Créer une application RShiny

Florent Veillon - Ingénieur d'études - EspaceDev - IRD

Antananarivo - Madagascar - Octobre 2023

EspaceDEV
OBSERVATION SPATIALE, MODÈLES & SCIENCE IMPLIQUÉE

Institut de Recherche pour le Développement
FRANCE

Téléchargement des packages

```
install.packages(“shiny”)
install.packages(“sf”)
install.packages(“leaflet”)
install.packages(“DT”)
install.packages(“plolty”)
```

Chargement du package

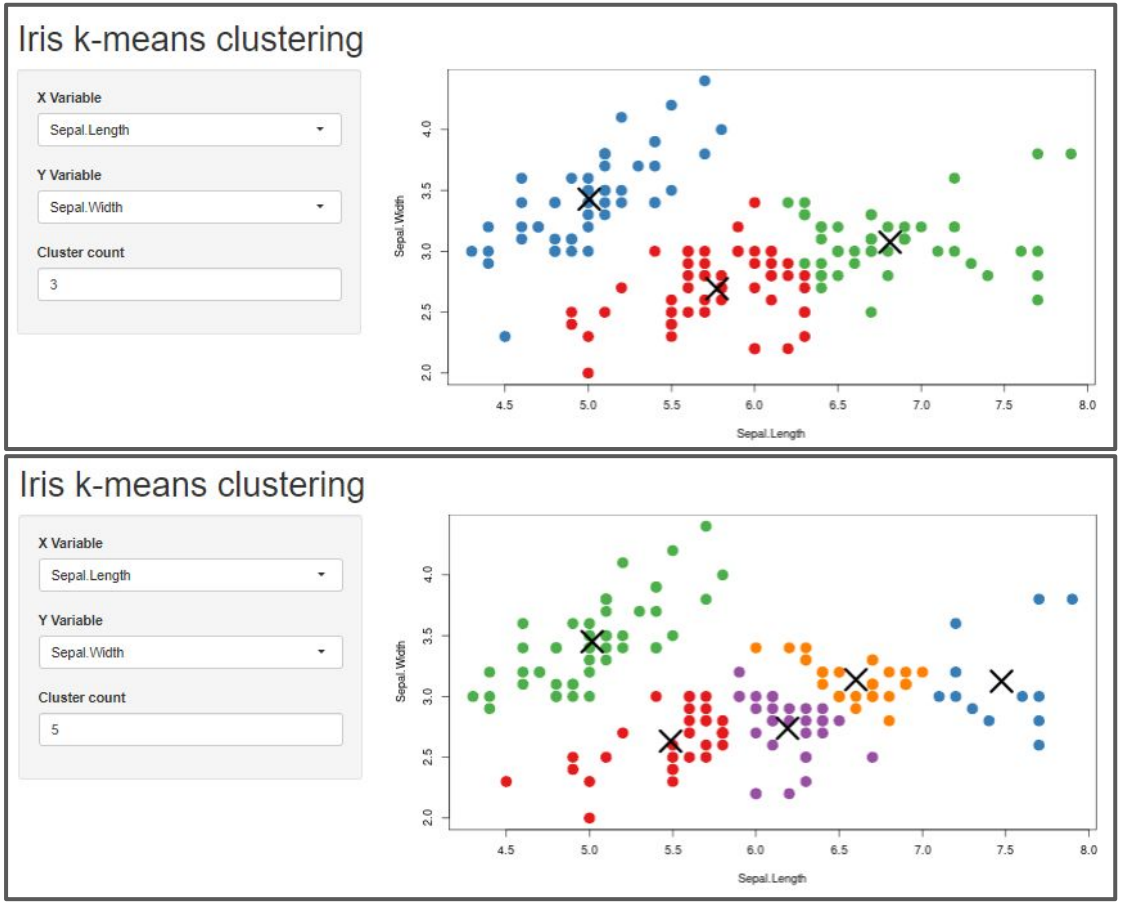
```
library(“sf”)
...
```

R Shiny ?

- **Package R** open source
- Permet de créer des **applications Web interactives**
- Utilisable par plusieurs personnes simultanément
- Requiert seulement un navigateur web pour l'utilisation

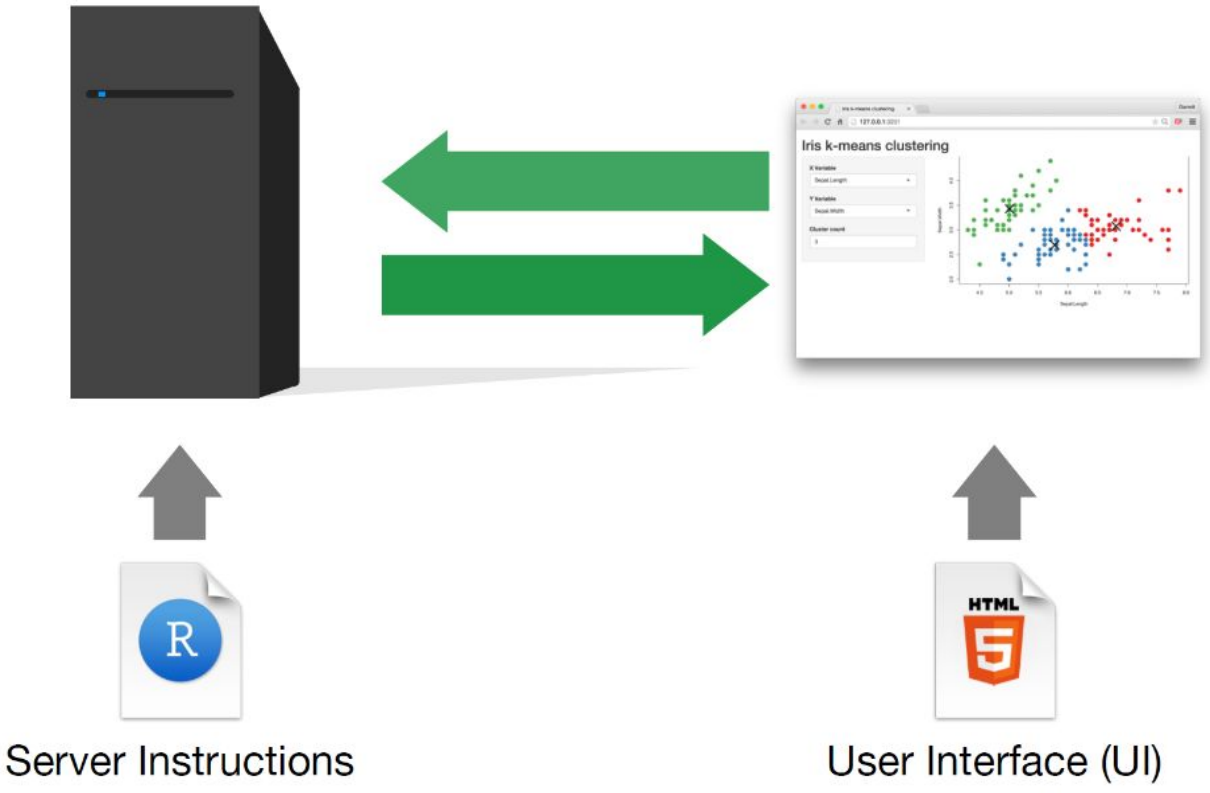
- Shiny interprète du langage R pour **générer** du HTML, CSS et Javascript ⇒ *possibilité d'intégrer des actions Javascript et CSS*
- Grand nombre d'exemples d'applications Shiny : *https://shiny.posit.co/r/gallery/*

R Shiny ?

