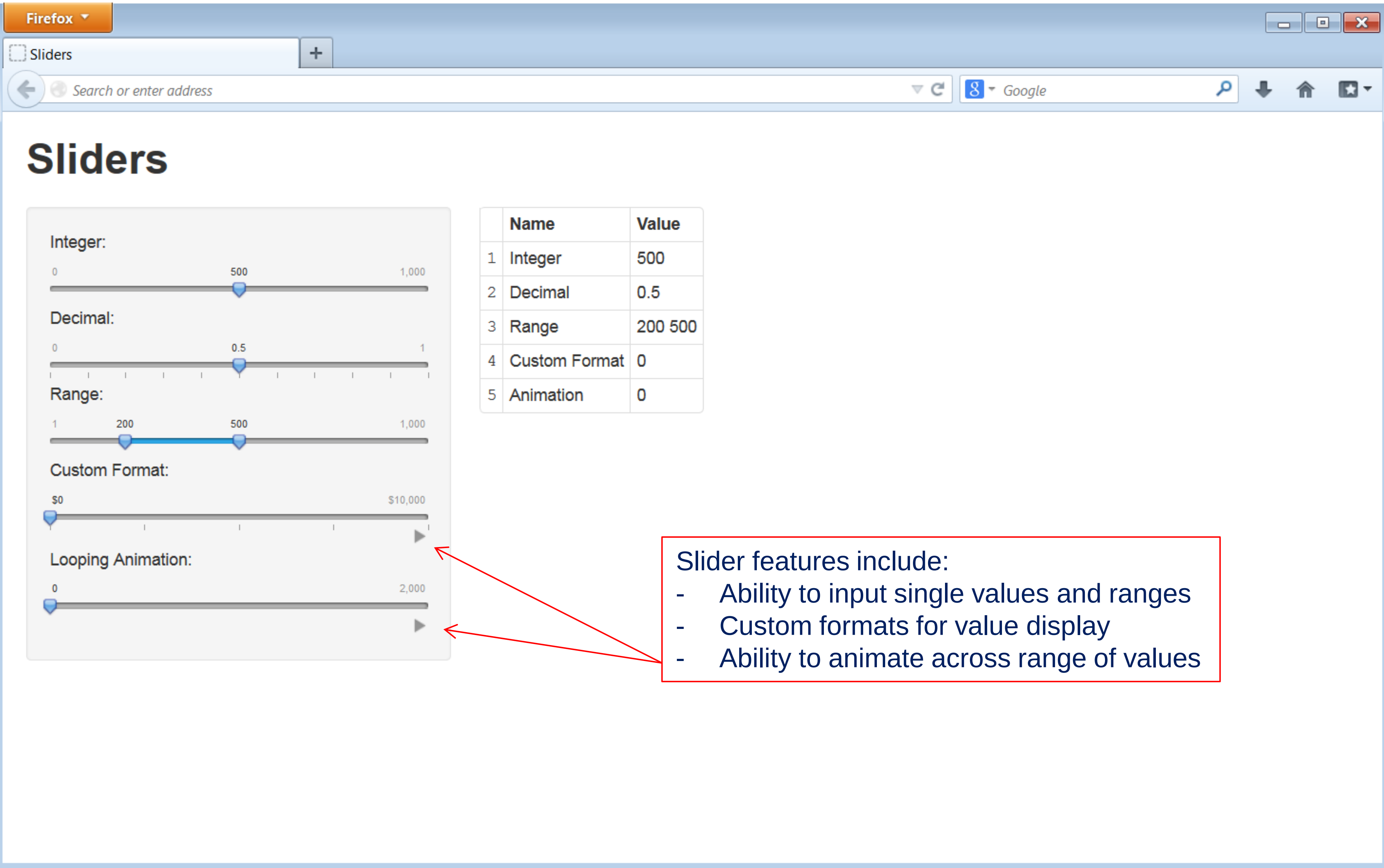
Session 3 Shiny Examples

“Sliders” Example



Sliders

Integer:

Decimal:

Range:

Custom Format:

Looping Animation:

	Name	Value
1	Integer	500
2	Decimal	0.5
3	Range	200 500
4	Custom Format	0
5	Animation	0

Slider features include:
- Ability to input single values and ranges
- Custom formats for value display
- Ability to animate across range of values

“Sliders” Example ui.R

```
# Sliders Example ui.R
library(shiny)

# Define UI for slider demo application
shinyUI(pageWithSidebar(

 #  Application title
 headerPanel("Sliders"),

 # Sidebar with sliders that demonstrate various available options
 sidebarPanel(
  # Simple integer interval
  sliderInput("integer", "Integer:",
            min=0, max=1000, value=500),

  # Decimal interval with step value
  sliderInput("decimal", "Decimal:",
            min = 0, max = 1, value = 0.5, step= 0.1),

  # Specification of range within an interval
  sliderInput("range", "Range:",
            min = 1, max = 1000, value = c(200,500)),

  # Provide a custom currency format for value display, with basic animation
  sliderInput("format", "Custom Format:",
            min = 0, max = 10000, value = 0, step = 2500,
            format="$#,##0", locale="us", animate=TRUE),

  # Animation with custom interval (in ms) to control speed, plus looping
  sliderInput("animation", "Looping Animation:", 1, 2000, 1, step = 10,
            animate=animationOptions(interval=300, loop=T))
 ),

 # Show a table summarizing the values entered
 mainPanel(
  tableOutput("values")
 )
))
```

integer

decimal

range

currency

animation

Slider controls created by calling sliderInput() function

Custom format (currency)

