1 List of Shiny Examples

This is a collection of Shiny examples.

- To run the examples locally, you can install the shiny package in R, and use the function runGithub(). For example, to run the example 001-hello:

  ```
  if (!require('shiny')) install.packages("shiny")
  shiny::runGitHub("shiny-examples", "rstudio", subdir = "001-hello")
  ```

  Or you can clone or download this repository, and use run

  `shiny::runApp("001-hello").`

- You can see them in action on *http://demo.shinyapps.io/example-name* where *example-name* is the directory name of an example here, e.g. *http://demo.shinyapps.io/001-hello*

1.1 `EX01` - Hello World Example

This small Shiny application demonstrates `Shiny`'s automatic UI updates. Move the Number of bins slider and notice how the `renderPlot` expression is automatically re-evaluated when its dependant, `input$bins`, changes, causing a histogram with a new number of bins to be rendered.

1.2 `EX02` - Text Example

This example demonstrates output of raw text from `R` using the `renderPrint` function in `server.R` and the `verbatimTextOutput` function in `ui.R`.
In this case, a textual summary of the data is shown using `R`'s built-in summary function.

1.3 `EX03` - Reactivity

This example demonstrates a core feature of Shiny: **reactivity**. In `server.R`, a reactive called `datasetInput` is declared.

Notice that the reactive expression depends on the input expression `input$dataset`, and that it's used by both the output expression `output$summary` and `output$view`.

Try changing the dataset (using *Choose a dataset*) while looking at the reactive and then at the outputs; you will see first the reactive and then its dependencies flash.

Notice also that the reactive expression doesn't just update whenever anything changes–only the inputs it depends on will trigger an update. Change the "Caption" field and notice how only the `output$caption` expression is re-evaluated; the reactive and its dependents are left alone.

1.4 `EX04` - mpg data set example

This example demonstrates the following concepts:

Global variables : The `mpgData` variable is declared outside the `shinyServer` function. This makes it available anywhere inside `shinyServer`. The code in `server.R` outside `shinyServer` is only run once when the app starts up, so it can't contain user input.

Reactive expressions : `formulaText` is a reactive expression. Note how it re-evaluates when the Variable field is changed, but not when the *Show Outliers* box is ticked.

1.5 `EX05` - sliders example

- This example demonstrates Shiny's versatile `sliderInput` widget.
- Slider inputs can be used to select single values, to select a continuous range of values, and even to animate over a range.

1.6 `EX06` - tabsets example

This example demonstrates the `tabsetPanel` and `tabPanel` widgets.

Notice that outputs that are not visible are not re-evaluated until they become visible. Try this:

1. Scroll to the bottom of `server.R`
2. Change the number of observations, and observe that only `output$plot` is evaluated.
3. Click the *Summary* tab, and observe that `output$summary` is evaluated.
4. Change the number of observations again, and observe that now only `output$summary` is evaluated.

1.7 EX07 - Widgets Example

This example demonstrates some additional widgets included in Shiny, such as `helpText` and `submitButton`. The latter is used to delay rendering output until the user explicitly requests it.

1.8 EX08 - HTML (short discussion only)

- Normally we use the built-in functions, such as `textInput()`, to generate the HTML UI in the R script `ui.R`.
- Actually shiny also works with a custom HTML page www/index.html. See the tutorial for more details.

1.9 EX19-MathJax

MathJax is an open source JavaScript display engine for mathematics that works in all browsers. MathJax downloads with web page content, scans the page content for equation markup, and typesets the math. Thus, MathJax requires no installation of software or extra fonts on the reader's system.

This allows MathJax to run in any browser with JavaScript support, including mobile devices.

- The function `withMathJax()` is a wrapper function to load the **MathJax** library in a shiny app.
- For static HTML content, we only need to call `withMathJax()` once.
- However, for dynamic UI output via `renderUI()` , we must wrap the content that contains math expressions in `withMathJax()` , because we have to call the MathJax function `MathJax.Hub.Typeset()` to render math manually, which is what `withMathJax()` does.