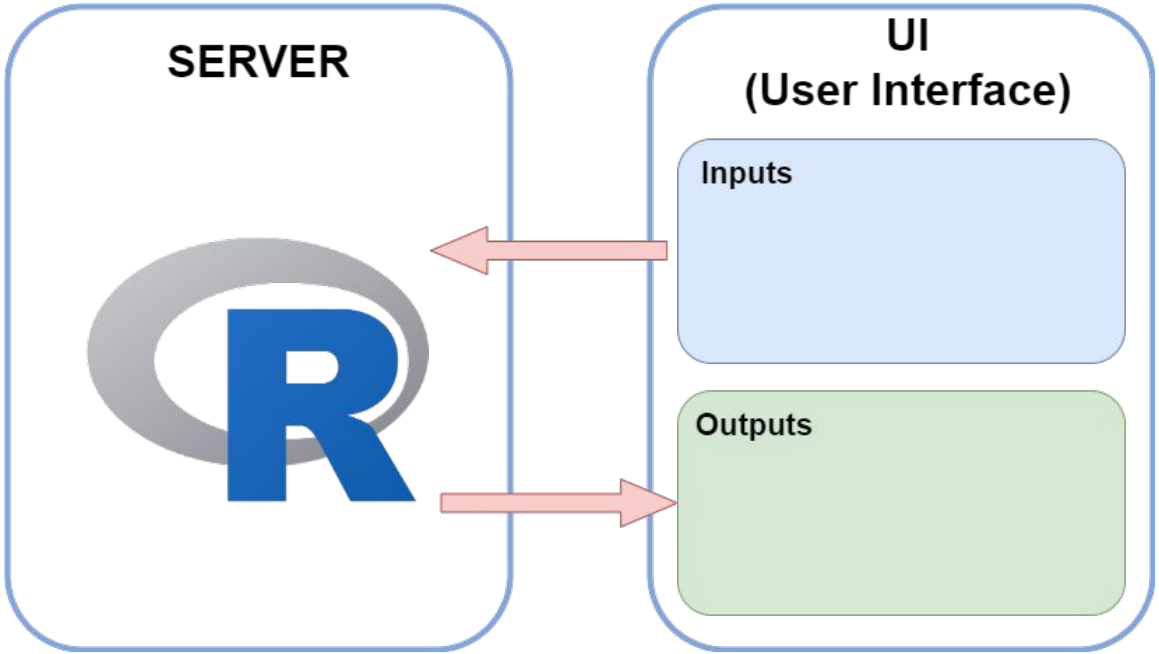

Iris k-means clustering
X Variable
Sepal.Length
Y Variable
Sepal.Width
Cluster count
3
Sepal.Width
Sepal.Length
Iris k-means clustering
X Variable
Sepal.Length
Y Variable
Sepal.Width
Cluster count
5
Sepal.Width
Sepal.Length

Fonctionnement de R Shiny

Fonctionnement de R Shiny

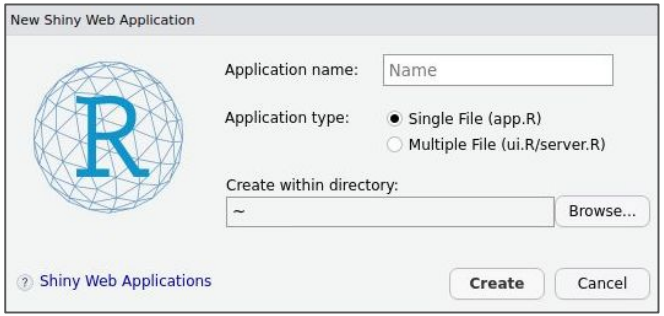


Fonctionnement de R Shiny



Par où commencer

- Ouvrir RStudio
- Installer le package shiny : `install.packages("shiny")`
- Allez vers `File` > `New File` > `Shiny Web App`



- Entrez un nom d'application
- Gardez l'option single file (*app.R*) sélectionnée
- Entrez le chemin où l'application sera sauvegardée

Le fichier *app.R* devrait s'ouvrir puis cliquez sur *Run App* pour voir le résultat (code exemple).

Squelette d’une application Shiny

```
# Chargement du package shiny
library(shiny)

# Définition du UI de l’application
ui <- fluidPage(
)

# Définition du server de l’application
server <- function(input, output) {
}

# Création et exécution de l’application
shinyApp(ui = ui, server = server)
```

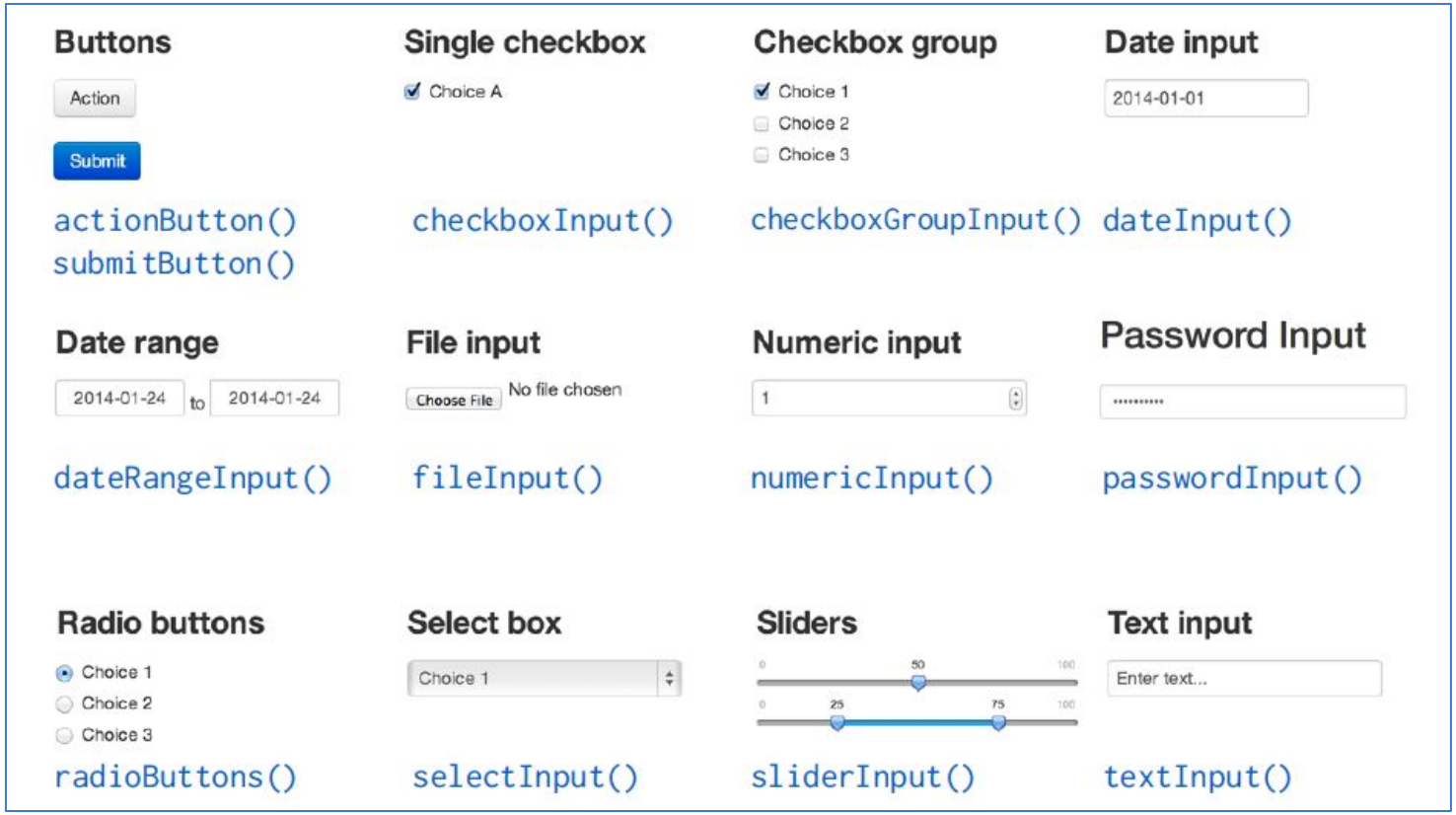
- La fonction `fluidPage()` **créer une interface utilisateur** HTML dynamique que vous voyez lors du chargement d’une application RShiny. La convention consiste à enregistrer cette interface sous la forme d'un objet nommé **ui**.