Animation sequences

"Sliders" Example server.R

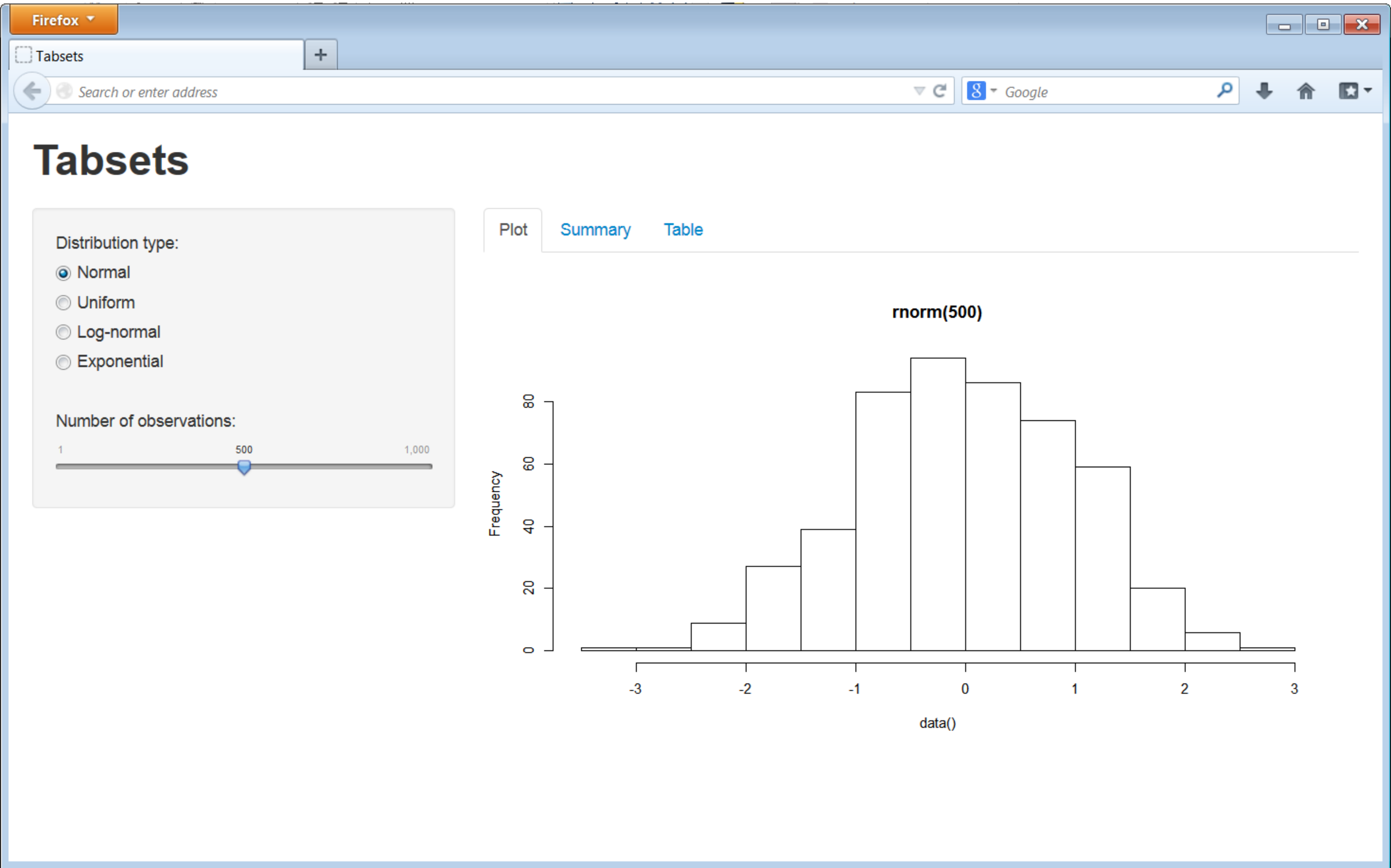
Creates a data frame with all input values

Renders data frame as HTML table

Sliders Example server.R

```
library(shiny)

# Define server logic for slider examples
shinyServer(function(input, output) {

  # Reactive expression to compose a data frame containing all
of the values
  sliderValues <- reactive({

    # Compose data frame
    data.frame(
      Name = c("Integer",
               "Decimal",
               "Range",
               "Custom Format",
               "Animation"),
      Value = as.character(c(input$integer,
                             input$decimal,
                             paste(input$range, collapse=' '),
                             input$format,
                             input$animation)),
      stringsAsFactors=FALSE)
  })

  # Show the values using an HTML table
  output$values <- renderTable({
    sliderValues()
  })
})
```

