Reproducible Shiny apps with **`shinymeta`**

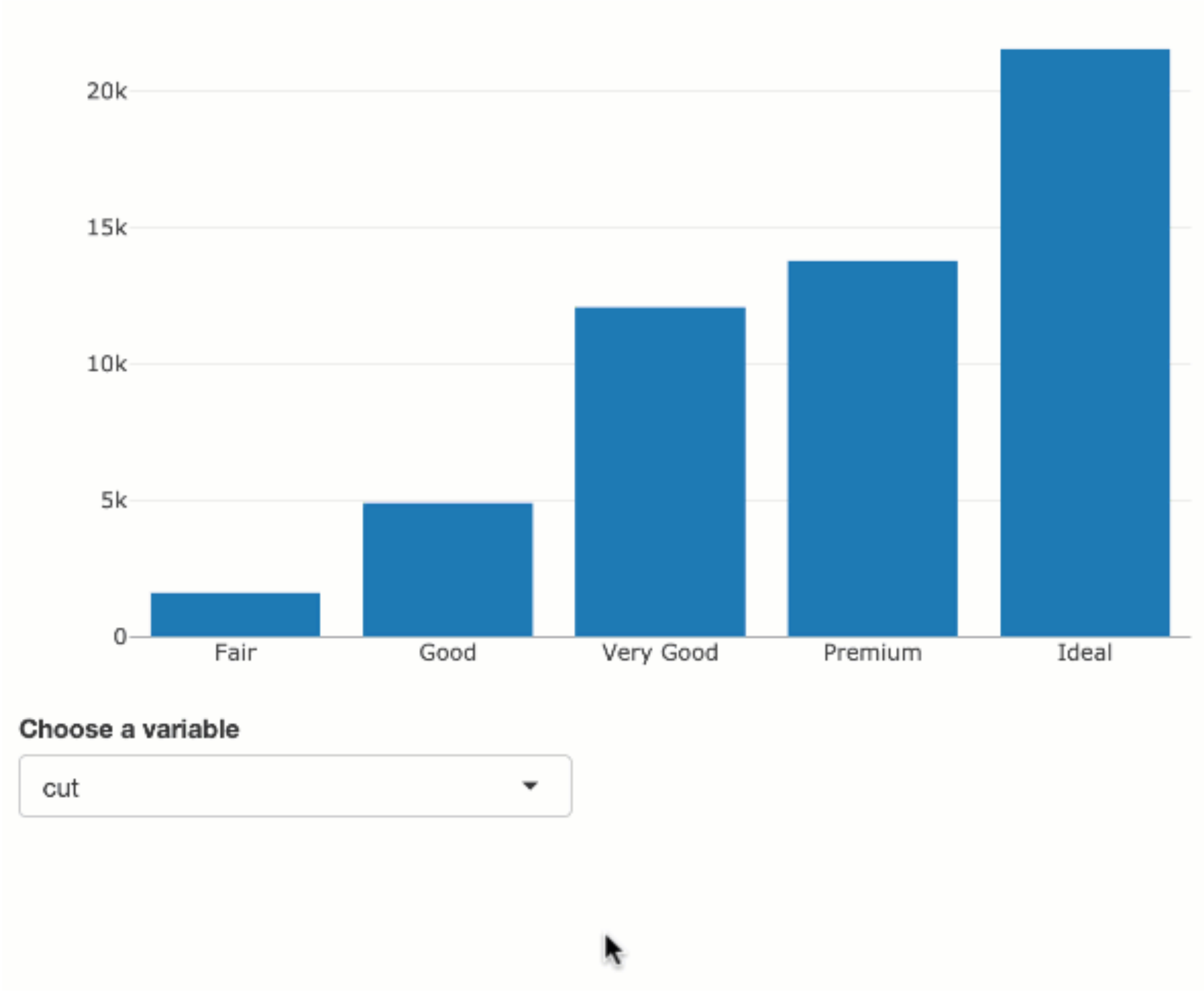
Barret Schloerke
Shiny Software Engineer, RStudio
@schloerke

