Shiny Apps

Juan G Rubalcaba

Shiny apps

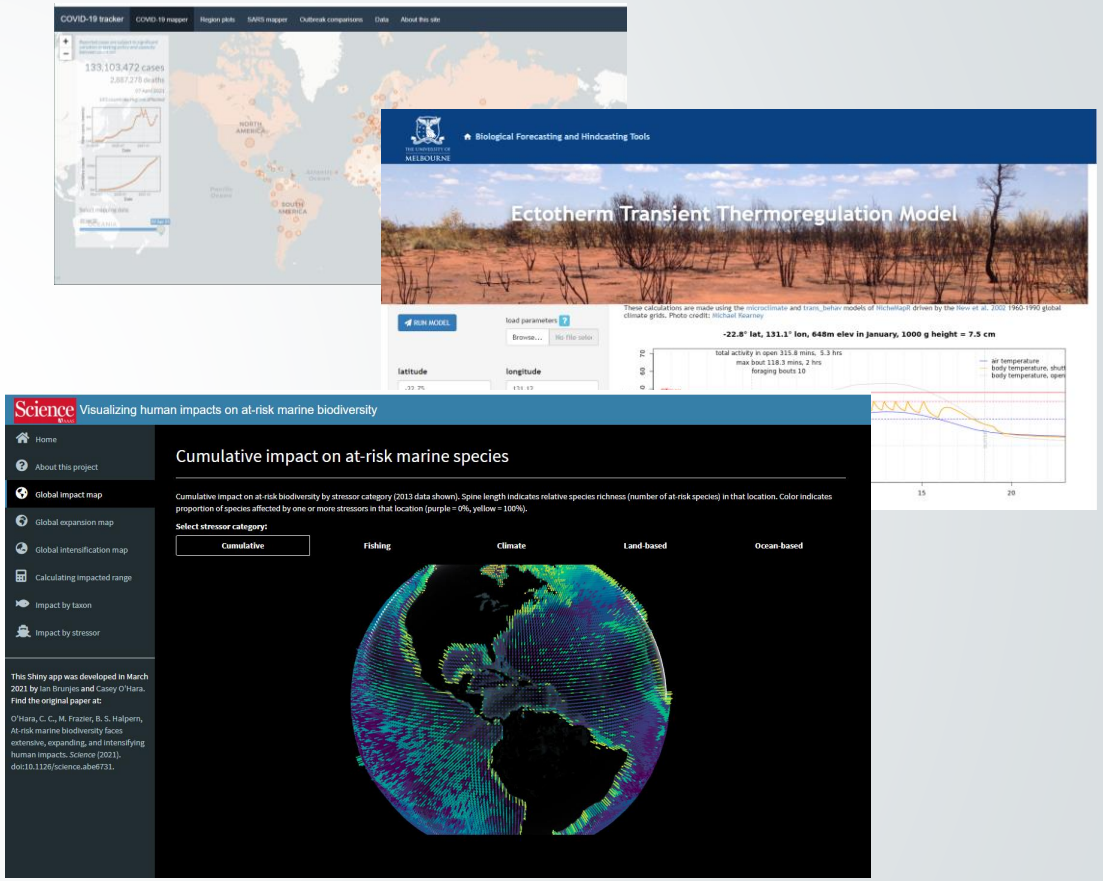
What is a Shiny app? **Gallery**

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

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

Interactive **plots and maps**

Launching Shiny apps



Shiny apps

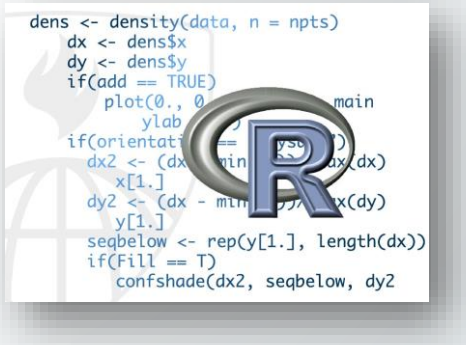
Interactive **web applications** for R

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

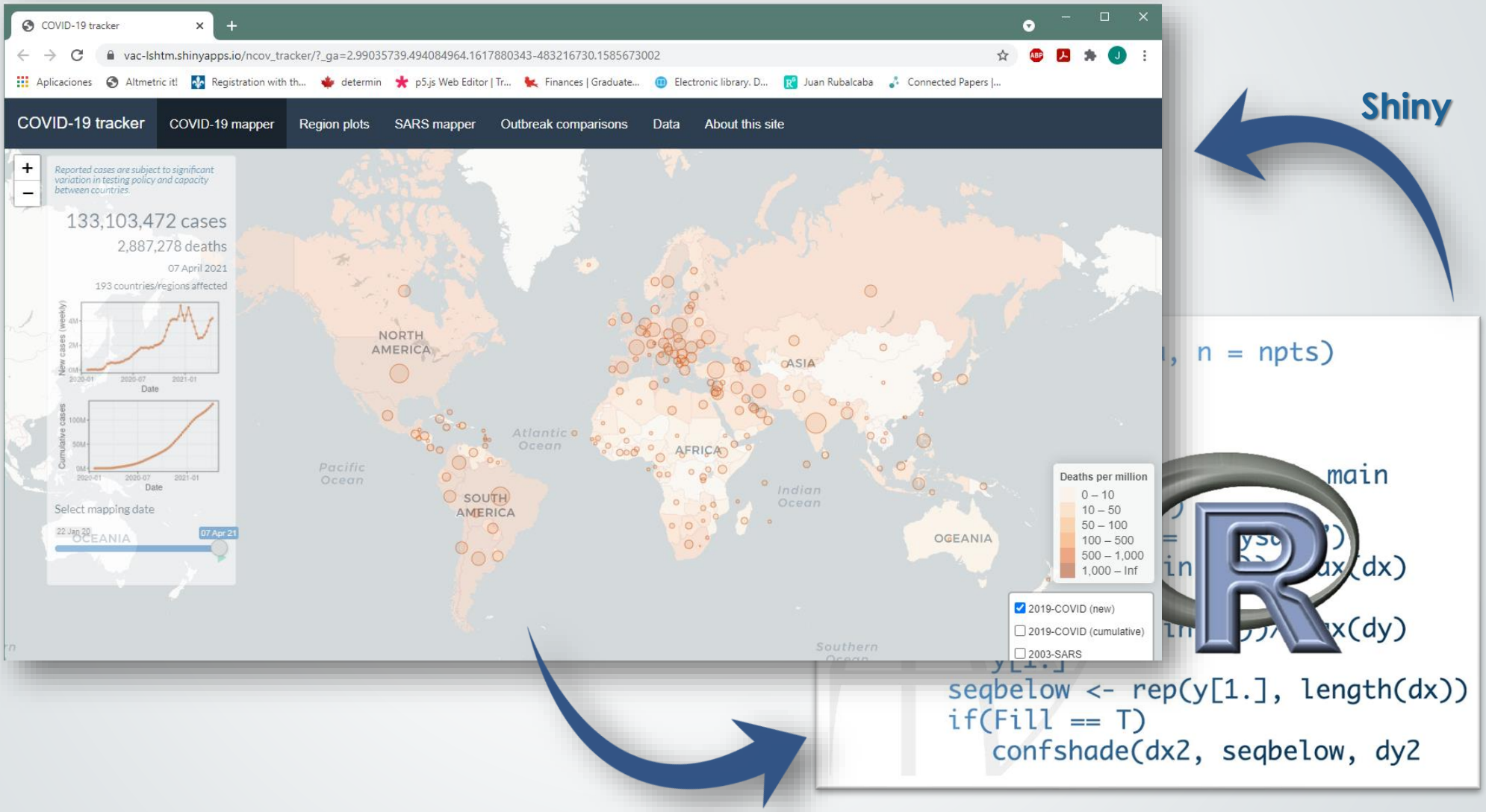
- manage datasets / perform statistics
- data visualization (graphs, maps…)
- simulations

} **Interactively!**

Extensions with HTML, CSS, JavaScript

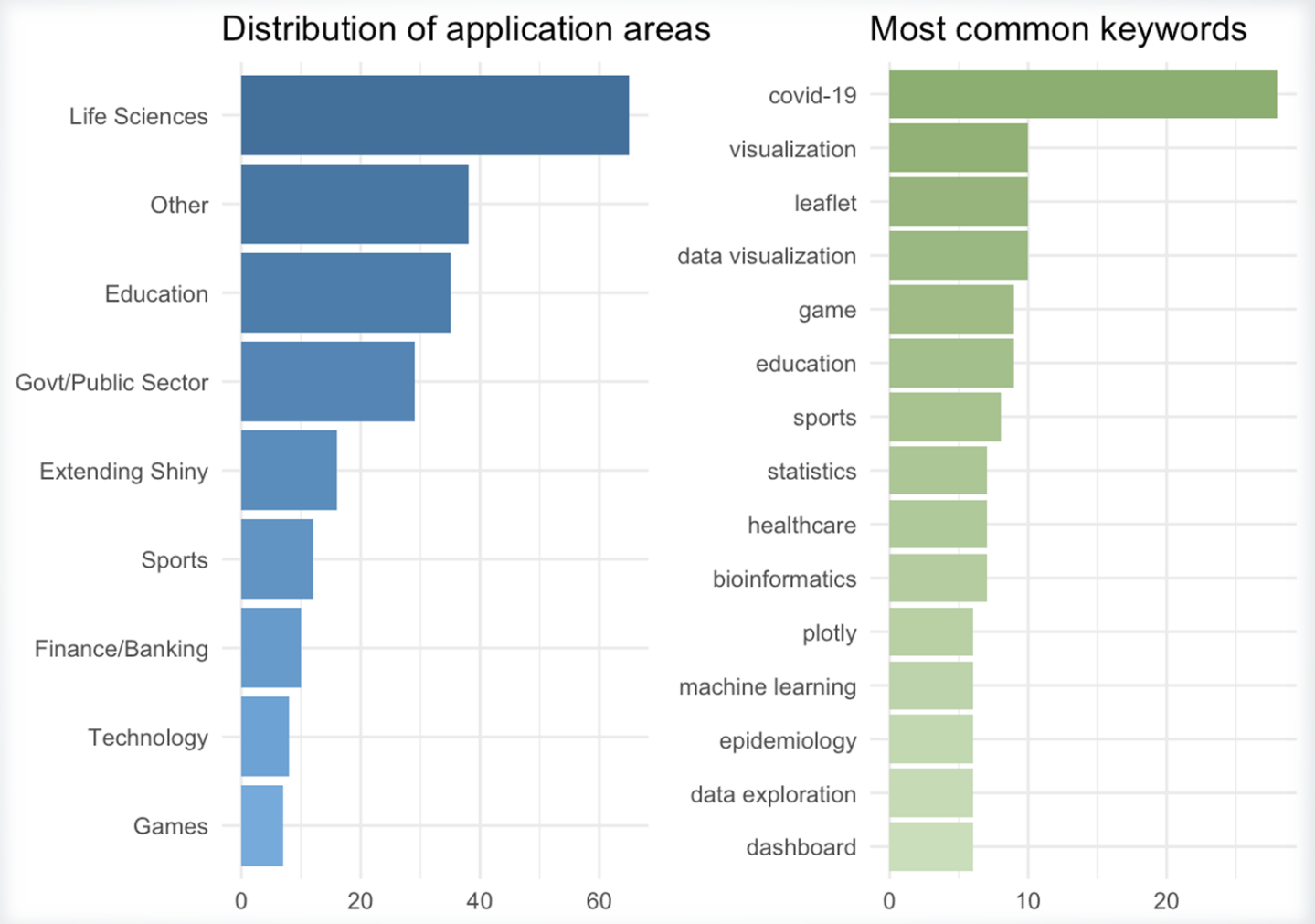


Shiny apps



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

Shiny apps


Distribution of application areas
Life Sciences
Other
Education
Govt/Public Sector
Extending Shiny
Sports
Finance/Banking
Technology
Games
0
20
40
60
Most common keywords
covid-19
visualization
leaflet
data visualization
game
education
sports
statistics
healthcare
bioinformatics
plotly
machine learning
epidemiology
data exploration
dashboard
0
10
20

Shiny examples

Global **data visualization**

Run **simulations** and download data

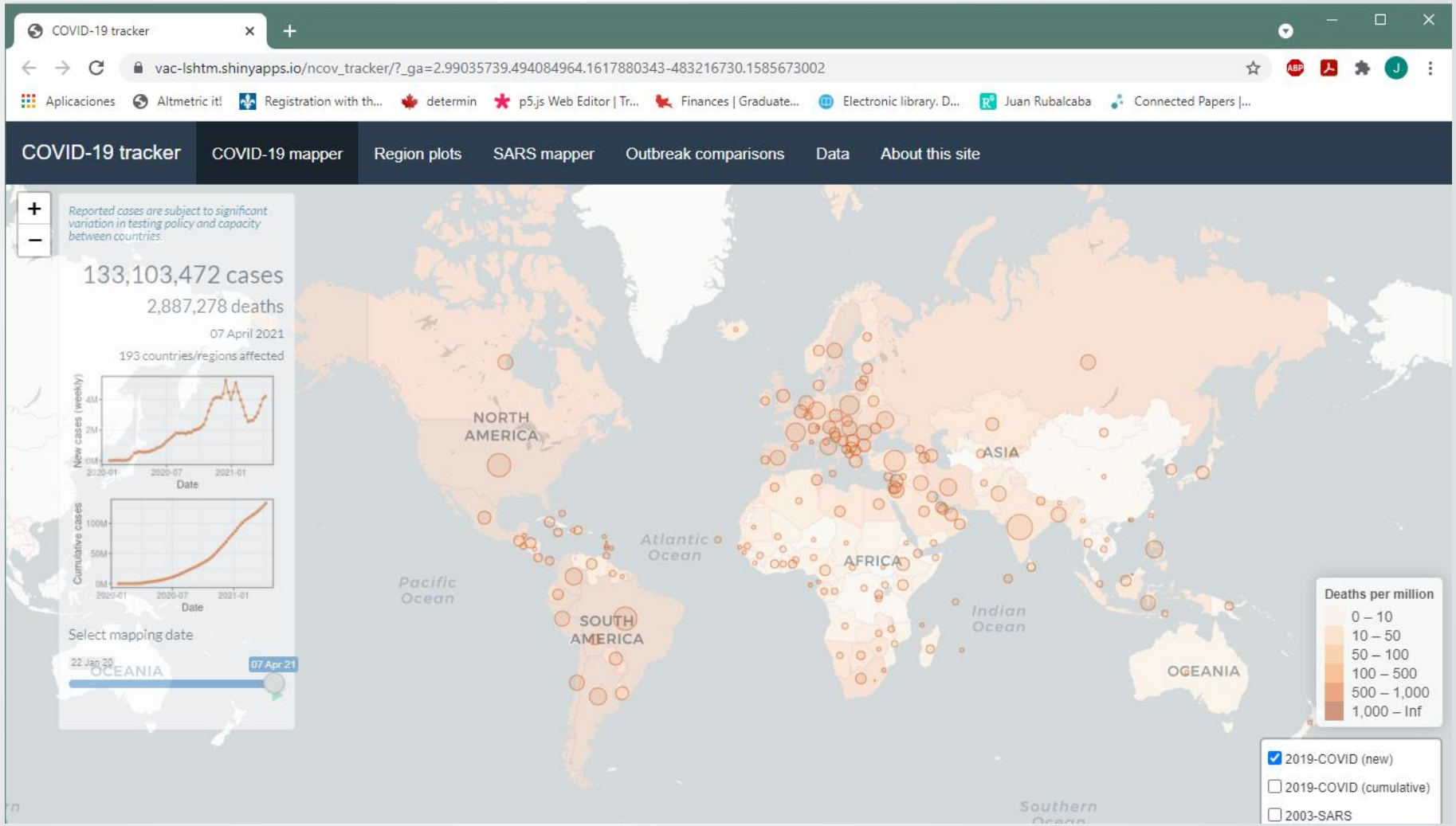
Data / code **transparency**

Understanding **models**

Educational apps

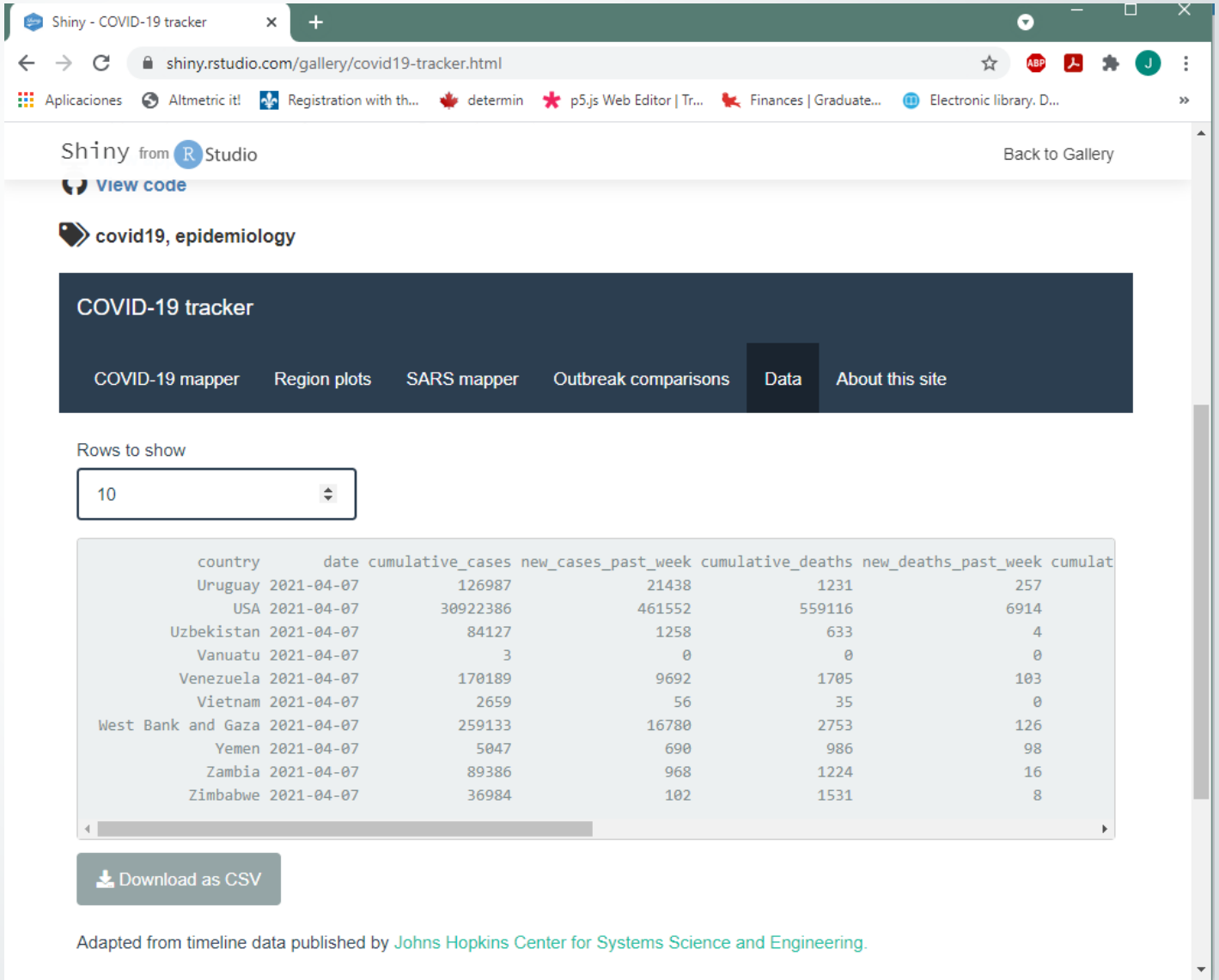
Data visualization – manage large datasets

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



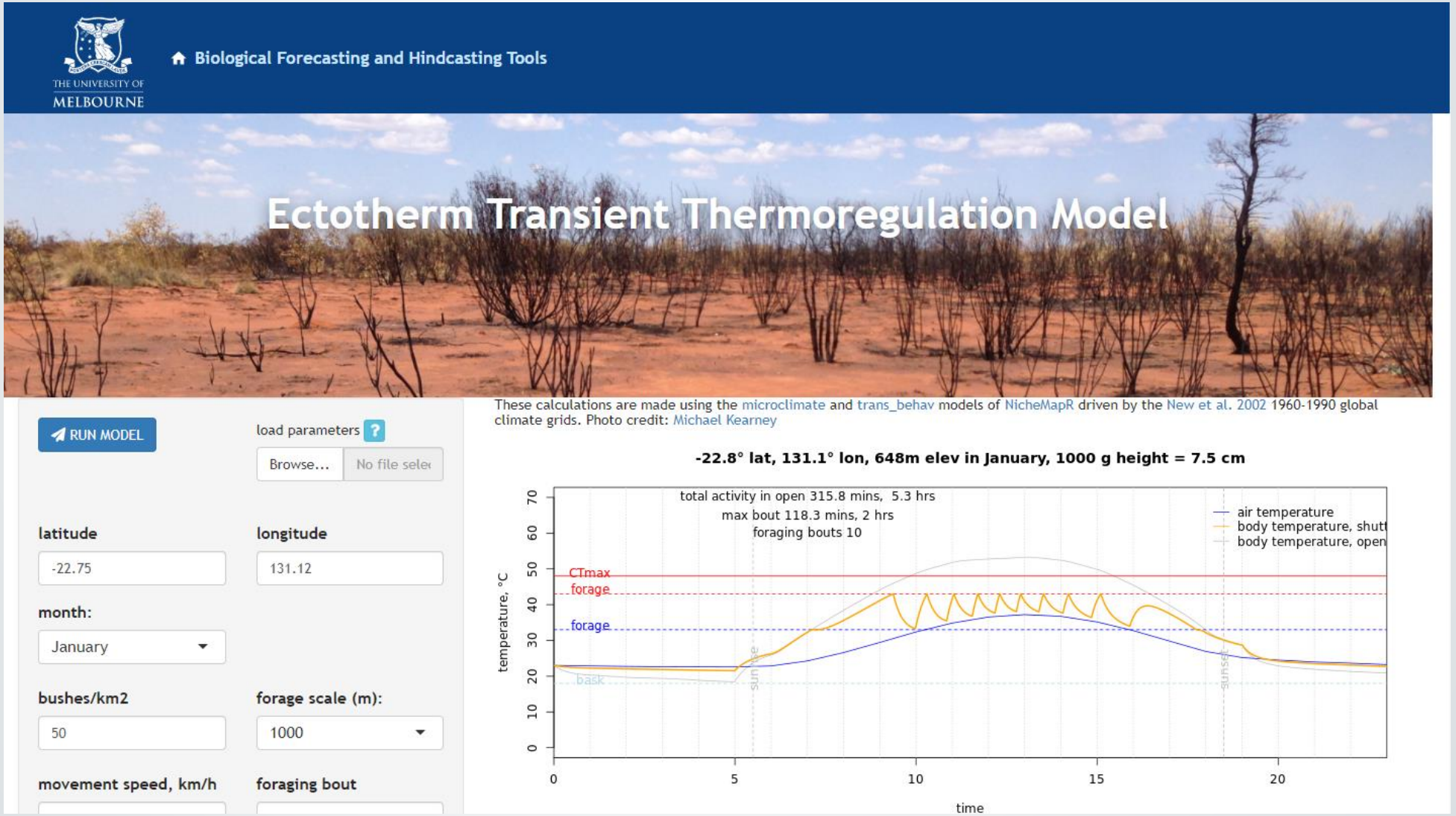
Data visualization – manage large datasets

