Utilisation de la bibliothèque Shiny & Esquisse pour la création d'applications web

Mame Thierno Ndiaye

2023-04-17

Présentation de R Shiny

Shiny est un package utilisé pour créer des applications Web interactive avec R. L'application Web peut être intégrée dans un document rmarkdown, une page Web, de manière autonome ou en tant que tableau de bord.

C'est un outil puissant avec lequel nous pouvons créer des applications et des tableaux de bord assez rapidement.

Dans cet exposé, Nous allons parcourir les bases de Shiny, comment créer des applications simples et présenter le concept de réactivité.

Comment installer le package R Shiny ?

```
install.packages("Shiny") #Installer le package Shiny
```

Exemples d'application Shiny

L'emballage Shiny comporte onze exemples intégrés qui démontrent chacun comment fonctionne Shiny. Chaque exemple est une application Shiny autonome. L'exemple **Hello Shiny** trace un histogramme du jeu de données de R avec un nombre configurable de bacs. Les utilisateurs peuvent modifier le nombre de bacs avec une barre de défilement, et l'application répondra immédiatement à leur entrée. Nous utiliserons **Hello Shiny** pour explorer la structure d'une application Shiny et créer notre première application.

Pour exécuter **Hello Shiny**, tapez :

```
# Exemple application Shiny
library(shiny)
shinyAppDir(system.file("examples/01_hello", package = "shiny"))
```

Structure d'une application Shiny

Les applications Shiny sont contenues dans un script unique appelé **app.R** . Le script **app.R** réside dans un répertoire (par exemple, **Exposé_Shiny**) et l'application peut être exécutée avec **runApp("Exposé_Shiny")**.

app.R comporte trois volets :

- un objet d'interface utilisateur (**ui**)
- une fonction serveur (**server**)
- un appel à la fonction **shinyApp**

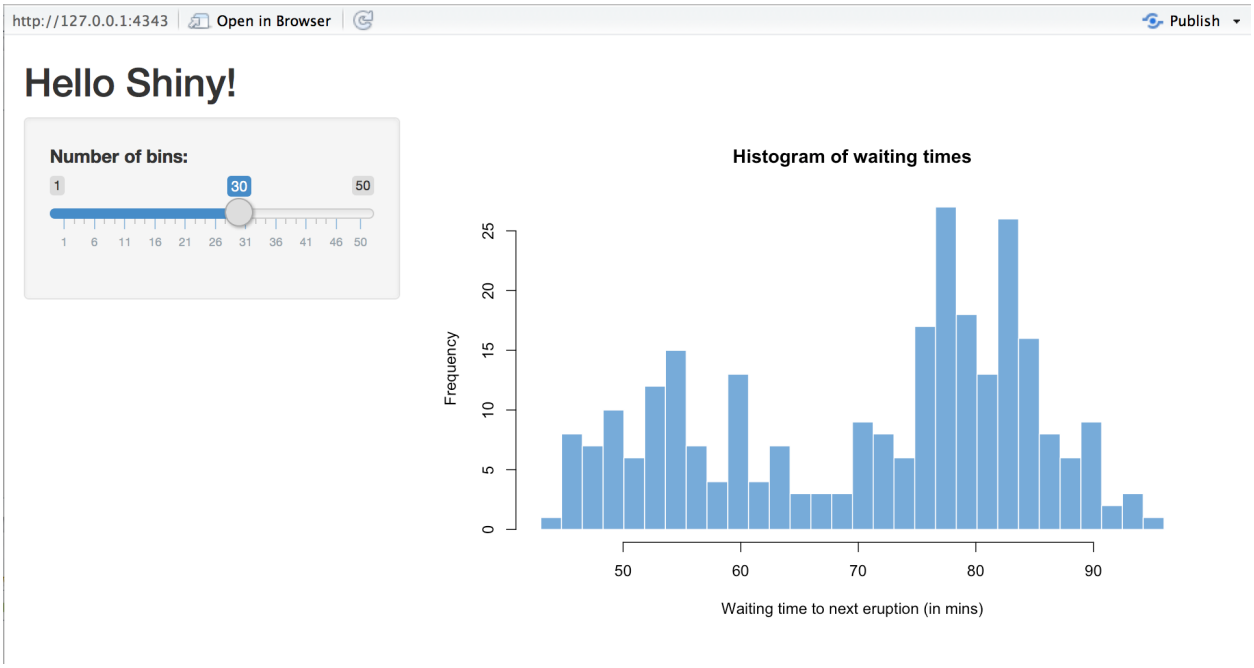

Figure 1: Hello Shiny.

