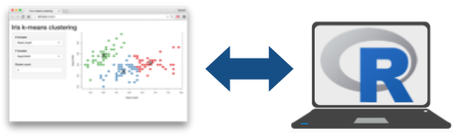
Ứng dụng phân tích dữ liệu với Shiny Cheat Sheet

Xem thêm tại shiny.rstudio.com

RStudio

Kiến thức cơ bản

Ứng dụng Shiny là một website (UI) được kết nối với máy tính thực hiện các phương pháp phân tích trên R (Server)



Khi người dùng thay đổi UI, Server sẽ cập nhật UI bằng cách thực hiện các câu lệnh trong R

Cấu trúc ứng dụng

Thực hiện ứng dụng đơn giản với cấu trúc dưới đây. Thực hiện các câu lệnh trong R để xem kết quả:

```
library(shiny)
ui <- fluidPage()
server <- function(input, output){}
shinyApp(ui = ui, server = server)
```

- **ui** – Hàm cho phép hiển thị giao diện UI dạng HTML trong R
- **server** – Hàm cho phép xây dựng các đối tượng được hiển thị trong UI
- **shinyApp** – ứng dụng hoàn chỉnh, kết hợp giữa UI và Server. Gọi kết quả bằng hàm **runApp()**

Chia sẻ ứng dụng Shiny

shinyapps.io

Cách đơn giản nhất để chia sẻ ứng dụng Shiny là host trên website **shinyapps.io** của RStudio

1. Tạo tài khoản tại: http://shinyapps.io
2. Ấn vào nút Pulish trên RStudio IDE (>=0.99) or run:

```
rsconnect::deployApp("<path to directory>")
```

Xây dựng hoặc mua Shiny Server tại www.rstudio.com/products/shiny-server/

Xây dựng ứng dụng – Bổ sung các tham số vào hàm **fluidPage()** và các câu lệnh của hàm server.

Thêm dữ liệu đầu vào với hàm ***Input()**

Thêm kết quả đầu ra vơi hàm ***Output()**

Khai báo cách thức trả kết quả đầu ra trên server:

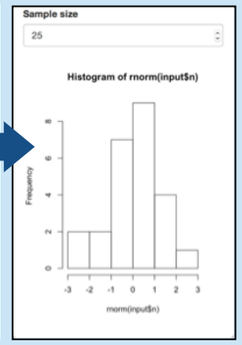
1. Trả kết quả đầu ra output$<id>
2. Gọi kết quả đầu vào input$<id>
3. Sắp xếp các câu lệnh trong hàm **render*()** trước khi lưu lại thành kết quả đầu ra (output)

```
library(shiny)

ui <- fluidPage(
  numericInput(inputId = "n",
  "Sample size", value = 25),
  plotOutput(outputId = "hist")
)

server <- function(input, output) {
  output$hist <- renderPlot({
   hist(rnorm(input$n))
  })
}

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



Lưu app về dạng **app.R,** hoặc lưu thành 2 file riêng biệt **ui.R** và **server.R**.

```
library(shiny)
ui <- fluidPage(
  numericInput(inputId = "n",
    "Sample size", value = 25),
  plotOutput(outputId = "hist")
)
server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$n))
  })
}
shinyApp(ui = ui, server = server)
```

