Tableaux de Bord Web

Applications Web et tableaux de bord

- **R Shiny** : librairie gratuite développée par R Studio, permettant le développement d'applications Web avec R.
- **shinydashboard** : développement aisé de tableaux de bord Web.
- Structure basée sur Admin LTE (exemple).
- **Fonctionnement général** :
  - créer un répertoire dédié à l'application,
  - dans ce répertoire, créer un fichier **app.R** (toujours ce nom) dans lequel seront chargées les librairies *shiny* et *shinydashboard*,
  - le format *shiny* est reconnu par R Studio, qui propose des boutons dédiés,
  - une fois le fichier **app.R** sauvegardé, le compiler avec le bouton *Run App* ou *Reload App*.

Bases de programmation

Le squelette du fichier *app.R* doit contenir les lignes suivantes :

```
library(shiny)

ui =
server = function(input, output) {}
shinyApp(ui = ui, server = server)
```

avec

- **ui** (*user interface*) : contient tous les éléments visibles par l'utilisateur (inputs = entrées et ouputs = sorties),
- **server** (*serveur*) : contient tous les éléments de calcul des *sorties* à partir des *entrées* renseignées par l'utilisateur.
- **Remarque** : possibilité de faire 3 fichiers *app.R*, *ui.R* et *server.R*.

Tableaux de bord

Page (*Page*) de tableau de bord avec *shinydashboard* composée

- d'un bandeau (*Header*),
- d'une barre de menu verticale (*Sidebar*),
- d'un corps principal (*Body*), qui contient le rendu souhaité.

```
library(shiny)
library(shinydashboard)

ui = dashboardPage(
    dashboardHeader(),
    dashboardSidebar(),
    dashboardBody(),
    title = "Titre dans le navigateur",
    skin = "yellow"
  )
```

Titre

Titre affiché sur la gauche du bandeau :

```
dashboardHeader(
      title = "Ventes immobilières au Texas",
      titleWidth = 300
    )
```

où *titlewidth* règle la taille du texte.

Construction de l'application

- Quelques fonctions utiles pour l'affichage (*ui*)
  - textOutput()
  - tableOutput()
  - plotOutput()
  - imageOutput()
- Quelques fonctions utiles pour les calculs (*server*) : pour retourner différents types d'objets (graphiques, tableaux,textes, etc. . .).
  - renderText()
  - renderTable()
  - renderPlot()
  - renderImage()
- **Remarque** : les fonctions de types render necessitent parenthèses et accolades (*renderText({})*)

Exemple : création d'une page dynamique avec 1 output

Insertion d'un curseur dans la barre de menu :

```
dashboardSidebar(
  # Input : nombre de classes ----
  sliderInput("bins",
              label = "Nombre de classes :",
              min = 1,
              max = 50,
              value = 30)
    )
```

où *bins* définit le nom de la valeur d'entrée indiquée par le curseur.

Exemple : création d'une page dynamique avec 1 output (2)

Rendu d'un histogramme dans le corps principal :

```
dashboardBody(
        # Output: histogramme ----
        plotOutput("distPlot")
    )
```

où *distPlot* définit le nom de la sortie à afficher, qui sera calculée dans la fonction *server*.

Exemple : création d'une page dynamique avec 1 output (3)

- Charger la librairie *ggplot2* au début du fichier **app.R**.
- Calcul de la sortie nommée *displot* dans le *server* :

```
server = function(input, output) {
  output$distPlot = renderPlot({
    ggplot(txhousing, aes(sales))+
      geom_histogram(aes(y=..density..),
                     bins=input$bins,
                     colour = "white",
                     fill = "steelblue"
      )+
      labs(x = "Nombre de ventes",
           y = "Densite",
           title = "Histogramme du nombre de ventes"
      )+
      theme_light()
  })
```