- La fonction `server()` est définie par l'utilisateur et contient **les commandes R** dont votre ordinateur a besoin pour **exécuter l'application**.

- La fonction `shinyApp()` **construit l'application** sur la base de l'interface utilisateur et des lignes de codes du serveur.

User Interface (UI)

User Interface (UI) : inputs



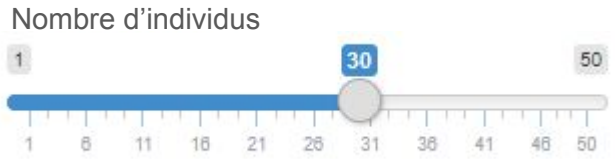
Accès à la valeur sélectionnée par l'utilisateur à travers un identifiant unique (**`inputid`**)
⇒ renseigné comme argument

Assortiment d’inputs

```
# Chargement du package shiny
library(shiny)

# Définition du UI de l’application
ui <- fluidPage(
)

# Définition du server de l’application
server <- function(input, output) {
}

# Création et exécution de l’application
shinyApp(ui = ui, server = server)
```

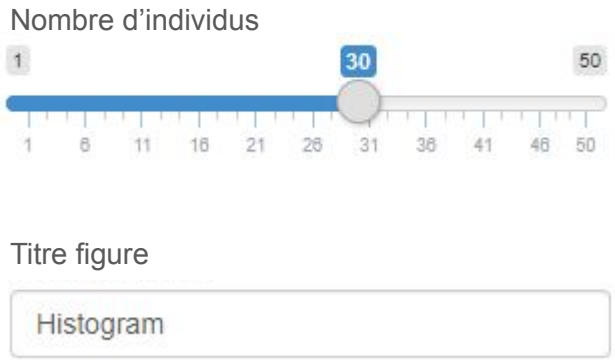
Assortiment d’inputs

```
library(shiny)

ui <- fluidPage(
     # ajout d’un slider
     sliderInput(inputId = "slider1",
                 label = "Nombre d’individus",
                 min = 1, max = 50, value = 30)
)

server <- function(input, output) {
}


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



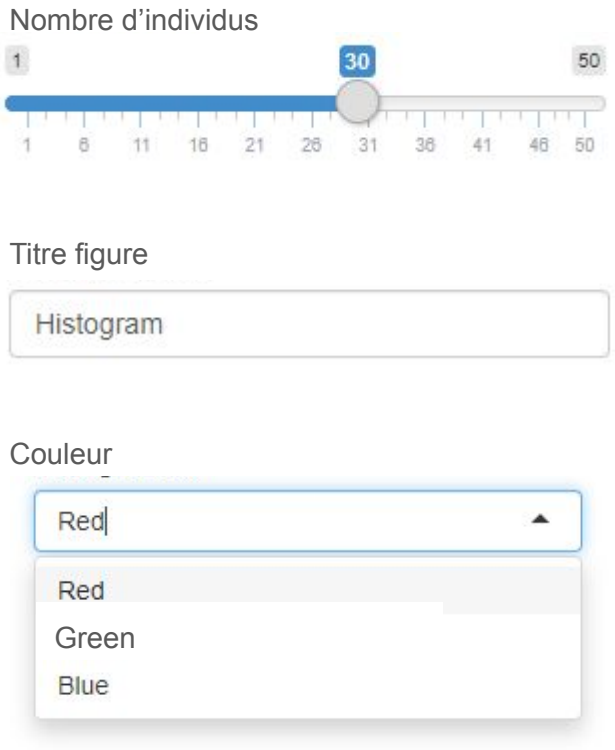
Assortiment d'inputs

```
library(shiny)

ui <- fluidPage(
     # ajout d'un slider
     sliderInput(inputId = "slider1",
                 label = "Nombre d'individus",
                 min = 1, max = 50, value = 30)

),
     # text box input
     textInput(inputId = "text1",
               label = "Titre figure",
               value = "Histogram"),


server <- function(input, output) {
}


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



Assortiment d'inputs

```
library(shiny)

ui <- fluidPage(
     # ajout d'un slider
     sliderInput(inputId = "slider1",
                 label = "Nombre d'individus",
                 min = 1, max = 50, value = 30),
     # ajout d'une box texte
     textInput(inputId = "text1",
               label = "Titre figure",
               value = "Histogram"),
     # ajout d'une liste déroulante
     selectInput(inputId = "select1", label = "Couleur",
                 choices = c("Red", "Green", "Blue"),
                 selected = "Red")
),

server <- function(input, output) {
}


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



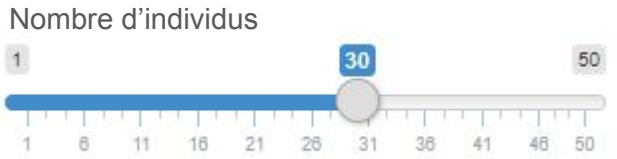
User Interface (UI) : outputs

Output function	Creates
`dataTableOutput()`	data table
`htmlOutput()`	raw HTML
`imageOutput()`	image
`plotOutput()`	plot
`tableOutput()`	table
`textOutput()`	text
`uiOutput()`	raw HTML
`verbatimTextOutput()`	text

Comme pour **les inputs**, les outputs doivent être munis d'un identifiant unique (`outputid=`) ⇒ renseigné comme argument

User Interface (UI) : outputs

```
library(shiny)

ui <- fluidPage(
     # ajout d'un slider
     sliderInput(inputId = "slider1",
                 label = "Nombre d'individus",
                 min = 1, max = 50, value = 30)
     # ajout d'un emplacement pour un graphique
     plotOutput(outputId = "hist")
)

server <- function(input, output) {
}


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



Emplacement du futur graphique

`outputId = "hist"`

`plotOutput(outputId = "hist")` alloue de l'espace pour un graphique.
Ce graphique n'a pas encore été créé, donc il n'est pas encore visible.