```
# ui.R
fluidPage(
    numericInput(inputId = "n",
        "Sample size", value = 25),
    plotOutput(outputId = "hist")
)
```

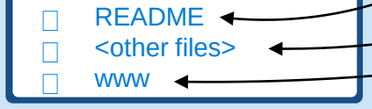
```
# server.R
function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$n))
  })
}
```

ui.R chứa tất cả những thông tin liên quan đến giao diện của ứng dụng

server.R chứa các hàm trên server

Không cần phải gọi hàm **shinyApp()**

Lưu mỗi app trong một thư mục có chứa file **app.R** (hoặc **ui.R** và **server.R**) và các file đính kèm khác:



Tên của thư mục là tên của app

Xác định các đối tượng cho ui.R và server.R (không bắt buộc)

Sử dụng khi xem demo (không bắt buộc)

Data, scripts, ... (không bắt buộc)

Thư mục khác, sử dụng khi chia sẻ trên web (không bắt buộc)

Browsers (images, CSS, .js, ...) Phải được đặt tên là **www**

Chạy ứng dụng bằng câu lệnh **runApp(<đường dẫn thư mục>)**

Outputs - render*() & *Output() được sử dụng cùng nhau để hiển thị kết quả lên UI

DT::renderDataTable(expr, options, callback, escape, env, quoted)

Thực hiện cùng hàm

dataTableOutput(outputId, icon, …)

renderImage(expr, env, quoted, deleteFile)

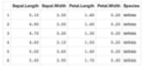
imageOutput(outputId, width, height, click, dblclick, hover, hoverDelay, hoverDelayType, brush, clickId, hoverId, inline)

renderPlot(expr, width, height, res, …, env, quoted, func)

plotOutput(outputId, width, height, click, dblclick, hover, hoverDelay, hoverDelayType, brush, clickId, hoverId, inline)

renderPrint(expr, env, quoted, func, width)

verbatimTextOutput(outputId)



renderTable(expr,…, env, quoted, func)

tableOutput(outputId)

foo

renderText(expr, env, quoted, func)

textOutput(outputId, container, inline)

renderUI(expr, env, quoted, func)

uiOutput(outputId, inline, container, …)

htmlOutput(outputId, inline, container, …)

&

Inputs – Các giá trị đầu vào từ người dùng

Gọi kết quả đầu vào với hàm **input$<inputId>**.

Action — **actionButton**(inputId, label, icon,…)

Link — **actionLink**(inputId, label, icon, …)

Choice 1, Choice 2, Choice 3 — **checkboxGroupInput**(inputId, label, choices, selected, inline)

Check me — **checkboxInput**(inputId, label, value)

dateInput(inputId, label, value, min, max, format, startview, weekstart, language)

dateRangeInput(inputId, label, start, end, min, max, format, startview, weekstart, language, separator)

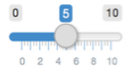
Choose File — **fileInput**(inputId, label, multiple, accept)

1 — **numericInput**(inputId, label, value, min, max, step)

••••••• — **passwordInput**(inputId, label, value)

Choice A, Choice B, Choice C — **radioButtons**(inputId, label, choices, selected, inline)

Choice 1 / Choice 1, Choice 2 — **selectInput**(inputId, label, choices, selected, multiple, selectize, width, size) (selectizeInput())



sliderInput(inputId, label, min, max, value, step, round, format, locale, ticks, animate, width, sep, pre, post)

Apply Changes — **submitButton**(text, icon)
(Tránh việc phân tích kết quả liên tục mỗi khi dữ liệu đầu vào thay đổi)

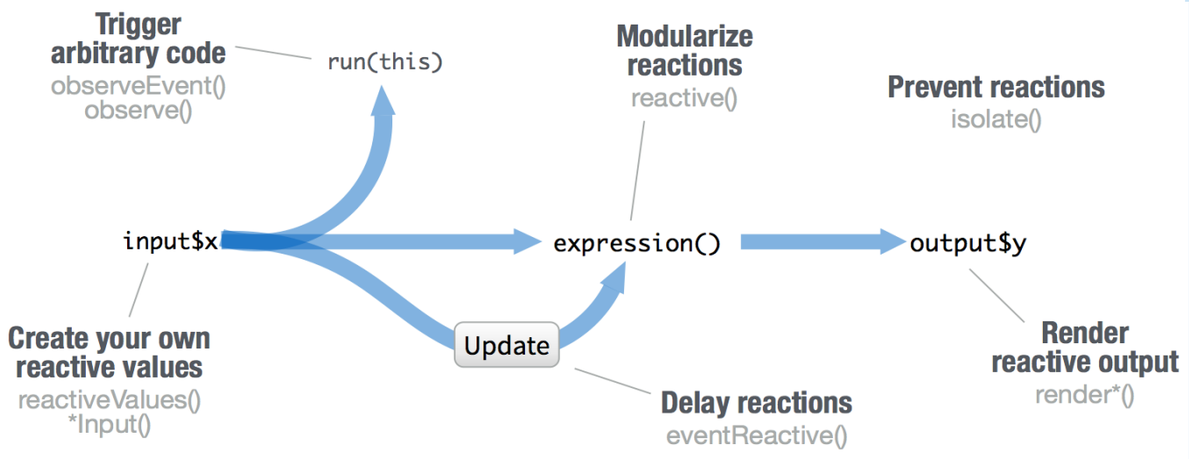
Enter text — **textInput**(inputId, label, value)

Xem thêm cheat sheets tại http://www.rstudio.com/resources/cheatsheets/

Translator: ranalytics.vn

Reactivity – Tái phân tích kết quả liên tục trong Shiny

Kết quả tái phân tích liên tục được thực hiện trong các hàm tái phân tích. Sử dụng các tham số trong các hàm sau để tránh gặp lỗi Operation not allowed without an active reactive context.



Tạo các giá trị thay đổi
(Create your own reactive values)

```
library(shiny)
ui <- fluidPage(
  textInput("a","","A")
)
server <-
  function(input,output){
    rv <- reactiveValues()
    rv$number <- 5
  }
shinyApp(ui, server)
```

Hàm *Input()

(Xem phần trước)

reactiveValues(…)

Mỗi hàm reactive tạo một giá trị được lưu tại **input$<inputId>**

reactiveValues() tạo một list của "reactive value" với dữ liệu đầu vào có thể thay đổi được

Hiển thị kết quả phân tích trên UI
(Render reactive output)

