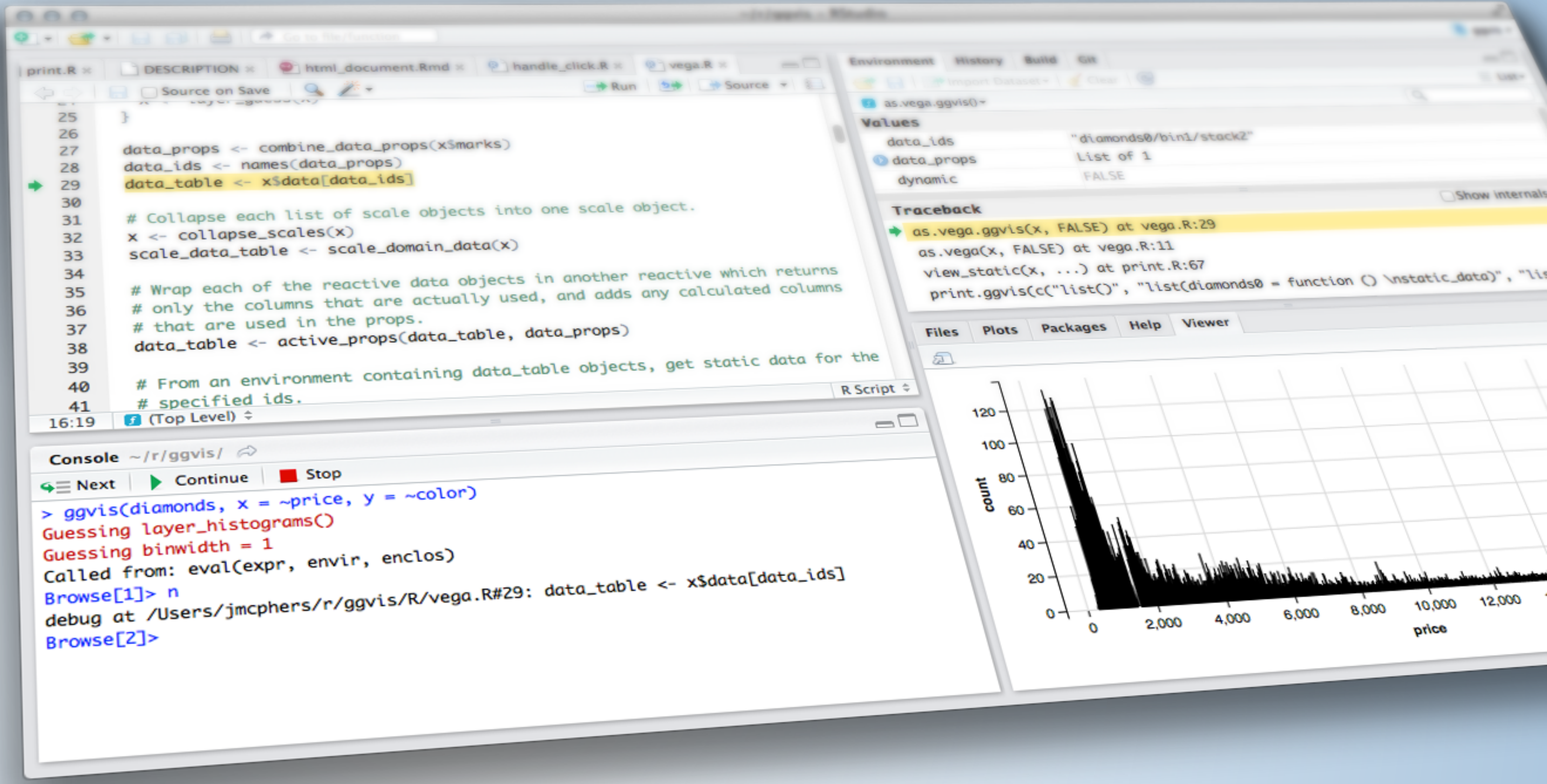


```
}

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.
```

```
> 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]>
```

```
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
```

LEAFLETPROXY

Shiny from R Studio

OUTLINE

- Leaflet in Shiny
  - renderLeaflet
  - leafletOutput
- What we know about reactivity
- LeafletProxy
  - observe
  - events

Leaflet

In Shiny

JUST LIKE ANY OTHER OUTPUT

- You need a render function and a subsequent output in the UI
- Reacts to reactive functions and inputs like any other plot
- However, it will take much longer to render than a typical ggplot2 graph.

