Initiation à RShiny

Découvrir RShiny et comprendre son fonctionnement

NZONDE David Christ

DeepStat Consulting

17 août 2025

Objectifs de la formation

- **Comprendre ce qu'est Shiny** : découvrir le framework R, son architecture et ses principaux usages pour créer des applications web interactives.
- **Manipuler widgets et outputs** : apprendre à interagir avec les inputs et afficher les résultats de manière dynamique.
- **Construire un dashboard** : organiser l'application avec `shinydashboard` pour présenter des données et visualisations de manière professionnelle.
- **Insertion d'un modèle IA** : apprendre à intégrer un modèle d'intelligence artificielle dans ses applications via des API, pour enrichir l'interactivité et les analyses.

But pédagogique *: À l'issue de cette formation, vous serez capables de créer une application interactive complète en RShiny, de manipuler des widgets et outputs, d'intégrer des modèles IA et de déployer votre application.*

Plan de la formation

Introduction

- **RShiny** est un package R permettant de créer facilement des **applications web interactives**.
- Il simplifie la construction de **dashboards dynamiques et interactifs** pour la visualisation de données, les analyses statistiques et l'intégration de modèles prédictifs.
- **Points clés** :
  - Exécute du R côté serveur et affiche les résultats dans un navigateur web.
  - Permet de rendre des analyses statistiques **interactives**, sans nécessiter de compétences en HTML/CSS/JS.
  - Idéal pour les **rapports dynamiques**, la visualisation de données et la démonstration de modèles IA.

Fonctionnement de Shiny

Shiny repose sur un principe simple : les **inputs** (ce que l'utilisateur saisit ou sélectionne) déclenchent des **outputs** (résultats affichés), le tout grâce à une **logique réactive**.

Architecture de base

Une application Shiny se compose de deux parties principales :

- **UI (User Interface)** : décrit l'interface utilisateur, c'est-à-dire ce que l'utilisateur voit et avec quoi il interagit (widgets, graphiques, tableaux...).
- **Server** : contient la logique et calcule les résultats à afficher en fonction des inputs.

Schéma simplifié :
Utilisateur -> Input (widget) -> Logique réactive -> Output (texte, graphique, tableau)

Structure d'une application Shiny

Fichier unique : app.R

```r
library(shiny)

ui <- fluidPage(
  textInput("nom", "Votre nom :"),
  textOutput("bonjour")
)

server <- function(input, output, session) {
  output$bonjour <- renderText({
    paste("Bonjour", input$nom)
  })
}

shinyApp(ui, server)
```

Les Inputs (Widgets)

Les **inputs** dans Shiny sont les éléments qui permettent à l'utilisateur de **saisir ou sélectionner des informations**. Ces informations sont ensuite utilisées par le serveur pour générer les outputs.

Exemples de widgets courants (1/2)

Widget	Description	Usage
`textInput`	Champ texte pour saisir une valeur	Nom, email
`numericInput`	Champ numérique	Quantité, âge
`selectInput`	Liste déroulante pour choisir une option	Pays, catégorie
`radioButtons`	Boutons radio pour choisir une option	Oui/Non
`checkboxInput`	Case à cocher (vrai/faux)	Accepter les conditions

Exemples de widgets courants (2/2)

Widget	Description	Usage
`sliderInput`	Curseur pour sélectionner une valeur	Nombre de classes d'un histogramme
`actionButton`	Bouton déclenchant une action	Soumettre, Actualiser

Plus de widgets

Retrouvez plus de widgets dans la galerie Shiny : Shiny Widget Gallery

Exemple pratique

```r
library(shiny)

ui <- fluidPage(
  textInput("nom", "Votre nom :"),
  selectInput("pays", "Pays :", choices = c("Sénégal", "Camero
  sliderInput("age", "Âge :", min = 0, max = 100, value = 25),
  actionButton("go", "Envoyer")
)

server <- function(input, output, session) {
  output$info <- renderText({
    paste("Bonjour", input$nom, "du", input$pays)
  })
}

shinyApp(ui, server)
```

Les Outputs

Les **outputs** permettent d'afficher les résultats calculés par le serveur Shiny dans l'interface utilisateur.
Chaque output correspond à une fonction `render*()` côté serveur et à un `*Output()` côté UI.

Exemples courants d'outputs

Output	Fonction render()	Description
`textOutput`	`renderText()`	Affiche du texte simple
`tableOutput`	`renderTable()`	Affiche un tableau de données
`plotOutput`	`renderPlot()`	Affiche un graphique
`verbatimTextOutput`	`renderPrint()`	Affiche le résultat d'un print() ou résumé R
`uiOutput`	`renderUI()`	Affiche des éléments UI dynamiques

Exemple pratique

```r
library(shiny)

ui <- fluidPage(
  numericInput("age", "Votre âge :", value = 25),
  textOutput("ageMessage")
)

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

  output$ageMessage <- renderText({
    paste("Vous avez", input$age, "ans")
  })
}

shinyApp(ui, server)
```

La logique réactive

La **logique réactive** est le coeur de Shiny : elle permet à l'application de **réagir automatiquement** aux changements des inputs et de recalculer les outputs.

Principe

1. Un utilisateur modifie un **input** (widget).
2. Les fonctions **reactives** calculent les résultats en fonction de cet input.
3. Les **outputs** sont mis à jour automatiquement dans l'interface.

Les fonctions réactives

- `reactive({ ... })` : crée un objet réactif que l'on peut réutiliser dans plusieurs outputs.
- `observe({ ... })` : exécute du code réactif sans produire de valeur pour l'UI.
- `observeEvent(input$...)` : déclenche du code lorsqu'un input spécifique change.