TP : Exercice 1 avec shiny, shinydashboard et ggplot2

- Créer un répertoire pour une application web concernant les Iris de Fisher
- Créer un fichier *app.R* dans ce répertoire avec les informations suivantes :
  - ui : titre, couleur, barre de menu avec une barre dynamique permettant de choisir le nombre de classes dans un histogramme, corps principal rendant l'output calculé dans *server*,
  - server : calcul et rendu avec ggplot2 de l'histogramme en densité de la longueur de Sépale, avec des bordures blanches et une couleur de remplissage bleue claire (indication : options *color=* et *fill=* dans la fonction *geom_histogram()*)

A faire

- Dans le fichier app.R de l'application sur les ventes immobilières au Texas, effacer l'exemple simple.
- Faire de même dans le fichier app.R sur les Iris de Fisher.

Boîtes

- *shinydashboard* : gestion des éléments du corps principal sous forme de **boîtes**.
  - **box()** : boîte générique, permettant notamment l'inclusion de graphiques.
  - **infoBox()** : boîte d'information courte, de type texte ou valeur numérique.
  - **valueBox()** : boîte d'information courte, de type valeur numérique.
  - **tabBox()** : boîte tableau, avec possibilité de rendu de plusieurs panneaux (*panels*).

```
dashboardBody(
  box(),
  infoBox(),
  valueBox(),
  tabBox()
)
```

- Gestion de l'aspect des boîtes via des options.

Boîte générique

- Exemple : affichage de l'évolution du volume des ventes immobilières au Texas entre 2000 et 2015.
- Ajouter la boîte suivante dans le corps principal :

```
box(
  title = "Evolution du volume des ventes",
  footer = "en US$",
  status = "info",
  solidHeader = TRUE,
  width = 8,
  plotOutput("evolution")
  )
```

- où
  - *title* donne un titre à la boîte,
  - *footer* met un texte en pied de page,
  - *status* définit la couleur de la boîte (voir ?validStatuses),
  - *solidHeader* indique si le titre a une couleur de fond,
  - *width* règle la taille de la boîte (de 1 à 12).

Boîte générique (2)

- Calcul de l'évolution globale, à inclure au début du fichier, avant les définitions de *ui* et *server* :

```
library(dplyr)
evol_globale = txhousing %>%
  group_by(year) %>%
  summarise(volume = sum(volume, na.rm = T))
```

- Calcul de la sortie *evolution* dans la fonction *server* :

```
server = function(input, output) {
    output$evolution = renderPlot({
      ggplot(evol_globale, aes(year, volume)) +
        geom_line() +
        theme_minimal() +
        labs(x = "Annee", y = "Volume des ventes")
    })
  }
```

Boîte information

- Exemple : information sur la progression du volume des ventes.
- Ajouter la boîte suivante dans le corps principal :

```
infoBox(
  title = "Progression",
  value = textOutput("progression"),
  subtitle = "Entre 2000 et 2015",
  icon = icon("line-chart"),
  fill = TRUE,
  color = "light-blue",
  width = 4
      )
```

- où
  - *value* indique la valeur à afficher, ici un élément calculé,
  - *subtitle* ajoute un sous-titre à la boîte,
  - *icon* définit l'icône à afficher à gauche de la boîte,
  - *fill* indique si le texte remplit la boîte,
  - *color* définit la couleur de la boîte.

Boîte information (2)

Calcul de la sortie *progression* : ajouter la ligne de commande suivante dans la fonction *server*

```
output$progression = renderText({
  paste(round(
    tail(evol_globale$volume, 1) /
      head(evol_globale$volume, 1) *
      100
  ),
  "%")
})
```

Boîte valeur

- Exemple : affichage du volume total des ventes.
- Ajouter la boîte suivante dans le corps principal :

```
valueBox(
        value = textOutput("volume"),
        subtitle = "Volume totale des ventes (en milliards)
        icon = icon("usd"),
        color = "green",
        width = 4
      )
```

- où *value* est une valeur calculée.

