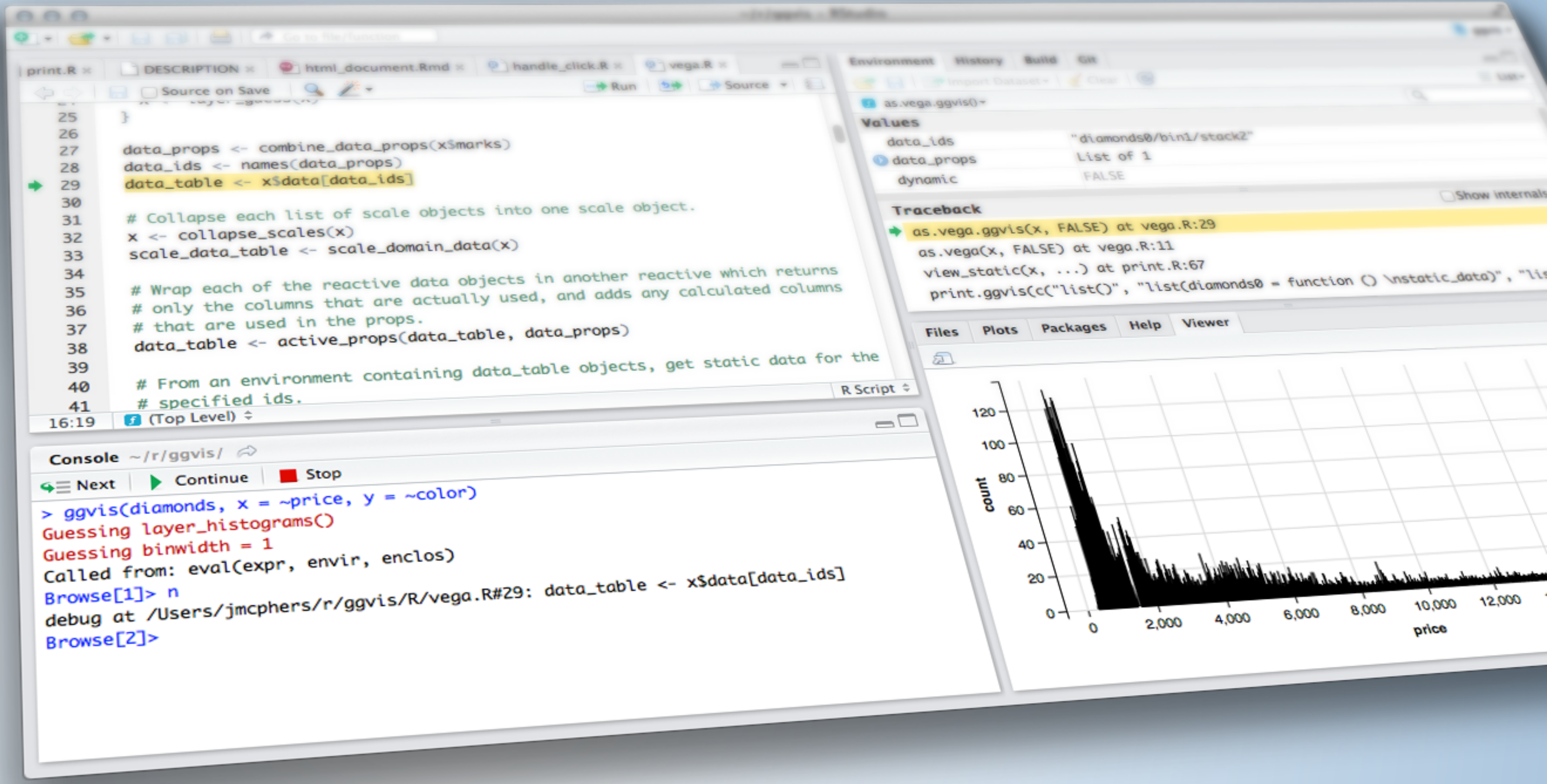


```
  }

  data_props <- combine_data_props(x$marks)
  data_ids <- names(data_props)
  data_table <- x$data[data_ids]

  # Collapse each list of scale objects into one scale object.
  x <- collapse_scales(x)
  scale_data_table <- scale_domain_data(x)

  # Wrap each of the reactive data objects in another reactive which returns
  # only the columns that are actually used, and adds any calculated columns
  # that are used in the props.
  data_table <- active_props(data_table, data_props)

  # From an environment containing data_table objects, get static data for the
  # specified ids.
```

```
Values
  data_ids      "diamonds0/bin1/stack2"
  data_props    List of 1
  dynamic       FALSE

Traceback
  as.vega.ggvis(x, FALSE) at vega.R:29
  as.vega(x, FALSE) at vega.R:11
  view_static(x, ...) at print.R:67
  print.ggvis(c("list()", "list(diamonds0 = function () \nstatic_data)", "li
```

