

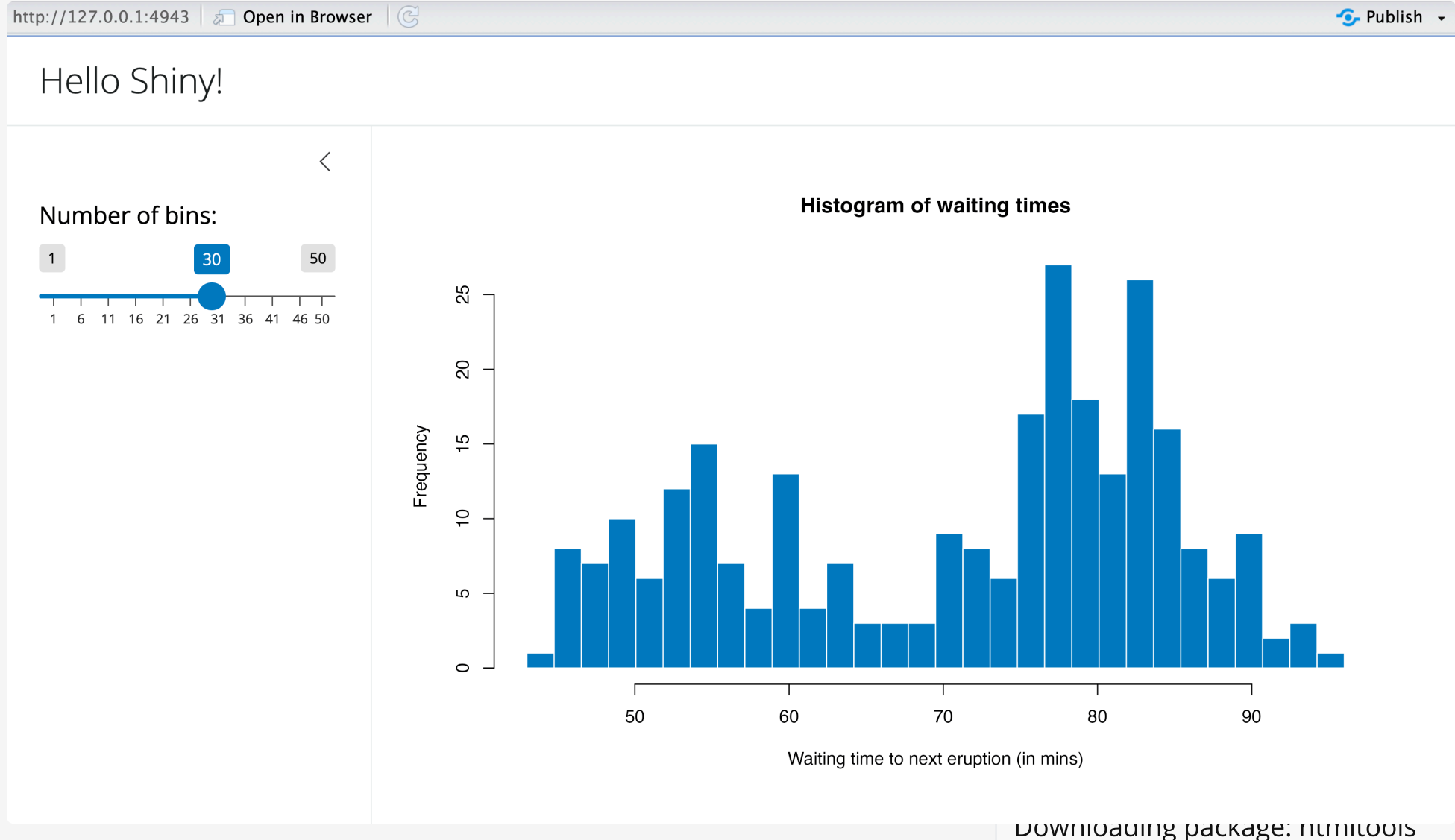
# Interactivity

Downloading webR

# Interactive elements

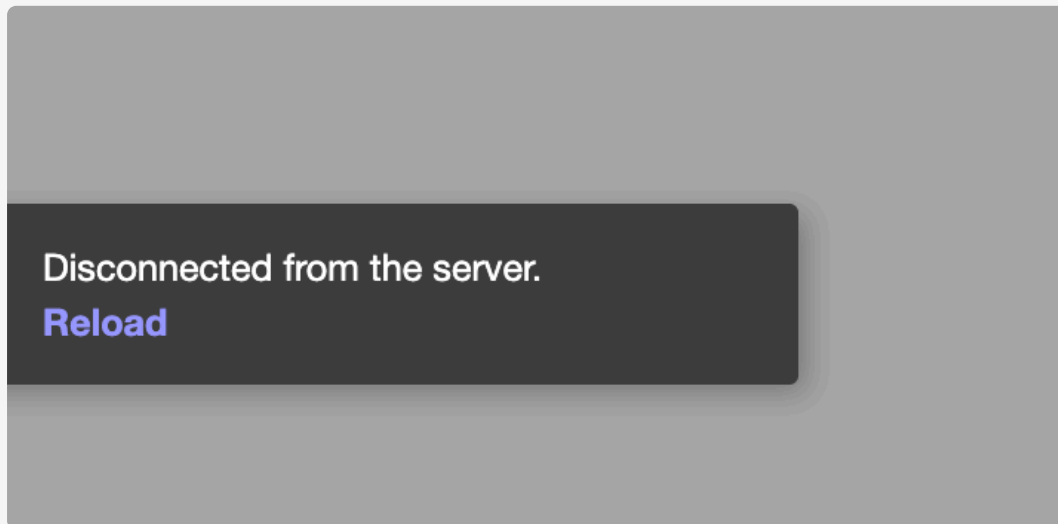
Downloading package: evaluate

# Shiny is fine!



# But Shiny has problems!

- Requires a whole live server
- Is often difficult to learn
- Slow to load
- Times out regularly



Downloading package: dplyr

