236963 ?		
Resurvey_data.zip	322.2 kB	Preview Download
md5:adb5ff75d23287f112a49d1565c7141e ?		
Sample_NETCDF.py	13.0 kB	Download
md5:2fa200cfa7299e58c7513bc436264b1a ?		
siom_jags_code.R	5.9 kB	Download

Riddell et al. (2021, *Science*)

Data / Code transparency

O'Hara et al. (2021) *Science*

http://ohi-science.nceas.ucsb.edu/visualizing_human_impacts/

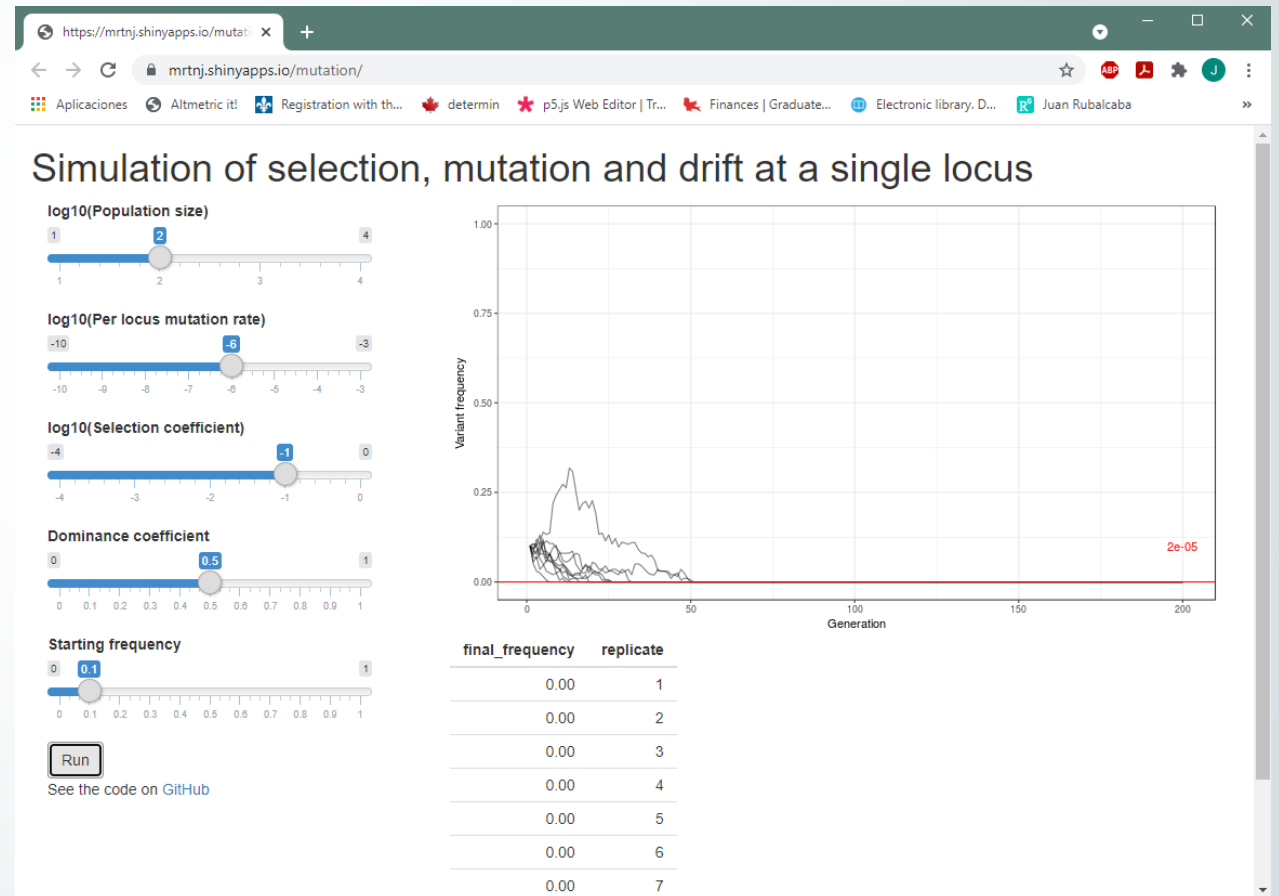


Educational apps

Population genetics model (Mutation, selection, and drift)

Blog: <https://www.r-bloggers.com/2017/05/mutation-selection-and-drift-with-shiny/>

Shiny App: <https://mrtmj.shinyapps.io/mutation/>



Educational apps

Shiny apps for models in **ecology and evolution**

<http://ecoevoeducation.github.io/EcoEvoApps/>

Phylogenetic comparative methods

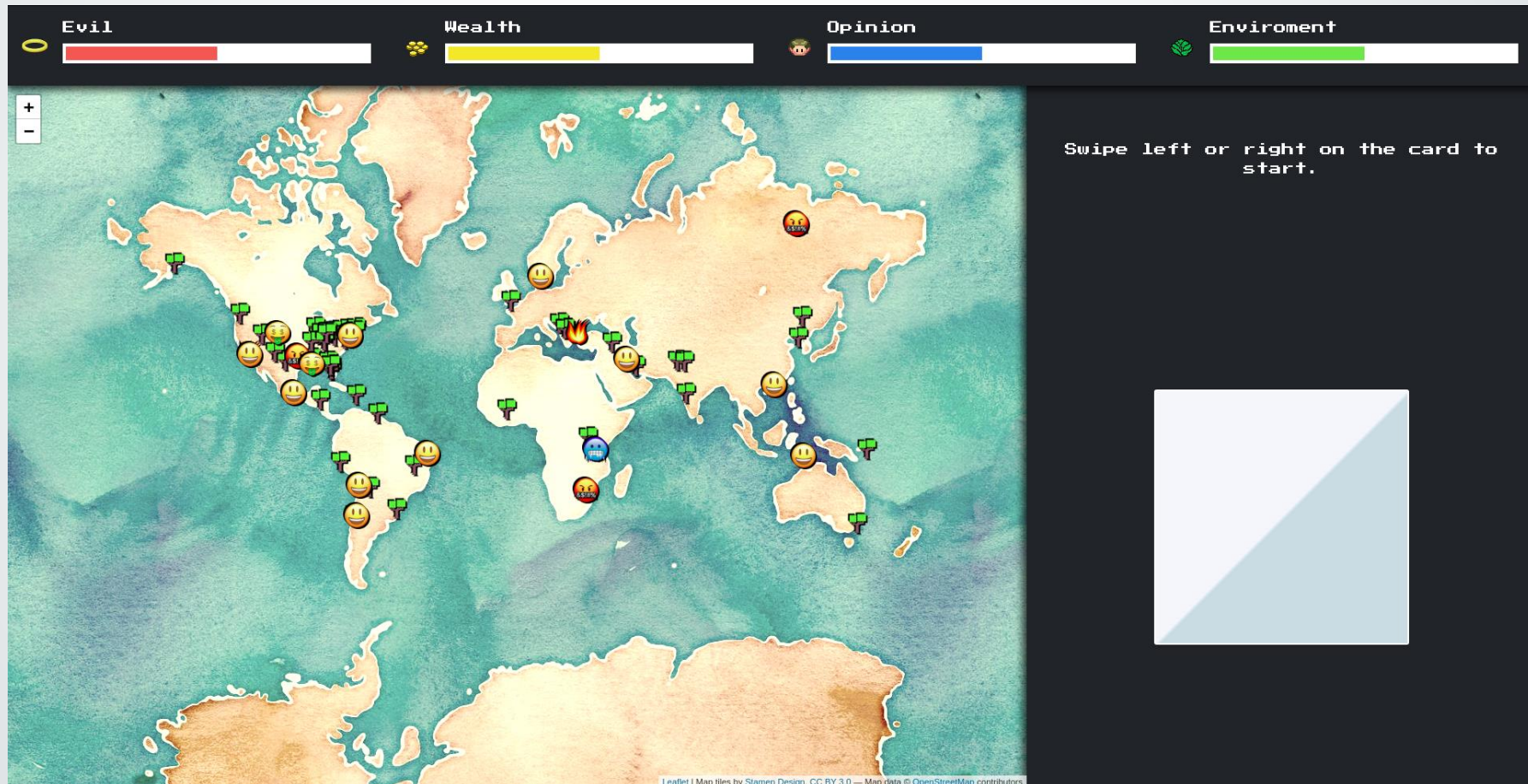
https://mossmatters.shinyapps.io/Phylo_Comp_Methods_Shiny/

Teaching **statistics**

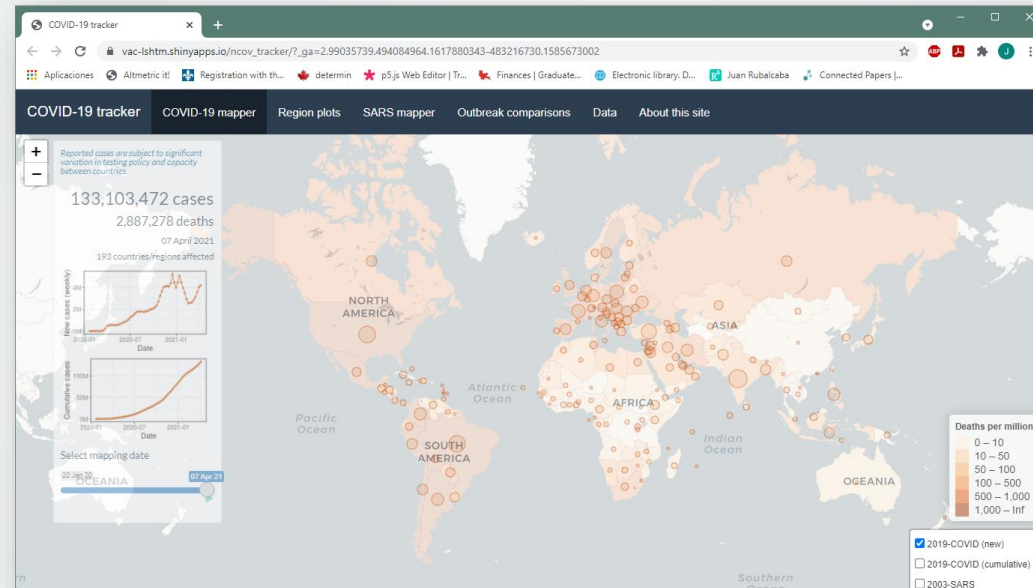
<https://saskiaotto.de/shiny/>

Video games (??)

<https://towardsdatascience.com/is-r-shiny-versatile-enough-to-build-a-video-game-5c93232ef4e2>



Structure of a shiny app



Shiny

```
dens <- density(data, n = npts)
dx <- dens$x
dy <- dens$y
if(add == TRUE)
  plot(0., 0., main = "Density Plot",
       xlab = "x", ylab = "y",
       if(orientation == "y") {
         dx2 <- (dx - min(dx)) / (max(dx) - min(dx))
         x[1.]
       } else {
         dy2 <- (dy - min(dy)) / (max(dy) - min(dy))
         y[1.]
       })
seqbelow <- rep(y[1.], length(dx))
if(Fill == T)
  confshade(dx2, seqbelow, dy2)
```

Structure of a shiny app

User interface

```
ui <- fluidPage(  
  "what is being displayed on the app page"  
)
```

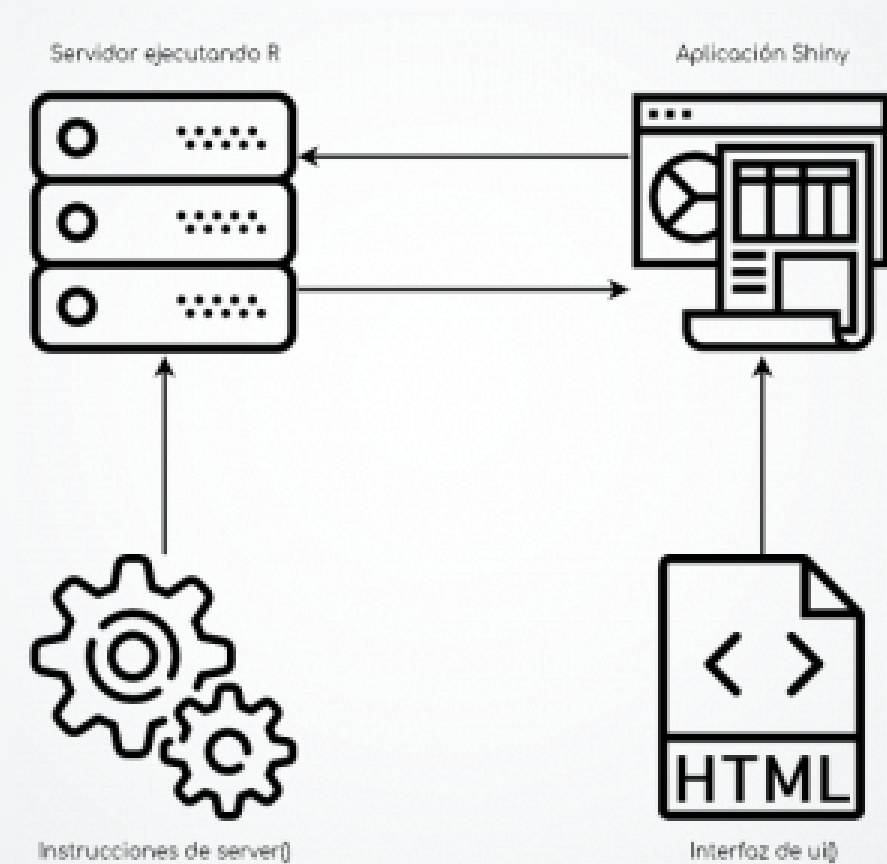
Server

```
server <- function(input, output, session) {  
  "what is the app doing with your input data"  
}
```

Run the app

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

How it works

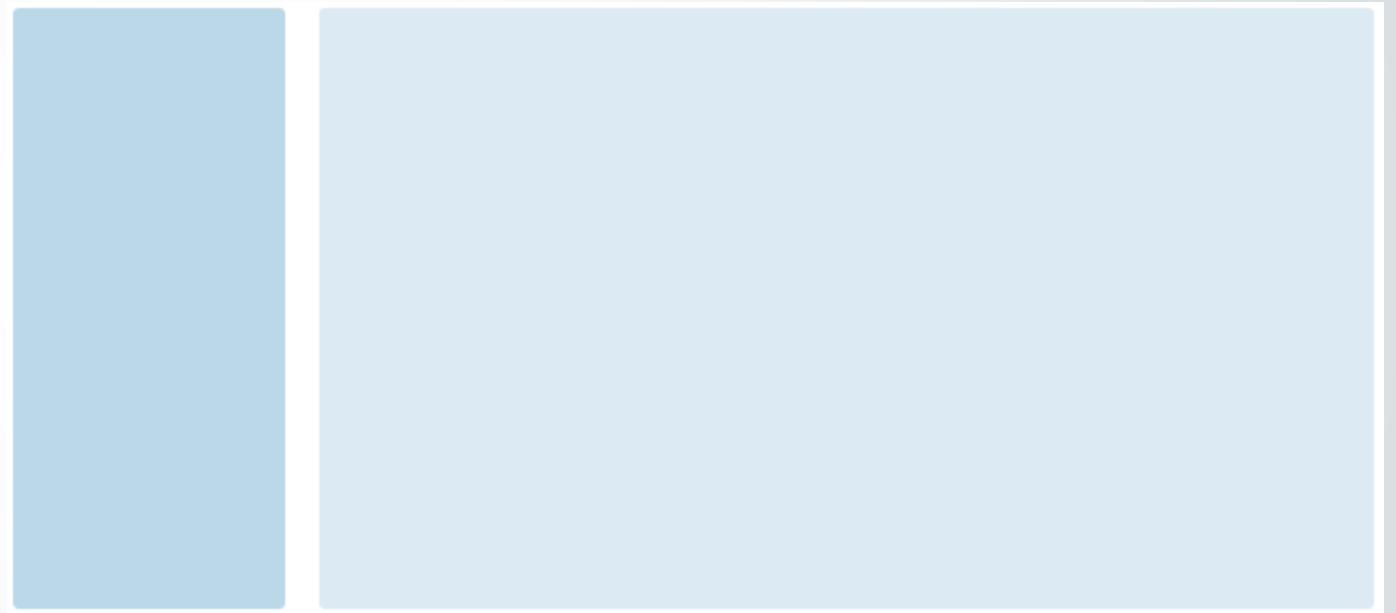


server

**user
interface**

First layout

```
ui <- fluidPage(  
  sidebarLayout(  
    sidebarPanel(  
    ),  
    mainPanel(  
  )  
)
```

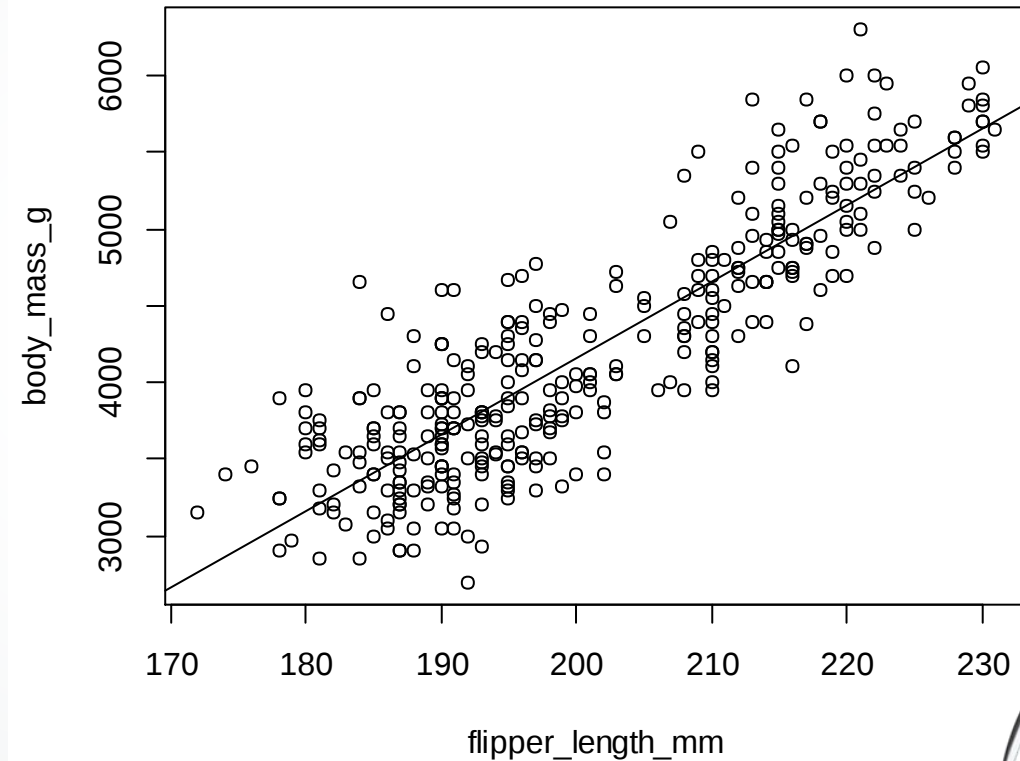


A quick example

Making a scatterplot with R

```
plot(body_mass_g ~ flipper_length_mm, penguins)
```

```
mod <- lm(body_mass_g ~ flipper_length_mm, penguins)  
abline(mod)
```