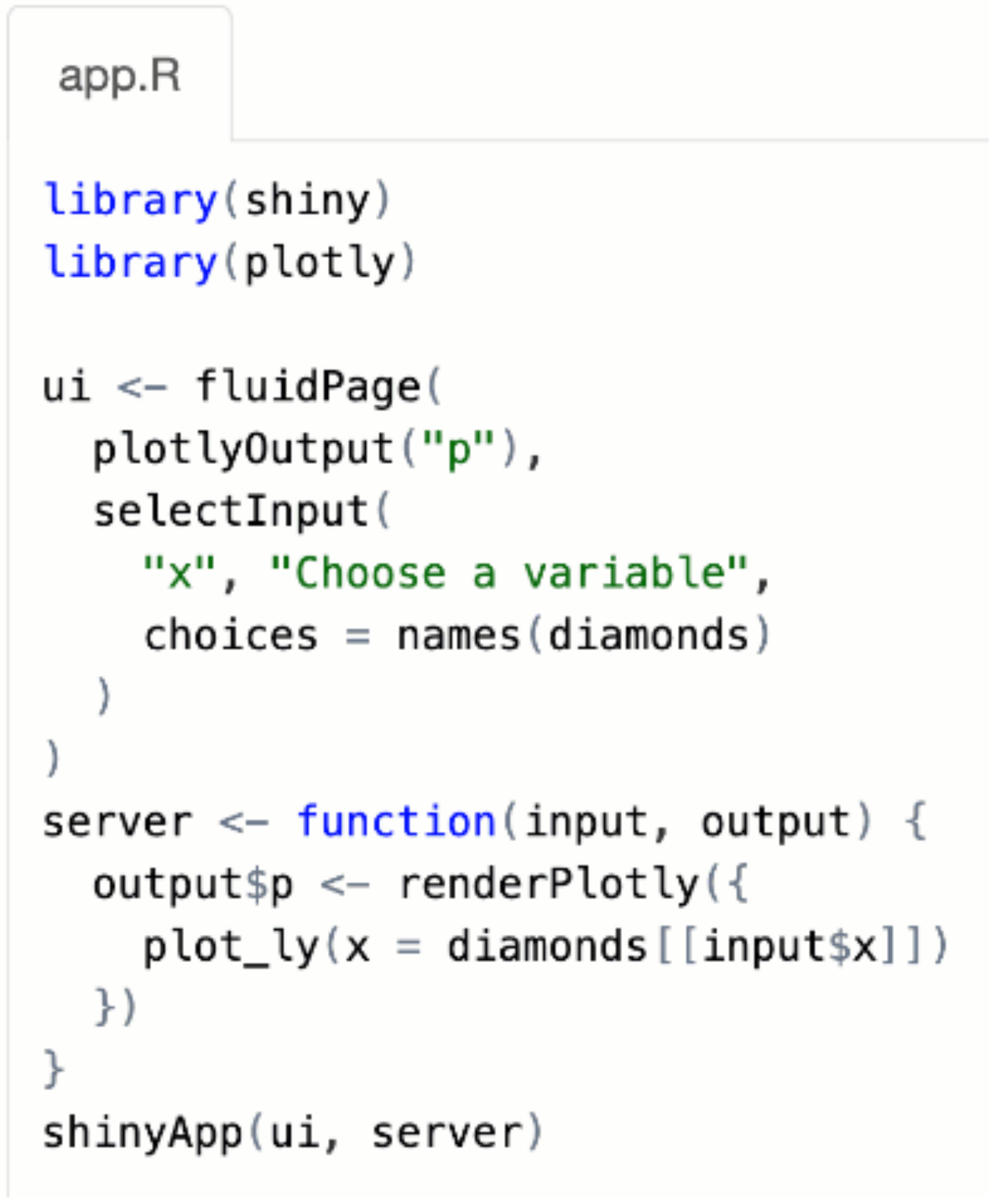
Slides: bit.ly/shinymeta-abacus-2019

Authored by Joe Cheng & Carson Sievert

Shiny: **Interactive** webapps in R

- Easily turn your R code into an interactive GUI.
- Allow users to **quickly explore** different parameters, models/algorithms, other information





```r
library(shiny)
library(plotly)

ui <- fluidPage(
  plotlyOutput("p"),
  selectInput(
    "x", "Choose a variable",
    choices = names(diamonds)
  )
)
server <- function(input, output) {
  output$p <- renderPlotly({
    plot_ly(x = diamonds[[input$x]])
  })
}
shinyApp(ui, server)
```

Interactivity is great, but **reproducibility suffers**

- Reproducing results is *possible* by replicating user events (or bookmarking), but results are **locked behind a GUI**
- Even if you can view the app's source code,
  the **domain logic is intertwined** with Shiny code
    - Methodology is less transparent
    - Harder to verify results are 'correct'

The goal: interactivity + reproducible code

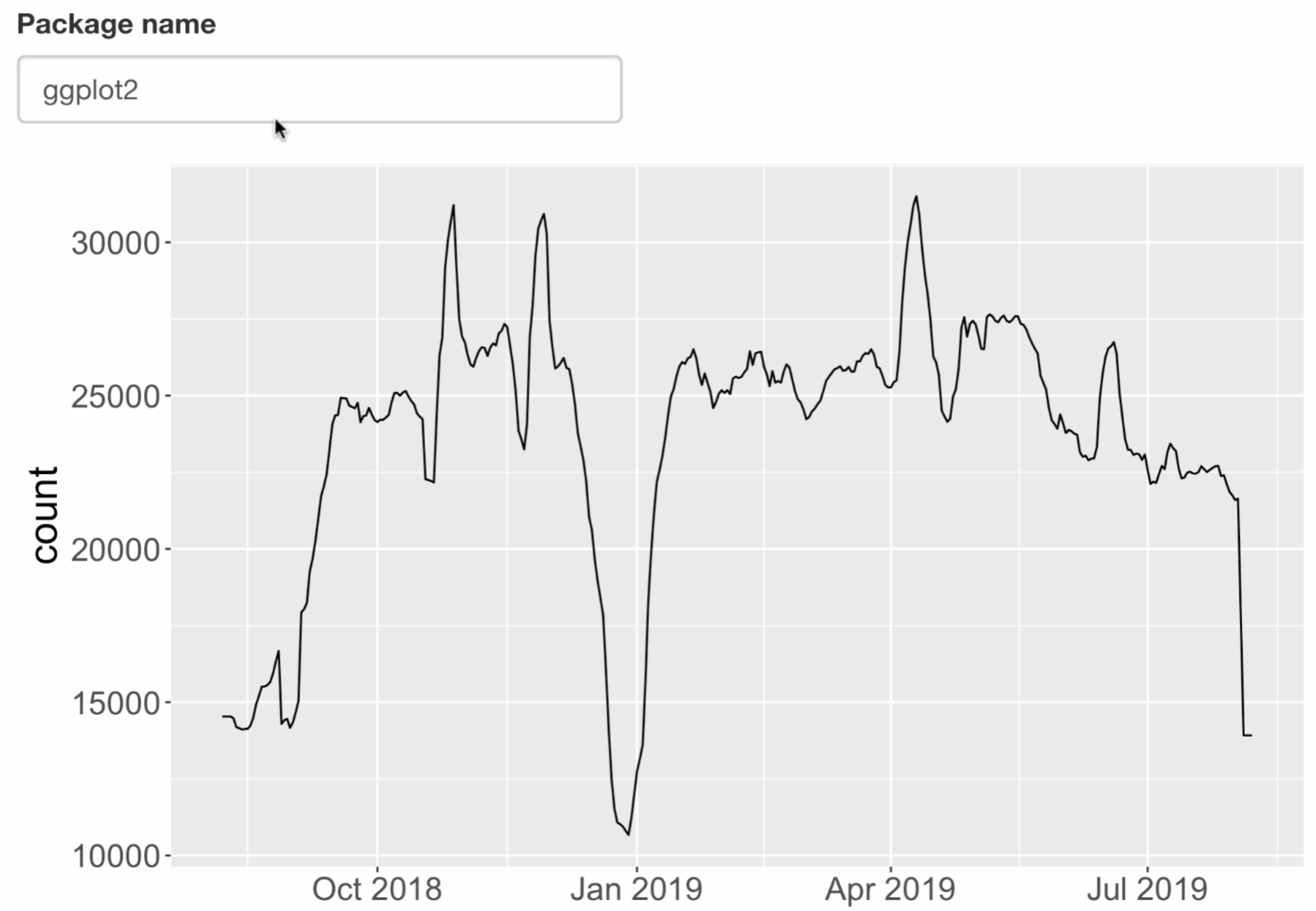
1. Find interesting results via interactive app
2. Export domain logic, on demand
   - As reproducible code/results that are independent of Shiny

