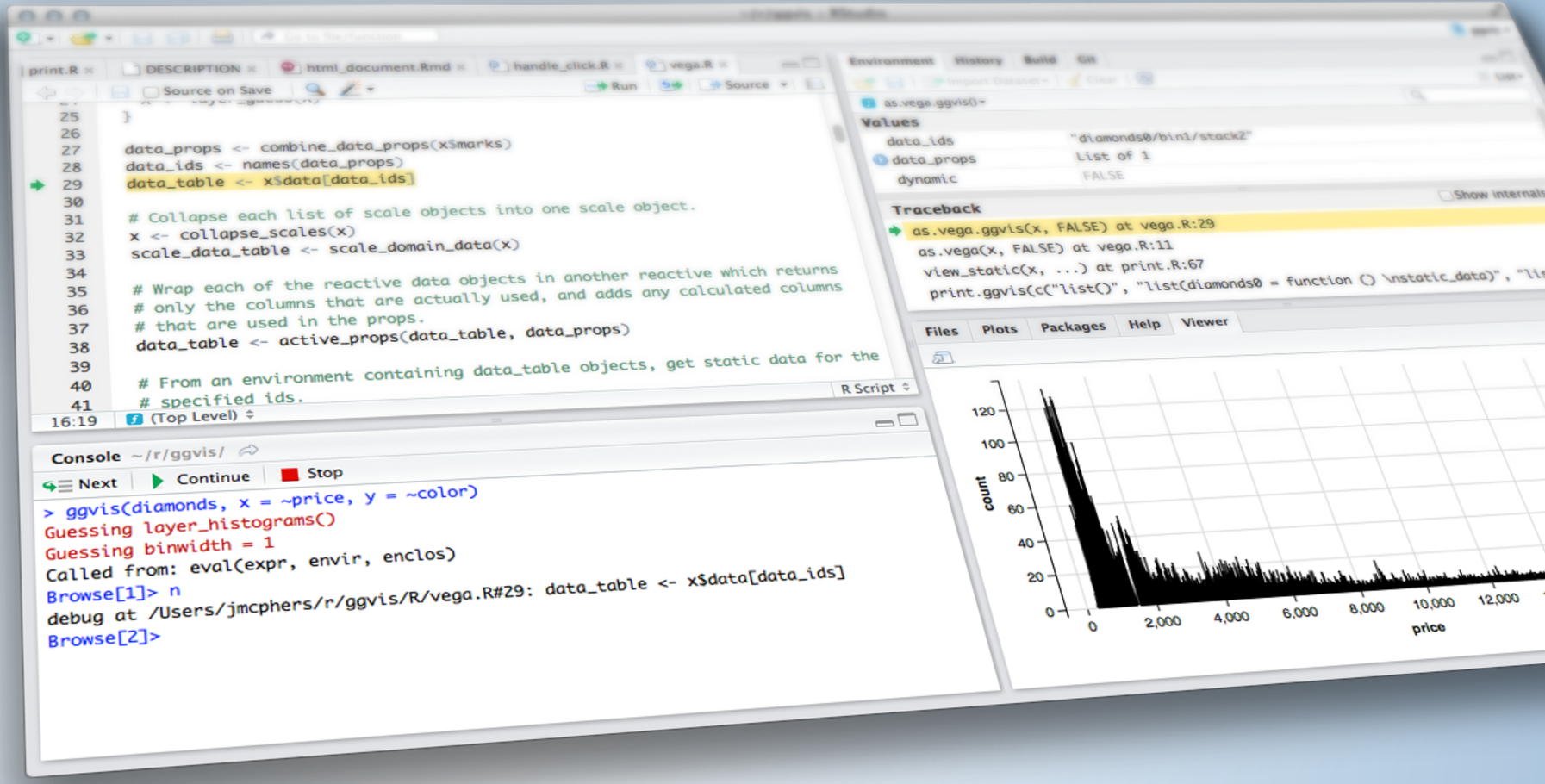


TROUBLESHOOTING SHINY

Shiny from RStudio™

OUTLINE

- Writing robust code
- Debugging tools at your disposal
- Techniques for debugging

Writing

robust code

WRITING ROBUST CODE

- Complexity is the problem; abstraction is the solution
  - Software programs are far too large to reason about in their entirety
  - Good programs are broken into fragments that you can reason about locally, and compose reliably
  - In other words, we break the program into simple fragments, and if we verify that each fragment is correct, then the whole program is correct
- **Are our fragments simple enough to understand?**
- **Do they compose reliably?**