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



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 | BRITISH ECOLOGICAL SOCIETY

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

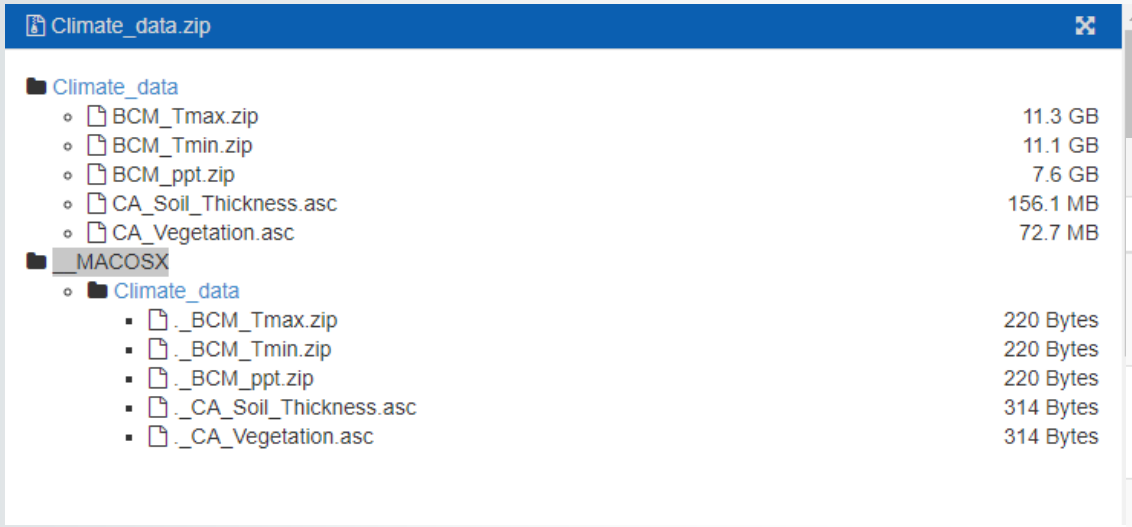
First published: 02 October 2020 | https://doi.org/10.1111/2041-210X.13501

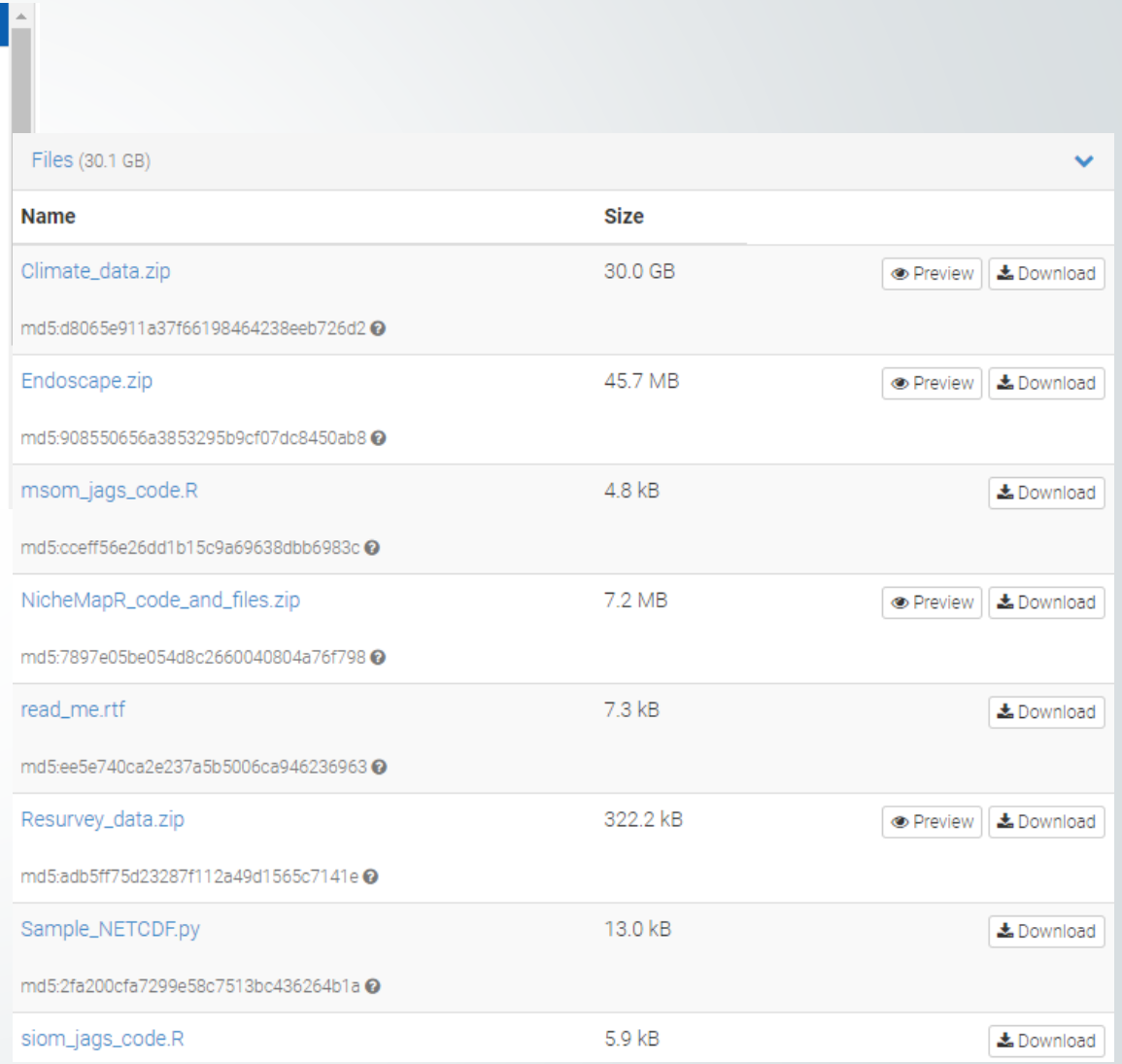
ranacapa: An R package and Shiny web app to explore environmental DNA data with exploratory statistics and interactive visualizations

Gaurav S. Kandlikar, Conceptualization, Data Curation, Methodology, Project Administration, Software, Validation,

Data / Code transparency

Data / Code transparency



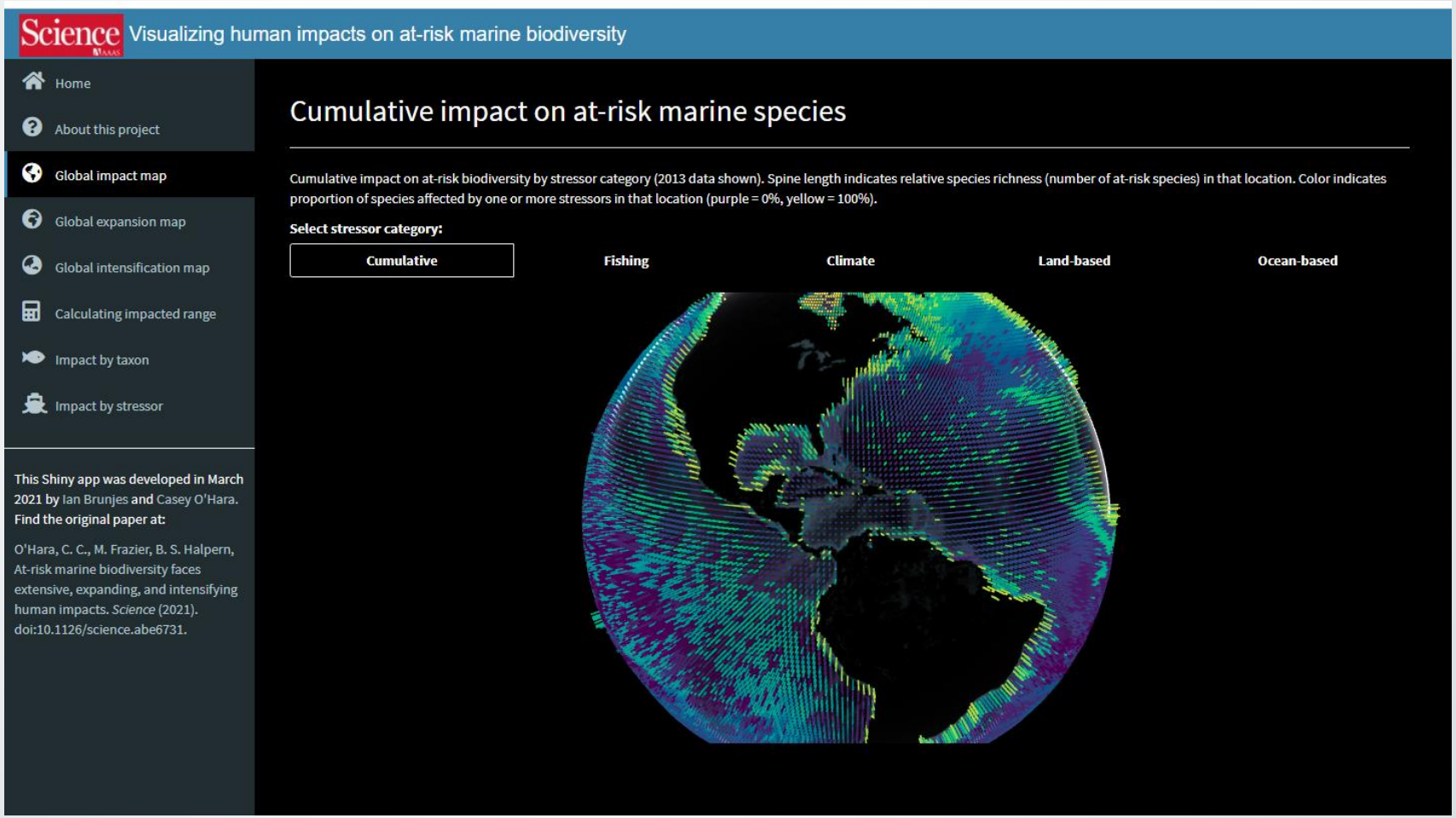


Riddell et al. (2021, *Science*)

Data / Code transparency

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

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



Educational apps

SIR (susceptible-infected-recovered) model

https://github.com/ekstroem/Shiny-Vaccine

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://mrtnj.shinyapps.io/mutation/

Educational apps

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

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

Phylogenetic comparative methods

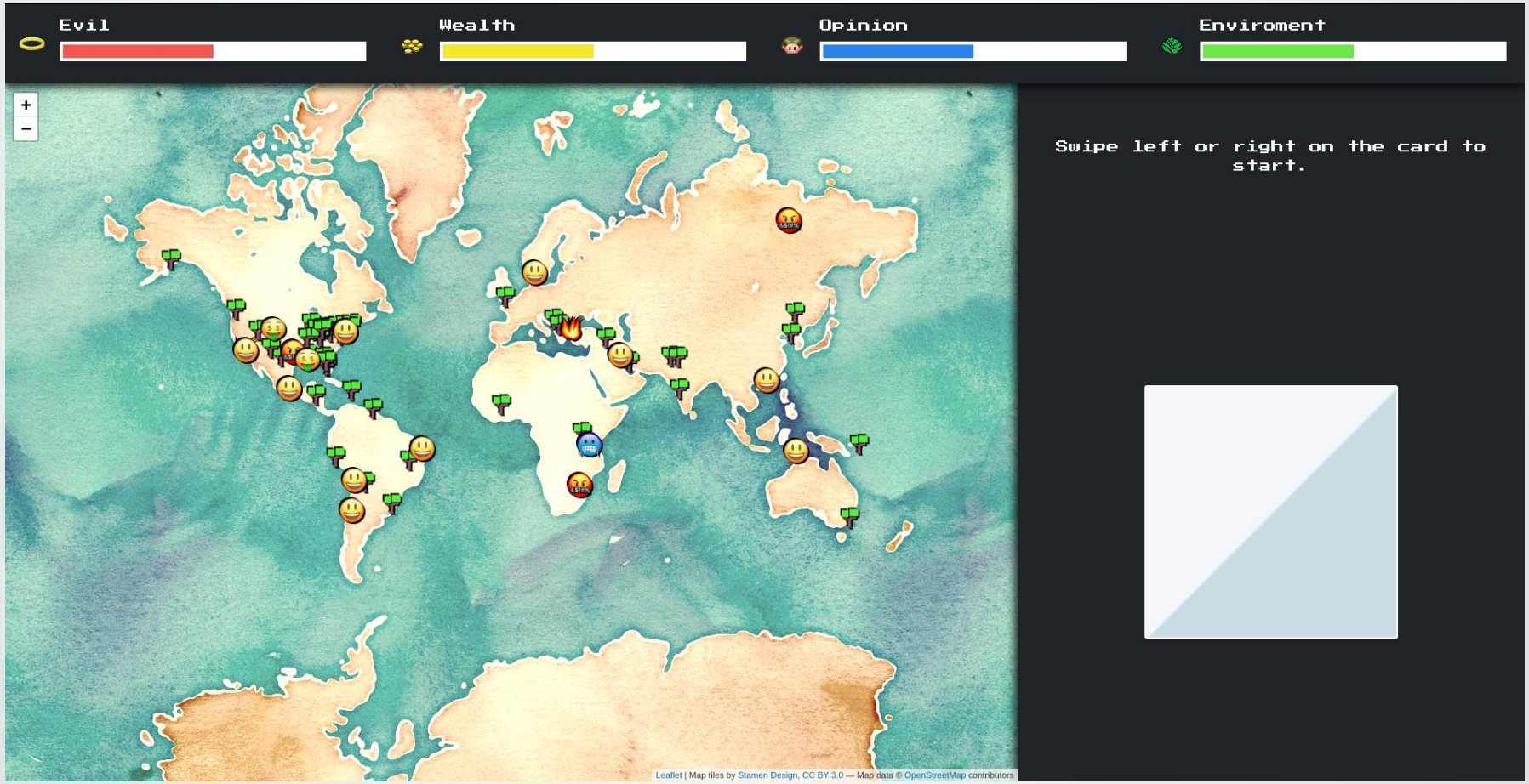
https://mossmatters.shinyapps.io/Phylo_Comp_Methods_Shiny/

Teaching **statistics**

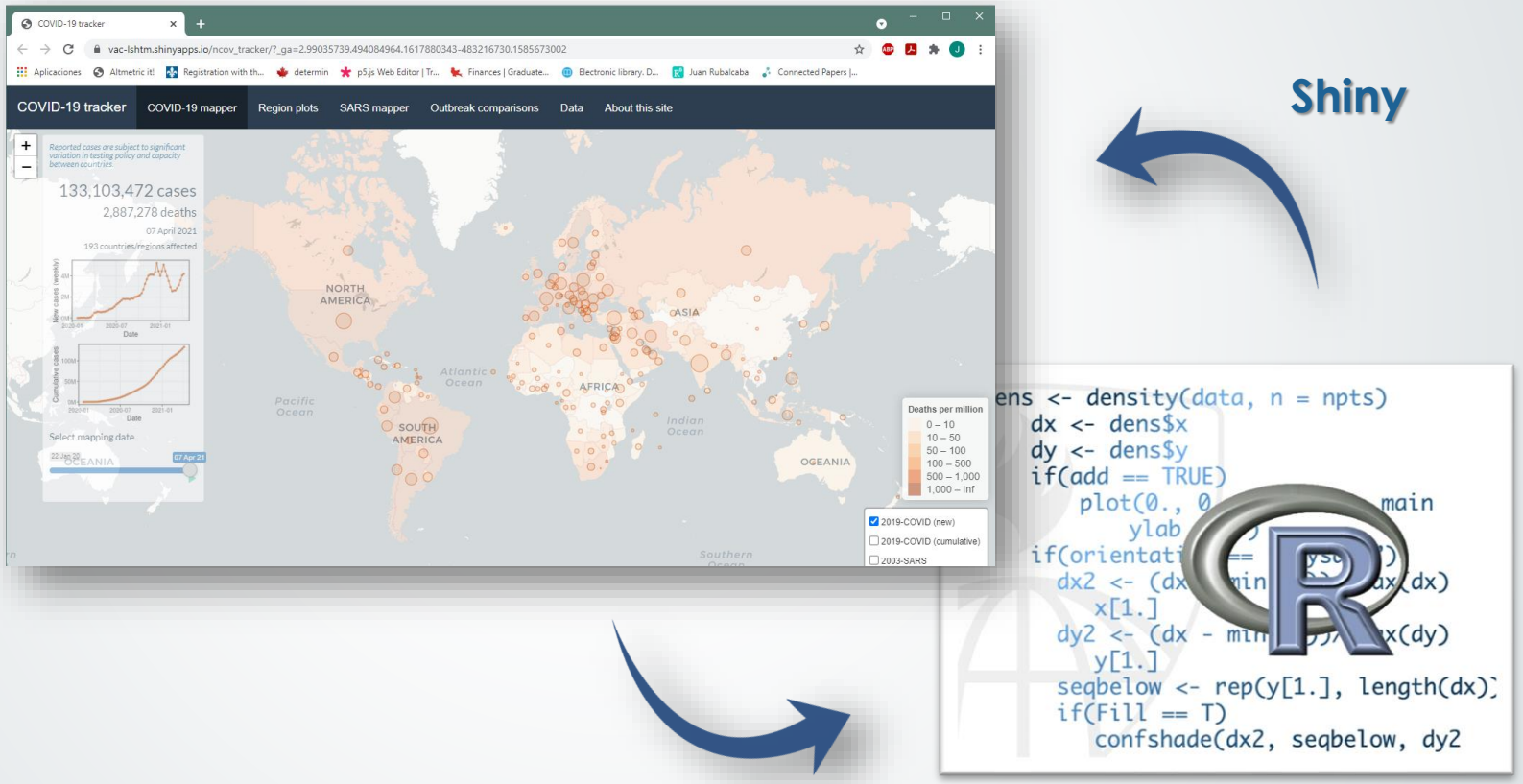
https://saskiaotto.de/shiny/

Video games (??)

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



Structure of a shiny app



Structure of a shiny app

User interface

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

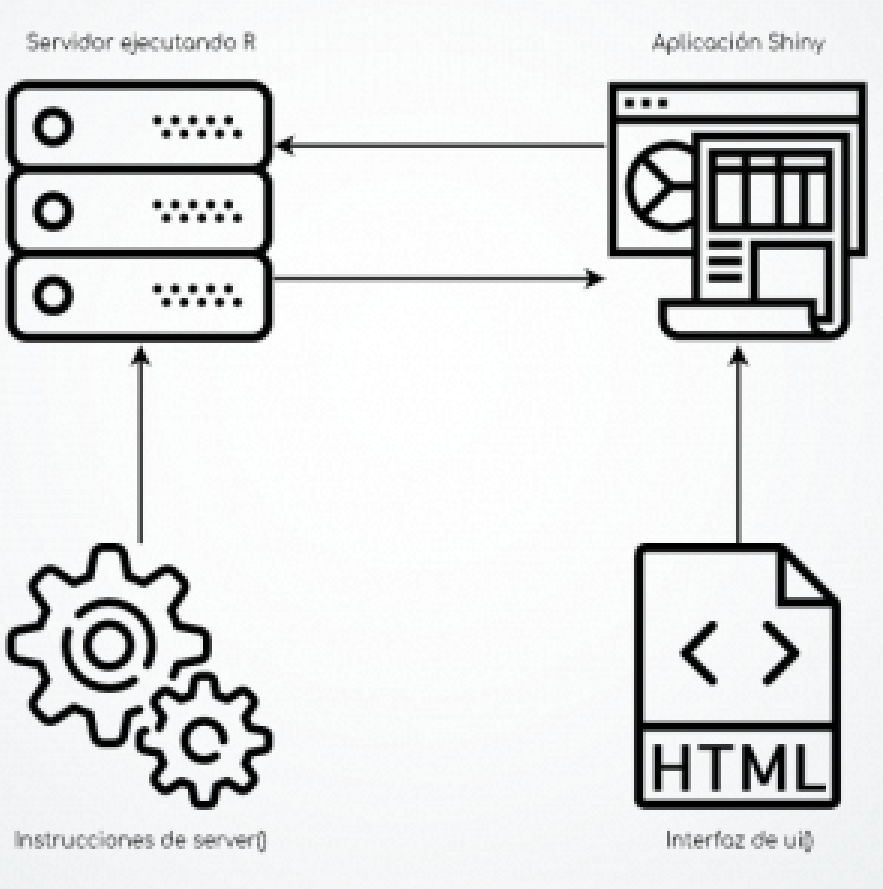
Server

```
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

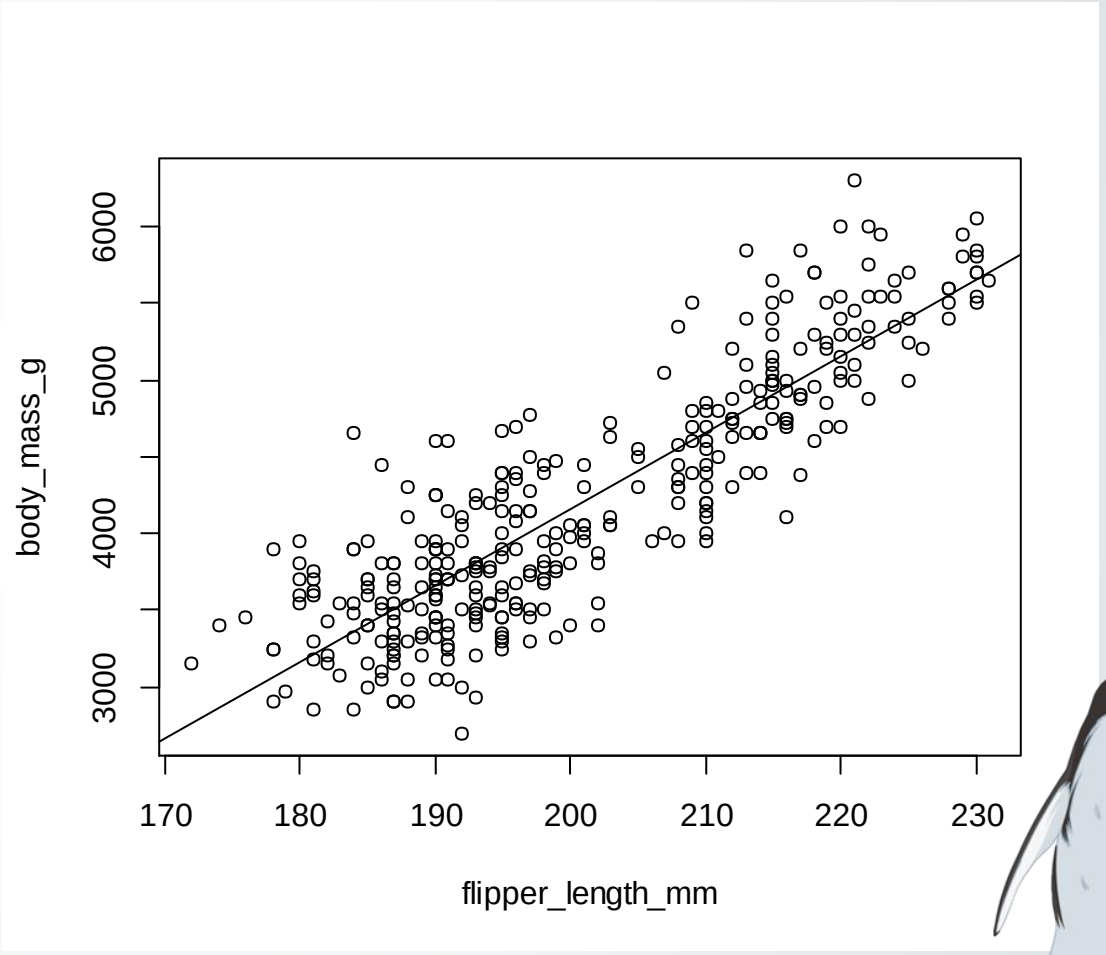
```
ui <- fluidPage(
    sidebarLayout(
        sidebarPanel(
        ),
        mainPanel(
        )
    )
)
```

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)
```