`shinymeta`:
tools for capturing logic in a Shiny app and exposing it as independent code

Install:

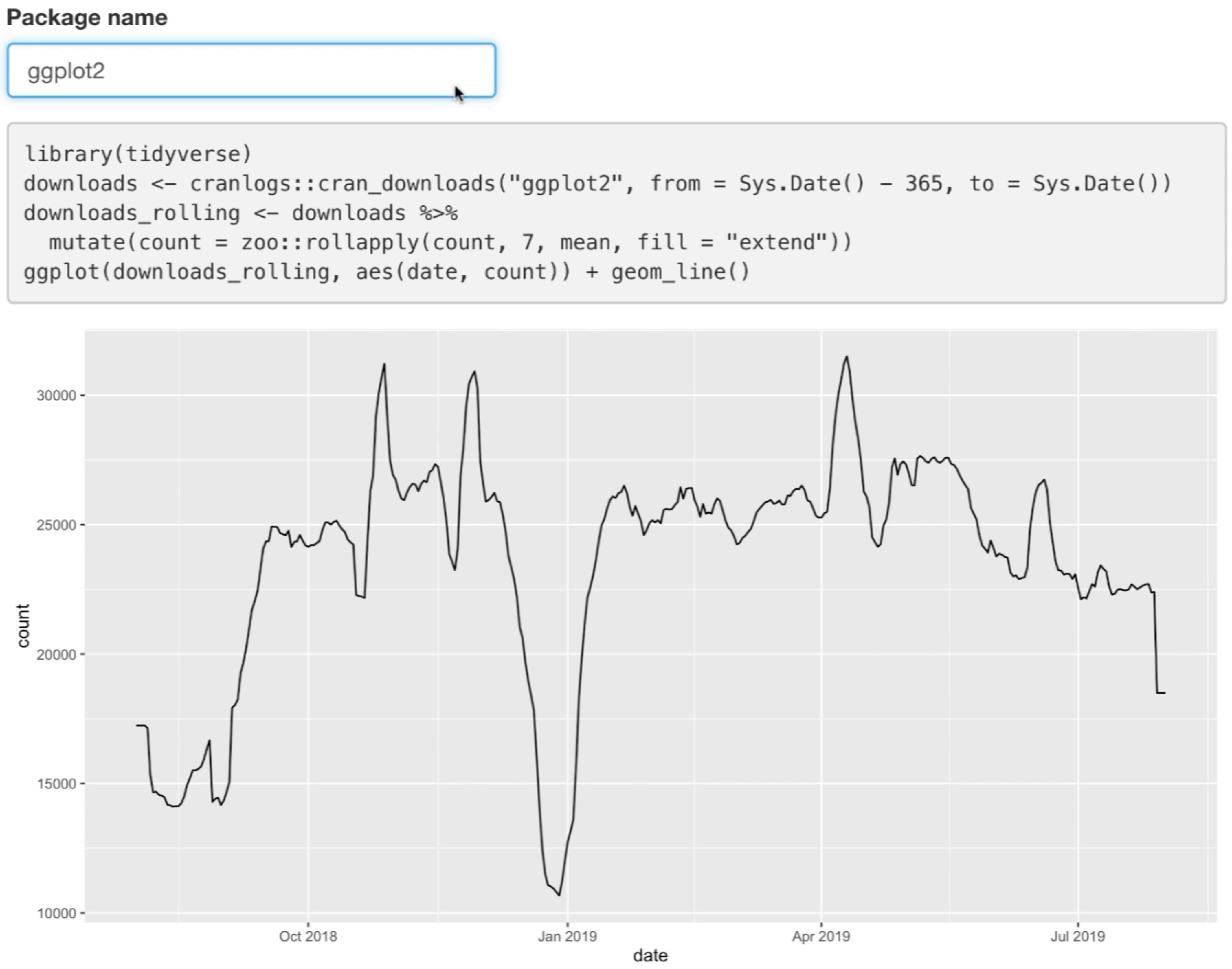
```
devtools::install_github("rstudio/shinymeta")
```

Example: CRAN downloads


Package name

ggplot2

Goal: reproducible plot code


Package name

ggplot2

```
library(tidyverse)
downloads <- cranlogs::cran_downloads("ggplot2", from = Sys.Date() - 365, to = Sys.Date())
downloads_rolling <- downloads %>%
  mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))
ggplot(downloads_rolling, aes(date, count)) + geom_line()
```

```r
library(shiny)
library(tidyverse)

ui <- fluidPage(
  textInput("package", "Package name", value = "ggplot2"),
  plotOutput("plot")
)

server <- function(input, output, session) {

  downloads <- reactive({
    cranlogs::cran_downloads(
      input$package,
      from = Sys.Date() - 365,
      to = Sys.Date()
    )
  })

  downloads_rolling <- reactive({
    validate(need(sum(downloads()$count) > 0, "Input a valid package name"))

    downloads() %>%
      mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))
  })

  output$plot <- renderPlot({
    ggplot(downloads_rolling(), aes(date, count)) + geom_line()
  })
}

shinyApp(ui, server)
```

Step 1: Identify domain logic

```
server <- function(input, output, session) {

  downloads <- reactive({
    cranlogs::cran_downloads(
      input$package,
      from = Sys.Date() - 365,
      to = Sys.Date()
    )
  })

  downloads_rolling <- reactive({
    validate(need(sum(downloads()$count) > 0, "Input a valid package name"))

    downloads() %>%
      mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))
  })

  output$plot <- renderPlot({
    ggplot(downloads_rolling(), aes(date, count)) + geom_line()
  })
}
```

Step 1: Identify domain logic

```r
server <- function(input, output, session) {

  downloads <- reactive({
    cranlogs::cran_downloads(
      input$package,
      from = Sys.Date() - 365,
      to = Sys.Date()
    )
  })

  downloads_rolling <- reactive({
    validate(need(sum(downloads()$count) > 0, "Input a valid package name"))

    downloads() %>%
      mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))
  })

  output$plot <- renderPlot({
    ggplot(downloads_rolling(), aes(date, count)) + geom_line()
  })
}
```

Only applies to Shiny, don't export it!

