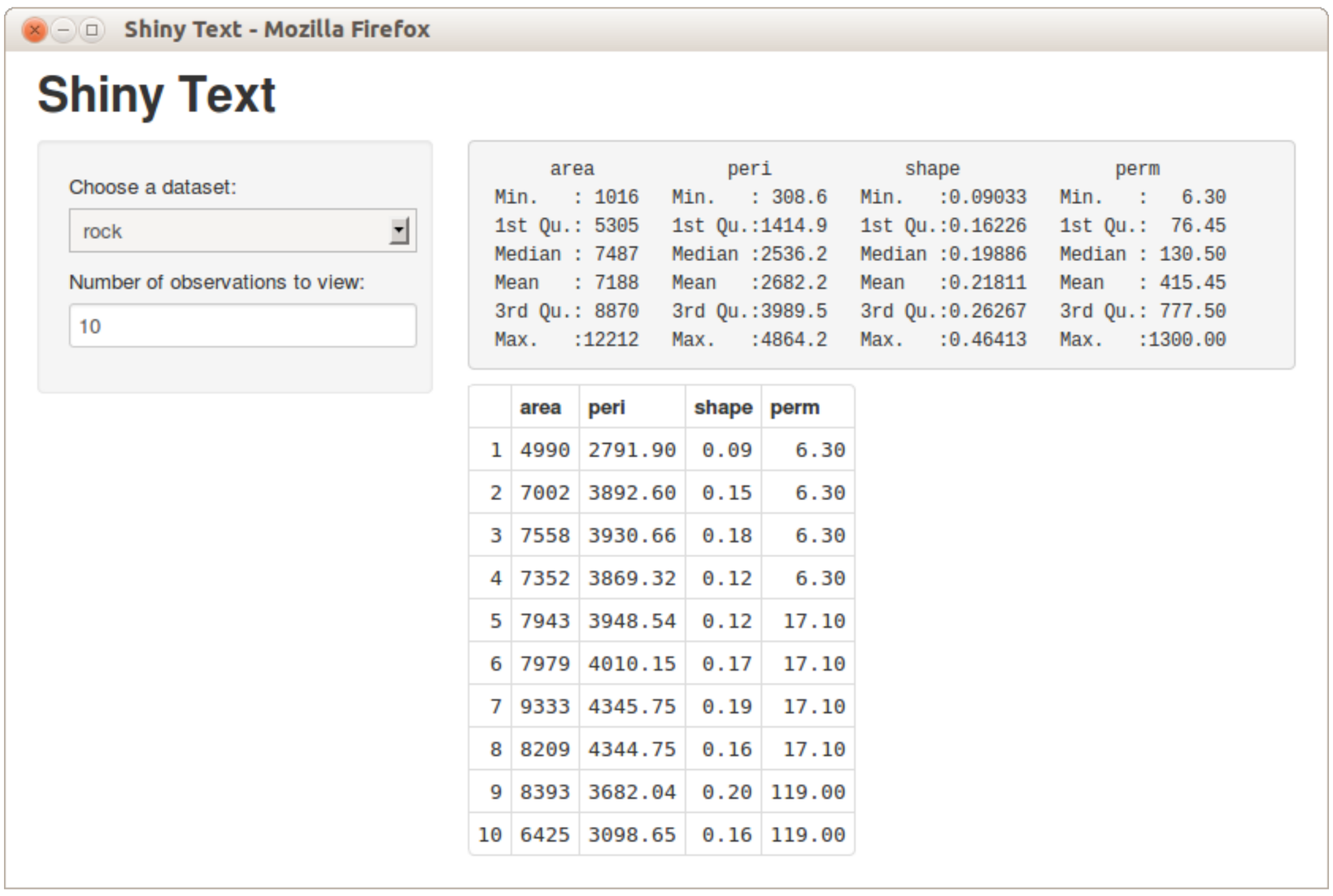
This tutorial is deprecated. Learn more about Shiny at our new location, shiny.rstudio.com.



The Shiny Text application demonstrates printing R objects directly, as well as displaying data frames using HTML tables. To run the example, type:

```
> library(shiny)
> runExample("02_text")
```

The first example had a single numeric input specified using a slider and a single plot output. This example has a bit more going on: two inputs and two types of textual output.

If you try changing the number of observations to another value, you'll see a demonstration of one of the most important attributes of Shiny applications: inputs and outputs are connected together "live" and changes are propagated immediately (like a spreadsheet). In this case, rather than the entire page being reloaded, just the table view is updated when the number of observations change.

Here is the user interface definition for the application. Notice in particular that the `sidebarPanel` and `mainPanel` functions are now called with two arguments (corresponding to the two inputs and two outputs displayed):

ui.R

```
library(shiny)

# Define UI for dataset viewer application
shinyUI(pageWithSidebar(

  # Application title
  headerPanel("Shiny Text"),

  # Sidebar with controls to select a dataset and specify the number
  # of observations to view
  sidebarPanel(
    selectInput("dataset", "Choose a dataset:",
                choices = c("rock", "pressure", "cars")),

    numericInput("obs", "Number of observations to view:", 10)
  ),

  # Show a summary of the dataset and an HTML table with the requested
  # number of observations
  mainPanel(
    verbatimTextOutput("summary"),

    tableOutput("view")
  )
))
```

The server side of the application has also gotten a bit more complicated. Now we create:

- A reactive expression to return the dataset corresponding to the user choice
- Two other rendering expressions (`renderPrint` and `renderTable`) that return the `output$summary` and `output$view` values

These expressions work similarly to the `renderPlot` expression used in the first example: by declaring a rendering expression you tell Shiny that it should only be executed when its dependencies change. In this case that's either one of the user input values (`input$dataset` or `input$n`).

server.R

```
library(shiny)
library(datasets)

# Define server logic required to summarize and view the selected dataset
shinyServer(function(input, output) {

  # Return the requested dataset
  datasetInput <- reactive({
    switch(input$dataset,
           "rock" = rock,
           "pressure" = pressure,
           "cars" = cars)
  })

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

  # Show the first "n" observations
  output$view <- renderTable({
    head(datasetInput(), n = input$obs)
  })
})
```

We've introduced more use of reactive expressions but haven't really explained how they work yet. The next example will start with this one as a baseline and expand significantly on how reactive expressions work in Shiny.

← Previous Next →