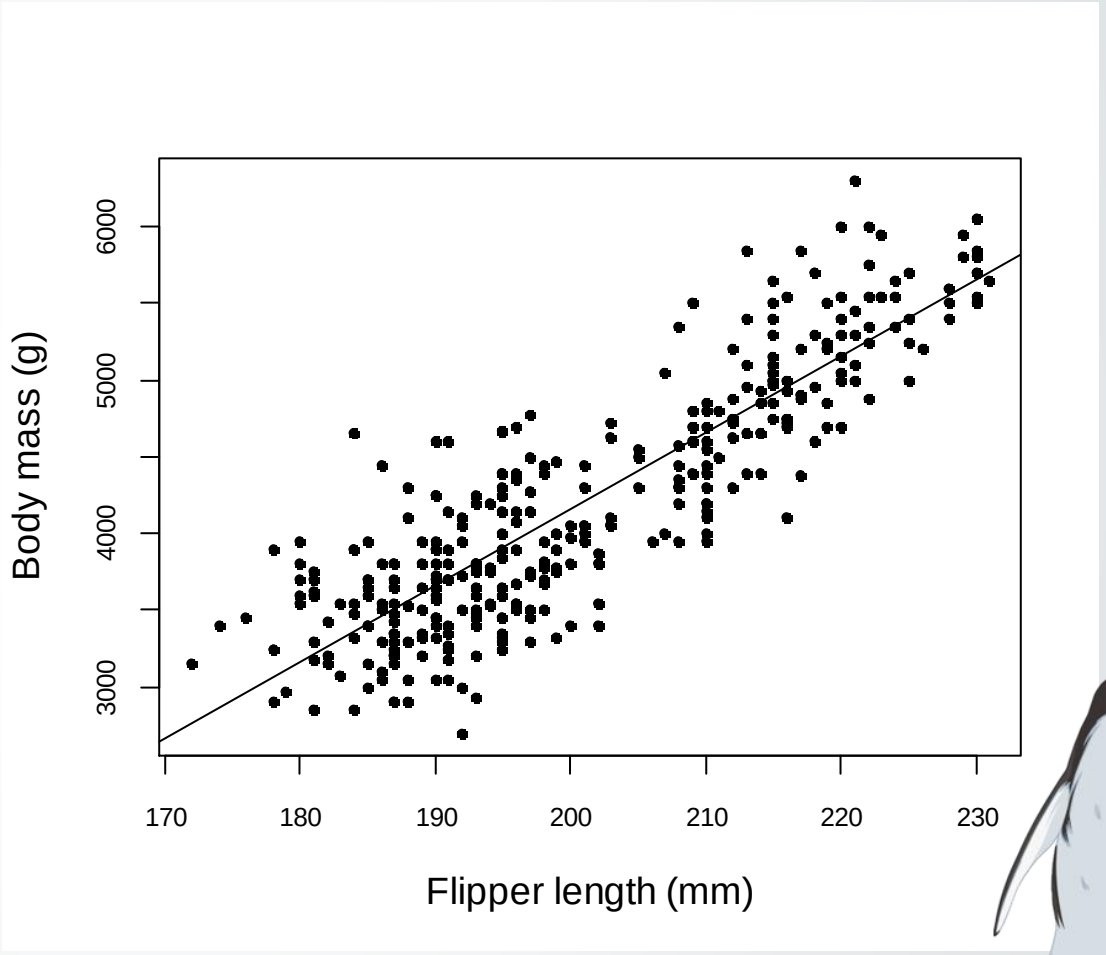


A quick example

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)
```

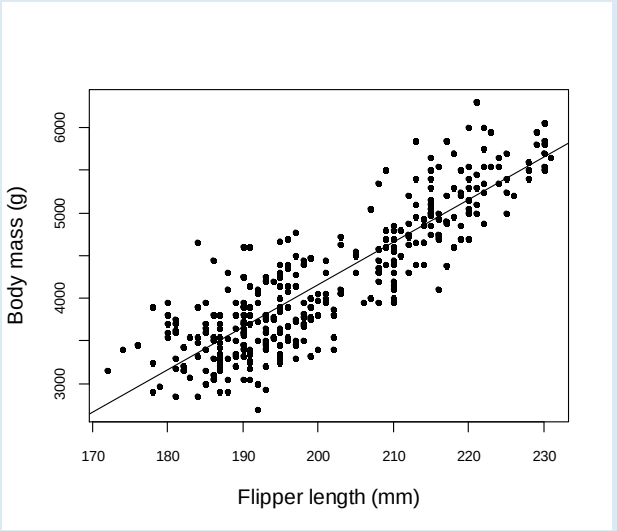


A quick example

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 scatteplot

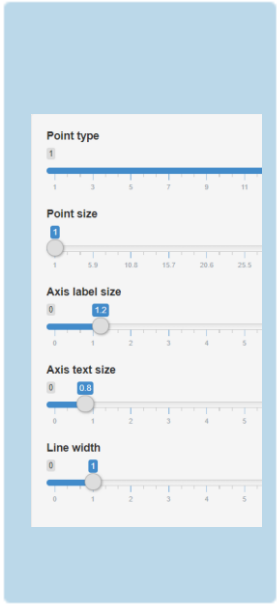


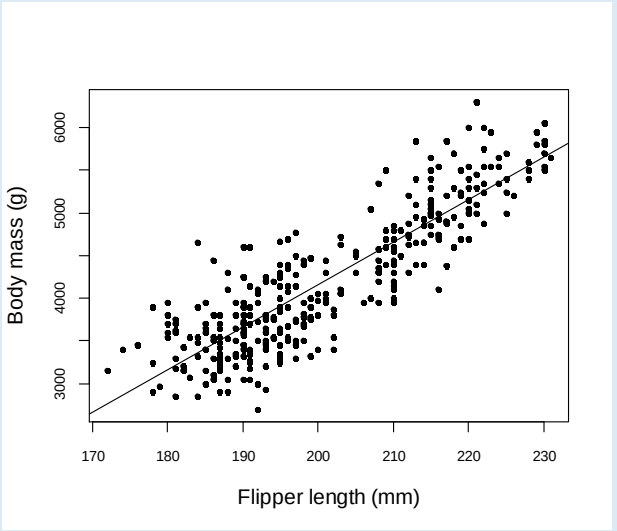
Example: 1_scatterplot.R

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)
                })
}
```

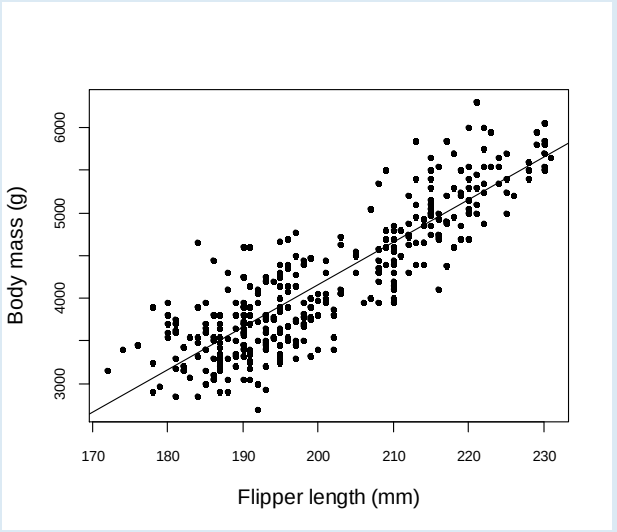




A quick example

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

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)
```

```
input <- list(
        “input_a”,
        “input_b”,
        “input_c”,
        …
)
```

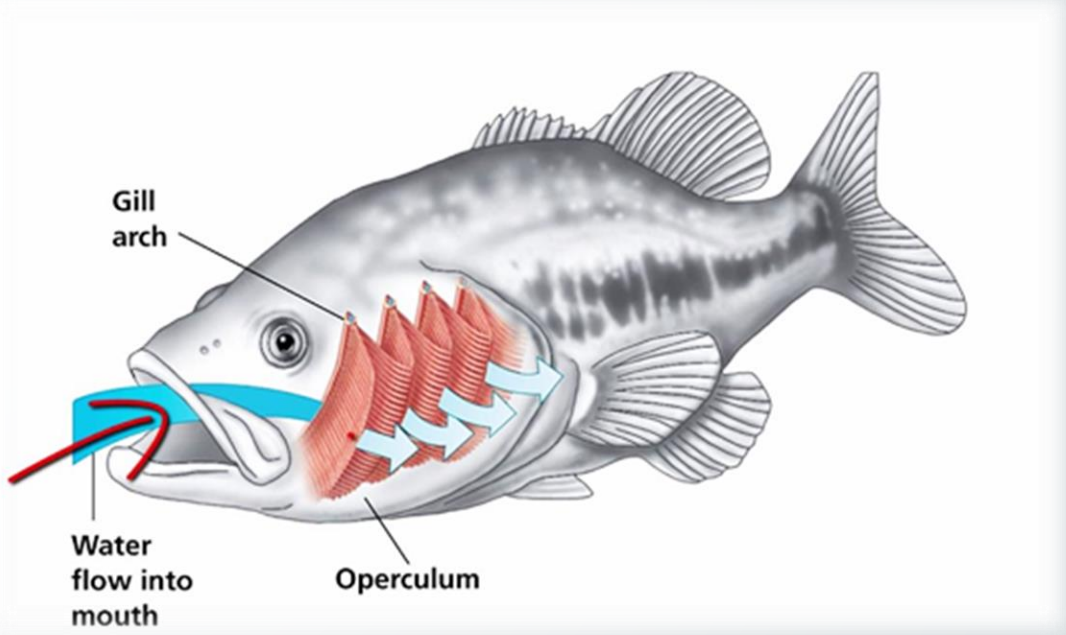
input$input_a

Example: 3_scatterplot_widgets.R

A more useful example

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

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



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 ($m_{02,\ demand}$) depends on…

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

A more useful example

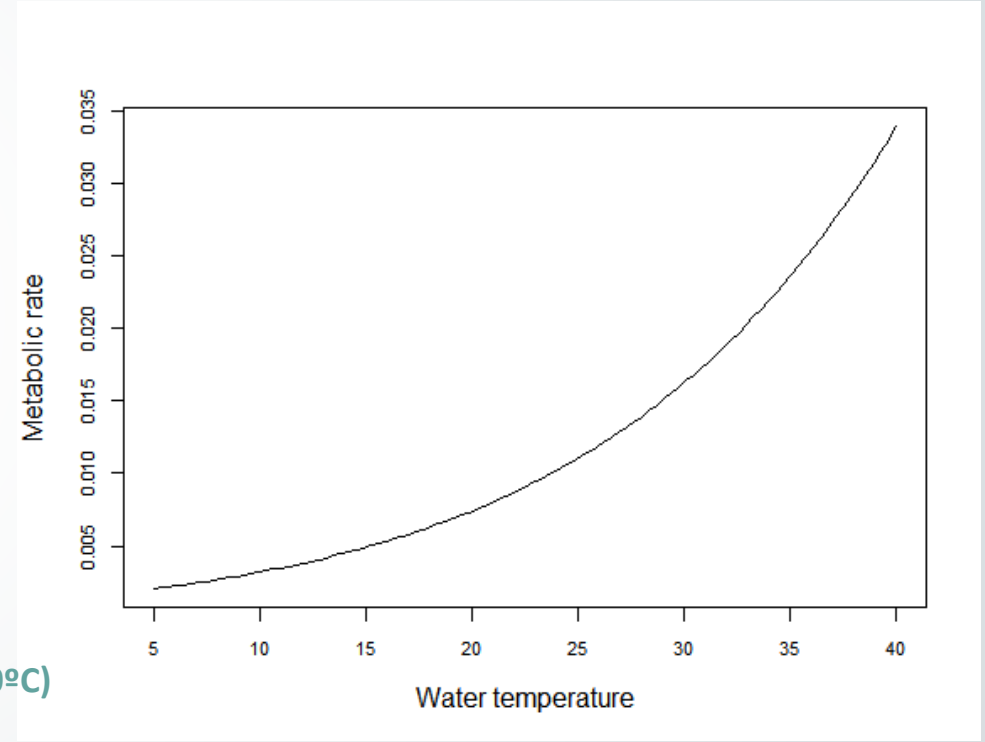
A model to derive **fish metabolic rate** as a function of **water temperature**

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

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 5ºC to 40ºC)

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**

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

Metabolic rate ($m_{02,\ demand}$) depends on…

delta,	Activity level
M,	Body mass
A,	Gill surface area
h_c,	Oxygen transfer coefficient
$O_{2,water}$,	Oxygen concentration
Km,	Michaelis constant
E,	Activation energy
b0,	Normalization constant

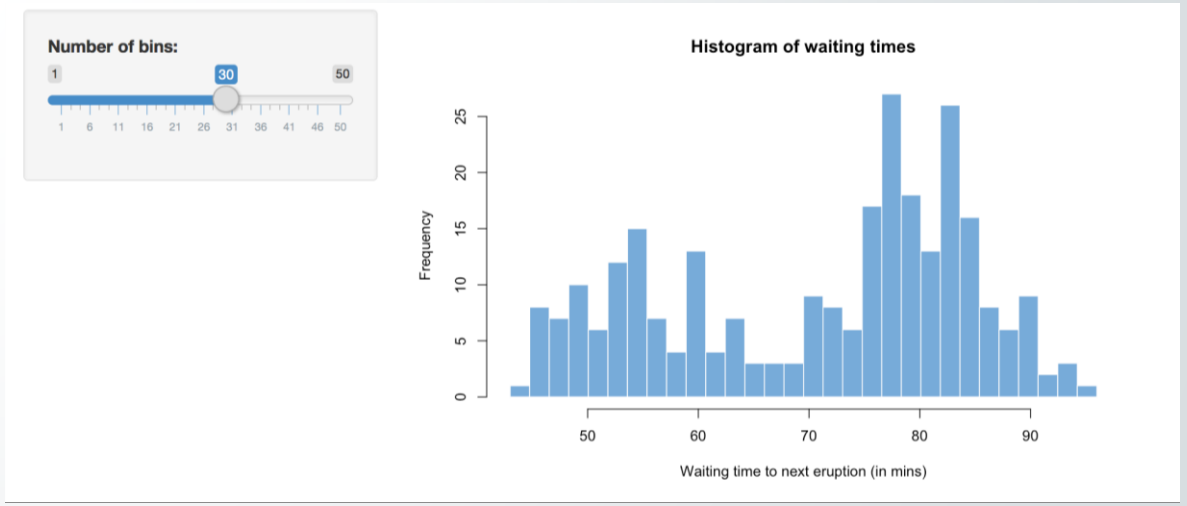
Example: 4_nonlinear_model.R

https://jrubalcaba.shinyapps.io/app_oxlim

UI Layouts

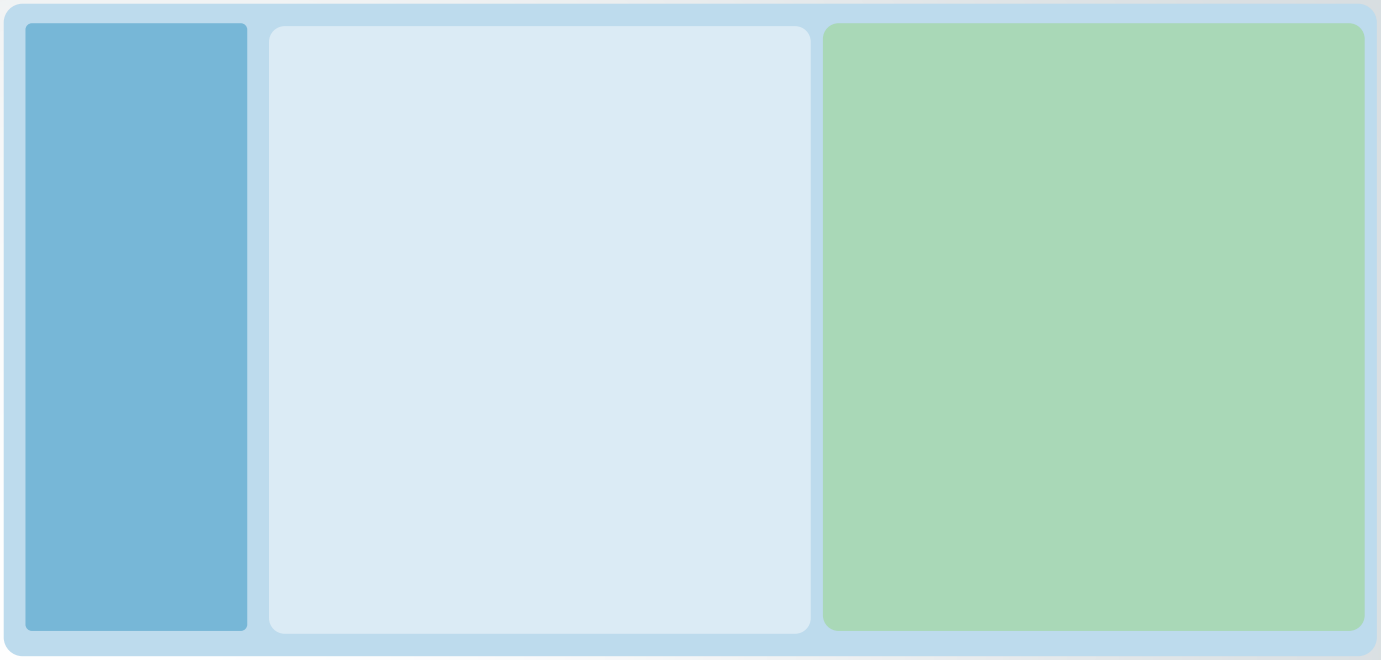
UI Layouts

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

UI Layouts

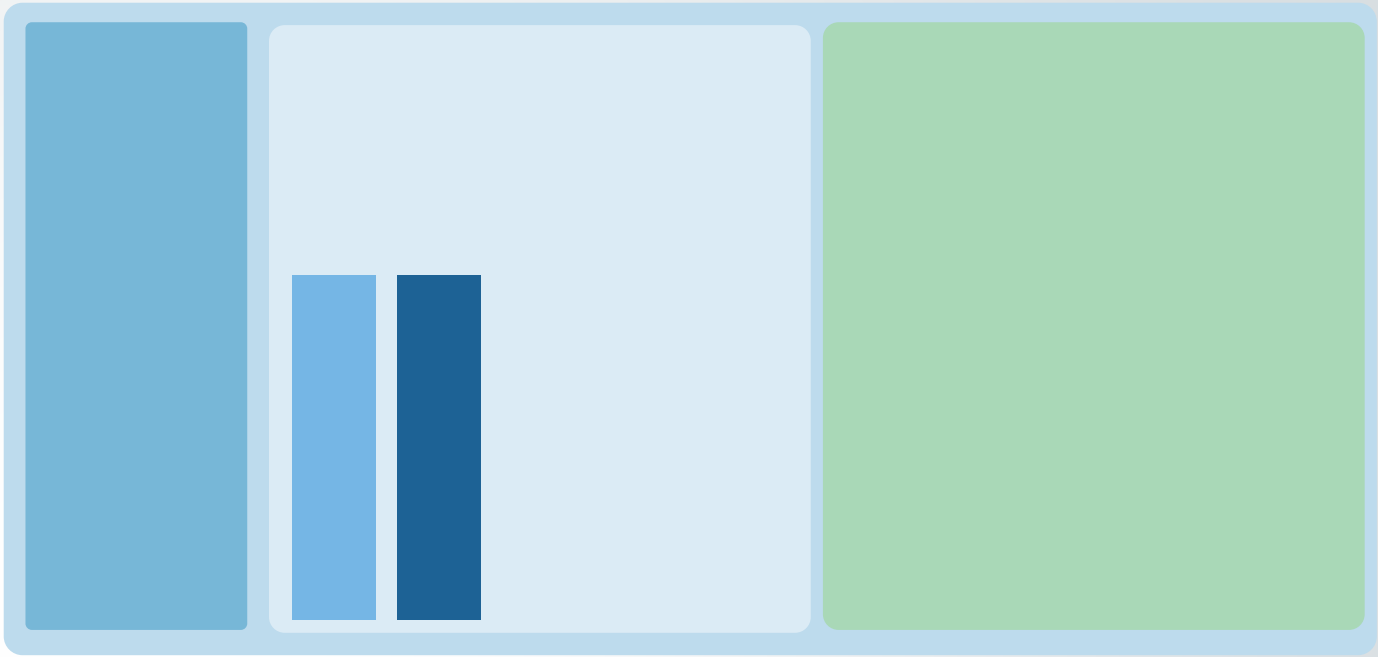
```
ui <- fluidPage(

    fluidRow(

        column(2,
        ),

        column(5,




        ),

        column(5,
        )
    )
)
```



*12 columns

UI Layouts

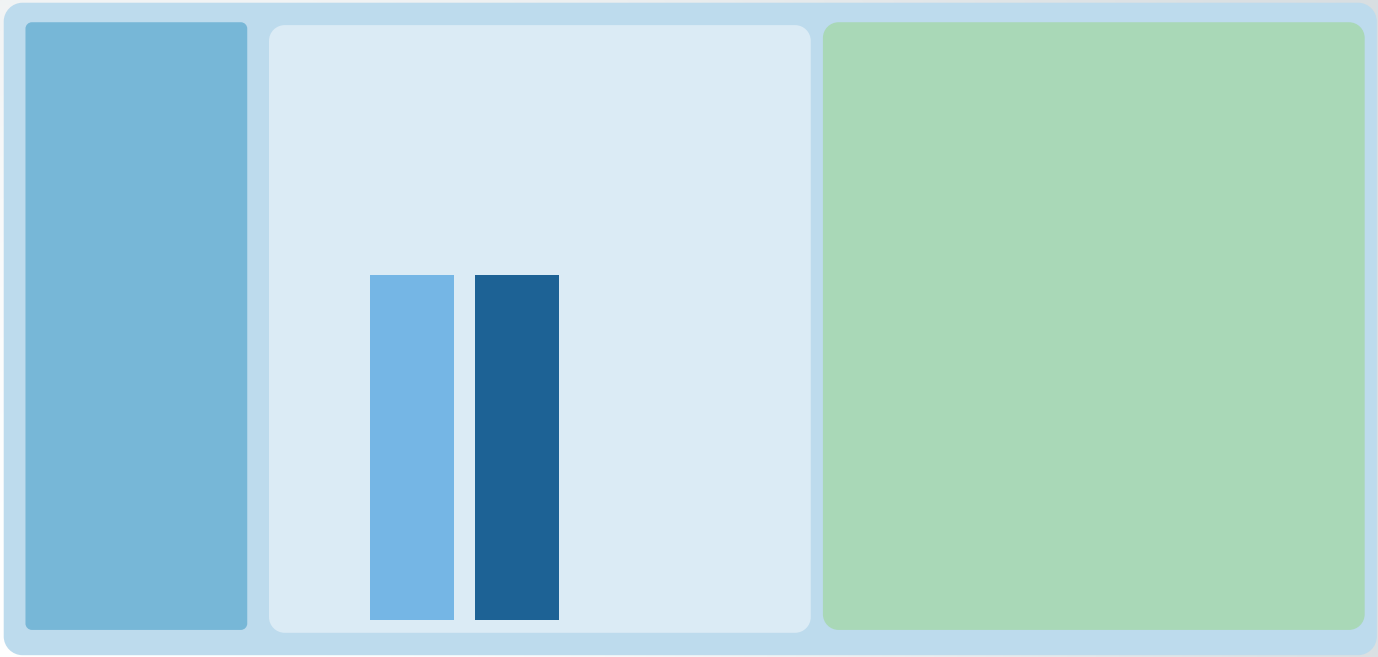
```
ui <- fluidPage(

    fluidRow(

        column(2,
        ),

        column(5,
            fluidRow(
                column(1,
                ),
                column(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,
        )
    )
)
```



*12 columns

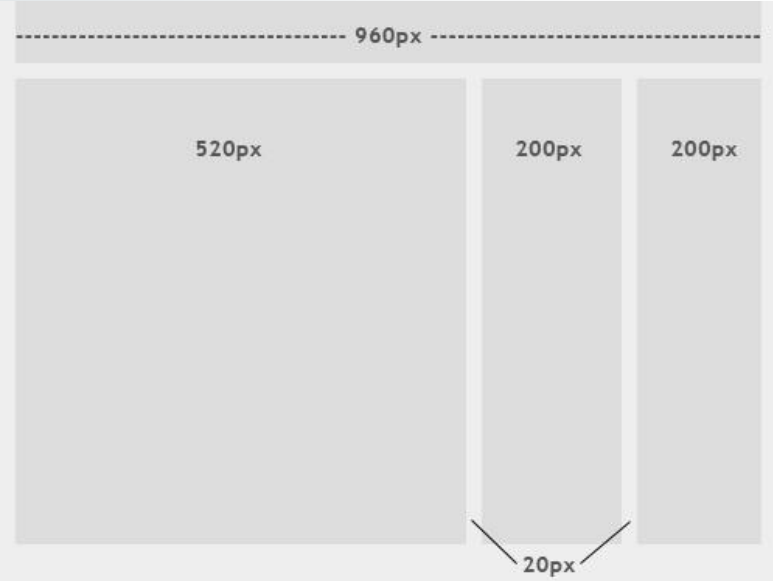
UI Layouts