"Tabsets" Example: Plot Tab



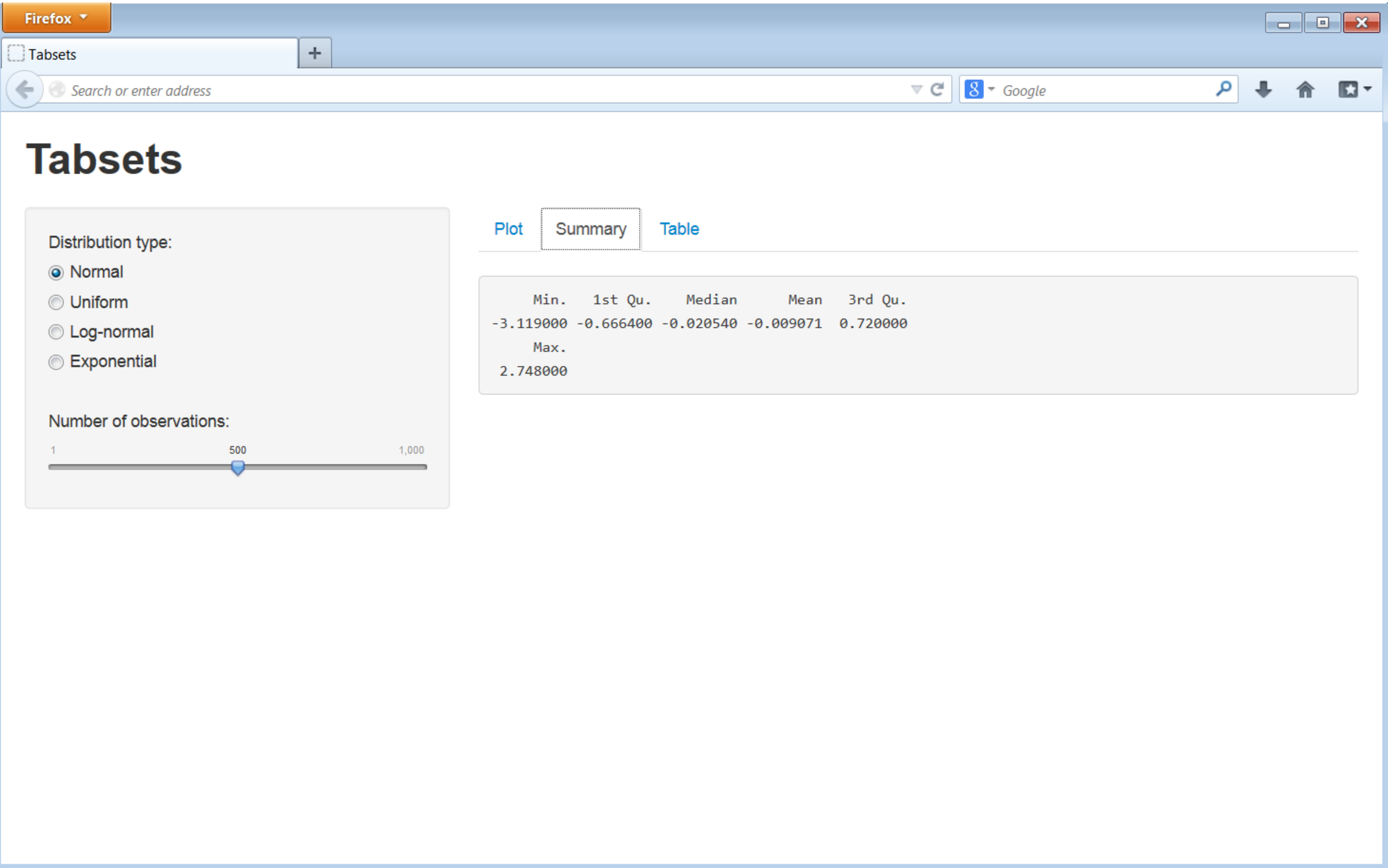
Tabsets

Distribution type:

- [✓ Normal]
- Uniform
- Log-normal
- Exponential

Number of observations: 500

“Tabsets” Example: Summary Tab



Tabsets

Distribution type:

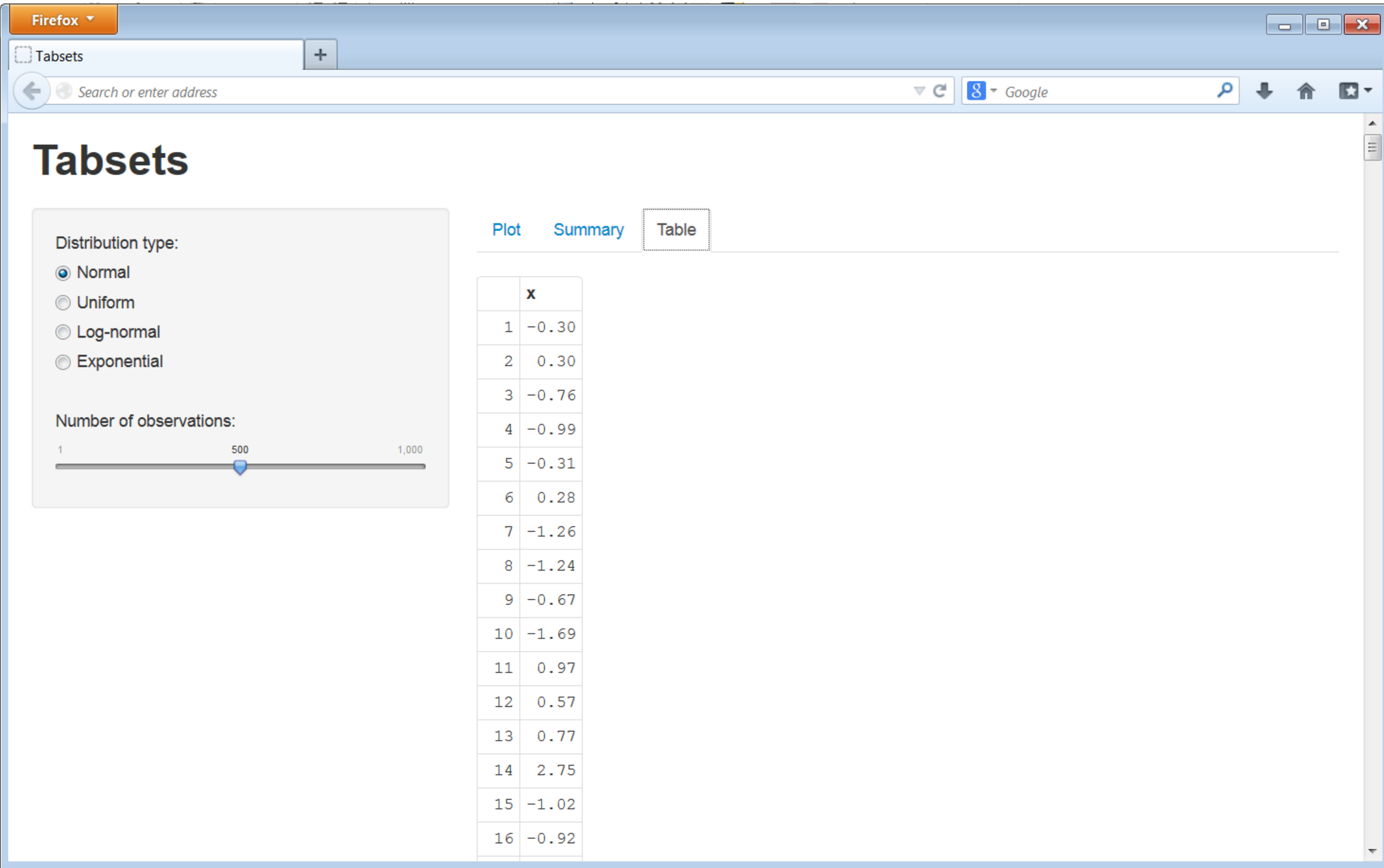
- [✓ Normal]
- Uniform
- Log-normal
- Exponential

Number of observations: 500 (slider 1 to 1,000)

Plot | Summary | Table

```
     Min.   1st Qu.    Median      Mean   3rd Qu.
-3.119000 -0.666400 -0.020540 -0.009071  0.720000
     Max.
 2.748000
```

“Tabsets” Example: Table Tab



Tabsets

Distribution type:
- [✓ Normal]
- Uniform
- Log-normal
- Exponential

Number of observations: 500 (range 1 to 1,000)

	x
1	-0.30
2	0.30
3	-0.76
4	-0.99
5	-0.31
6	0.28
7	-1.26
8	-1.24
9	-0.67
10	-1.69
11	0.97
12	0.57
13	0.77
14	2.75
15	-1.02
16	-0.92

“Tabsets” Example: ui.R

```
# Tabsets Example 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"))
    )
  )
))
```

Tabsets are created by calling tabsetPanel() function

with list of tabs created by tabPanel() function

Each tab panel is provided a list of outout lelements which are rendered vertically

"Tabsets" Example: server.R

Using tabs underlines the importance of creating reactive expressions for shared data.

Each tab provides own view of dataset.

'data' calculated once reactively.

