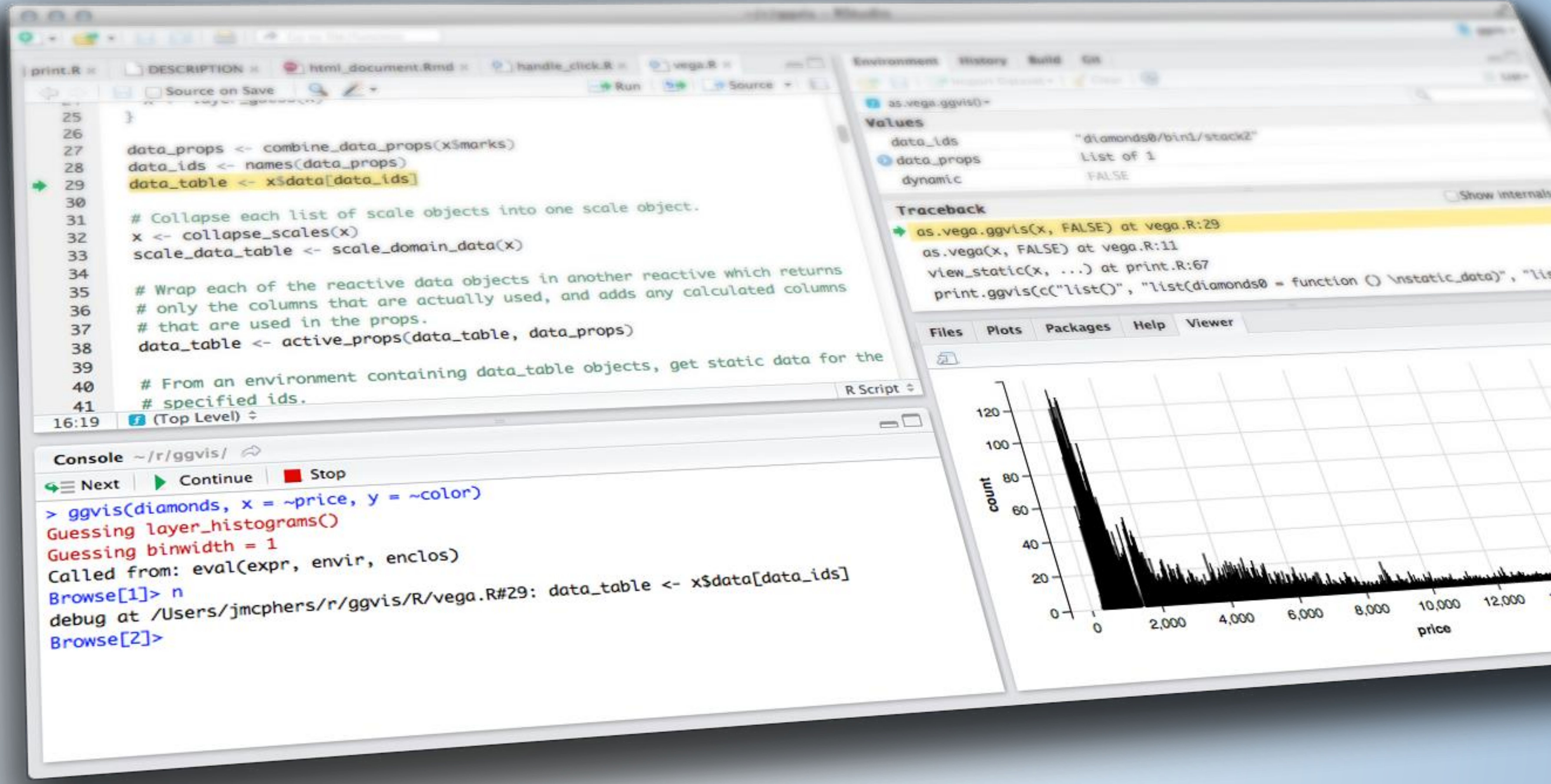


```
}

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.
```

16:19 (Top Level)

as.vega.ggvis()