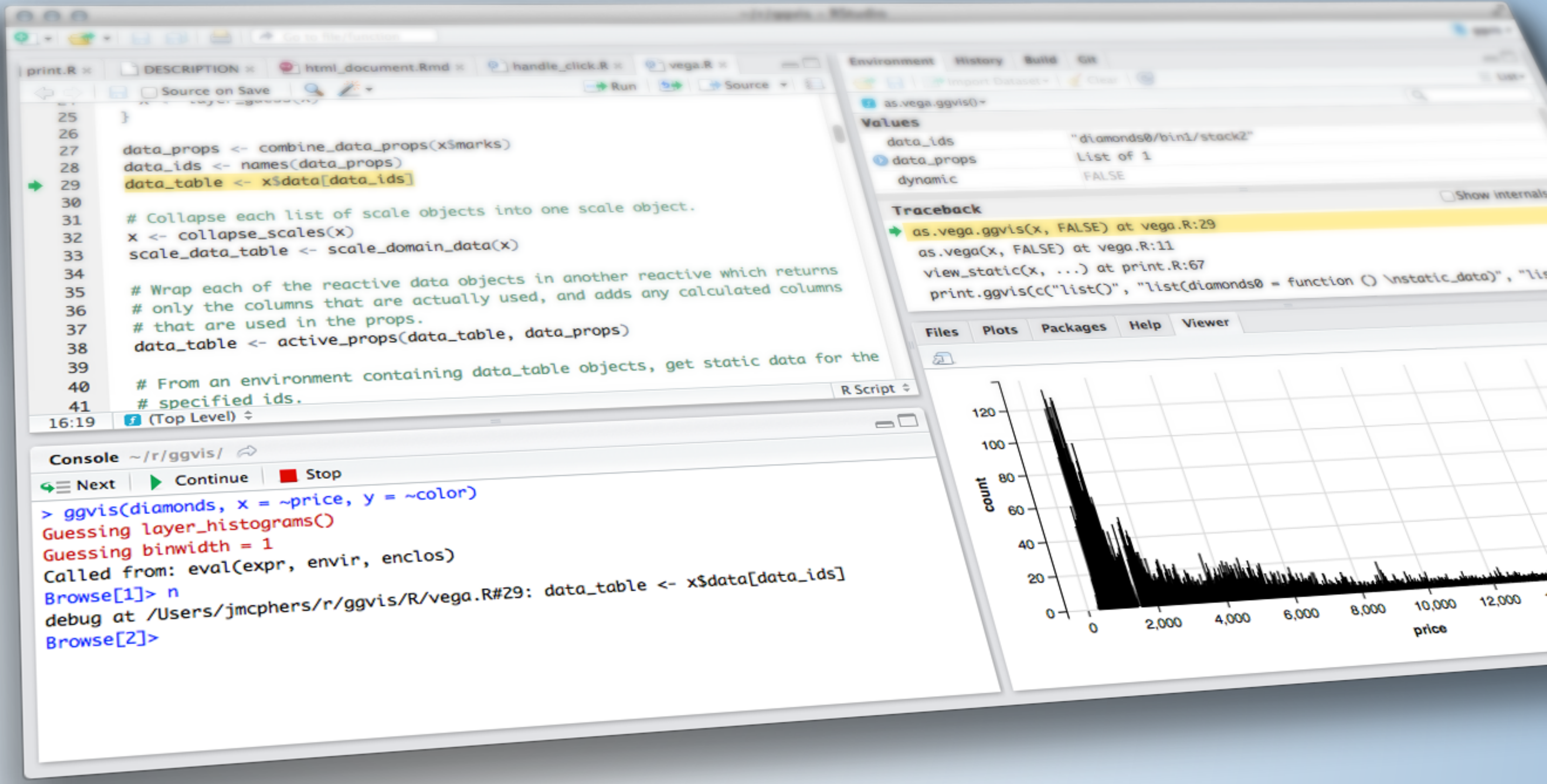
```
> ggvis(diamonds, x = ~price, y = ~color)
Guessing layer_histograms()
Guessing binwidth = 1
Called from: eval(expr, envir, enclos)
Browse[1]> n
debug at /Users/jmcphers/r/ggvis/R/vega.R#29: data_table <- x$data[data_ids]
Browse[2]>
```