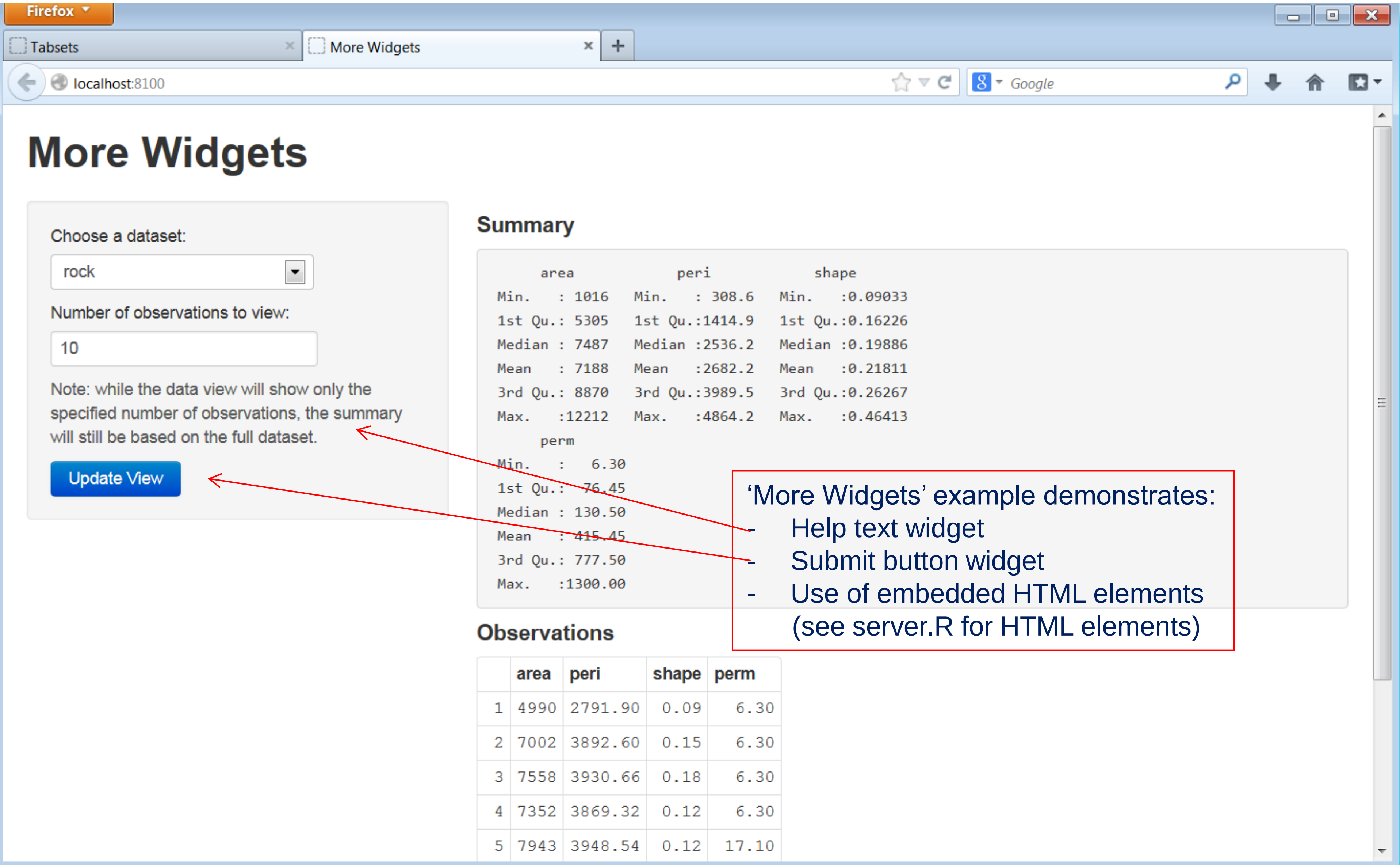
'data' calculated once reactively.

'data' calculated once reactively.

```
# Tabsets Example 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())
  })
})
```

So we use a reactive expression to calculate the data once and have the result shared by all of the output tabs.

“More Widgets” Example



More Widgets

Choose a dataset:

rock

Number of observations to view:

10

Note: while the data view will show only the specified number of observations, the summary will still be based on the full dataset.

Update View

Summary

```
      area           peri            shape        
 Min.   : 1016   Min.   : 308.6   Min.   :0.09033  
 1st Qu.: 5305   1st Qu.:1414.9   1st Qu.:0.16226  
 Median : 7487   Median :2536.2   Median :0.19886  
 Mean   : 7188   Mean   :2682.2   Mean   :0.21811  
 3rd Qu.: 8870   3rd Qu.:3989.5   3rd Qu.:0.26267  
 Max.   :12212   Max.   :4864.2   Max.   :0.46413  
      perm        
 Min.   :   6.30  
 1st Qu.:  76.45  
 Median : 130.50  
 Mean   : 415.45  
 3rd Qu.: 777.50  
 Max.   :1300.00  
```

Observations

	area	peri	shape	perm
1	4990	2791.90	0.09	6.30
2	7002	3892.60	0.15	6.30
3	7558	3930.66	0.18	6.30
4	7352	3869.32	0.12	6.30
5	7943	3948.54	0.12	17.10

‘More Widgets’ example demonstrates:
- Help text widget
- Submit button widget
- Use of embedded HTML elements (see server.R for HTML elements)

More Widgets Example ui.R

Added helpText() control to provide additional clarifying text alongside our input controls.

Added submitButton() control to indicate we don’t want live connection between inputs and outputs. User must click button to update the output.

Added h4 elements (heading level 4) into output pane.

