First Steps to Shiny Apps

W. John Braun, UBC

Workshop - Calgary

December 4, 2019

First Steps to Shiny Apps

We will illustrate *Shiny apps* through a sequence of simple examples.

We begin with a bar plot app.

First, install the *shiny* package into R:

```
install.packages("shiny")
```

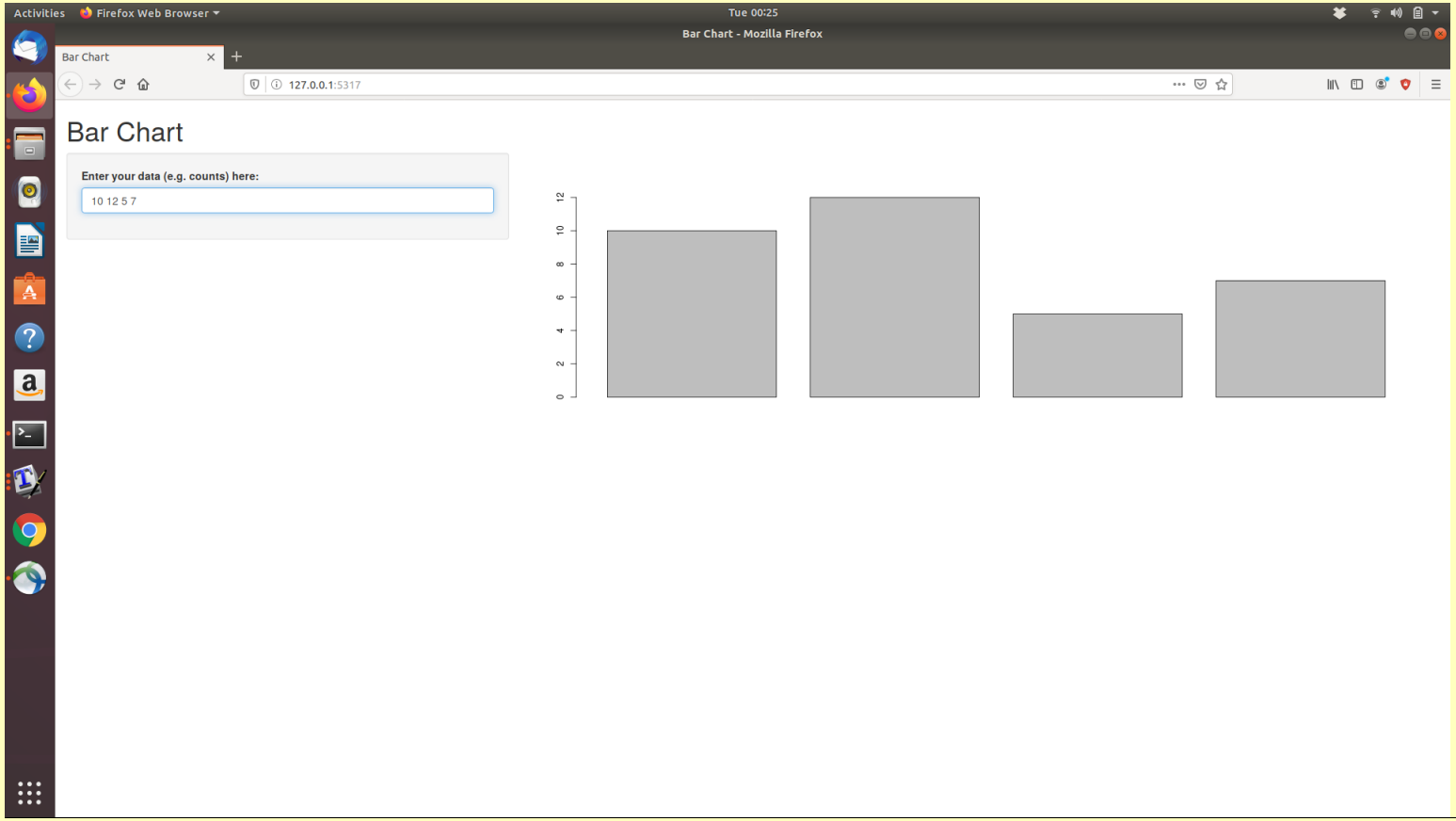
Next construct a directory called *barChart*.