Step 1: Identify domain logic

```
server <- function(input, output, session) {

  downloads <- reactive({
    cranlogs::cran_downloads(
      input$package,
      from = Sys.Date() - 365,
      to = Sys.Date()
    )
  })

  downloads_rolling <- reactive({
    validate(need(sum(downloads()$count) > 0, "Input a valid package name"))

    downloads() %>%
      mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))
  })

  output$plot <- renderPlot({
    ggplot(downloads_rolling(), aes(date, count)) + geom_line()
  })
}
```

Step 1: Capture domain logic

```r
server <- function(input, output, session) {

  downloads <- metaReactive({
    cranlogs::cran_downloads(
      input$package,
      from = Sys.Date() - 365,
      to = Sys.Date()
    )
  })

  downloads_rolling <- metaReactive2({
    validate(need(sum(downloads()$count) > 0, "Input a valid package name"))

    metaExpr({
      downloads() %>%
        mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))
    })
  })

  output$plot <- metaRender(renderPlot, {
    ggplot(downloads_rolling(), aes(date, count)) + geom_line()
  })
}
```

reactive becomes metaReactive

render functions must be wrapped in metaRender

Step 1: Capture domain logic

```
server <- function(input, output, session) {

  downloads <- metaReactive({
    cranlogs::cran_downloads(
      input$package,
      from = Sys.Date() - 365,
      to = Sys.Date()
    )
  })

  downloads_rolling <- metaReactive2({
    validate(need(sum(downloads()$count) > 0, "Input a valid package name"))

    metaExpr({
      downloads() %>%
        mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))
    })
  })

  output$plot <- metaRender(renderPlot, {
    ggplot(downloads_rolling(), aes(date, count)) + geom_line()
  })
}
```

Capture domain logic with metaExpr inside meta***2 variants

Step 2: Identify reactive reads

```
server <- function(input, output, session) {

  downloads <- metaReactive({
    cranlogs::cran_downloads(
      input$package,
      from = Sys.Date() - 365,
      to = Sys.Date()
    )
  })

  downloads_rolling <- metaReactive2({
    validate(need(sum(downloads()$count) > 0, "Input a valid package name"))

    metaExpr({
      downloads() %>%
        mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))
    })
  })

  output$plot <- metaRender(renderPlot, {
    ggplot(downloads_rolling(), aes(date, count)) + geom_line()
  })
}
```

Step 2: Mark reactive reads

```
server <- function(input, output, session) {

  downloads <- metaReactive({
    cranlogs::cran_downloads(
      ..(input$package),
      from = Sys.Date() - 365,
      to = Sys.Date()
    )
  })

  downloads_rolling <- metaReactive2({
    validate(need(sum(downloads()$count) > 0, "Input a valid package name"))

    metaExpr({
      ..(downloads()) %>%
        mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))
    })
  })

  output$plot <- metaRender(renderPlot, {
    ggplot(..(downloads_rolling()), aes(date, count)) + geom_line()
  })
}
```

Step 2: Mark reactive reads

```
server <- function(input, output, session) {

  downloads <- metaReactive({
    cranlogs::cran_downloads(
      ..(input$package),
      from = Sys.Date() - 365,
      to = Sys.Date()
    )
  })

  downloads_rolling <- metaReactive2({
    validate(need(sum(downloads()$count) > 0, "Input a valid package name"))

    metaExpr({
      ..(downloads()) %>%
        mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))
    })
  })

  output$plot <- metaRender(renderPlot, {
    ggplot(..(downloads_rolling()), aes(date, count)) + geom_line()
  })
}
```

Replaced by a static value or name (when code is generated)

Step 2: Mark reactive reads

```r
server <- function(input, output, session) {

  downloads <- metaReactive({
    cranlogs::cran_downloads(
      ..(input$package),
      from = Sys.Date() - 365,
      to = Sys.Date()
    )
  })

  downloads_rolling <- metaReactive2({
    validate(need(sum(downloads()$count) > 0, "Input a valid package name"))

    metaExpr({
      ..(downloads()) %>%
        mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))
    })
  })

  output$plot <- metaRender(renderPlot, {
    ggplot(..(downloads_rolling()), aes(date, count)) + geom_line()
  })
}
```

Step 2: Mark reactive reads

```
server <- function(input, output, session) {

  downloads <- metaReactive({
    cranlogs::cran_downloads(
      ..(input$package),
      from = ..(format(Sys.Date() - 365)),
      to = Sys.Date()
    )
  })

  downloads_rolling <- metaReactive2({
    validate(need(sum(downloads()$count) > 0, "Input a valid package name"))

    metaExpr({
      ..(downloads()) %>%
        mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))
    })
  })

  output$plot <- metaRender(renderPlot, {
    ggplot(..(downloads_rolling()), aes(date, count)) + geom_line()
  })
}
```

Pro tip: use ..() to return the *value of* an expression

Step 3: Generate code with `expandChain()`

```
server <- function(input, output, session) {

  output$code <- renderPrint({
    expandChain(output$plot)
  })

  downloads <- metaReactive({
    cranlogs::cran_downloads(
      ..(input$package),
      from = ..(format(Sys.Date() - 365)),
      to = Sys.Date()
    )
  })

  downloads_rolling <- metaReactive2({
    validate(need(sum(downloads()$count) > 0, "Input a valid package name"))

    metaExpr({
      ..(downloads()) %>%
        mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))
    })
  })

  output$plot <- metaRender(renderPlot, {
    ggplot(..(downloads_rolling()), aes(date, count)) + geom_line()
  })
}
```

Step 3: Generate code with `expandChain()`

```
> expandChain(output$plot)

downloads <-
  cranlogs::cran_downloads(
    ..(input$package),
    from = ..(format(Sys.Date() - 365)),
    to = Sys.Date()
  )

downloads_rolling <-
  ..(downloads()) %>%
    mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))

ggplot(..(downloads_rolling()), aes(date, count)) + geom_line()
```

expandChain() returns the relevant domain logic

Step 3: Generate code with `expandChain()`

```
> expandChain(output$plot)

downloads <-
  cranlogs::cran_downloads(
    ..(input$package),
    from = ..(format(Sys.Date() - 365)),
    to = Sys.Date()
  )

downloads_rolling <-
  ..(downloads()) %>%
    mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))

ggplot(..(downloads_rolling()), aes(date, count)) + geom_line()
```

