

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

GETTING STARTED

[Welcome](#)
[Hello Shiny](#)
[Shiny Text](#)
[Reactivity](#)

BUILDING AN APP

[UI & Server](#)
[Inputs & Outputs](#)
[Run & Debug](#)

TOOLING UP

[Sliders](#)
[Tabsets](#)
[DataTables](#)
[More Widgets](#)
[Uploading Files](#)
[Downloading Data](#)
[HTML UI](#)
[Dynamic UI](#)

ADVANCED SHINY

[Scoping](#)
[Client Data](#)
[Sending Images](#)

UNDERSTANDING REACTIVITY

[Reactivity Overview](#)
[Execution Scheduling](#)
[Isolation](#)

DEPLOYING AND SHARING APPS

[Deploying Over the Web](#)
[Sharing Apps to Run Locally](#)

EXTENDING SHINY

[Building Inputs](#)
[Building Outputs](#)

Inputs & Outputs

Adding Inputs to the Sidebar

The application we'll be building uses the `mtcars` data from the `R datasets` package, and allows users to see a box-plot that explores the relationship between miles-per-gallon (MPG) and three other variables (Cylinders, Transmission, and Gears).

We want to provide a way to select which variable to plot MPG against as well as provide an option to include or exclude outliers from the plot. To do this we'll add two elements to the sidebar, a `selectInput` to specify the variable and a `checkboxInput` to control display of outliers. Our user-interface definition looks like this after adding these elements:

ui.R

```
library(shiny)

# Define UI for miles per gallon application
shinyUI(pageWithSidebar(

  # Application title
  headerPanel("Miles Per Gallon"),

  # Sidebar with controls to select the variable to plot against mpg
  # and to specify whether outliers should be included
  sidebarPanel(
    selectInput("variable", "Variable:",
               list("Cylinders" = "cyl",
                    "Transmission" = "am",
                    "Gears" = "gear")),

    checkboxInput("outliers", "Show outliers", FALSE)
  ),

  mainPanel()
))
```

If you run the application again after making these changes you'll see the two user-inputs we defined displayed within the sidebar:



Creating the Server Script

Next we need to define the server-side of the application which will accept inputs and compute outputs. Our `server.R` file is shown below, and illustrates some important concepts:

- Accessing input using slots on the `input` object and generating output by assigning to slots on the `output` object.
- Initializing data at startup that can be accessed throughout the lifetime of the application.
- Using a reactive expression to compute a value shared by more than one output.

The basic task of a Shiny server script is to define the relationship between inputs and outputs. Our script does this by accessing inputs to perform computations and by assigning reactive expressions to output slots.

Here is the source code for the full server script (the inline comments explain the implementation techniques in more detail):

server.R

```
library(shiny)
library(datasets)

# We tweak the "am" field to have nicer factor labels. Since this doesn't
# rely on any user inputs we can do this once at startup and then use the
# value throughout the lifetime of the application
mpgData <- mtcars
mpgData$am <- factor(mpgData$am, labels = c("Automatic", "Manual"))

# Define server logic required to plot various variables against mpg
shinyServer(function(input, output) {

  # Compute the formula text in a reactive expression since it is
  # shared by the output$caption and output$mpgPlot expressions
  formulaText <- reactive({
    paste("mpg ~", input$variable)
  })

  # Return the formula text for printing as a caption
  output$caption <- renderText({
    formulaText()
  })

  # Generate a plot of the requested variable against mpg and only
  # include outliers if requested
  output$mpgPlot <- renderPlot({
    boxplot(as.formula(formulaText()),
            data = mpgData,
            outline = input$outliers)
  })
})
```

The use of `renderText` and `renderPlot` to generate output (rather than just assigning values directly) is what makes the application reactive. These reactive wrappers return special expressions that are only re-executed when their dependencies change. This behavior is what enables Shiny to automatically update output whenever input changes.

Displaying Outputs

The server script assigned two output values: `output$caption` and `output$mpgPlot`. To update our user interface to display the output we need to add some elements to the main UI panel.