# Newer approaches to interactivity

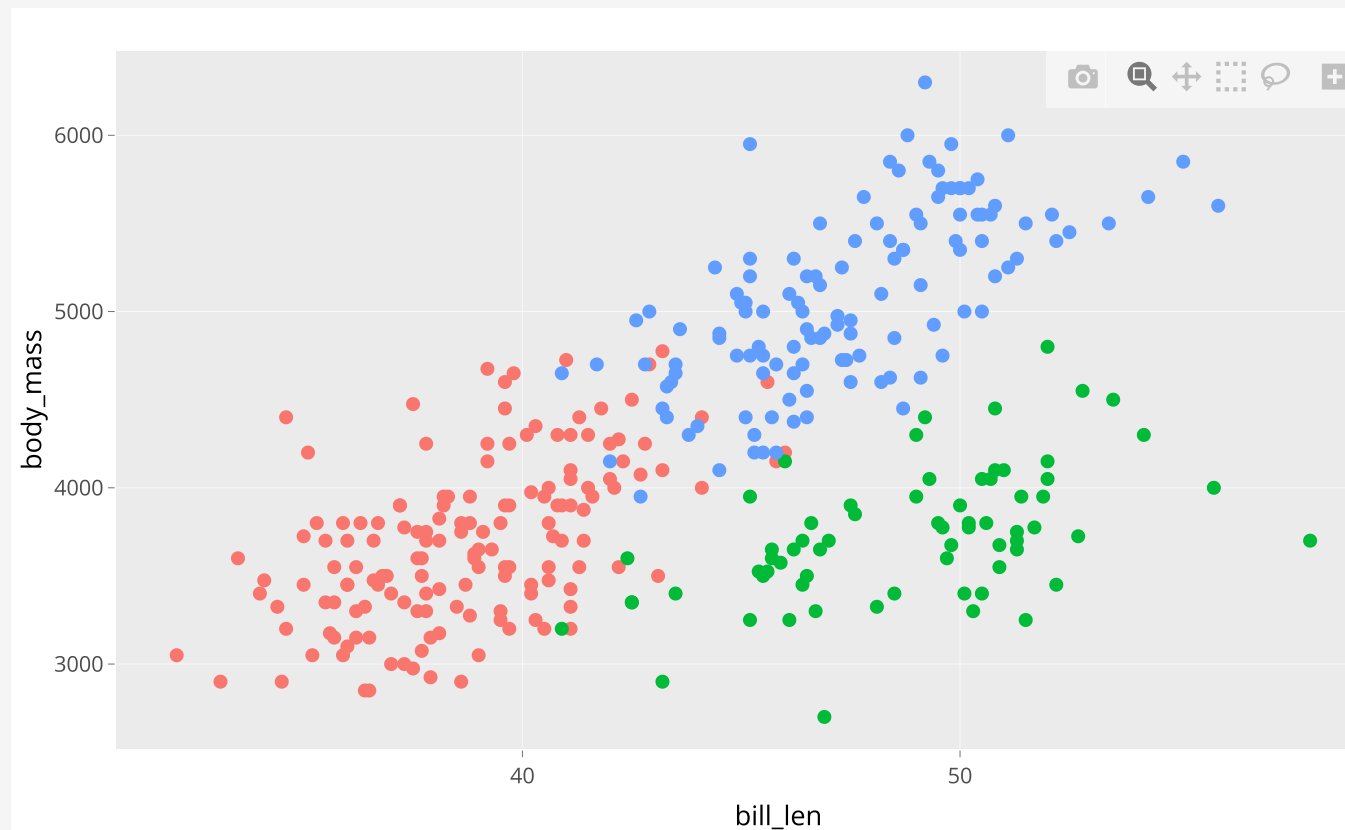
- `{plotly}` and `{ggiraph}`
  - Regular R + ggplot, but can't deal with live data
- Observable JS
  - Can deal with live data (even remote APIs), but uses a different language
- quarto-live (LITERAL MAGIC)
  - Can deal with live data *and* uses regular R

