Reproducible Shiny apps

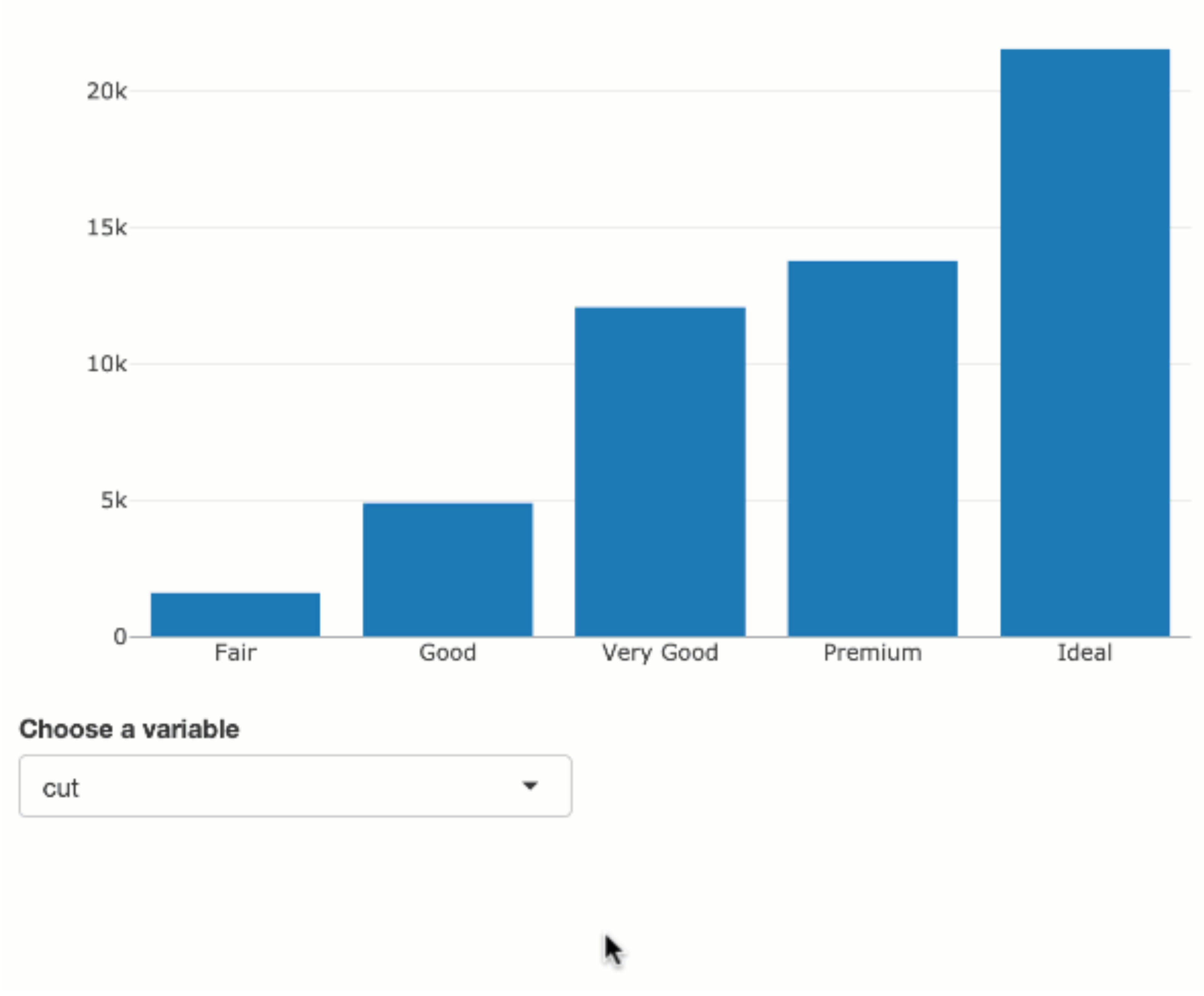
Carson Sievert
Software Engineer, RStudio
@cpsievert