L'objet **ui** contrôle la disposition et l'apparence de l'application. La fonction **server** contient les instructions dont l'ordinateur a besoin pour générer notre application. Enfin, la fonction **shinyApp** crée des objets d'application Shiny à partir d'une paire **UI/serveur** explicite.

Une fonctionnalité intéressante des applications à fichier unique est que vous pouvez copier et coller l'application entière dans la console R, ce qui facilite le partage rapide de code pour que d'autres puissent l'expérimenter. Par exemple, si vous copiez et collez le code ci-dessus dans la ligne de commande R, il démarrera une application Shiny.

UI

Voici l'objet **ui** de l'exemple **Hello Shiny**.

```
library(shiny)

# Définir l'interface utilisateur d'une application qui dessine un histogramme ----
ui <- fluidPage(

  # Titre de l'appli ----
  titlePanel("Hello Shiny!"),

  # Disposition de la barre latérale avec définitions d'entrée et de sortie ----
  sidebarLayout(

    # Panneau de la barre latérale pour les entrées ----
    sidebarPanel(

      # Entrée : Curseur pour le nombre de bacs ----
      sliderInput(inputId = "bins",
                  label = "Nombre de bacs:",
                  min = 1,
```

```
                  max = 50,
                  value = 30)

    ),

    # Panneau principal pour l'affichage des sorties ----
    mainPanel(

      # Sorties: Histogramme ----
      plotOutput(outputId = "distPlot")

    )
  )
)
```

serveur

Voici la fonction server pour l'exemple **Hello Shiny**.

```
# Définir la logique serveur requise pour dessiner un histogramme ----
server <- function(input, output) {

  # Histogramme des données du geyser Old Faithful ----
  # avec le nombre de bacs demandé
  # Cette expression qui génère un histogramme est encapsulée dans un appel
  # pour renderPlot pour indiquer que :
  # 1. Il est «réactif» et devrait donc être automatiquement
  # réexécuté lorsque les entrées (input$bins) changent
  # 2. Son type de sortie est un tracé
  output$distPlot <- renderPlot({

    x    <- faithful$waiting
    bins <- seq(min(x), max(x), length.out = input$bins + 1)

    hist(x, breaks = bins, col = "#007bc2", border = "white",
         xlab = "Waiting time to next eruption (in mins)",
         main = "Histogram of waiting times")

    })

}
```

shinyApp

Voici la fonction **shinyApp** pour l'exemple **Hello Shiny**

```
 # Voir ci-dessus pour les définitions de ui et server
shinyApp(ui = ui, server = server)
```

Exécution d'une application

Chaque application Shiny a la même structure : un fichier **app.R** qui contient **ui** et **server** . On peut créer une application Shiny en créant un nouveau répertoire et en enregistrant un fichier **app.R** à l'intérieur. Il est recommandé que chaque application vive dans son propre répertoire unique.

On peut exécuter une application Shiny en donnant le nom de son répertoire à la fonction **runApp**.

Remarque : **runApp** est similaire à **read.csv**, **read.table** , et à de nombreuses autres fonctions de R. Le premier argument de **runApp** est le chemin d'accès de votre répertoire de travail au répertoire de l'application.

Pqr exemple, si l'application Shiny se trouve dans un répertoire appelé **Exposé_Shiny**, exécutez-la avec le code suivant :

```
library(shiny)
runApp("Exposé_Shiny")    # Executer une application Shiny
```

Le code ci-dessus suppose que le répertoire de l'application se trouve dans votre répertoire de travail. Dans ce cas, le chemin d'accès au fichier est simplement le nom du répertoire.