# Plotly

`plotly::ggplotly()` automatically converts ggplot objects to plotly plots

```
1 library(plotly)
2
3 basic_plot <- ggplot(
4   penguins,
5   aes(
6     x = bill_len,
7     y = body_mass,
8     color = species
9   )
10 ) +
11   geom_point()
```

```
1 ggplotly(basic_plot)
```



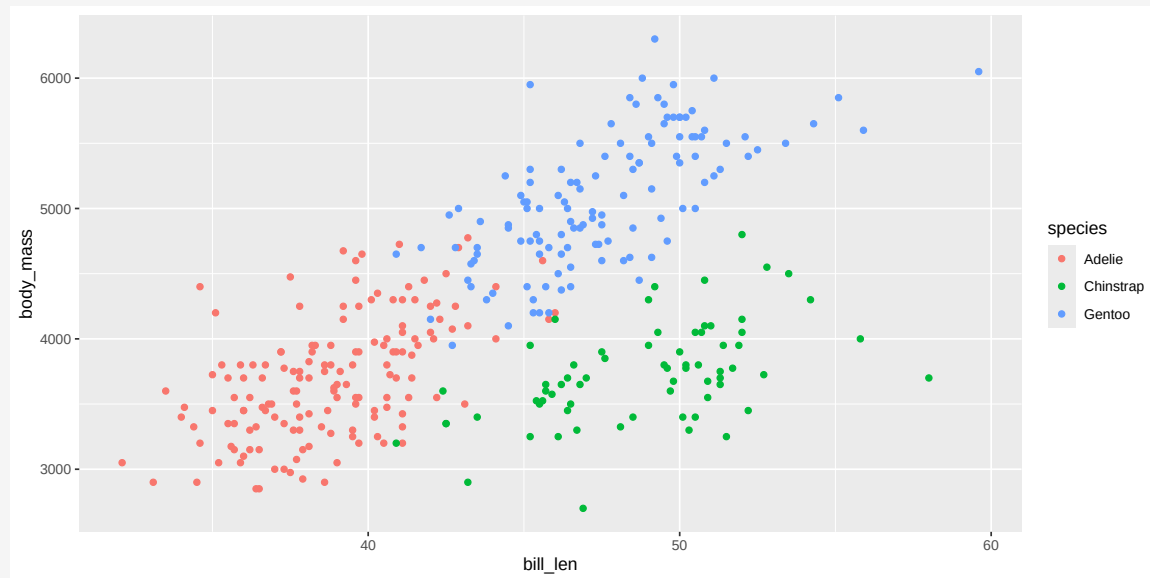
Downloading package: palmerpenguins

Documentation: <https://plotly.com/ggplot2/>

# {ggiraph}

```
1 library(ggiraph)
2
3 plot_thing <- ggplot(
4   penguins,
5   aes(
6     x = bill_len,
7     y = body_mass,
8     color = species
9   )
10 ) +
11   geom_point_interactive(
12     aes(tooltip = species,
13   )
```

```
1 girafe(ggobj = plot_thing)
```