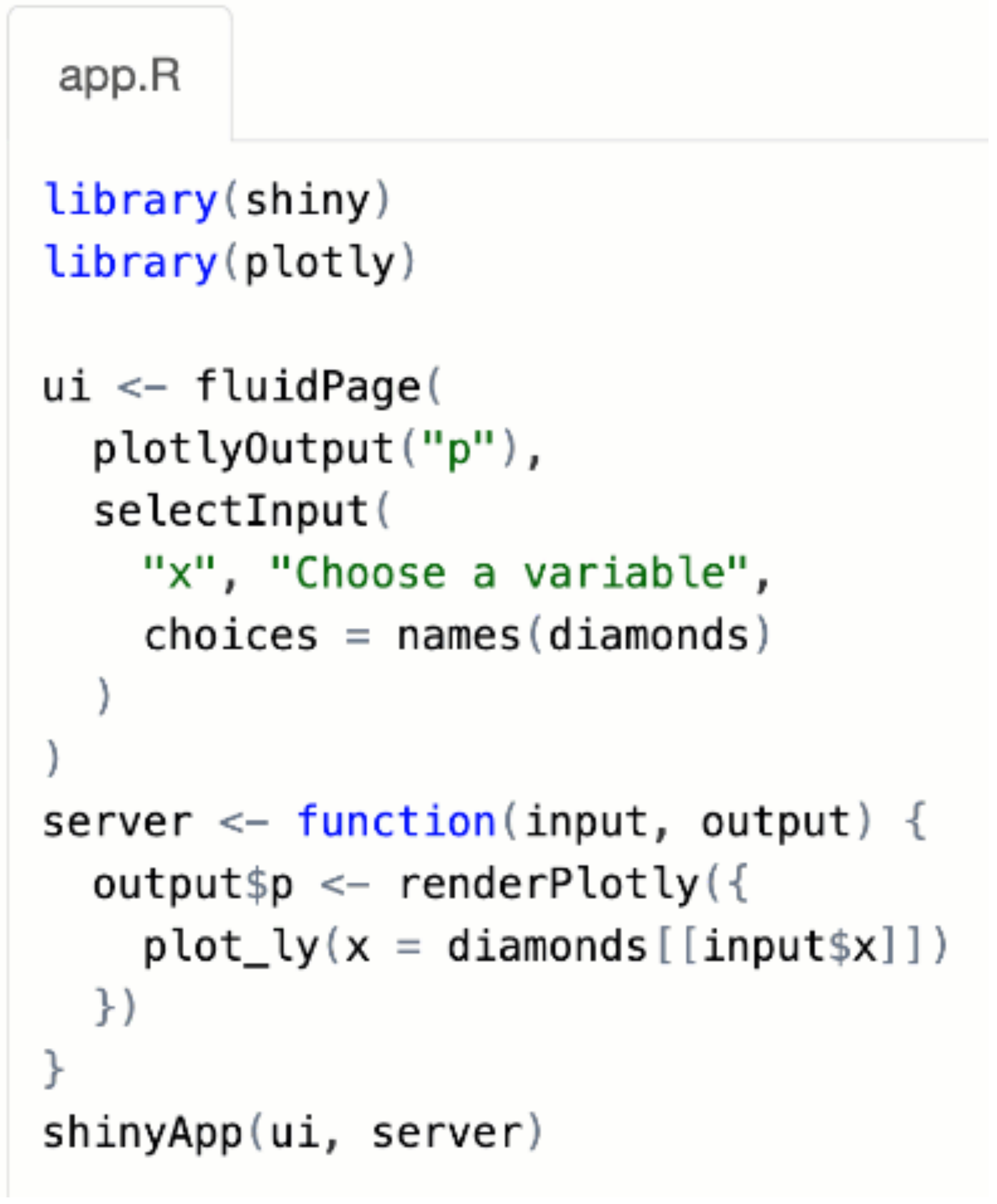
Slides bit.ly/noRth

Joint work with Joe Cheng

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)
```

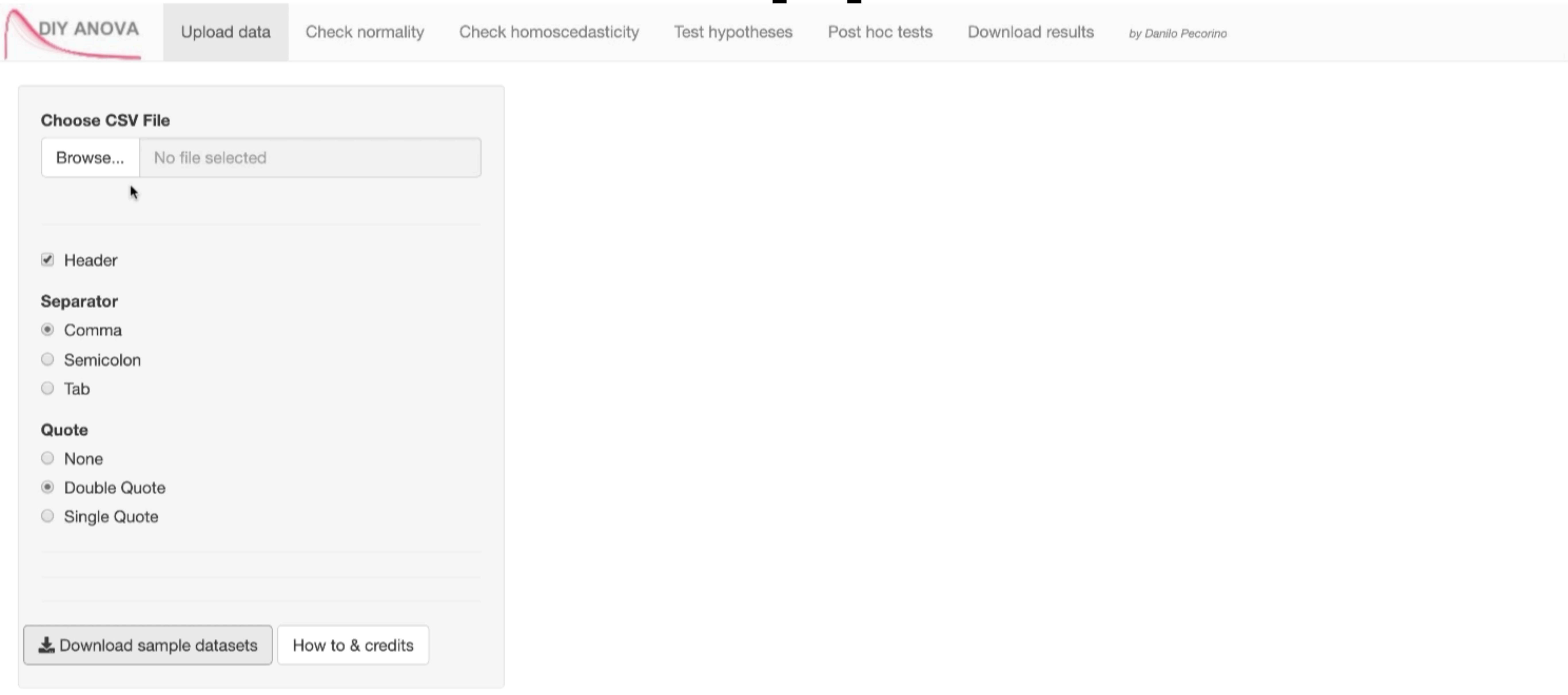
Interactivity is great, but **reproducibility suffers**

- 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 app

ANOVA app demo



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/

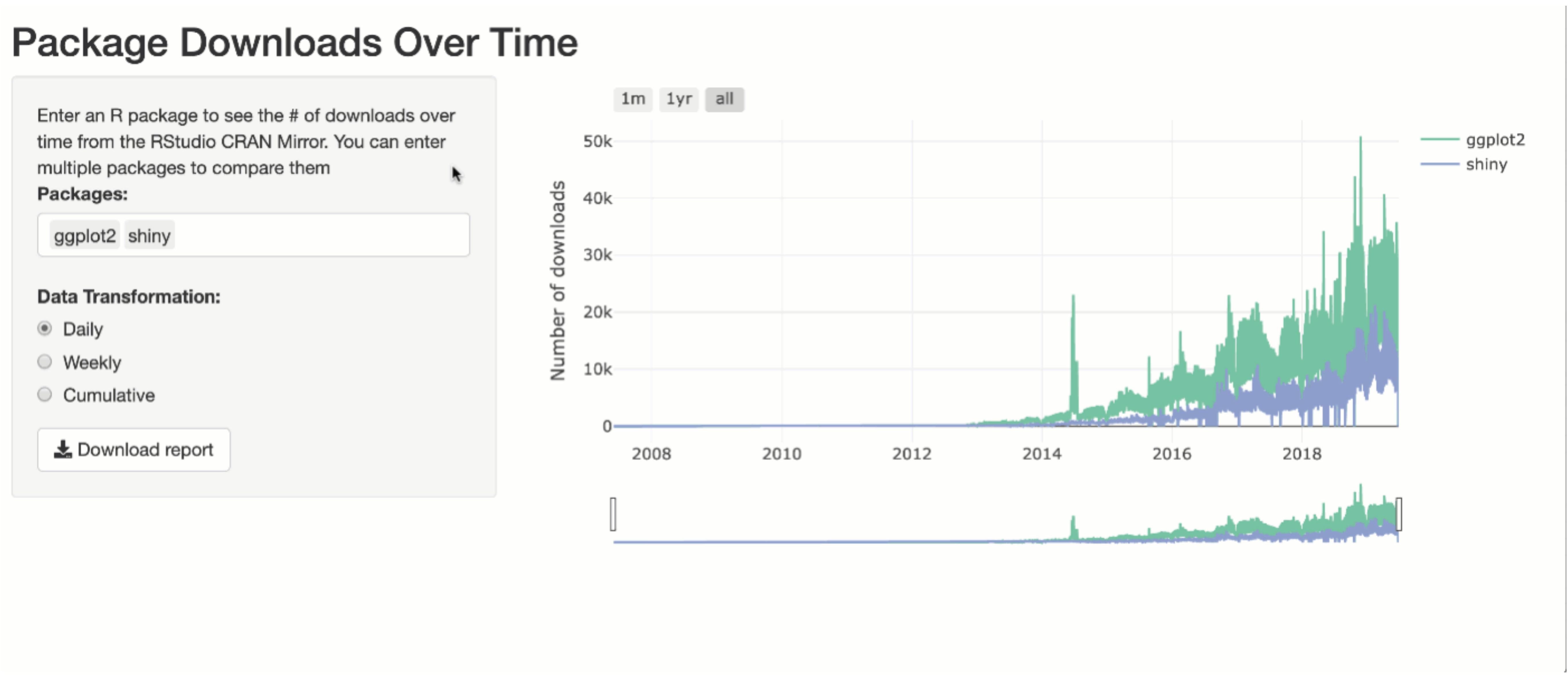
Benefits of exporting reproducible code

- **Enable**: Users to verify and extend your methodology.
- **Educate:** Users how to code.
- **Document:** “Your closest collaborator is you six months ago but you don’t reply to email.” - Mark T. Holder