Création d'une interface d'utilisateur (UI)

Ici nous allons voir comment créer une interface utilisateur pour notre application. Nous verrons comment ajouter du texte, des images et d'autres éléments HTML à Notre application Shiny.

Ce code est le strict minimum nécessaire pour créer une application Shiny. Le résultat est une application vide avec une interface utilisateur vide.

```
library(shiny)

# definir UI ----
ui <- fluidPage(

)

# definir server logic ----
server <- function(input, output) {

}

# executer l'application  ----
shinyApp(ui = ui, server = server)
```

Disposition

Shiny utilise la fonction **fluidPage** pour créer un affichage qui s'ajuste automatiquement aux dimensions de la fenêtre du navigateur de votre utilisateur. Nous disposons de l'interface utilisateur de notre application en plaçant des éléments dans la fonction **fluidPage**.

titlePanel et **sidebarLayout** sont les deux éléments les plus populaires à ajouter à **fluidPage**. Ils créent une application Shiny de base avec une barre latérale.

sidebarLayout prend toujours deux arguments :

-**sidebarPanel:** Sortie de fonction

-**mainPanel:** Sortie de fonction

Ces fonctions placent le contenu dans la barre latérale ou dans les panneaux principaux.

On peut également créer des mises en page plus avancées. On peut utiliser **navbarPage** pour doter l'application d'une interface utilisateur de plusieurs pages comprenant une barre de navigation. Ou on peut utiliser **fluidRow** et construire une mise en page **column** à partir d'un système de grille.Pour en savoir plus sur

ces options avancées, vous pouvez utiliser le lien ci-dessous: **https://shiny.rstudio.com/articles/layout-guide.html**

Par exemple, la fonction ci-dessous crée une interface utilisateur **fluidPage** dotée d'un panneau de titre et d'une disposition de barre latérale, qui comprend un panneau de barre latérale et un panneau principal. Tous ces éléments sont placés dans la fonction **ui**.

```
ui <- fluidPage(
  #Titre page
  titlePanel("Exposé_Shiny"),
   # Disposition latérale
  sidebarLayout(
    sidebarPanel("sidebar panel"),
    #panneeau principale
    mainPanel("main panel")
  )
)
```

Le panneau de la barre latérale **sidebarLayout** apparaîtra par défaut sur le côté gauche de l'application. On peut le déplacer vers la droite en donnant l'argument facultatif **position = "right"**.

```
ui <- fluidPage(
  titlePanel("Exposé_Shiny"),

  sidebarLayout(position = "right",
                sidebarPanel("sidebar panel"),
                mainPanel("main panel")
  )
)
```

Contenu HTML

On peut ajouter du contenu à l'application Shiny en la plaçant dans une fonction **Panel**. Par exemple, les applications ci-dessus affichent une chaîne de caractères dans chacun de leurs panneaux.

```
ui <- fluidPage(
  titlePanel("Exposé_Shiny"),

  sidebarLayout(
                sidebarPanel("sidebar panel"),
                mainPanel(
                  h1("premier niveau "),
                  h2("deuxième niveau "),
                  h3("Troisième niveau"),
                  h4("Quatrième niveau "),
                  h5("Cinquième niveau "),
                  h6("Sixième niveau ")
                )
  )
)
```

Il est possible aussi de centre les textes avec la syntaxe **align="center"**.

Texte mis en forme

Shiny offre de nombreuses fonctions de balise pour la mise en forme du texte. On peut utiliser par exemple les fonction précedentes pour faire des mises en forme.

Exemple pour mise en forme:

```
ui <- fluidPage(
  titlePanel("Exposé_shiny"),
  sidebarLayout(
    sidebarPanel(),
    mainPanel(
      p("Mon nom est Mame Thierno Ndiaye, je vais vous faire une brève présentation de ma personne."),
      p("Je suis élève à", strong("l'Ecole Nationale de la Statistique et de l'Analyse Economique."), "
      em("En dehors de la Statistique, je suis aussi très passionné par l'astronomie."),
      br(),
      div("Un plus d'informations:", style = "color:blue"),
      br(),
      p("j'ai fait mon cycle primaire à l'école",span("Al Azhar de Guédiawaye,", style = "color:blue"),
      "Pour mon bac, je l'ai obtenu au ", span("Lycée Seydina Limoulaye de Guédiawaye.", style = "color
    ),
  )
)
)
```

Images

Pour insérer une image, on donne à la fonction **img** le nom de notre fichier image comme argument.