In the updated user-interface definition below you can see that we've added the caption as an `h3` element and filled in its value using the `textOutput` function, and also rendered the plot by calling the `plotOutput` function:

ui.R

```
library(shiny)

# Define UI for miles per gallon application
shinyUI(pageWithSidebar(

  # Application title
  headerPanel("Miles Per Gallon"),

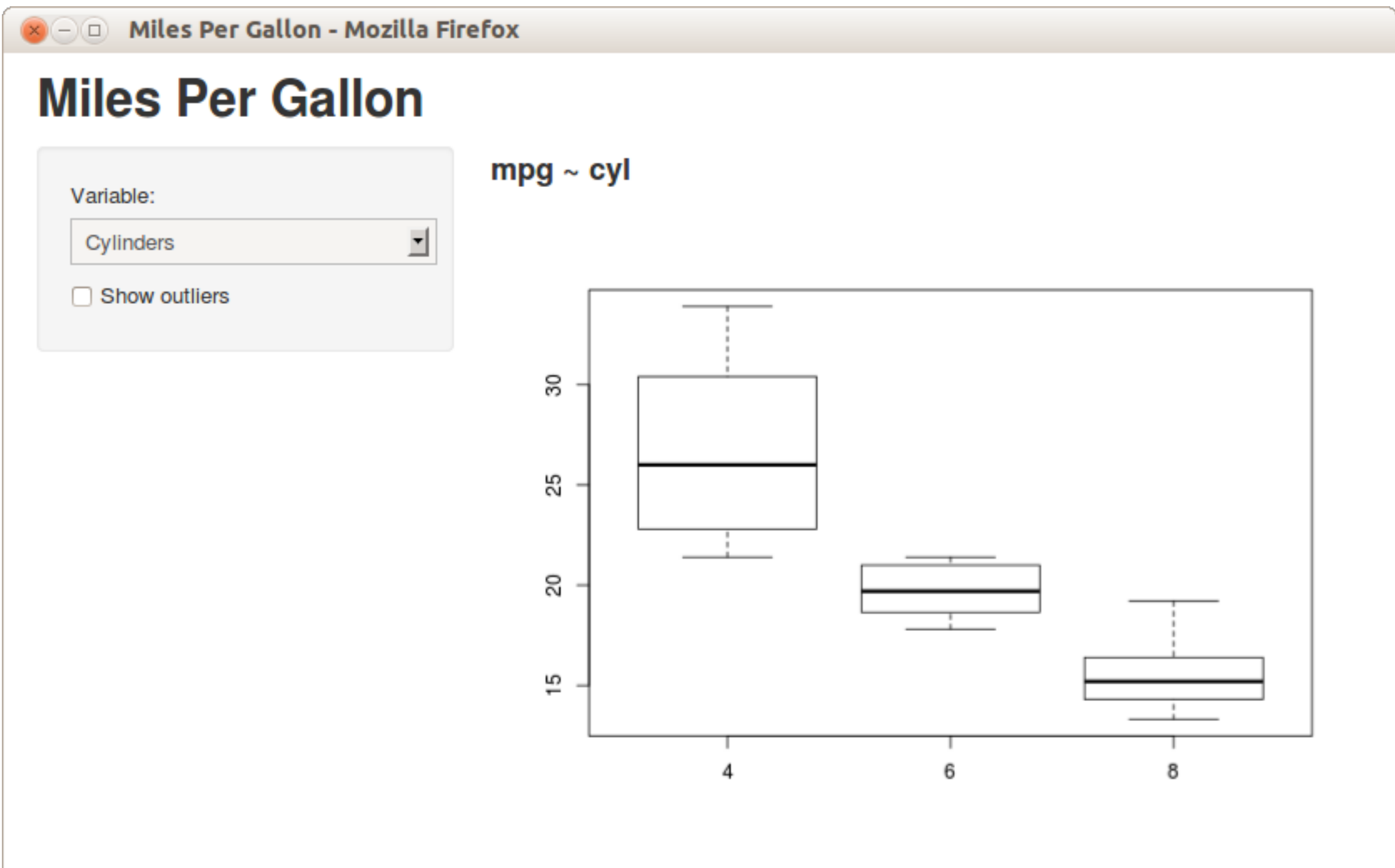
  # Sidebar with controls to select the variable to plot against mpg
  # and to specify whether outliers should be included
  sidebarPanel(
    selectInput("variable", "Variable:",
               list("Cylinders" = "cyl",
                    "Transmission" = "am",
                    "Gears" = "gear")),

    checkboxInput("outliers", "Show outliers", FALSE)
  ),

  # Show the caption and plot of the requested variable against mpg
  mainPanel(
    h3(textOutput("caption")),

    plotOutput("mpgPlot")
  )
))
```

Running the application now shows it in its final form including inputs and dynamically updating outputs:



Now that we've got a simple application running we'll probably want to make some changes. The next topic covers the basic cycle of editing, running, and debugging Shiny applications.

[← Previous](#)[Next →](#)