- **Permanence:** Download a standalone artifact that can be saved locally (useful if server goes down or the app's features change)

Benefits of exporting reproducible code

- **Automation**: Shiny apps often use data that changes over time: stock quotes, sensor readings, centralized databases, etc. By providing reproducible R code, you enable users to *take that logic into other workflows (e.g., schedule a dynamic report)*

Cranview app demo



Package Downloads Over Time

Enter an R package to see the # of downloads over time from the RStudio CRAN Mirror. You can enter multiple packages to compare them

Packages: ggplot2 shiny

Data Transformation:

- [✓ Daily]
- Weekly
- Cumulative

Download report

The Shiny app: testing-apps.shinyapps.io/cranview
An automated report: connect.rstudioservices.com/connect/#/apps/345

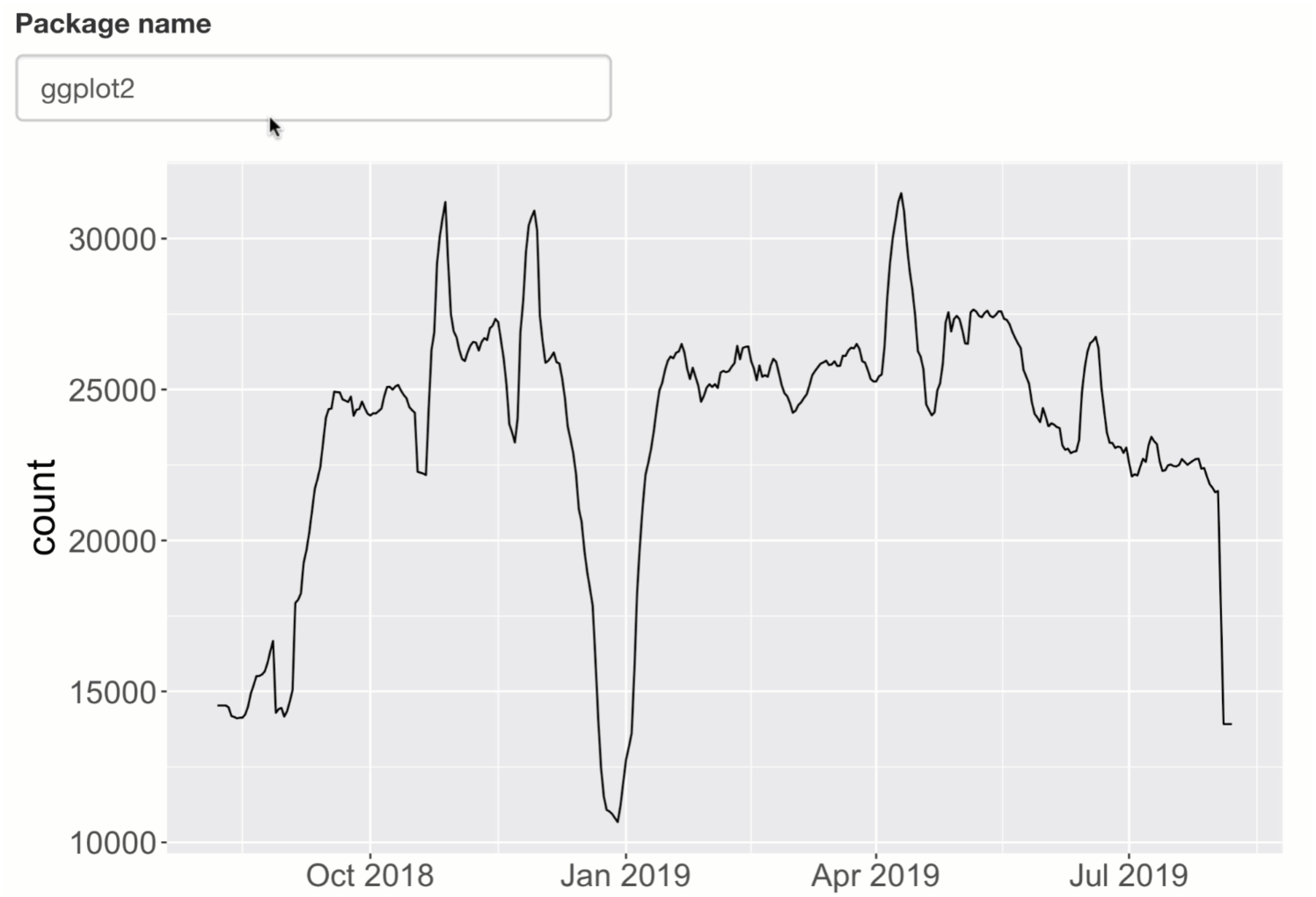
Ok, so ***how*** do we get Shiny to generate reproducible code?

shinymeta: tools for capturing logic in a Shiny app and exposing it as code that can be run outside of Shiny.

Not yet on CRAN, but can install with:

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

Basic cranview app demo



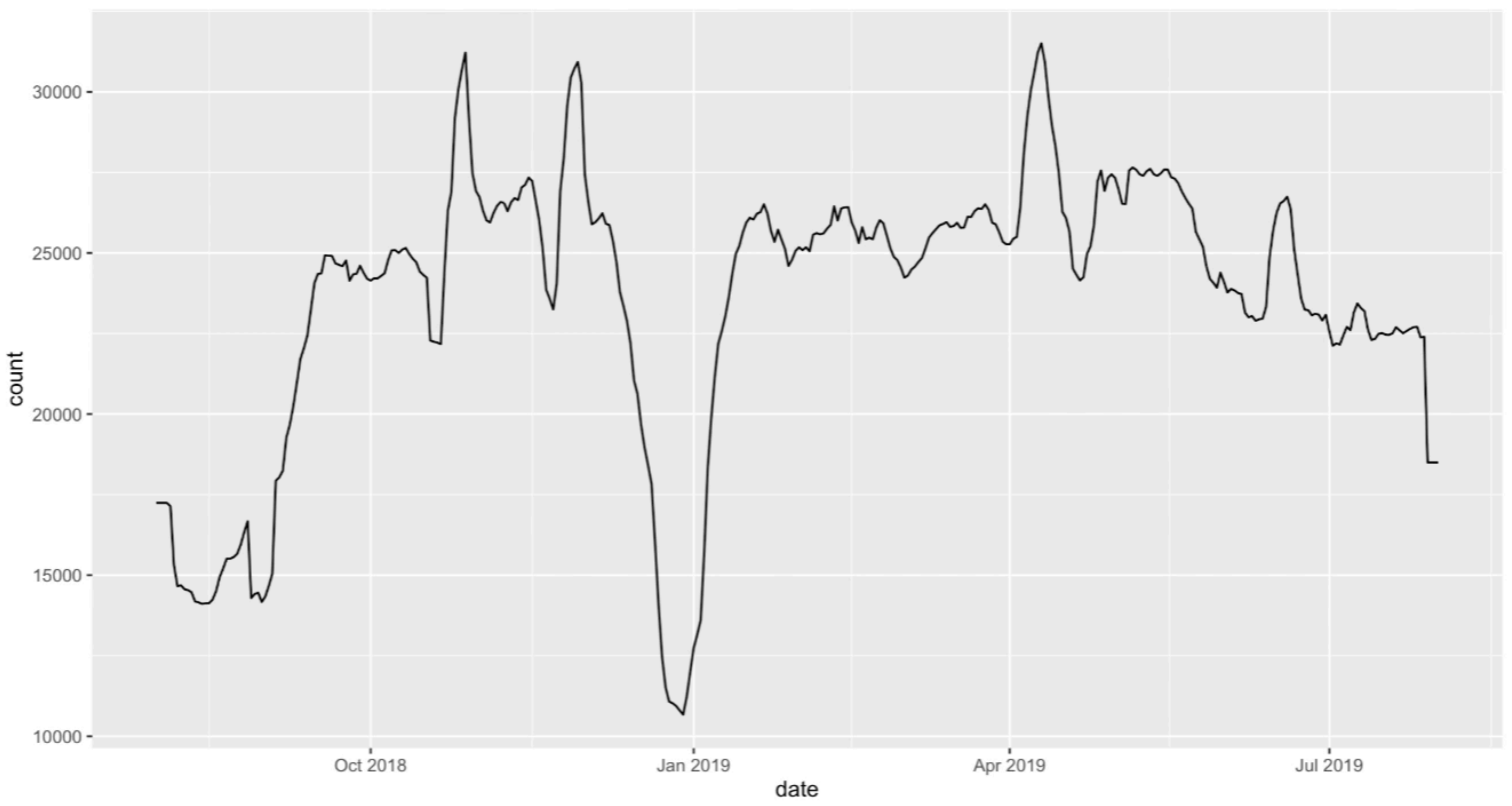
```r
library(shiny)
library(tidyverse)