Boîte valeur (2)

Calcul de la sortie *volume* : ajouter la ligne de commande suivante dans la fonction *server*

```
output$volume = renderText({
      round(
        sum(evol_globale$volume, na.rm = T) / 1e+9, 1
        )
    })
```

Boîte tableau

- Exemple : affichage d'un tableau résumant les informations sur le prix médian et le nombre de ventes.
- Ajouter la boîte suivante dans le corps principal :

```
tabBox(title = "Informations",
        width = 4,
        tabPanel(title = "Prix médian",
          tableOutput("info_prix")
          ),
        tabPanel(title = "Nombre",
          tableOutput("info_nombre")
          )
      )
```

- où *tabPanel()* définit un panneau, dans lequel la sortie est affichée, généralement calculée dans *server* sous forme de *data.frame*.

Boîte tableau (2)

Calcul de la sortie *info_prix* : ajouter la ligne de commande suivante dans la fonction *server*

```
output$info_prix = renderTable({
      data.frame(
        Statistique = c("Minimum", "Médiane", "Maximum"),
        Valeur = c(
          min(txhousing$median, na.rm = T),
          median(txhousing$median, na.rm = T),
          max(txhousing$median, na.rm = T)
        )
      )
    })
```

Boîte tableau (3)

Calcul de la sortie *info_nombre* : ajouter la ligne de commande suivante dans la fonction *server*

```
output$info_nombre = renderTable({
      data.frame(
        Statistique = c("Minimum", "Médiane", "Maximum"),
        Valeur = c(
          min(txhousing$sales, na.rm = T),
          median(txhousing$sales, na.rm = T),
          max(txhousing$sales, na.rm = T)
        )
      )
    })
```

TP : Exercice 2 avec shiny, shinydashboard, dplyr et ggplot2

- Reprendre l'application sur les Iris de Fisher.
- Créer une page affichant les informations suivantes :
  - une boîte donnant les informations sur les données (nombre d'observations, nombre d'iris de chaque espèce, et toute information utile),
  - une boîte donnant les informations sur les variables,
  - une boîte affichant le diagramme circulaire de la distribution des espèces (voir les TPs sur la visualisation de données),
  - une boîte affichant un tableau résumant les données par espèce, avec un panneau par variable.
- Voir ici pour une liste d'icônes.
- **Rappel** : charger les librairies et effectuer le calcul de tout élément utile à la fonction *server* en début de code dans le fichier *app.R*.

Menus

- Possibilité d'avoir plusieurs pages de tableau de bord, cliquables à partir du menu à gauche.
- Squelette de *dashboardSidebar* :

```
dashboardSidebar(
  sidebarMenu(
    menuItem("titre",
             tabName = "nom",
             icon = icon("dashboard")
             )
  )
)
```

- où
  - *titre* = nom apparaissant dans le menu, et *icon* l'icône associée,
  - *nom* = nom donné à la page permettant d'associer le lien du menu à la page correspondante dans le corps principal.

Menus (2)

- Création d'une page par élément du menu dans le corps principal :

```
dashboardBody(
  tabItems(
        tabItem(
          "nom",
          ...
        )
  )
)
```

- où *nom* permet de faire le lien entre la page et l'élément du menu correspondant.

Menus (3)

- Exemple : ajout de deux liens dans le menu

```
dashboardSidebar(
      sidebarMenu(
        menuItem("Vue globale", tabName = "vue",
                 icon = icon("dashboard")
                 ),
        menuItem("Données",
                 icon = icon("database"),
                 href = "https://www.recenter.tamu.edu/"
                 ),
        menuItem("Liste des icônes",
                 icon = icon("font-awesome"),
                 href = "http://fontawesome.io/icons/"
                 )
      )
    )
```

Menus (4)

- Exemple : relier le lien de menu nommé *vue* à la page créée précédemment :

```
dashboardBody(
      tabItems(
        tabItem(
          "vue",
          "coller ici les boîtes"
        )
        )
      )
```

TP : Exercice 3 avec shiny, shinydashboard, dplyr, ggplot2, scales et forcats

- Reprendre l'application sur les Iris de Fisher.
- Créer un menu avec deux éléments :
  - un élément de résumé global,
  - un élément concernant la description des variables.
- Intégrer la page calculée précédemment dans l'élément de résumé global.
- Créer une page pour l'élément de description de la variable *Sepal.Length* contenant
  - l'histogramme en densité,
  - les histogrammes en densité (sur le même graphique), un par espèce d'iris,
  - la distribution par espèce sous forme de boîtes à moustaches, avec les moyennes et les barres d'erreur, classées par ordre croissant de la moyenne.
- **Indication** : pour les différents graphiques, reprendre les codes des TPs sur la visualisation de données.