```
ui <- fluidPage(

    fluidRow(

        column(2,
        ),

        column(5,
            fluidRow(
                column(1, offset = 1,
                ),
                column(1, offset = 1,
                ),
            )
        ),

        column(5,
        )
    )
)
```

UI Layouts

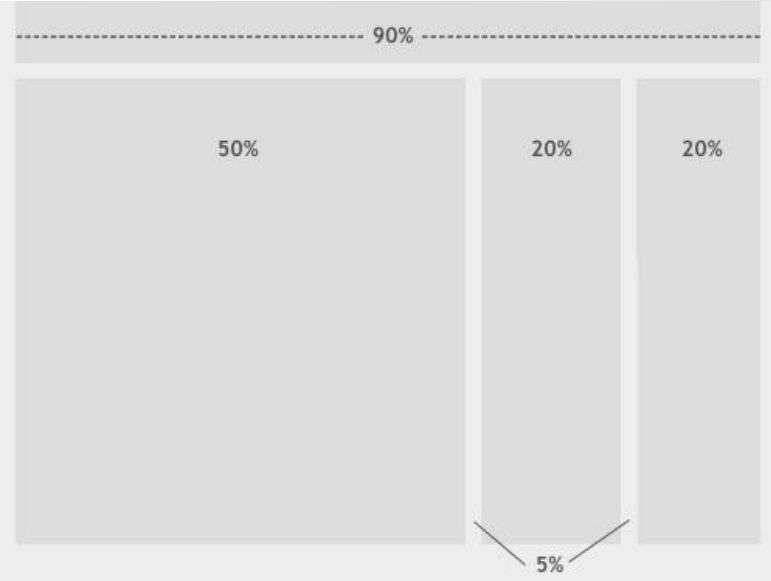
Fluid or fixed layouts?

Fixed



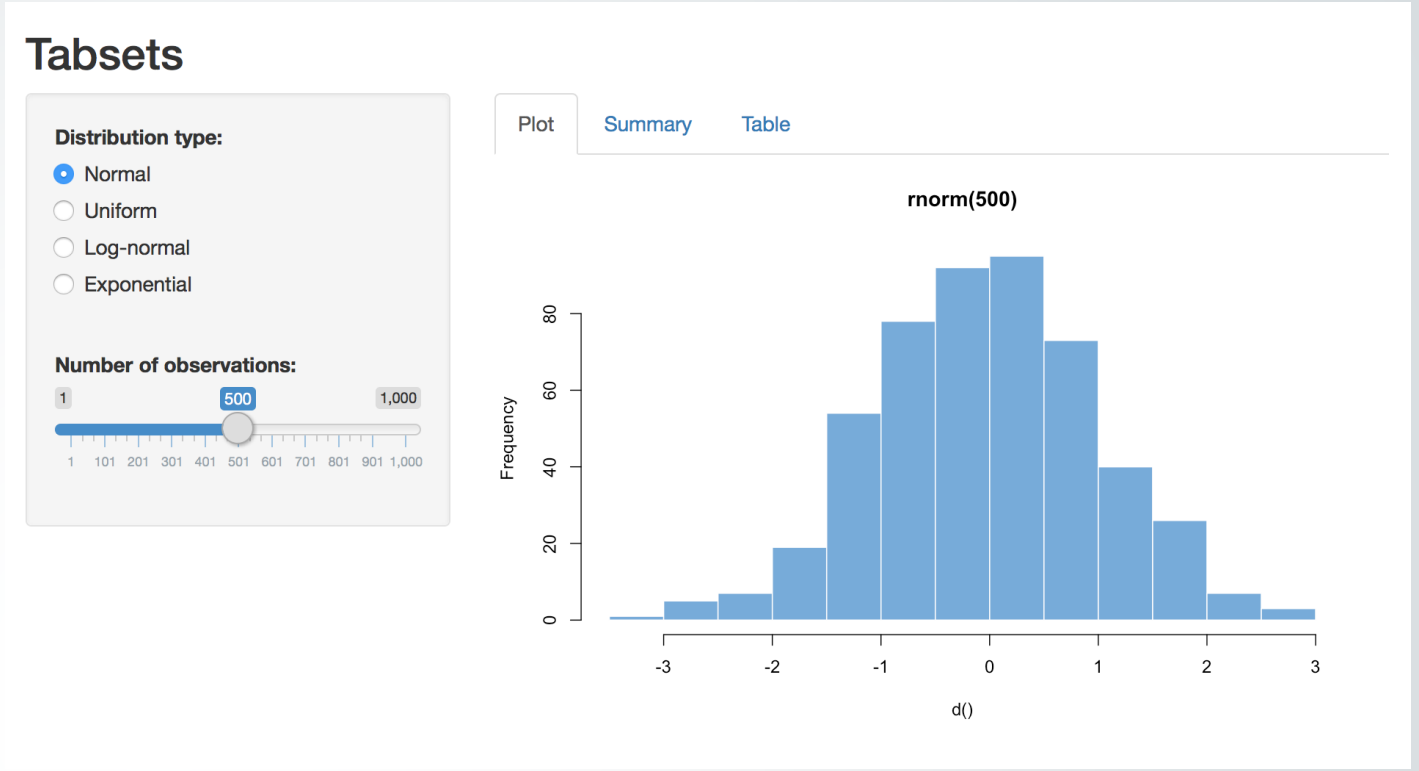
* 940px in Shiny

Fluid



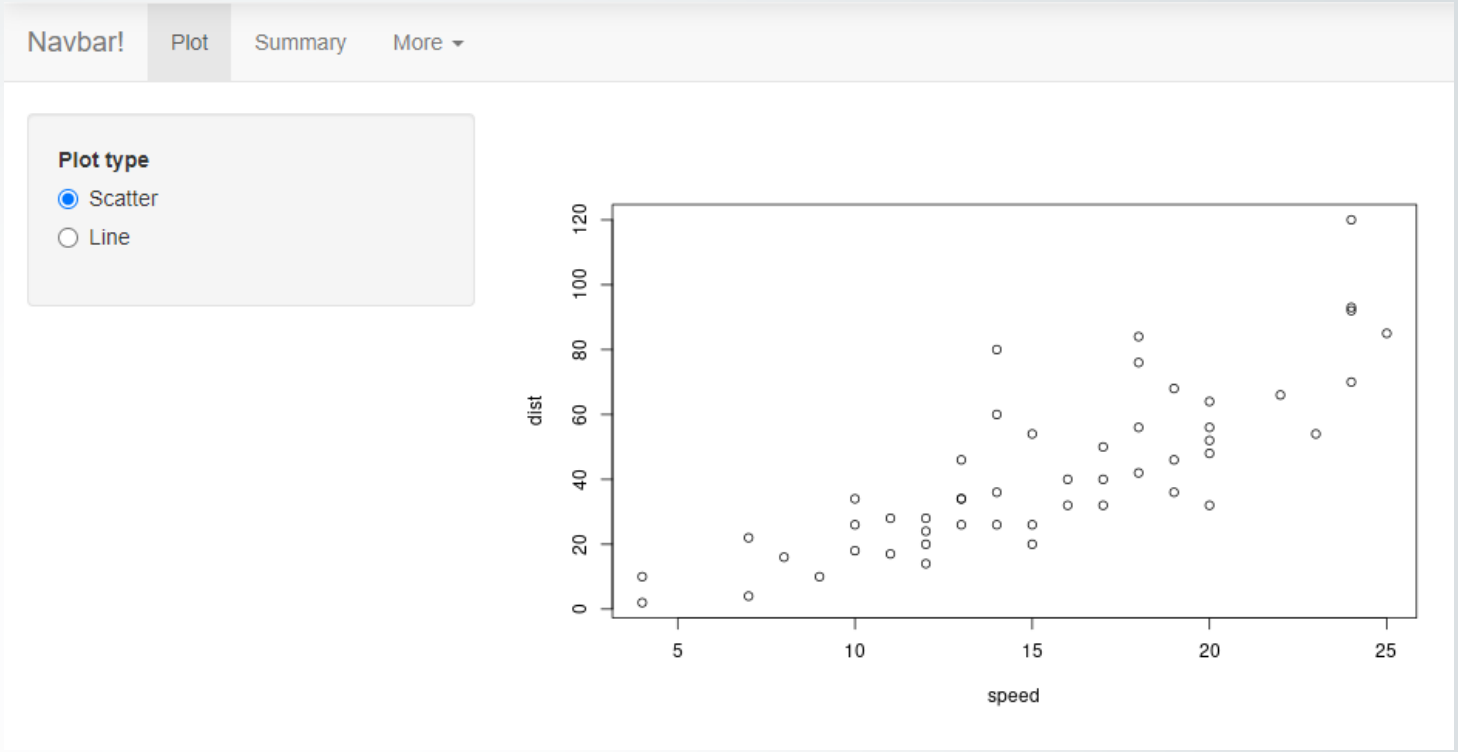
UI Layouts

```
ui <- fluidPage(

        sidebarLayout(

                sidebarPanel(
                ),

                mainPanel(
                        tabsetPanel(
                                tabPanel("Plot", plotOutput("plot")),
                                tabPanel("Summary", verbatimTextOutput("summary")),
                                tabPanel("Table", tableOutput("table"))
                        )
                )
        )
)
```



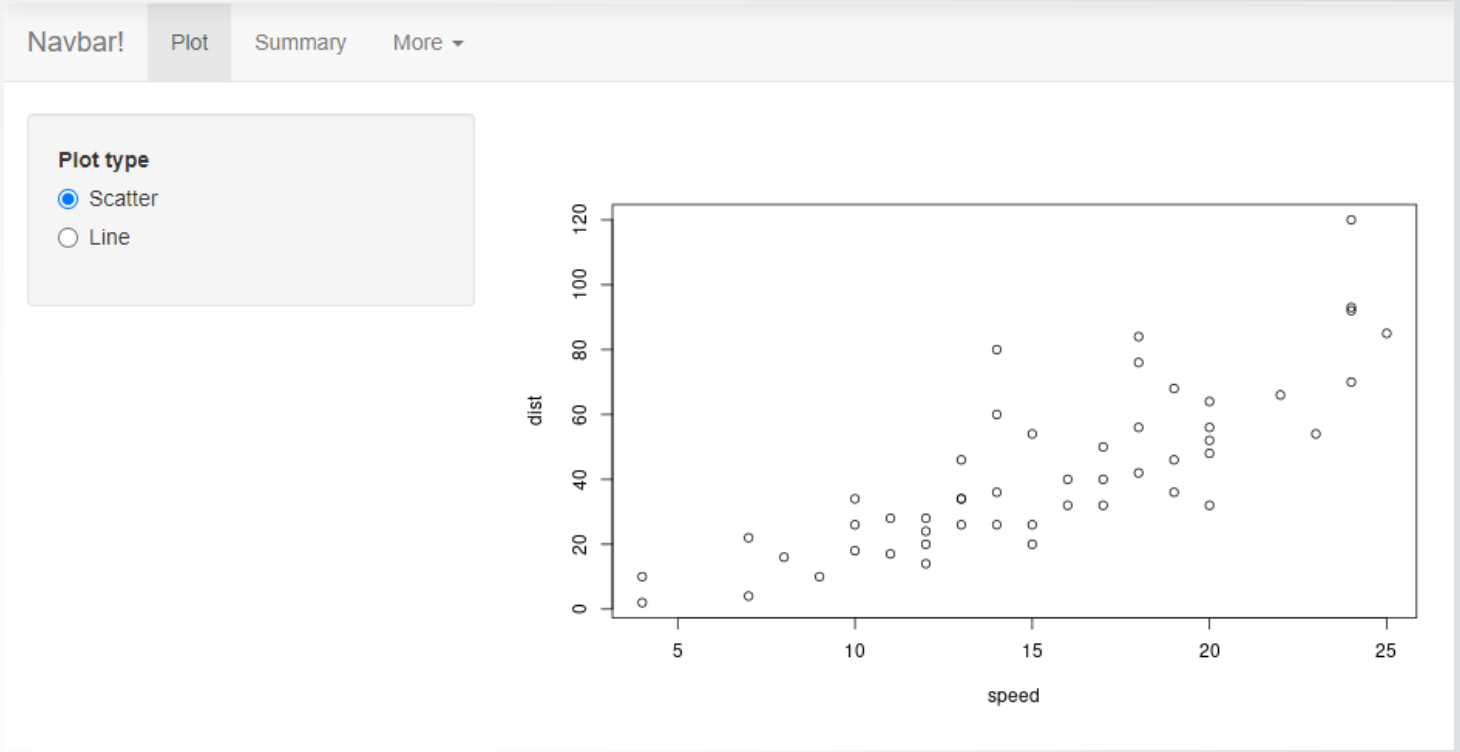
UI Layouts

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

        tabPanel("Plot",
        - Layout of section 1 -
    ),

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

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

)
```



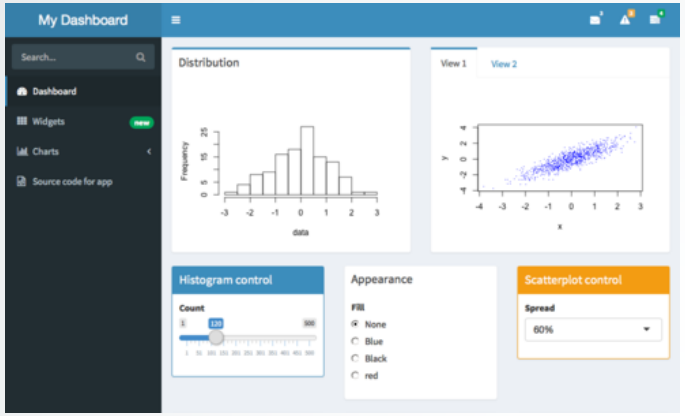
UI Layouts

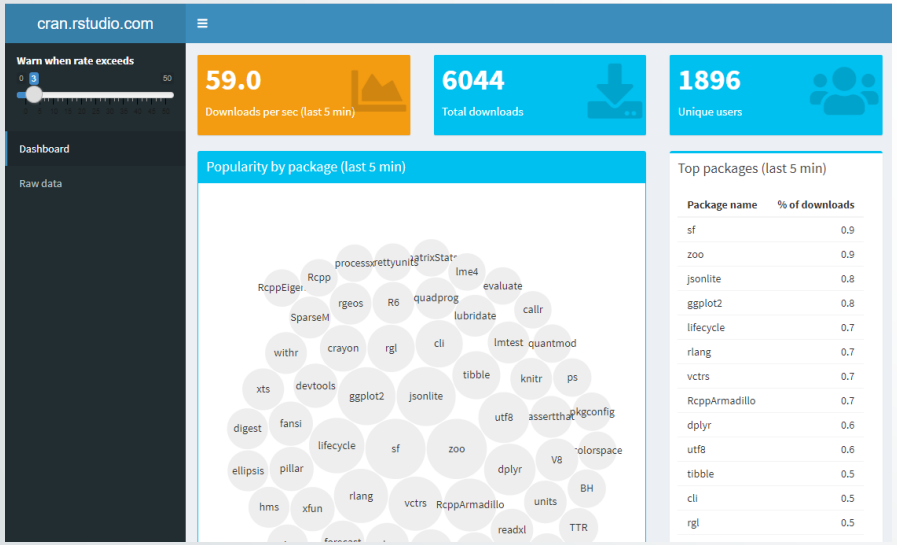
```
ui <- navbarPage("App name",

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

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

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

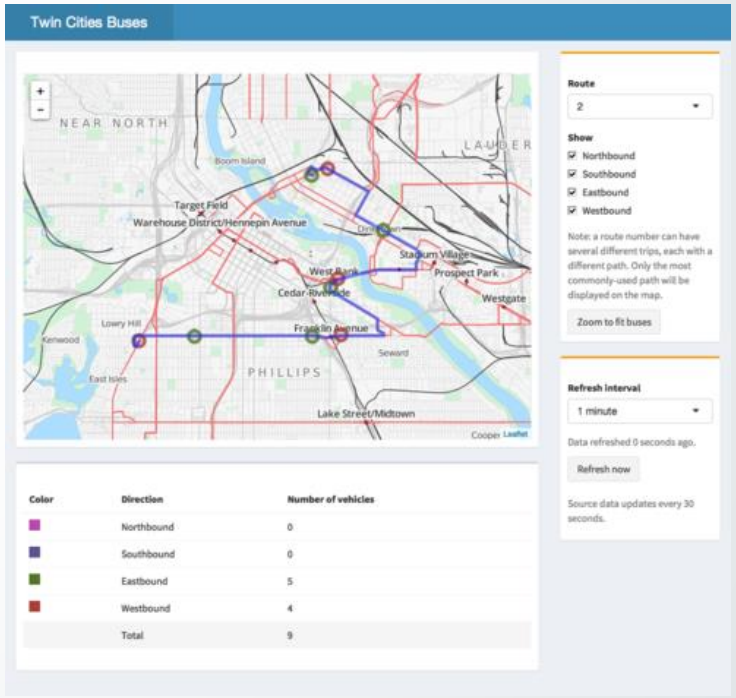


https://shiny.rstudio.com/articles/layout-guide.html

Making nicer UI with **shinydashboard**

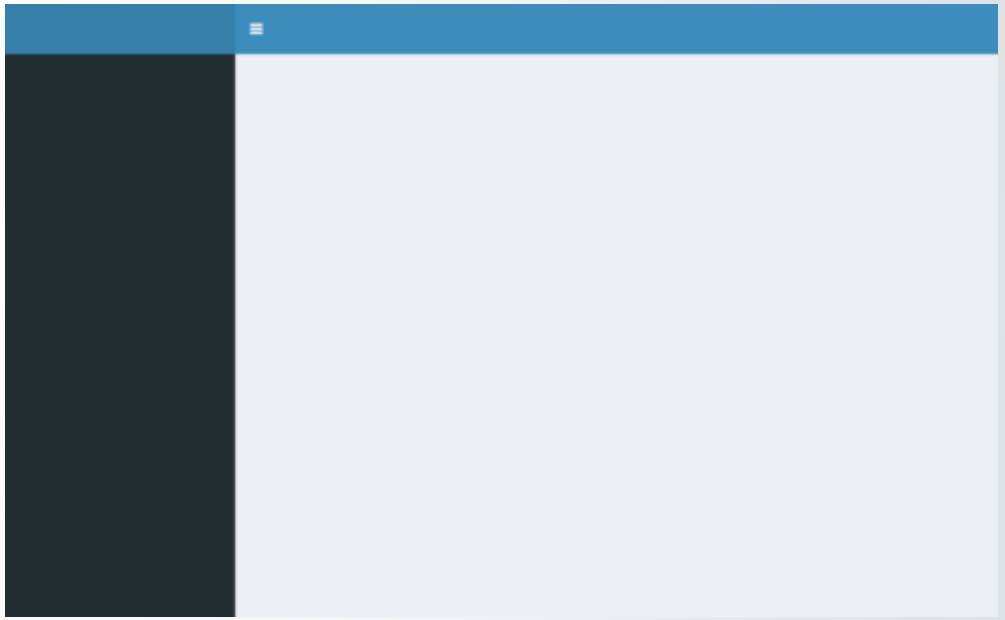




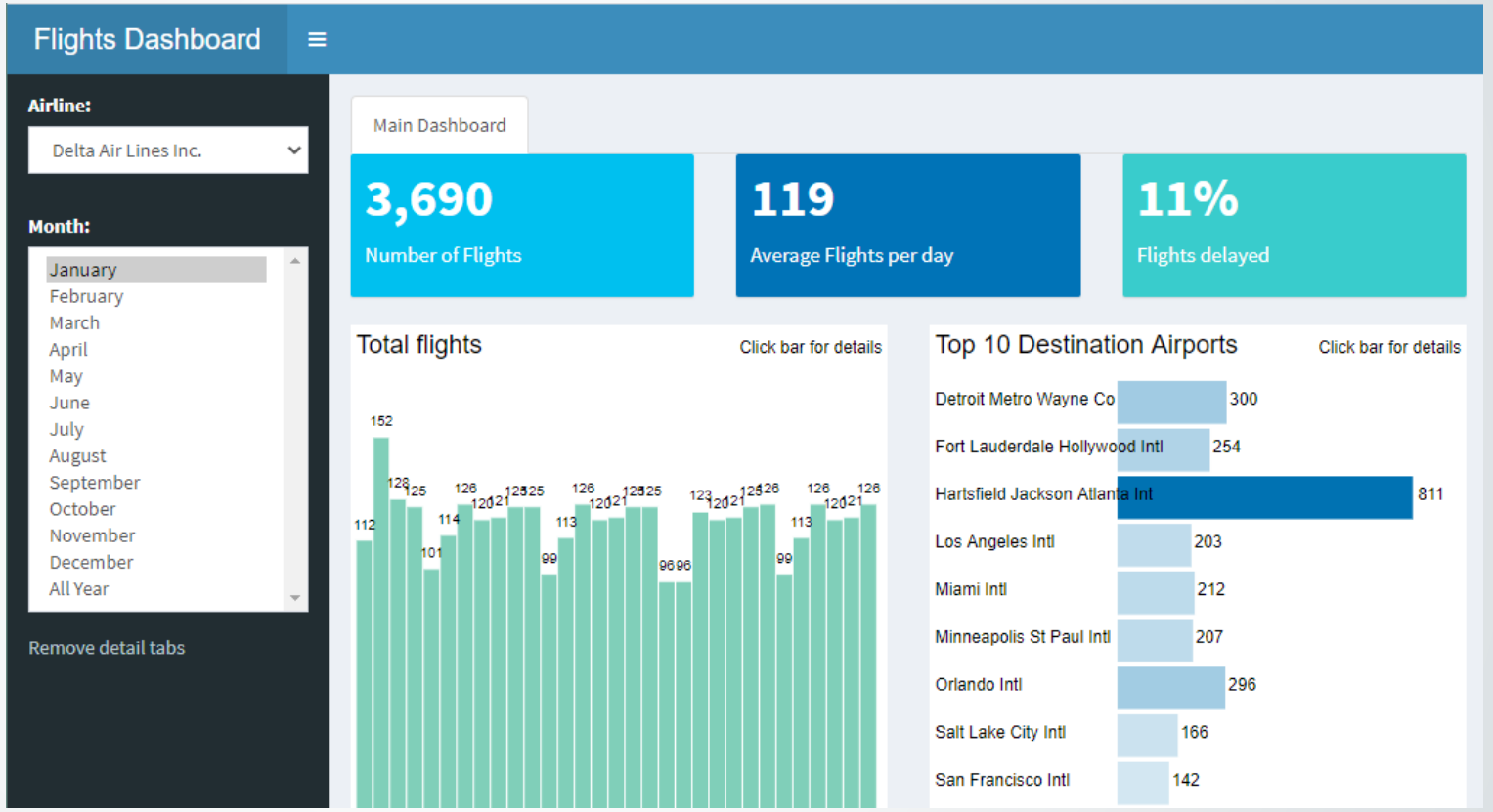


Making nicer UI with **shinydashboard**

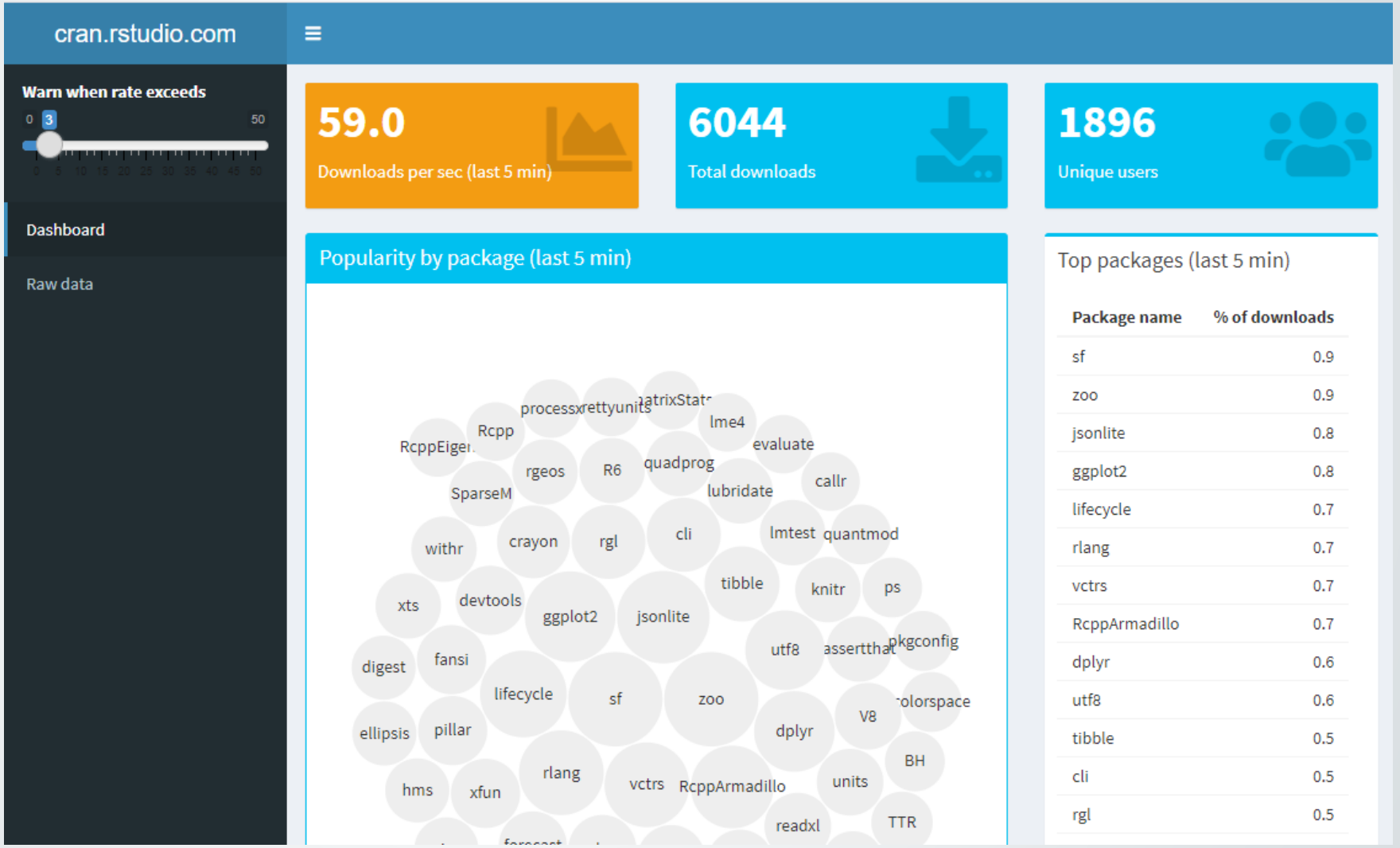
```
ui <- dashboardPage(
        dashboardHeader(),
        dashboardSidebar(),
        dashboardBody()
)
```



Making nicer UI with **shinydashboard**



Making nicer UI with **shinydashboard**



Making nicer UI with **shinydashboard**