Important: L'image doit d'abord être dans le dossier **WWW** de shiny.

```
ui <- fluidPage(
  titlePanel("Exposé_shiny"),
  sidebarLayout(
    sidebarPanel(
      p(strong("Mame Thierno Ndiaye")),
      img(src="Thier.jpg") # Insertion d'une image
    ),
    mainPanel(
      p("Mon nom est Mame Thierno Ndiaye, je vais vous faire une brève présentation de ma personne."),
      p("Je suis élève à", strong("l'Ecole Nationale de la Statistique et de l'Analyse Economique."), "
      em("En dehors de la Statistique, je suis aussi très passionné par l'astronomie."),
      br(),
      div("Un plus d'informations:", style = "color:blue"),
      br(),
      p("j'ai fait mon cycle primaire à l'école",span("Al Azhar de Guédiawaye,", style = "color:blue"),
      "Pour mon bac, je l'ai obtenu au ", span("Lycée Seydina Limoulaye de Guédiawaye.", style = "color
    ),
  )
)
)
```

Ajouter des widgets de contrôle

Qu'est-ce qu'un widget ?: Élément Web avec lequel les utilisateurs peuvent interagir. Les widgets permettent aux utilisateurs d'envoyer des messages à l'application Shiny.

Les widgets Shiny collectent une valeur auprès de l'utilisateur. Lorsqu'un utilisateur modifie le widget, la valeur change également. Cela crée des opportunités que nous explorerons dans la suite.