Interaction avec l'utilisateur

- Possibilité d'ajouter des boîtes, ou d'inclure dans une boîte, un choix (*control widget*) rentré par l'utilisateur.
- Choix géré dans l'application sous forme d'**input**.
- Sous forme de curseur, de choix dans une liste, de rentrée d'un chiffre, etc...
- Pour une liste des choix les plus communs, voir ici.
- Choix gérés par la fonction **selectInput()**, à intégrer dans l'*ui* à la place voulue (dans le menu, dans le corps principal, dans une boîte, etc...).

Interaction avec l'utilisateur (2)

- Exemple : ajouter une boîte contenant un choix déroulant des villes dans le corps principal, sous la première boîte

```
box(
  width = 4,
  selectInput("ville",
              "Ville choisie",
              choices = c(
                "Toutes les villes",
                unique(txhousing$city)
                )
              )
          )
```

- où :
  - *ville* est le nom de l'input, qui sera repris dans la fonction *server*,
  - *Ville choisie* est le nom associé à la boîte.

Interaction avec l'utilisateur (3)

- Exemple : changer le calcul de la sortie *evolution* dans la fonction *server*

```
output$evolution = renderPlot({
      if (input$ville == "Toutes les villes") {
        evol = evol_globale
      } else {
        evol = txhousing %>%
          filter(city == input$ville) %>%
          group_by(year) %>%
          summarise(volume = sum(volume, na.rm = T))
      }
      ggplot(evol, aes(year, volume)) +
        geom_line() +
        theme_minimal() +
        labs(x = "", y = "Volume des ventes")
    })
```

Interaction avec l'utilisateur (4)

- Exemple : changer le calcul de la sortie *progression* dans la fonction *server*

```
output$progression = renderText({
      if (input$ville == "Toutes les villes") {
        evol = evol_globale
      } else {
        evol = txhousing %>%
          filter(city == input$ville) %>%
          group_by(year) %>%
          summarise(volume = sum(volume, na.rm = T))
      }
      paste(round(tail(evol$volume, 1) /
                    head(evol$volume, 1) * 100),
            "%")
    })
```

TP : Exercice 4 avec shiny, shinydashboard, dplyr, ggplot2, scales et forcats

- Reprendre l'application sur les Iris de Fisher.
- Sur la page "description", ajouter :
  - une boîte avec un curseur permettant de choisir le nombre de classes dans les deux histogrammes,
  - une boîte permettant de choisir la variable à décrire parmi les 4 variables de mesures de Sépales et de Pétales, et rendant les mêmes graphiques que pour la variable *Sepal.Length* seule.

Interaction avec l'utilisateur (5)

- Variable *reactive()* : une fois l'*input* choisi, permet le calcul une seule fois d'un même élément pour plusieurs *output*.
- Exemple : ajouter la variable *reactive()* suivante dans la fonction *server*

```
donnees = reactive({
      if (input$ville == "Toutes les villes") {
        evol = evol_globale
      } else {
        evol = txhousing %>%
          filter(city == input$ville) %>%
          group_by(year) %>%
          summarise(volume = sum(volume, na.rm = T))
      }
      evol
    })
```

Interaction avec l'utilisateur (6)

- Exemple (suite) : dans le calcul des *output*, changer *evol* par *donnees()*

```
output$evolution = renderPlot({
  ggplot(donnees(), aes(year, volume)) +
    geom_line() +
    theme_minimal() +
    labs(x = "", y = "Volume des ventes")
})

output$progression = renderText({
  evol = donnees()
  paste(round(
    tail(evol$volume, 1) /
      head(evol$volume, 1) * 100
      ),
    "%")
})
```