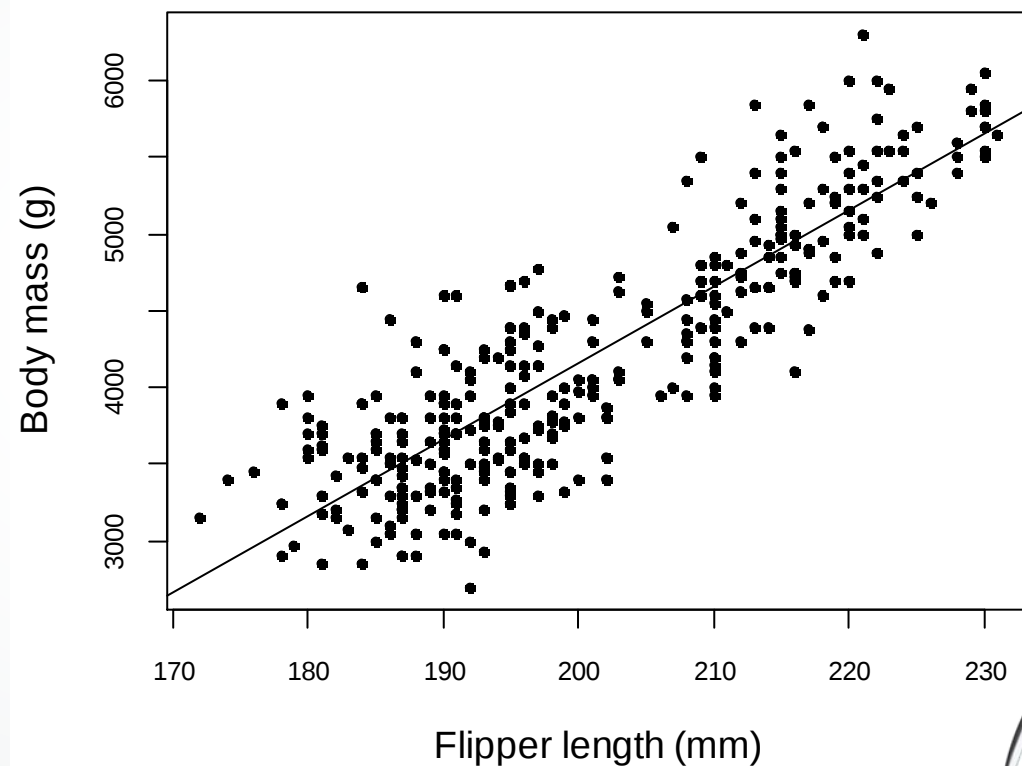


A quick example

Making a scatterplot with R

```
plot(body_mass_g ~ flipper_length_mm, penguins,  
     pch = 20,  
     cex.lab = 1.2,  
     cex.axis = 0.8,  
     ylab = "Body mass (g)",  
     xlab = "Flipper length (mm)")
```

```
mod <- lm(body_mass_g ~ flipper_length_mm, penguins)  
abline(mod)
```



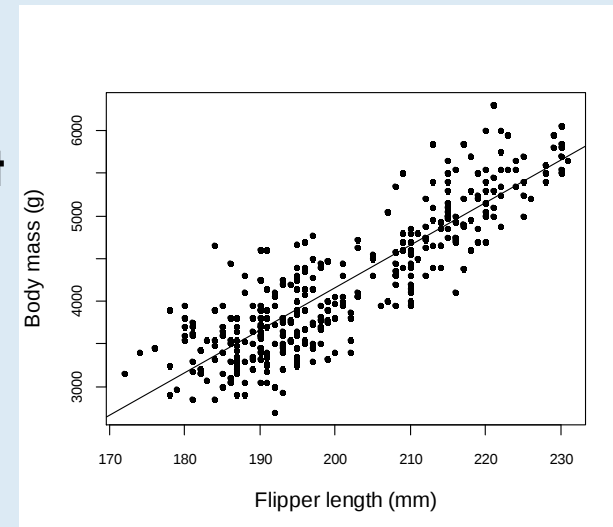
A quick example

Making a scatterplot with Shiny

```
ui <- fluidPage(  
  sidebarLayout(  
    sidebarPanel(  
  
    ),  
    mainPanel(  
      plotOutput(outputId = "scatterplot")  
    )  
  )  
)  
  
server <- function(input, output, session) {  
  output$scatterplot <- renderPlot({  
    plot(body_mass_g ~ flipper_length_mm, penguins)  
    mod <- lm(body_mass_g ~ flipper_length_mm, penguins)  
    abline(mod)  
  })  
}
```

shinyApp(ui, server)

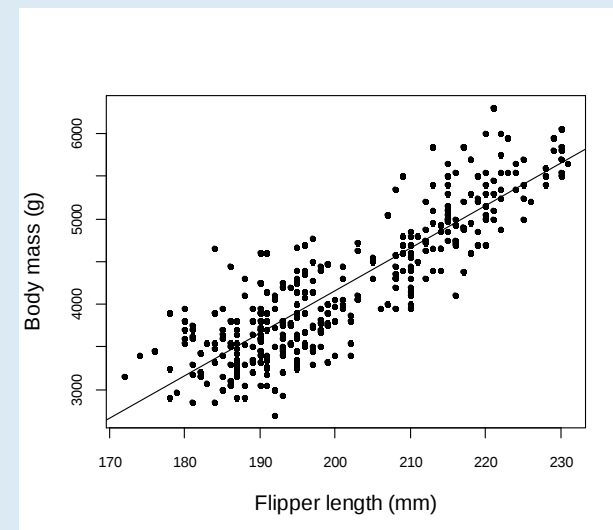
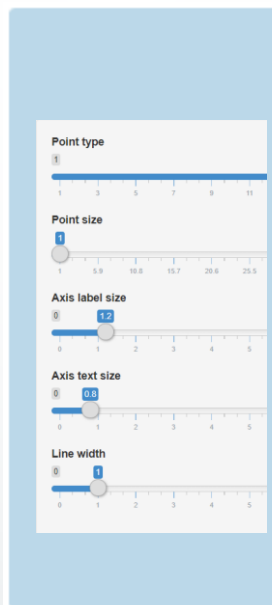
R code for
the scatterplot



A quick example

Making a scatterplot with Shiny

```
ui <- fluidPage(  
  sidebarLayout(  
    sidebarPanel(  
      sliderInput(inputId = "pch", label = "Point type", min = 1, max = 21, value = 20, step = 1),  
      sliderInput(inputId = "cex", label = "Point size", min = 1, max = 50, value = 1, step = 0.1),  
      sliderInput(inputId = "cex.lab", label = "Axis label size", min = 0, max = 10, value = 1.2, step = 0.1),  
      sliderInput(inputId = "cex.axis", label = "Axis text size", min = 0, max = 10, value = 0.8, step = 0.1),  
      sliderInput(inputId = "lwd", label = "Line width", min = 0, max = 10, value = 1, step = 0.1)  
    ),  
    mainPanel(  
      plotOutput(outputId = "scatterplot")  
    )  
  )  
)  
  
server <- function(input, output, session) {  
  output$scatterplot <- renderPlot({  
    plot(body_mass_g ~ flipper_length_mm, penguins,  
      pch = input$pch,  
      cex = input$cex,  
      ylab = "Body mass (g)",  
      xlab = "Flipper length (mm)",  
      cex.lab = input$cex.lab,  
      cex.axis = input$cex.axis)  
    mod <- lm(body_mass_g ~ flipper_length_mm, penguins)  
    abline(mod, lwd = input$lwd)  
  })  
}
```

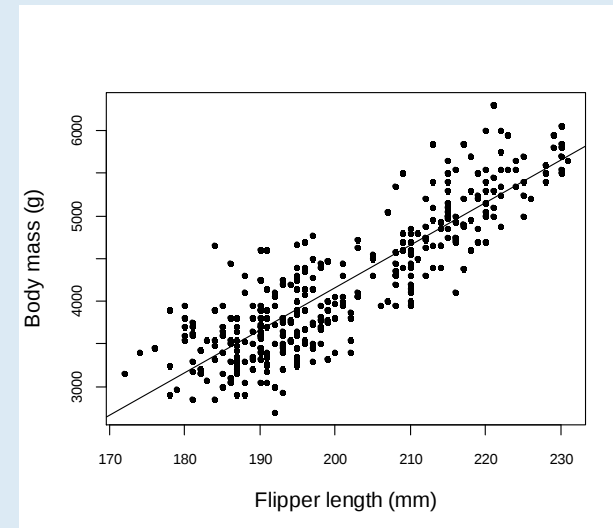
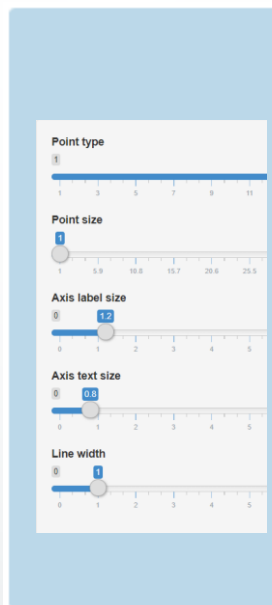


A quick example

Making a scatterplot with Shiny

```
ui <- fluidPage(  
  sidebarLayout(  
    sidebarPanel(  
      sliderInput(inputId = "pch", label = "Point type", min = 1, max = 21, value = 20, step = 1),  
      sliderInput(inputId = "cex", label = "Point size", min = 1, max = 50, value = 1, step = 0.1),  
      sliderInput(inputId = "cex.lab", label = "Axis label size", min = 0, max = 10, value = 1.2, step = 0.1),  
      sliderInput(inputId = "cex.axis", label = "Axis text size", min = 0, max = 10, value = 0.8, step = 0.1),  
      sliderInput(inputId = "lwd", label = "Line width", min = 0, max = 10, value = 1, step = 0.1)  
    ),  
    mainPanel(  
      plotOutput(outputId = "scatterplot")  
    )  
  )  
)
```

```
server <- function(input, output, session) {  
  output$scatterplot <- renderPlot({  
    plot(body_mass_g ~ flipper_length_mm, penguins,  
      pch = input$pch,  
      cex = input$cex,  
      ylab = "Body mass (g)",  
      xlab = "Flipper length (mm)",  
      cex.lab = input$cex.lab,  
      cex.axis = input$cex.axis)  
    mod <- lm(body_mass_g ~ flipper_length_mm, penguins)  
    abline(mod, lwd = input$lwd)  
  })  
}
```

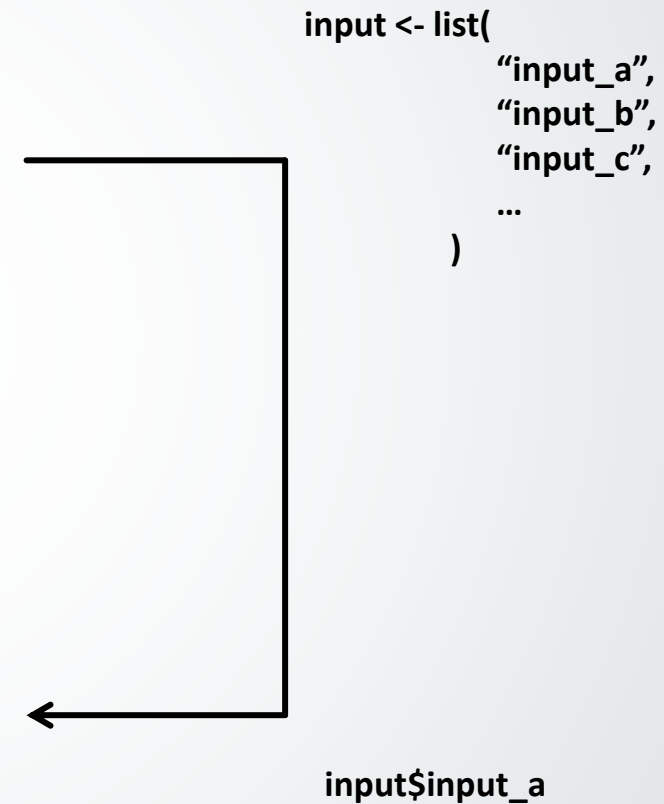


Example: 2_scatterplot_lm.R

Example: 3_scatterplot_widgets.R

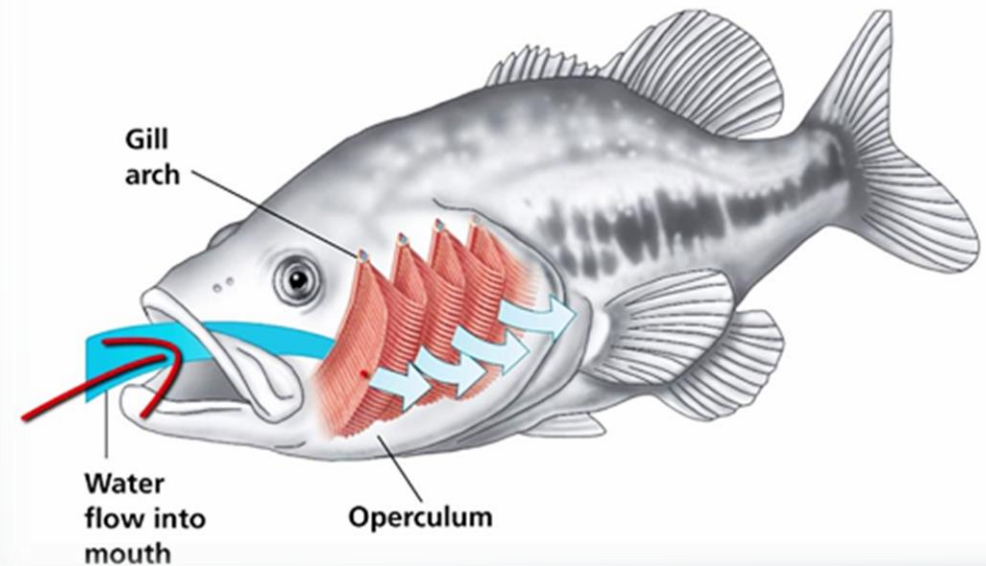
Basic Shiny app structure

```
ui <- fluidPage(  
  sidebarLayout(  
    sidebarPanel(  
      Input parameters, sliders, action buttons....  
    ),  
    mainPanel(  
      What are we going to show (plots, tables...)  
    )  
  )  
)  
  
server <- function(input, output, session) {  
  output$scatterplot <- renderPlot({  
    R code for your plots  
  })  
}  
  
shinyApp(ui, server)
```



A more useful example

A model to derive **fish metabolic rate** as a function of **water temperature**



A more useful example

A model to derive **fish metabolic rate** as a function of **water temperature**

$$\dot{m}_{O_2,demand} = \delta b_0 M^{\frac{3}{4}} e^{-\frac{E}{kT}}$$

Metabolic rate ($\dot{m}_{O_2,demand}$) depends on...

delta,	Activity level
b0,	Normalization constant
M,	Body mass
E,	Activation energy
T,	Temperature

A more useful example

A model to derive **fish metabolic rate** as a function of **water temperature**

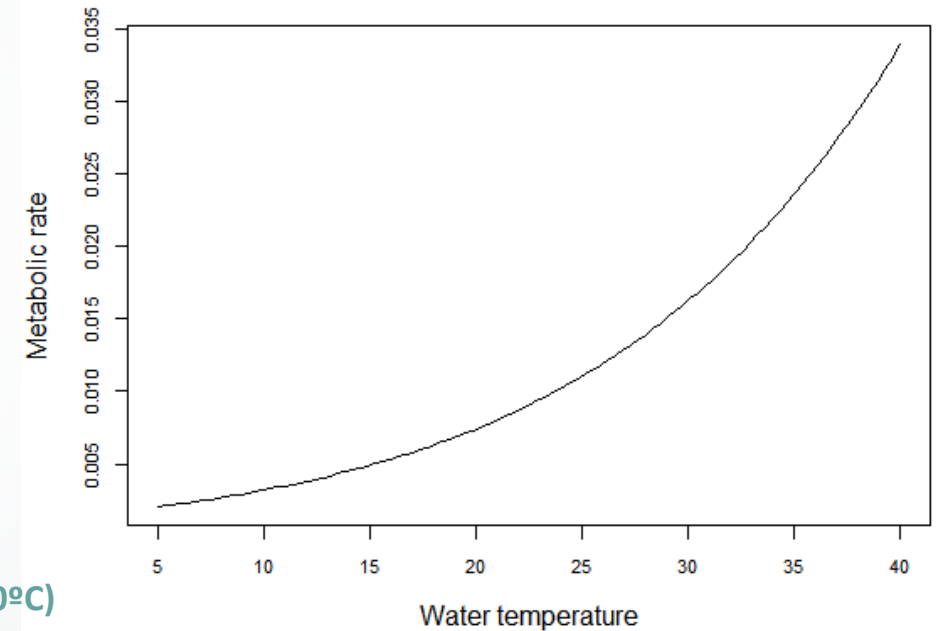
```
MetRate <- function(Tw, delta, b0, M, E){  
  met_rate <- delta * b0 * M^(3/4) * exp(-E/(8.61e-05*(Tw+273)))  
  return(met_rate)  
}
```

```
M = 100    # Body mass  
E = 0.6    # activation energy  
b0 = 5e6   # normalization constant  
delta = 1  # Activity level
```

```
Tw <- seq(5, 40, length.out = 100) # water temperature (100 values from 5°C to 40°C)
```

```
mO2 <- MetRate(Tw=Tw, delta=delta, , b0=b0, M=M, E=E)
```

```
plot(mO2 ~ Tw, type = "l", cex.axis = 0.8, cex.lab = 1.2, ylab = "Metabolic rate", xlab = "Water temperature")
```



A more useful example

A model to derive **fish metabolic rate** as a function of **water temperature**

$$\dot{m}_{O_2,demand} = \delta b_0 M^{\frac{3}{4}} e^{-\frac{E}{kT}} \frac{\widehat{\rho O_{2,gill}}}{\widehat{\rho O_{2,gill}} + K_M}$$

Metabolic rate ($\dot{m}_{O_2,demand}$) depends on...