What we know about reactivity

REVIEW: OBSERVERS

- Use to execute actions based on changing reactive values and other reactive expressions.
- Doesn't return a value. So performing side effects is usually the only reason you'd want to create one of these.
- Eagerly executed by Shiny.

```
observe({
  print(paste("The value of x is", input$x))
})

## [1] The value of x is 10
## [1] The value of x is 16
## [1] The value of x is 9
```

OBSERVERS IN LEAFLET

Observers in leaflet can do a number of thing

- Move the map (fitBounds,)
- Remove specific markers/shapes/lines (removeShape…)
- Remove entire groups
- Change the basemap
- And more!

LEAFLET PROXY

- Inside an observer target the map with the leafletProxy() function
- Clear the group before re-adding it
- Note: This doesn't really work well with clusterOptions.

```
leafletProxy("map") %>% # this should be the main map id
    clearGroup(“groupName”) %>%
    addMarkers(markersReactive(), group = “groupName" …)
```

EXERCISE

- Open /apps/green_inf.R
  - Right now this application re-runs the entire map every time a change is made to an input.
  - Make a new observer expression that only edits the layer of green infrastructure projects, and does not reload the entire map itself.

10m 00s

SOLUTION

`apps/green_inf_proxy_01.R`

WHAT ABOUT SHAPES?

- You can do this with shapes too!
- It works the same way as points. This is also true for any other layer type, lines, heat maps etc.

EXERCISE

- Open /apps/green_inf_proxy_01.R
  - Add a new reactive expression that selects the polygon of the selected borough
  - Add a new observer that adds the selected borough to the map as well.