Step 3: Generate code with `expandChain()`

```
> expandChain(output$plot)

downloads <-
  cranlogs::cran_downloads(
    "shiny",
    from = ..(format(Sys.Date() - 365)),
    to = Sys.Date()
  )

downloads_rolling <-
  downloads %>%
    mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))

ggplot(downloads_rolling, aes(date, count)) + geom_line()
```

Marked reactive reads are replaced with a suitable value or name

Step 3: Generate code with `expandChain()`

```
> expandChain(output$plot)

downloads <-
  cranlogs::cran_downloads(
    "shiny",
    from = ..(format(Sys.Date() - 365)),
    to = Sys.Date()
  )

downloads_rolling <-
  downloads %>%
    mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))

ggplot(downloads_rolling, aes(date, count)) + geom_line()
```

Other code wrapped in ..() is evaluated (i.e. unquoted)

Step 3: Generate code with `expandChain()`

```
> expandChain(output$plot)

downloads <-
  cranlogs::cran_downloads(
    "shiny",
    from = "2019-11-11",
    to = Sys.Date()
  )

downloads_rolling <-
  downloads %>%
    mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))

ggplot(downloads_rolling, aes(date, count)) + geom_line()
```

This allows dynamic results to be 'hard coded'

Step 3: Generate code with `expandChain()`

```
> expandChain(quote(library(tidyverse)), output$plot)

library(tidyverse)

downloads <-
  cranlogs::cran_downloads(
    "shiny",
    from = "2019-11-11",
    to = Sys.Date()
  )

downloads_rolling <-
  downloads %>%
    mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))

ggplot(downloads_rolling, aes(date, count)) + geom_line()
```

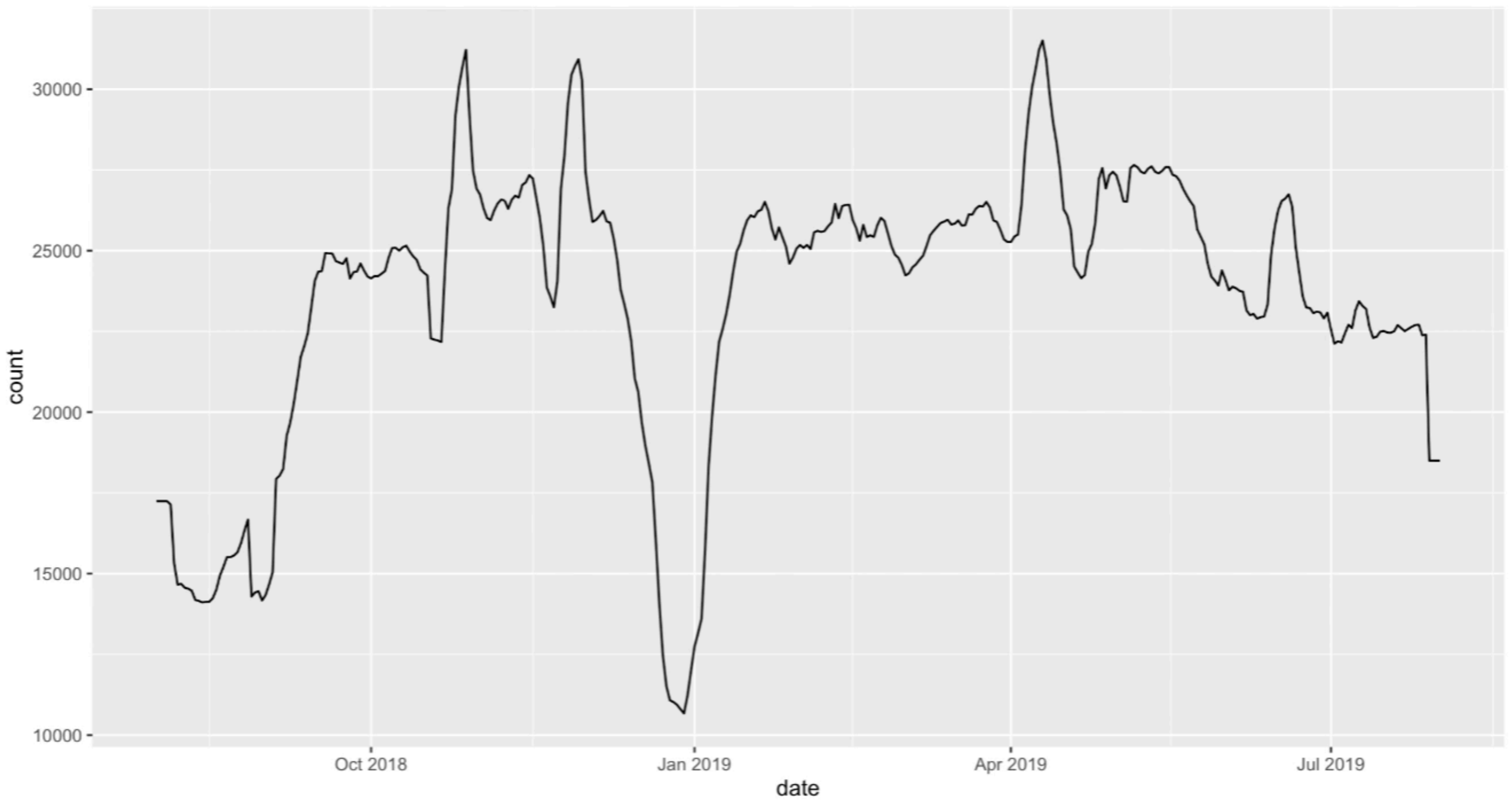
Add quoted code to supply ‘setup code’

TaDa!!