δ	Activity level
M	Body mass
A	Gill surface area
h_c	Oxygen transfer coefficient
$O_{2,water}$	Oxygen concentration
K_M	Michaelis constant
E	Activation energy
b_0	Normalization constant

Example: 4_nonlinear_model.R

https://jrubicaba.shinyapps.io/app_oxlim

UI Layouts

UI Layouts

```
ui <- fluidPage(
```

```
  sidebarLayout(
```

```
    sidebarPanel(
```

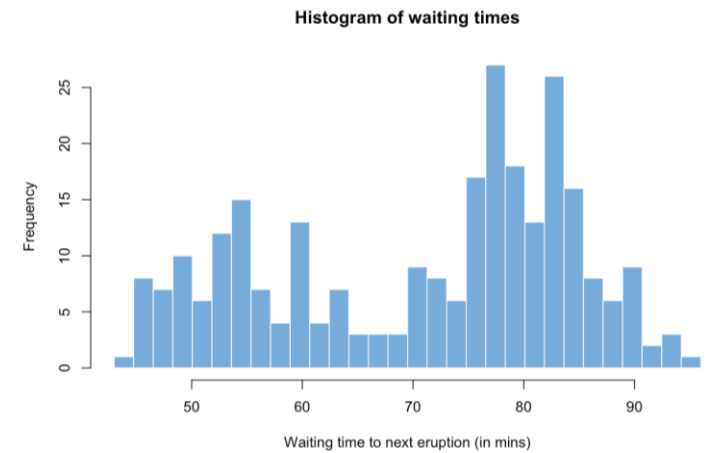
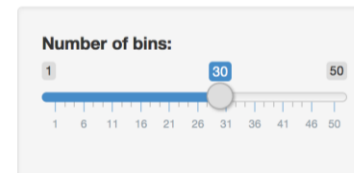
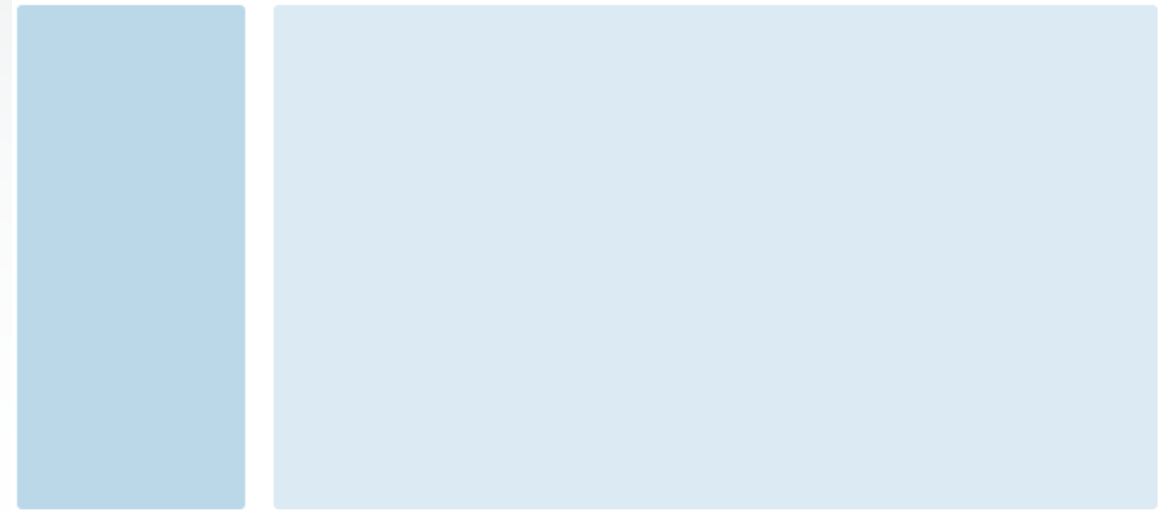
```
      ),
```

```
    mainPanel(
```

```
      )
```

```
  )
```

```
)
```



UI Layouts

```
ui <- fluidPage(
```

```
  fluidRow(
```

```
    column(2,
```

```
    ),
```

```
    column(5,
```

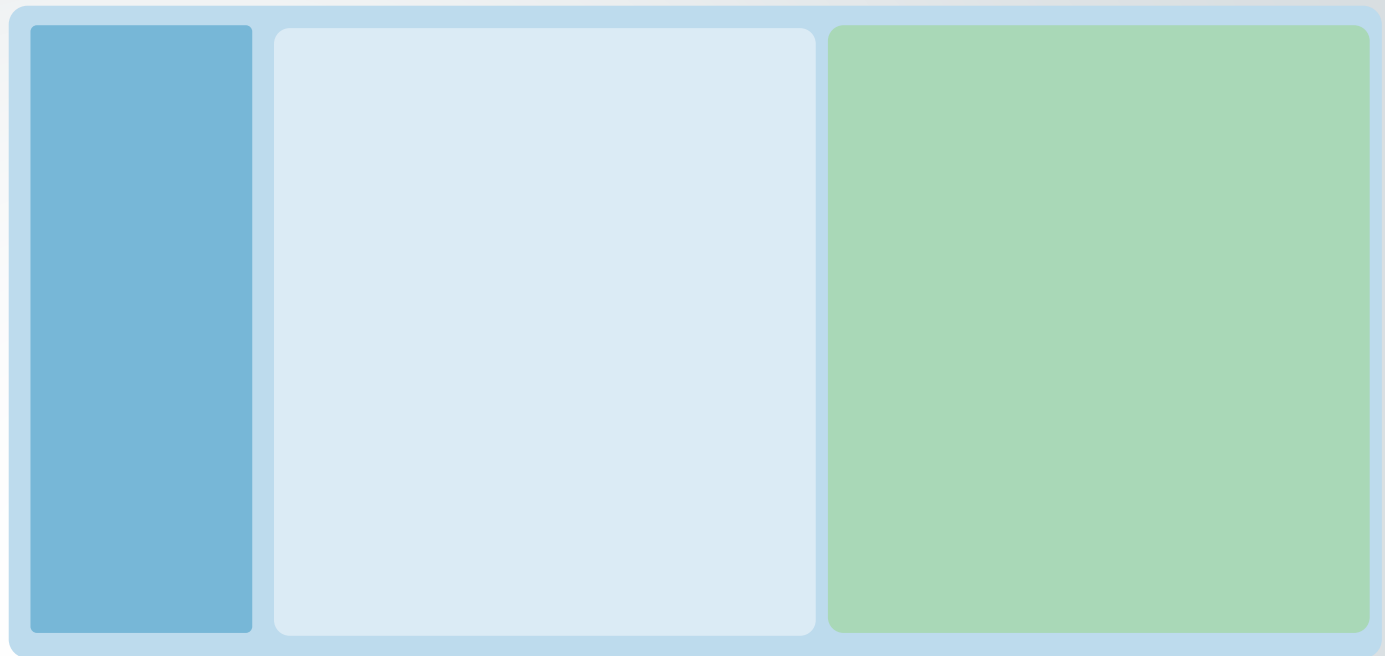
```
    ),
```

```
    column(5,
```

```
    )
```

```
  )
```

```
)
```



*12 columns

UI Layouts

```
ui <- fluidPage(
```

```
  fluidRow(
```

```
    column(2,
```

```
    column(5,
```

```
      fluidRow(
```

```
        column(1, offset = 1,
```

```
        ),
```

```
        column(1, offset = 1,
```

```
        ),
```

```
      )
```

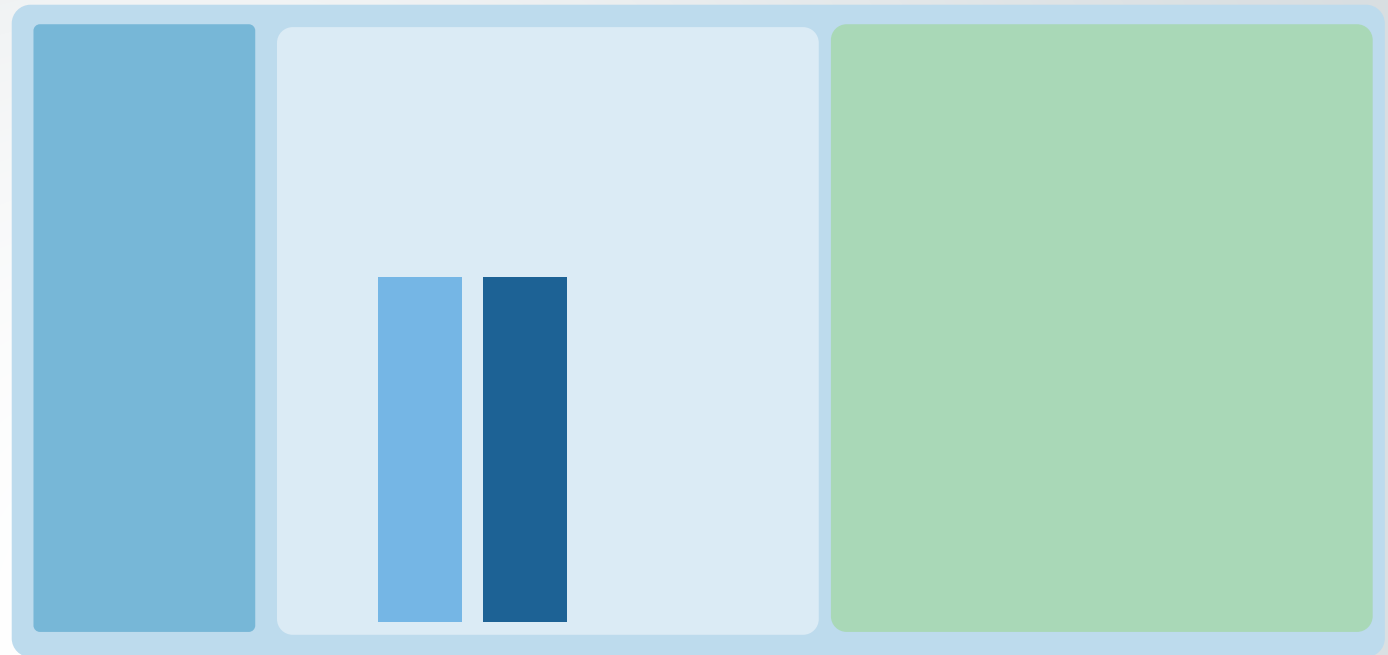
```
    ),
```

```
    column(5,
```

```
  )
```

```
)
```

```
)
```



*12 columns

UI Layouts

```
ui <- fluidPage(
```

```
  fluidRow(
```

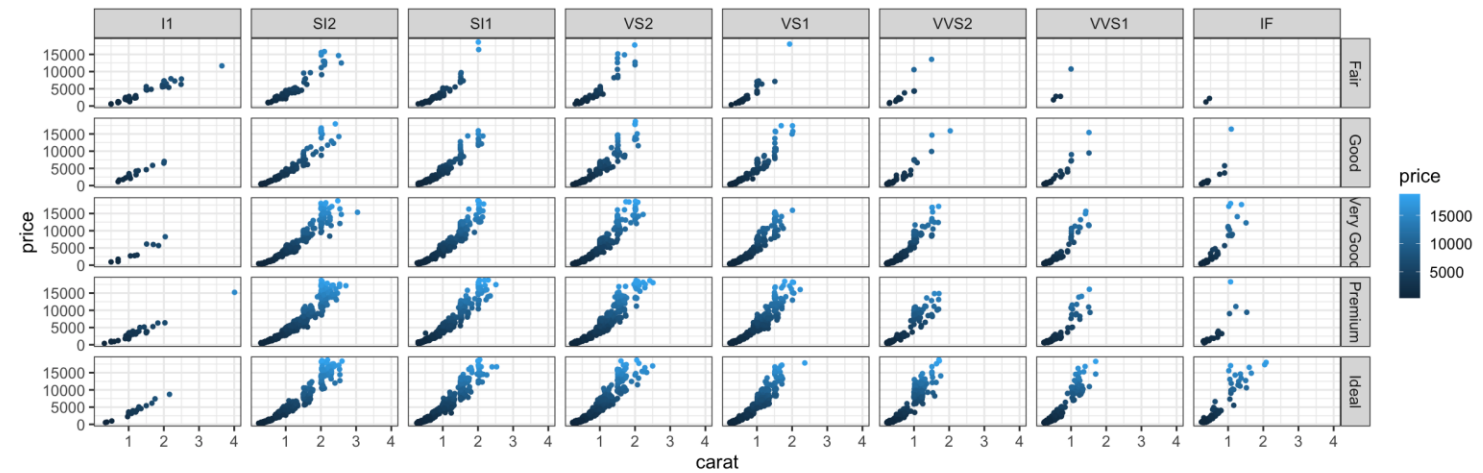
```
    column(2,
```

```
      column(5,
        fluidRow(
```

```
          column(1, offset = 1,
            ),
          column(1, offset = 1,
            ),
        )
      ),

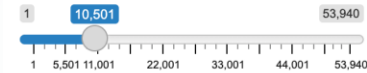
```

```
    column(5,
      )
    )
  )
)
```



Diamonds Explorer

Sample Size



☐ Jitter

☐ Smooth

X

carat

Y

price

Color

price

Facet Row

cut

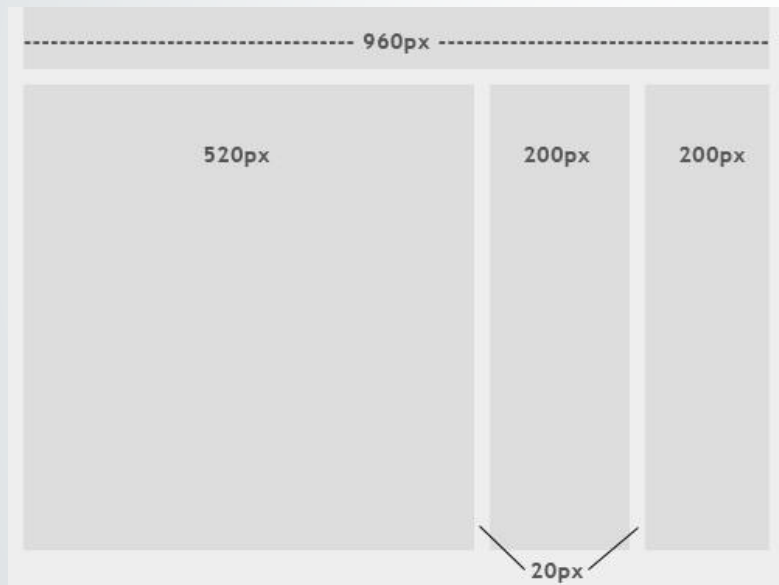
Facet Column

clarity

UI Layouts

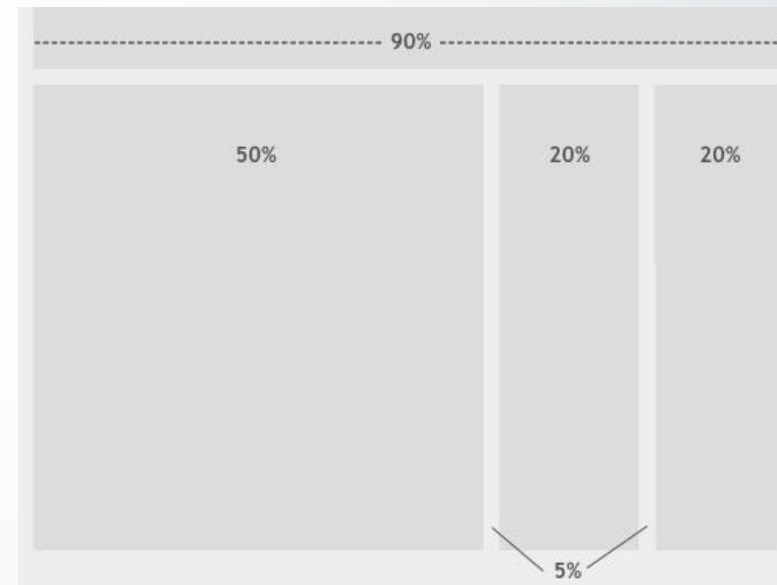
Fluid or fixed layouts?

Fixed



* 940px in Shiny

Fluid



UI Layouts

```
ui <- fluidPage(
```

```
  sidebarLayout(
```

```
    sidebarPanel(
      ),
```

```
    mainPanel(
```

```
      tabsetPanel(
```

```
        tabPanel("Plot", plotOutput("plot")),
```

```
        tabPanel("Summary", verbatimTextOutput("summary")),
```

```
        tabPanel("Table", tableOutput("table"))
```

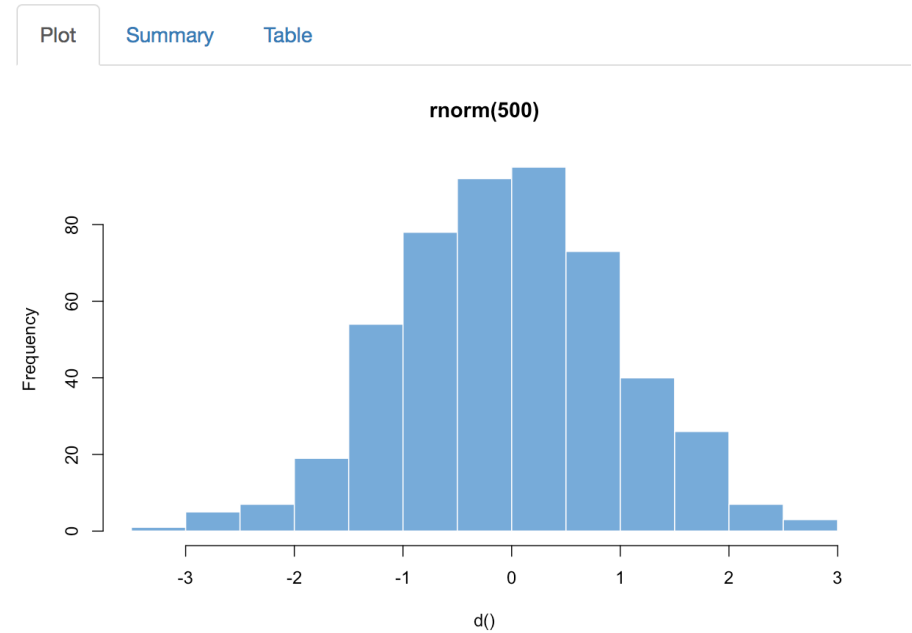
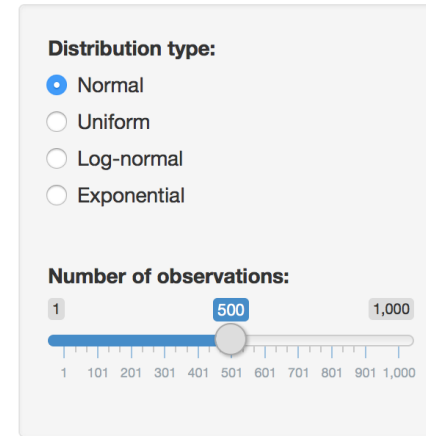
```
      )
```

```
    )
```

```
  )
```

```
)
```