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
```

Console ~/r/ggvis/

```
> 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]>
```

LEAFLET & LEAFLETPROXY

Shiny from R Studio

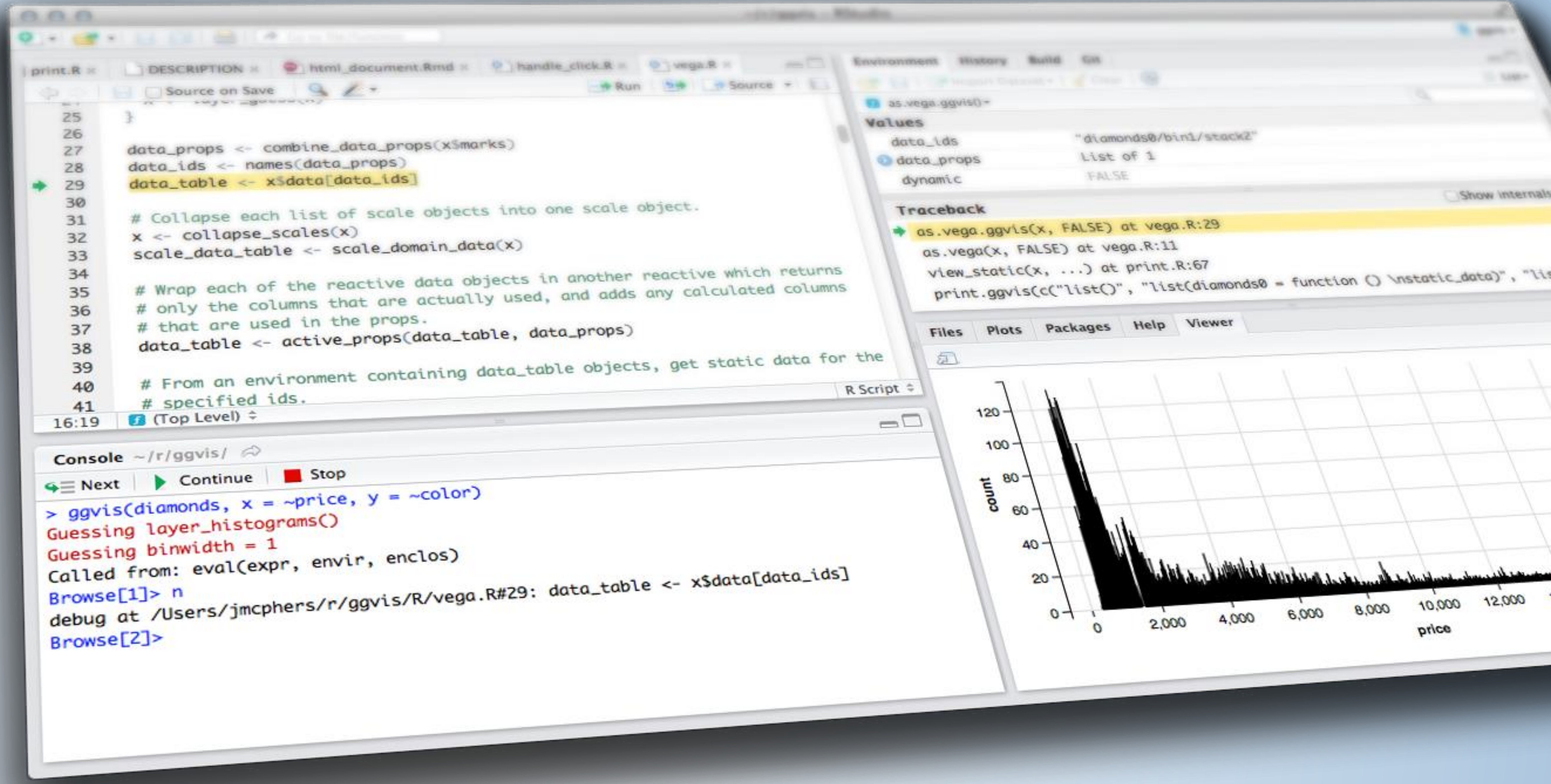
OUTLINE

- Motivation
- Shapefiles in R
- Leaflet
  - Basemaps
  - Lines
  - Points
  - Shapes
  - Legends & Colors
- leaflet.extras
  - Heatmaps
- Leaflet in Shiny
  - renderLeaflet
  - leafletOutput
- What we know about reactivity
- LeafletProxy
  - Observe
  - events

But first…

10m 00s

SmartEvals!



```
}

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

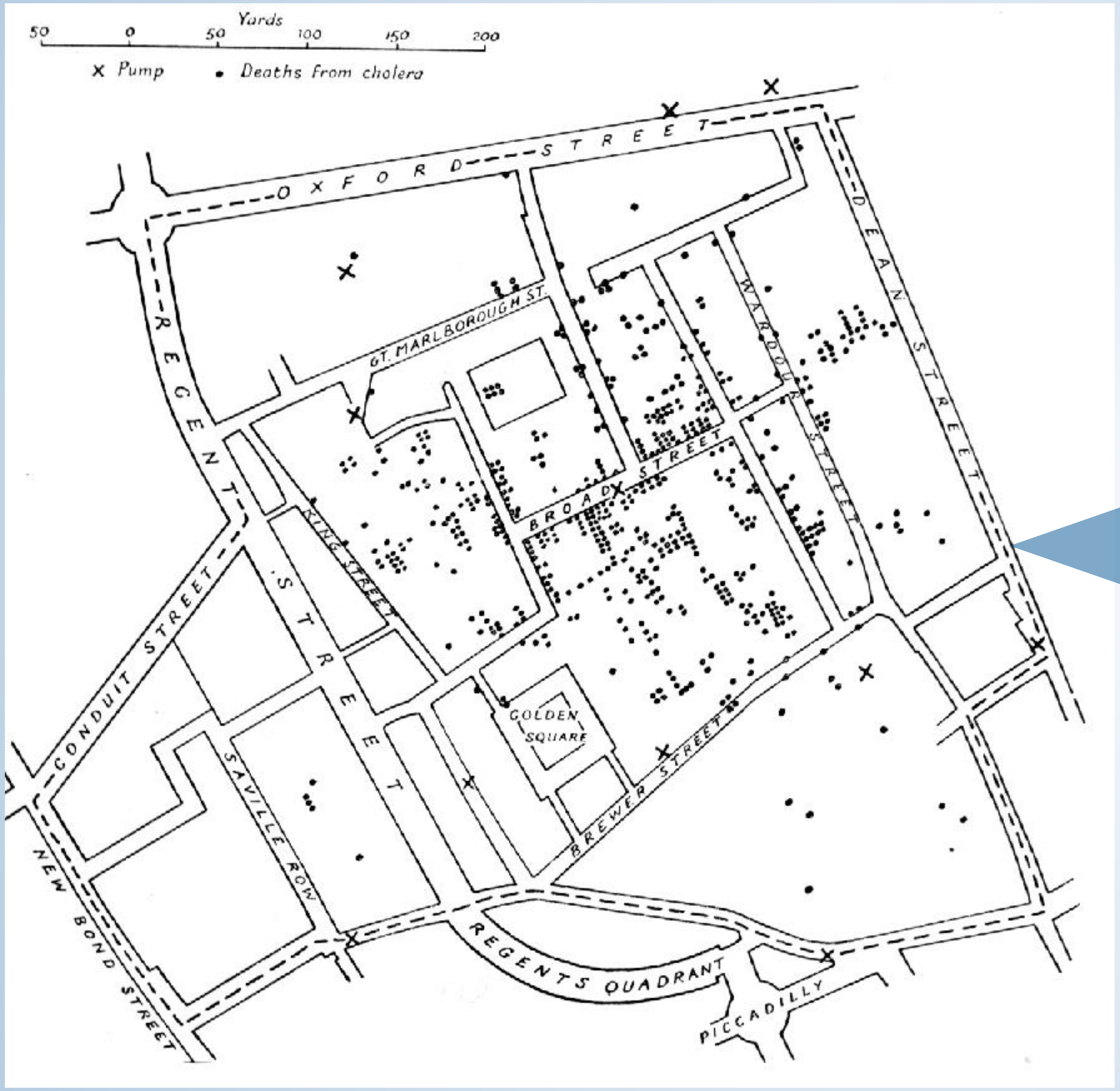
```
as.vega.ggvis(x, FALSE) at vega.R:29
as.vega(x, FALSE) at vega.R:11
view_static(x, ...) at print.R:67
print.ggvis(c("list()", "list(diamonds0 = function () \nstatic_data)", "li
```