Package name

ggplot2

```
library(tidyverse)
downloads <- cranlogs::cran_downloads("ggplot2", from = Sys.Date() - 365, to = Sys.Date())
downloads_rolling <- downloads %>%
  mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))
ggplot(downloads_rolling, aes(date, count)) + geom_line()
```

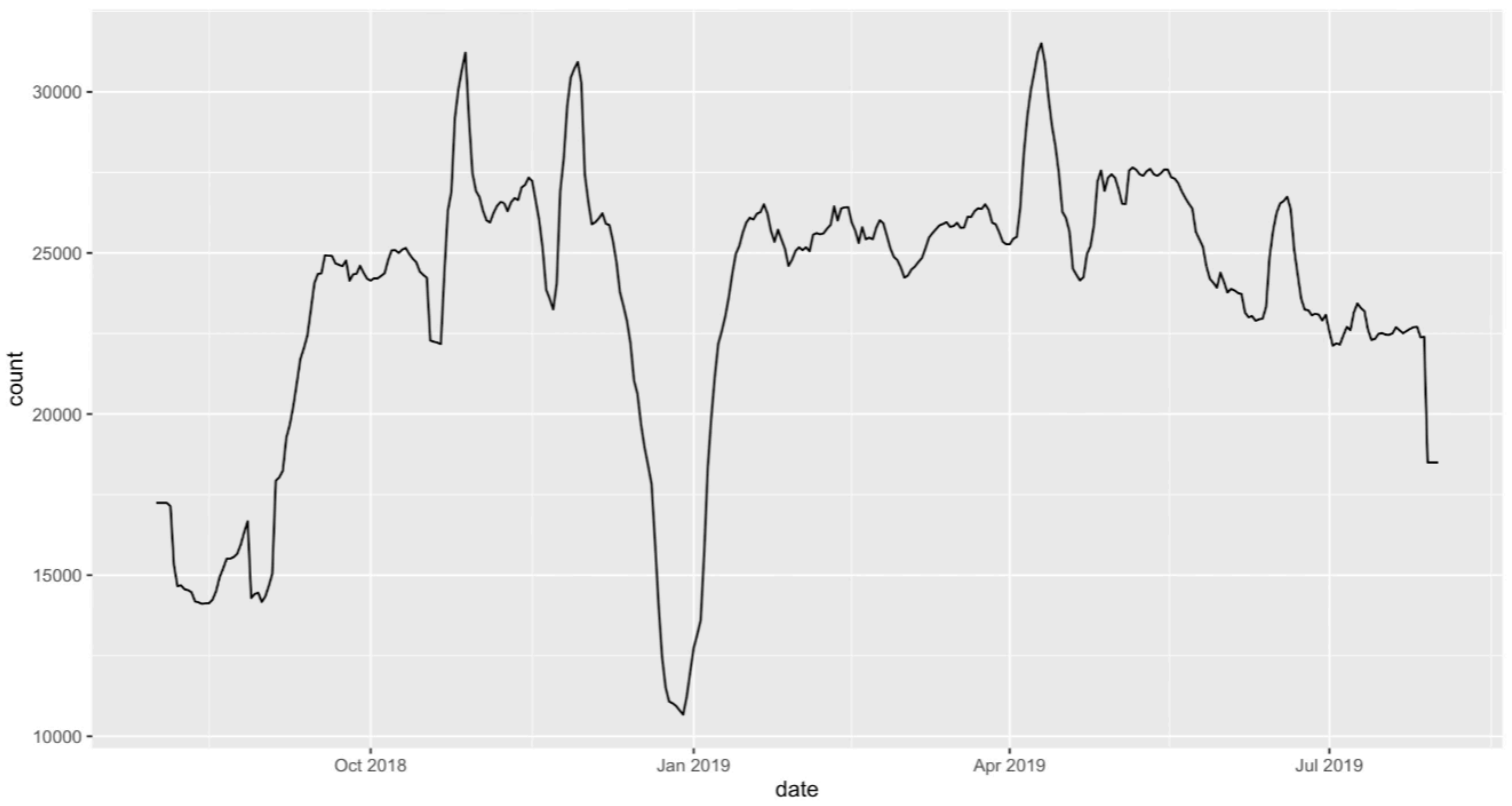


… but I don't need the code ***yet***

Package name

ggplot2

```
library(tidyverse)
downloads <- cranlogs::cran_downloads("ggplot2", from = Sys.Date() - 365, to = Sys.Date())
downloads_rolling <- downloads %>%
  mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))
ggplot(downloads_rolling, aes(date, count)) + geom_line()
```



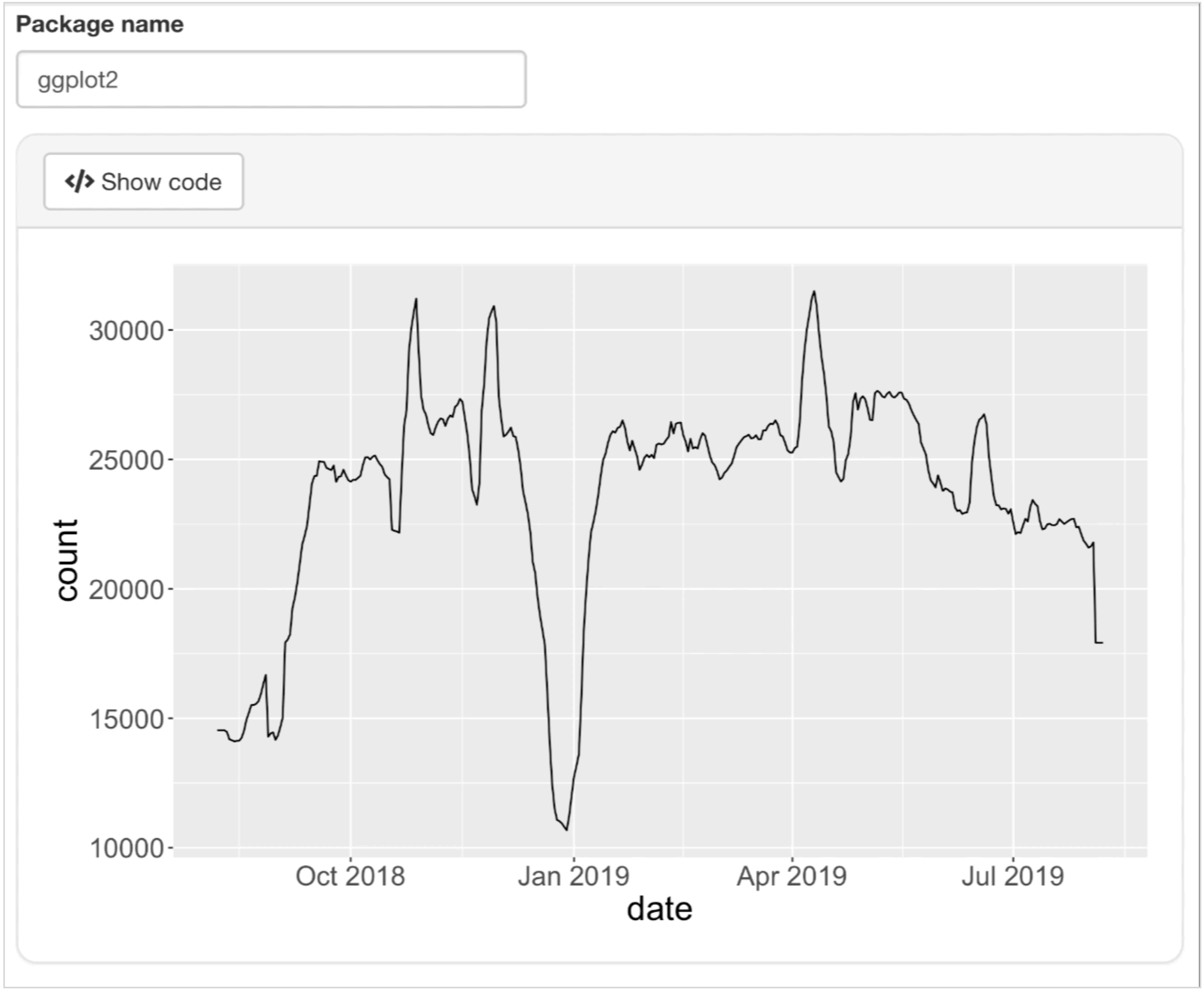
Better ways to distribute code (& results)