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)
```

The 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

```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()
  })
}
```

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()
  })
}
```

Only applies to Shiny, don't export it!

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()
  })
}
```

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()
  })
}
```

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()
  })
}
```

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()
  })
}
```

-2 variants only capture metaExpr()

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 = ..(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()
```

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 = “2019-08-01”,
    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 = “2019-08-01”,
    to = Sys.Date()
  )

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

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

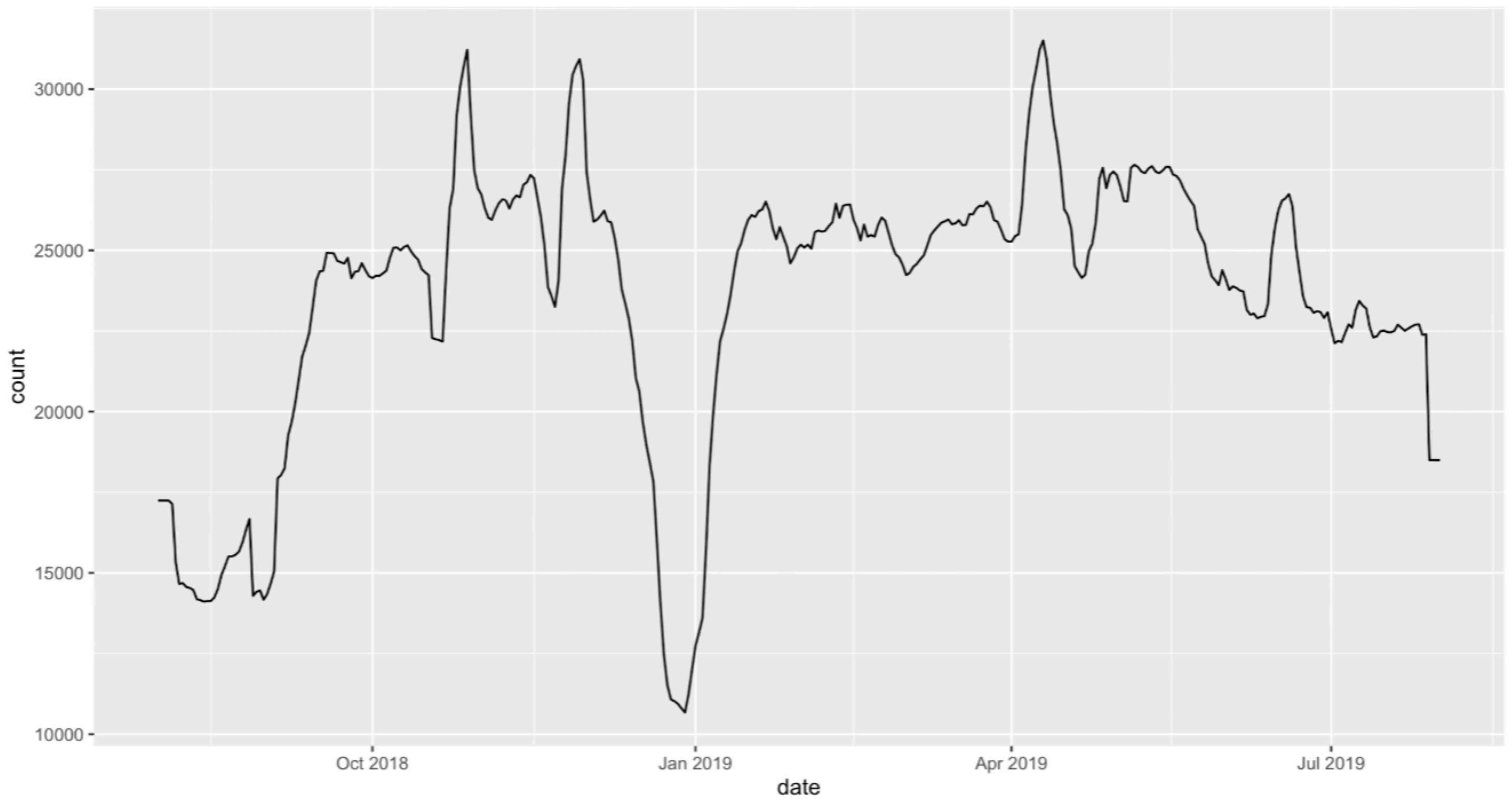
Add quoted code to supply ‘setup code’

Huzzah!

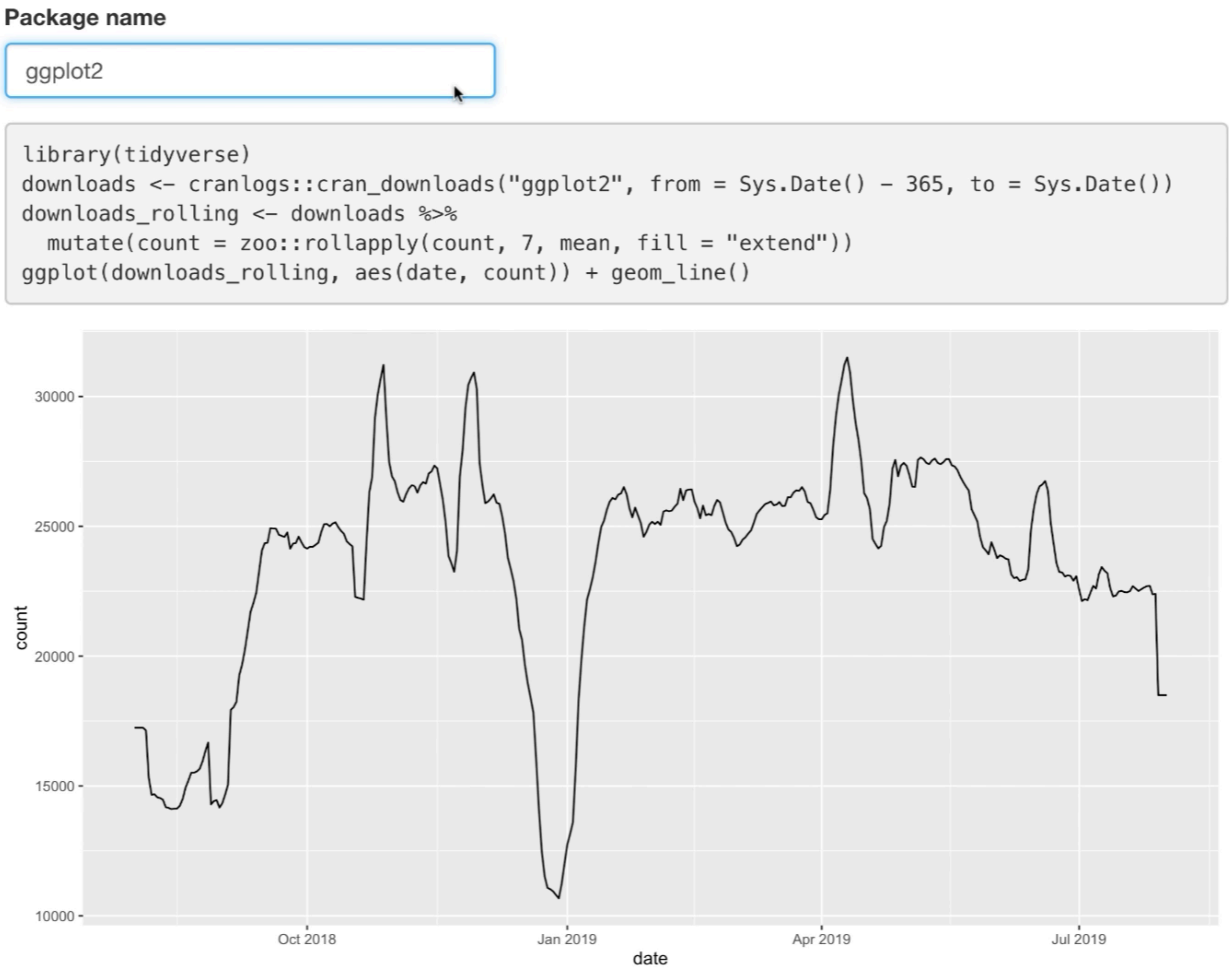
Package name

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()
```