LEAFLET

FOR INTERACTIVE MAPS

Shiny from R Studio™

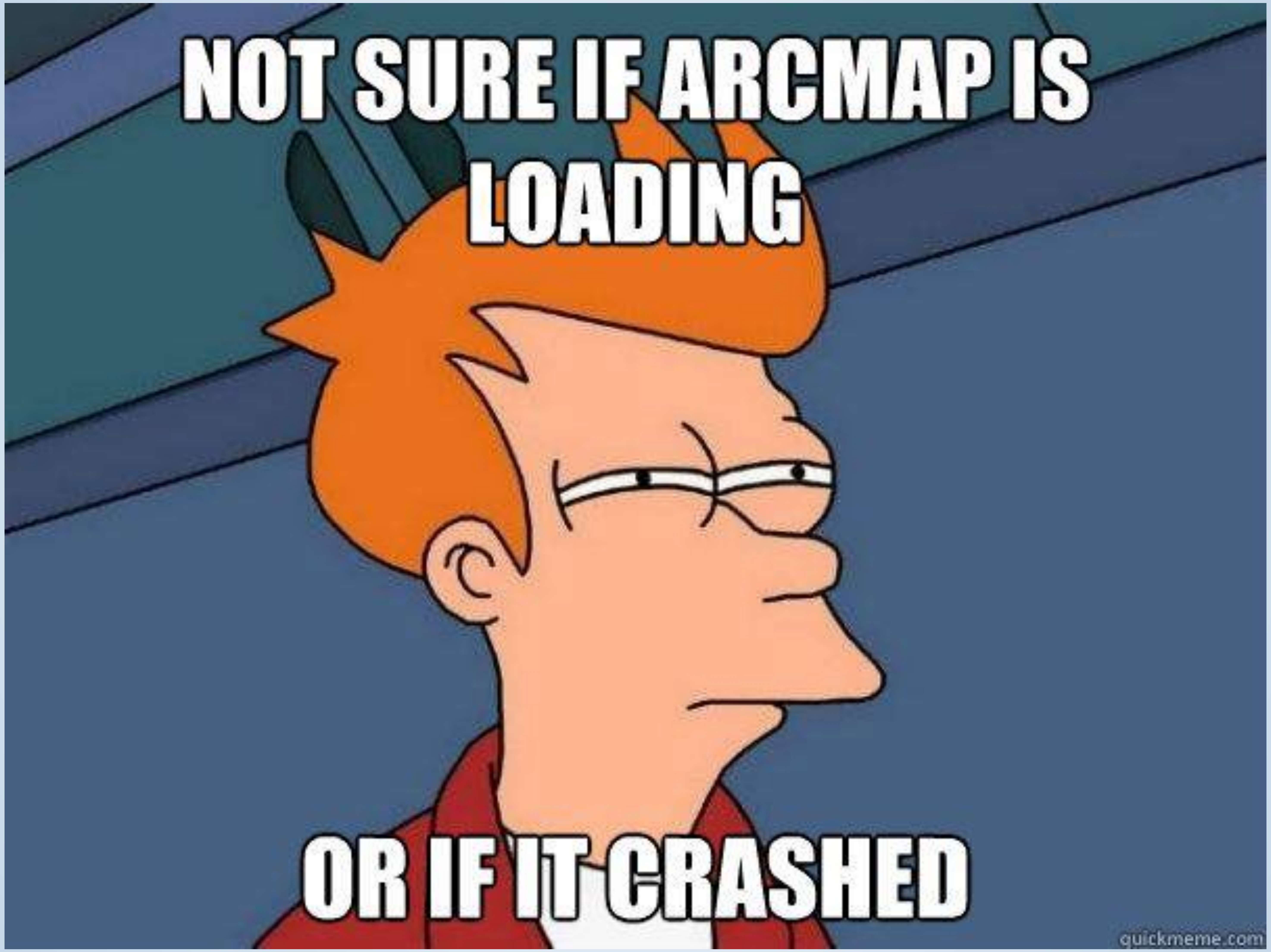
Motivation



Map of the 1854 Cholera outbreak - John Snow

*“‘I have the same problem’ is a famous **last** post in many forum-threads on the esri forum.”*

–sebastian


NOT SURE IF ARCMAP IS LOADING
OR IF IT CRASHED
quickmeme.com

“R and Leaflet are free and open-source, ArcMap is very much not.”