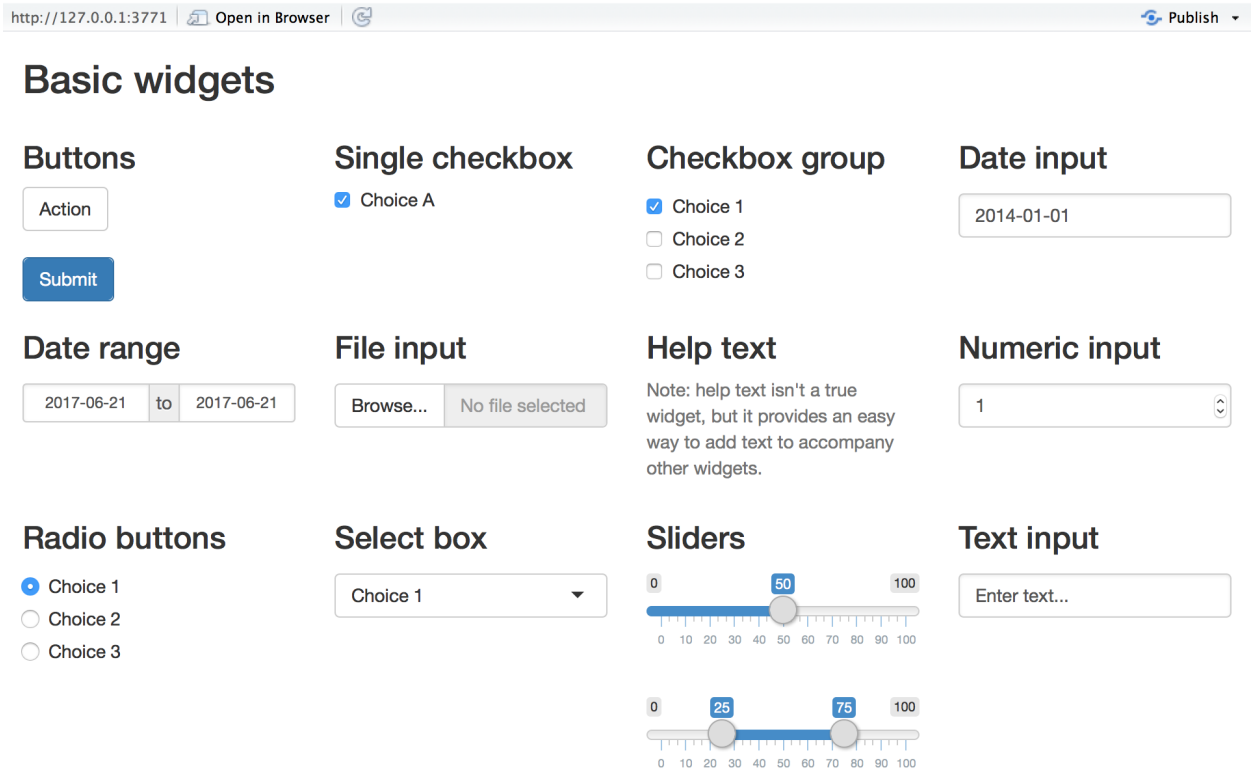

Figure 2: widgets.

Widgets de contrôle

Les widgets Shiny standard sont :

fonction	**widget**
actionButton	Bouton d'action
checkboxGroupInput	Un groupe de cases à cocher
checkboxInput	Une seule case à cocher
dateInput	Un calendrier pour faciliter la sélection de la date
dateRangeInput	Une paire de calendriers pour sélectionner une plage de dates
fileInput	Un assistant de contrôle de téléchargement de fichiers
helpText	Texte d'aide pouvant être ajouté à un formulaire de saisie
numericInput	Un champ pour entrer des nombres
radioButtons	Un ensemble de boutons radio
selectInput	Une boîte avec des choix
sliderInput	Une barre de défilement
submitButton	Un bouton d'envoi
textInput	Un champ pour saisir du texte

Ajout de widgets

Nous pouvons ajouter des widgets à notre page Web de la même manière qu'on a ajouté d'autres types de contenu HTML précédement. Pour ajouter un widget à notre application, plaçons une fonction de widget dans **sidebarPanel** ou dans **mainPanel** votre objet **ui**.

Chaque fonction de widget nécessite plusieurs arguments. Les deux premiers arguments de chaque widget sont les suivants :

- **un nom pour le widget** : l'utilisateur ne verra pas ce nom, mais vous pouvez l'utiliser pour accéder à la valeur du **widget**. Le nom doit être une chaîne de caractères.
- **une étiquette** : cette **étiquette** apparaîtra avec le widget dans notre application. Il doit s'agir d'une chaîne de caractères, mais il peut s'agir d'une chaîne vide." "