Downloading package: ggplot2

Documentation: <https://davidgoheh.github.io/ggiraph/>

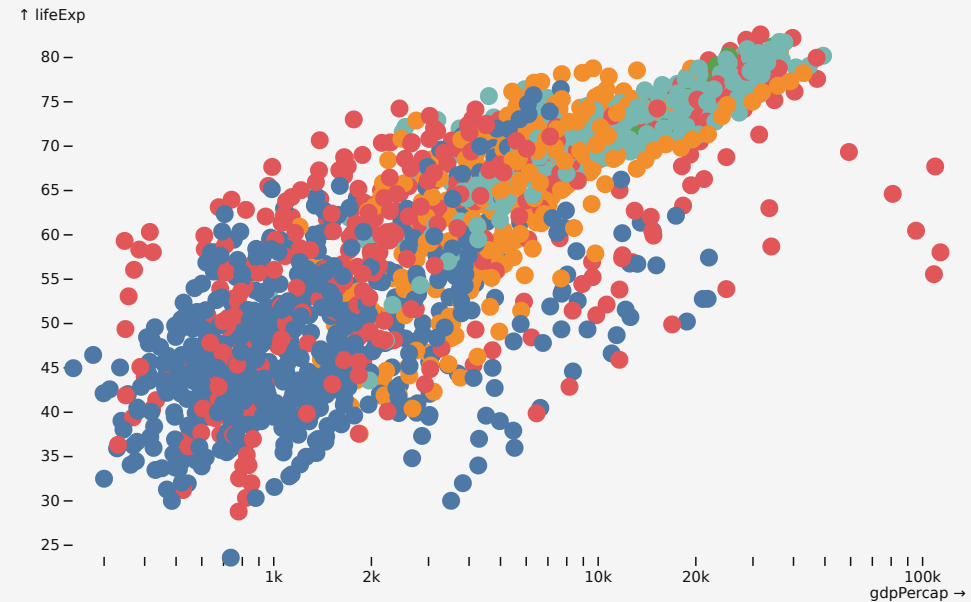
# Observable Plot

This is R:

```
1 library(gapminder)
2
3 # Make the gapminder data available to Observable JS
4 ojs_define(gapminder = gapminder)
```

This is Observable JS:

```
4 gapminder_js = transpose(gapminder)
5
6 Plot.plot({
7   x: {type: "log"},
8   marks: [
9     Plot.dot(gapminder_js, {
10       x: "gdpPercap", y: "lifeExp", fill: "continent",
11       channels: {
12         Country: d => d.country
13       },
14       tip: true
15     })
16   ]
17 })
18 )
```