–Geoffrey Arnold

Shapefiles

In R

PACKAGES

- Install these packages:
  - rgdal: https://github.com/cran/rgdal/blob/master/inst/README
  - rgeos
  - sf
  - leaflet
  - leaflet.extras

WHAT IS LEAFLET?

- R version of the Javavscript API of Leaflet
- One of the ways to get interactive maps
    - Others include: tmap, mapview and plotly
- Documentation: https://rstudio.github.io/leaflet/

Getting spatial data loaded into R

LOADING SPATIAL DATA

- Sometimes your source data will be a data frame which you will need to join to spatial data, other times it will be included

- It may also be a CSV with coordinates, in those instances no further cleaning needs to take place

DEMO

leaflet_examples.Rmd

Go to loading code chunk

Shapes

TYPICAL ARGUMENTS

- lng (if a column)
- lat (if a column)
- layerId
- group (for layerControls)
- stroke (boolean, shape outline)
- color (outline color, hex values)
- weight (outline width)
- opacity (alpha of the line)
- fill (boolean)
- fillColor (hex color)
- fillOpacity (alpha of fill)
- And more!

EXERCISE

- Using the example cds to create a polygon layer
  - Make sure to include a basemap

5m 00s

SOLUTION

See leaflet_exercises_solutions.Rmd

```
leaflet(data = cds) %>%
  addProviderTiles("Stamen.Toner") %>%
  addPolygons()
```

PALETTES

- palette (color brewer palette)
- domain (values to be colored)
- na.color
- alpha
- reverse (values of palette)

- bins
- pretty
- right (for cutting)
- n (number of quanitites)
- probs
- levels
- ordered

LEGENDS

- position (“bottomright” etc…)
- pal (palette from colorBrewer)
- values (domain from data)
- na.labels
- bins (buckets)
- colors
- labFormat (separate function labelFormat)
- title
- className (for custom CSS to apply)
- layerId (for input usage)
- group (for layerControls)
- digits
- big.mark
- transform (function to be applied to labels)

EXERCISE

- Choropleth Polygon map with
  - Create a color palette for Life Expectancy at Birth (years) column in the merged Congressional District data.
    - addPolygons to a leaflet map and apply palette to column
      - Stretch goal: Add a legend

5m 00s

Hint: https://rstudio.github.io/leaflet/choropleths.html

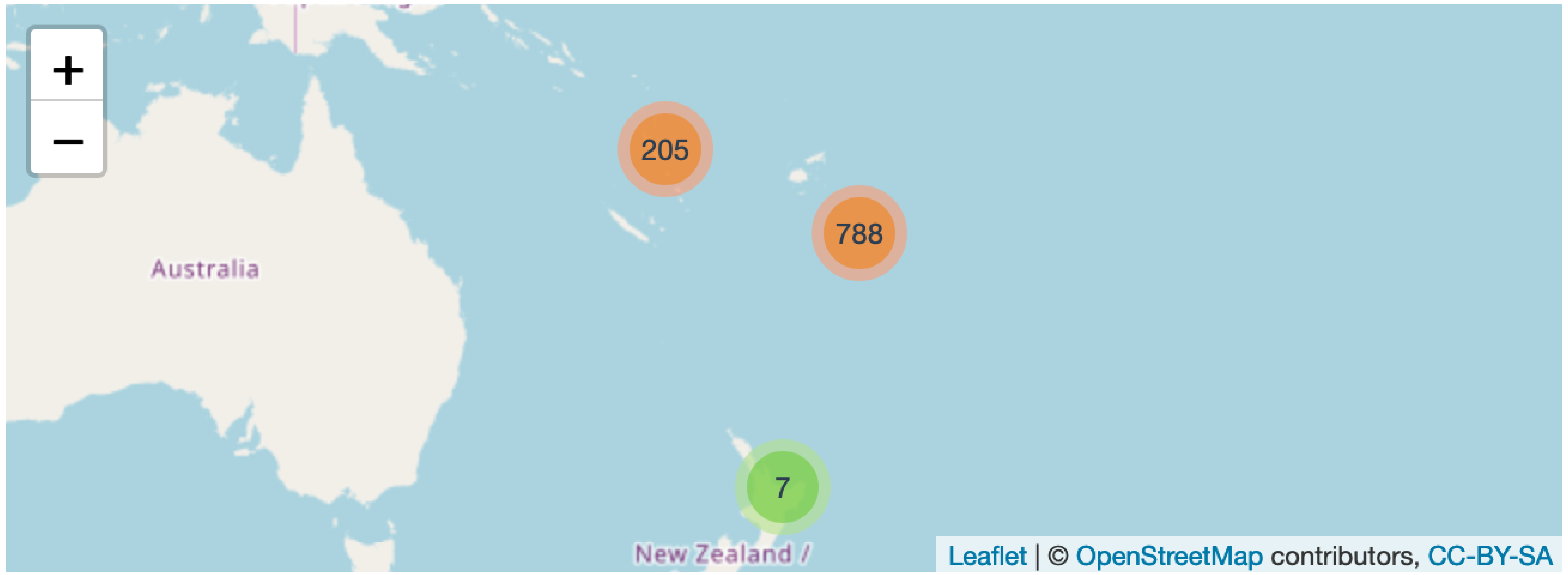
DEMO

leaflet_examples.Rmd

Go to polygon code chunk

CLUSTERS

```
leaflet(quakes) %>%
addTiles() %>%
addMarkers(
  clusterOptions =
markerClusterOptions()
)
```



WHY CLUSTERS?