Tabsets



UI Layouts

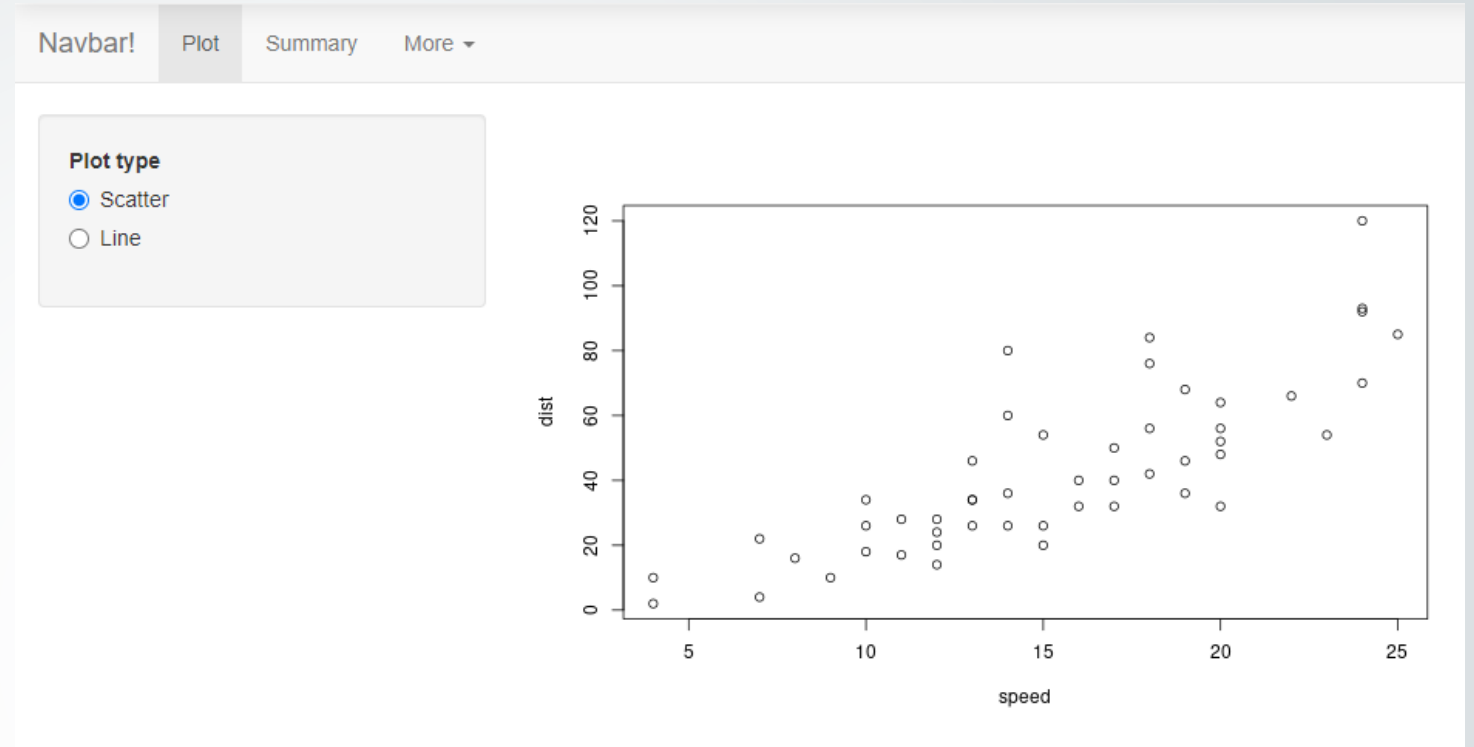
```
ui <- navbarPage("App name",
```

```
  tabPanel("Plot",
    sidebarLayout(
      sidebarPanel(
      ),
      mainPanel(
      )
    )
  ),
```

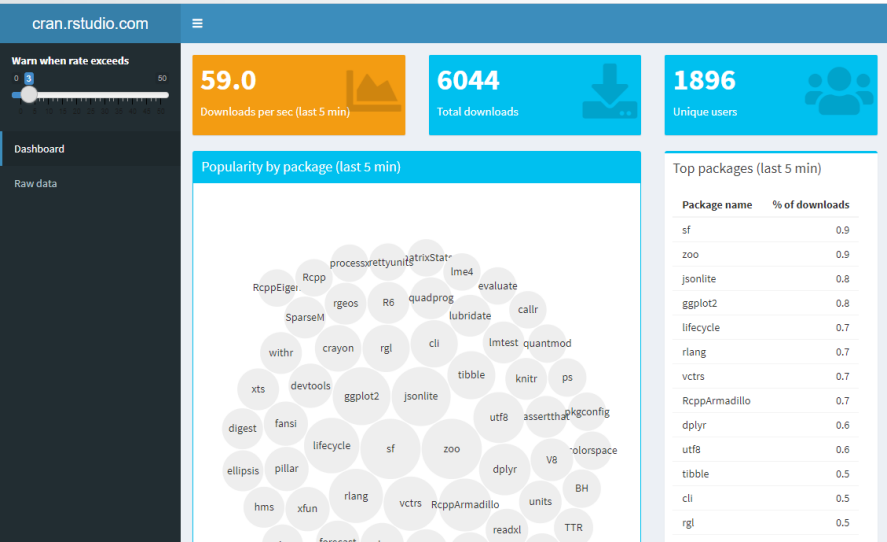
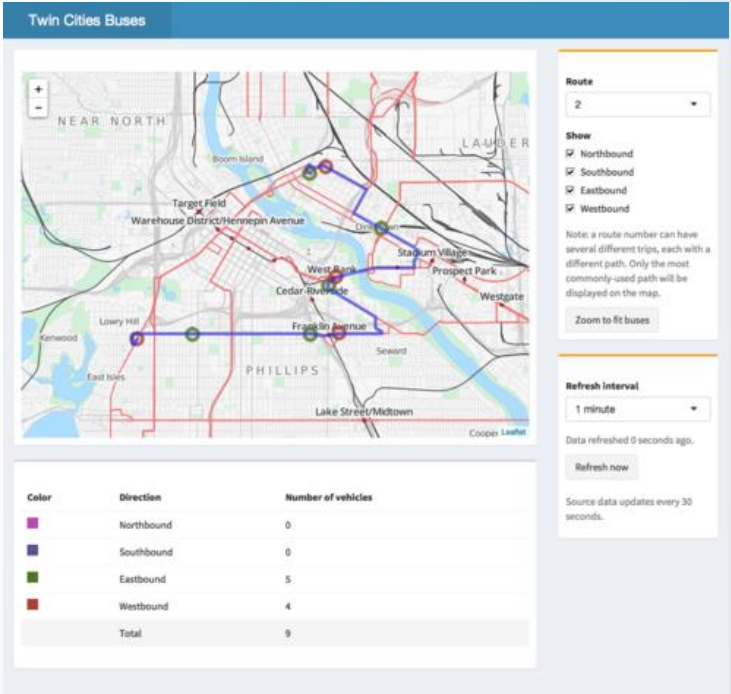
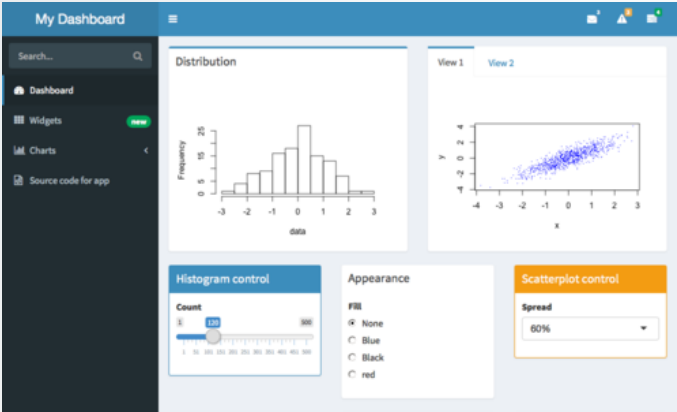
```
  tabPanel("Summary",
    - Layout of section 2 -
  ),
```

```
  tabPanel("More",
    - Layout of section 3 -
  )
```

```
)
```

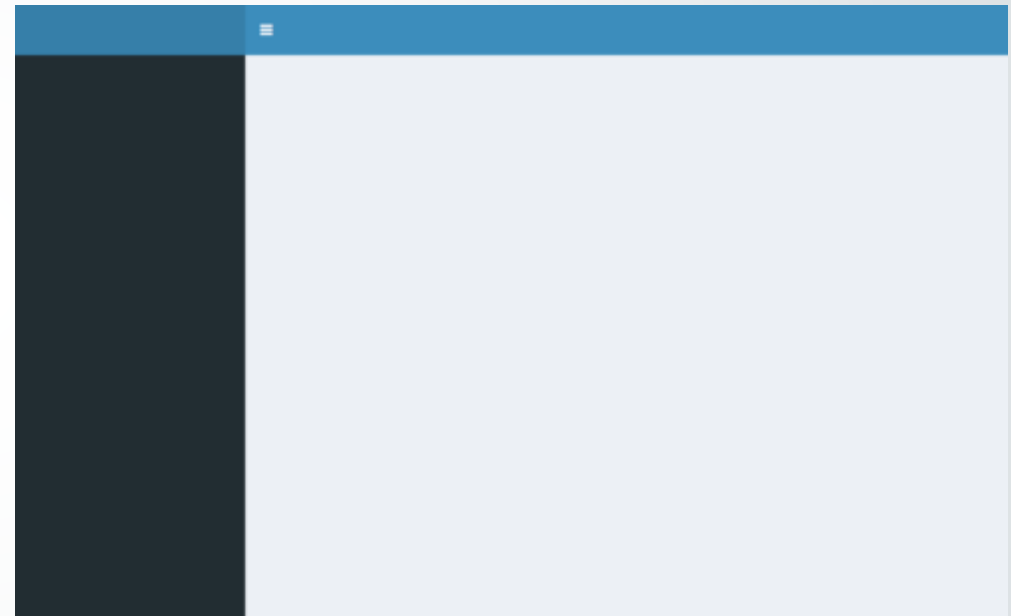


Making nicer UI with shinydashboard

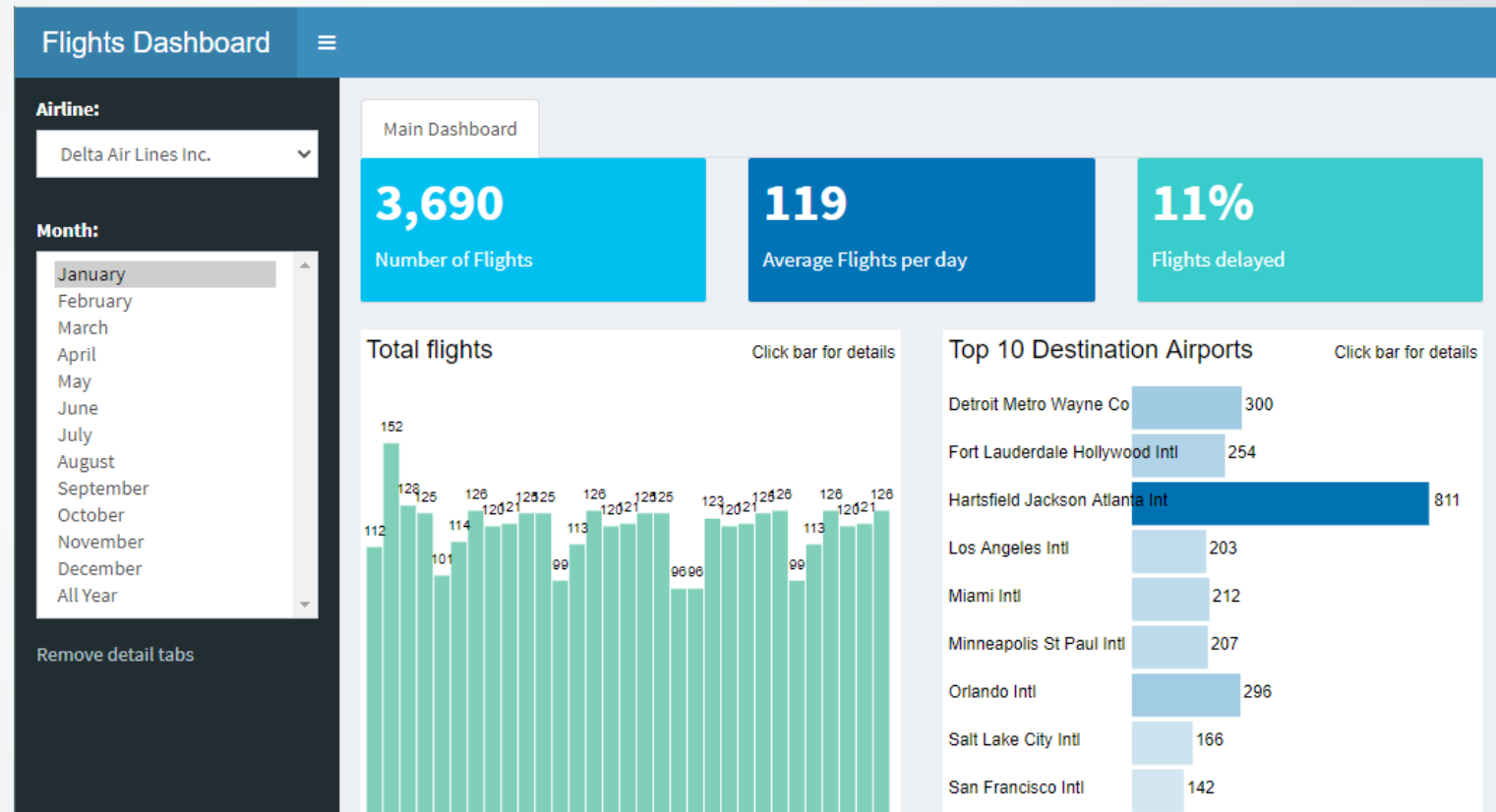


Making nicer UI with **shinydashboard**

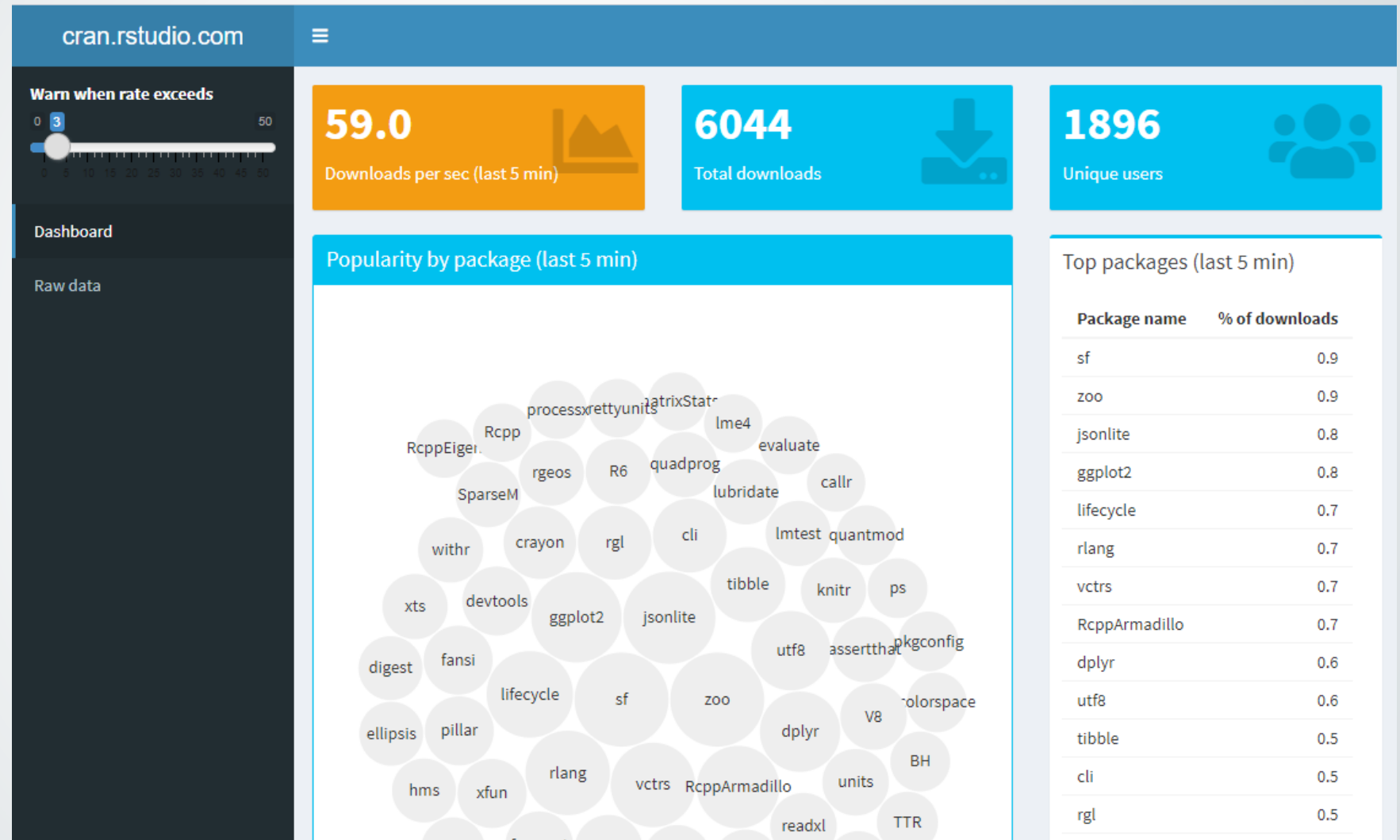
```
ui <- dashboardPage(  
  dashboardHeader(),  
  dashboardSidebar(),  
  dashboardBody()  
)
```



Making nicer UI with shinydashboard



Making nicer UI with shinydashboard



Making nicer UI with **shinydashboard**

```
ui <- fluidPage(
```

```
  sidebarLayout(  
    sidebarPanel(
```

Input parameters, sliders, action buttons....

```
  ),
```

```
    mainPanel(
```

What are we going to show (plots, tables...)

```
  )
```

```
)
```

```
server <- function(input, output, session) {
```

```
  output$scatterplot <- renderPlot({
```

R code to create our plots

```
  })
```

```
}
```

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

Making nicer UI with **shinydashboard**

```
ui <- fluidPage(
```

```
  sidebarLayout(
```

```
    sidebarPanel(
```

Input parameters, sliders, action buttons....

```
  ),
```

```
  mainPanel(
```

What are we going to show (plots,
tables...)

```
  )
```

```
)
```

```
)
```

```
ui <- dashboardPage(
```

```
  dashboardHeader(title = " Some title"),
```

```
  dashboardSidebar(
```

```
  ),
```