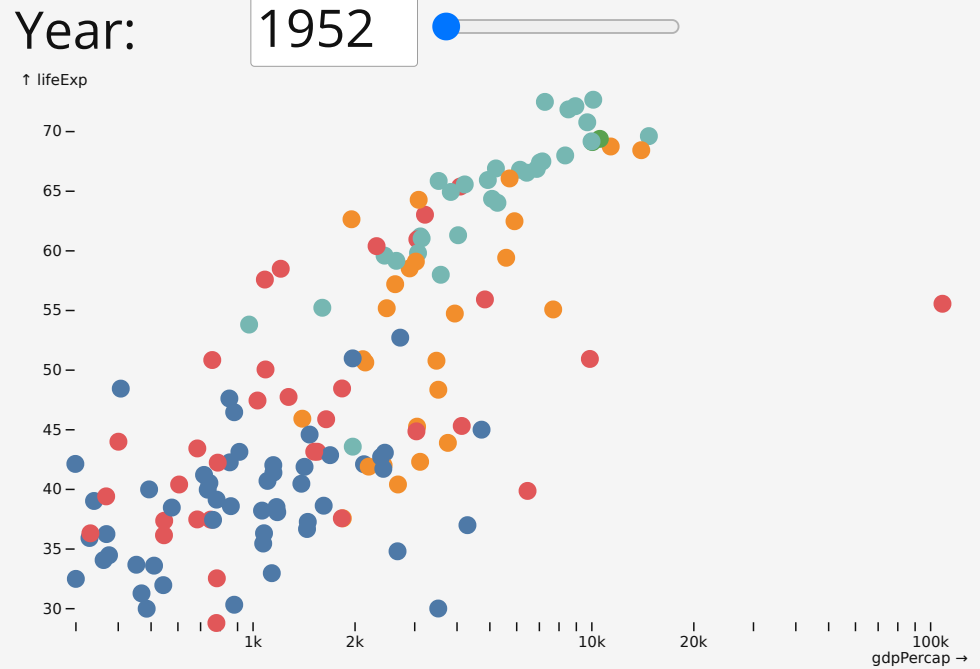


Downloading package: ggplot2



# Observable Plot

```
4 viewof current_year = Inputs.range(  
5   [1952, 2007],  
6   {value: 1952, step: 5, label: "Year:"}  
7 )  
8  
9 // Filter the data based on the selected year  
10 gapminder_filtered = gapminder_js.filter(d => d.year === current_year)  
11  
12 Plot.plot({  
13   x: {type: "log"},  
14   marks: [  
15     Plot.dot(gapminder_filtered, {  
16       x: "gdpPercap", y: "lifeExp", fill: "continent",  
17       channels: {  
18         Country: d => d.country  
19       },  
20       tip: true  
21     })  
22   ]  
23 })  
24 )
```



# Observable Plot examples

- [Quarto documentation](#)
- [Quarto OJS examples](#)
- [Using USAID data to make fancy world maps with Observable Plot + ForeignAssistance dot gov emergency backup](#)
- [Dashboard with OJS chunks](#)

# Our turn

## Play with {plotly} and {ggiraph}

Together we'll make some plots with  
`plotly::ggplotly()` and `ggiraph::girafe()`

I'll post all the final code on the course website when we're done.

10:00

# Our turn

## ~~Play with Observable~~

jk we won't do that today. It's a whole different language and takes a while to get used to. Do this on your own—it's neat!

# Dashboards

# Dashboard detour

With `{plotly}` and `{ggiraph}`, you know enough to make basic dashboards!

```
---  
title: My dashboard  
format: dashboard  
---
```

```
## Row
```

```
`{r}`  
`{r}`
```

```
## Row
```

```
`{r}`  
`{r}`
```

# Dashboard layouts

Quarto uses special Markdown syntax to place **dashboard components** in **dashboard layouts**.

```
---  
title: "Palmer Penguins"  
author: "Someone cool"  
format: dashboard  
---
```

```
## Row {height=70%}
```

```
` `{r}  
` `
```

```
## Row {height=30%}
```

```
` `{python}  
` `
```

```
` `{ojs}  
` `
```



Chart 1

Chart 2

Chart 3

# Dashboard components

- Plots
- Tables
- Value boxes
- Text and content cards



# Dynamic content

- Dashboards are still static HTML sites
  - If using live data, you're limited(ish) to OJS
- Dashboards can **connect to Shiny servers** though
  - But then the site has to live on a Shiny server

# Our turn

## Make a dashboard about penguins

Together we'll make an interactive dashboard about the **Palmer Penguins**.

I'll post all the final code on the course website when we're done.

20:00

# webR and Quarto Live

# R in the browser

**webR** is a special version of R that's compiled for Javascript and Node.js using WebAssembly



Through compiled Javascript magic, you can run R in your browser.

**Quarto Live** makes it trivial to use (and it works with Python and **Pyodide**)

# Enabling webR

Install the extension:

Terminal

```
quarto add r-wasm/quarto-live
```

Use special format and include special file (for now)

```
---  
format: live-html  
engine: knitr  
---  
  
{{< include ./_extensions/r-wasm/live/_knitr.qmd >}}
```

# Using webR

Make `webr` chunks

```
1 ```{webr}  
2 mean(1:5)  
3 plot(1:10)  
4 ```
```

R Code [↺ Start Over](#)

[▶ Run Code](#)

```
1 mean(1:5)  
2 plot(1:10)
```

# Install packages

You don't have access to *every* package on CRAN; packages have to be compiled for WebAssembly/Javascript (many/most are though!)