```
ui <- fluidPage(

        sidebarLayout(
                sidebarPanel(

                ),
                mainPanel(

                )
        )
)

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

        })
}

shinyApp(ui, server)
```

Input parameters, sliders, action buttons….

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

R code to create our plots

Making nicer UI with **shinydashboard**

```
ui <- fluidPage(

    sidebarLayout(
        sidebarPanel(


        ),
        mainPanel(


        )
    )
)
```

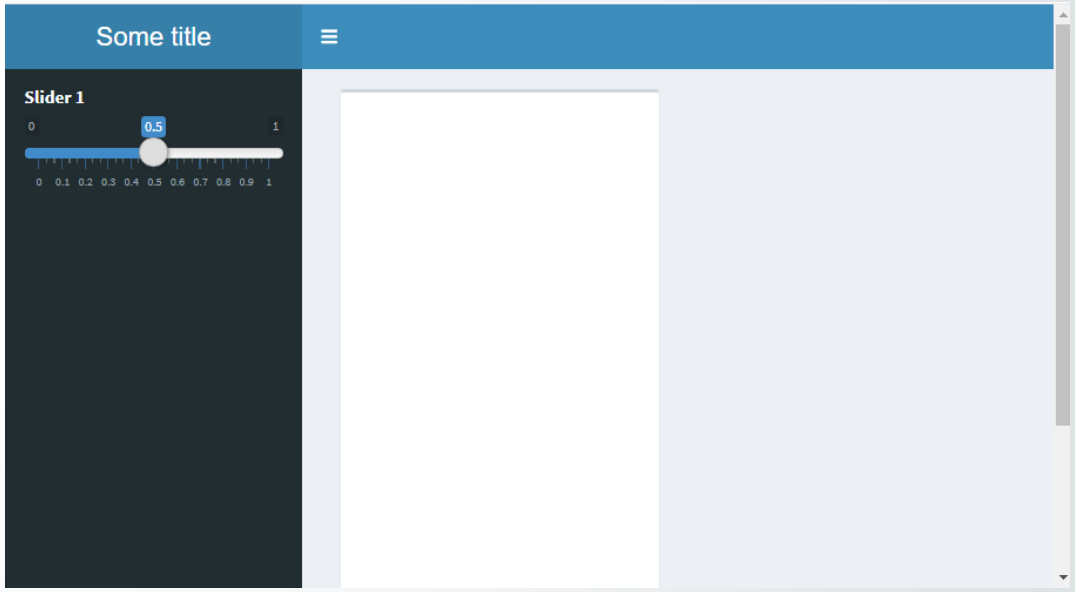
Input parameters, sliders, action buttons….

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(
              sliderInput("slider1", "Slider 1", 0, 1, 0.5)

          ),
        dashboardBody(
         box(
              plotOutput(outputId = "modelPlot")
         )
    )
)
```



More **profesional-looking Shiny apps**

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

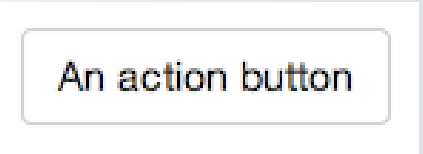
Example: 5_shinydashboard.R

More Widgets

More Widgets

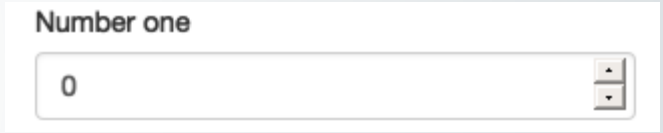
actionButton

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



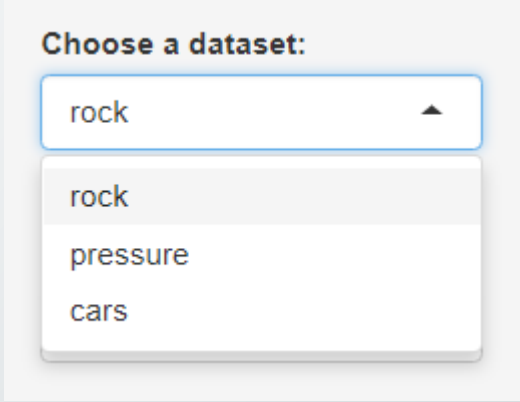
numericInput

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



selectInput

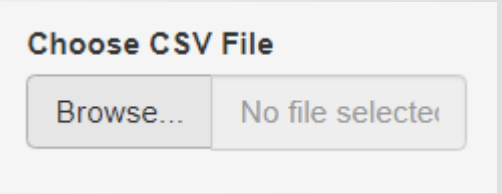
```
selectInput("dataset", "Choose a dataset:",
            choices = c("rock", "pressure", "cars"))
```



More Widgets

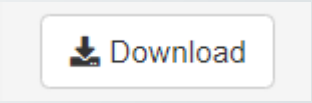
fileInput

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



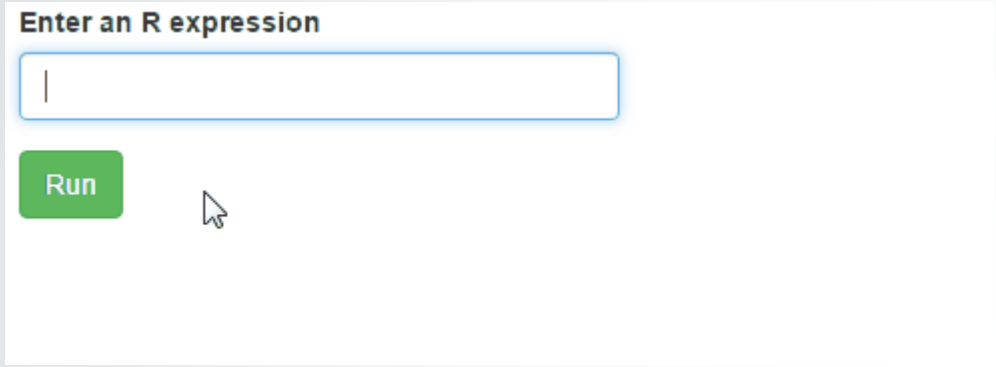
downloadButton

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



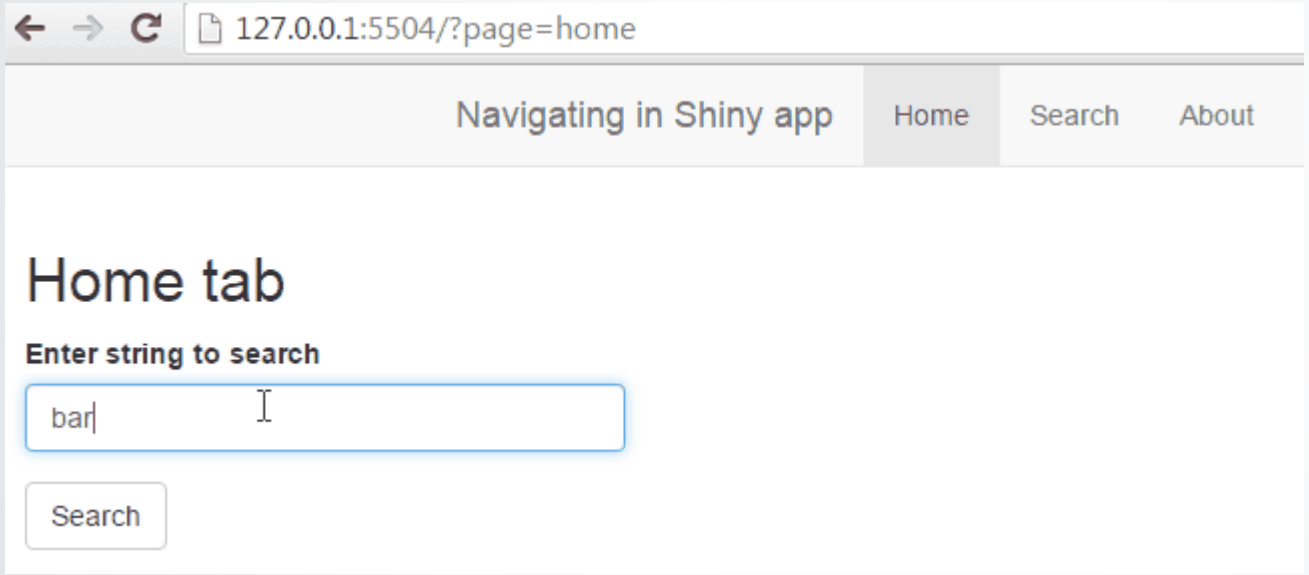
https://shiny.rstudio.com/gallery/widget-gallery.html

… and more widgets



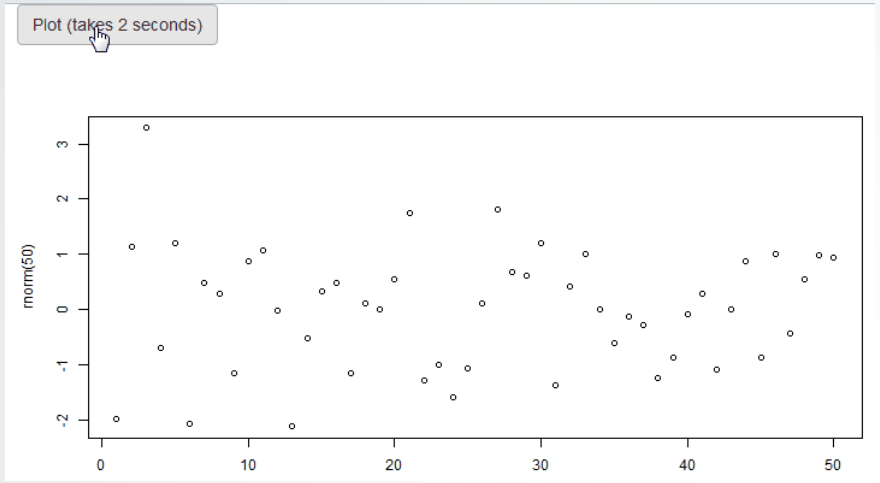
https://github.com/daattali/advanced-shiny#shinyjs

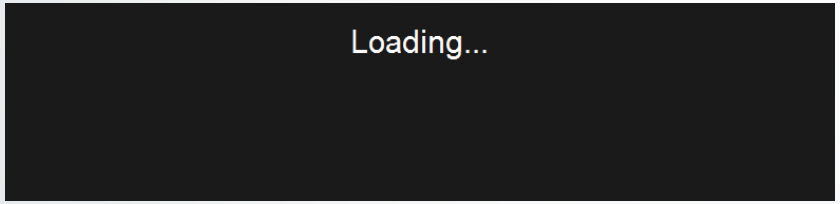
… and more widgets



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

… and more widgets



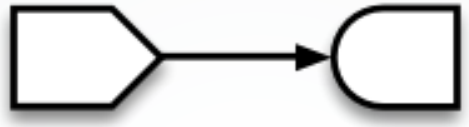


https://github.com/daattali/advanced-shiny#shinyjs

The server …more than an R code

The server ...more than an R code

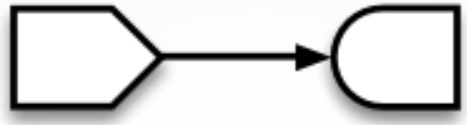
UI



Server

The server ...more than an R code

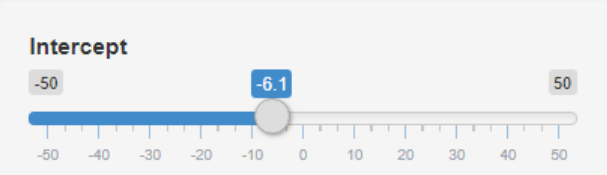
UI

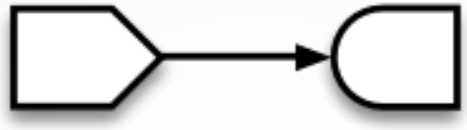


Server

The server …more than an R code

UI



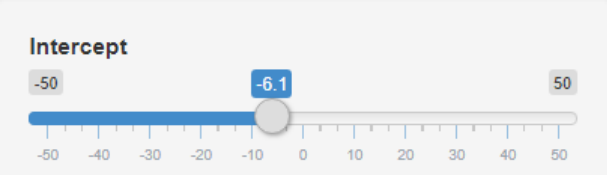


Server

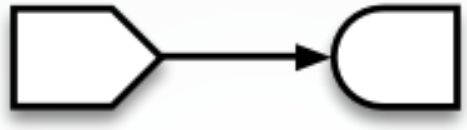
sliderInput(inputId=“intercept” […])

The server ...more than an R code

UI



Server



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

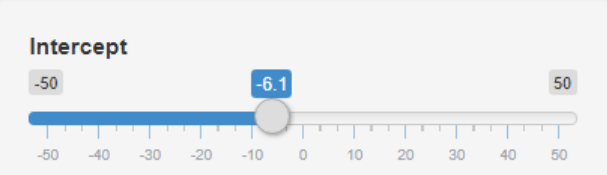
Shiny creates an object:

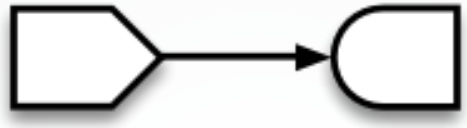
input

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

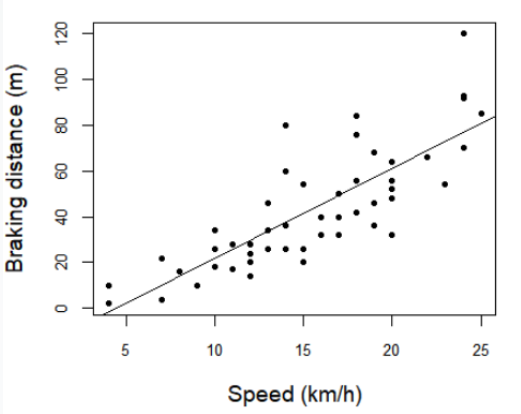
The server ...more than an R code

UI





Server



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

input$intercept ⟶ renderPlot

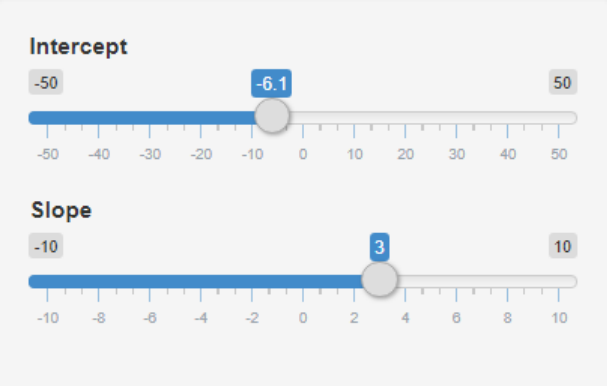
Shiny creates an object:

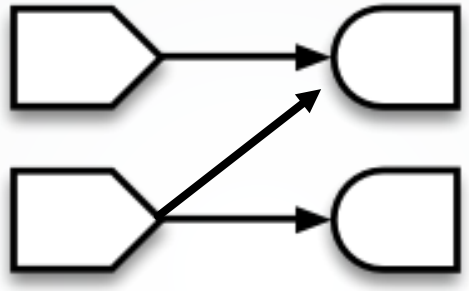
input

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

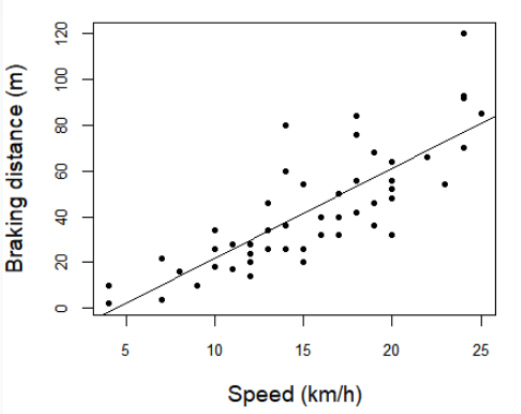
The server ...more than an R code

UI





Server



slope = 3.9

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

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

input$intercept → renderPlot

input$slope → renderText

Shiny creates an object:

input

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

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

The server

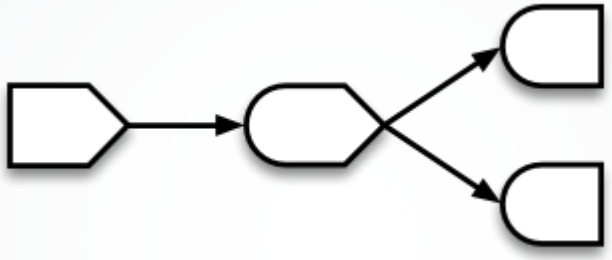
Reactive functions and values

The server

Reactive functions and values

UI

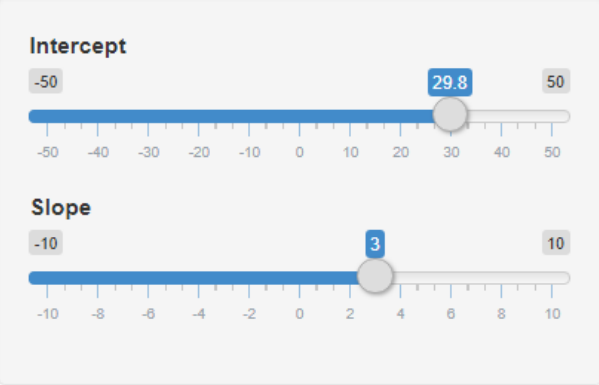
Server



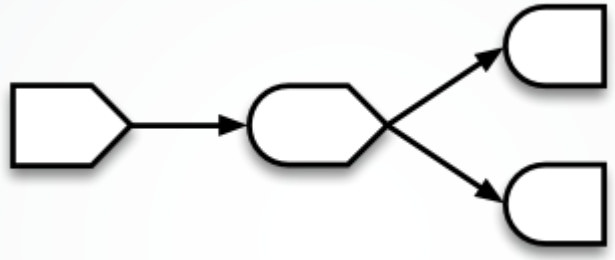
The server

Reactive functions and values

UI



Server

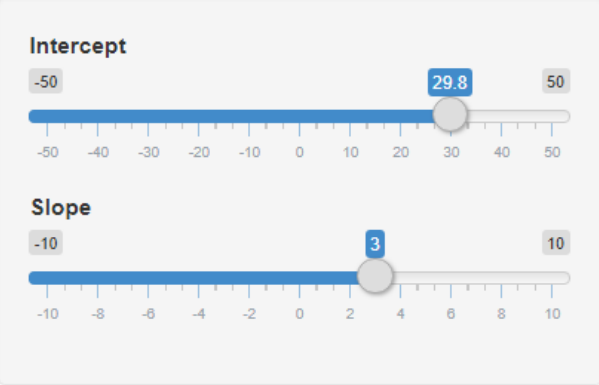


The user updates a parameter in the UI

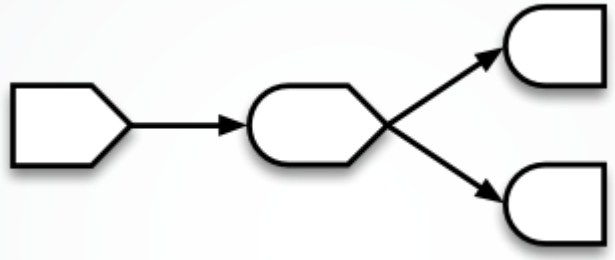
The server

Reactive functions and values

UI



Server



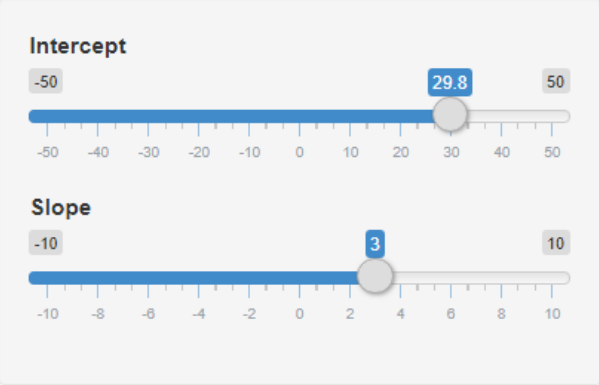
The user updates a parameter in the UI

Shiny creates/modifies the “input” object

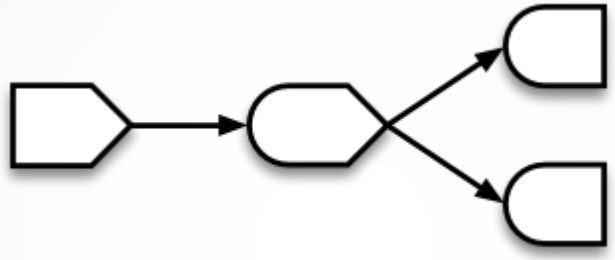
The server

Reactive functions and values

UI



Server



The user updates a parameter in the UI

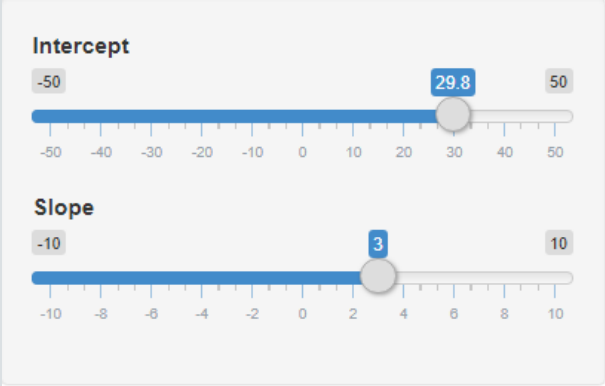
Shiny creates/modifies the “input” object

The “observer” detects what parameter has changed

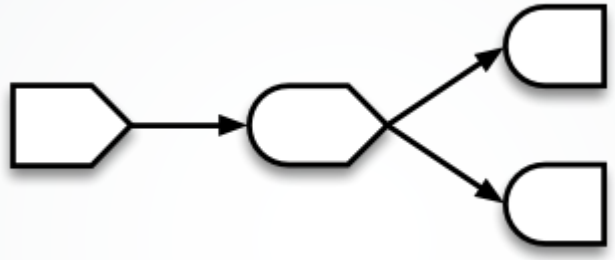
The server

Reactive functions and values

UI

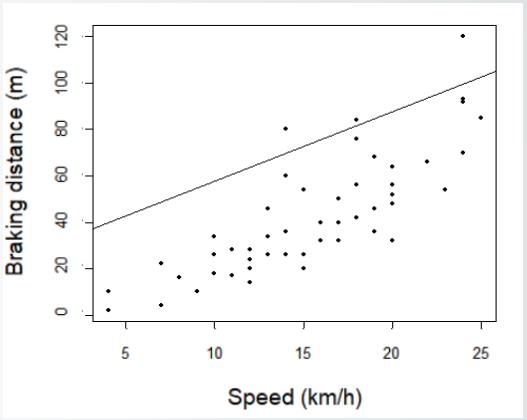


The user updates a parameter in the UI



Shiny creates/modifies the “input” object

Server



slope = 3.0

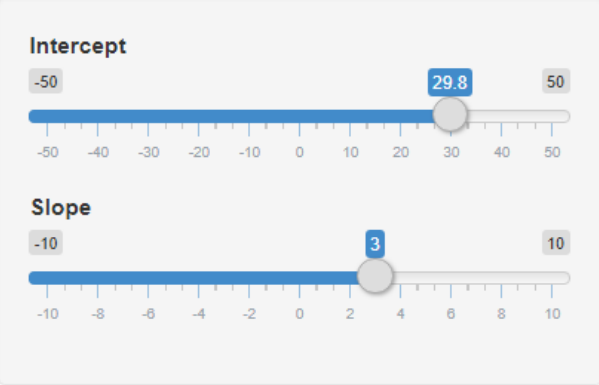
The output is updated only runing the code involving this parameter

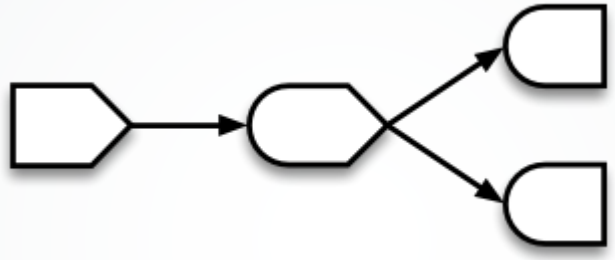
The “observer” detects what parameter has changed

The server

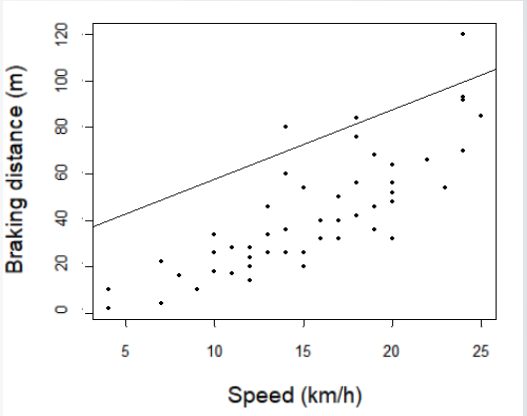
Reactive functions and values

UI





Server



slope = 3.0

The user updates a parameter in the UI

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

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

The output is updated only runing the code involving this parameter

Shiny creates/modifies the "input" object

input

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

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

The "observer" detects what paramerter has changed

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

The server

Reactive functions and values

Example: modeling body temperature of lizards

The server

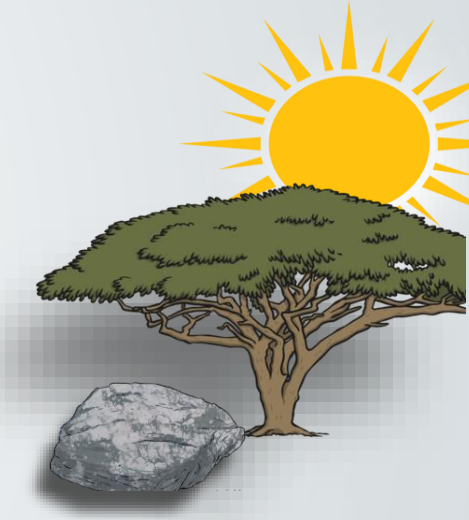
Reactive functions and values


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

        store_lat_lon <- reactiveValues(  store x, y values when the user clicks on the map  )

}
```

The server

Reactive functions and values

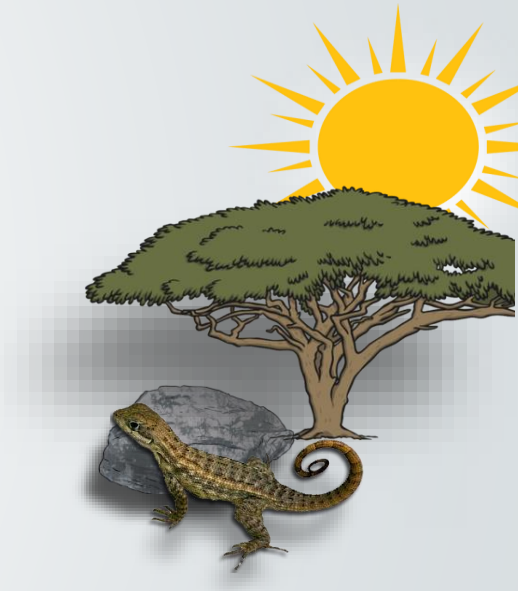

```
server <- function(input, output) {

    store_lat_lon <- reactiveValues(  store x, y values when the user clicks on the map  )

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

}
```



The server

Reactive functions and values

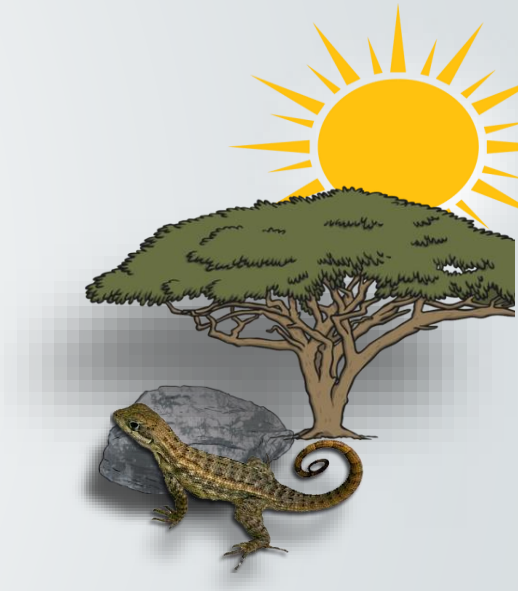

```
server <- function(input, output) {

    store_lat_lon <- reactiveValues(  store x, y 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 })

}
```

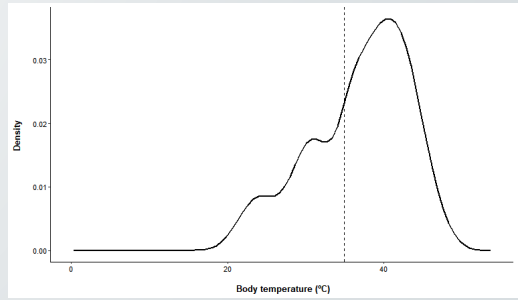


The server

Reactive functions and values

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

    store_lat_lon <- reactiveValues(  store x, y 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
    })

}
```

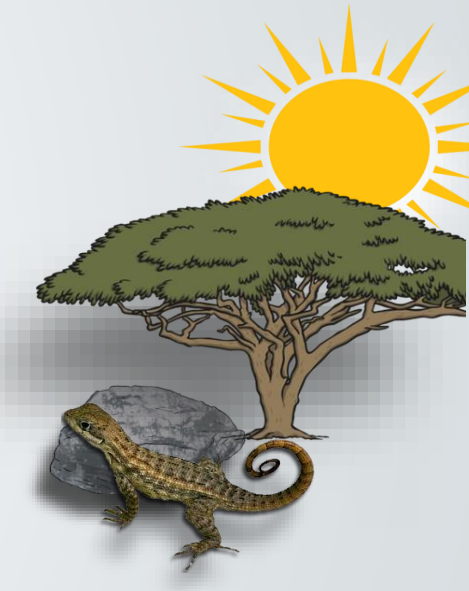


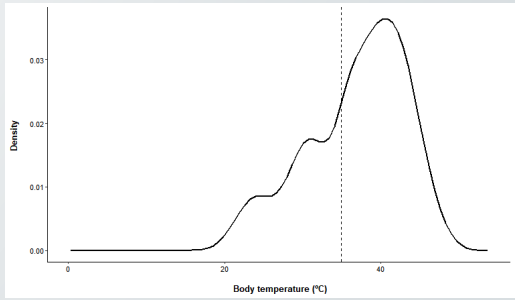


The server

Reactive functions and values


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

        store_lat_lon <- reactiveValues(  store x, y 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
        })

}
```



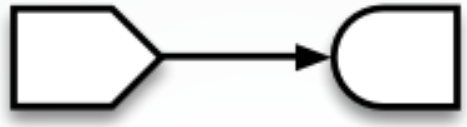


The server observeEvent

The server observeEvent

UI

Server

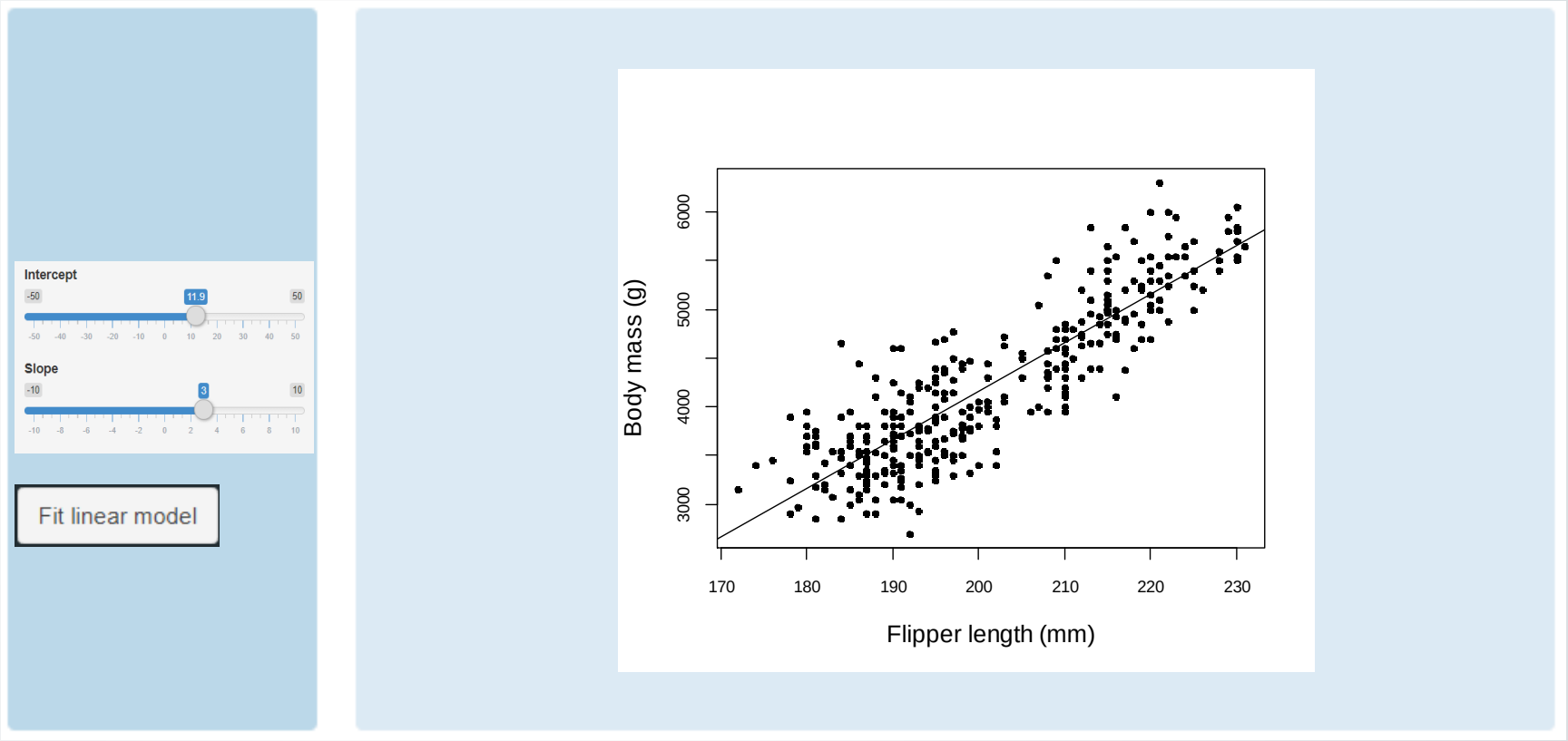


actionButton(“press_button”)

observeEvent(input$press_button { do something })

The server

observeEvent



Example: 7_Observe_event.R

The server

observeEvent

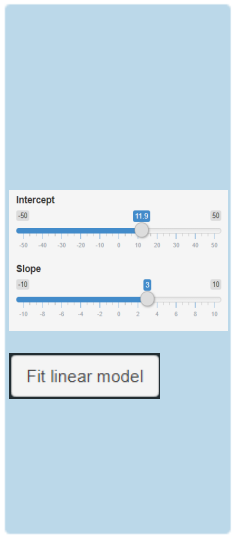
```
ui <- dashboardPage(
        dashboardHeader(title = " Some title"),

        dashboardSidebar(

        ),

        dashboardBody(
        box(

          )
       )
)
```

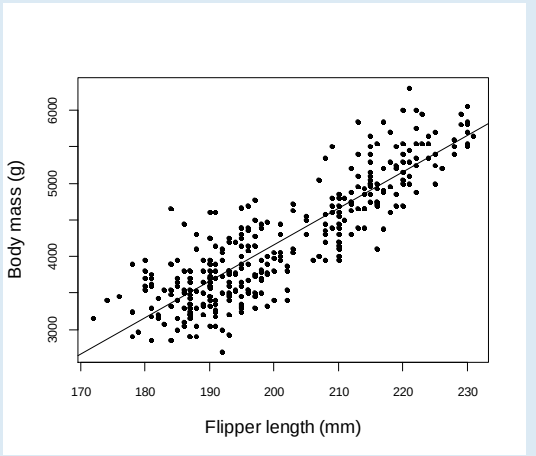
Input parameters, sliders, action buttons….

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")
           )
        )
)
```





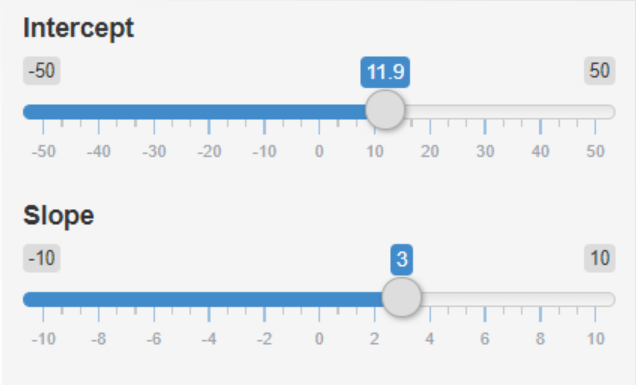
The server observeEvent

```
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)
    })

}
```

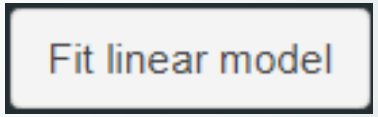


If the user **moves the sliders**, update the stored parameter values

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



The server observeEvent

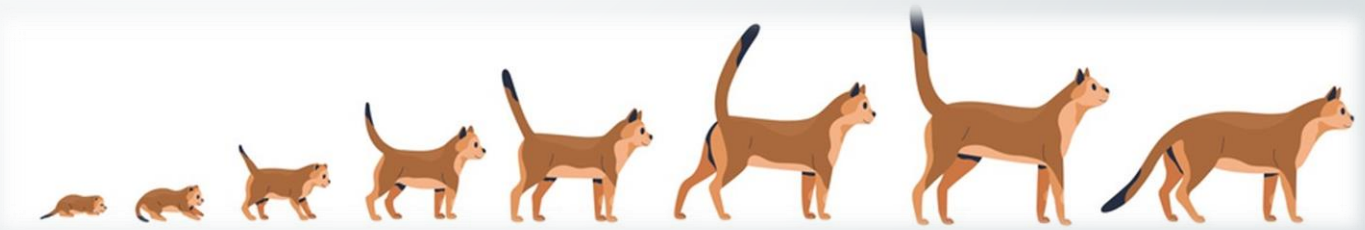
```
server <- function(input, output) {

        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)
        })
}
```