- Sometimes there’s just a ton of data to show and trying to visualize them will bring any browser window it a crawl.

- Example: https://pittsburghpa.shinyapps.io/TreesNAt/

EXERCISE

- Go to leaflet_exercises.Rmd clusters code chunk
  - Use 311 data to create a cluster map
    - Create a factor palette
      - Create a legend
        - Make sure to include a basemap

5m 00s

Hint: https://rstudio.github.io/leaflet/markers.html

SOLUTION

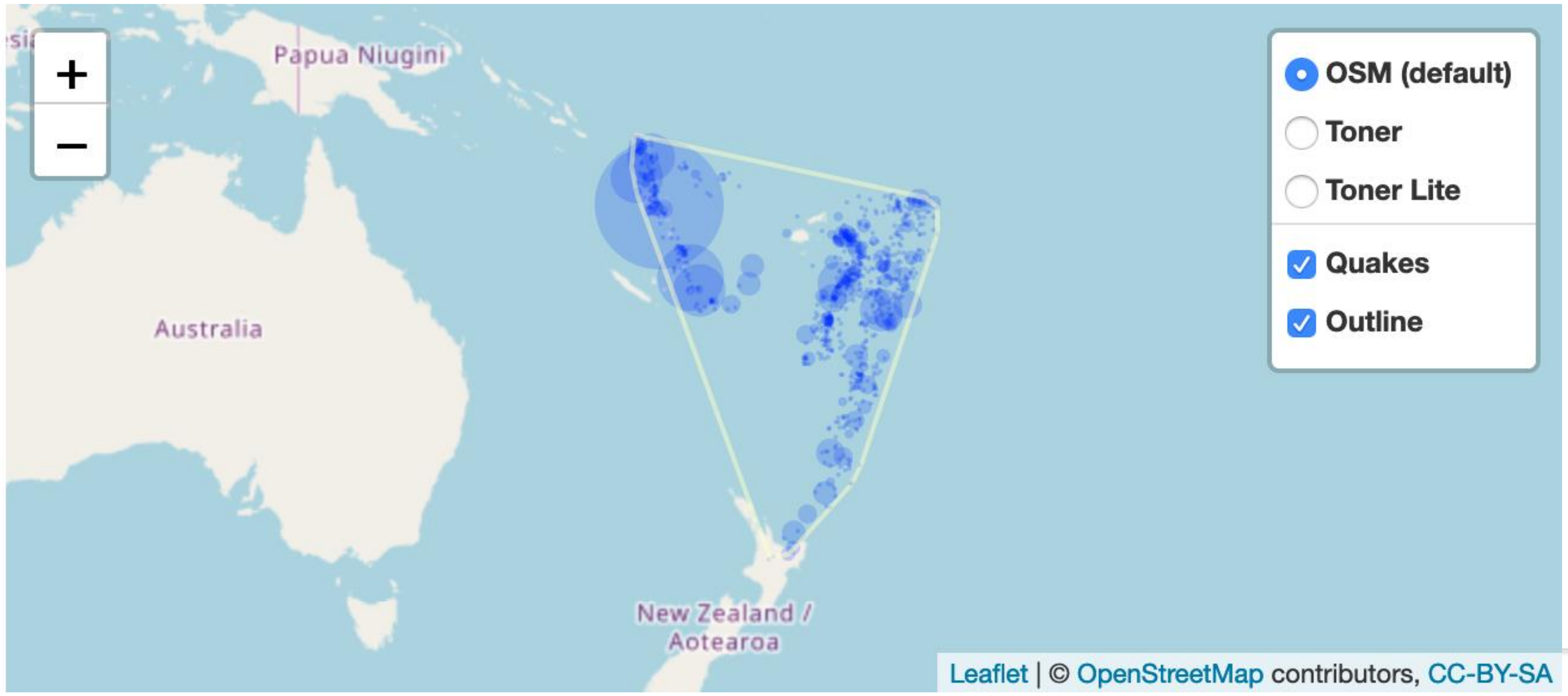
```
Go to leaflet_exercises_solutions.Rmd
```

GROUP CONTROLS

In leaflet you can add groups and give them a name.

Group names are what show up in the layer control

```
addProviderTiles(providers$Stamen.Toner, group = “Toner”) %>%
addLayersControl(
    baseGroups = c("OSM (default)", "Toner", "Toner Lite"),
    overlayGroups = c("Quakes", "Outline"),
    options = layersControlOptions(collapsed = FALSE)
  )
```



● OSM (default)
○ Toner
○ Toner Lite
✓ Quakes
✓ Outline

EXERCISE

- Create a map with layer controls
  - Get basemap provider names from here: https://leaflet-extras.github.io/leaflet-providers/preview/
    - Create 3 named basemap groups
    - Add a layer control that has the named base groups