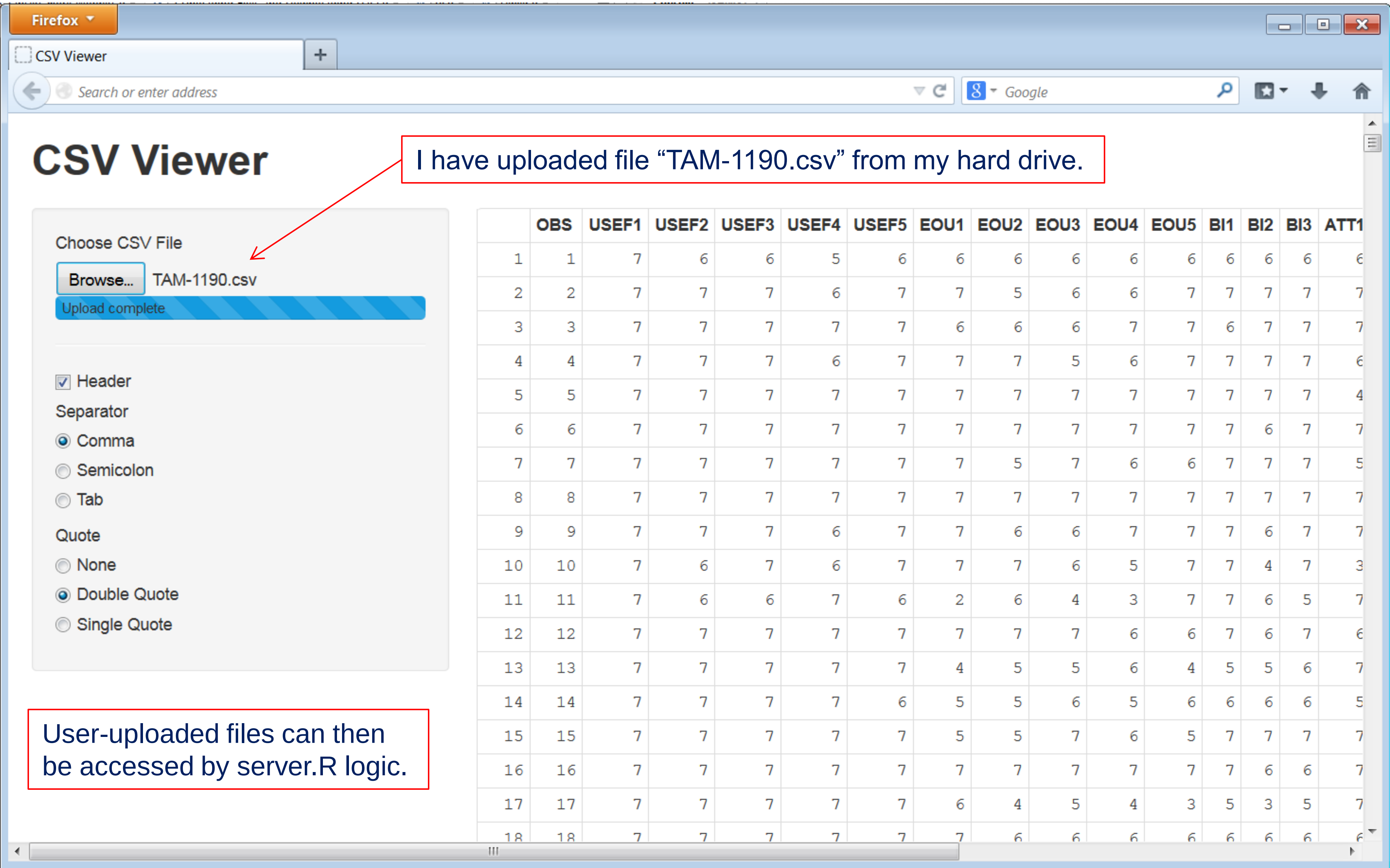
```
# More Widgets Example ui.R
library(shiny)

# Define UI for dataset viewer application
shinyUI(pageWithSidebar(

  # Application title.
  headerPanel("More Widgets"),

  # Sidebar with controls to select a dataset and specify the number
  # of observations to view. The helpText function is also used to
  # include clarifying text. Most notably, the inclusion of a
  # submitButton defers the rendering of output until the user
  # explicitly clicks the button (rather than doing it immediately
  # when inputs change). This is useful if the computations required
  # to render output are inordinately time-consuming.
  sidebarPanel(
    selectInput("dataset", "Choose a dataset:",
                choices = c("rock", "pressure", "cars")),

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

    helpText("Note: while the data view will show only the specified",
             "number of observations, the summary will still be based",
             "on the full dataset."),

    submitButton("Update View")
  ),

  # Show a summary of the dataset and an HTML table with the requested
  # number of observations. Note the use of the h4 function to provide
  # an additional header above each output section.
  mainPanel(
    h4("Summary"),
    verbatimTextOutput("summary"),

    h4("Observations"),
    tableOutput("view")
  )
))
```

More Widgets Example server.R

All changes to original ‘Shiny Text’ application were to the ui.R file, server.R is unchanged.

```
# More Widgets Example server.R
# All changes in this example were to the user-interface

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)
  })
})
```

“Uploading Files” Example



CSV Viewer

I have uploaded file “TAM-1190.csv” from my hard drive.

