# Connecting UI Widgets with R Code in the Server

Justin Post

# Recap

- `app.R` file contains `ui`, `server`, and code to run the app
- UI can be built in many ways!
    - `bslib` functions give nice layouts and functionality (`page_sidebar()`, `cards()`, `value_box()`, etc.)
- Widgets (`*Input` functions), Text, HTML elements, etc. are added to the UI

# UI: More About Widgets

- Widgets all follow the same structure
  - `widgetName("inputId", label = "Title the user sees", ...)`
  - The `inputId` is how you access the inputs when creating plots, summaries, etc. in the server

# UI: Adding Elements

- Within **layout** functions add elements to UI separated by ,
  - Can add plain strings and formatted text (using HTML type functions)

| shiny function | HTML5 equivalent | creates |
|---|---|---|
| p | <p> | A paragraph of text |
| h1 | <h1> | A first level header |
| h2 | <h2> | A second level header |
| h3 | <h3> | A third level header |
| h4 | <h4> | A fourth level header |
| h5 | <h5> | A fifth level header |
| h6 | <h6> | A sixth level header |
| a | <a> | A hyper link |
| br | <br> | A line break (e.g. a blank line) |
| div |  | A division of text with a uniform style |
| span |  | An in-line division of text with a uniform style |
| pre | <pre> | Text 'as is' in a fixed width font |
| code | <code> | A formatted block of code |
| img | <img> | An image |
| strong |  | Bold text |
| em |  | Italicized text |
| HTML | | Directly passes a character string as HTML code |

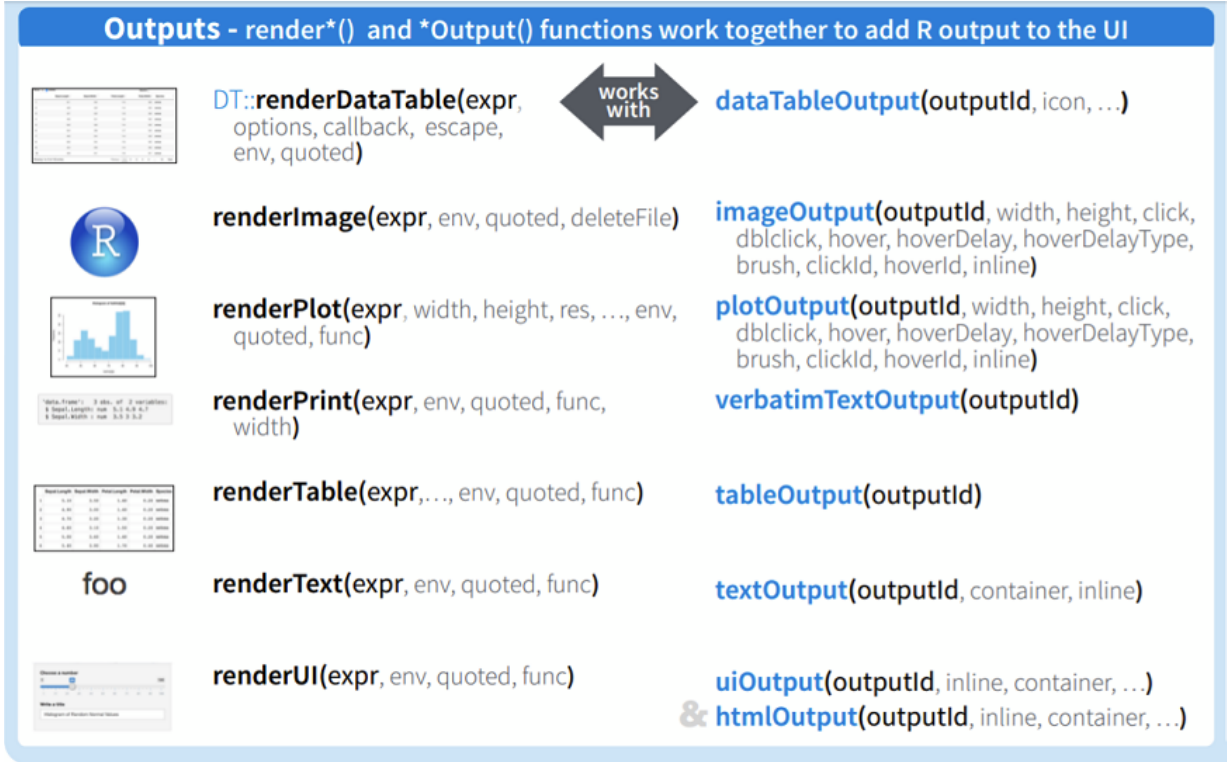
# Widget & Text Example

- Check code for kNN app (github site)
  - Note the separation of elements by , within a layout-style function!

```
ui <- fluidPage(
  pageWithSidebar(
    headerPanel('k-Nearest Neighbours Classification'),
    sidebarPanel(
      sliderInput('k', 'Select the Number of Nearest Neighbours', value = 25, min = 1, max = 150),
      checkboxInput('showN', label = "Show the neighbourhood for one point (click to select a point)"),
      a("App credit: https://github.com/schoonees/kNN", href = "https://github.com/schoonees/kNN")
    ),
    mainPanel(
      plotOutput('plot1', width = "600px", height = "600px",  click = "click_plot")
    )
  )
)
```