User Interface (UI) : Résumé

- Construisez l'interface utilisateur à l'intérieur de la fonction `fluidPage()` et enregistrez-la en tant qu'objet nommé **ui**.
- La fonction `fluidPage()` adapte ses composants en temps réel pour remplir toute la largeur disponible du navigateur - interface utilisateur HTML dynamique.
- Construire les entrées avec ***Input**(inputId, label, ...).
- Construire les sorties avec ***Output**(outputId, ...).
- Séparez les entrées et sorties multiples par des **virgules**.
- Exécutez votre application après chaque entrée ou sortie ajoutée afin de minimiser les complications ultérieures.

Construction du UI de notre application Shiny

- Application sur l'étude des **cyclones** à Madagascar
- **Scinder l'écran en deux**, une colonne d'inputs et une colonne d'outputs
- Afficher **2 inputs** :
  - Liste déroulante avec le nom de différents cyclones
  - Liste de choix sur des propriétés du cyclone (vitesse de vent, pression)
- Préparer l'emplacement pour **3 outputs** :
  - Une table comportant les propriétés du cyclone
  - Un graphique sur l'évolution de ces propriétés en fonction du temps
  - Une carte interactive (via leaflet) avec les trajectoires du cyclone sélectionné

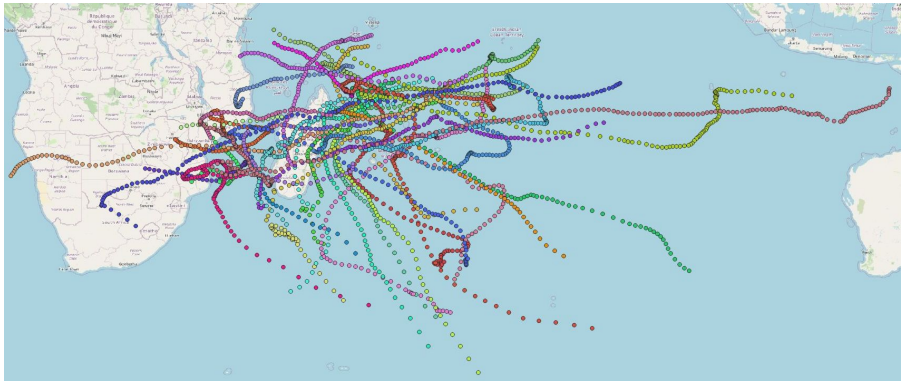
Construction du UI de notre application Shiny

Données externes utilisées dans l'application

name_date_cyclone.txt

name	start	end
FREDDY	2023-03-02	2023-03-14
CHENESO	2023-01-17	2023-01-29
JASMINE	2022-04-24	2022-04-28
GOMBE	2022-03-08	2022-03-14
EMNATI	2022-02-16	2022-02-24
DUMAKO	2022-02-13	2022-02-16
BATSIRAI	2022-01-27	2022-02-08

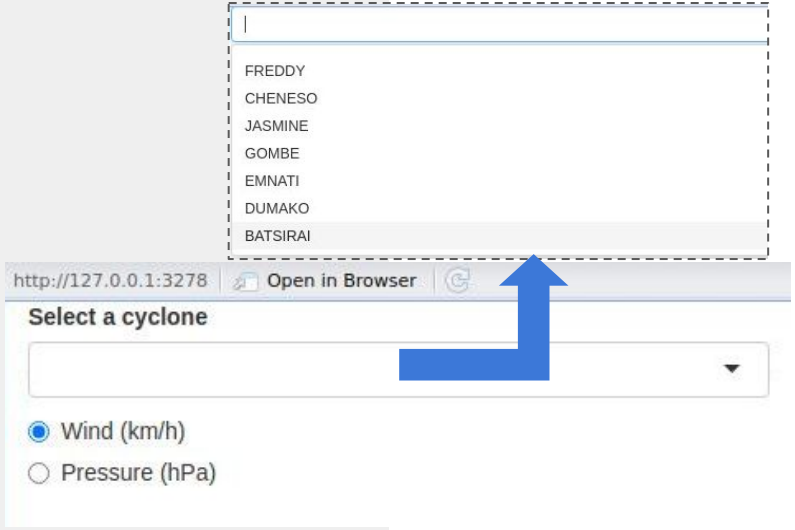
IBTrACS.SI.list.v04r00.points_Mada_2015_present.shp



```
# Import et reclassement de la liste des cyclones en fonction de la date
name_cyclone <- read.csv("./data/name_date_cyclone.txt",sep=',', header = TRUE )
name_cyclone <- name_cyclone[order(as.Date(name_cyclone$start),decreasing = TRUE),]
# Import de la couche .shp des trajectoires/points de mesure des différents cyclones
mada_cycl_p <- st_read("./data/IBTrACS.SI.list.v04r00.points_Mada_2015_present.shp")
```

Construction du UI de notre application Shiny

```
# Import et reclassement de la liste des cyclones en fonction de la date
name_cyclone <- read.csv("./data/name_date_cyclone.txt",sep=',', header = TRUE )
name_cyclone <- name_cyclone[order(as.Date(name_cyclone$start),decreasing = TRUE),]
# Import de la couche .shp des trajectoires/points de mesure des différents cyclones
mada_cycl_p <- st_read("./data/IBTrACS.SI.list.v04r00.points_Mada_2015_present.shp")

ui <- fluidPage(
  # Liste déroulante
  selectInput(inputId = "picker_cyclone",
              label =”Select a cyclone”,
              width = "100%",
              choices = c(" ", name_cyclone$name),
              selected = " ",
              multiple = FALSE),
  # RadioButtons
  radioButtons(inputId = "radio_button", label = NULL,
               choices = list("Wind (km/h)" = "wind_kmh",
                              "Pressure (hPa)" = "USA_PRES"),
               selected = "wind_kmh")
)

server <- function(input, output) {
}

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



Construction du UI de notre application Shiny

```
ui <- fluidPage(
      fluidRow(column(width = 3,
        # Liste déroulante
        selectInput(inputId = "picker_cyclone",
                    label =”Select a cyclone”,
                    width = "100%",
                    choices = c(" ", name_cyclone$name),
                    selected = " ",
                    multiple = FALSE),
        # RadioButtons
        radioButtons(inputId = "radio_button", label = NULL,
                     choices = list("Wind (km/h)" = "wind_kmh",
                                    "Pressure (hPa)" =
                                  "USA_PRES"),
                     selected = "wind_kmh")
                        )
            )
)

server <- function(input, output) {
}

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

- `fluidRow()` permet de diviser la fenêtre de l'application en plusieurs colonnes
- De base, la fenêtre est **divisée en 12 colonnes**
- Ici, nous créons un espace composé de **3 colonnes**