```
library(shiny)
ui <- fluidPage(
textInput("a","","A"),
textOutput("b")
)
server <-
function(input,output){
output$b <-
  renderText({
   input$a
  })
}
shinyApp(ui, server)
```

Hàm render*()

(Xem phần trước)

Xây dựng object để hiển thị trên UI. Hàm này sẽ được thực hiện lại khi các giá trị đầu vào của code thay đổi. Kết quả được lưu tại:

output$<outputId>

Tránh thay đổi dữ liệu đầu vào
(Prevent reactions)

```
library(shiny)
ui <- fluidPage(
  textInput("a","","A"),
  textOutput("b")
)
server <-
  function(input,output){
    output$b <-renderText({
      isolate({input$a})
    })
  }
shinyApp(ui, server)
```

isolate(expr)

Thực hiện kết quả câu lệnh và trả về kết quả độc lập với dữ liệu đầu vào

Thực hiện câu lệnh sau khi xác nhận giá trị dữ liệu đầu vào (Trigger arbitrary code)

```
library(shiny)

ui <- fluidPage(
  textInput("a","","A"),
  actionButton("go","Go")
)

server <-
  function(input,output){
    observeEvent(input$go,{
      print(input$a)
    })
  }

shinyApp(ui, server)
```

observeEvent(eventExpr, handlerExpr, event.env, event.quoted, handler.env, handler.quoted, labe, suspended, priority, domain, autoDestroy, ignoreNULL)

Thực hiện câu lệnh tại nhóm tham số thứ 2 khi dữ liệu đầu vào của nhóm tham số thứ nhất thay đổi. Tham khảo thêm **observe()**

Đơn bộ hóa quá trình thay đổi dữ liệu đầu vào
(Modularize reactions)

```
library(shiny)
ui <- fluidPage(
  textInput("a","","A"),
  textInput("z","","Z"),
  textOutput("b")
)
server <- function(input,output){
  re <- reactive({
  paste(input$a,input$z)})
  output$b <-renderText({
   re()
  })
}
shinyApp(ui, server)
```

reactive(x, env, quoted, label, domain)

Tạo hàm có các đặc điểm sau:

- Lưu tại bộ nhớ đệm để giảm khối lượng tính toán
- Có thể được trả ra từ các câu lệnh khác
- Thông báo điều kiện rằng buộc khi cần thiết

Gọi hàm với câu lệnh, VD: **re()**

Chậm hóa quá trình thay đổi dữ liệu đầu vào
(Delay reactions)

```
library(shiny)
ui <- fluidPage(
  textInput("a","","A"),
  actionButton("go","Go"),
  textOutput("b")
)
server <-
  function(input,output){
    re <-eventReactive(
      input$go,{input$a})
    output$b <-renderText({
      re()
    })
  }
shinyApp(ui, server)
```

eventReactive(eventExpr, valueExpr, event.env, event.quoted, value.env, value.quoted, label, domain, ignoreNULL)

Tạo hàm với câu lệnh trong tham số thứ 2 thay đổi chỉ khi giá trị trong tham số thứ nhất thay đổi

UI

Giao diện của App là một file HTML. Sử dụng các hàm trong Shiny để tạo HTML với R.

```
fluidPage(
  textInput("a","")
)
## <div class="container-fluid">
##   <div class="form-group shiny-input-container">
##     <label for="a"></label>
##     <input id="a" type="text"
##       class="form-control" value=""/>
##   </div>
## </div>
```

Kết quả trả ra tập tin HTML

Thêm các thành phần trong HTML với **tags**, tương tự như tag trong HTML, VD: **tags$a()**.

tags$a	tags$data	tags$h6	tags$nav	tags$span
tags$abbr	tags$datalist	tags$head	tags$noscript	tags$strong
tags$address	tags$dd	tags$header	tags$object	tags$style
tags$area	tags$del	tags$hgroup	tags$ol	tags$sub
tags$article	tags$details	tags$hr	tags$optgroup	tags$summary
tags$aside	tags$dfn	tags$HTML	tags$option	tags$sup
tags$audio	tags$div	tags$i	tags$output	tags$table
tags$b	tags$dl	tags$iframe	tags$p	tags$tbody
tags$base	tags$dt	tags$img	tags$param	tags$td
tags$bdi	tags$em	tags$input	tags$pre	tags$textarea
tags$bdo	tags$embed	tags$ins	tags$progress	tags$tfoot
tags$blockquote	tags$eventsource	tags$kbd	tags$q	tags$th
tags$body	tags$fieldset	tags$keygen	tags$ruby	tags$thead
tags$br	tags$figcaption	tags$label	tags$rp	tags$time
tags$button	tags$figure	tags$legend	tags$rt	tags$title
tags$canvas	tags$footer	tags$li	tags$s	tags$tr
tags$caption	tags$form	tags$link	tags$samp	tags$track
tags$cite	tags$h1	tags$mark	tags$script	tags$u
tags$code	tags$h2	tags$map	tags$section	tags$ul
tags$col	tags$h3	tags$menu	tags$select	tags$var
tags$colgroup	tags$h4	tags$meta	tags$small	tags$video
tags$command	tags$h5	tags$meter	tags$source	tags$wbr

Các tags thông dụng nhất thường có hàm rút gọn, không cần phải thêm tiền tố **tags$**