DASHBOARDS & TROUBLESHOOTING

Shiny from RStudio

OUTLINE

- Dashboards
  - What is in a dashboard?
  - Server
    - reactiveFileReader
    - reactivePoll
  - UI
    - Static vs. dynamic dashboards
    - flexdashboard
    - Shiny pre-rendered
    - shinydashboard
- Troubleshooting
  - Writing robust code
  - Debugging tools at your disposal
  - Techniques for debugging



```r
}

data_props <- combine_data_props(x$marks)
data_ids <- names(data_props)
data_table <- x$data[data_ids]

# Collapse each list of scale objects into one scale object.
x <- collapse_scales(x)
scale_data_table <- scale_domain_data(x)

# Wrap each of the reactive data objects in another reactive which returns
# only the columns that are actually used, and adds any calculated columns
# that are used in the props.
data_table <- active_props(data_table, data_props)

# From an environment containing data_table objects, get static data for the
# specified ids.
```

```
Values
data_ids      "diamonds0/bin1/stack2"
data_props    List of 1
  dynamic     FALSE
```

```
Traceback
as.vega.ggvis(x, FALSE) at vega.R:29
as.vega(x, FALSE) at vega.R:11
view_static(x, ...) at print.R:67
print.ggvis(c("list()", "list(diamonds0 = function () \nstatic_data)", "li
```

```
> ggvis(diamonds, x = ~price, y = ~color)
Guessing layer_histograms()
Guessing binwidth = 1
Called from: eval(expr, envir, enclos)
Browse[1]> n
debug at /Users/jmcphers/r/ggvis/R/vega.R#29: data_table <- x$data[data_ids]
Browse[2]>
```

DASHBOARDS

Shiny from R Studio™

What is in a dashboard?

DASHBOARDS

- Automatically updating
    - Not just based on user gestures
    - But also when data source changes
- Many viewers looking at the same data
- May or may not be interactive

Server

MOTIVATION

- You have new data coming in — constantly, continuously, or on a schedule
- When new data comes in, it's automatically received, and transformed, aggregated, summarized, etc.
- May want to call attention to exceptional results

EXERCISE

- Why might this not be a good idea?

```
dataset <- reactive({
  result <- read.csv(“data.csv")
  invalidateLater(5000)
  result
})

output$plot <- renderPlot({
  plot(dataset()) # or whatever
})
```

SOLUTION

Lots of overhead!

reactiveFileReader

REACTIVEFILEREADER

- Reads the given file (`"data.csv"`) using the given function (`read.csv`)
- Periodically reads the last-modified time of the file
- If the timestamp changes, then (and only then) re-reads the file