Packages come from the **webR public package repository**

```
1 ---
2 format: live-html
3 webr:
4   packages:
5     - dplyr
6     - palmerpenguins
7     - ggplot2
8 ---
```

# Teaching with webR

exercise.qmd

```
1 Fill in the blank to fill the density plots by species
2
3 ```{webr}
4 #| exercise: ex_1
5 ggplot(palmerpenguins::penguins, aes(x = body_mass_g)) +
6   geom_density(aes(_____), alpha = 0.7)
7 ```
```

Exercise [↻ Start Over](#)

[▶ Run Code](#)

```
1 ggplot(palmerpenguins::penguins, aes(x = body_mass_g)) +
2   geom_density(aes(_____), alpha = 0.7)
```



# More with exercises

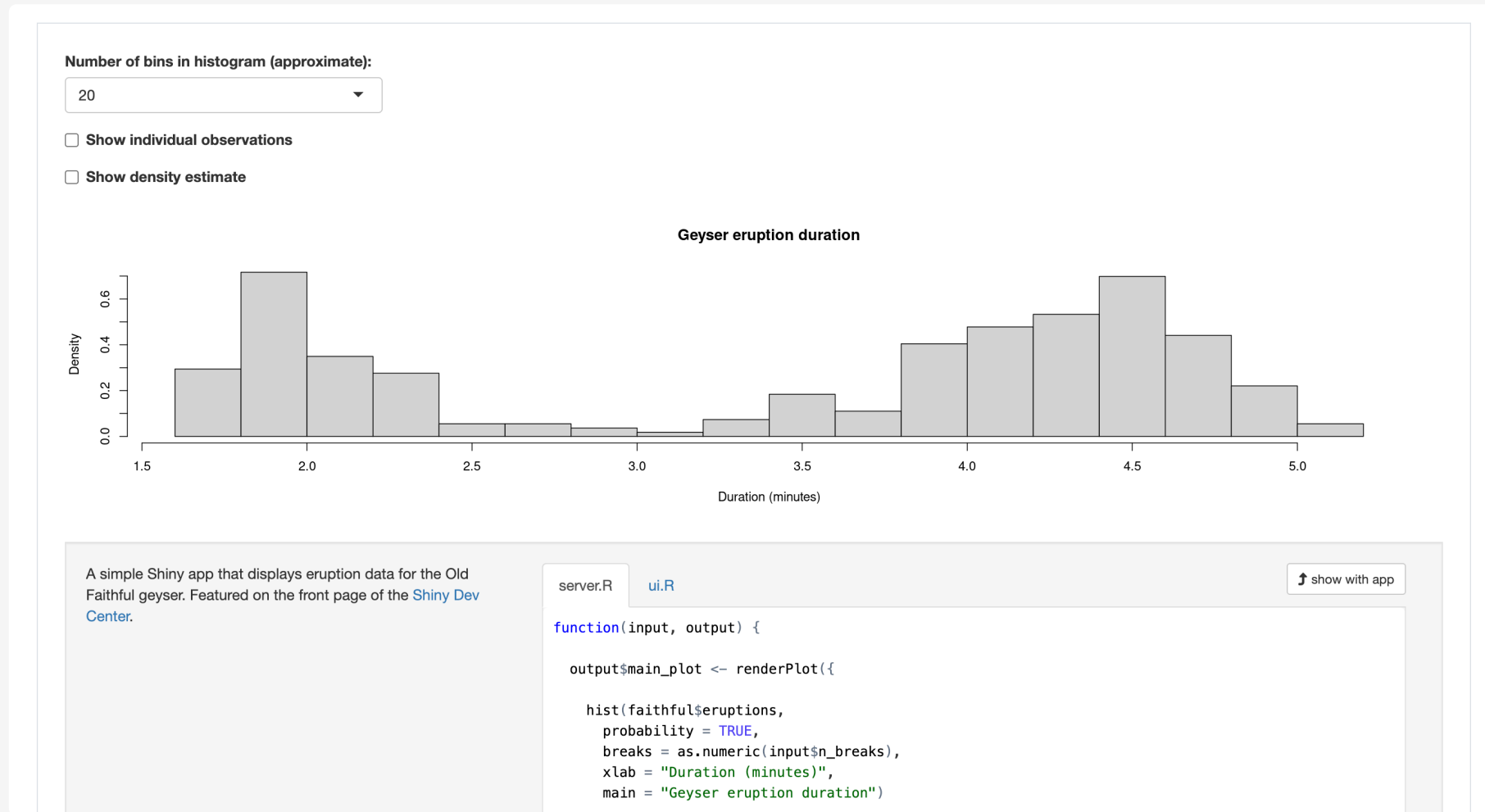
- **R for Data Science exercises and lessons**
- **“Grade” exercises and provide feedback**