```
ui <- fluidPage(
  h1("Header 1"),
  hr(),
  br(),
  p(strong("bold")),
  p(em("italic")),
  p(code("code")),
  a(href="", "link"),
  HTML("<p>Raw html</p>")
)
```

Header 1

bold

italic

`code`

link

Raw html

Để thêm tập tin CSS, sử dụng hàm includeCSS(), hoặc

1. Đặt tập tin CSS trong thư mục **www**
2. Link đến tập tin CSS bằng câu lệnh

tags$head(tags$link(rel = "stylesheet", type = "text/css", href = "<file name>"))

Để thêm JavaScript, sử dụng hàm includeScript(), hoặc

1. Đặt tập tin trong thư mục **www**
2. Link đến tập tin bằng câu lệnh

tags$head(tags$script(src = "<file name>"))

Để thêm ảnh:

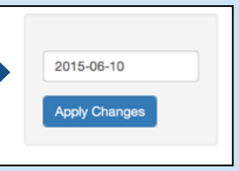
1. Đặt file ảnh vào thư mục **www**
2. Link đến file ảnh bằng câu lệnh

img(src="<file name>")

Cấu trúc giao diện

Kết hợp nhiều thành phần khác nhau trong cùng một giao diện người dùng với hàm panel (panel function), ví dụ:

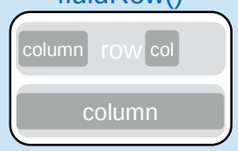
```
wellPanel(
 dateInput("a", ""),
 submitButton()
)
```



absolutePanel()	inputPanel()	tabPanel()
conditionalPanel()	mainPanel()	tabsetPanel()
fixedPanel()	navlistPanel()	titlePanel()
headerPanel()	sidebarPanel()	wellPanel()

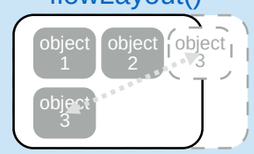
Sắp xếp cấu trúc và thành phần của giao diện trong ứng dụng. Mỗi thành phần là tham số của hàm cấu trúc giao diện (layout functions)

fluidRow()




```
ui <- fluidPage(
  fluidRow(column(width = 4),
    column(width = 2,  offset = 3)),
  fluidRow(column(width = 12))
)
```

flowLayout()

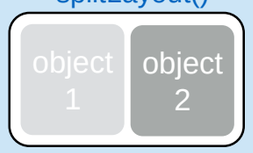



```
ui <- fluidPage(
  flowLayout( # object 1,
              # object 2,
              # object 3
  )
)
```

sidebarLayout()

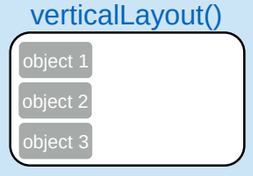

```
ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(),
    mainPanel()
  )
)
```

splitLayout()




```
ui <- fluidPage(
  splitLayout( # object 1,
               # object 2
  )
)
```

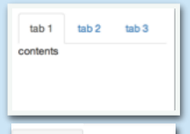
verticalLayout()




```
ui <- fluidPage(
  verticalLayout(# object 1,
                 # object 2,
                 # object 3
  )
)
```

Cấu trúc lớp **tabPanels**:

```
ui <- fluidPage( tabsetPanel(
    tabPanel("tab 1", "contents"),
    tabPanel("tab 2", "contents"),
    tabPanel("tab 3", "contents")))
```




```
ui <- fluidPage( navlistPanel(
    tabPanel("tab 1", "contents"),
    tabPanel("tab 2", "contents"),
    tabPanel("tab 3", "contents")))
```


```
ui <- navbarPage(title = "Page",
    tabPanel("tab 1", "contents"),
    tabPanel("tab 2", "contents"),
    tabPanel("tab 3", "contents"))
```

Xem thêm cheat sheets tại http://www.rstudio.com/resources/cheatsheets/

Translator: ranalytics.vn

Xem thêm tại shiny.rstudio.com/tutorial • shiny 0.12.0 • Cập nhật: 01/16