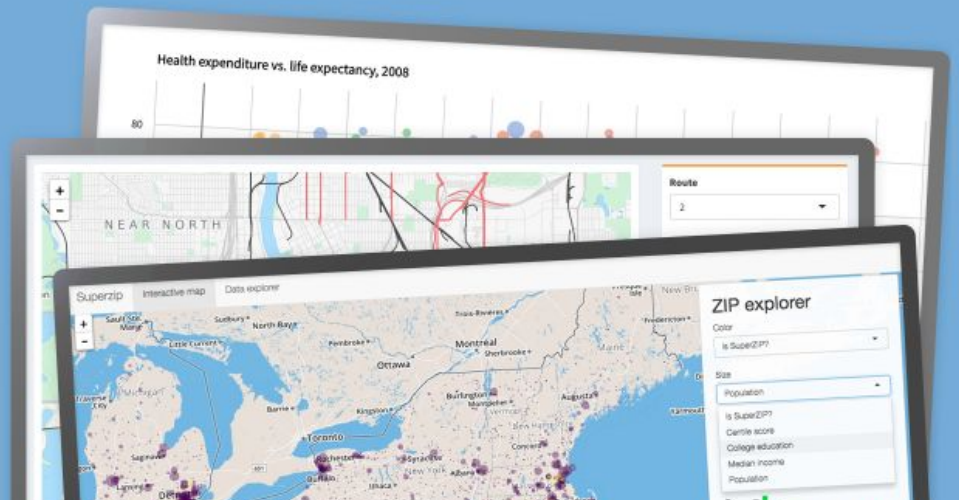
Less [exciting] is more

Shiny strategies for everyday

Marianna Foos | 8 Dec 2017
@mariannafoos
https://mfoos.github.io/blog/

Shiny, in brief

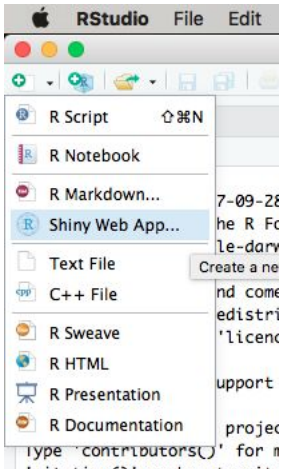
- Web “framework” for R
- Interactive analyses for your stakeholders or customers
- Your new best friend

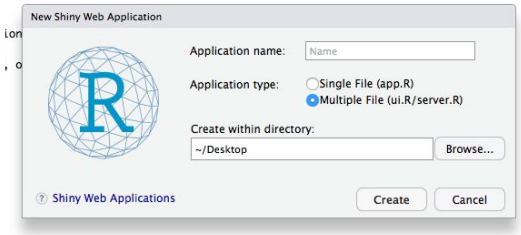


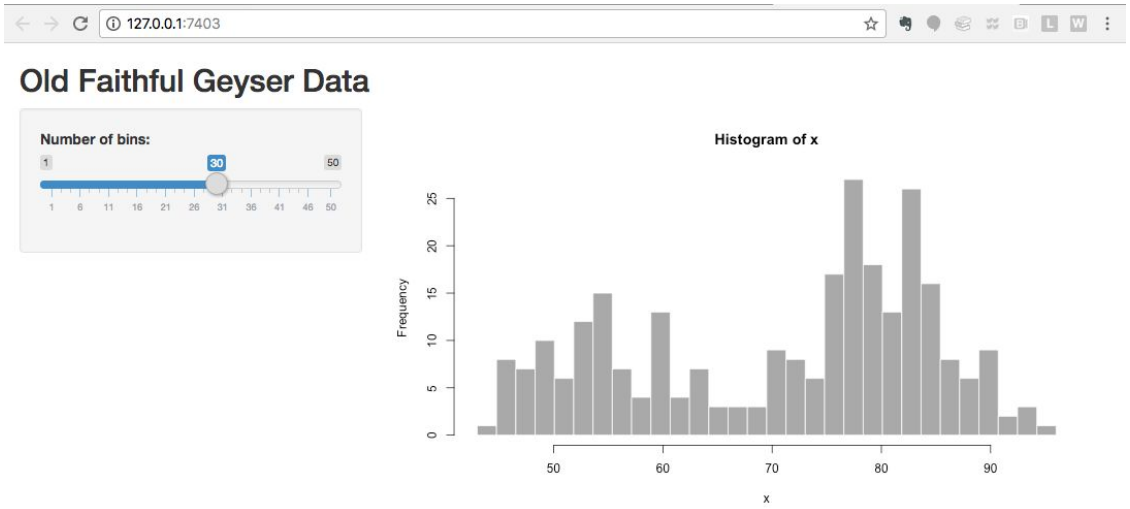
Interact. Analyze. Communicate.

