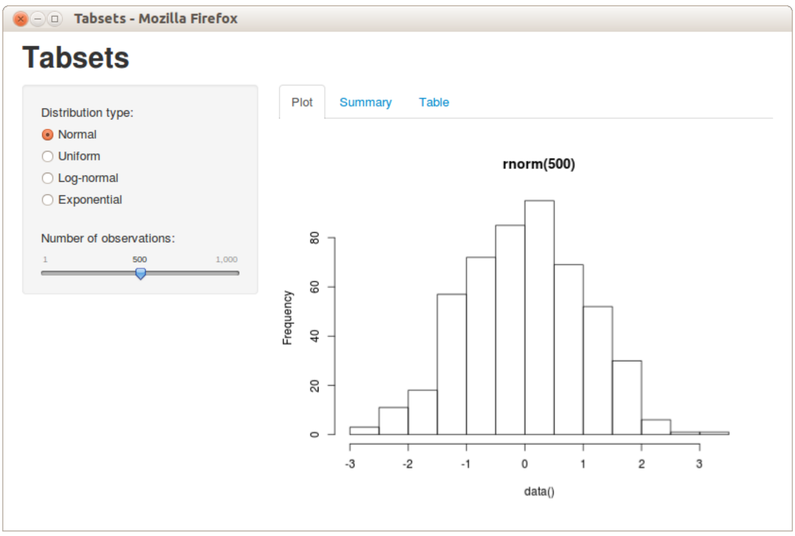
This tutorial is deprecated. Learn more about Shiny at our new location, shiny.rstudio.com.



The Tabsets application demonstrates using tabs to organize output. To run the example type:

```
> library(shiny)
> runExample("06_tabsets")
```

Tab Panels

Tabsets are created by calling the `tabsetPanel` function with a list of tabs created by the `tabPanel` function. Each tab panel is provided a list of output elements which are rendered vertically within the tab.

In this example we updated our Hello Shiny application to add a summary and table view of the data, each rendered on their own tab. Here is the revised source code for the user-interface:

ui.R

```
library(shiny)

# Define UI for random distribution application
shinyUI(pageWithSidebar(

  # Application title
  headerPanel("Tabsets"),

  # Sidebar with controls to select the random distribution type
  # and number of observations to generate. Note the use of the br()
  # element to introduce extra vertical spacing
  sidebarPanel(
    radioButtons("dist", "Distribution type:",
                 list("Normal" = "norm",
                      "Uniform" = "unif",
                      "Log-normal" = "lnorm",
                      "Exponential" = "exp")),
    br(),

    sliderInput("n",
                "Number of observations:",
                 value = 500,
                 min = 1,
                 max = 1000)
  ),

  # Show a tabset that includes a plot, summary, and table view
  # of the generated distribution
  mainPanel(
    tabsetPanel(
      tabPanel("Plot", plotOutput("plot")),
      tabPanel("Summary", verbatimTextOutput("summary")),
      tabPanel("Table", tableOutput("table"))
    )
  )
))
```

Tabs and Reactive Data

Introducing tabs into our user-interface underlines the importance of creating reactive expressions for shared data. In this example each tab provides its own view of the dataset. If the dataset is expensive to compute then our user-interface might be quite slow to render. The server script below demonstrates how to calculate the data once in a reactive expression and have the result be shared by all of the output tabs:

server.R

```
library(shiny)

# Define server logic for random distribution application
shinyServer(function(input, output) {

  # Reactive expression to generate the requested distribution. This is
  # called whenever the inputs change. The renderers defined
  # below then all use the value computed from this expression
  data <- reactive({
    dist <- switch(input$dist,
                   norm = rnorm,
                   unif = runif,
                   lnorm = rlnorm,
                   exp = rexp,
                   rnorm)

    dist(input$n)
  })

  # Generate a plot of the data. Also uses the inputs to build the
  # plot label. Note that the dependencies on both the inputs and
  # the 'data' reactive expression are both tracked, and all expressions
  # are called in the sequence implied by the dependency graph
  output$plot <- renderPlot({
    dist <- input$dist
    n <- input$n

    hist(data(),
         main=paste('r', dist, '(', n, ')', sep=''))
  })

  # Generate a summary of the data
  output$summary <- renderPrint({
    summary(data())
  })

  # Generate an HTML table view of the data
  output$table <- renderTable({
    data.frame(x=data())
  })
})
```

← Previous Next →