R Shiny Tutorial - Day 1 - **Tips & Tricks**

Indentation

Which one is easier to understand?

```
if(input$statistics == T && !is.null(data_subset())) {
pl <- pl +
stat_compare_means(
comparisons = pairwise_comp( unique(data_subset()$Species)),
method = 'wilcox.test',
label = 'p.signif')}

pl
})

output$violinplot <- renderPlot({
violinplot()
})

output$save_violinplot <- downloadHandler(filename = 'plot.pdf', content = function(file) {ggsave(file, plot=violinplot(), width

output$table_selected <- renderDataTable(selected(), extensions = 'Buttons', options = list(
pageLength = 10, scrollX = T, pagingType = 'simple_numbers', dom = 'Bfrtip', buttons = list(list(extend = 'csv', filename = 'sel
)
),
rownames = F,
server = F # necessary to download whole table
)
```

vs (same code)

```
    if(input$statistics == T && !is.null(data_subset())) {
      pl  <- pl +
        stat_compare_means(
          comparisons = pairwise_comp( unique(data_subset()$Species)),
          method      = 'wilcox.test',
          label       = 'p.signif'
        )
    }

    pl
  })

  output$violinplot <- renderPlot({
    violinplot()
  })

  output$save_violinplot <- downloadHandler(
    filename    = 'plot.pdf',
    content     = function(file) {
      ggsave(file, plot=violinplot(), width=297, height=210, unit='mm')
    }
  )

  output$table_selected <- renderDataTable(
    selected(),
    extensions  = 'Buttons',
```

Increase indentation for each level of nesting in R code to improve readability and clarity. This helps to easily distinguish different levels of code structure. **Select the code and press (Shift +) the Tab key** to indent multiple lines of code in RStudio. Also **use line breaks** often and try to keep the length of rows below the vertical line. Beautiful/clearly structured code makes it much easier to spot errors.

Using reactive()

Wrapping your code within reactive({ … }) enables automatic updates when dependencies change and helps to prevent code duplication. Remember to **use brackets after the name of a reactive expression to access its value**. Additionally, keep in mind that reactive expressions only function within reactive consumers.

```
  result <- reactive({
    if (input$operation == "divide") {
      req(input$num2 != 0)
    }

    switch(input$operation,
      "add"      = input$num1 + input$num2,
      "subtract" = input$num1 - input$num2,
      "multiply" = input$num1 * input$num2,
      "divide"   = input$num1 / input$num2
    )
  })
```

```
  output$result <- renderText({
    paste("Result:", result())
  })
```

Keep track of brackets

RStudio helps you identifying related opening and closing brackets by highlighting them when you put the cursor next to them. When saving, it even shows errors next to the line numbers. **Position opening and closing brackets carefully in your code**, use line breaks and an appropriate indentation style.

```
output$violinplot <- renderPlot({
  violinplot()
})
```

```
output$violinplot <- renderPlot({
  violinplot()
})))
```

unexpected token ')'

Understand the Layout

In a sidebarLayout, the input widgets usually go inside the sidebarPanel(), while the **output goes inside the mainpanel()**.

```
sidebarLayout(
  sidebarPanel(
    numericInput("num1", "Enter first number:", value = 0),
    numericInput("num2", "Enter second number:", value = 0),
    radioButtons("operation", "Choose operation:", choices = list(
      "add",
      "subtract",
      "multiply",
      "divide"
    ))
  ),

  mainPanel(
    textOutput("result")
  )
)
```

Check the parameters

For input widgets, the first parameter is always the ID. You can check the help for a function by executing ?function. Try to **run ?sliderInput at least once**.

Do not create copy pasta

Carefully check what you put into your R Shiny app. For the **slider input**, you **only need the two selected lines** (!) from the Shiny widget gallery. Also check the comments in the code (text after #).

server.R | ui.R — show with app

```
fluidPage(
  fluidRow(
    column(4,

      # Copy the line below to make a slider bar
      sliderInput("slider1", label = h3("Slider"), min = 0,
        max = 100, value = 50)
    ),
    column(4,

      # Copy the line below to make a slider range
      sliderInput("slider2", label = h3("Slider Range"), min = 0,
        max = 100, value = c(40, 60))
    )
  ),

  hr(),

  fluidRow(
    column(4, verbatimTextOutput("value")),
    column(4, verbatimTextOutput("range"))
  )

)
```

Check input/output IDs

Carefully check the IDs of inputs and outputs, that we use to connect the user interface to the server logic. **Upper/lower-case does matter**.

```
numericInput("num1", "Enter first number:", value = 0),
```

ID — label

Enter first number: 0

```
switch(input$operation,
  "add"      = input$num1 + input$num2,
  "subtract" = input$num1 - input$num2,
  "multiply" = input$num1 * input$num2,
  "divide"   = input$num1 / input$num2
)
```

```
output$result <- renderText({
  paste("Result:", result())
})
```

```
mainPanel(
  textOutput("result")
)
```