TP : Exercice 5 avec shiny, shinydashboard, dplyr, ggplot2, scales et forcats

- Reprendre l'application sur les Iris de Fisher.
- Créer une variable *reactive()* permettant de fixer la variable choisie sur la page "description" pour les 3 graphiques représentés.
  Indication : afin de gérer les noms de variables à la fois avec *ggplot* et *dplyr*, utiliser la fonction *aes_string()* dans ggplot (voir l'aide).

Tableaux : présentation améliorée

- Librairie *DT* :
  - génération de tableaux paramétrables,
  - meilleure gestion des grandes tables de données,
  - présentations plus intéressantes (et interactives).
- Génération du tableau avec la fonction *datatable()* :

```
datatable(tableau)
```

où *tableau* est le tableau à afficher, de type *data.frame*.

- Rendu de la sortie (*output*) avec la fonction *renderDataTable({})*

```
renderDataTable({
    datatable(tableau)
  })
```

- Documentation consultable sur https://rstudio.github.io/DT/.

Tableaux : présentation améliorée (2)

- Exemple : ajouter la table de données dans le menu (*dashboardSidebar()*), associé à la page *donnees*

```
menuItem("Tableau", tabName = "donnees",
         icon = icon("table")
       )
```

- Définir la page “donnees” (dans *dashboardBody()*)

```
tabItem(
         "donnees",
         dataTableOutput("tableau")
       )
```

- Dans la fonction *server()*, calculer la sortie *tableau*

```
output$tableau <- renderDataTable({
    datatable(txhousing)
  })
```

Tableaux : présentation améliorée (3)

Quelques options utiles de la fonction *datatable()* :

- *rownames* = *FALSE* : supprime les noms des lignes,
- *colnames* = *c('nouveau nom' = 'nom d'origine', ...)* : change les noms des variables d'origine indiquées dans le tableau en de nouveaux noms,
- *caption* = *'titre'* : ajoute un titre au tableau,
- *filter* = *xxx* : ajoute un filtre sur les colonnes, où
  - *xxx* = *'none'* => pas de filtre,
  - *xxx* = *'top'* => filtre positionné en haut du tableau,
  - *xxx* = *'bottom'* => filtre positionné en bas du tableau.

Tableaux : présentation améliorée (4)

- Exemple : supprimer les noms des lignes, ajouter un titre, changer les noms de 3 variables, ajouter un filtre dans la sortie *tableau*

```
output$tableau <- renderDataTable({
  datatable(txhousing,
            rownames = FALSE,
            colnames = c('Ville' = 'city',
                         'Année' = 'year',
                         'Mois' = 'month'),
            caption = "Données concernant
            les ventes immobilières au Texas
            - 2000-2015",
            filter = 'top')
  })
```

Tableaux : présentation améliorée (5)

- Possibilité de formater une ou plusieurs colonnes :
  - *formatStyle()* : fromatage du style (couleur de fond, couleur du texte, police, etc. . .),
  - *formatCurrency()* : formatage monétaire,
  - *formatDate()* : formatage de la date.
- Possibilité de colorer suivant les valeurs dans une colonne avec *styleColorBar()*.

Tableaux : présentation améliorée (6)

- Exemple : formater le tableau issu de la fonction *datatable()* :

```
library(lubridate)
datatable(txhousing %>%
            mutate(date = date_decimal(date))
) %>%
  formatCurrency(c("volume", "median")) %>%
  formatDate("date", "toLocaleDateString") %>%
  formatStyle('city',
              color = 'white',
              backgroundColor = 'slategrey',
              fontWeight = 'bold') %>%
  formatStyle('median',
              background = styleColorBar(
                range(txhousing$median, na.rm = TRUE),
                'lightblue')
  )
```

Tableaux : présentation améliorée (7)