Choose CSV File

Browse... TAM-1190.csv

Upload complete

[✓] Header

Separator

[✓ Comma] Semicolon Tab

Quote

None [✓ Double Quote] Single Quote

	OBS	USEF1	USEF2	USEF3	USEF4	USEF5	EOU1	EOU2	EOU3	EOU4	EOU5	BI1	BI2	BI3	ATT1
1	1	7	6	6	5	6	6	6	6	6	6	6	6	6	6
2	2	7	7	7	6	7	7	5	6	6	7	7	7	7	7
3	3	7	7	7	7	7	6	6	6	7	7	6	7	7	7
4	4	7	7	7	6	7	7	7	5	6	7	7	7	7	6
5	5	7	7	7	7	7	7	7	7	7	7	7	7	7	4
6	6	7	7	7	7	7	7	7	7	7	7	7	6	7	7
7	7	7	7	7	7	7	7	5	7	6	6	7	7	7	5
8	8	7	7	7	7	7	7	7	7	7	7	7	7	7	7
9	9	7	7	7	6	7	7	6	6	7	7	7	6	7	7
10	10	7	6	7	6	7	7	7	6	5	7	7	4	7	3
11	11	7	6	6	7	6	2	6	4	3	7	7	6	5	7
12	12	7	7	7	7	7	7	7	7	6	6	7	6	7	6
13	13	7	7	7	7	7	4	5	5	6	4	5	5	6	7
14	14	7	7	7	7	6	5	5	6	5	6	6	6	6	5
15	15	7	7	7	7	7	5	5	7	6	5	7	7	7	7
16	16	7	7	7	7	7	7	7	7	7	7	7	6	6	7
17	17	7	7	7	7	7	6	4	5	4	3	5	3	5	7
18	18	7	7	7	7	7	7	6	6	6	6	6	6	6	6

User-uploaded files can then be accessed by server.R logic.

Uploading Files Example ui.R

File upload controls created by calling fileInput() function.

Then can access data by input$name-of-variable

fileInput() function accepts a ‘multiple’ parameter which can be set to TRUE to allow selection of multiple files, and an ‘accept’ parameter can be used to give user clues as to types of files application expects.

Shiny limits file upload sizes to 5MB by default, can modify to 30MB (or to larger sizes) using options(shiny.maxRequestSize=30*1024^2)

Feature doesn’t work in all browsers: Internet Explorer 9 or earlier.

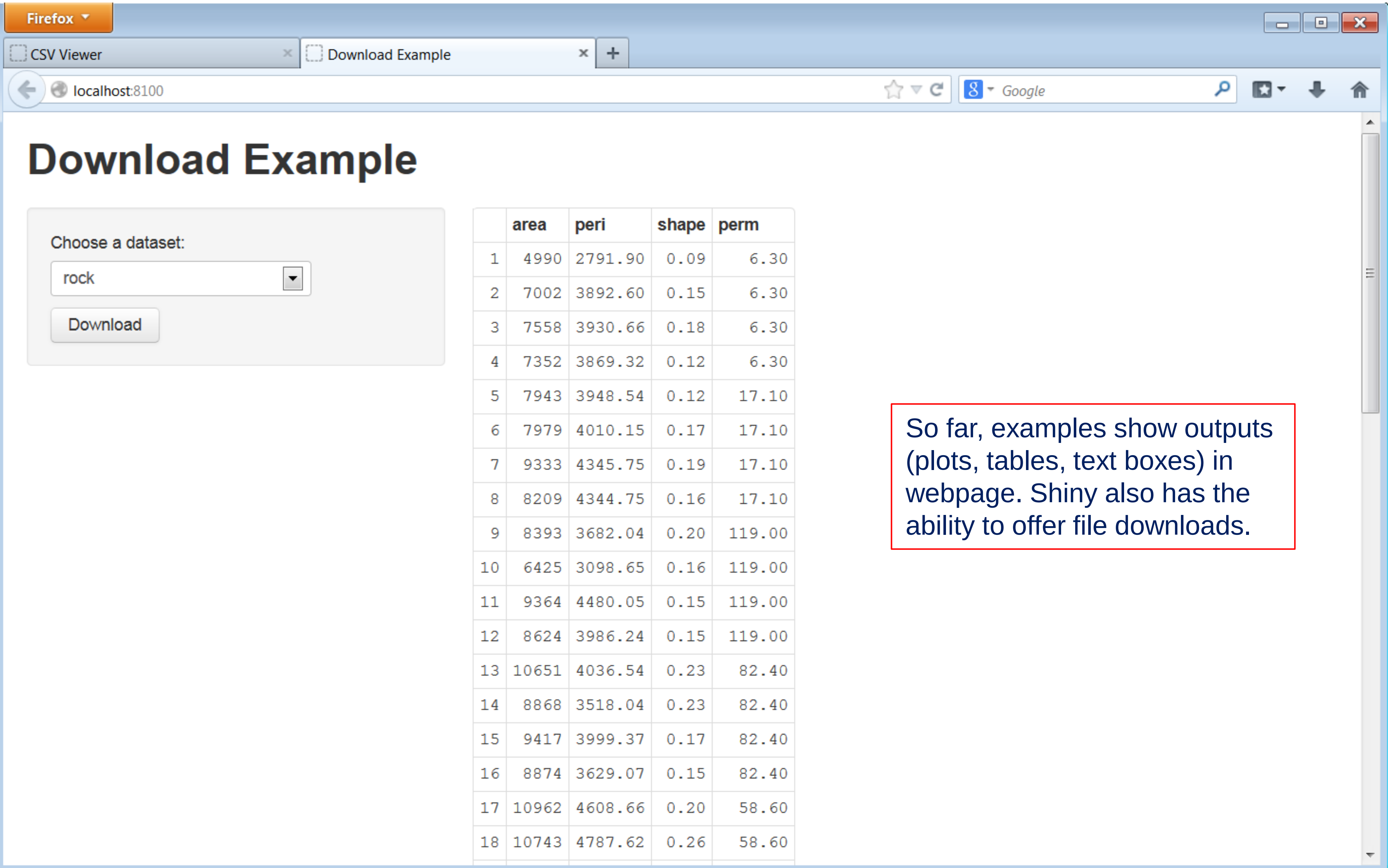
```
# Uploading Files Example ui.R
library(shiny)

shinyUI(pageWithSidebar(
  headerPanel("CSV Viewer"),
  sidebarPanel(
    fileInput('file1', 'Choose CSV File',
              accept=c('text/csv',
                       'text/comma-separated-values,
                       text/plain')),
    tags$hr(),
    checkboxInput('header', 'Header', TRUE),
    radioButtons('sep', 'Separator',
                 c(Comma=',',
                   Semicolon=';',
                   Tab='\t'),
                 'Comma'),
    radioButtons('quote', 'Quote',
                 c(None='',
                   'Double Quote'='"',
                   'Single Quote'="'"),
                 'Double Quote')
  ),
  mainPanel(
    tableOutput('contents')
  )
))
```

