NC STATE UNIVERSITY
DEPARTMENT OF STATISTICS

Debugging & Useful Things

Justin Post

Other Shiny Stuff!

- Validating inputs and such
- Plotly
- Debugging!

Validating Inputs

- Often errors will show if computations are running or inputs are temporarily changed (Bad Shiny Example)
    - Can validate inputs/data!

```
validate(
  need(!is.null(sample_corr$corr_data), "Please select your variables, subset,
       and click the 'Get a Sample!' button.")
)
```

- Check this out here

Validating Data

- We also need to verify values supplied are of the right type
    - Easy to do with the `shinyalert` package!

```
observeEvent(input$submit_proportion, {
  if(!is.numeric(input$proportion)){
    shinyalert(title = "Oh no!", "You must supply a number between 0 and 1!", type = "error")
  }
```

- Check this out here

Loaders

- Sometimes a plot or computation will take a while to show
    - User may think an error has occurred and click away or reclick causing more delay...
    - Can add spinners and things via `shinycssloaders`

```
tabPanel(title = "Map View",
         shinycssloaders::withSpinner(
           leaflet::leafletOutput("map_plot"))
         )
```

- Check this out here

Plotly!

- Know how to plot with `ggplot2`
- Plots are not inherently interactive...
  - Install `plotly` package
  - Wrap any ggplot in `ggp
  - Change `renderPlot()` and `plotOutput()` functions to `renderPlotly()` and `plotlyOutput()`

```
...
 plotlyOutput("boot_graph")
...
output$boot_graph <- renderPlotly({
  g <- ggplot(my_plot_data, aes(x = phat)) +
  geom_histogram(bins = 50, fill = "black", aes(group = Quantile))
  ggplotly(g, tooltip = c("x", "group"))
```

- Check this out here

Debugging

- Much harder in shiny!
- Shiny debugging page
- Recommendations:
  - Get static working code, then transfer to shiny
  - Build app in small pieces, testing as you go

Basic Debugging

- Can use `observe({print(...)})`

```
observe({print(input$NI + 10)})
```

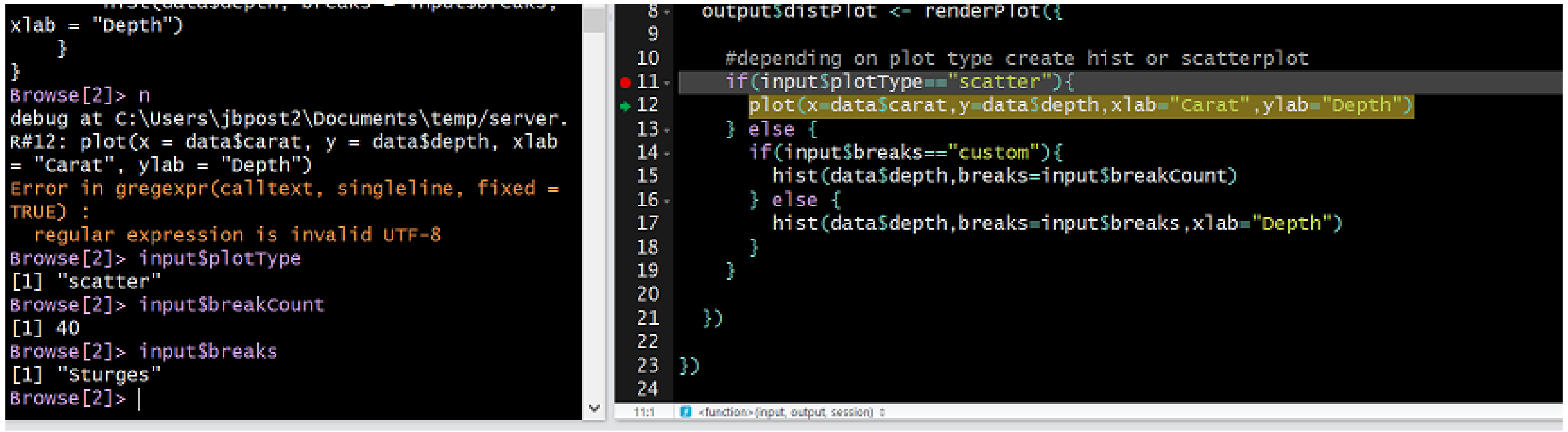
Debugging

Three major approaches:

1. Breakpoints - Pausing execution of your program
2. Tracing - Collecting information as your program runs
3. Error handling - Finding the source of errors (both on the client and server side) and ascertaining their cause.