To this directory, copy the two files: `server.R` and `ui.R`, whose contents are supplied later (or are downloaded from the relevant webpage.

Running the bar plot app

From within an R session, we can open the shiny app in a web browser by typing

```
runApp("barChart/")
```



The server and ui files

The web output is obtained by executing the commands in the `ui.R` and `server.R` files via some Javascript programs that are included with the *shiny* package.

The `server.R` file contains the R commands that the user will want to execute through the app.

The `ui.R` file contains commands for the user interface.

The server file

```
server <- function(input,output){
    output$main_plot <- renderPlot({
        data <- input$datavalues
        data <- as.numeric(strsplit(data, " ")[[1]])
        barplot(data)
    })
}
```

The bar plot app takes input data values, which we have enter into the user interface as character data, separated by single spaces.

The `server` communicates with the user interface by receiving input (data, usually) and transmitting output (in this case, a rendered plot).

The server file

In order to see the need for the awkward looking syntax used to convert the data into a form that `barplot()` will accept, we try out an example on artificial data:

```
data <- "3 5 7 11"
strsplit(data, " ")


## [[1]]
## [1] "3"  "5"  "7"  "11"
```

The output from `strsplit()` is a list, containing 1 element, indexed by `[[1]]`.

Before converting to the numeric data type, we need to access the list element, and then apply `as.numeric()`.

Compare

```
as.numeric(strsplit(data, " "))

## Error in eval(expr, envir, enclos):  (list) object
cannot be coerced to type 'double'
```

```
as.numeric(strsplit(data, " ")[[1]])

## [1]  3  5  7 11
```

The output from the second form is now ready for entry into `barplot()`.

The ui file

```
ui <- shinyUI(pageWithSidebar(
  headerPanel("Bar Chart"),
  sidebarPanel(
    textInput("datavalues",
        "Enter your data (e.g. counts) here:", "1")
  ),
  mainPanel(
    plotOutput(outputId='main_plot')
)
)
```

Focus on the `textInput` line. This is converting the user input to character data.

If we had asked for `numericInput` instead, our server file would be treating the input data differently.

Enhancing the output

By adjusting the server and ui files, we can produce plots with more features:

```
server <- function(input,output){
    output$main_plot <- renderPlot({
        data <- input$datavalues
        data <- as.numeric(strsplit(data, " ")[[1]])
        labels <- strsplit(input$labels, " ")[[1]]
        plotTitle <- input$title
        names(data) <- labels
        barplot(data)
        title(plotTitle)
    })
}
```

Here, we are including labels for the bars and a title. The server needs the information from the user interface.

Enhancing the output

```
ui <- shinyUI(pageWithSidebar(
  headerPanel("Bar Chart"),
  sidebarPanel(
    textInput("datavalues", "Enter your data (e.g. counts) here:", "1"),
    textInput("labels", "Enter the category labels here:", "A"),
    textInput("title", "Enter the plot title here:", "Bar Chart")
  ),
  mainPanel(
    plotOutput(outputId='main_plot')
)
)
)
```

Note the third argument of the `textInput()`: this is a default value or starting value which can be overwritten by the user.

The server file is:

```
server <-
function(input,output){
    output$main_plot <- renderPlot({
        datax <- input$xdatavalues
        x <- as.numeric(strsplit(datax, " ")[[1]])
        datay <- input$ydatavalues
        y <- as.numeric(strsplit(datay, " ")[[1]])
        ylabel <- strsplit(input$labels, " ")[[1]]
        plotTitle <- input$title
        plot(y ~ x, las = 1, ylab=ylabel)
        title(plotTitle)
        lines(y ~ x)
    })
}
```

A scatterplot app

The ui file is:

```
ui <- shinyUI(pageWithSidebar(
  headerPanel("Scatterplot"),
  sidebarPanel(
    textInput("xdatavalues", "Enter your data (x) here:", "1"),
    textInput("ydatavalues", "Enter your data (y) here:", "1"),
    textInput("labels", "Enter the y-axis label here:", "A"),
    textInput("title", "Enter the plot title here:", "Scatterplot")
  ),
  mainPanel(
    plotOutput(outputId='main_plot')
)
)
)
```

Note that the *las* parameter controls the orientation of the *axis labels*.