Les arguments restants varient d'un widget à l'autre. Ils incluent des éléments tels que les **valeurs initiales**, les **plages** et les **incréments**.

```
ui <- fluidPage(
  titlePanel("Exemples de widgets"),

  fluidRow(

    column(3,
           h3("Buttons"),
           actionButton("action", "Action"),
           br(),
           br(),
           submitButton("Submit")),

    column(3,
           h3("Single checkbox"),
           checkboxInput("checkbox", "Choice A", value = TRUE)),

    column(3,
           checkboxGroupInput("checkGroup",
                              h3("Checkbox group"),
                              choices = list("Choice 1" = 1,
                                             "Choice 2" = 2,
                                             "Choice 3" = 3),
                              selected = 1)),

    column(3,
           dateInput("date",
                     h3("Date input"),
                     value = "2014-01-01"))
  ),

  fluidRow(

    column(3,
           dateRangeInput("dates", h3("Date range"))),

    column(3,
           fileInput("file", h3("File input"))),

    column(3,
           h3("Help text"),
           helpText("Remarque : le texte d'aide n'est pas un véritable widget,",
                    "mais il fournit un moyen facile d'ajouter du texte à",
                    "accompagner d'autres widgets."),
    ),
```

```
    column(3,
           numericInput("num",
                        h3("Numeric input"),
                        value = 1))
  ),

  fluidRow(

    column(3,
           radioButtons("radio", h3("Radio buttons"),
                        choices = list("Choice 1" = 1, "Choice 2" = 2,
                                       "Choice 3" = 3),selected = 1)),

    column(3,
           selectInput("select", h3("Select box"),
                       choices = list("Choice 1" = 1, "Choice 2" = 2,
                                      "Choice 3" = 3), selected = 1)),

    column(3,
           sliderInput("slider1", h3("Sliders"),
                       min = 0, max = 100, value = 50),
           sliderInput("slider2", "",
                       min = 0, max = 100, value = c(25, 75))
    ),

    column(3,
           textInput("text", h3("Text input"),
                     value = "Enter text..."))
  )

)

server <- function(input, output) {

}

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

Affichage de la sortie réactive

Il est temps de donner à notre application Shiny une qualité « live »! On verra comment créer une sortie réactive aux écrans de notre application Shiny.

Deux étapes

On peut créer une sortie réactive avec un processus en deux étapes.

1.Ajouter un objet R à l'interface utilisateur.

2.Indiquez à Shiny comment construire l'objet dans la fonction **serveur**. L'objet sera réactif si le code qui le génère appelle une valeur de widget.

Étape 1 : Ajouter un objet R à l'interface utilisateur

Shiny fournit une famille de fonctions qui transforment les objets R en sortie pour votre interface utilisateur. Chaque fonction crée un type de sortie spécifique.

Fonction de sortie	Crée
dataTableOutput	DataTable
htmlOutput	HTML brut
imageOutput	image
plotOutput	complot
tableOutput	table
textOutput	SMS
uiOutput	HTML brut
verbatimTextOutput	SMS

Exemple: l'objet **ui** ci-dessous permet d'ajouter une ligne de texte **textOutput** réactive au panneau principal de l'application Shiny illustrée ci-dessus.

```
ui <- fluidPage(
  titlePanel("censusVis"),

  sidebarLayout(
    sidebarPanel(
      helpText("Create demographic maps with
               information from the 2010 US Census."),

      selectInput("var",
                  label = "Choose a variable to display",
                  choices = c("Percent White",
                              "Percent Black",
                              "Percent Hispanic",
                              "Percent Asian"),
                  selected = "Percent White"),

      sliderInput("range",
                  label = "Range of interest:",
                  min = 0, max = 100, value = c(0, 100))
    ),

    mainPanel(
      textOutput("selected_var")
    )
  )
)
```

Notons que **textOutput** prend un argument, la chaîne de caractères **"selected_var"** . Chacune des fonctions **Output** nécessite un seul argument : une chaîne de caractères que Shiny utilisera comme nom de l'élément réactif. l'utilisateur ne vera pas ce nom, mais on l'utilisera ultérieurement.

Étape 2 : Fournir le code R pour générer l'objet

Placer une fonction dans **ui** indique à Shiny où afficher notre objet. Ensuite, on doit dire à Shiny comment construire l'objet.

Pour ce faire, on fournit le code R qui construit l'objet dans la fonction **server**.

La fonction **server** joue un rôle particulier dans le processus Shiny; il génère un objet de type liste nommé **output** qui contient tout le code nécessaire pour mettre à jour les objets R dans otre application. Chaque objet R doit avoir sa propre entrée dans la liste.

On peut créer une entrée en définissant un nouvel élément **output** pour dans la fonction **server**, comme ci-dessous. Le nom de l'élément doit correspondre au nom de l'élément réactif que vous avez créé dans le fichier **ui**.

Dans la fonction **server** ci-dessous, **output$selected_var** correspond à votre fichier **ui textOutput("selected_var")**.

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

  output$selected_var <- renderText({
    "Vous avez sélectionné ceci"
  })
}
```

Chaque entrée à doit contenir la sortie d'une des fonctions *output* de Shiny. Ces fonctions capturent une expression R et effectuent un léger prétraitement sur l'expression *render*. Utilisez la fonction *render* qui correspond au type d'objet réactif que vous créez.