UNDERSTANDABLE FRAGMENTS

- Indent your code! (Ctrl+I/Cmd+I)
- Extract out complicated processing logic (as opposed to UI logic) into top-level functions so you can test them separately
- Each function, reactive, observer, or module should be small, and do one thing
  - Function/reactive/observer bodies that don't fit on a single screen is a bad code smell
  - If you're having trouble giving something a meaningful name, maybe it's doing too much
- When you encounter unavoidable complexity, at least try to firewall the complexity behind as simple/straightforward an API as possible
  - Even if it's hard to verify if the scary piece itself is correct, it's still easy to verify that its callers are correct

RELIABLE COMPOSITION

- Prefer "pure functions"—functions without side effects. Much less likely to surprise you.
  - When you do need side effects, don't put them in surprising places. Consider following command-query separation—"asking a question should not change the answer"
- Reactive expressions must not have side effects
- Avoid observers and reactive values, where possible; use reactive expressions if you can help it
- Don't pass around environments and reactive values objects; this is similar to sharing global variables, it introduces hidden coupling
- For ease of reasoning, prefer: pure functional > reactive > imperative (observers)

Debugging tools

STANDARD R DEBUGGING TOOLS

- Tracing
  - `print()`/`cat()`/`str()`
  - `renderPrint` eats messages, must use `cat(file = stderr(), ...)`
  - Also consider shinyjs package's `logjs`, which puts messages in the browser's JavaScript console
- Debugger
  - Set breakpoints in RStudio
  - `browser()`
  - Conditionals: `if (!is.null(input$x)) browser()`

SHINY DEBUGGING TOOLS

- **Symptom:** Outputs or observers don't execute when expected, or execute too often
- Reactlog
  - Restart R process
  - Set `options(shiny.reactlog = TRUE)`
  - In the browser, Ctrl+F3 (or Cmd+F3)
- Showcase mode: DESCRIPTION file or `runApp(display.mode = "showcase")`

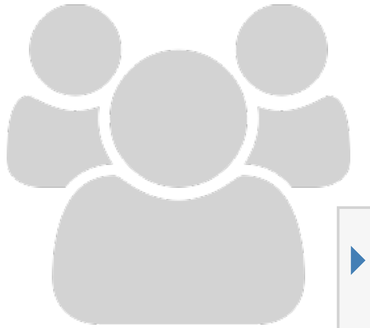
SHINY DEBUGGING TOOLS

- **Symptom:** Red error messages in the UI *or* session abruptly terminates
- This means an R error has occured
- Look in R console for stack traces
  - By default, Shiny hides "internal" stack traces. Use `options(shiny.fullstacktrace = TRUE)` if necessary to show.
- Newer versions of Shiny/Shiny Server "sanitize" errors, for security reasons (every error message is displayed as "An error has occurred")
  - See sanitizing errors article for more details, including how to view the real errors

SHINY DEBUGGING TOOLS

- **Symptom:** Server logic seems OK, but unexpected/broken/missing results in browser
- Check browser's JavaScript console for errors
- Listen in on conversation between client and server
  - `options(shiny.trace=TRUE)` logs messages in the R console
  - Use Chrome's Network tab to show individual websocket messages

Your turn



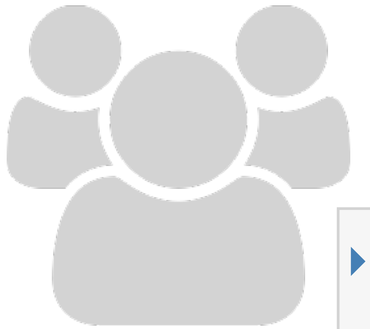
EXERCISE

- Open **`movies_broken_01.R`**. It is broken in a not-very-subtle way. See if you can find and fix the bug.
- Continue on for **`movies_broken_02.R`** through **`movies_broken_04.R`**.

10m 00s

SOLUTION

- **`movies_broken_01.R`**: Missing commas, as explained in the R console
- **`movies_broken_02.R`**: ggplot call was missing "+"
- **`movies_broken_03.R`**: Reactive was not being called with "()"
- **`movies_broken_04.R`**: Output ID was not consistent between UI and server



EXERCISE

- Open **`movies_broken_05.R`**. It is broken in a subtle way. See if you can find and fix the bug.
  - Check the box for one other type of movie and see how the text about number of movies changes.
  - Choose a low sample size and get a new sample.
  - Choose a high sample size and get a new sample.

3m 00s

SOLUTION

- `movies_broken_05.R`: With a low sample size there are not necessarily at least one of each type of movie, hence the way the paste function is written you get length coercion.

```
uiOutput(outputId = "n"),

output$n <- renderUI({
  types <- movies_sample()$title_type %>%
  factor(levels = input$selected_type)
  counts <- table(types)

  HTML(paste("There are",
             counts, input$selected_type,
             "movies in this dataset. <br>"))
  })
```

Common errors

COMMON ERRORS

- `"Object of type 'closure' is not subsettable"`
  - You forgot to use () when retrieving a value from a reactive expression `plot(userData)` should be `plot(userData())`

COMMON ERRORS

- `"Unexpected symbol"`
  `"Argument xxx is missing, with no default"`
  - Missing or extra comma in UI. Sometimes Shiny will realize this and give you a hint, or use RStudio editor margin diagnostics.