- Quelques extensions de la fonction *datatable()* :
  - *extensions* = *'Button'* : affiche un bouton permettant de télécharger le tableau de données,
  - *extensions* = *'FixedColumns'* : fixe des colonnes, et permet le scrolling lorsque toutes les colonnes ne sont pas visibles.
- Options pour les extensions, sous la forme *options* = *list()* :
  - *dom* = : définit dans quel ordre apparaissent les éléments additionnels (boutons ou scrolling),
  - *buttons* = *c('copy', 'csv', 'excel', 'pdf', 'print')* : boutons proposés pour le format du fichier téléchargé,
  - *scrollX* = *TRUE* : permet le scrolling sur les colonnes,
  - *fixedColumns* = *list(leftColumns* = *a, rightColumns* = *b)* : nombre de colonnes fixes à gauche (=*a*), et à droite (=*b*) lorsque l'on scroll.
- Une liste des extensions possibles est consultable sur cette page.

Tableaux : présentation améliorée (8)

- Exemple : ajouter une barre de scrolling sur le tableau

```
datatable(
  txhousing,
  rownames = FALSE,
  extensions = 'FixedColumns',
  options = list(
    dom = 't',
    scrollX = TRUE,
    fixedColumns = list(leftColumns = 1, rightColumns = 1)
  )
)
```

TP : Exercice 6 avec shiny, shinydashboard, dplyr, ggplot2, scales, forcats et DT

- Reprendre l'application sur les Iris de Fisher.
- Avec la librairie *DT*, améliorer le tableau résumant les variables sur la page "résumé global" :
  - Changer les noms des variables afin d'obtenir un tableau plus lisible,
  - changer la couleur de fond, la fonte et la couleur de texte de la colonne concernant les espèces.

Cartes choroplèthes avec leaflet

- Génération de cartes avec *leaflet*.
- Dans la fonction *ui()* et *dashboarBody()*, la commande de rendu est

```
leafletOutput("nom")
```

où *nom* est le nom de l'*output* souhaité.

- Dans la fonction *server()*, la commande de calcul est

```
library(leaflet)
nom$output = renderLeaflet({
  leaflet() %>%
    addTiles() %>%
    addPolygons()
})
```

Cartes choroplèthes avec leaflet (2)

- Exemple : Calcul de la carte couleurs fonction de la somme des volumes des ventes, en amont des fonctions *ui()* et *server()*. Calcul des éléments et de la palette de couleurs :

```
resume = txhousing %>%
  group_by(city) %>%
  summarise(volume = sum(volume, na.rm = TRUE))

txgeo = geojson_read("texas-city.geojson", what = "sp")
txgeo = subset(txgeo, sub(", TX", "", name) %in%
                 unique(txhousing$city))

txgeo@data$city = sub(", TX", "", txgeo@data$name)
txgeo@data = txgeo@data %>%
  left_join(resume, all.x = TRUE)

pal = colorNumeric("viridis", NULL)
```

Cartes choroplèthes avec leaflet (3)

- Calcul de la carte (toujours en amont des fonctions *ui()* et *server()*) :

```
map = leaflet(txgeo) %>%
  addTiles() %>%
  addPolygons(fillColor = ~pal(volume),
              fillOpacity = .5,
              color = "red", weight = 1,
              label = ~paste0(
                city, ": ",
                formatC(volume, big.mark = ",")
                )
              ) %>%
  addLegend(pal = pal, values = ~volume, opacity = 1.0,
            labFormat = labelFormat(
              transform = function(x) round(x)
              )
            )
```

Cartes choroplèthes avec leaflet (3)

- Ajout de la carte dans un menu dans *dashboardSidebar()*,

```
menuItem("Carte", tabName = "carte",
         icon = icon("map")
       )
```

- Ajout de la carte sur une page dans *dashboardBody()*,

```
tabItem(
         "carte",
         leafletOutput("carte")
       )
```

- et calcul dans *server()*.

```
output$carte = renderLeaflet({
  map
})
```