5m 00s

SOLUTION

```
Go to leaflet_exercises_solutions.Rmd
```

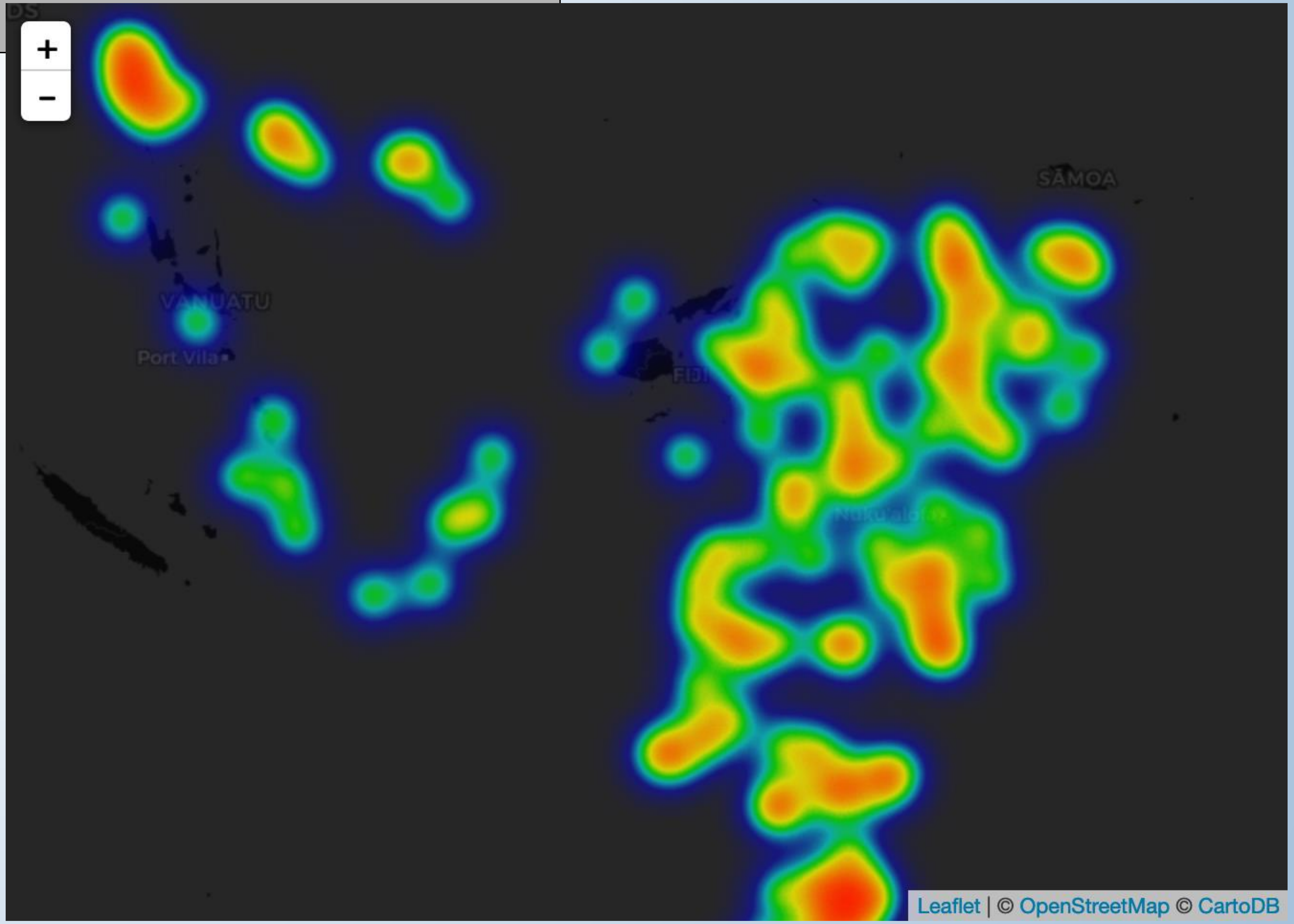
leaflet

.extras

LEAFLET.EXTRAS

- Adds some pretty nice functions to complement leaflet
  - Tile caching: http://rpubs.com/bhaskarvk/TileLayer-Caching
  - weather icons: http://rpubs.com/bhaskarvk/leaflet-weather
  - Pulse icons: http://rpubs.com/bhaskarvk/leaflet-pulseIcon
  - and heat maps!