Uploading Files Example server.R

```
# Uploading Files Example server.R
library(shiny)

shinyServer(function(input, output) {
  output$contents <- renderTable({

    # input$file1 will be NULL initially.
    # After the user selects and uploads a
    # file, it will be a data frame with
    # 'name', 'size', 'type', and 'datapath'
    # columns. The 'datapath' column will
    # contain the local filenames where the
    # data can be found.

    inFile <- input$file1

    if (is.null(inFile))
      return(NULL)

    read.csv(inFile$datapath,
header=input$header,
             sep=input$sep, quote=input$quote)
  })
})
```

Example receives a file and attempts to read it as comma-separated values with read.csv(), and then display the results in a table.

“Downloading Data” Example



Download Example

Choose a dataset: rock

Download

	area	peri	shape	perm
1	4990	2791.90	0.09	6.30
2	7002	3892.60	0.15	6.30
3	7558	3930.66	0.18	6.30
4	7352	3869.32	0.12	6.30
5	7943	3948.54	0.12	17.10
6	7979	4010.15	0.17	17.10
7	9333	4345.75	0.19	17.10
8	8209	4344.75	0.16	17.10
9	8393	3682.04	0.20	119.00
10	6425	3098.65	0.16	119.00
11	9364	4480.05	0.15	119.00
12	8624	3986.24	0.15	119.00
13	10651	4036.54	0.23	82.40
14	8868	3518.04	0.23	82.40
15	9417	3999.37	0.17	82.40
16	8874	3629.07	0.15	82.40
17	10962	4608.66	0.20	58.60
18	10743	4787.62	0.26	58.60

So far, examples show outputs (plots, tables, text boxes) in webpage. Shiny also has the ability to offer file downloads.

Download Data Example ui.R

```
# Downloading Data Example ui.R
library(shiny)

shinyUI(pageWithSidebar(
  headerPanel('Download Example'),
  sidebarPanel(
    selectInput("dataset", "Choose a dataset:",
                choices = c("rock", "pressure", "cars")),
    downloadButton('downloadData', 'Download')
  ),
  mainPanel(
    tableOutput('table')
  )
))
```

Define download using downloadHandler() function in server.R with either downloadButton() or a downloadLink() in the ui.R source file.

Download Data Example server.R

```
# Downloading Data Example server.R
library(shiny)

shinyServer(function(input, output) {
  datasetInput <- reactive({
    switch(input$dataset,
           "rock" = rock,
           "pressure" = pressure,
           "cars" = cars)
  })

  output$table <- renderTable({
    datasetInput()
  })

  output$downloadData <- downloadHandler(
    filename = function() { paste(input$dataset, '.csv', sep='') },
    content = function(file) {
      write.csv(datasetInput(), file)
    }
  )
})
```

Define download using downloadHandler() function in server.R with either downloadButton() or a downloadLink() in the UI

filename= argument provides default file save name

content= argument is function with single file name of the (as yet) non-existent temp file that will have contents written to.