On Button click:

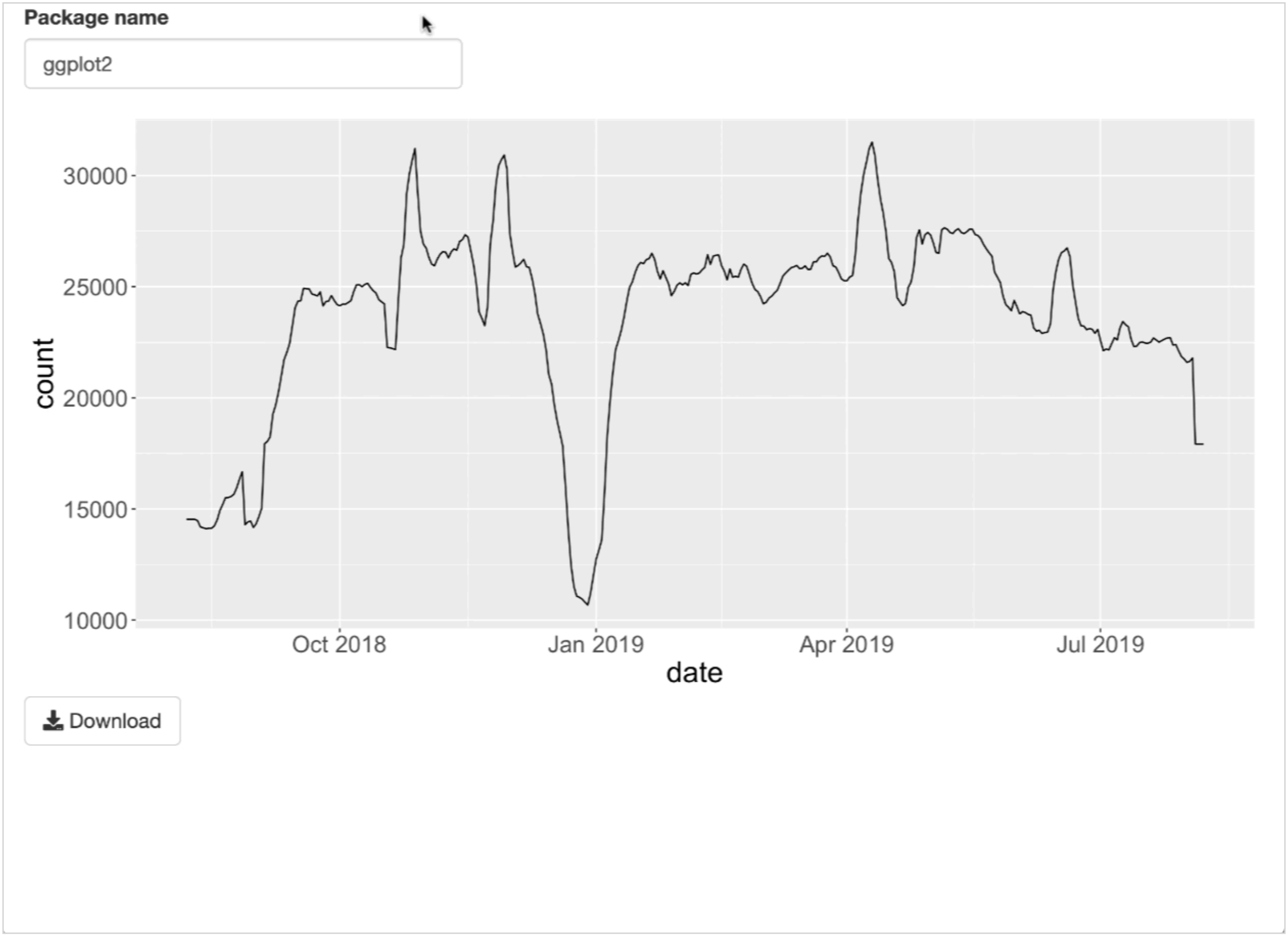
1. Display code with `displayCodeModal()`
2. Generate zip bundle with `buildRmdBundle()`
    - code (e.g., R/Rmd)
    - supporting files (e.g., csv, rds, etc)
    - results (e.g., pdf, html, etc)

Learn about these approaches at
https://rstudio.github.io/shinymeta/articles/code-distribution.html