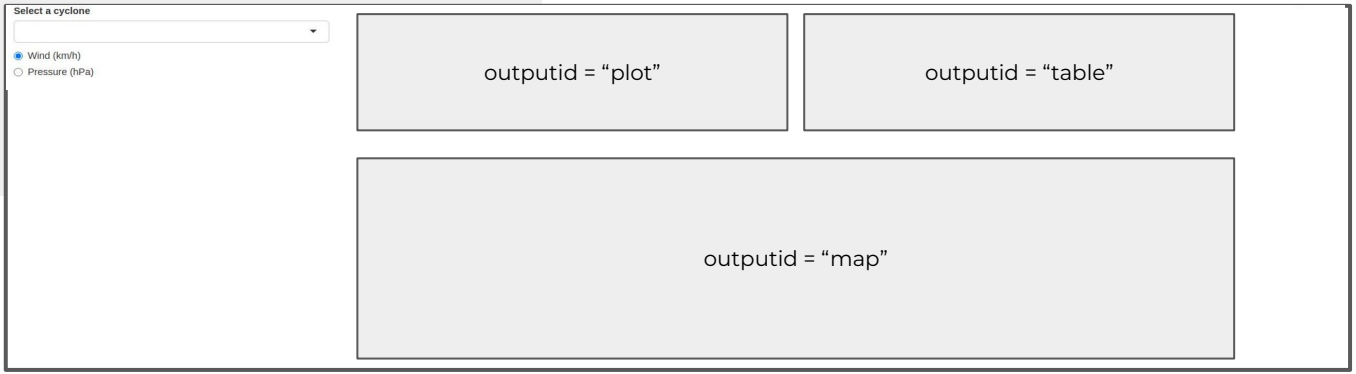
Construction du UI de notre application Shiny

```
ui <- fluidPage(
        fluidRow(column(width = 3,
          # Liste déroulante
          selectInput(inputId = "picker_cyclone",
                      label =”Select a cyclone”,
                         width = "100%",
                      choices = c(" ", name_cyclone$name),
                      selected = " ",
                      multiple = FALSE),
          # RadioButtons
          radioButtons(inputId = "radio_button", label = NULL,
                       choices = list("Wind (km/h)" = "wind_kmh",
                                      "Pressure (hPa)" = "USA_PRES"),
                       selected = "wind_kmh")
                       ),
                  column(width = 4,
                         plotOutput(outputId ="plot")),
                  column(width = 4,
                         DatatableOutput(outputId = "table"))
                )
)

server <- function(input, output) {
}

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

Construction du UI de notre application Shiny

```
ui <- fluidPage(
        fluidRow(column(width = 3,
          # Liste déroulante
          selectInput(inputId = "picker_cyclone",
                      label =”Select a cyclone”,
                         width = "100%",
                      choices = c(" ", name_cyclone$name),
                      selected = " ",
                      multiple = FALSE),
          # RadioButtons
          radioButtons(inputId = "radio_button", label = NULL,
                       choices = list("Wind (km/h)" = "wind_kmh",
                                      "Pressure (hPa)" = "USA_PRES"),
                       selected = "wind_kmh")
                       ),
                  column(width = 4,
                         plotOutput(outputId ="plot")),
                  column(width = 4,
                         tableOutput(outputId = "table"))
                ),
        leafletOutput(outputid= “map”)
)

server <- function(input, output) {
}

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

- `leafletOutput()` permet d’ajouter une carte intéractive **leaflet** dans une application



Server

Fonction `server()`

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

}
```

- `server()` construit un objet de type liste nommé *output* qui contient tout le code nécessaire pour **mettre à jour les objets R** dans votre application
- Chaque objet R doit avoir sa propre entrée dans la liste.
- Vous pouvez créer une entrée en définissant un nouvel élément `output` dans la fonction server(). Le nom de l'élément doit correspondre au nom de l'élément réactif que vous avez créé dans l'interface utilisateur.
- C'est ici qu'entre en jeu les `inputIds` et `outputIds` que vous définissez dans les inputs et les outputs.

Étapes pour créer la fonction `server()`

1. Initialiser notre sortie via le format `output$<outpuid>`

```
server <- function(input, output) {
       output$hist <- # code
}
```

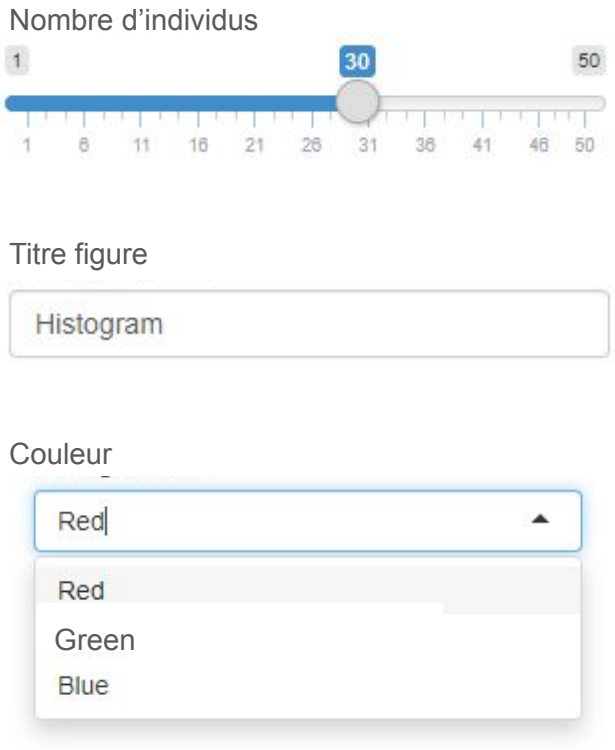
2. Construction de l'objet `output$<outpuid>` avec la famille de fonction `render*()`

```
server <- function(input, output) {
       output$hist <- renderPlot({
                  # code pour créer
                  # un histogramme
                })
}
```

3. Accédez à vos valeurs d'entrée avec `input$<inputId>`, où `<inputId>` est le nom que vous avez fourni dans la fonction `Input()`.

Étapes pour créer la fonction server ( )

```
library(shiny)

ui <- fluidPage(
     # ajout d'un slider
     sliderInput("slider1", "Nombre d'individus",
                 min = 1, max = 50, value = 30)
     # ajout d'une box texte
     textInput("text1", "Titre figure",
                 value = "Histogram"),
     # ajout d'une liste déroulante
     selectInput("select1", "Couleur",
                   choices = c("Red", "Green", "Blue"),
                   selected = "Red")
),

server <- function(input, output) {
}


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



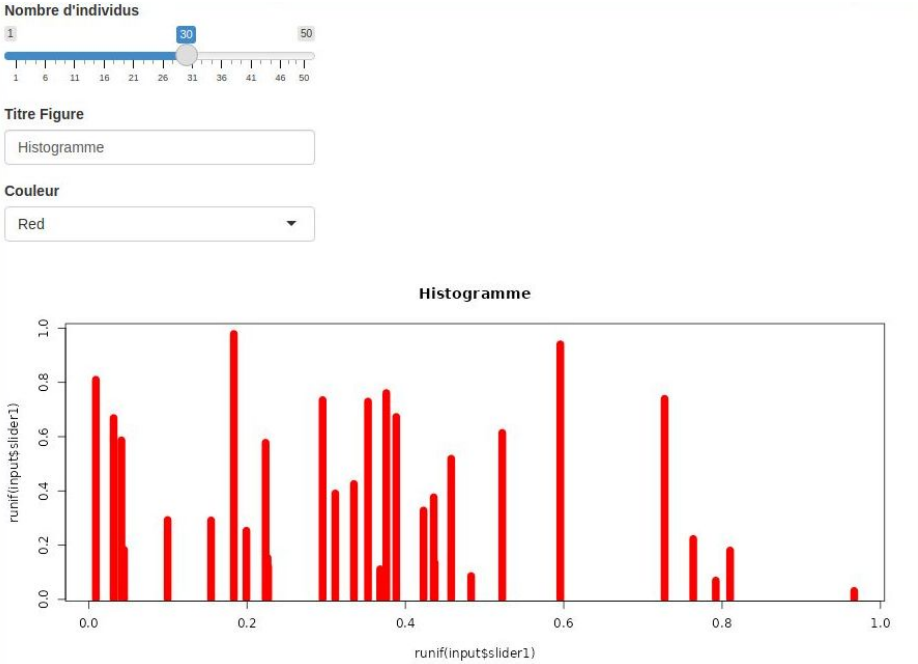
Étapes pour créer la fonction server ( )