Take a fresh, interactive approach to telling your data story with Shiny. Let users interact with your data and your analysis. And do it all with R.

Try it out in RStudio







ui.R file/function

- Lays out your page (HTML)
- 99% Static
- Functions end with `...Output()` or `...Input()`
- Reads from the `output` object, ingests `input` object
- [Some] variables are strings 

server.R file/function

- Runs your R code
- “Outputting” functions start with `render…()`
- Reads from `input` object, writes to `output` object
- Each chunk/function executes as the result of a change to `input`, directly or in a cascade

Execution environment[s]

server.R

```
output$age_plot <- renderPlot({
  x <- data %>% filter(age == input$age)
  plot(x)
})
```

- Isolated, capsule environment
- Order arbitrary (dependencies aside)
- “No Man’s Land” outside

Reslicing & reusing data

- But the repetition :(
- Reactive objects to the rescue!

server.R

```
agedata <- reactive({
   data %>% filter(age == input$age)
})
```

```
output$plot <- renderPlot({ agedata() })
```

```
output$table <- renderTable({ agedata() })
```

Subset and reformat further with `dplyr` (and `ggplot2`!)

server.R

```
agedata <- reactive({
   data %>% filter(age == input$age)
})
```

```
output$table <- renderDataTable({
  agedata() %>%
    mutate(PubMed = createLinkout(PubMed))
})
```

```
output$dl <- downloadHandler(
  filename = "age_table.csv",
  content = function(file){
    write_csv(agedata %>% select(-gene), file)
  }
)
```

updateSelectizeInput()

ui.R

```
selectizeInput("gene",  "Select a gene:",
  choices = c("HOX1"))
```

server.R

```
server <- function(input, output, session){
  updateSelectizeInput(session, "gene",
     choices = data, server = [TRUE|FALSE])
}
```

- Help yourself
- Help your customers

rsqlite databases

- Fast & simple
- Probably already installed

```
conn <- dbConnect(RSQLite::SQLite(), "db")
dbWriteTable(conn, "expr", expr_df)
dbGetQuery(conn,
  'SELECT *
  FROM expr
  WHERE gene == ":s"',
  params = list(s = input$gene))
dbDisconnect(conn)
```

Modules

- Wrap and reuse SHINY code
- Server function:
  - regular Shiny “chunks”
  - takes Shiny environmental variables as implied parameters
- UI function:
  - list of Shiny UI functions
  - “id” wrapped in special `ns()` function
- Both written in a file that gets source()’d

Module File

plotExpr.R

```
plotExpr_UI <- function(id){
  ns <- NS(id)
  taglist(
    selectizeInput(ns("ingene"), choices = genelist)
    plotOutput(ns("geneplot"))
  )
}
plotExpr <- function(input, output, session, dataset){
  output$geneplot <- renderPlot({
    dataset %>% filter(gene == input$ingene) %>%
    ggplot(aes()) + geom_boxplot()
  }
}
```

Calling Modules

ui.R

```
plotExpr_UI("braineac"),
plotExpr_UI("gtex")
```

server.R

```
callModule(plotExpr, "braineac",
 "In Braineac Study")
callModule(plotExpr, "gtex", "In
 GTEX")
```

Increasing number of plugins available

- library(shinyjs) - finetune stock Shiny widgets
- library(htmltools) - just how it sounds
- library(shinydashboard) - make sweet dashboards with Shiny
- library(shinycssloaders) - literally just “loading” animations

http://www.htmlwidgets.org/

https://shiny.rstudio.com/articles/

http://shiny.rstudio.com/images/shiny-cheatsheet.pdf

Test the waters with customization

- Write HTML directly
- Bring your own CSS
  - Themes available
- Integrate Javascript
  - Google analytics

Questions?