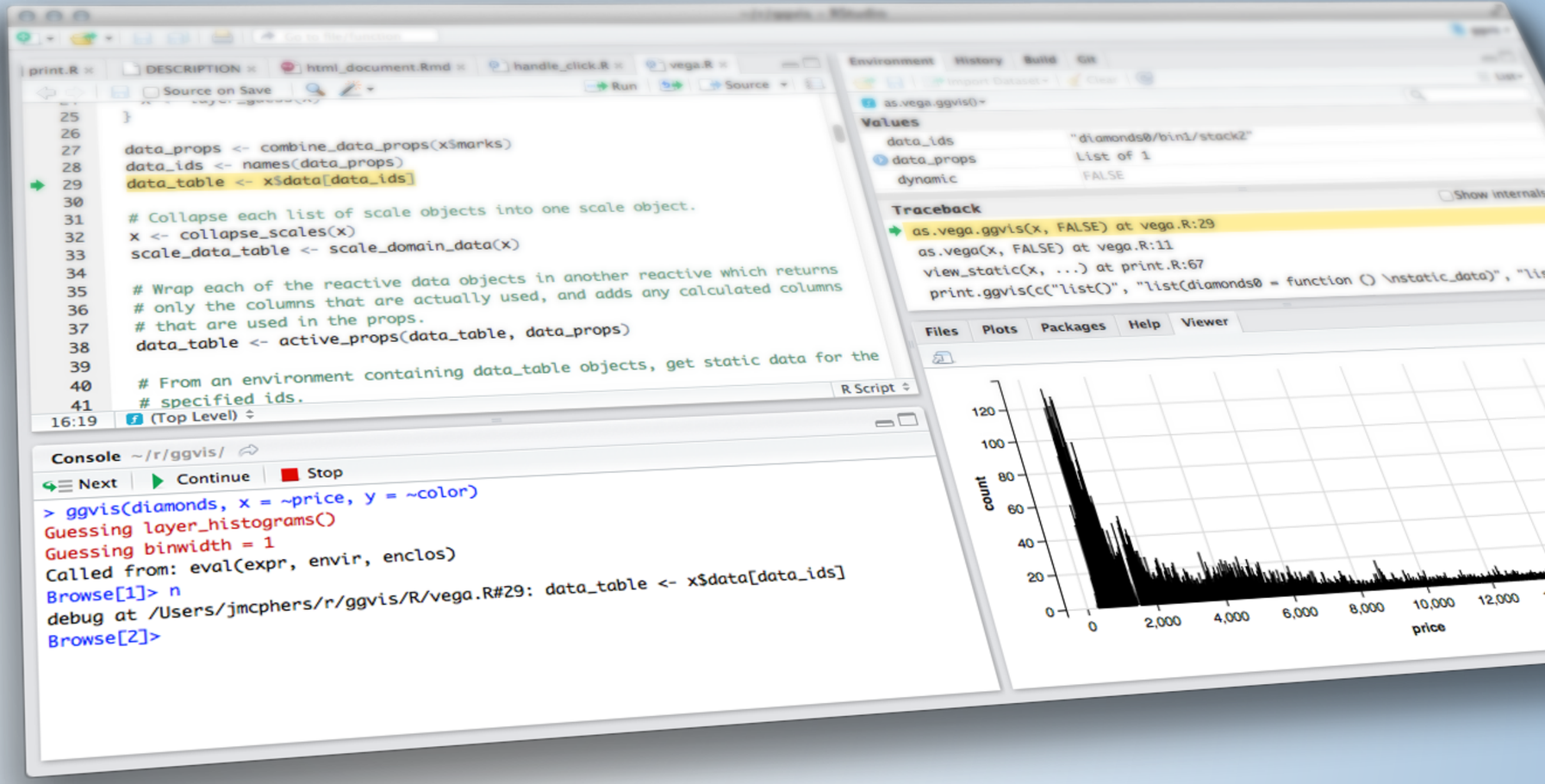
COMMON ERRORS

- `"Operation not allowed without an active reactive context. (You tried to do something that can only be done from inside a reactive expression or observer.)"`
  - Tried to access an input or reactive expression from directly inside the server function. You must use a reactive expression or observer instead.
    - Or if you really *only* care about the value of that input at the time that the session starts, then use **`isolate()`**.

More resources

RESOURCES

- Debugging article on shiny.rstudio.com
- Jonathan McPherson's talk at Shiny Developer conference (video, slides)
- Hadley Wickham's *Advanced R* has a chapter on debugging



TROUBLESHOOTING SHINY

Shiny from RStudio™