Breakpoints

- Easiest method to debug!
  - Can be used in `server.r`
  - Click to the left of the line number



```
xlab = "Depth")
    }
}
Browse[2]> n
debug at C:\Users\jbpost2\Documents\temp/server.
R#12: plot(x = data$carat, y = data$depth, xlab
= "Carat", ylab = "Depth")
Error in gregexpr(calltext, singleline, fixed =
TRUE) :
  regular expression is invalid UTF-8
Browse[2]> input$plotType
[1] "scatter"
Browse[2]> input$breakCount
[1] 40
Browse[2]> input$breaks
[1] "Sturges"
Browse[2]> |
```

```
  output$distPlot <- renderPlot({

    #depending on plot type create hist or scatterplot
    if(input$plotType=="scatter"){
      plot(x=data$carat,y=data$depth,xlab="Carat",ylab="Depth")
    } else {
      if(input$breaks=="custom"){
        hist(data$depth,breaks=input$breakCount)
      } else {
        hist(data$depth,breaks=input$breaks,xlab="Depth")
      }
    }

  })

})
```

- Now can access values and step through program

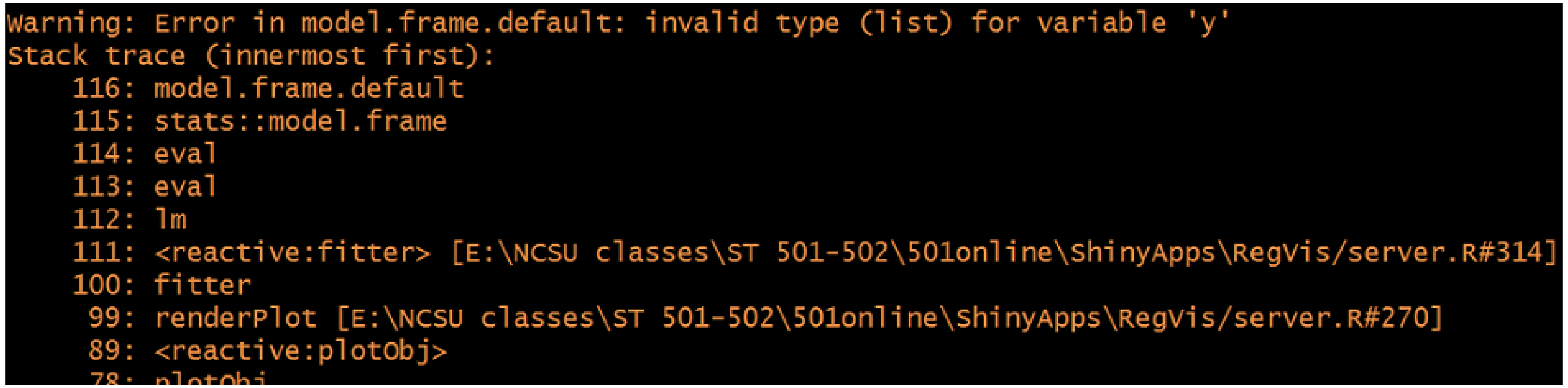
Dynamic Breakpoints

- You can add an actionButton() that when clicked calls `browser()`
    - This kicks you into a debugger with all current inputs!
- Can make Shiny enter the debugger when an error occurs by using the following statement:

```
options(shiny.error = browser)
```

Error Handling

- Check stack trace shiny returns



```
Warning: Error in model.frame.default: invalid type (list) for variable 'y'
Stack trace (innermost first):
    116: model.frame.default
    115: stats::model.frame
    114: eval
    113: eval
    112: lm
    111: <reactive:fitter> [E:\NCSU classes\ST 501-502\501online\ShinyApps\RegVis/server.R#314]
    100: fitter
     99: renderPlot [E:\NCSU classes\ST 501-502\501online\ShinyApps\RegVis/server.R#270]
     89: <reactive:plotObj>
     78: plotObj
```

Recap

- Validating inputs and such
- Plotly
- Debugging!