# Server: Creating Outputs

- Outputs can be created in the UI using `*Output` functions
- These correspond to a particular `render*` function in the server



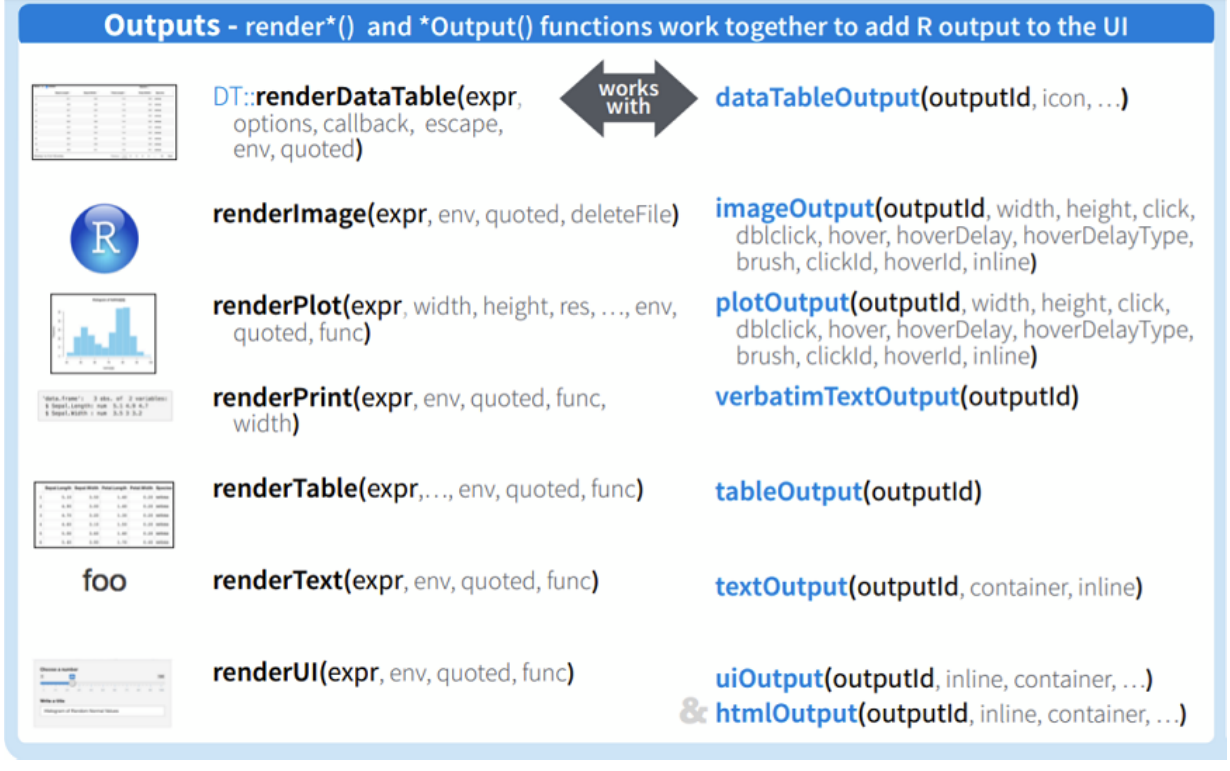
# render* Functions

- These define **reactive contexts** that allow you to use info from widgets (via `input$inputId`)

```
output$plot1 <- renderPlot({...
    ## Fit model
    fit <- knn(train = train,
               test = test,
               cl = trainclass,
               k = input$k,
               prob = TRUE)
    ...
    ## Plot create empty plot
    plot(train, asp = 1, type = "n", xlab = "x1", ylab = "x2",
         xlim = range(pts2), ylim = range(pts2),
         main =  paste0(input$k, "-Nearest Neighbours"))
    ...
```

# `render*` Functions

- These define **reactive contexts** that allow you to use info from widgets (via `input$inputId`)
- Each `render*` function tries to coerce the last code run to the appropriate type of output



# `render*` Functions

- These define **reactive contexts** that allow you to use info from widgets (via `input$inputId`)
- Each `render*` function tries to coerce the last code run to the appropriate type of output
- Corresponding `*Output` function goes in the UI

```
mainPanel(
  plotOutput('plot1'),
  textOutput('my_text') #goes with output$my_text <- renderText({...}) in server
)
```

# Back to the Tutorial!

- Read through the following pages of the Posit tutorial (**complete the `Your Turn` sections within these lessons** - no need to turn anything in, this is just to help you learn!)
    - Display reactive outputs
    - Use R scripts and data
    - Use reactive expressions
    - Share your apps