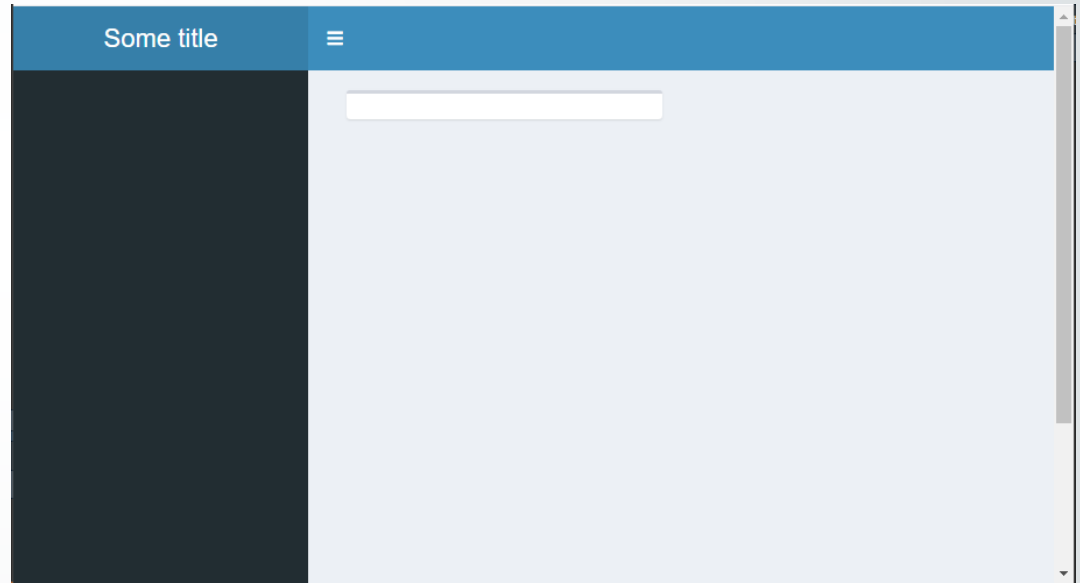
```
  dashboardBody(
```

```
  )
```

```
)
```

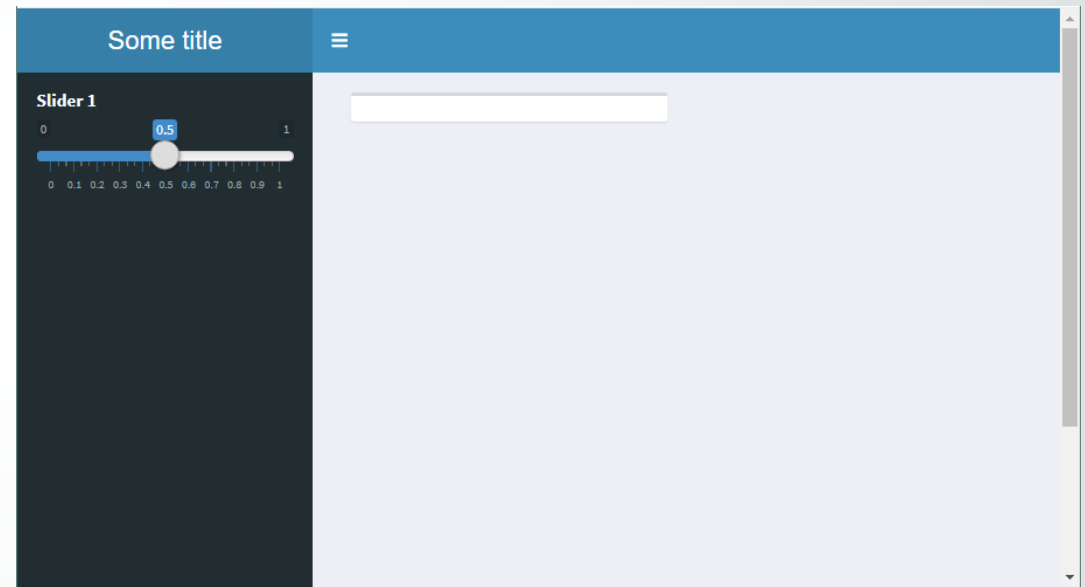
Making nicer UI with **shinydashboard**

```
ui <- dashboardPage(  
  dashboardHeader(title = "Some title"),  
  dashboardSidebar(  
  
  ),  
  dashboardBody(  
  
  )  
)
```



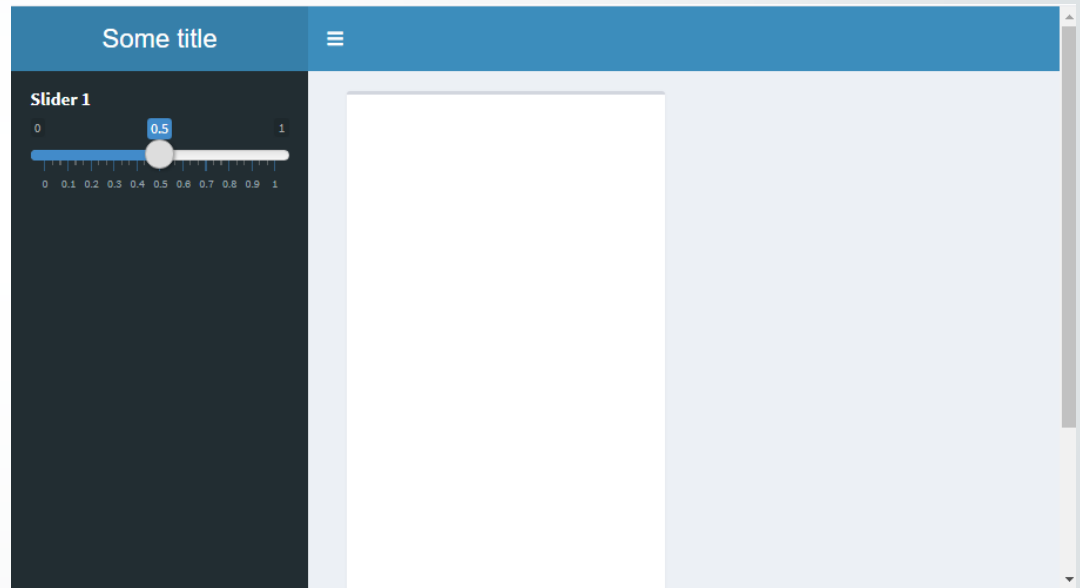
Making nicer UI with **shinydashboard**

```
ui <- dashboardPage(  
  dashboardHeader(title = "Some title"),  
  dashboardSidebar(  
    sliderInput("slider1", "Slider 1", 0, 1, 0.5)  
  ),  
  dashboardBody(  
  )  
)
```



Making nicer UI with **shinydashboard**

```
ui <- dashboardPage(  
  dashboardHeader(title = "Some title"),  
  dashboardSidebar(  
    sliderInput("slider1", "Slider 1", 0, 1, 0.5)  
  ),  
  dashboardBody(  
    box(  
      plotOutput(outputId = "modelPlot")  
    )  
  )  
)
```



More **profesional-looking Shiny apps**

shinydashboardPlus

<https://rinterface.com/shiny/shinydashboardPlus/>

Example: 5_shinydashboard.R

More Widgets

More Widgets

actionButton

```
actionButton("button", "An action button")
```

A rectangular button with rounded corners and a thin border, containing the text "An action button".

numericInput

```
numericInput("num", "Number one", value = 0)
```

Number one

A numeric input field with a light gray background and a thin border. It contains the number "0" and has small up and down arrow buttons on the right side.

selectInput

```
selectInput("dataset", "Choose a dataset:",  
            choices = c("rock", "pressure", "cars"))
```

Choose a dataset:

rock



rock

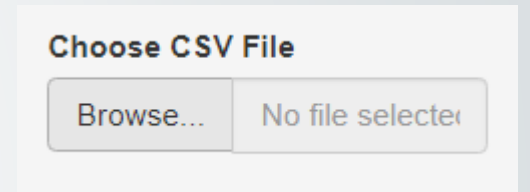
pressure

cars

More Widgets

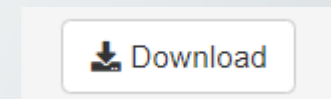
fileInput

```
fileInput("file1", "Choose CSV File", multiple = TRUE,  
         accept = c("text/csv", "text/comma-separated-values,text/plain", ".csv"))
```



downloadButton

```
downloadButton("downloadData", "Download")
```

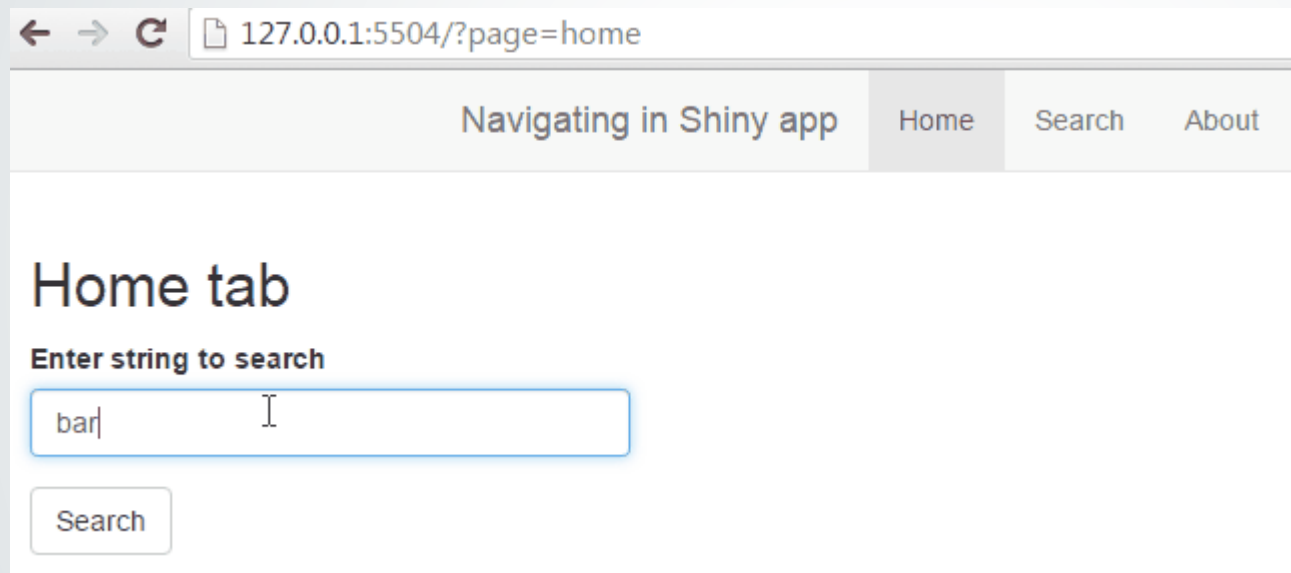


... and more widgets

Enter an R expression

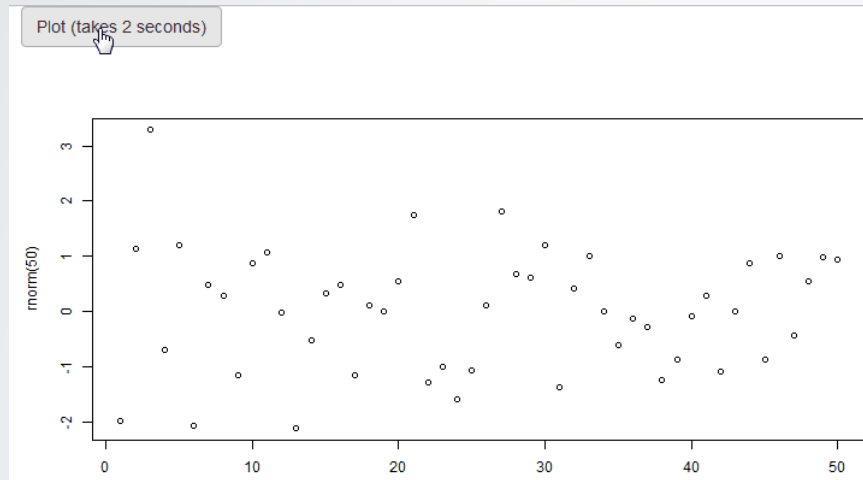
... and more widgets



A screenshot of a web browser displaying a Shiny application. The browser's address bar shows the URL `127.0.0.1:5504/?page=home`. The application has a navigation bar with the title "Navigating in Shiny app" and three tabs: "Home", "Search", and "About". The "Home" tab is currently selected. Below the navigation bar, the main content area is titled "Home tab". It contains a text input field with the placeholder text "Enter string to search". The input field has the text "bar" entered and a cursor at the end. Below the input field is a "Search" button.

<https://github.com/daattali/advanced-shiny#shinyjs>

... and more widgets

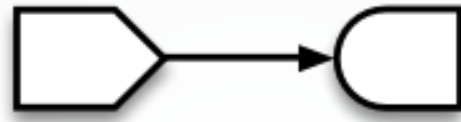


Loading...

The server ...more than an R code

The server ...more than an R code

UI

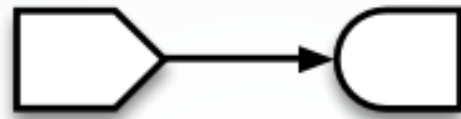


Server



The server ...more than an R code

UI

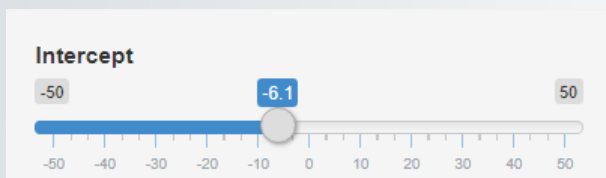


Server

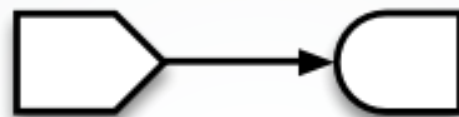


The server ...more than an R code

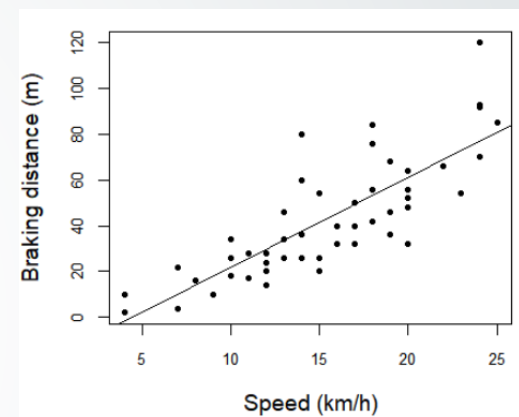
UI



```
sliderInput(inputId="intercept" [...] )
```



Server



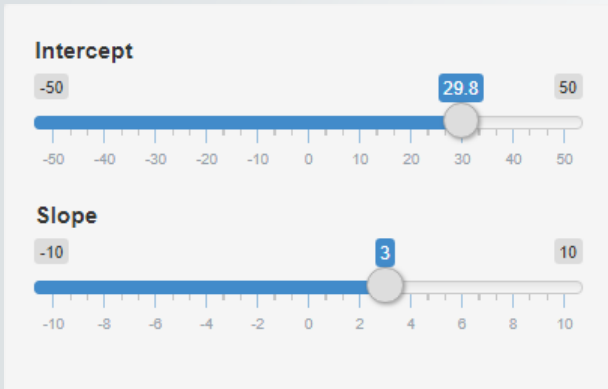
input\$intercept → renderPlot

Shiny creates an object:

```
input  
[[1]] "intercept" = [selected value]
```

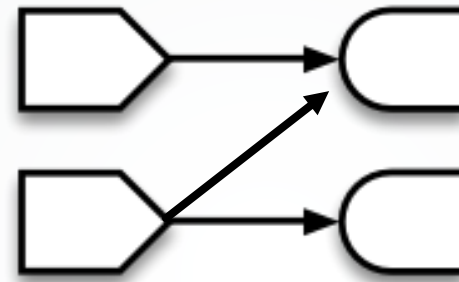
The server ...more than an R code

UI



```
sliderInput(inputId="intercept" [...])
```

```
sliderInput(inputId="slope" [...])
```



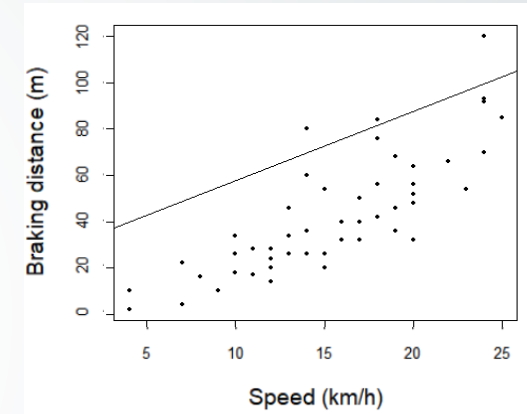
Shiny creates an object:

input

[[1]] "intercept" = [selected value]

[[2]] "slope" = [selected value]

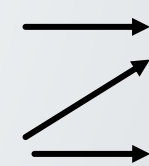
Server



slope = 3.0

input\$intercept

input\$slope



renderPlot

renderText

The server

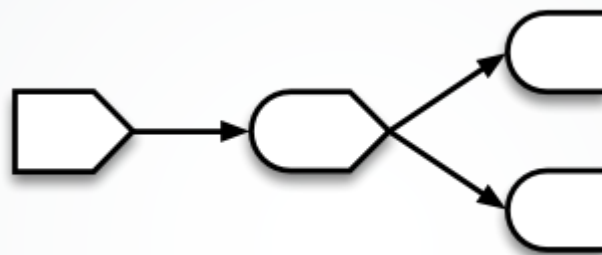
Reactive functions and values

The server

Reactive functions and values

UI

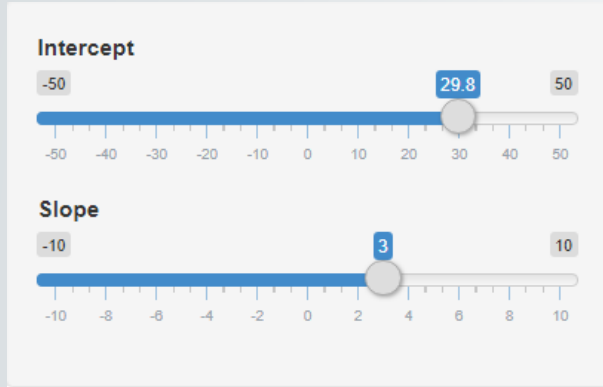
Server



The server

Reactive functions and values

UI



The user updates a parameter in the UI

```
sliderInput(inputId="intercept" [...])
```

```
sliderInput(inputId="slope" [...])
```

Shiny creates/modifies the "input" object

input

```
[[1]] "intercept" = [selected value]
```

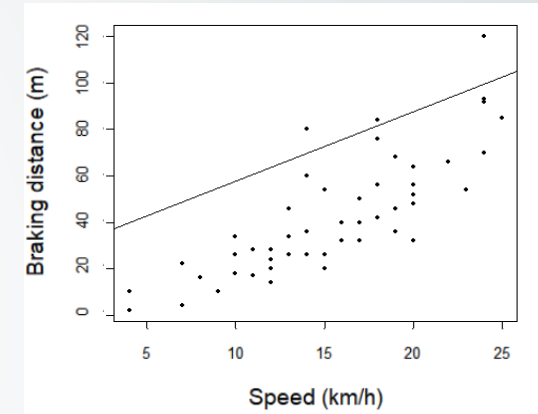
```
[[2]] "slope" = [selected value]
```

The "observer" detects what parameter has changed

```
reactive({ [some function using the "input" object] })
```