Single file, on disk
(not database or web API)

```
dataset <- reactiveFileReader(
  intervalMillis = 1000,
  session = session,
  filePath = "data.csv",
  readFunc = read.csv
)

output$plot <- renderPlot({
  plot(dataset()) # or whatever
})
```

Must have data path as first argument

REACTIVEFILEREADER

```r
dataset <- reactiveFileReader(
  intervalMillis = 1000,
  session = session,
  filePath = "data.csv",
  readFunc = read.csv,
  stringsAsFactors = FALSE
)

output$plot <- renderPlot({
  plot(dataset()) # or whatever
})
```

Add any named arguments

reactivePoll

REACTIVEPOLL

- `reactiveFileReader` is limited to files on disk. It doesn't work for non-file-based data sources like databases or web APIs
- `reactivePoll` is a generalization of reactiveFileReader
    - `checkFunc`: A function that can execute quickly, and merely determine if anything has changed
        - Should be fast as it will block the R process while it runs! The slower it is, the greater you should make the polling interval.
        - Should not return `TRUE` or `FALSE` for changed/unchanged. Instead, just return a value (like the timestamp, or the count); it's `reactivePoll`'s job, not yours, to keep track of whether that value is the same as the previous value or not.
    - `valueFunc`: A function with the (potentially expensive) logic for actually reading the data

UI

Static vs. dynamic dashboards

STATIC VS. DYNAMIC

- Static:
  - R code runs once and generates an HTML page
  - Generation of this HTML can be scheduled
- Dynamic:
  - Client web browser connects to an R session running on server
  - User input causes server to do things and send information back to client
  - Interactivity can be on client and server
  - Can update data in real time
  - User potentially can do anything that R can do

FLEX VS. SHINY DASHBOARD

flexdashboard	shinydashboard
R Markdown	Shiny UI code
Super easy	Not quite as easy
Static or dynamic	Dynamic
CSS flexbox layout	Bootstrap grid layout

flexdashboard

EXERCISE

- `library(flexdashboard)`
- File → New file → R Markdown → From Template
- Create three plots that go in each of the panes using built-in R datasets or any data we have used in the worksho (or your own data)

3m 00s

EXERCISE

- Open `apps/flexdashboard_01.Rmd`
- How is it different than Shiny apps we have been building so far, how is it similar?
- Make a change to the layout of the dashboard, see http://rmarkdown.rstudio.com/flexdashboard/using.html#layout for help
- Change the theme of the dashboard, see http://rmarkdown.rstudio.com/flexdashboard/using.html#appearance for help

5m 00s

SHINY DOCUMENTS

- Add runtime: shiny to header.
- Add `inputs` in code chunks.
- Add `renderXyz` functions in code chunks.
  - No need for `output$x <-` assignment, or for `xyzOutput` functions.

EXERCISE

- Continue working on `apps/dashboards/flexdashboard_01.Rmd`
- Add another UI widget, a radioButton, that allows the user to select whether the plot used to visualize the distribution of weight should be histogram or a violin plot

3m 00s

SOLUTION

Sample solution at `apps/dashboards/flexdashboard_02.Rmd`

SHINY DOCUMENT DRAWBACKS

- Start-up time: knits document every time someone visits it
- Resizing can trigger re-knit
- Auto-reconnection doesn’t work (i.e. client browsers cannot automatically reconnect afer being disconnected due to network problems)

- The solution: Pre-rendered Shiny Documents

Shiny

pre-rendered

SHINY PRE_RENDERED

- Rendering phase: UI code (and select other code) is run once, before users connect.
- Serving phase: Server code is run once for each user session.
- Each phase is run in a separate R sessions and can’t access variables from the other phase.