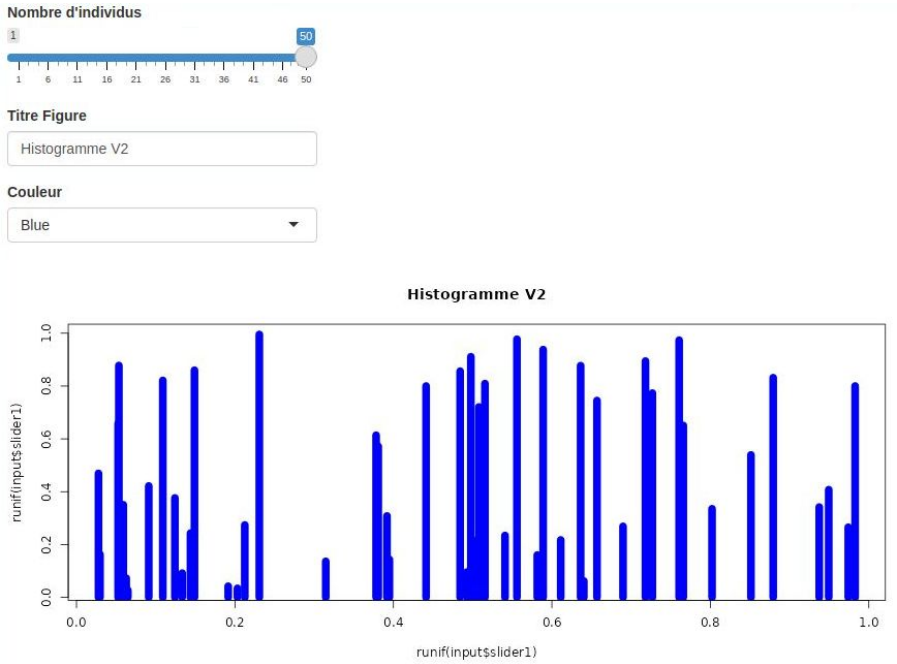
```
library(shiny)

ui <- fluidPage(
       # ajout d'un slider
       sliderInput("slider1", "Nombre d'individus",
              min = 1, max = 50, value = 30)
       # ajout d'une box texte
       textInput("text1", "Titre figure",
              value = "Histogram"),
       # ajout d'une liste déroulante
       selectInput("select1", "Couleur",
                    choices = c("Red", "Green", "Blue"),
                    selected = "Red"),
       plotOutput("plot")
),

server <- function(input, output) {
       output$plot <- renderPlot({
                           plot(x = runif(input$sliderInput),
                                y = runif(input$sliderInput),
                                main = input$text1, col = input$select1,
                                type ="h", lwd = 10)
                    })
}

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

- `runif(n)` : fonction qui génère `n` nombres aléatoires compris en 0 et 1

Étapes pour créer la fonction server ( )

Résultat





server() : Résumé

- La fonction `serveur()` se charge de construire et de reconstruire les objets R qui seront finalement affichés à l'utilisateur
- Enregistrez la sortie que vous construisez dans `output$<outputId>`
- Construisez la sortie avec une fonction `render*()`
- Accédez aux entrées avec `input$<inputId>`
- Plusieurs sorties peuvent être placées dans la fonction serveur
- La réactivité se produit automatiquement lorsque vous utilisez des entrées pour construire des sorties

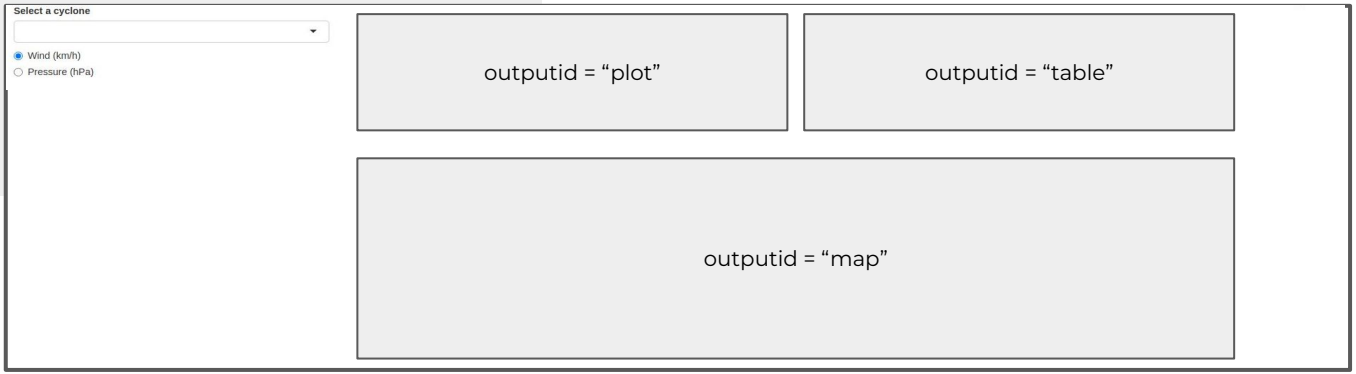
Construction du server() de notre application Shiny

- Application sur l'étude des **cyclones** à Madagascar
- **Scinder l'écran en deux**, une colonne d'inputs et une colonne d'outputs
- Afficher **2 inputs** :
  - Liste déroulante avec le nom de différents cyclones
  - Liste de choix sur des propriétés du cyclone (vitesse de vent, pression)
- Remplir les emplacements des **3 outputs** :
  - Une table comportant les propriétés du cyclone
  - Un graphique sur l'évolution de ces propriétés en fonction du temps
  - Une carte interactive (via leaflet) avec les trajectoires du cyclone sélectionné

Construction du UI de notre application Shiny

```
ui <- fluidPage(
        fluidRow(column(width = 3,
          # Liste déroulante
          selectInput(inputId = "picker_cyclone",
                      label =”Select a cyclone”,
                         width = "100%",
                      choices = c(" ", name_cyclone$name),
                      selected = " ",
                      multiple = FALSE),
          # RadioButtons
          radioButtons(inputId = "radio_button", label = NULL,
                       choices = list("Wind (km/h)" = "wind_kmh",
                                      "Pressure (hPa)" = "USA_PRES"),
                       selected = "wind_kmh")
                       ),
                  column(width = 4,
                         plotlyOutput(outputId ="plot")),
                  column(width = 4,
                         tableOutput(outputId = "table"))
                ),
        leafletOutput(outputid = “map”)
)

server <- function(input, output) {
}

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

- **`plotlyOutput(n)`** : fonction qui permet d’introduire un graphique intéractive et non statique comme peut l’être **`plotOutput`**



Construction du server() de notre application Shiny

Un graphique sur l'évolution de ces propriétés en fonction du temps

```
server <- function(input, output) {
     output$plot <- renderPlotly({
                  })
}
```