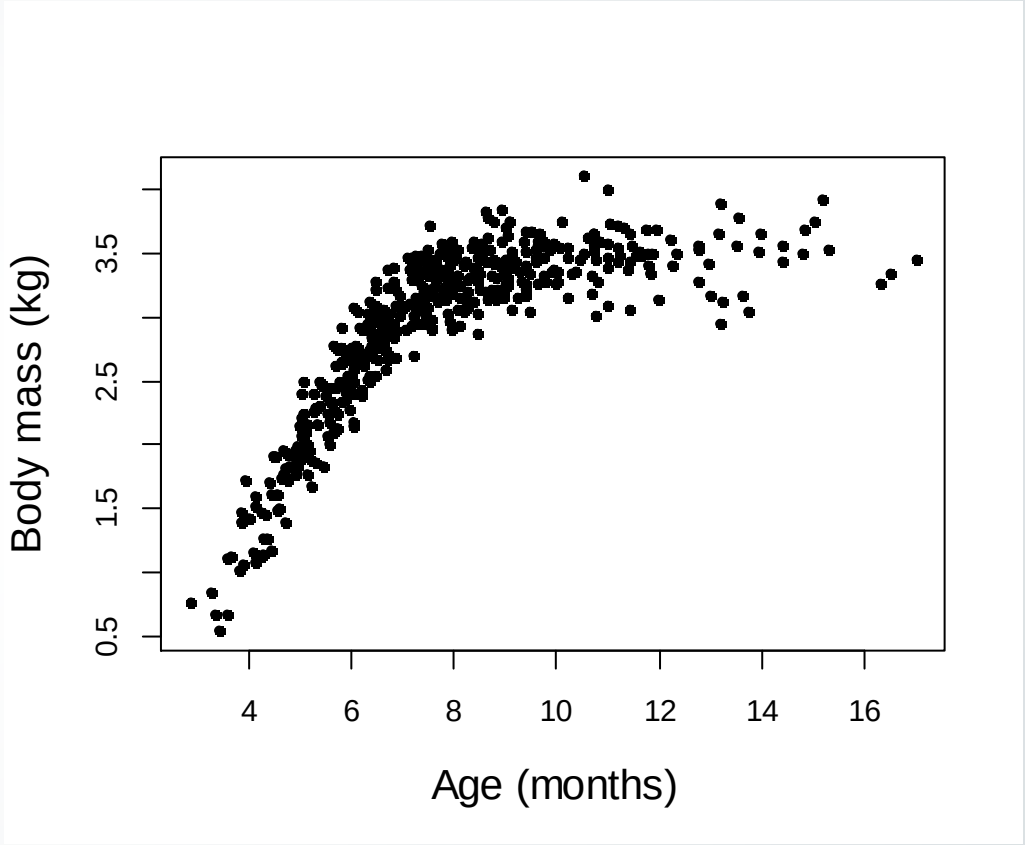
Fitting **nonlinear models**

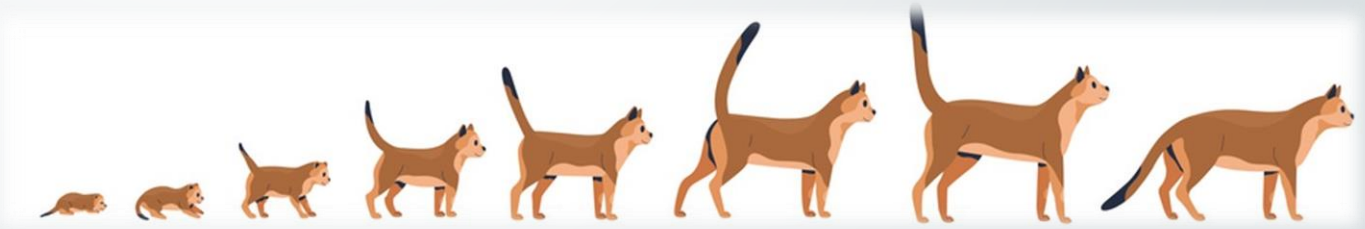
observeEvent



Fitting **nonlinear models**

observeEvent

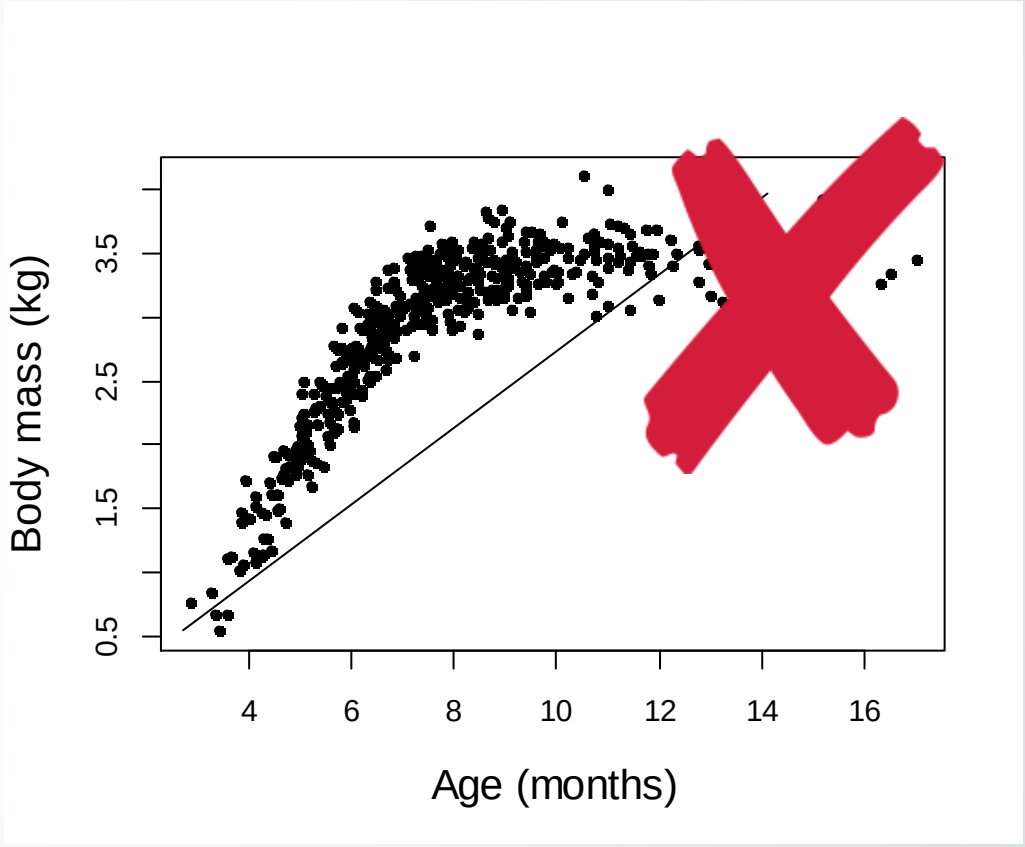


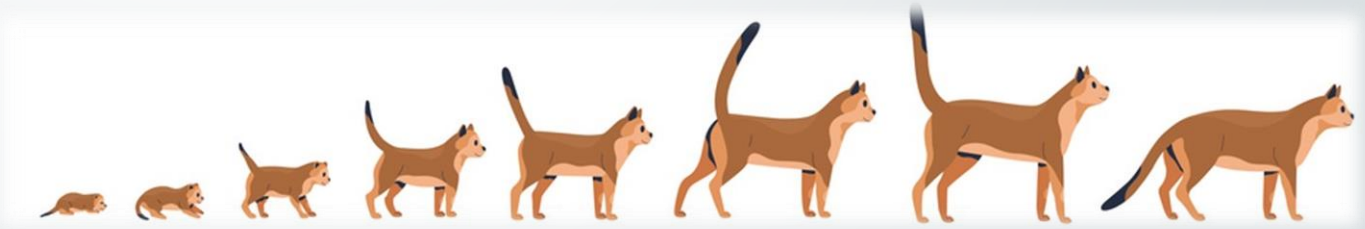


Fitting **nonlinear models**

observeEvent

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

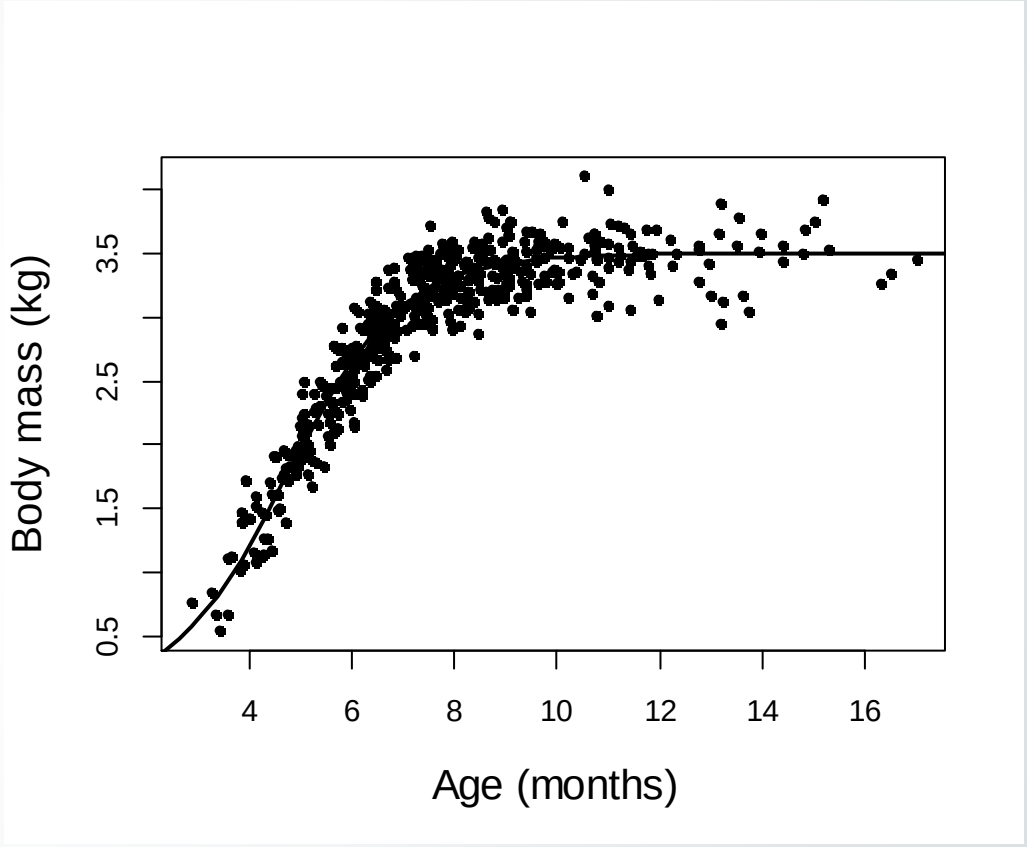


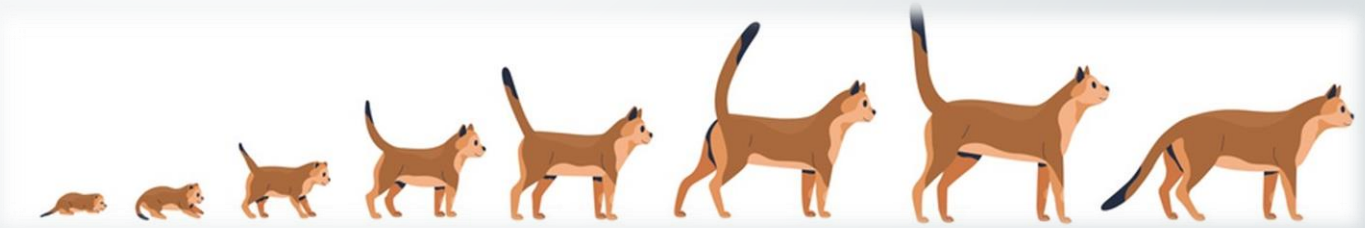


Fitting **nonlinear models**

observeEvent

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



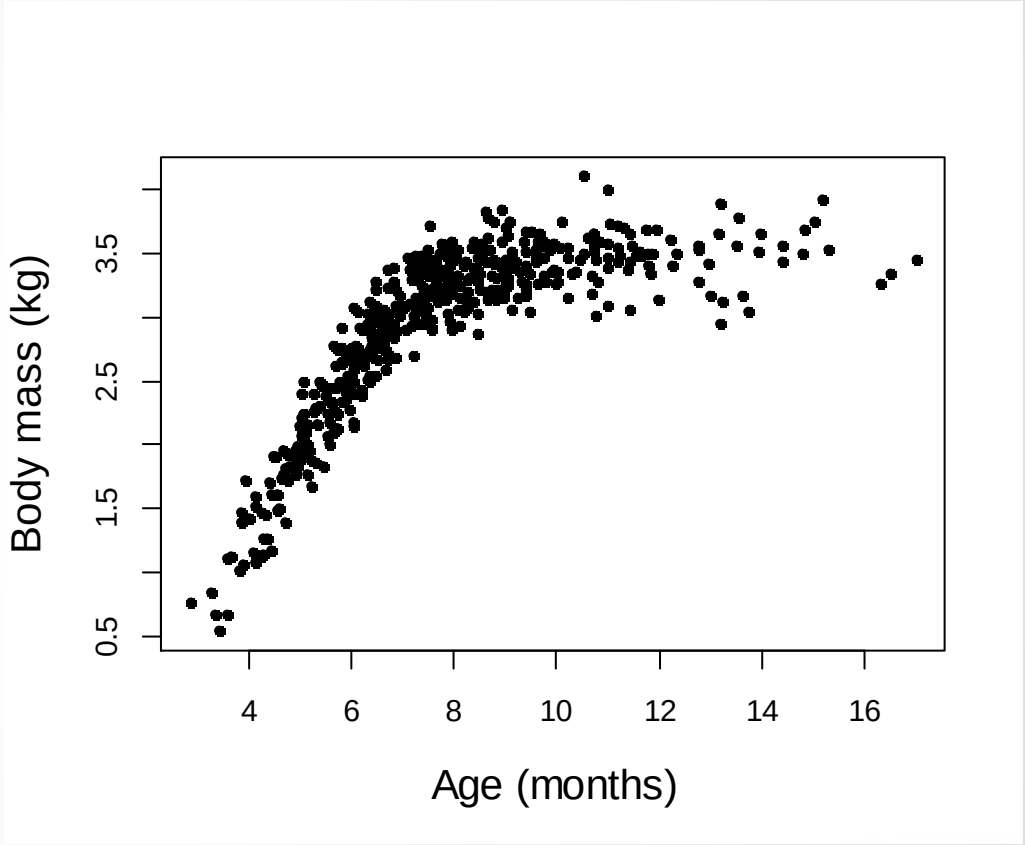


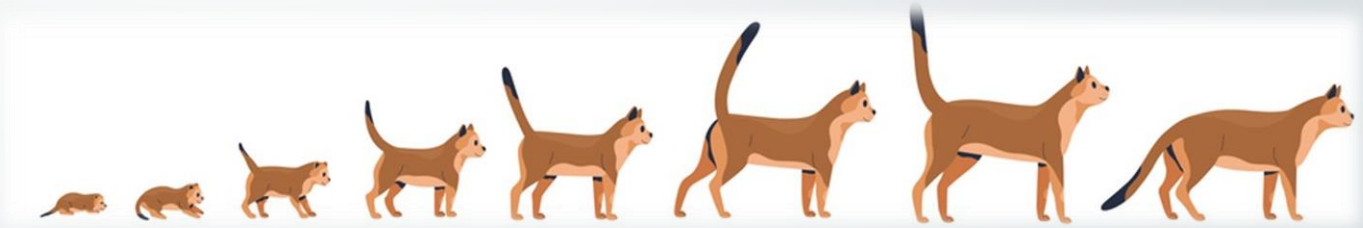
Fitting **nonlinear models**

observeEvent

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

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





Fitting **nonlinear models**

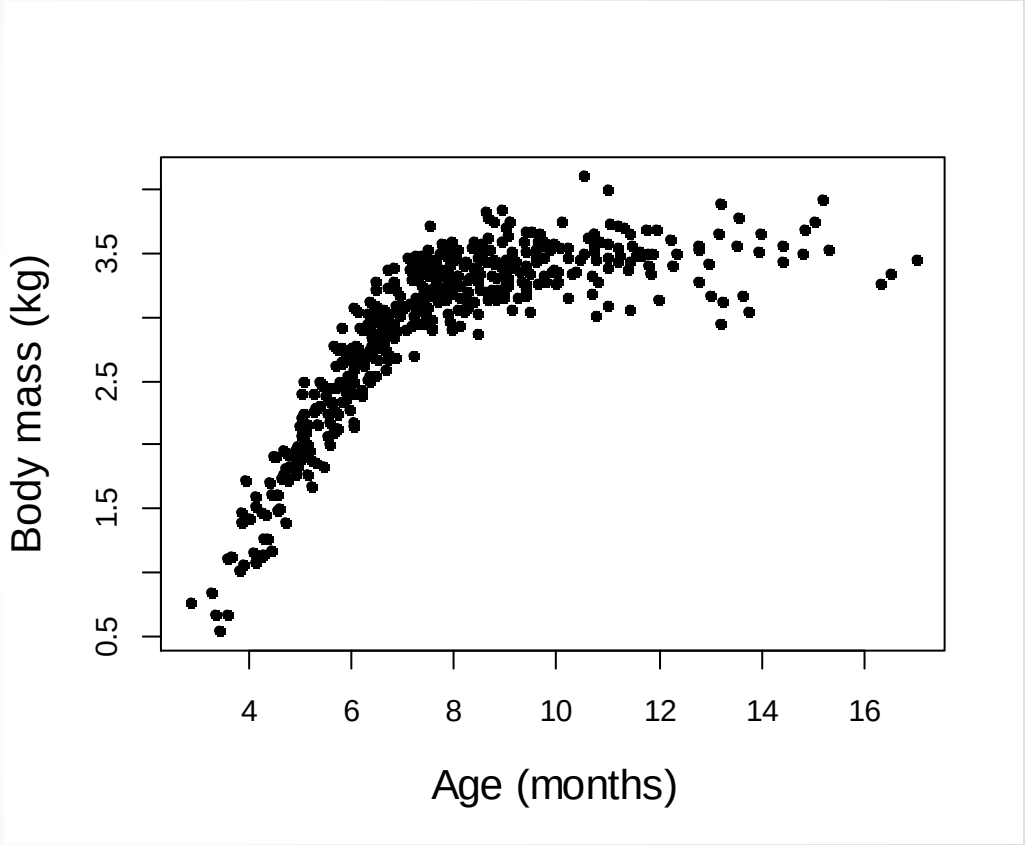
observeEvent

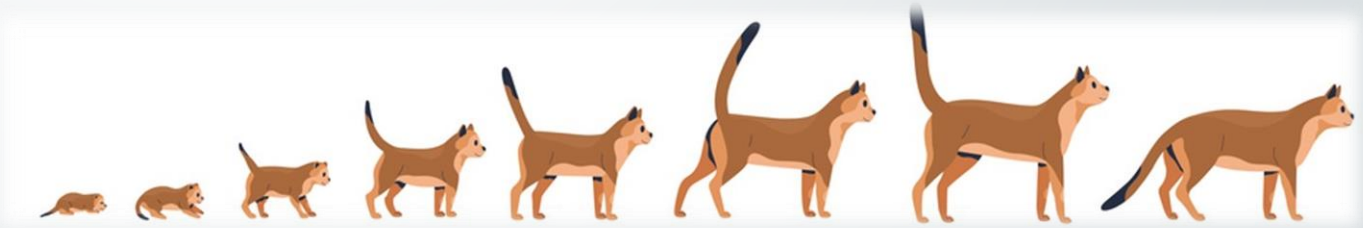
```
logistic_model <- function(a, b, c, x){
        y <- c / (1 + exp(a – b * x))
        return(y)
}
```