**Use OJS to interact with live R**

**Replicate/replace basic Shiny apps!**

# Old Faithful app

## Classic Old Faithful Shiny example



# Old Faithful app with OJS

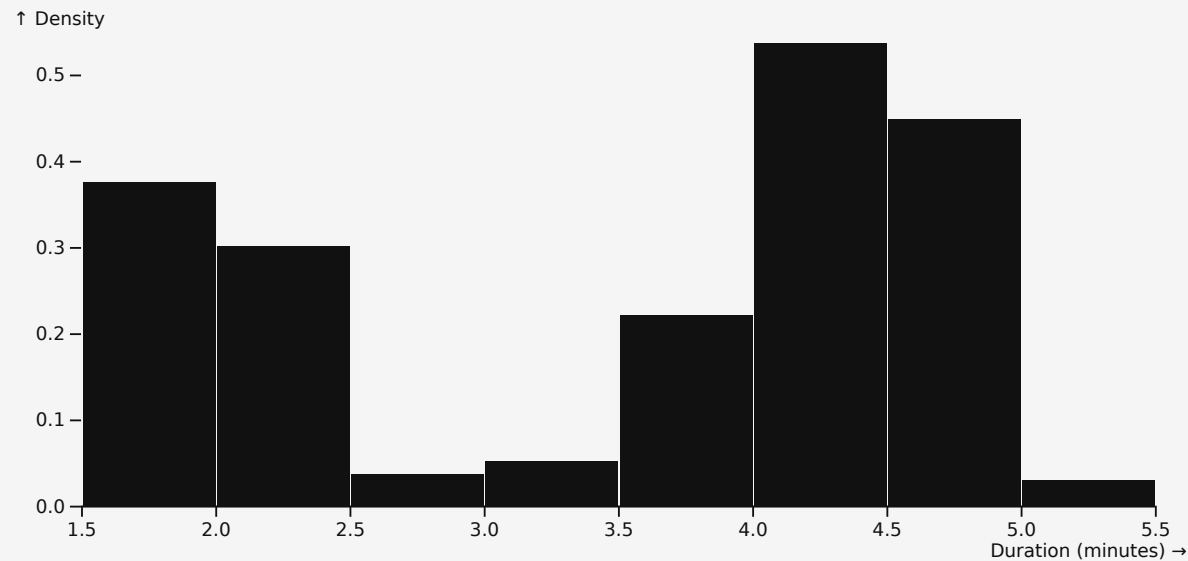
(2 hours fighting with Claude...); see website for full code

Number  
of bins

☐ Show individual observations

☐ Show density estimate

Bandwidth  
adjustment.  



# Old Faithful app with webR

*(8 minutes reading the documentation); see website for full code*

Number  
of bins

10



Show  
individual  
observations



Show  
density  
estimate



**Geyser eruption duration**

0.5



# Our turn

Together we'll do this:







1. Create a {webr} chunk that helps teach something and provides feedback
2. Recreate the **Shiny k-means example**
3. **Bonus:** Make a live ggplot plot!

I'll post all the final code on the course website when we're done.

20:00



# Course outline

-  ~~Intro to Quarto~~
-  ~~Creating basic websites~~
-  ~~Advanced website features~~
-  ~~Publishing~~
-  ~~Customization and branding~~
-  ~~Interactivity~~



**Stay curious and  
keep playing**