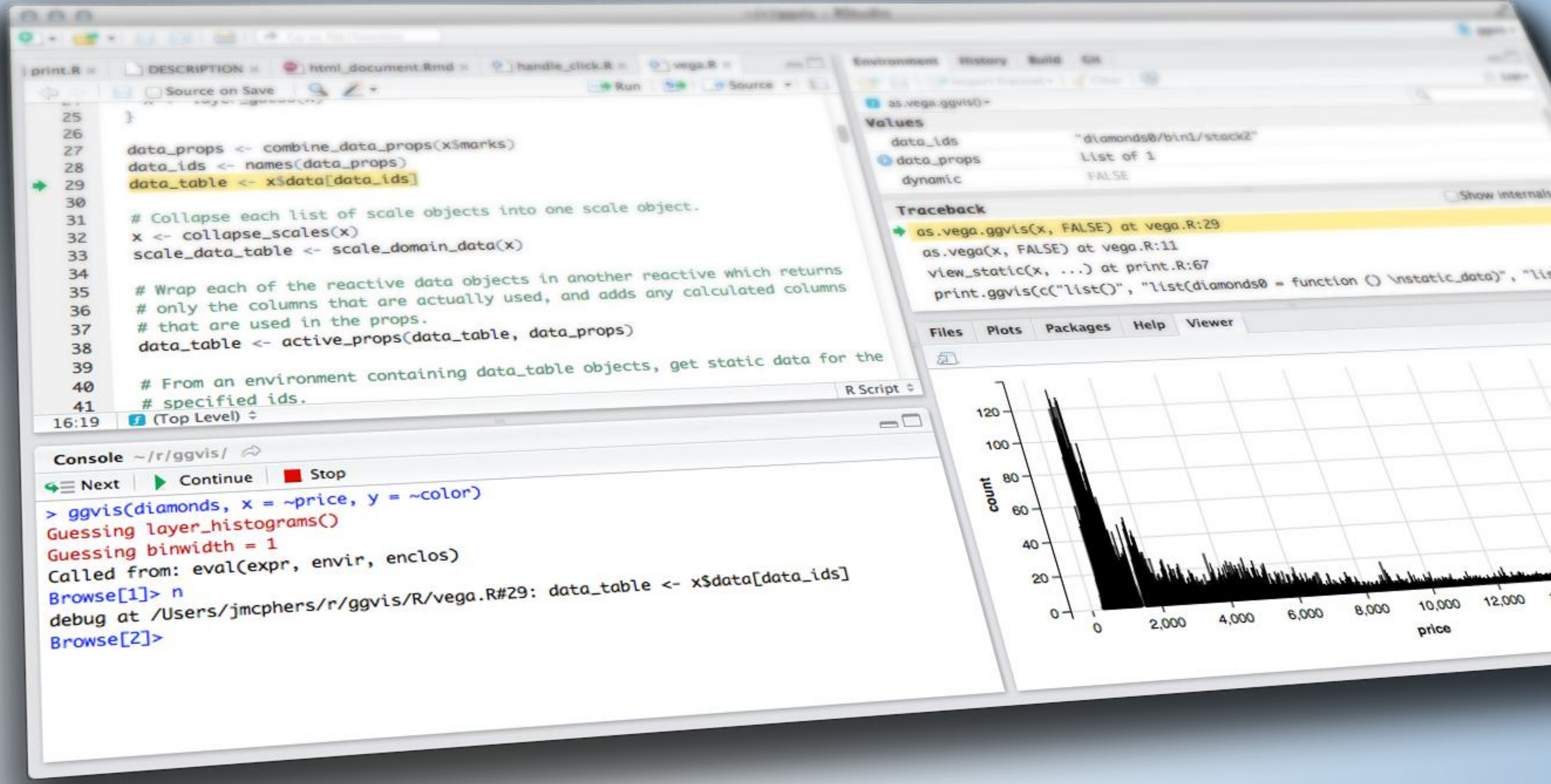
```
leaflet(quakes) %>%
addProviderTiles(providers$CartoDB.DarkMatter) %>%
  setView( 178, -20, 5 ) %>%
  addHeatmap(lng = ~long, lat = ~lat, intensity = ~mag,
            blur = 20, max = 0.05, radius = 15)
```



EXERCISE

- Using the pothole data create a heat map
  - This won’t be very different from your cluster map from before.
    - Play around with the arguments for heat maps and find a radius that looks good.
- Compare your outcome with your neighbor

5m 00s



```
}

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.
```

```
as.vega.ggvis() ▾
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]>
```