CONTEXTS FOR SHINY_PRERENDERED

- `"render"`: Runs in rendering phase (like `ui`)
- `"server"`: Runs in serving phase (like `server`)
- Additional contexts:
  - `"setup"`: Runs in both phases (like `global.R`)
  - `"data"`: Runs in rendering phase (any variables are saved to a file, and available to serving phase, useful for data preprocessing)
  - `"server-start"`: Runs once in serving phase, when the Shiny document is first run and is not re-executed for each new user of the document, appropriate for
    - establishing shared connections to remote servers (e.g. databases, Spark contexts, etc.)
    - creating reactive values to be shared across sessions (e.g. with `reactivePoll`, `reactiveFileReader`)

EXERCISE

- Start with `apps/flexdashboard_02.Rmd`
- Turn your document into `runtime: shiny_prerendered`
- Note: You will need to use `output$x <-` assignment and `xyzOutput` functions

5m 00s

SOLUTION

Sample solution at `apps/flexdashboard_03.Rmd`

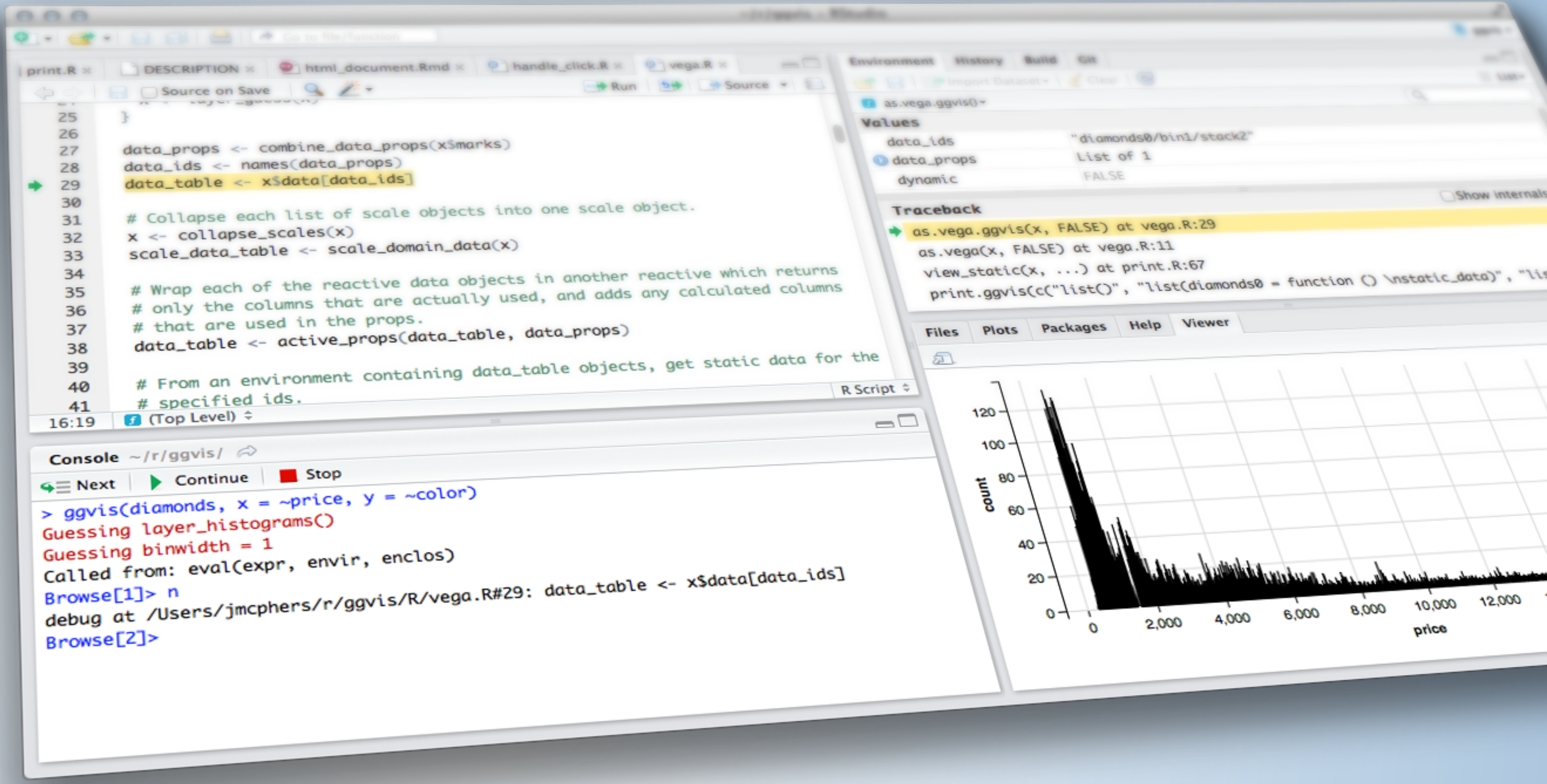
shinydashboard

SHINYDASHBOARD

- The UI for Shiny is built on the Bootstrap web framework
- Shinydashboard is a theme for Shiny, built on top of Bootstrap
- See http://rstudio.github.io/shinydashboard/ for more

HOMEWORK

Project 1



```
25  }
26