Initialisation de notre borne `output$<outputid>` à l'aide de `renderPlotly()` ⇒ adapté aux graphiques **plotly**

```
server <- function(input, output) {
     output$plot <- renderPlotly({
                select <- mada_cycl_p[mada_cycl_p$NAME == input$picker_cyclone,]
                df <- data.frame(select)
                df$time <- as.POSIXct(df$ISO_TIME)
                  })
}
```

- Sélection des **points de mesure** pour le **cyclone sélectionné** à travers le menu déroulant (ex : Freddy…)
- **Transformation en tableau** des différentes mesures
- **Création d'un nouveau champs** en format temps

```
server <- function(input, output) {
     output$plot <- renderPlotly({
                select <- mada_cycl_p[mada_cycl_p$NAME == input$picker_cyclone,]
                df <- data.frame(select)
                df$time <- as.POSIXct(df$ISO_TIME)

        if (input$radio_button== "wind_kmh"){
                yl <- "Wind Speed (km/h)"
                color_plot <- "red"
        }else{
                yl <- "Pressure (hPa)"
                color_plot <- "blue"
        }
 })
}
```

Condition en fonction de la valeur d' `input$radiobutton`

- **Titre axe ordonnées** (*yl*)
- **Couleur du tracé** *(color_plot)*

Construction du server() de notre application Shiny

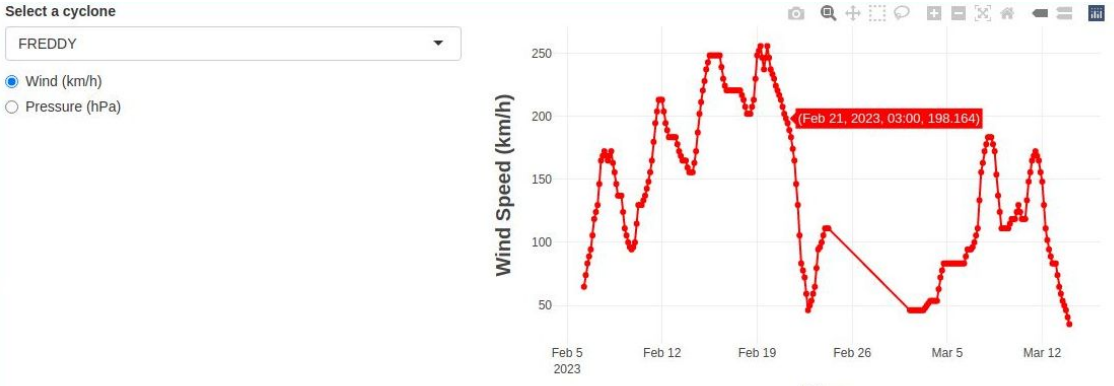
Un graphique sur l'évolution de ces propriétés en fonction du temps

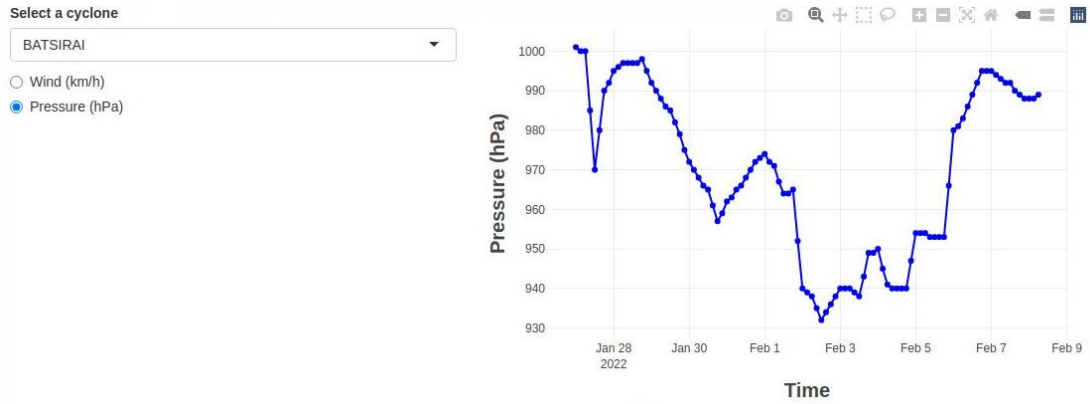
```
server <- function(input, output) {

     output$plot <- renderPlotly({
         select <- mada_cycl_p[mada_cycl_p$NAME == input$picker_cyclone,]
         df <- data.frame(select)
         df$time <- as.POSIXct(df$ISO_TIME)

         if (input$radio_button== "wind_kmh"){
                  yl <- "Wind Speed (km/h)"
                  color_plot <- "red"
         }else{
                  yl <- "Pressure (hPa)"
                  color_plot <- "blue"
         }

         plot_ly()%>% add_trace(data = df ,
                                x = df$time,
                                y = df[,input$radio_button],
                                line = list(color = colorplot),
                                marker = list(color = colorplot),
                                type = 'scatter', mode = 'lines+markers')%>%
                      layout(title = " " ,
                             xaxis = list(title = list(text="<b>Time</b>",font=list(size=20))),
                             yaxis = list(title=list(text=paste0("<b>",yl,"</b>"),
                             font=list(size=20))),
                             showlegend = FALSE)

 })
}
```

Ajout du graphique interactif **plolty**

Construction du server() de notre application Shiny

Un graphique sur l'évolution de ces propriétés en fonction du temps : Résultats





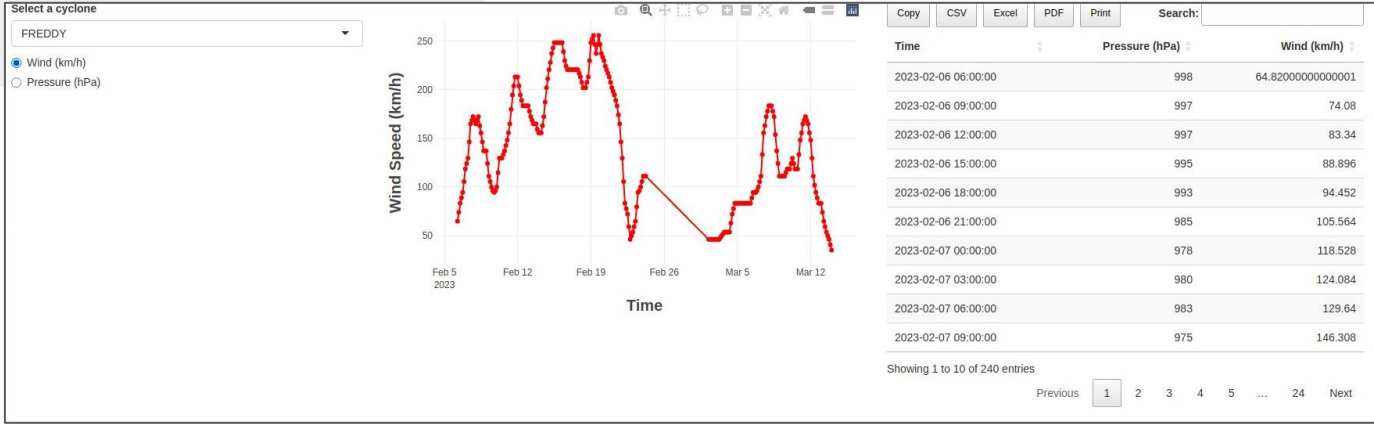
Construction du server() de notre application Shiny