10m 00s

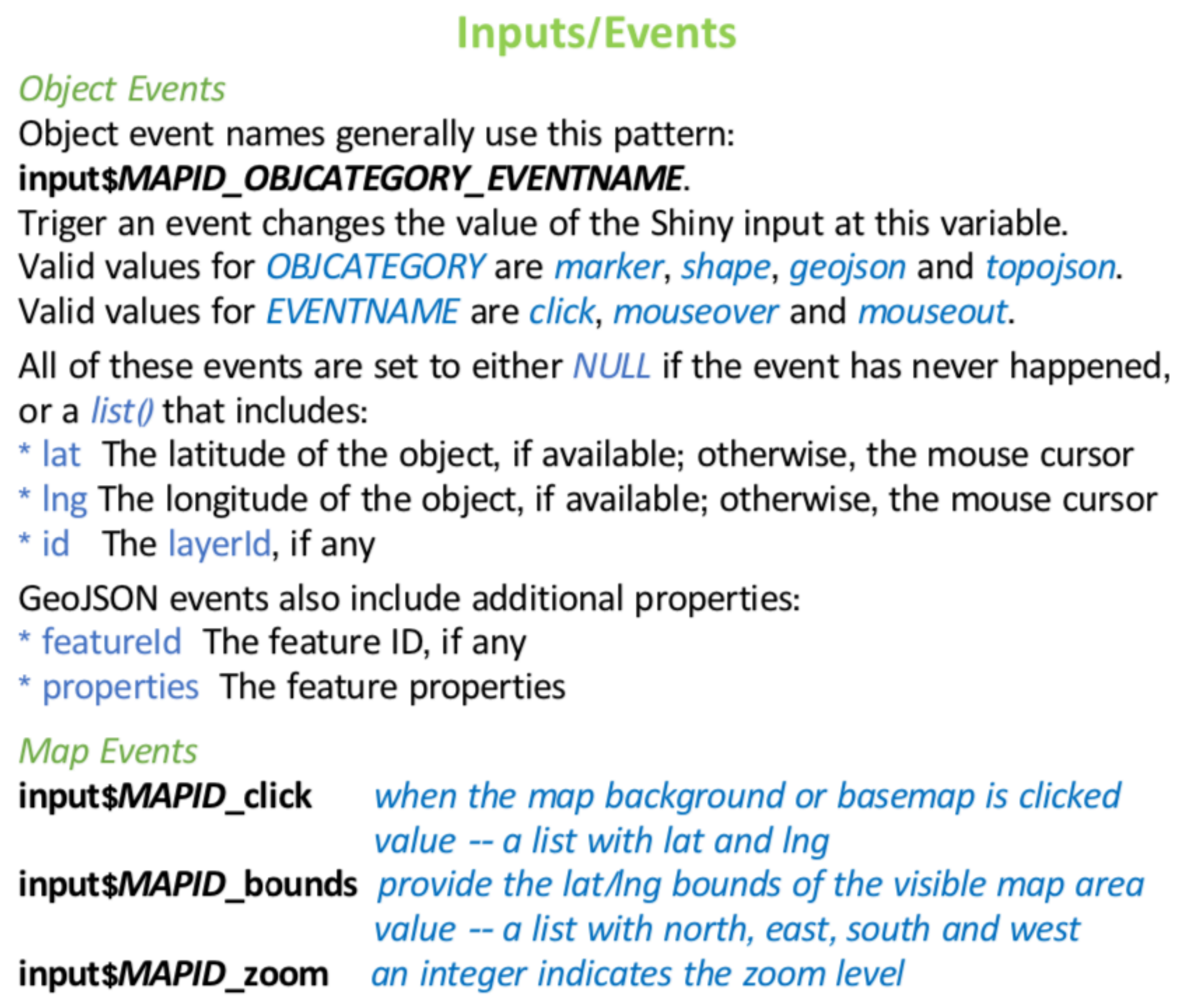
SOLUTION

`apps/green_inf_proxy_02.R`

Inputs and Events

USES FOR INPUTS

- Sometimes you might need to have things happen based off of user inputs or map bounds
  - Charts showing information based off of what is within the map bounds
  - Showing details/plots of a selected item



Inputs/Events

Object Events

Object event names generally use this pattern:
input$*MAPID_OBJCATEGORY_EVENTNAME*.
Triger an event changes the value of the Shiny input at this variable.
Valid values for *OBJCATEGORY* are *marker*, *shape*, *geojson* and *topojson*.
Valid values for *EVENTNAME* are *click*, *mouseover* and *mouseout*.

All of these events are set to either *NULL* if the event has never happened, or a *list()* that includes:

* lat The latitude of the object, if available; otherwise, the mouse cursor
* lng The longitude of the object, if available; otherwise, the mouse cursor
* id The layerId, if any

GeoJSON events also include additional properties:

* featureId The feature ID, if any
* properties The feature properties

Map Events

input$*MAPID*_click	*when the map background or basemap is clicked* *value -- a list with lat and lng*
input$*MAPID*_bounds	*provide the lat/lng bounds of the visible map area* *value -- a list with north, east, south and west*
input$*MAPID*_zoom	*an integer indicates the zoom level*

leaflet.pdf cheatsheet

INPUTS GENERATED

In all of these examples "leaflet" is whatever you called your (ie: output$leaflet). These are the kinds of things you may want to hide from your bookmarking.

- input$leaflet_center
  - $lat, $lng
- input$leaflet_zoom
- input$leaflet_bounds
  - $north, $east, $south, $west
- input$leaflet_marker_mouseout$
  - $id, $group, $lat, $lng
- input$leaflet_marker_click$
  - $id, $group, $lat, $lng
- input$leaflet_groups (list of active group names)

DEMO

http://shiny.rstudio.com/gallery/superzip-example.html

USING PROXY

- Unlike ggplot2 compute time for a leaflet map is time consuming.
- The leafletProxy function and observers allow us to only change the parts of the map that are necessary
  - This saves computing power and speeds up rendering
  - It also keeps the user from having to deal with a resetting map

EXERCISE

- Open apps/green_inf_proxy_02.R
  - Let’s create a UI element that shows the user the number of projects they are viewing.
    - Hint: Check the code from superzip and see how they used the bounds input
    - Note: I added some lines to this app that creates the coordinates as variables in the @data frame.

10m 00s

SOLUTION

`apps/green_inf_proxy_03.R`

REACTIVE VALUES REVIEW

- Like an R environment object (or what other languages call a hash table or dictionary), but reactive
- Like the input object, but not read-only

```
rv <- reactiveValues(x = 10)
rv$x <- 20
rv$y <- mtcars
```

REACTIVE VALUES REVIEW CONT.

- Reading a value from a reactiveValues object is a reactive operation.
  - The act of reading it means the current reactive conductor or endpoint will be notified the next time the value changes.
- Maybe surprisingly, setting/updating a value on a reactiveValues object is not in itself a reactive operation, meaning no relationship is established between the current reactive conductor or endpoint (if any!) and the reactiveValues object.