27  data_props <- combine_data_props(x$marks)
28  data_ids <- names(data_props)
29  data_table <- x$data[data_ids]
30
31  # Collapse each list of scale objects into one scale object.
32  x <- collapse_scales(x)
33  scale_data_table <- scale_domain_data(x)
34
35  # Wrap each of the reactive data objects in another reactive which returns
36  # only the columns that are actually used, and adds any calculated columns
37  # that are used in the props.
38  data_table <- active_props(data_table, data_props)
39
40  # From an environment containing data_table objects, get static data for the
41  # specified ids.
16:19  (Top Level)
```

```
Values
data_ids      "diamonds0/bin1/stack2"
data_props    List of 1
  dynamic     FALSE

Traceback
as.vega.ggvis(x, FALSE) at vega.R:29
as.vega(x, FALSE) at vega.R:11
view_static(x, ...) at print.R:67
print.ggvis(c("list()", "list(diamonds0 = function () \nstatic_data)", "li
```

```
> ggvis(diamonds, x = ~price, y = ~color)
Guessing layer_histograms()
Guessing binwidth = 1
Called from: eval(expr, envir, enclos)
Browse[1]> n
debug at /Users/jmcphers/r/ggvis/R/vega.R#29: data_table <- x$data[data_ids]
Browse[2]>
```

TROUBLESHOOTING SHINY

Shiny from RStudio

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.