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

=

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



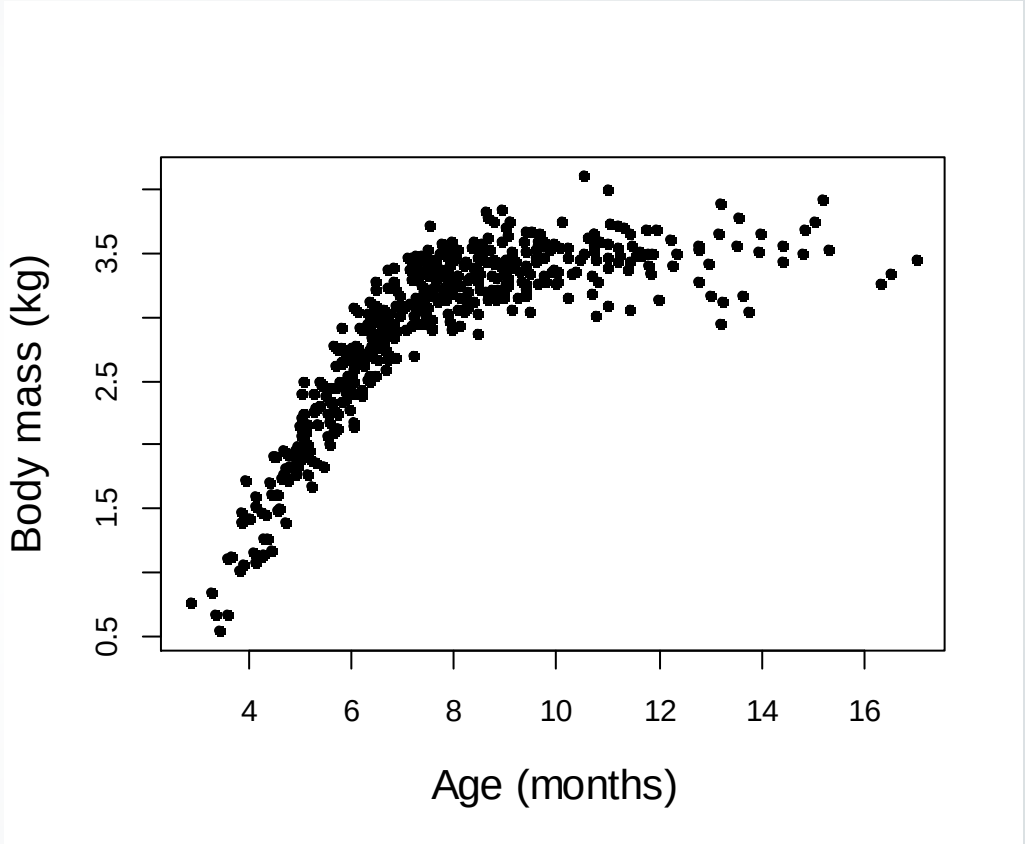


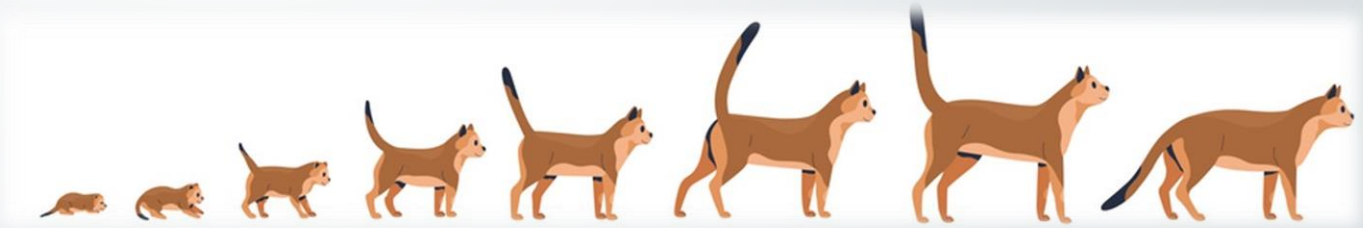
Fitting **nonlinear models**

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))
```

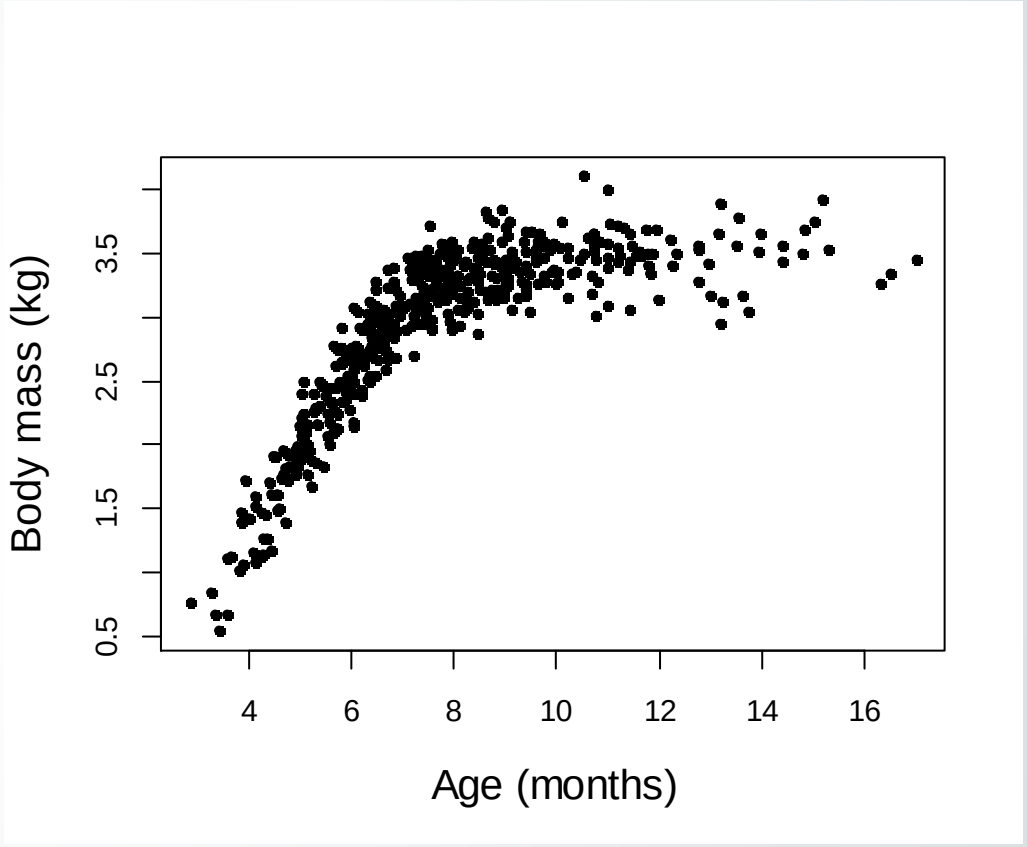


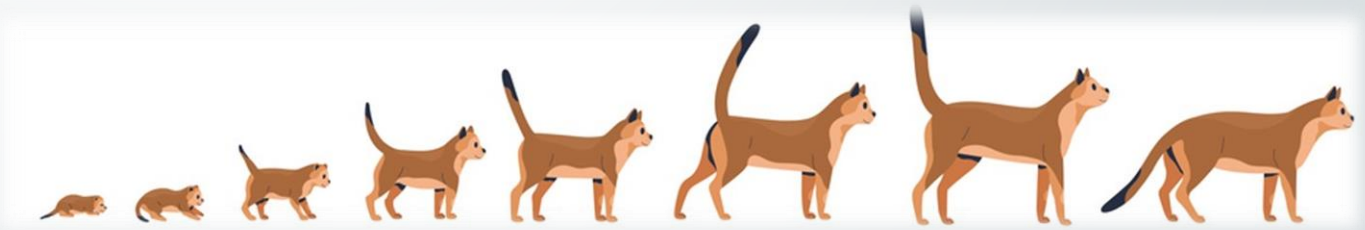


Fitting **nonlinear models**

observeEvent

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

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

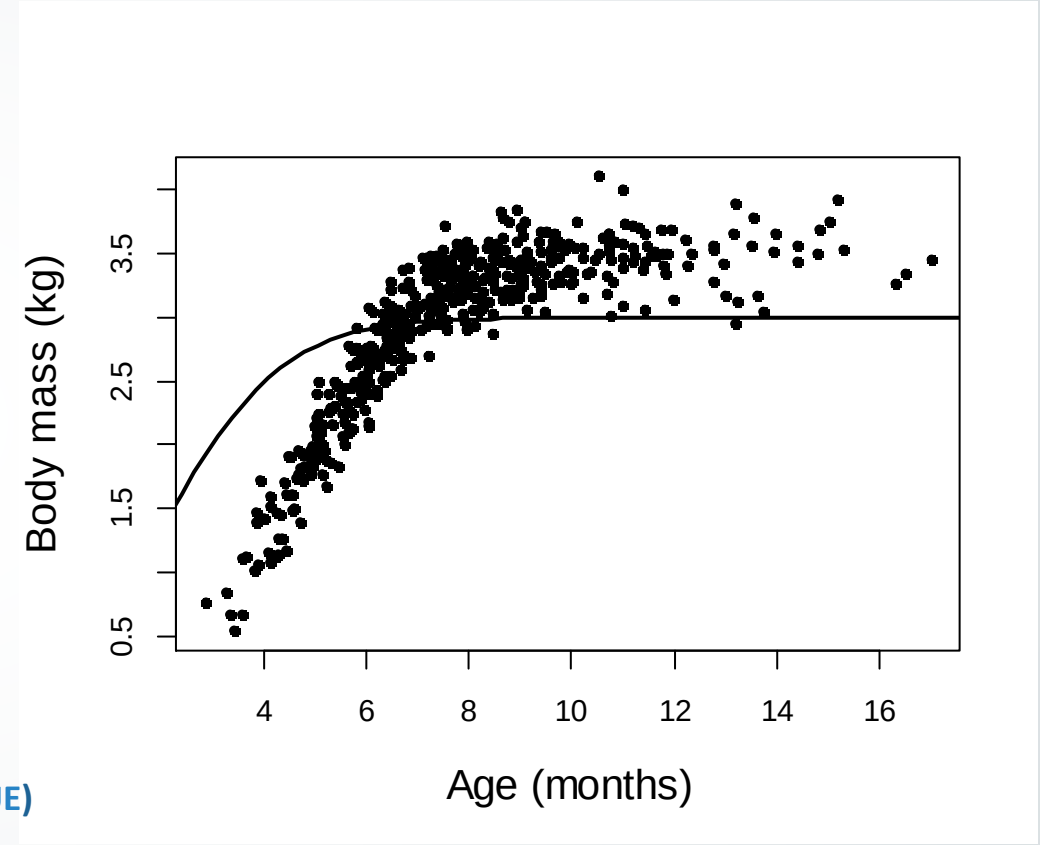




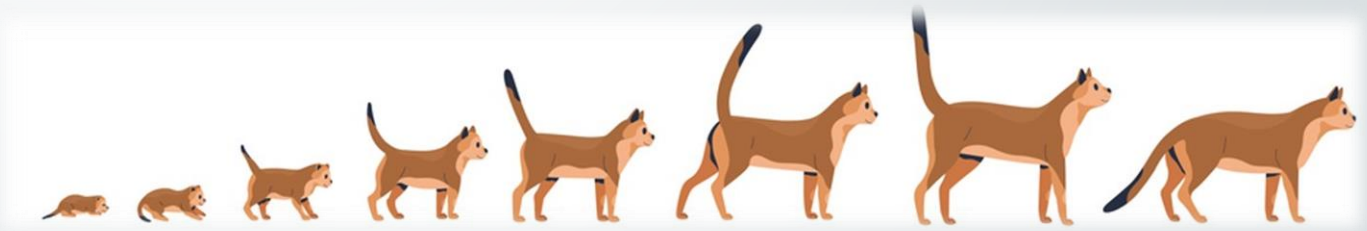
Fitting **nonlinear models**

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

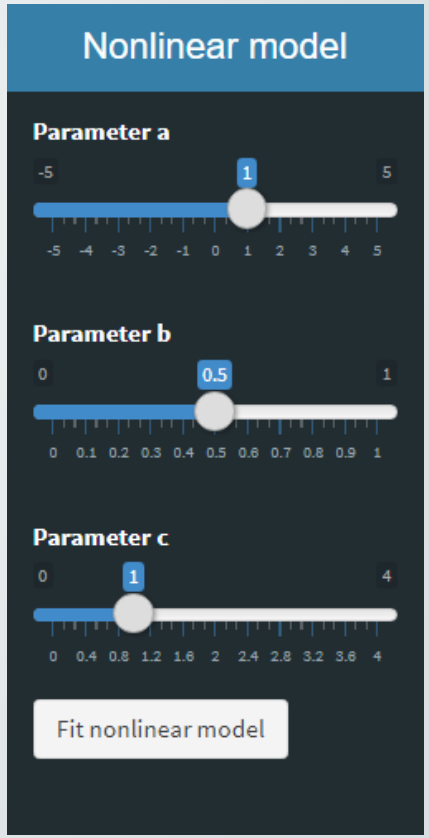
Help us, Shiny!

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$fit_nls2,{
                mod <- nls2(mass ~ logistic_model(a, b, c, age), simulated_data,
                            start = list(a=stored_parameters$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)
        })

}
```

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)
        })

}
```

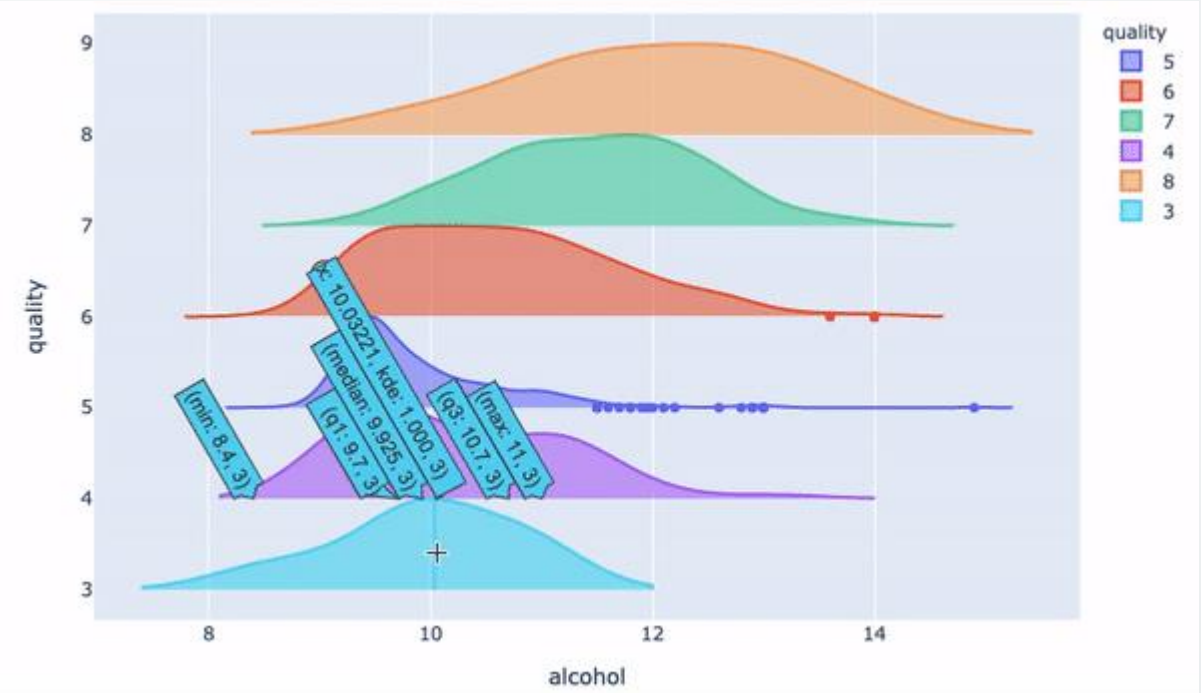
Fitting **nonlinear models**

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)
        })

}
```

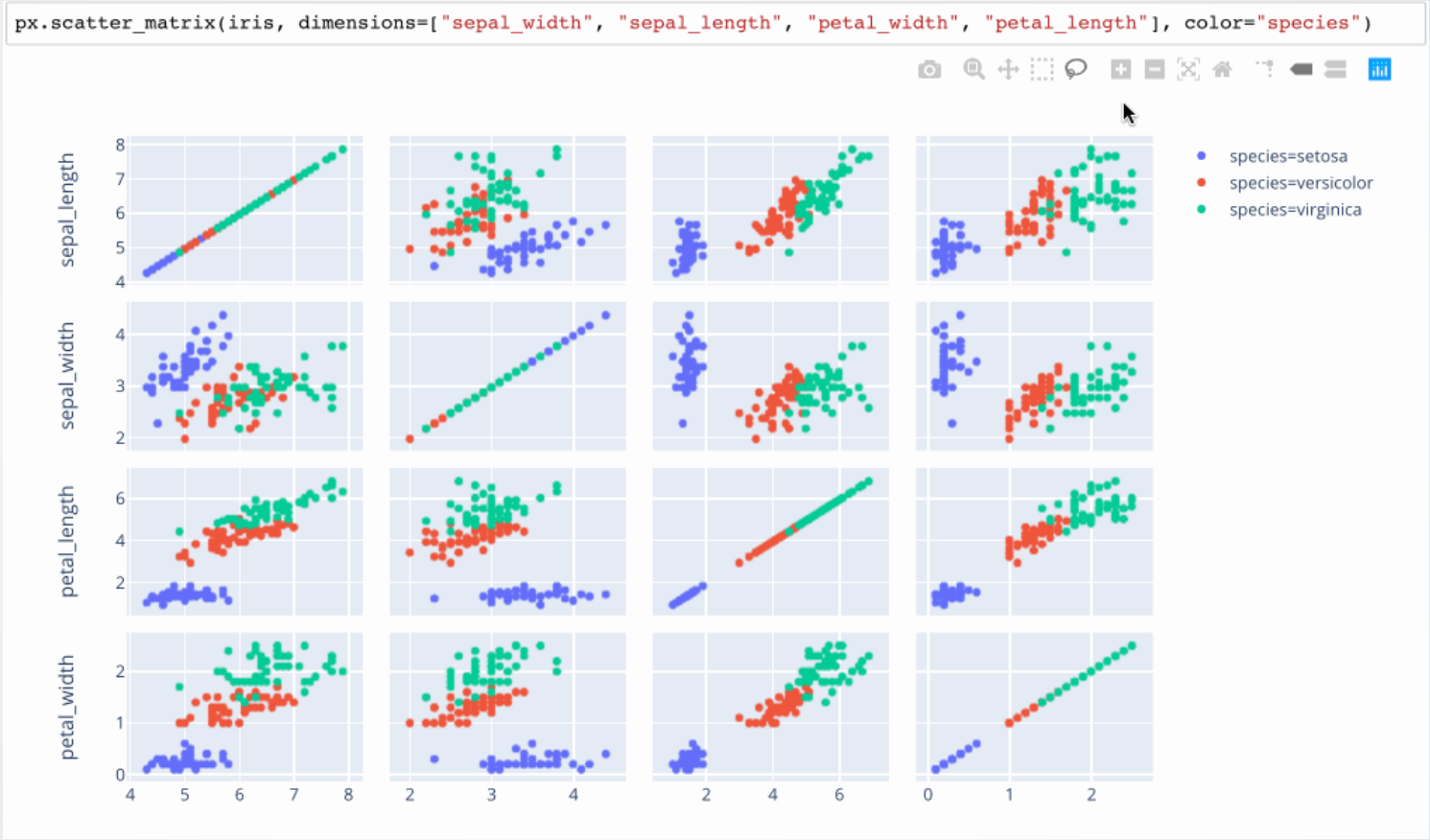
Making Shiny **more** interactive

Making Shiny **more** interactive



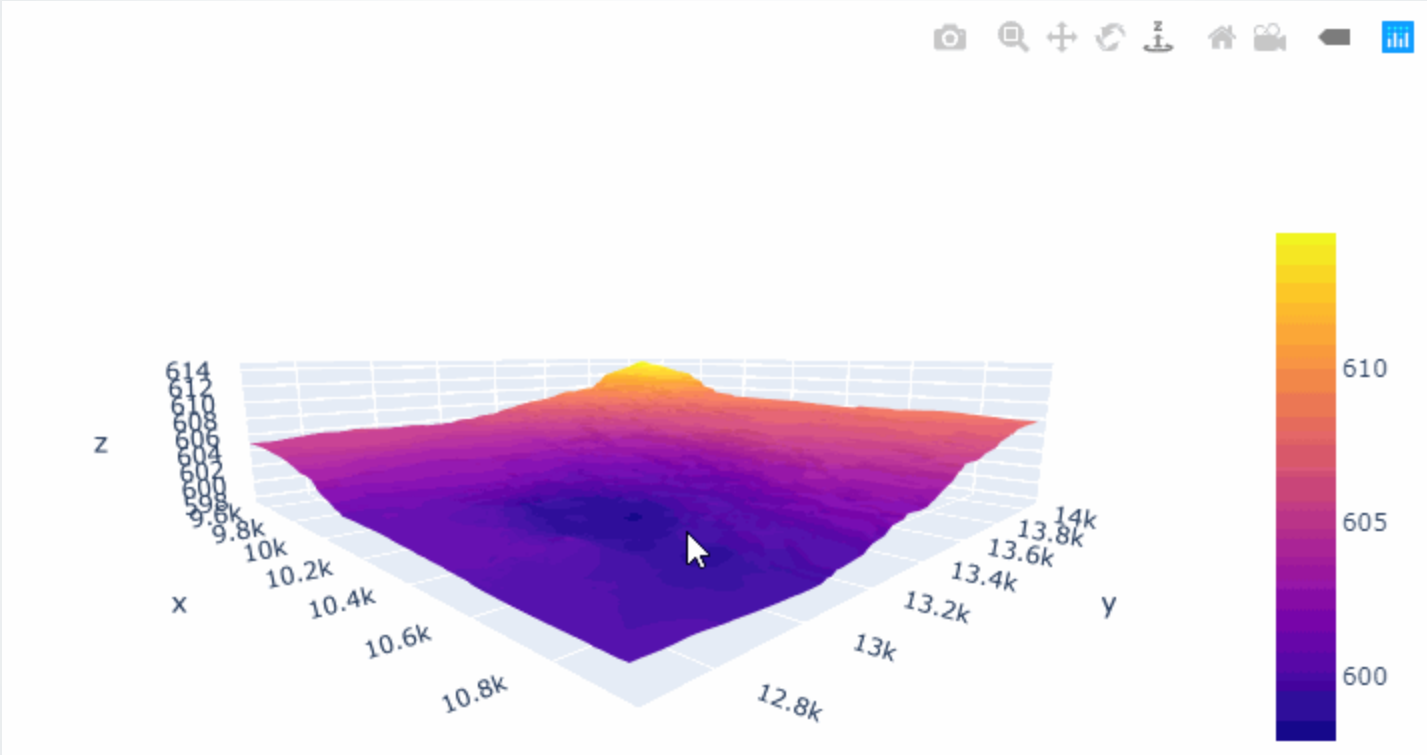
https://plotly-r.com/

Making Shiny **more** interactive



https://plotly-r.com/

Making Shiny **more** interactive



https://plotly-r.com/

Making Shiny **more** interactive

```
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)
```

plotlyOutput("plot")

output$plot <- renderPlotly({ })

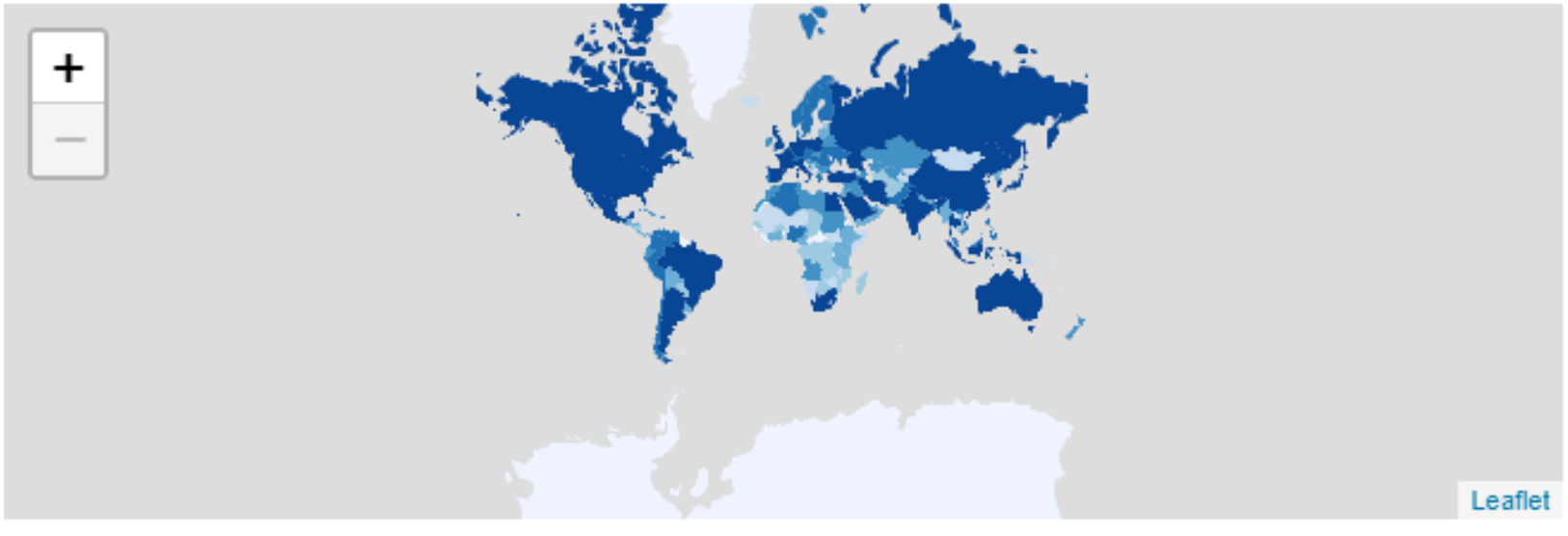
Making Shiny **more** interactive

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

https://leafletjs.com/

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))
```



https://leafletjs.com/

Launching **Shiny Apps**

Launching **Shiny Apps**

library(rsconnect)

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

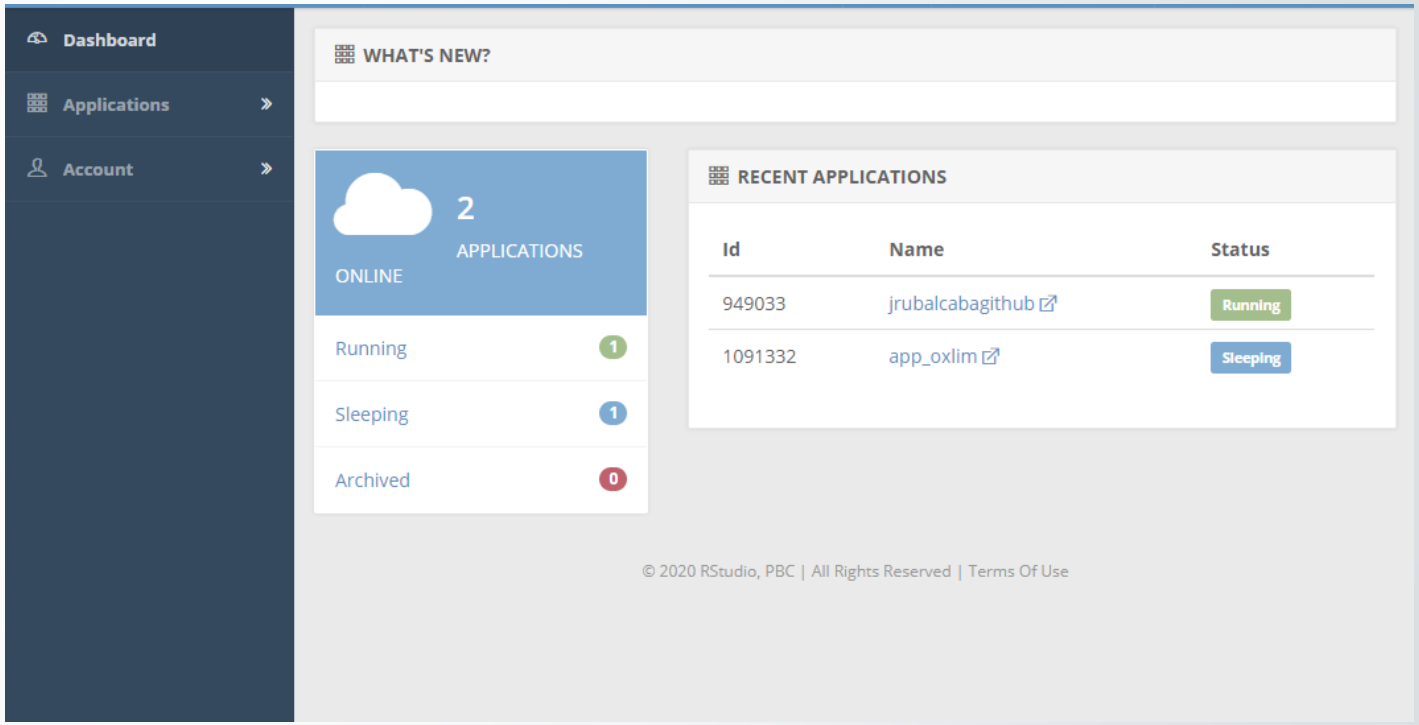
* make sure that your app R file is named "app.R"

Shinyapps.io

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

Launching **Shiny Apps**

Shinyapps.io



Launching **Shiny Apps**

Run from GitHub

Create a GitHub repository and include the file ‘app.R’

runGitHub(“repo_name”)

Launching **Shiny Apps**

Run from GitHub

Create a GitHub repository and include the file ‘app.R’

runGitHub(“repo_name”)

Virtual machine – AWS/RStudio