Week 13: Interactivity

November 18, 2026

WHY BUILD AN INTERACTIVE CHART?

An interactive chart lets the reader do things a static chart can't: **hover** for exact values, **filter** to the part they care about, **zoom** into a crowded region, **play** through time. That power has a price.

The reader can...	The cost
Hover for exact values	Doesn't work on paper or in a PDF
Filter and zoom	Hard to share (e.g., social media)
Play through time	More to build, and more to *check*
Step through a story	Hides information behind a mouse

Two ways to get there

	Packages	D3 + Claude
Tools	`plotly`, `reactable`	D3.js, prompted
Code	Short R you could write	JavaScript you don't write
Limits	Only what the package does	Anything you can describe

PACKAGES THAT WRAP JAVASCRIPT

`ggplotly()`: any ggplot, now interactive

```
library(plotly)

plot <- gapminder %>%
  filter(year == 2007) %>%
  ggplot(aes(x = gdpPercap, y = lifeExp,
             size = pop, color = continent,
             label = country)) +
  geom_point(alpha = 0.7) +
  scale_x_log10()

ggplotly(plot, tooltip = c("country", "pop"))
```

`ggplotly()` is a **translation**, and translations lose things. Tooltips show raw aesthetic names (`x`, `colour`) and unrounded numbers; subtitles, captions, and annotations can move or vanish; fonts and legends don't always survive. When it gets messy, use plotly directly.

`reactable()`: interactive tables

```
library(reactable)

gapminder %>%
  reactable(
    searchable = TRUE, filterable = TRUE,
    highlight = TRUE, defaultPageSize = 5
  )
```

Save it, then embed it

```
htmlwidgets::saveWidget(
  ggplotly(plot),
  file = file.path('figs', 'gapminder.html'))
```

Then put it in a `.qmd` with an iframe, the same way you'll embed a D3 chart.

CUSTOM CHARTS: D3 + CLAUDE

D3.js is the JavaScript library behind most custom interactive graphics (The Pudding, NYT, The Guardian). It can build a custom scale, a bespoke animation, linked views, or a layout no package offers. The price is code: dozens of lines for what `geom_point()` does in one.

You don't write D3. You prompt, and iterate. You decide what the chart should show, describe it precisely, and check that it's right. Claude writes the JavaScript and revises until it looks right.

> **Verify against the data, not the code.** You can't check D3 by reading it, so build a static chart first and check it carefully. That static chart is your reference: the interactive version has to show the same numbers.

The workflow

1. **Make a static chart** and verify it
2. **Sketch** what the interactive version should do: what moves, what hovers, what filters
3. **Prompt** Claude with your code, your data, and your sketch
4. **Iterate** until it looks right
5. **Verify** against the static chart and the data

A prompt that works

> Here is my static chart (code below) and my data. Build an animated, interactive version in D3: [describe your interaction]. Add a tooltip, labeled axes, and a title that states the main takeaway. Put it in a single self-contained file called `d3-chart.html`, loading D3 from a CDN. Embed the data directly in the file.

Example: checking the gapminder bubble chart

The static 2007 chart says there are 142 countries, China and India are the biggest bubbles, and life expectancy runs from about 40 to 83 years. So in the D3 version:

1. Set the slider to **2007**: does it match the static chart?
2. **Hover** over China: do the values match the data?
3. **Count** the bubbles: 142?
4. Click **everything**: first and last year, each filter

```
gapminder %>%
  filter(country == "China", year == 2007)

gapminder %>%
  filter(year == 2007) %>%
  nrow()
```

Checks for any chart you didn't write

Check	Ask
The numbers	Hover several points: do they match the data?
The axes	Labels, units, ranges? Still works when resized?
The claim	Does the data support exactly what the title says?
The interaction	Does every control work, at the first and last year?

If something is off, tell Claude exactly what's wrong and give it the right value from your data.