EXERCISE

- Open /apps/green_inf_proxy_03.R
  - Add a reactive list to store removed projects
  - Edit the reactive expression to remove projects that have been removed by the user
    - Hint: append stored values in a reactive list
- Stretch Goal: add a function that restores the

10m 00s

SOLUTION

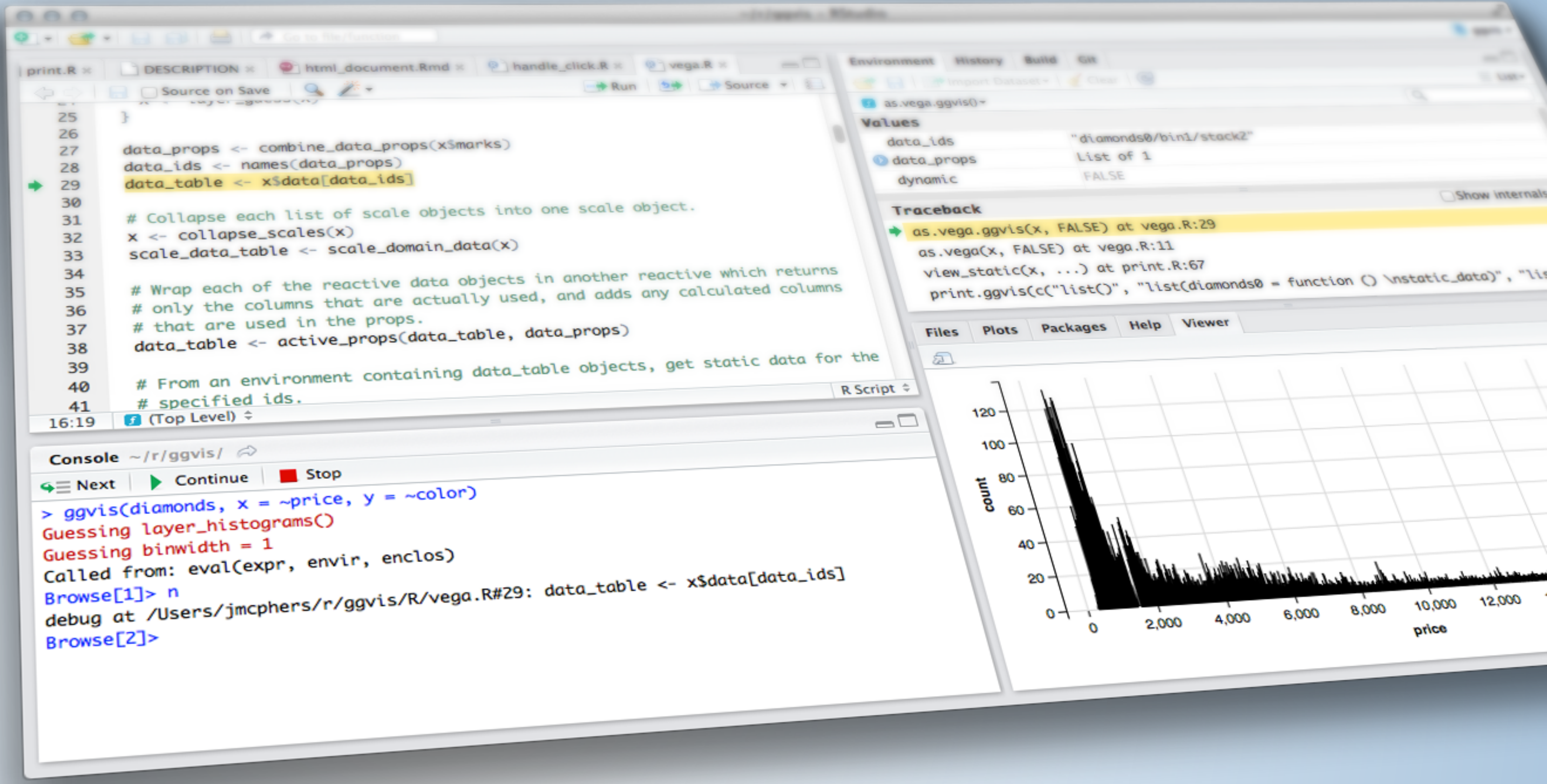
`apps/green_inf_proxy_04.R`

OTHER CONCEPTS TO RECALL

- Refresh limits on LeafletProxy maps are a very good idea

- Keep in mind that add ons like layer controls mean you don't have to build tons of functionality into your app

- You can target individual shapes if you give them an id, just like row number in DT package



```
}

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.
```

```
> 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]>
```

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
```

LEAFLETPROXY

Shiny from RStudio

HOMEWORK

Project 2