Server



slope = 3.0

The output is updated only
running the code involving this parameter

The server

Reactive functions and values

Example: modeling body temperature of lizards

The server

Reactive functions and values

```
server <- function(input, output) {
```

```
  stored_values <- reactiveValues( store all simulated data )
```

```
  Location_xy <- observeEvent( observe LATITUDE / LONGITUDE values when the user clicks on the map )
```

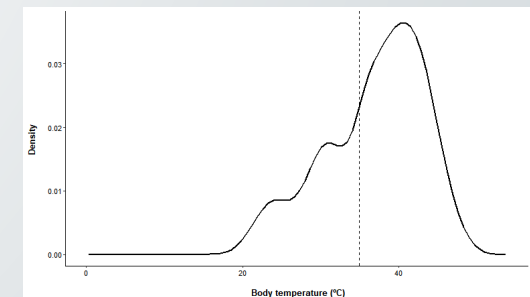
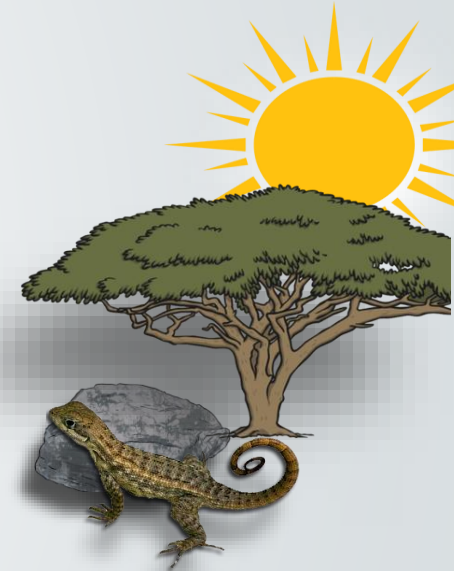
```
  model_microclimates <- reactive({ run a function to compute ambient temperature in the sun and shade })
```

```
  model_body_temperature <- reactive({ run a function to simulate thermoregulating lizards })
```

```
  generate_temperature_distribution <- reactive({ derive probability distribution of body temperatures })
```

```
  output$Tb_distribution <- renderPlot({  
    model_microclimates()  
    model_body_temperature()  
    generate_temperature_distribution()  Make the plot  
  })
```

```
}
```



The server

observeEvent



The server

observeEvent

UI



`actionButton("press_button")`

Server



`observeEvent(input$press_button { do something })`



The server

observeEvent

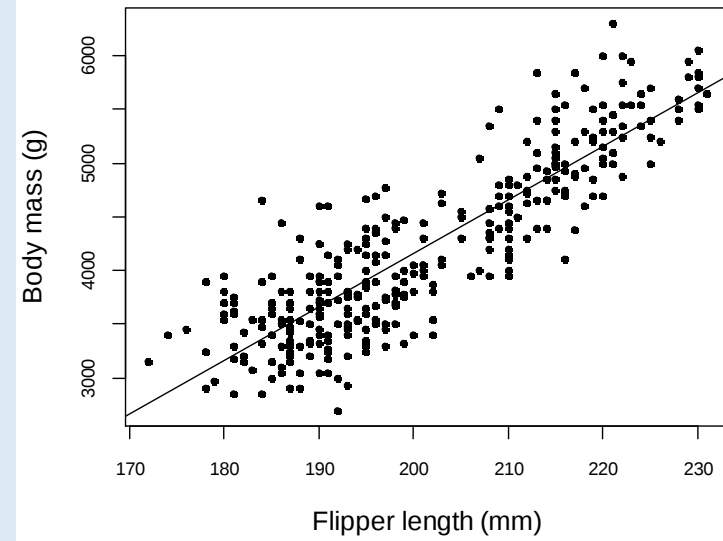
Intercept

-50 11.9 50

Slope

-10 3 10

Fit linear model



Example: 7_Observe_event.R

The server

observeEvent

```
ui <- dashboardPage(  
  dashboardHeader(title = " Some title"),  
  
  dashboardSidebar(  
  
  ),  
  
  dashboardBody(  
    box(  
  
    )  
  )  
)
```

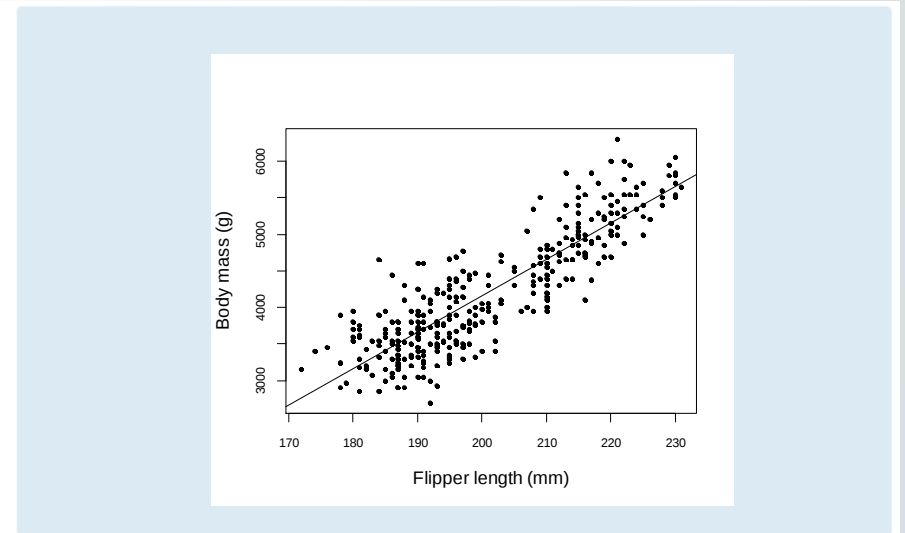
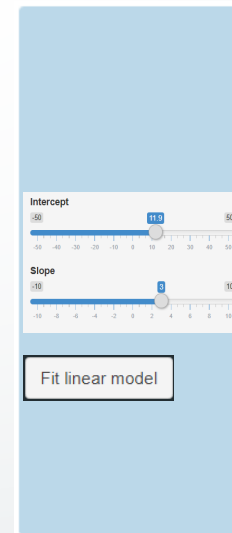
Input parameters, sliders, action buttons....

What are we going to show (plots, tables...)

The server

observeEvent

```
ui <- dashboardPage(  
  dashboardHeader(title = " Some title"),  
  
  dashboardSidebar(  
    sliderInput(inputId = "intercept", label = "Intercept", min = -50, max = 50, value = 0, step = 0.1),  
    sliderInput(inputId = "slope", label = "Slope", min = -10, max = 10, value = 3, step = 0.1),  
    actionButton("fit_linear_model", "Fit linear model")  
  ),  
  
  dashboardBody(  
    box(  
      plotOutput(outputId = "scatterplot")  
    )  
  )  
)
```



The server

observeEvent

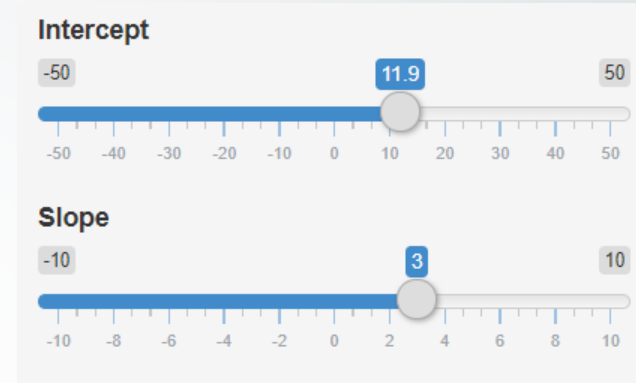
```
server <- function(input, output) {  
  
  stored_parameters <- reactiveValues(  
    intercept = NULL,  
    slope = NULL  
  )  
  
  observeEvent(list(input$intercept, input$slope), {  
    stored_parameters$intercept <- input$intercept  
    stored_parameters$slope <- input$slope  
  })  
  
  observeEvent(input$fit_linear_model, {  
    mod <- lm(body_mass_g ~ flipper_length_mm, penguins)  
    stored_parameters$intercept <- coef(mod)[1]  
    stored_parameters$slope <- coef(mod)[2]  
  })  
  
  output$scatterplot <- renderPlot({  
    plot(dist ~ speed, cars)  
    abline(a = stored_parameters$intercept, b = stored_parameters$slope)  
  })  
}
```

Create an object to **store parameter values**

The server

observeEvent

```
server <- function(input, output) {  
  
  stored_parameters <- reactiveValues(  
    intercept = NULL,  
    slope = NULL  
  )  
  
  observeEvent(list(input$intercept, input$slope), {  
    stored_parameters$intercept <- input$intercept  
    stored_parameters$slope <- input$slope  
  })  
  
  observeEvent(input$fit_linear_model, {  
    mod <- lm(body_mass_g ~ flipper_length_mm, penguins)  
    stored_parameters$intercept <- coef(mod)[1]  
    stored_parameters$slope <- coef(mod)[2]  
  })  
  
  output$scatterplot <- renderPlot({  
    plot(dist ~ speed, cars)  
    abline(a = stored_parameters$intercept, b = stored_parameters$slope)  
  })  
}
```



If the user **moves the sliders**,
update the stored parameter values

The server

observeEvent

```
server <- function(input, output) {  
  
  stored_parameters <- reactiveValues(  
    intercept = NULL,  
    slope = NULL  
  )  
  
  observeEvent(list(input$intercept, input$slope), {  
    stored_parameters$intercept <- input$intercept  
    stored_parameters$slope <- input$slope  
  })  
  
  observeEvent(input$fit_linear_model, {  
    mod <- lm(body_mass_g ~ flipper_length_mm, penguins)  
    stored_parameters$intercept <- coef(mod)[1]  
    stored_parameters$slope <- coef(mod)[2]  
  })  
  
  output$scatterplot <- renderPlot({  
    plot(dist ~ speed, cars)  
    abline(a = stored_parameters$intercept, b = stored_parameters$slope)  
  })  
}
```

If the user **presses the button**,
compute the linear model and
update the stored parameter values

Fit linear model

The server

observeEvent

```
server <- function(input, output) {  
  
  stored_parameters <- reactiveValues(  
    intercept = NULL,  
    slope = NULL  
  )  
  
  observeEvent(list(input$intercept, input$slope), {  
    stored_parameters$intercept <- input$intercept  
    stored_parameters$slope <- input$slope  
  })  
  
  observeEvent(input$fit_linear_model, {  
    mod <- lm(body_mass_g ~ flipper_length_mm, penguins)  
    stored_parameters$intercept <- coef(mod)[1]  
    stored_parameters$slope <- coef(mod)[2]  
  })  
  
  output$scatterplot <- renderPlot({  
    plot(dist ~ speed, cars)  
    abline(a = stored_parameters$intercept, b = stored_parameters$slope)  
  })  
}
```

The server

observeEvent

```
server <- function(input, output) {  
  
  stored_parameters <- reactiveValues(  
    intercept = NULL,  
    slope = NULL  
  )  
  
  observeEvent(list(input$intercept, input$slope), {  
    stored_parameters$intercept <- input$intercept  
    stored_parameters$slope <- input$slope  
  })  
  
  observeEvent(input$fit_linear_model, {  
    mod <- lm(body_mass_g ~ flipper_length_mm, penguins)  
    stored_parameters$intercept <- coef(mod)[1]  
    stored_parameters$slope <- coef(mod)[2]  
  })  
  
  output$scatterplot <- renderPlot({  
    plot(dist ~ speed, cars)  
    abline(a = stored_parameters$intercept, b = stored_parameters$slope)  
  })  
}
```

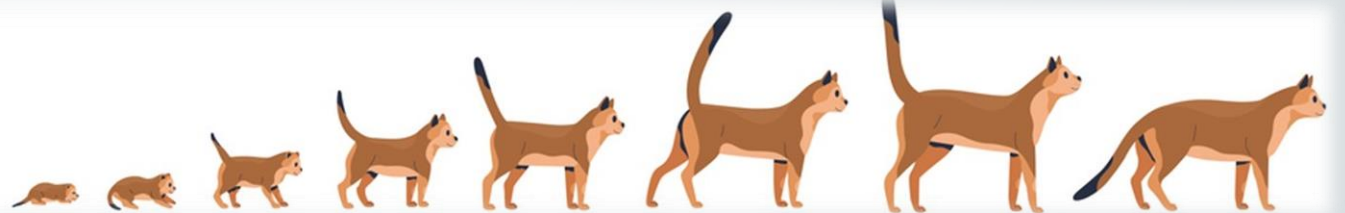
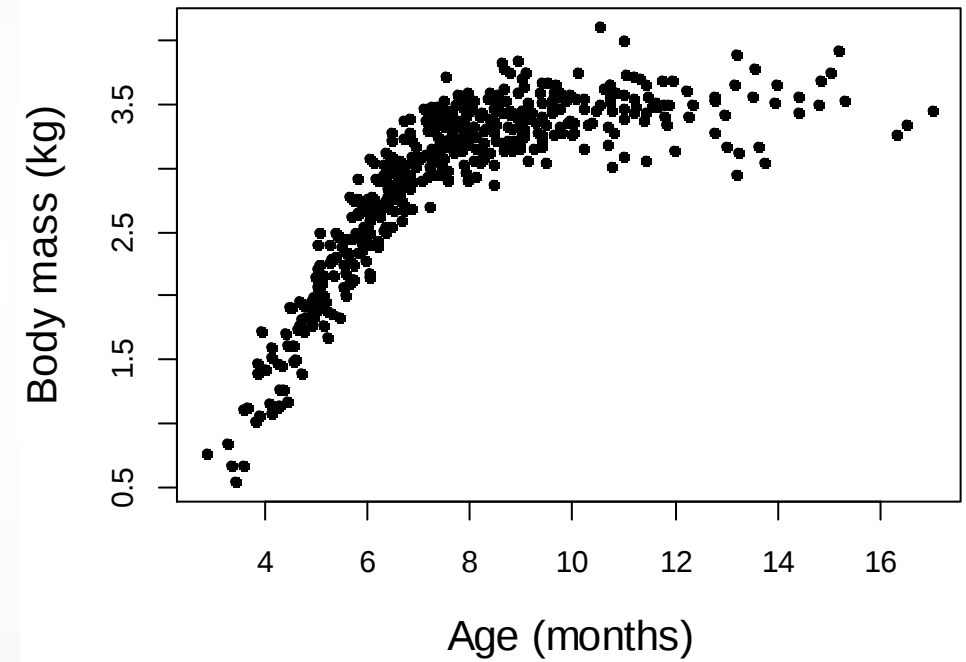
Fitting **nonlinear** models

observeEvent



Fitting **nonlinear** models

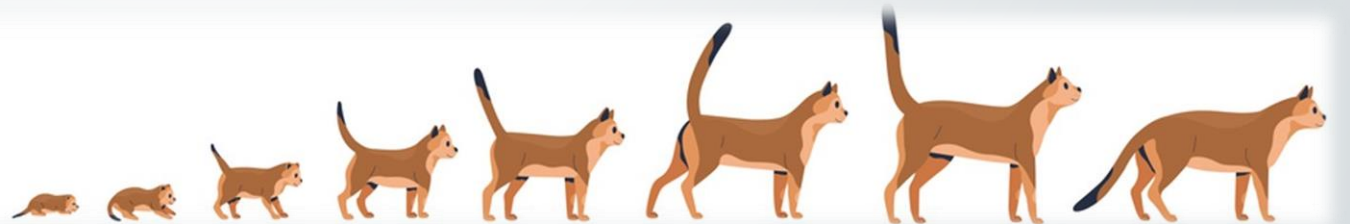
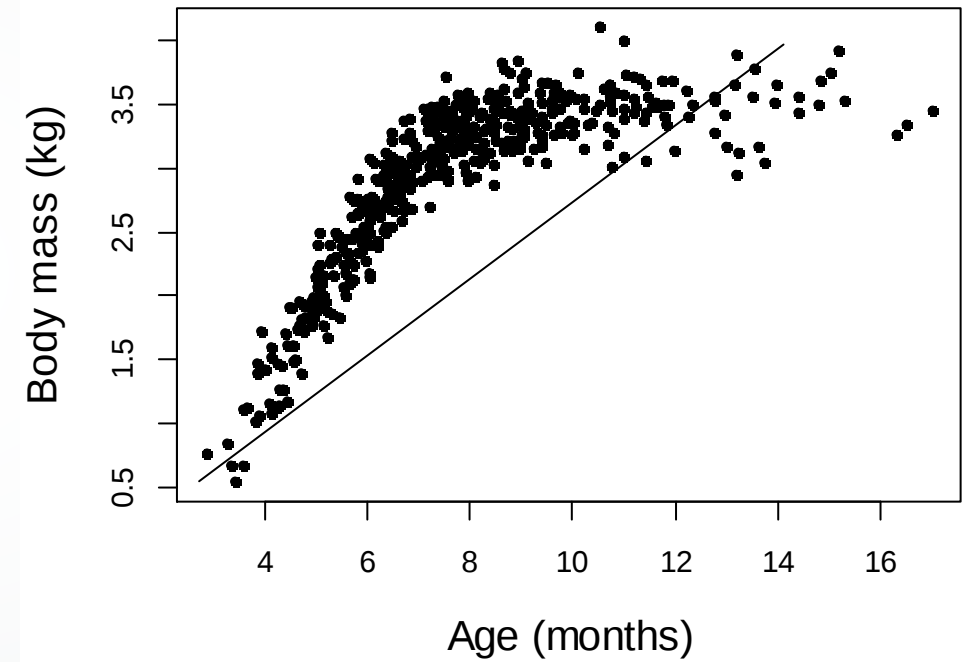
observeEvent