Not the best user experience :/


Package name

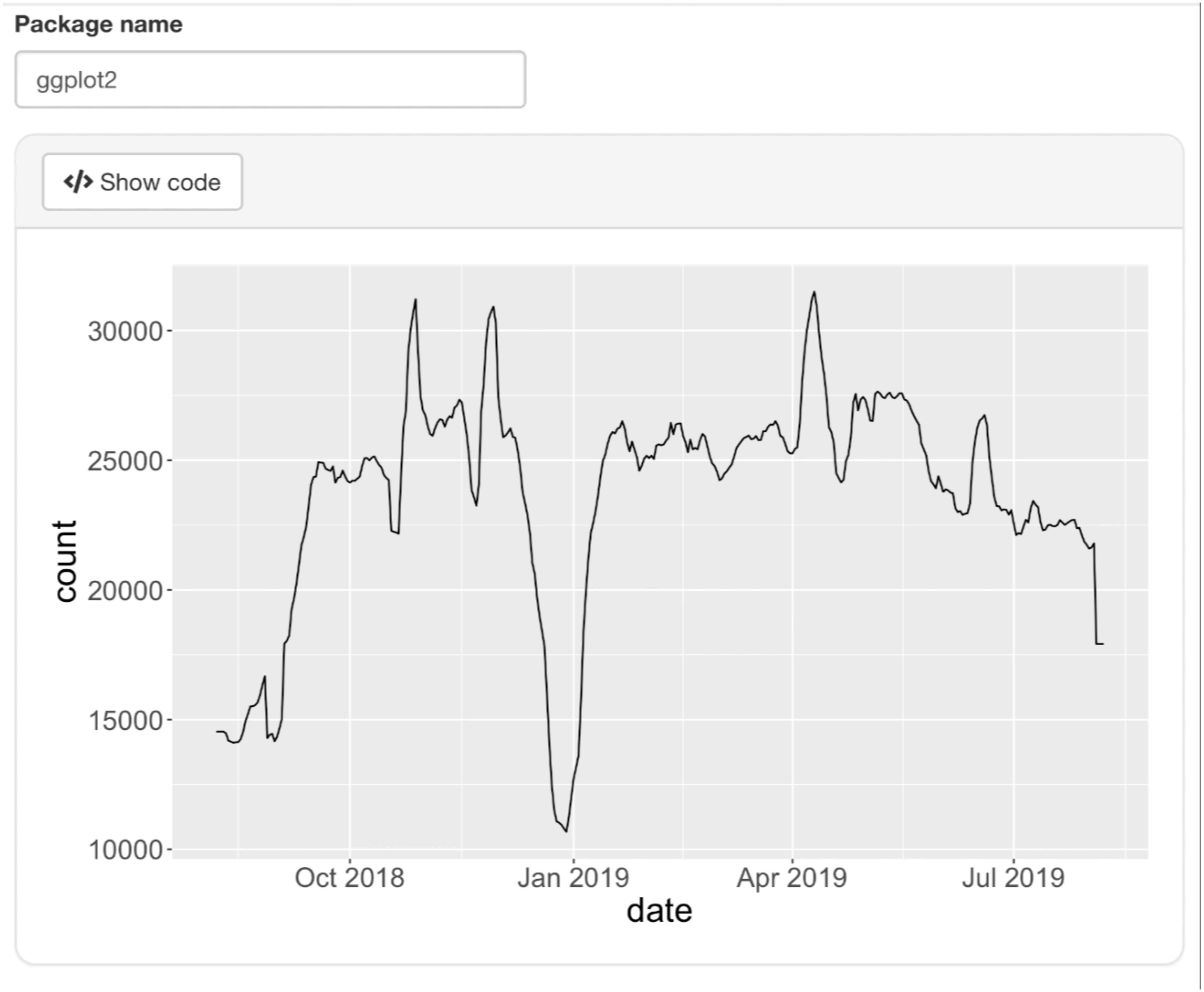
ggplot2

```
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()
```