LEAFLETPROXY

Shiny from R Studio

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

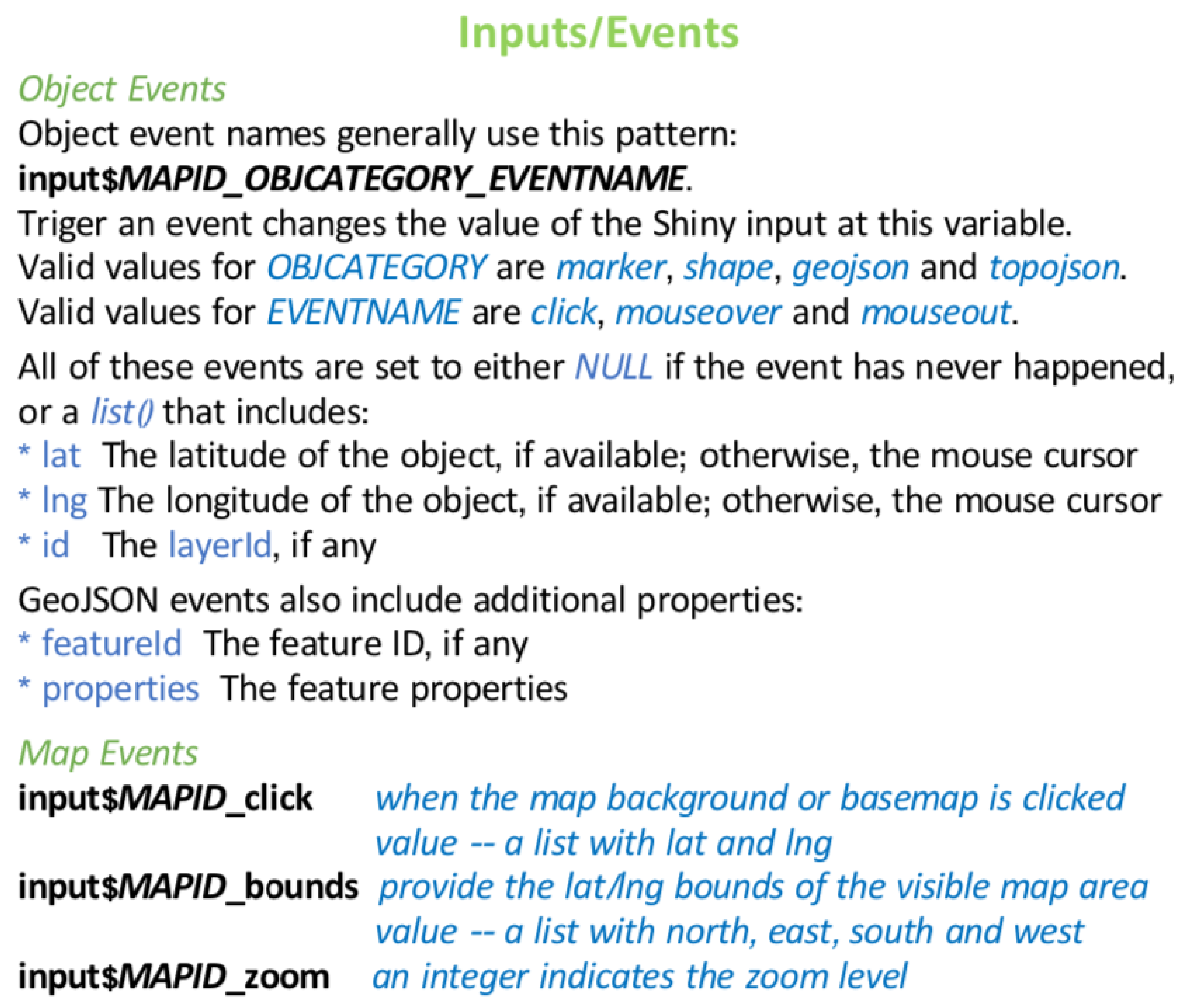
SOLUTION

```
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

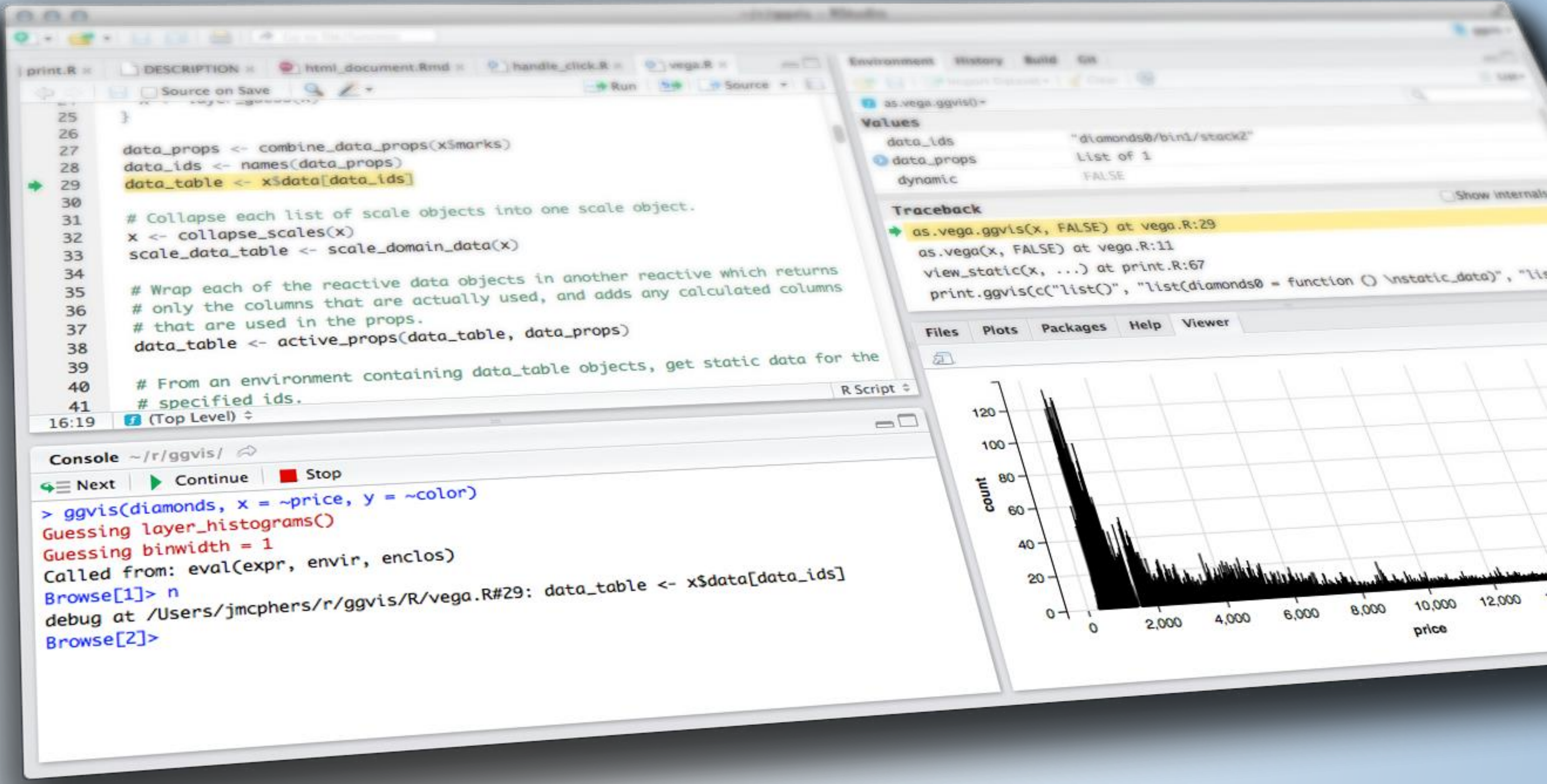
```
apps/green_inf_proxy_04.R
```

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.
```

16:19 (Top Level)

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]>
```

LEAFLETPROXY

Shiny from RStudio

HOMEWORK

Final Project