Fonction de rendu	Crée
renderDataTable	DataTable
renderImage	images (enregistrées en tant que lien vers un fichier source)
renderPlot	Emplacements
renderPrint	toute sortie imprimée
renderTable	Trame de données, matrice, autres structures de type table
renderText	chaînes de caractères

Chaque fonction **render** prend un seul argument : une expression R entourée d'accolades, "{}" . L'expression peut être une simple ligne de texte, ou elle peut impliquer plusieurs lignes de code, comme s'il s'agissait d'un appel de fonction compliqué.

Utiliser des valeurs de widget Si nous exécutons l'application avec la fonction **server** ci-dessus, elle affichera « Vous avez sélectionné ceci » dans le panneau principal. Cependant, le texte ne sera pas réactif. Cela ne changera pas même si nous manipulons les widgets de votre application.

On peut rendre le texte réactif en demandant à Shiny d'appeler une valeur de widget lorsqu'il crée le texte. Voyons comment procéder avec l'exemple suivant:

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

  output$selected_var <- renderText({
    paste("You have selected", input$var) #On appelle le widget avec son nom en utilisant input
  })

}
```

C'est ainsi que nous créons de la réactivité avec Shiny, en reliant les valeurs de **input** aux objets dans **output** . Shiny s'occupe de tous les autres détails.

Création d'une application

On va créer une application qui permettra entre autres d'importer une base de données sous format csv puis de donner pour chaque variables les caractéristiques et de de choisir une variable quantitative pour

representer son histogramme.

```
library(shiny)
library(colourpicker)
shinyApp(
  ui= fluidPage(
  titlePanel("Mon application"),
  sidebarLayout(
    sidebarPanel(
      fileInput("file1", "Choisir un fichier CSV",
            accept = c("text/csv", "text/comma-separated-values,text/plain",
                       ".csv")),
       sliderInput("bins",
                   "Number of bins:",
                   min = 1,
                   max = 50,
                   value = 30),
       #colourInput(inputId = "color", label = "Color:", value = "purple"),
      selectInput(inputId = "color", label = "Couleur :",
                  choices = c("Red" = "red", "Green" = "green", "Blue" = "blue")),
       textInput(inputId = "titre", label = "Title:", value = "Histogram"),
          radioButtons(
        "var",
        "Choisir la variable:",
        choices = names(data())
      ),
       downloadLink('download_plot', 'Download the graph')
    ),
    mainPanel(
       plotOutput("distPlot"),
       div(textOutput("n_bins"), align = "center"),
       tags$hr(),
       verbatimTextOutput("summary"),
       dataTableOutput("table"),
    )
  )
),
server=function(input, output) {
  data <- reactive({
    infile <- input$file1
    if (is.null(infile)) {
      return(NULL)
    }
    read.csv2(infile$datapath)
  })
   observe({
    updateRadioButtons(
      session = shiny::getDefaultReactiveDomain(),
      inputId = "var",
      choices = colnames(data())
    )
  })

  output$distPlot <- renderPlot({
```

```
    x    <- data()[, input$var]
    bins <- seq(min(x), max(x), length.out = input$bins + 1)
    hist(x, breaks = bins, col = input$color, border = 'white', main = input$titre)

  })

  output$summary <- renderPrint({
    summary(data())
  })
  output$table <- renderDataTable({
    data()
  })

  output$n_bins <- renderText({
    paste("Number of bins: ", input$bins)
  })
  output$download_plot <- downloadHandler(
    filename = function() {
      paste('plot-', Sys.Date(), '.jpeg', sep='')
    },
    content = function(con) {
      jpeg(file = con)
      x    <- data()[, input$var]
      bins <- seq(min(x), max(x), length.out = input$bins + 1)
      hist(x, breaks = bins, col = input$color, border = 'white', main = input$titre)
      dev.off()
    }
  )
}
)
```

**Exemple des tableaux de bord