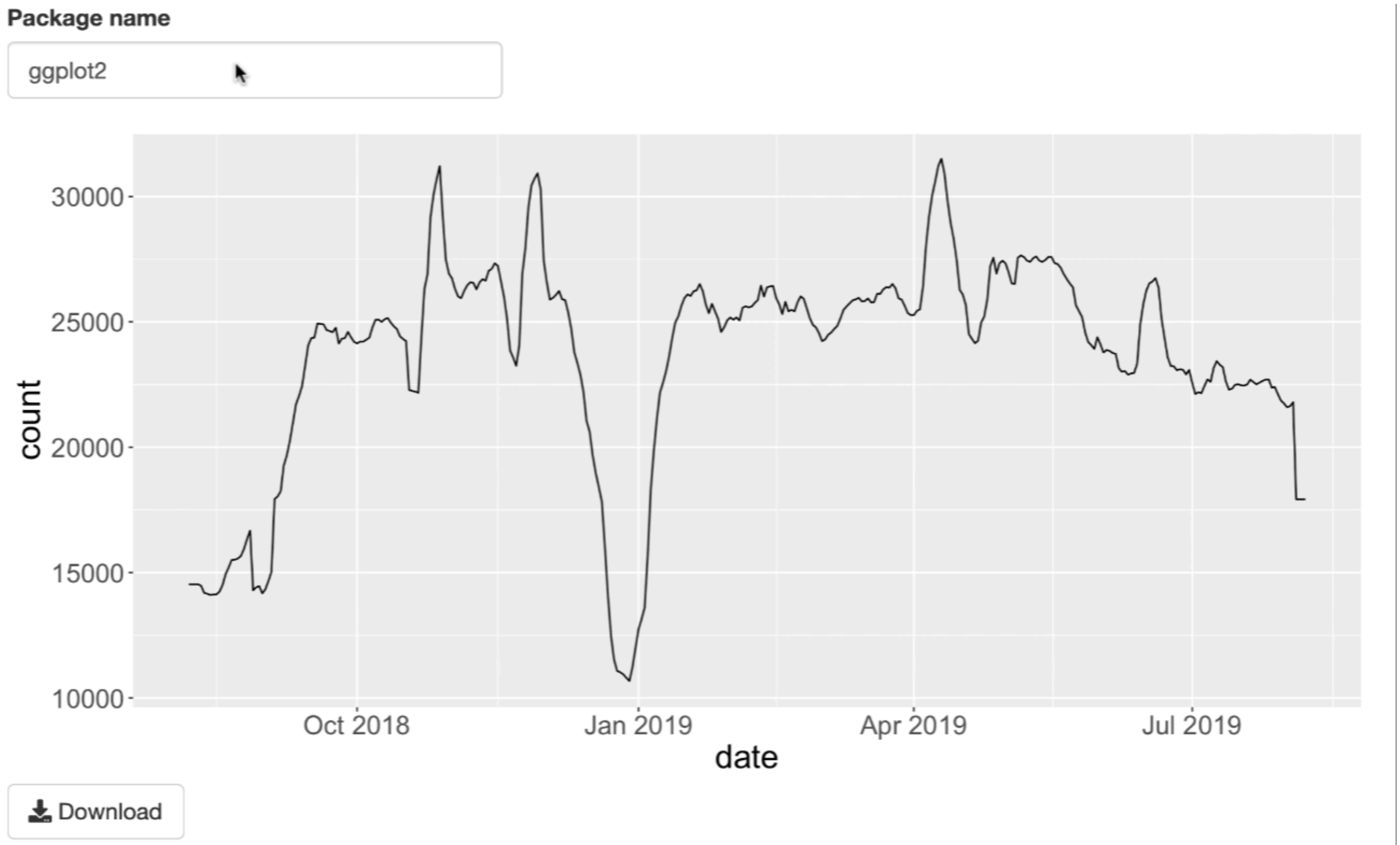
Better ways to distribute code (& results)