Gotchas

Gotcha	Why
Embed the data	Browsers block a local `.html` from reading a local CSV, so the chart is blank
Load D3 from a CDN	The file links to a hosted copy of D3, so it needs internet
Check the console	Right-click → Inspect → Console shows red errors
Same folder, committed	The iframe can only find a file that's next to the `.qmd`

```
<iframe src="d3-chart.html" width="100%"
  height="500" style="border:none;"></iframe>
```

SHINY APPS

A Shiny app is a web page (the **UI**) connected to a computer running a live R session (the **server**). It all lives in one file, `app.R`. Type `shiny` in Positron and the snippet fills in the skeleton:

```
library(shiny)
ui <- fluidPage()
server <- function(input, output, session) {}
shinyApp(ui, server)
```

Run it with the **Run App** button, or `shiny::runApp()` with the path to the app file.

Building one

In the `ui`	In the `server`
1. Pick a layout function	1. Make outputs with `render*()`
2. Add input widgets	2. Link outputs with `output$<id>`
3. Add `*Output()` functions	3. Link inputs with `input$<id>`

Layouts

Layout functions nest like Matryoshka dolls, so watch your parentheses and commas.

Layout	What it makes
`fluidPage()`	The usual top level
`sidebarLayout()`	`sidebarPanel()` beside a `mainPanel()`
`navbarPage()`	A page of tabs, one `tabPanel()` each
`navbarMenu()`	A dropdown of tabs inside a navbar

Inputs collect, outputs display

Input widget	Collects
`sliderInput()`	A number from a range
`radioButtons()`	One choice from a list
`selectInput()`	A choice from a dropdown
`textInput()`	Free text

In the `ui`	In the `server`
`plotOutput()`	`renderPlot()`
`textOutput()`	`renderText()`
`dataTableOutput()`	`renderDataTable()`
`imageOutput()`	`renderImage()`

Linking it together

The `inputId` in the UI becomes `input$xvar` in the server; the `outputId` becomes `output$mpg_plot`.

```
ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(
      radioButtons(
        inputId = "xvar",
        label = "Select the x-axis variable:",
        choices = c("hwy", "cty", "displ"))
    ),
    mainPanel(
      plotOutput(outputId = "mpg_plot")
    )
  )
)

server <- function(input, output, session) {
  output$mpg_plot <- renderPlot({
    ggplot(mpg) +
      geom_point(aes_string(
        x = input$xvar, y = "hwy",
        color = "class"))
  })
}
```

Common mistakes

Symptom	Likely cause
App won't start	Missing comma or parenthesis in the UI
Plot never appears	`outputId` doesn't match `output$<id>`
Input does nothing	Server never uses `input$<id>`
Output is blank	Code isn't inside a `render*()`

Deploy to Posit Connect Cloud (free)

Use **Posit Publisher**, built into Positron:

1. Click the **Posit Publisher** icon in the left sidebar
2. Under **Credentials**, click **+** → **Posit Connect Cloud** and sign in
3. Click **+** to add a deployment, pick `app.R`, check the project files, and click **Deploy Your Project**

Extras

- **Themes:** add `theme = shinytheme("sandstone")` as the first argument of the top layout function (from the shinythemes package)
- **Fancier widgets:** the shinyWidgets package
- **Reference:** the cheatsheet and Mastering Shiny

CHOOSING A TOOL

You want…	Use
Hover and zoom on a ggplot you have	`ggplotly()`
A searchable, sortable table	`reactable()`
A custom animation, scale, or layout	D3 + Claude
Readers to run R (filter, refit, recompute)	Shiny

The first three save to plain `.html` files that work anywhere, including a Quarto site on GitHub Pages. A Shiny app needs a live R session behind it, so it has to be deployed to a server like Connect Cloud.

BEFORE NEXT CLASS

Next class is **Dec 2** (no class Nov 25, Thanksgiving).

- **HW12: Story Telling**, due **Nov 30**.
- **Final Report**, due **Dec 6** (17% of grade).

You should be able to:

- Say when an interactive chart is worth its cost over a static one.
- Make a ggplot interactive with `ggplotly()` and control its tooltip.
- Build an interactive table with `reactable()`.
- Prompt Claude for a D3 chart and verify it against a static chart and the data.
- Embed a saved `.html` chart in a `.qmd` with an iframe.
- Build a Shiny app that links an input to a plot.
- Deploy a Shiny app to Posit Connect Cloud from Positron.