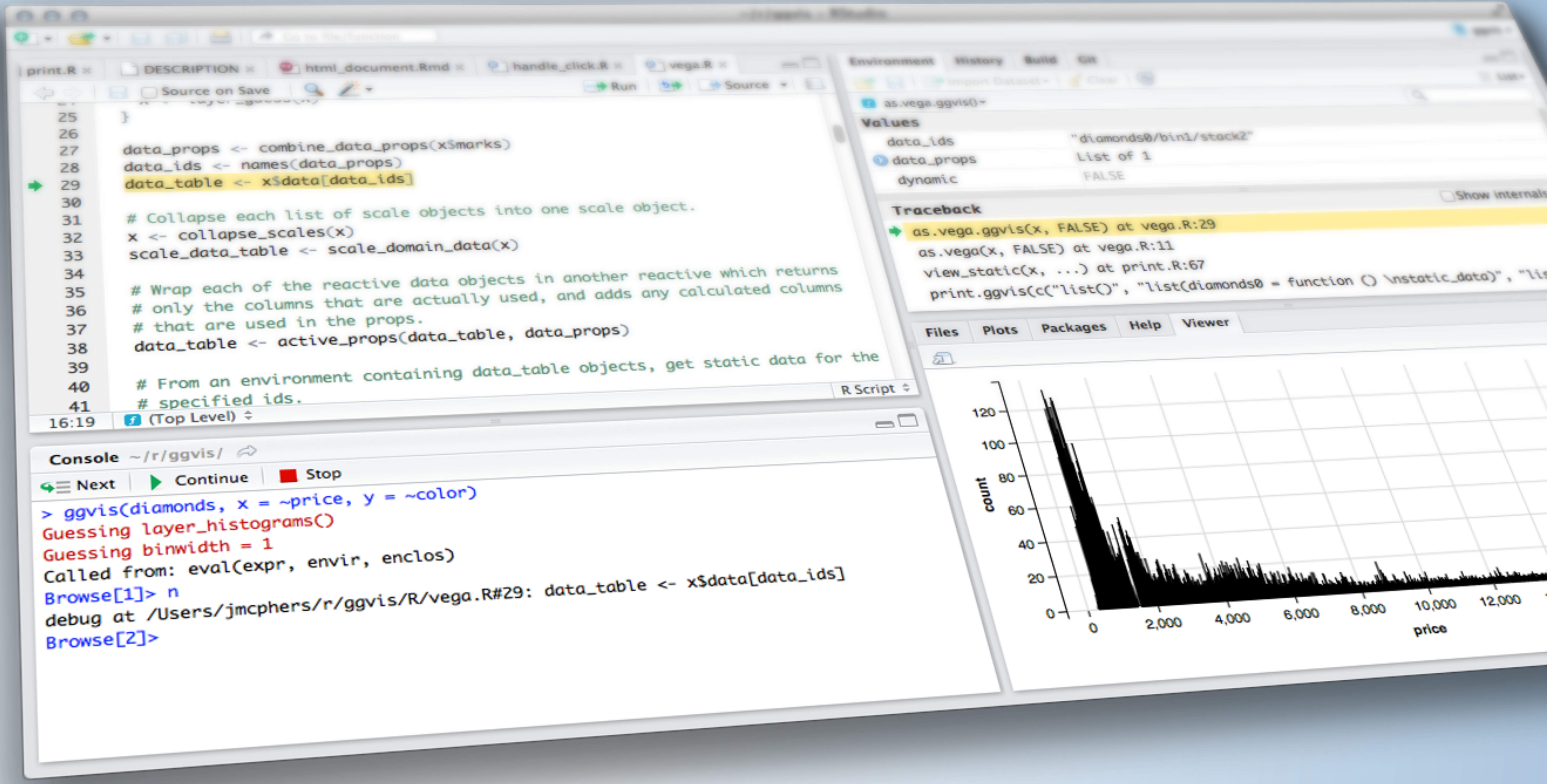
COMMON ERRORS

- "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



```
}

data_props <- combine_data_props(x$marks)
data_ids <- names(data_props)
data_table <- x$data[data_ids]

# Collapse each list of scale objects into one scale object.
x <- collapse_scales(x)
scale_data_table <- scale_domain_data(x)

# Wrap each of the reactive data objects in another reactive which returns
# only the columns that are actually used, and adds any calculated columns
# that are used in the props.
data_table <- active_props(data_table, data_props)

# From an environment containing data_table objects, get static data for the
# specified ids.
```

```
data_ids      "diamonds0/bin1/stack2"
data_props    List of 1
dynamic       FALSE
```

```
as.vega.ggvis(x, FALSE) at vega.R:29
as.vega(x, FALSE) at vega.R:11
view_static(x, ...) at print.R:67
print.ggvis(c("list()", "list(diamonds0 = function () \nstatic_data)", "li
```

```
> ggvis(diamonds, x = ~price, y = ~color)
Guessing layer_histograms()
Guessing binwidth = 1
Called from: eval(expr, envir, enclos)
Browse[1]> n
debug at /Users/jmcphers/r/ggvis/R/vega.R#29: data_table <- x$data[data_ids]
Browse[2]>
```

DASHBOARDS & TROUBLESHOOTING

Shiny from RStudio