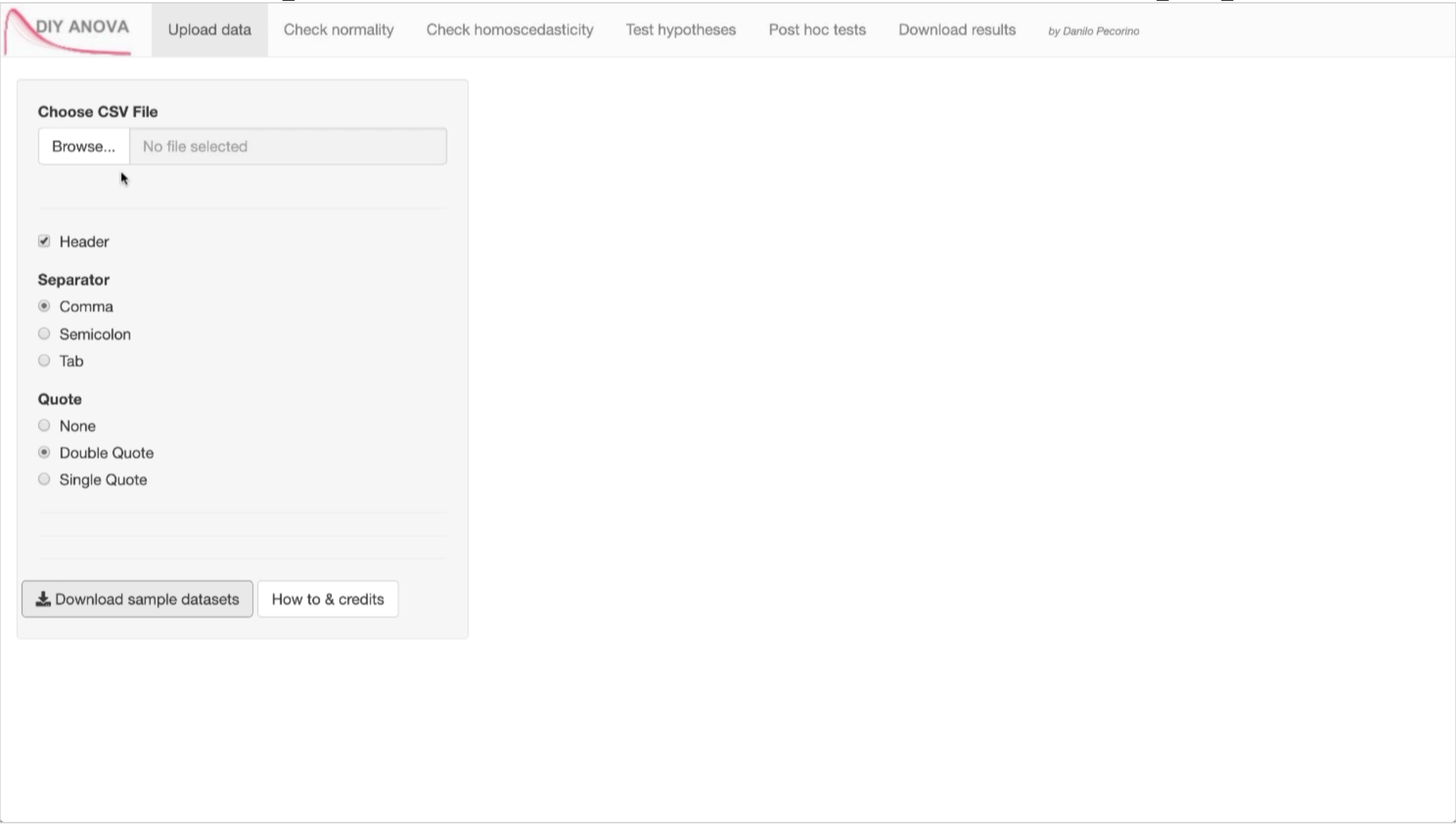
outputCodeButton() + displayCodeModal()



downloadButton() + buildRmdBundle()



Inspiration: ANOVA app



Choose CSV File

No file selected

[✓ Header]

Separator

[✓ Comma] Semicolon Tab

Quote

None [✓ Double Quote] Single Quote

The Shiny app: https://testing-apps.shinyapps.io/diy_anova/

`?GAD::rats`

Acknowledgments

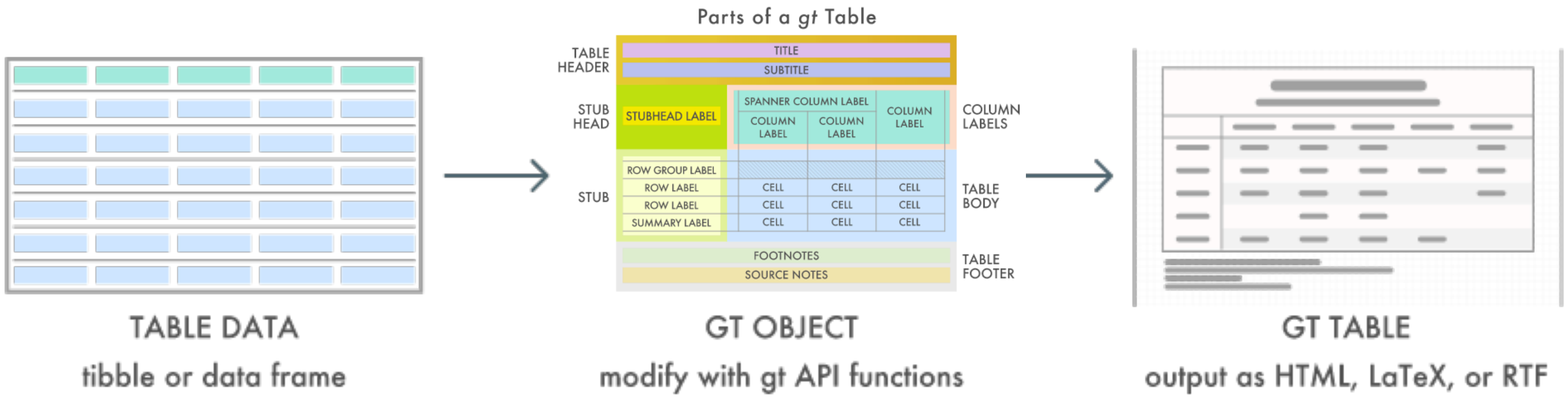
Many people have provided motivation, inspiration, and ideas that have lead to **`shinymeta`.**

Special thanks to:

- Adrian Waddell for inspiring the over-arching metaprogramming approach
- Doug Kelkhoff for his work in **`scriptgloss`**

Psst!

gt: Presentation-Ready Tables



- Web: https://gt.rstudio.com
- 2019 Progress
  - Making infrastructure robust, maintainable, and extendable
- Future work
  - Heterogeneous columns

Thank you!
Questions?

- **`shinymeta`:** Reproduce domain logic code from Shiny apps
- Web: https://rstudio.github.io/shinymeta/
- Integration:
  1. Identify and capture domain logic
  2. Mark reactive reads with `..()`
  3. Export domain logic with `expandChain()`

- Barret Schloerke RStudio®
- Slides: bit.ly/shinymeta-abacus-2019
- @schloerke.com