1. On button click, display code with `displayCodeModal()` and `outputCodeButton()`
2. On button click, download R script and results with `buildScriptBundle()`
3. On button click, download Rmd and results with `buildRmdBundle()`

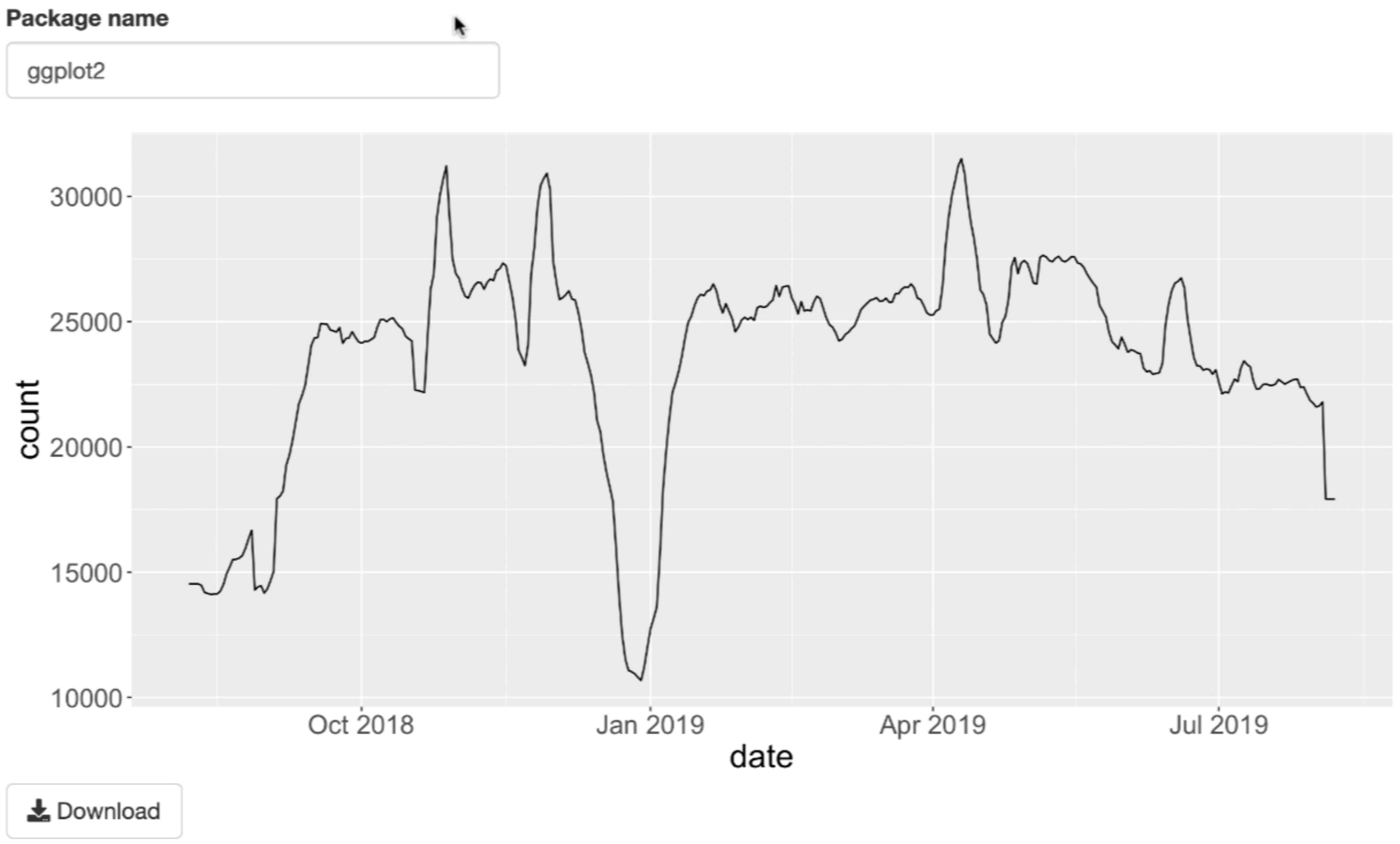
Output code button



Download R script



Download Rmd



Display code inline

```
library(shiny)
library(tidyverse)
library(shinymeta)

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

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()
  })

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

Display code on button click

```
library(shiny)
library(tidyverse)
library(shinymeta)

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

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()
  })

  observeEvent(input$plot_output_code, {
    code <- expandChain(output$plot())
    displayCodeModal(code)
  })
}
```

Downloading R script on button click

```
library(shiny)
library(tidyverse)
library(shinymeta)

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

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()
  })

  output$download <- downloadHandler("report.zip",
    content = function(out) {
      code <- expandChain(output$plot())
      buildScriptBundle(code, out)
    }
  )
}
```

Downloading Rmd on button click

```
library(shiny)
library(tidyverse)
library(shinymeta)

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

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()
  })

  output$download <- downloadHandler("report.zip",
    content = function(out) {
      code <- expandChain(output$plot())
      buildRmdBundle("cran-report.Rmd", out, vars = list(code = code))
    }
  )
}
```

In summary

- Many benefits to having an interactive GUI generate reproducible code (transparency, permanence, automation)
- **shinymeta:** new R package for capturing logic in a Shiny app and exposing it as code that can be run outside of Shiny
- Add **shinymeta** integration to a Shiny app by:
    1. Identify and capture domain logic
    2. Mark reactive reads with ..()
    3. Export domain logic with expandCode()

Thank you! Questions?

https://rstudio.github.io/shinymeta/

Slides: http://bit.ly/noRth

@cpsievert

cpsievert1@gmail.com

http://cpsievert.me