Fitting **nonlinear** models

observeEvent

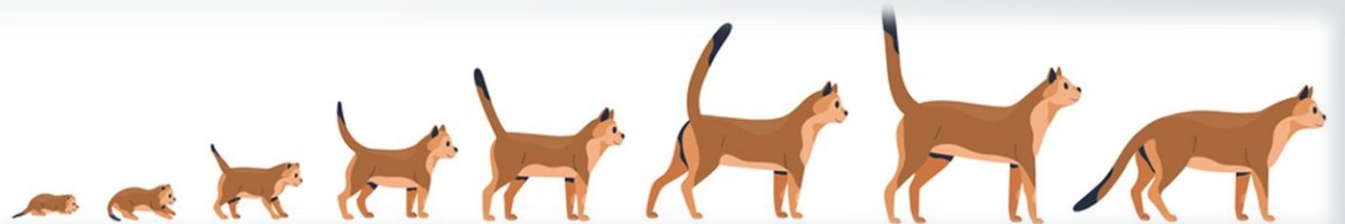
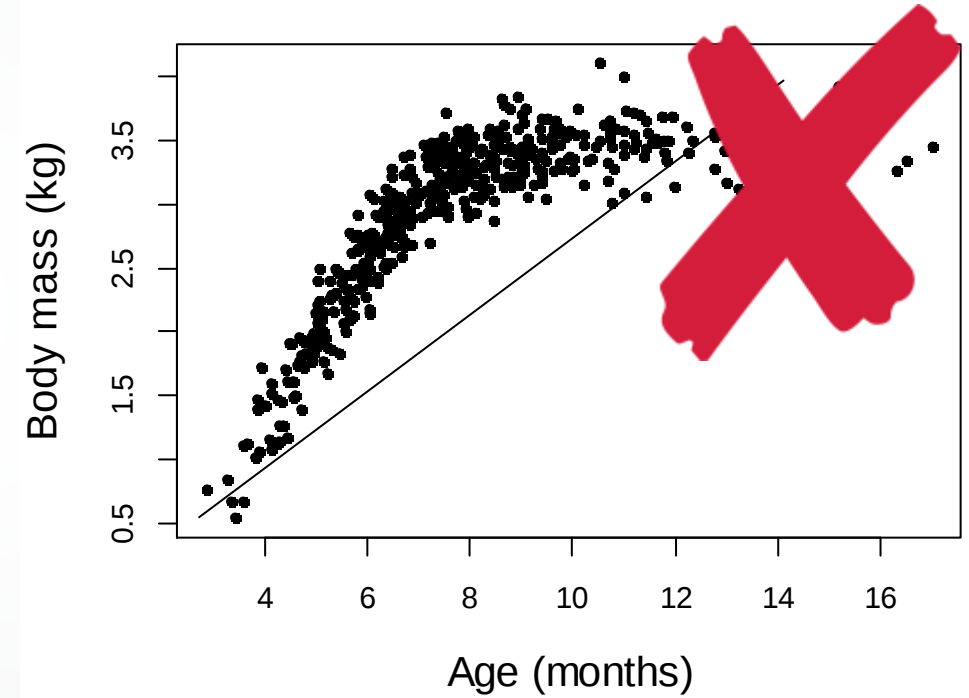
$$mass = a + b \times age$$



Fitting **nonlinear** models

observeEvent

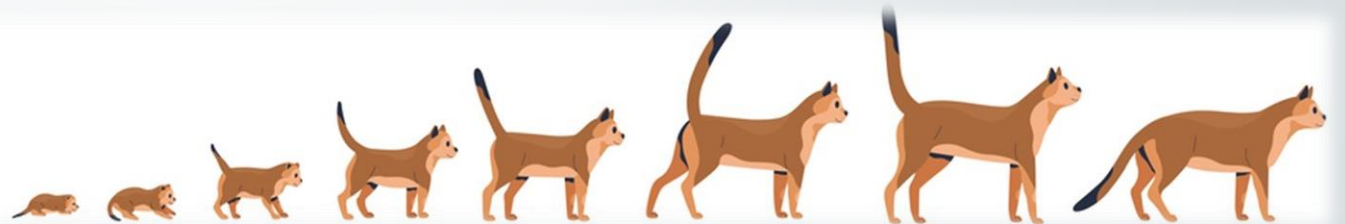
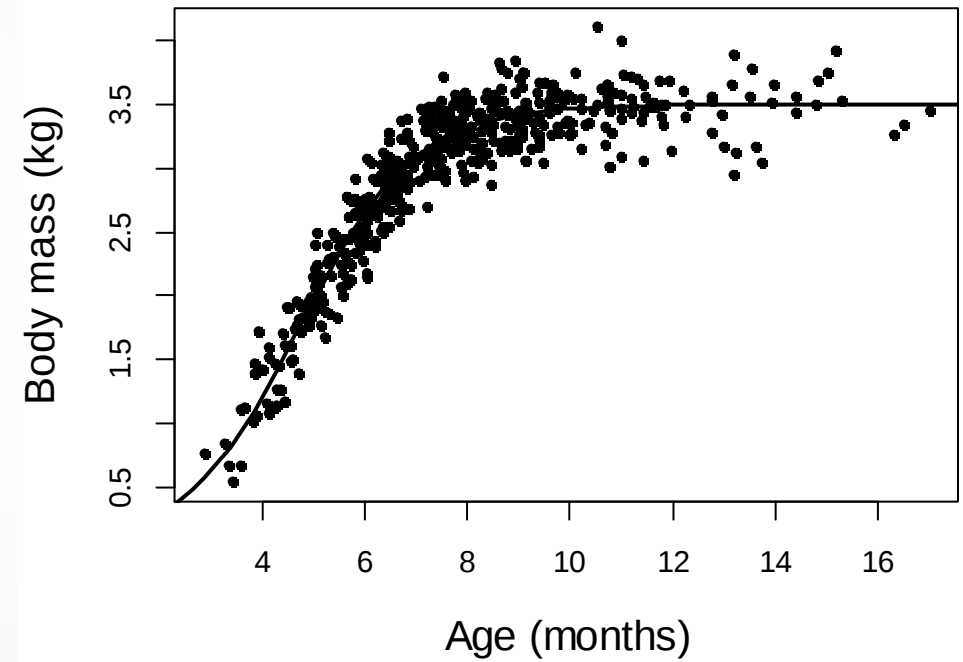
$$mass = a + b \times age$$



Fitting **nonlinear** models

observeEvent

$$mass = \frac{c}{1 + e^{a-b \times age}}$$



Fitting **nonlinear models**

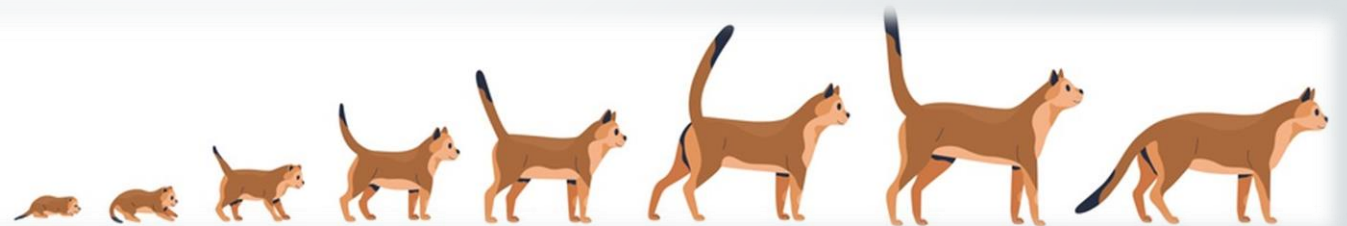
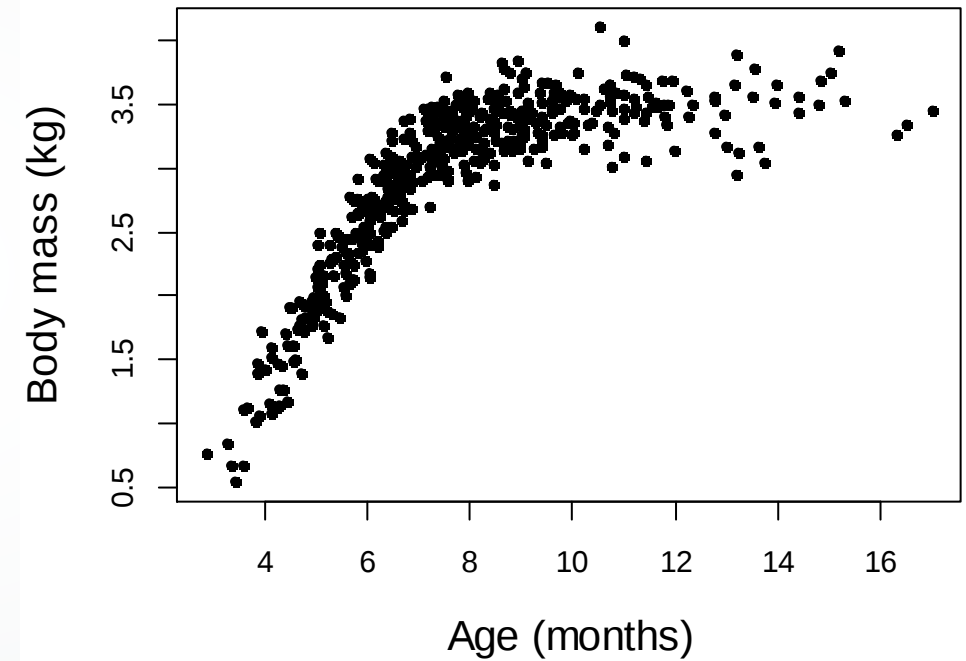
observeEvent

```
logistic_model <- function(a, b, c, x){  
  y <- c / (1 + exp(a - b * x))  
  return(y)  
}
```

How can we **fit a nonlinear model**?

=

what are the values for a, b, and c?

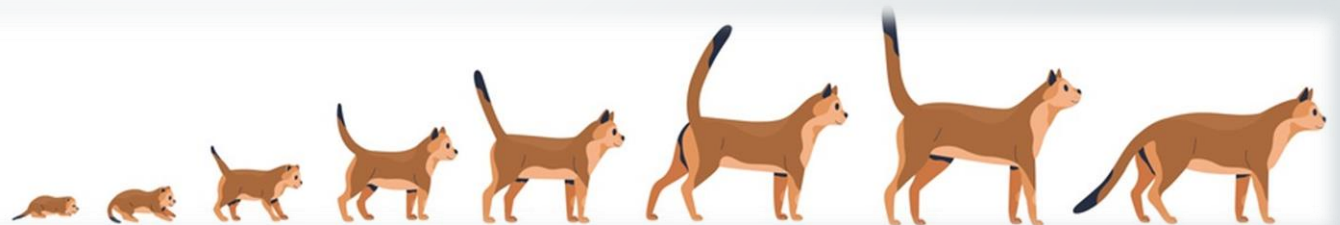
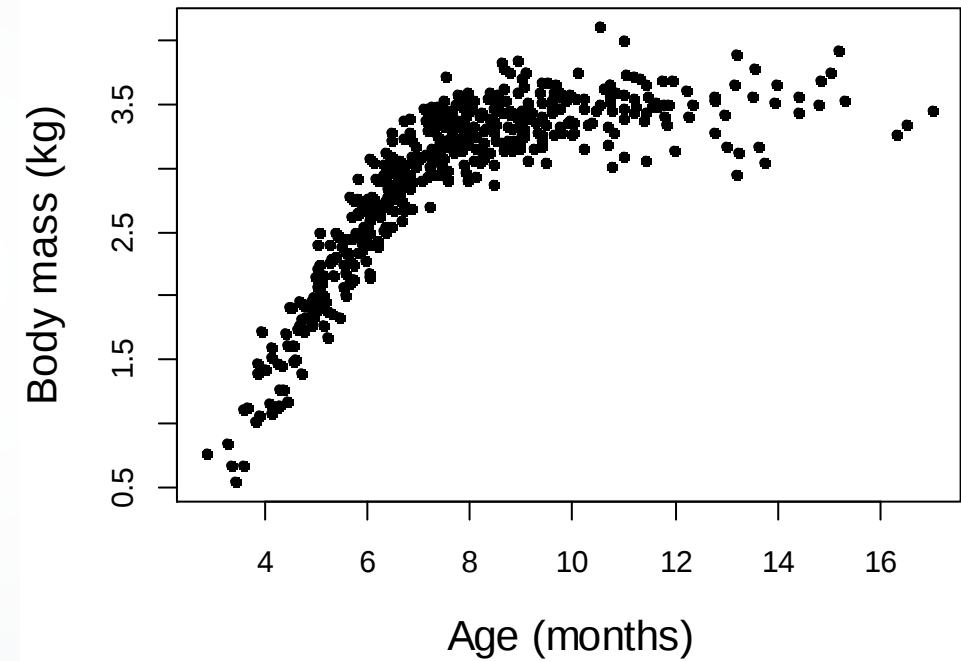


Fitting **nonlinear** models

observeEvent

```
logistic_model <- function(a, b, c, x){  
  y <- c / (1 + exp(a - b * x))  
  return(y)  
}
```

```
mod <- nls2(mass ~ logistic_model(a, b, c, age),  
  start = list(a = 2, b = 0.9, c = 2))
```



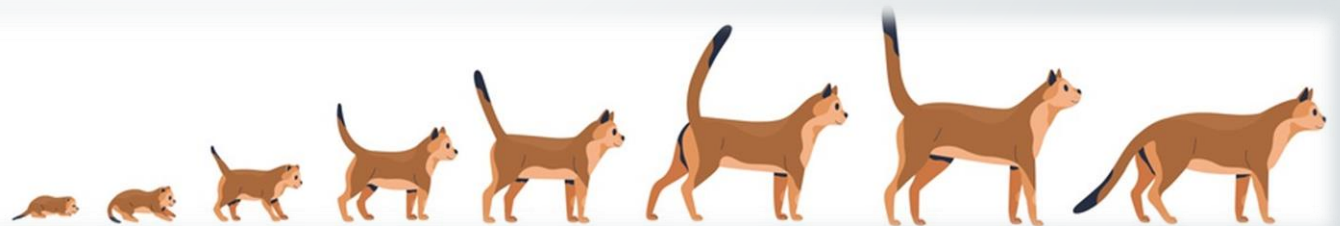
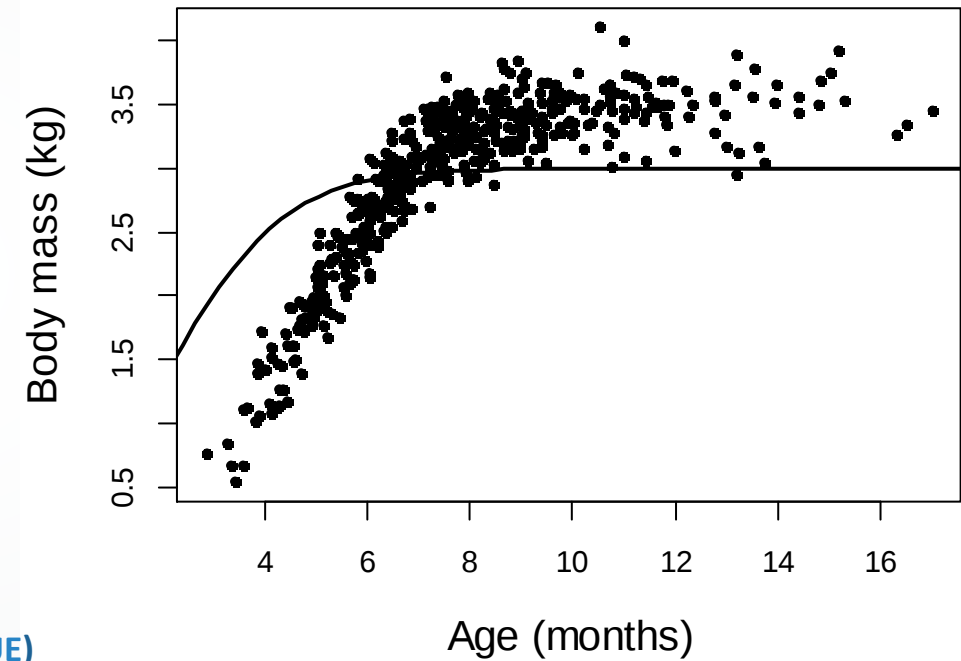
Fitting **nonlinear** models

observeEvent

```
logistic_model <- function(a, b, c, x){  
  y <- c / (1 + exp(a - b * x))  
  return(y)  
}
```

```
a <- 2  
b <- 0.9  
c <- 3
```

```
curve(logistic_model(a = a, b = b, c = c, x), from = 0, to = 20, add = TRUE)
```



Fitting **nonlinear models**

observeEvent

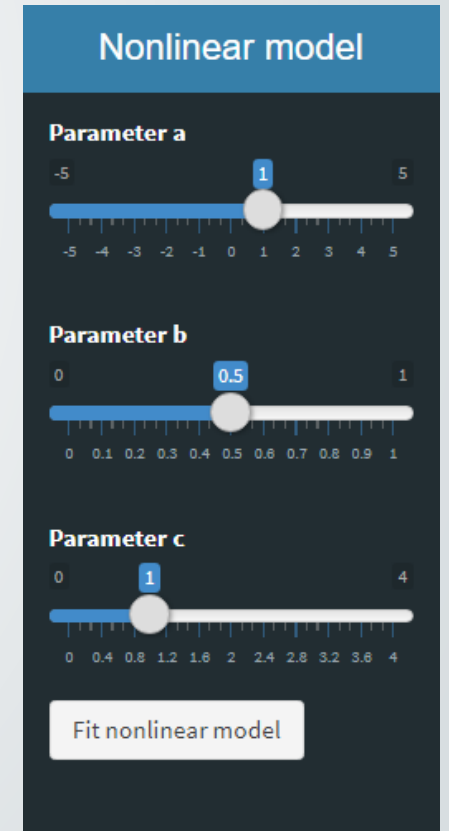
Help us, Shiny!