Ajout d'une table comportant les propriétés du cyclone

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

     output$plot <- renderPlotly({>>
 >>})

     output$table <- renderDataTable({
                 select <- mada_cycl_p[mada_cycl_p$NAME == input$picker_cyclone,]
                 df <- data.frame(select)
                 df$time <- as.POSIXct(df$ISO_TIME)

         datatable(df[,c('ISO_TIME','USA_PRES',"wind_kmh")],
                 colnames = c('Time' = 1, 'Pressure (hPa)' = 2,"Wind (km/h)"=3),
                 rownames=FALSE,
                 extensions = "Buttons",
                 options = list(
                 dom = 'Bfrtip',
                 buttons = c('copy', 'csv', 'excel', 'pdf', 'print')))
         })


}
```

`renderDataTable()` est un fonction du package `DT` permettant :

- un affichage simplifié des tableaux
- de donner l'option à l'utilisateur d'exporter le tableau de données



Construction du server() de notre application Shiny

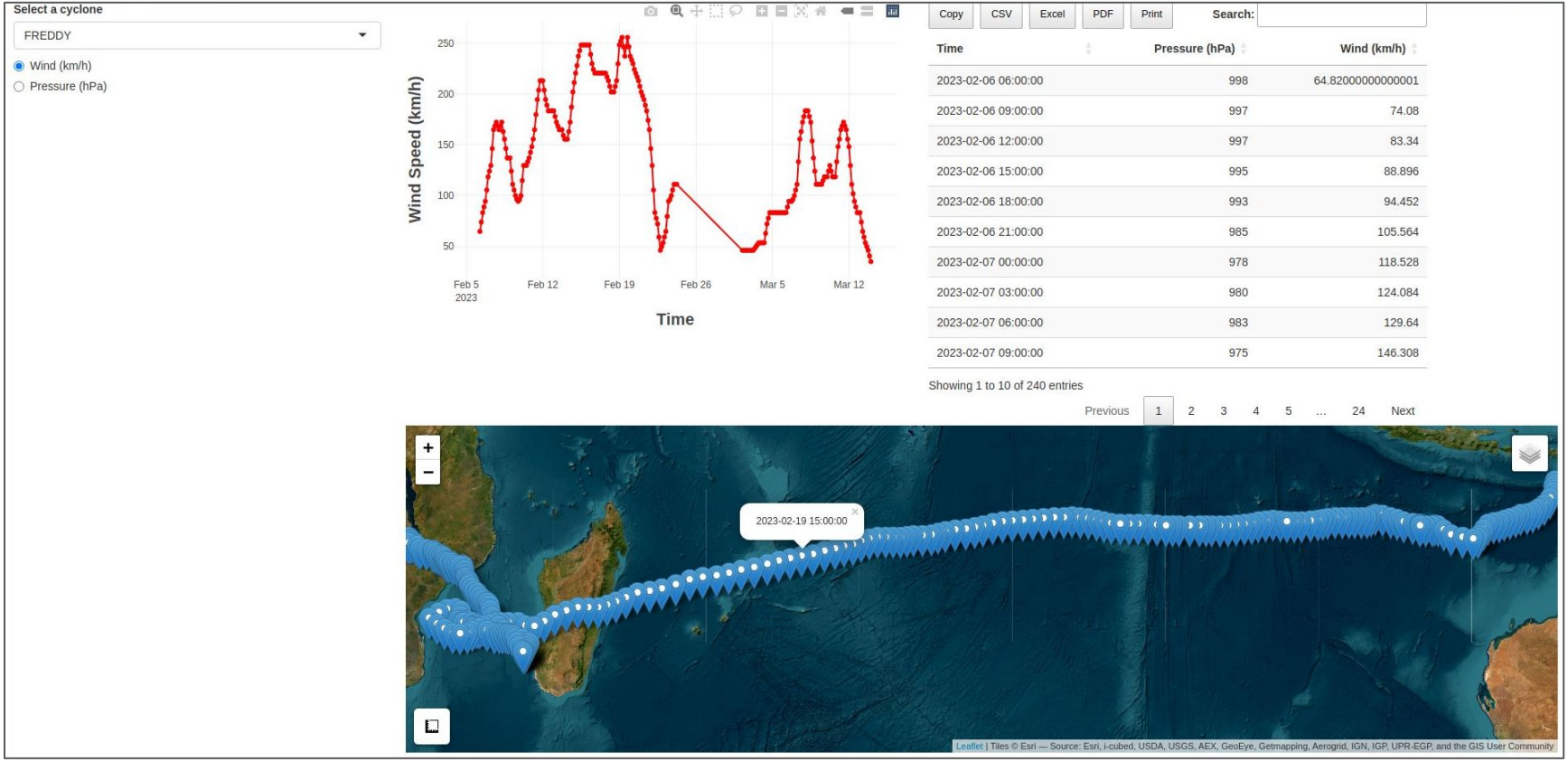
Ajout d'une table comportant les propriétés du cyclone

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

     output$plot <- renderPlotly({>>
 >>})

     output$table <- renderDataTable({>>
 >>})

 output$map <- renderLeaflet({

         select <- mada_cycl_p[mada_cycl_p$NAME == input$picker_cyclone,]

         leaflet(options = leafletOptions(wheelPxPerZoomLevel=150, zoomSnap= 0.05) )%>%

                 addMarkers(data=select, group = "Trajectoire cyclone",layerId = paste(substr(select$ISO_TIME,1,10),substr(select$ISO_TIME, 12,19)),
                            popup= select$ISO_TIME,popupOptions = popupOptions(style = list("font-weight" = "normal",padding = "3px 50px"),direction = "auto",
                            maxWidth = 500))%>%

                 addProviderTiles(providers$Esri.WorldImagery,group = "Satellite")%>%
                  addProviderTiles(providers$OpenStreetMap.Mapnik,group = "OSM")%>%

                 addLayersControl(baseGroups = c("Satellite","OSM"),
                                  overlayGroups = c("Trajectoire cyclone"),
                                  options = layersControlOptions(collapsed = TRUE))%>%

                 addMeasure(position = "bottomleft",
                            primaryLengthUnit = "meters",
                            primaryAreaUnit = "sqmeters")
   })

}
```

`renderleaflet()` permet d'importer et d'afficher une carte *leaflet* dans l'interface

Construction du server() de notre application Shiny

Ajout d'une table comportant les propriétés du cyclone : Résultat

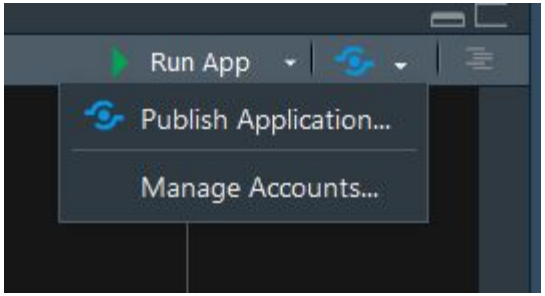


Partager son application RShiny

Plusieurs possibilités de partage

ShinyApps.io

1. Créez un compte gratuit sur https://www.shinyapps.io/
2. Créez votre application Shiny.
3. Publiez votre application.



Gratuit :

- 5 applications actives
- 25 heures d'utilisation active par mois

GitHub

docker

https://shiny.posit.co/r/getstarted/shiny-basics/lesson7/

Exemple d’application avancée

https://shiny.posit.co/r/gallery/life-sciences/covid19-tracker/