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

Benoît Thieurmel
Julien Bretteville - julien.bretteville@datastorm.fr
Thibaut Dubois - thibaut.dubois@datastorm.fr

1 Premiers pas

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

```
File -> New File -> New Shiny Web App
```

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` :

```r
# 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` (`DT::renderDT` & `DT::DTOutput`).
- 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**.)
- Remplacer le `selectInput` créé pour le choix de la couleur par un `colourInput` (fonction présente dans le package **colourpicker**)

https://github.com/daattali/colourpicker

- **Aller plus loin** : Essayer de rajouter des options à la visualisation des données. Vous pouvez aller voir également le package complèmentaire DT (https://rstudio.github.io/DT/).
  - Supprimer les rownames (*rownames = FALSE*)
  - Rendre possible l'édition des valeurs des cellules (utiliser l'argument *editable*)
  - Modifier les valeurs de la colonne 1 en rouge et en gras (utiliser la fonction `DT::formatStyle`)
  - Modifier la couleur de fond de la colonne 2 en vert clair
  - Mettre toutes les valeurs de la colonne 2 inférieures à 70 en violet et les autres en orange
- **Aller plus loin** : Regarder les inputs disponibles dans le package shinyWidgets https://dreamrs.github.io/shinyWidgets/ et, quand c'est possible, remplacer les inputs utilisés dans votre application par des inputs shinyWidgets
- **Aller plus loin** : Permettre à l'utilisateur d'exporter les graphiques. (`downloadButton` & `jpeg`)

3 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

Pour cette nouvelle section nous modifions le jeu de données et allons utiliser **iris**.

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` - `sidebarPanel` - `mainPanel` 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,
         wellPanel(...)), # 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.

4 Premiers graphiques intéractifs

- Remplacer l'histogramme et le boxplot par des graphiques javascript en utilisant le package **plotly** (https://github.com/plotly/plotly.R). (fonction `plot_ly()`),
  - https://plotly.com/r/histograms/ pour un exemple d'histogramme
  - https://plotly.com/r/box-plots/ pour un exemple de boxplot

La syntaxe pour l'histogramme côté serveur est la suivante avec le remplacement de **renderPlot** par **renderPlotly**

```
output$distPlot <- renderPlotly({
    x    <- iris[, input$var]
    bins <- input$bins
    # Le I fait parti de la syntaxe plotly et est nécessaire pour
```

```
    # l'utilisation de couleur, la taille des points etc.
    # Le %>% ou pipe permet d'enchainer les opérations successives.
    # Dans le cas de ce graphique, un premier appel est fait
    # pour créer le graph et un second pour y ajouter un titre.
    plot_ly(x = x, type = "histogram", nbinsx = bins, color = I(input$color)) %>%
      layout(title = input$titre)
})
```

La syntaxe pour l'histogramme côté ui est la suivante avec le remplacement de **plotOutput** par **plotlyOutput**

```
plotlyOutput("distPlot")
```

- **Aller plus loin** : Essayer d'embellir vos visualisations
  - Ajouter la variable utilisée en titre sur l'axe du boxplot (utilisation de l'argument name dans la fonction plot_ly)
  - Utiliser la variable Species pour avoir un boxplot par fleur sur le graphique (penser à utiliser la documentation https://plotly.com/r/box-plots/)
  - Changer la couleur et la taille du titre du graphique du boxplot (voir https://plotly.com/r/figure-labels/)
  - Ajouter la moyenne et l'écart-type sur les boxplots
- **Aller plus loin** : Et pourquoi ne pas tester d'autres packages ? (http://gallery.htmlwidgets.org/)
  - Créer un nouvel onglet et y ajouter une carte avec le package leaflet. Y renseigner un marker avec les coordonnées de l'université (48.119352469264065, -1.701461600501221). Voir https://rstudio.github.io/leaflet/articles/leaflet.html

5 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`, l'onglet affiché soit toujours celui de l'histogramme. (`updateTabsetPanel`)

```
# penser à rajouter "session" en haut du server
server <- 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`).
- **Aller plus loin** : Changer le thème de l'application en insérant un .css externe. Par exemple en utilisant bootswatch http://bootswatch.com/3). Attention, **shiny** utilise la version `3.3.7` de bootstrap .