Premiers pas avec shiny... et les packages de visualisation interactive

B.Thieurmel, benoit.thieurmel@datastorm.fr
T.Robert, titouan.robert@datastorm.fr

1 Premiers pas

Créer un nouveau répertoire pour l'application shiny (**dans RStudio**):

`File -> New Project -> New Directory -> Shiny Web Application`

Choisir une application **Multiple File**.

Deux fichiers sont créés automatiquement : ui.R et server.R. Lancer directement l'application depuis RStudio via le bouton Run App (flèche verte) situé en haut à droite du script.

- Remplacer le titre de l'application par "Premiers pas avec shiny".
- Mettre à jour l'application et vérifier la prise en compte de la modification.

2 Input - Output

Pour l'instant, nous mettrons :

- Les nouveaux *inputs* dans le `sidebarPanel`, après le `sliderInput` déjà présent, en n'oubliant pas de les séparer avec une virgule !
- Les nouveaux *outputs* dans le `mainPanel`, après le `plotOutput` déjà présent, en n'oubliant pas de les séparer avec une virgule !

Créons notre premier `input` ensemble :

- En utilisant le code ci-dessous, ajouter un input dans le `ui.R` pour choisir la couleur de l'histogramme, et modifier le code dans le `server.R` (`input$color` <-> argument `col` de la fonction `hist`)

```
selectInput(inputId = "color", label = "Couleur :",
                choices = c("Rouge" = "red", "Vert" = "green", "Bleu" = "blue"))
```

Créons notre premier `ouput` ensemble en rajoutant le `summary` des données `faithful` :

```
# ui.R
verbatimTextOutput("summary")

# server.R
output$summary <- renderPrint({
  summary(faithful)
})
```

A vous de jouer !

- Permettre de renseigner le titre de l'histogramme. (Utiliser un `textInput` dans l'**ui**, et l'argument `main` de la fonction `hist` côté **server**)
- Proposer à l'utilisateur de choisir la colonne du jeu de données `faithful` qu'il souhaite représenter. (Utiliser un `radioButtons` avec comme choix `colnames(faithful)`)

- Ajouter la visualisation des données `faithful` (`renderDataTable` & `dataTableOutput`).
- Ajouter un texte sous le graphique spécifiant le nombre de classes (`input$bins`) de l'histogramme. (Utiliser un `renderText` et la fonction `paste` côté **server**, avec un `textOutput` dans le **ui**.)
- **Aller plus loin** : Essayer de rajouter des options à la visualisation des données. Vous pouvez aller voir également le package complèmentaire DT (https://github.com/rstudio/DT).

2.1 Structure

Repartons de l'application **app_structure**, équivalente à la précédente avec l'ajout d'une `navbarPage` avec maintenant :

- un onglet *Data* : visualisation des données (table + summary)
- un onglet *Visualisation* : inputs + histogramme

A vous de jouer !

- Onglet *Data* : utiliser un `navlistPanel` pour séparer le `summary` et la `table` dans deux onglets

```
# rappel de la structure (ui.R)
navlistPanel(
    "Titre de la structure de séléction",
    tabPanel("Titre de l'onglet", ... "(contenu de l'onglet)"),
    tabPanel("Titre de l'onglet", ... "(contenu de l'onglet)")
)
```

- Onglet *Visualisation* : Remplacer le `sidebarLayout` par une `fluidRow` composée de deux colonnes :
  - 1/4 : contenu actuel du `sidebarPanel`
  - 3/4 : contenu actuel du `mainPanel`

Indication : utiliser un `wellPanel` pour la colonne de gauche.

```
# rappel de la structure (ui.R)
# initialisation de la ligne
fluidRow(
  column(width = 3, ...), # colonne 1/4 (3/12)
  column(width = 9, ...)  # colonne 3/4 (9/12)
)
```

- Dans l'onglet de visualisation, rajouter le boxplot (`boxplot`, même variable et couleur, nouvel output `renderPlot` et placement dans le **ui** avec `plotOutput`). Utiliser ensuite un `tabsetPanel` pour mettre l'histogramme et le boxplot dans deux onglets distincts.

```
# rappel de la structure (ui.R)
tabsetPanel(
    tabPanel("Titre de l'onglet", ... "(contenu de l'onglet)"),
    tabPanel("Titre de l'onglet", ... "(contenu de l'onglet)")
)
```

- **Aller plus loin** : utiliser shinydashboard (https://rstudio.github.io/shinydashboard/) pour restructurer votre application.

2.2 Premiers graphiques intéractifs

- Remplacer l'histogramme et le boxplot par des graphiques javascript en utilisant le package **rAmCharts** (http://datastorm-open.github.io/introduction_ramcharts/). (`amHist`, `amBoxplot`, ...)

```r
# syntaxe pour l'histogramme côté server
# renderPlot <- renderAmCharts
output$distPlot <- renderAmCharts({
    x    <- faithful[, input$var]
    bins <- round(seq(min(x), max(x), length.out = input$bins + 1), 2)

    # use amHist
    amHist(x = x, control_hist = list(breaks = bins),
           col = input$color, main = input$titre,
           export = TRUE, zoom = TRUE)
})

# syntaxe pour l'histogramme côté ui
# plotOutput -> amChartsOutput
amChartsOutput("distPlot")
```

- **Aller plus loin** : Et pourquoi ne pas tester d'autres packages ? (http://gallery.htmlwidgets.org/)

2.3 Réactivité, isolation, observe, html, ...

- Ajouter un `actionButton` couplé à un `isolate` pour que les graphiques se mettent à uniquement lorsque l'on clique sur le bouton.
- Avec l'aide d'un `observeEvent`, faire en sorte que, lors de la validation des paramètres avec l'`actionButton`, le graphique affiché en premier soit toujours l'histogramme. (`updateTabsetPanel`)

```r
# penser à rajouter "session" en haut du server
shinyServer(function(input, output, session)

# et un identifiant au panel d'onglet
tabsetPanel(id = "viz",
  tabPanel("Histogramme", ...

# et puis finalement :
observeEvent(input$go, {
    updateTabsetPanel(session, inputId = "viz", selected = "Histogramme")
})
```

- Utiliser un `reactive` pour stocker le vecteur de la variable sélectionnée par l'utilisateur, et modifier la génération des graphiques en conséquence.

```r
# rappel de la syntaxe
data <- reactive({
  ...
})

output$plot <- renderPlot({
  # recuperation des donnees
  x <- data()
  ...
})
```

- Ajouter un titre au tableau, de couleur bleu, en utilisant `h1`, et en lui affectant un style `css`.
- Dans un troisième onglet, rédiger un petit résumé sur vous/votre société. Essayer d'ajouter un image (`div` & `img`) et un lien vers un site internet (`a`).