A simple example using reactivity

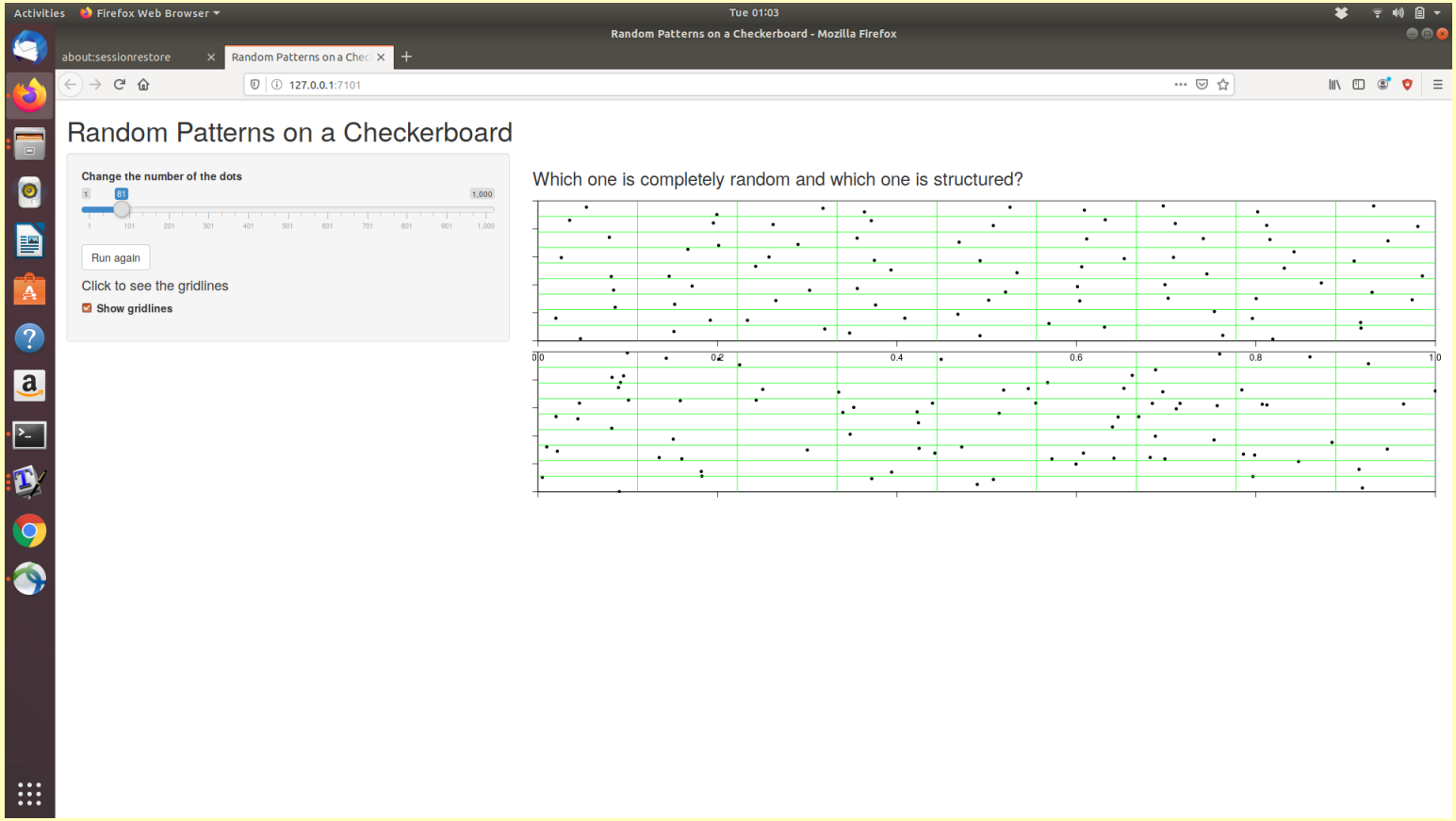
One feature of shiny apps is the ability to *cache* information and update the cache only when necessary.

We illustrate this feature with a very simple app that tests ones ability to discern complete randomness from more structured data.

Running the correct app - with reactivity

From within an R session, we can open the shiny app in a web browser by typing

```
runApp("check/")
```



Running the incorrect app - without reactivity

From within an R session, we can open the shiny app in a web browser by typing

```
runApp("check1/")
```