Example: 8_Fitting_nls.R

Fitting **nonlinear models**

observeEvent

```
ui <- dashboardPage(  
  dashboardHeader(title = " Nonlinear model"),  
  
  dashboardSidebar(  
    sliderInput(inputId = "a", label = "Parameter a", min = -5, max = 5, value = 1, step = 0.01),  
    sliderInput(inputId = "b", label = "Parameter b", min = 0, max = 1, value = 0.5, step = 0.01),  
    sliderInput(inputId = "c", label = "Parameter c", min = 0, max = 4, value = 1, step = 0.01),  
    actionButton("fit_nls", "Fit nonlinear model")  
  ),  
  
  dashboardBody(  
    box(  
      plotOutput(outputId = "scatterplot")  
    )  
  )  
)
```

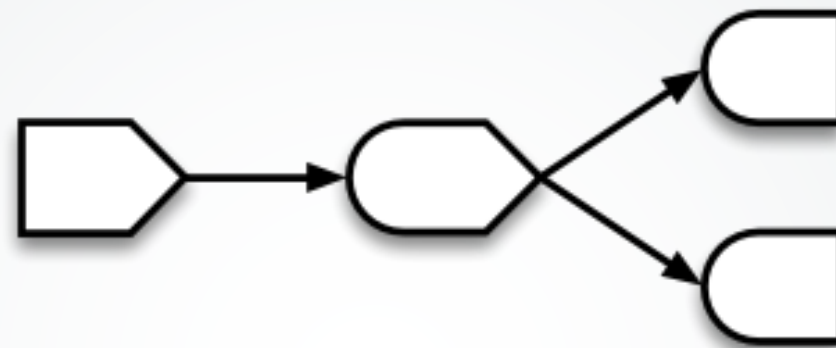


Fitting **nonlinear models**

observeEvent

```
server <- function(input, output) {  
  stored_parameters <- reactiveValues(  
    a = NULL,  
    b = NULL,  
    c = NULL  
  )  
  observeEvent(list(input$a, input$b, input$c), {  
    stored_parameters$a <- input$a  
    stored_parameters$b <- input$b  
    stored_parameters$c <- input$c  
  })  
  
  observeEvent(input$fit_nls, {  
    mod <- nls2(mass ~ logistic_model(a, b, c, age), simulated_data,  
               start = list(a=stored_parameters$a, b=stored_parameters$b, c=stored_parameters$c))  
    stored_parameters$a <- coef(mod)[1]  
    stored_parameters$b <- coef(mod)[2]  
    stored_parameters$c <- coef(mod)[3]  
  })  
  
  output$scatterplot <- renderPlot({  
    plot(mass ~ age, simulated_data, pch = 20, col = "grey")  
    curve(logistic_model(stored_parameters$a, stored_parameters$b, stored_parameters$c, x), 0, 24, lwd = 2, add = T)  
  })  
}
```

The server



UI

input

reactiveValues()



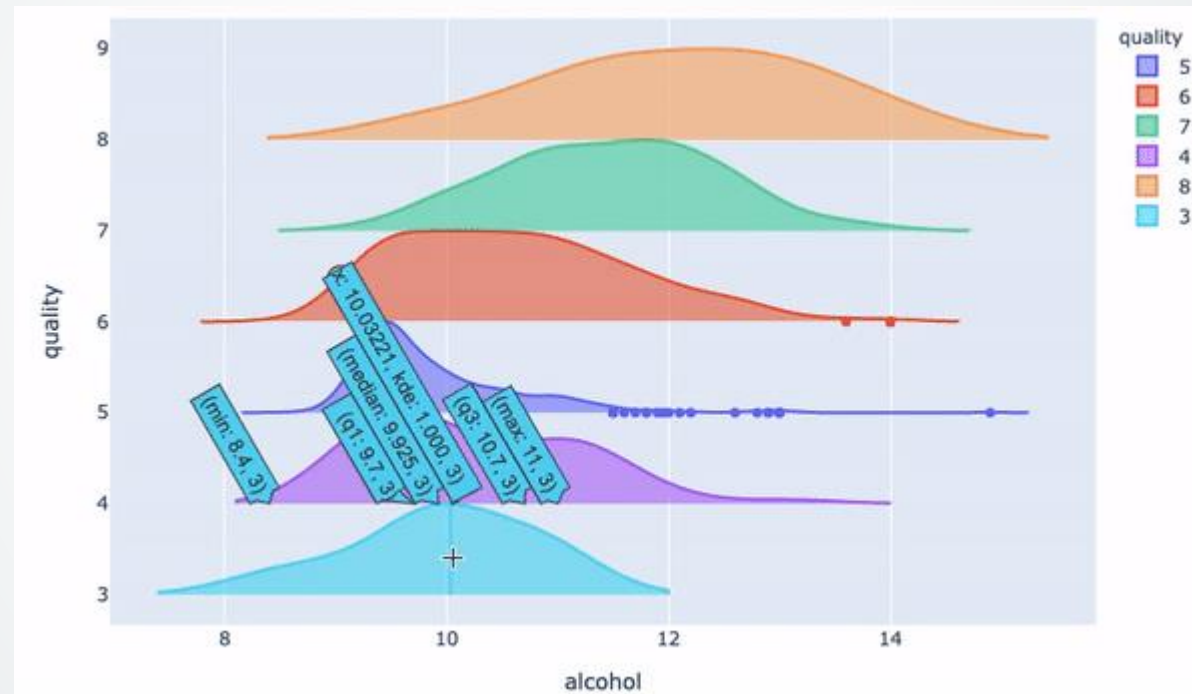
reactive()

Output

Making Shiny **more** interactive

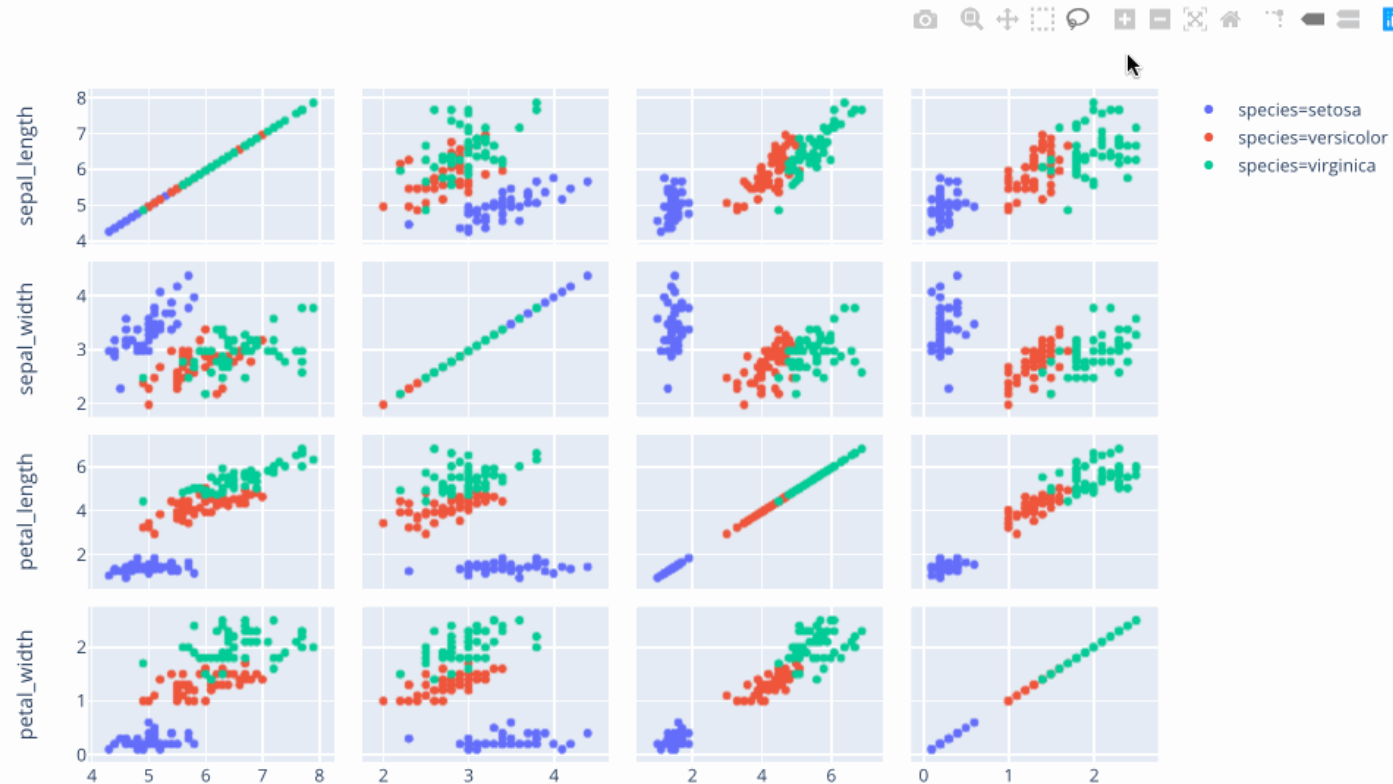


Making Shiny **more** interactive

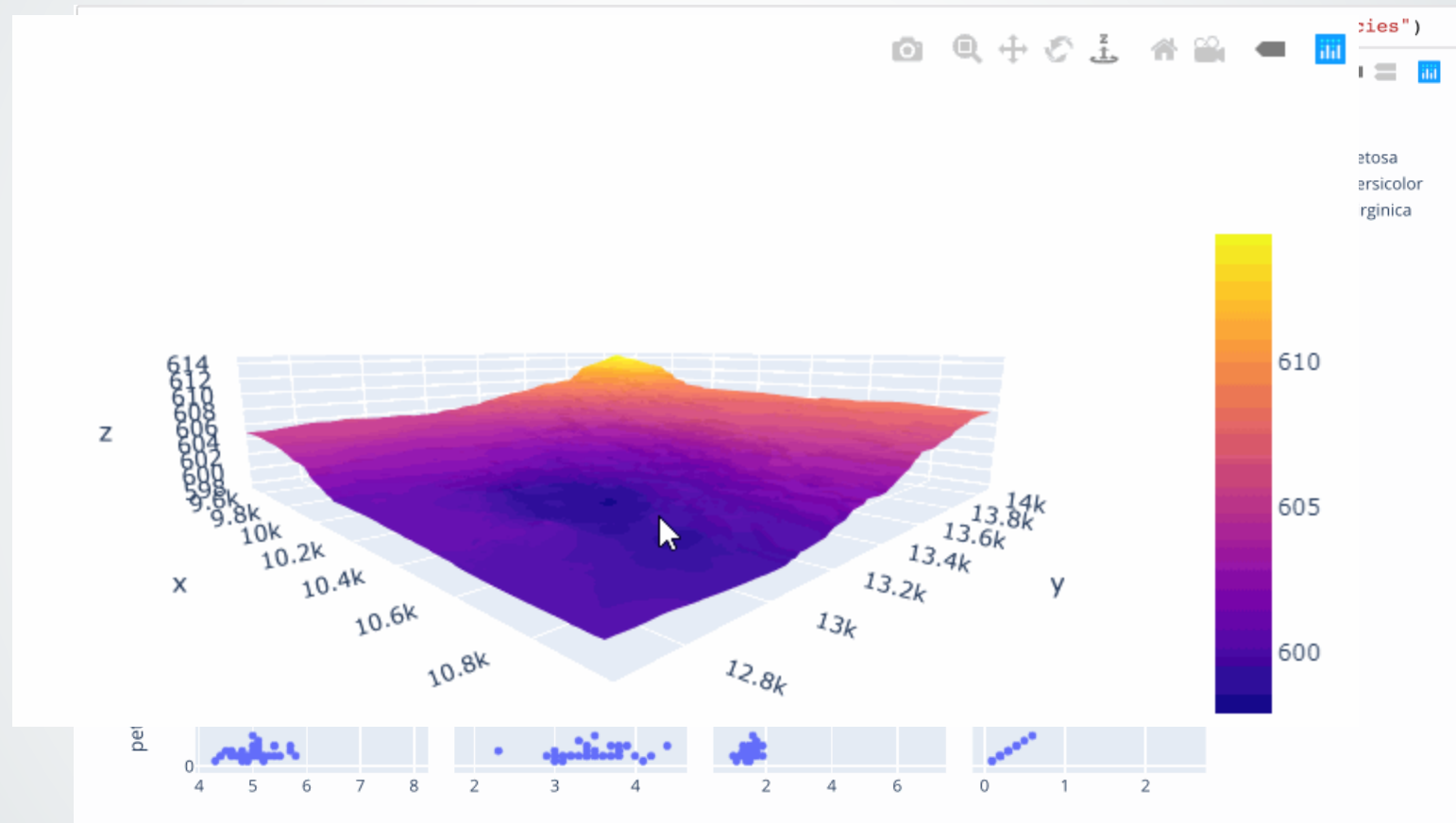


Making Shiny **more** interactive

```
px.scatter_matrix(iris, dimensions=["sepal_width", "sepal_length", "petal_width", "petal_length"], color="species")
```



Making Shiny **more** interactive



Making Shiny **more** interactive

```
ui <- fluidPage(  
  sidebarLayout(  
    sidebarPanel(  
      
```

Input parameters, sliders, action buttons....

```
    ),  
    mainPanel(  
      
```

What are we going to show (plots, tables...)

plotlyOutput("plot")

```
    )  
  )  
)
```

```
server <- function(input, output, session) {  
  output$scatterplot <- renderPlot({  
    
```

R code for your plots

output\$plot <- renderPlotly({ })

```
  })  
}
```

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

Making Shiny **more** interactive



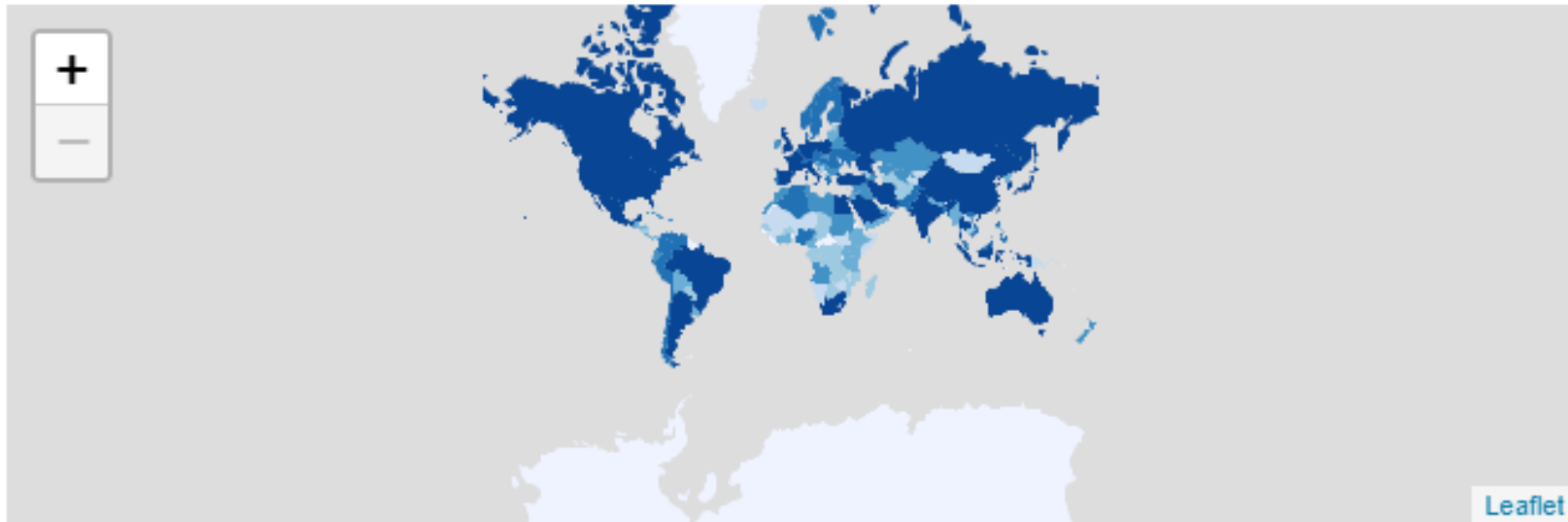
```
m <- leaflet() %>% setView(lng = -71.0589, lat = 42.3601, zoom = 12)
m %>% addTiles()
```



Making Shiny **more** interactive



```
qpal <- colorQuantile("Blues", countries$gdp_md_est, n = 7)
map %>%
  addPolygons(stroke = FALSE, smoothFactor = 0.2, fillOpacity = 1,
    color = ~qpal(gdp_md_est))
```



Launching **Shiny** Apps



Launching **Shiny Apps**

```
library(rsconnect)
```

```
rsconnect::deployApp("directory in your computer")
```

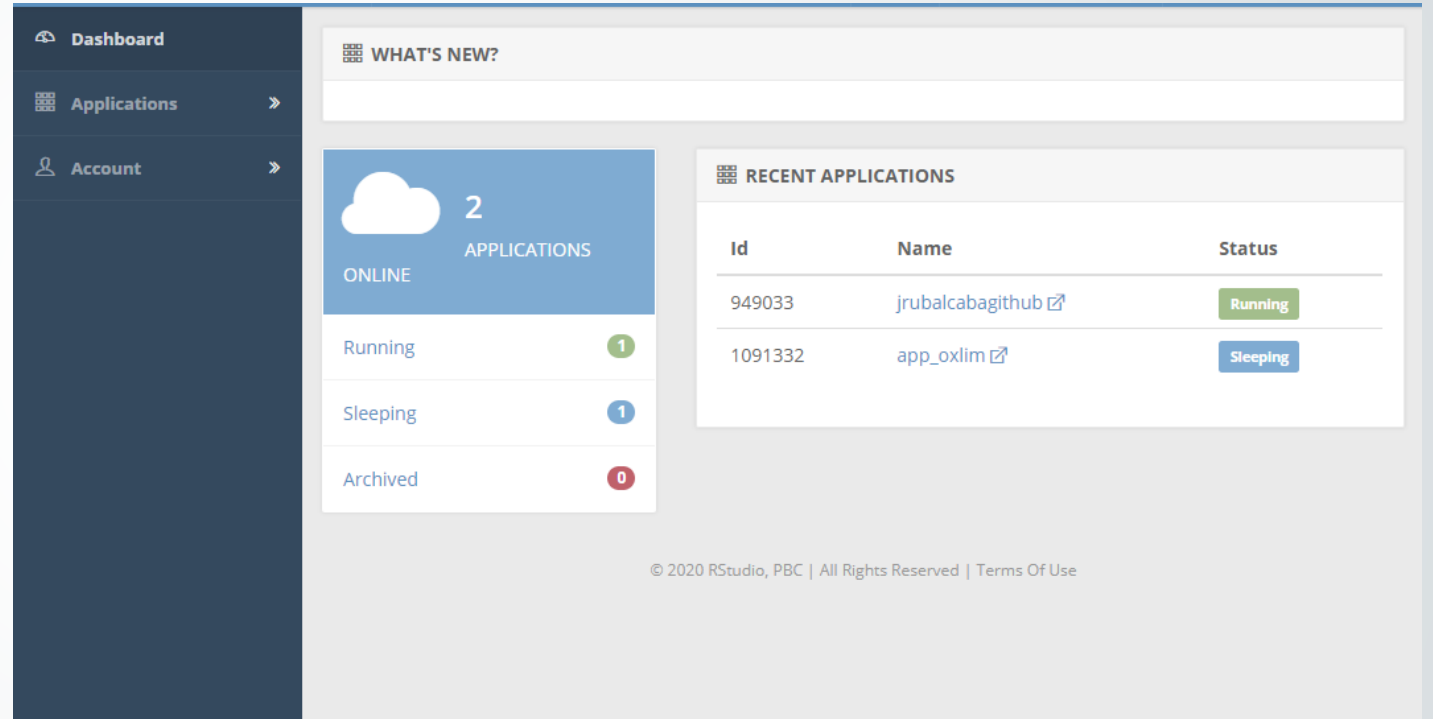
* make sure that your app R file is named "app.R"

Shinyapps.io

<https://shiny.rstudio.com/tutorial/written-tutorial/lesson1/>

Launching **Shiny Apps**

Shinyapps.io



The screenshot displays the Shinyapps.io dashboard. On the left is a dark blue sidebar with navigation links: 'Dashboard' (with a cloud icon), 'Applications' (with a grid icon), and 'Account' (with a person icon). The main content area has a light gray background. At the top is a 'WHAT'S NEW?' section. Below it is a blue box labeled 'ONLINE 2 APPLICATIONS'. Under this box is a list of application statuses: 'Running' with a green circle containing '1', 'Sleeping' with a blue circle containing '1', and 'Archived' with a red circle containing '0'. To the right of this is a 'RECENT APPLICATIONS' section containing a table with two rows of application data. At the bottom right of the dashboard is the copyright notice: '© 2020 RStudio, PBC | All Rights Reserved | Terms Of Use'.

Id	Name	Status
949033	jrubalcabagithub	Running
1091332	app_oxlim	Sleeping

Launching **Shiny Apps**

Run from GitHub

Create a GitHub repository and include the file `'app.R'`

```
runGitHub("repo_name")
```

Virtual machine – AWS/RStudio