Exemple pratique (1/2)

```
library(shiny)

ui <- fluidPage(
  numericInput("length", "Taille de la base :", value = 200),
  sliderInput("bins", "Nombre de classes :", min = 5, max = 50
  value = 20),
  plotOutput("distPlot")
)

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

  # Données réactives
  data <- reactive({
    rnorm(input$length)
  })
```

Exemple pratique (2/2)

```
  # Breaks réactifs pour l'histogramme
  breaks <- reactive({
    seq(min(data()), max(data()), length.out = input$bins + 1)
  })

  # Histogramme réactif
  output$distPlot <- renderPlot({
    hist(data(), breaks = breaks(), col = "skyblue", border =
         main = "Histogramme interactif")
  })
}

shinyApp(ui, server)
```

Organisation avec `shinydashboard`

`shinydashboard` est un package R qui facilite la création de dashboards professionnels et organisés.

Structure d'un dashboard

Un dashboard est construit avec `dashboardPage()` et se compose de trois parties :

1. **`dashboardHeader()`** : l'en-tête du tableau de bord
   - Contient le titre et éventuellement des menus de notifications ou d'icônes.
2. **`dashboardSidebar()`** : la barre latérale
   - Permet d'ajouter un menu de navigation avec `sidebarMenu()` et `menuItem()`.
   - Idéal pour organiser les différentes sections de l'application.
3. **`dashboardBody()`** : le corps principal
   - Contient les graphiques, tableaux et autres outputs.
   - On peut organiser le contenu avec `fluidRow()` et `box()`.

Exemple minimal (1/3)

```
library(shiny)
library(shinydashboard)

ui <- dashboardPage(
  dashboardHeader(title = "Mon Dashboard"),
  dashboardSidebar(
    sidebarMenu(
      menuItem("Accueil", tabName = "home", icon = icon("home"
      menuItem("Analyse", tabName = "analyse", icon = icon("ch
    )
  ),
  dashboardBody(
    tabItems(
      tabItem(tabName = "home",
              h2("Bienvenue sur le dashboard")
      ),
```

Exemple minimal (2/3)

```
tabItem(tabName = "analyse",
                fluidRow(
                  box(plotOutput("plot1"), width = 6),
                  box(tableOutput("table1"), width = 6)
                )
        )
    )
  )
)

server <- function(input, output, session) {
  output$plot1 <- renderPlot({
    hist(rnorm(100), col = "skyblue", border = "white")
  })
```

Exemple minimal (3/3)

```
  output$table1 <- renderTable({
    head(mtcars)
  })
}

shinyApp(ui, server)
```

Cas pratique + insertion d'un modèle IA

Nous allons réaliser un dashboard pour la structure DeepStat, à partir d'une base de données fictive, renseignant sur les activités de la structure au fil du temps, ainsi que la composition du personnel. Nous intègrerons un assistant IA à l'aide de l' API Groq

Le dashboard se fera suivant ce modèle : DeepStat-Dashboard

Déploiement d'une application Shiny

1. Déploiement local

- Utilisez la fonction `runApp()` pour exécuter l'application sur votre ordinateur.
- L'application s'ouvre dans votre navigateur par défaut.
- Pratique pour tester et développer avant le déploiement en ligne.

```
library(shiny)
runApp("chemin/vers/votre/app")
```

2. Déploiement sur shinyapps.io

- Plateforme officielle pour héberger vos applications Shiny en ligne.
- Gratuit pour un usage limité, avec plans payants pour plus de ressources.
- Étapes :
  1. Créer un compte sur shinyapps.io
  2. Installer le package `rsconnect` (**install.packages("rsconnect")**)
  3. Connecter votre compte

```
library(rsconnect)
rsconnect::setAccountInfo(name='votre_nom', token='xxxx',
```

  4. Déployer l'application (**rsconnect::deployApp("chemin/vers/votre/app")**)

Remarque

- Il est également possible de déployer en cliquant sur `Publish` dans sa fenêtre **RStudio**

Bonnes pratiques et Conclusion

Bonnes pratiques (1/2)

1. **Organisation du code**
   - Séparer la logique en modules si l'application devient complexe.
   - Garder le code lisible et commenté.
2. **Interface utilisateur**
   - Concevoir une interface claire et intuitive.
   - Éviter de surcharger l'écran avec trop de widgets ou graphiques.

Bonnes pratiques (2/2)

3. **Réactivité et performance**
   - Utiliser `reactive()` et `observe()` pour optimiser les calculs.
   - Éviter de recalculer des données lourdes inutilement.
4. **Test et maintenance**
   - Tester régulièrement chaque input et output.
   - Vérifier la compatibilité des packages et mises à jour R.
5. **Sécurité et confidentialité**
   - Ne pas exposer de données sensibles.
   - Valider les entrées utilisateur si nécessaire.

Conclusion

- **Shiny** permet de transformer vos scripts R en applications web interactives.
- La **réactivité** est au cœur du fonctionnement : tout changement d'input déclenche la mise à jour automatique des outputs.
- Avec `shinydashboard`, vous pouvez structurer vos applications comme de vrais dashboards analytiques.
- Le **déploiement** peut se faire localement, sur shinyapps.io, un serveur Shiny ou via Docker.
- Prochaine étape : **pratiquez en créant vos propres applications et dashboards** pour maîtriser pleinement Shiny.

Merci de votre attention !!!

Contacts

- DeepStat sur linkedIn: DeepStat Consulting
- Mon compte linkedIn: David Christ