```
library(shiny)
library(shinydashboard)
header<- dashboardHeader(
  title = "KAY-DIANG",# titre pour l'en tête'
  titleWidth = 300,#Taille
  dropdownMenu(type = "messages", badgeStatus = "success",
               messageItem("Message de Mame Thierno",
                           "Je voudrais de l'aide sur un exo .",
                           time = "1h"
               ),
               messageItem("Moniseur Camara",
                           "Bonjour, demain je serais un peu occupé. On va reprogrammer le cours pour la
                           time = "30 min"
               ),
               messageItem("New User",
                           "puis je avoir de l'aide, s'il vous plaît?",
                           time = "Today"
               )
  ),
```

```
  dropdownMenu(type = "notifications", badgeStatus = "warning",
               notificationItem("Vous pouvez voir votre progréssion pendant ce mois.", icon = icon("excl

               ),
               notificationItem("Vous avez un nouveau cours en PC",icon = icon("exclamation"), status =

               ),
               notificationItem("Vous avez un nouveau exercice en PC",icon = icon("exclamation"), statu
               )
  ),
  dropdownMenu(type = "tasks", badgeStatus = "danger",
               taskItem("Mise à jour de votre profil", value = 70, color = "aqua",
               ),
               taskItem("evaluation", value = 80, color = "green",
               ),
               taskItem("Mise à jour de vos notes", value = 60, color = "yellow",
               ),
               taskItem("Write documentation", value = 80, color = "red",
               )
  )
)
shinyApp(
  ui = dashboardPage(
    #l'en tête
    header,
    #la partie gauche
    dashboardSidebar(
      sidebarMenu(
      #Définir ce qui sera dans la partie gauche
      menuItem("Connexion", tabName = "Conex", icon = icon("users")),
      menuItem("ACCEUIL", tabName = "acceuil", icon = icon("dashboard")),
      menuItem("PROFIL", tabName = "profil", icon = icon("pen")),
      menuItem("COURS", tabName = "cours", icon = icon("book-open")),
      menuItem("EXERCICES", tabName = "exo", icon=icon("person-chalkboard")),
      menuItem("DISCUSSION", tabName = "chat", icon = icon("comment")),
      menuItem("DONNEES", icon = icon("database"), href = "https://www.recenter.tamu.edu/"),
      menuItem("Liste des icônes", icon = icon("font-awesome"), href = "http://fontawesome.io/icons/")
    )
    ),
    #la partie droite principale
    dashboardBody(
     tabItems(
      tabItem(
        "Conex",
        box(
          title = "Bienvenue dans votre plate-forme KAY-DIANG",
          #footer = "Pour plus d'informations",
          status = "info",
          solidHeader = TRUE,
          width = 8,
          "Connexion"
        ),
        textInput(1, "Identifiant_élève"),
```

```
        textInput(2, "mot de passe_élève"
        ),
          textInput(1, "Identifiant_enséignant"),
          textInput(2, "Mot de passe_enséignant"
          )
      ),

      tabItem(
        "acceuil",
      box(
        title = "Le Saviez vous ?",
        footer = "Pour plus d'informations",
        status = "info",
        solidHeader = TRUE,
        width = 8,
        " Il y a autant d'étoiles dans l'Univers que de grains de sable sur Terre !
        la réponse à cette question est très simple : on..."
      ),
      infoBox(
        title = "Le Saviez vous ?",
        value = "Astronomie",
        subtitle = "Il y a autant d'étoiles dans l'Univers que de grains de sable sur Terre !La réponse
        icon = icon("planet-ringed"),
        fill = TRUE,
        color = "light-blue",
        width = 5,
        href = "https://scienceetonnante.com/2012/07/23/y-a-t-il-plus-detoiles-dans-lunivers-que-de-gra
      ),
      tabBox(
        title = "Informations",
        width = 4,
        tabPanel(title = "Niveau actuel", "8"),
        tabPanel(title = "Progression", "bien")
    )
  ),
    tabItem(
      "profil",
    box(title = "Ville", width = 4, "TOP des meilleures villes"),
    box(title = "Année", width = 4, "TOP des meilleurs années"),
    box(title = "Mois", width = 4, "TOP des meilleurs mois")
    )
  )
),
   title = "Plate-forme KAY-DIANGr",
    skin = "red" #Couleur
  ),
  server = function(input, output) {
  }
)
```