2018 Predictive Analytics Symposium

Session 31: AP - Developing Web Application With R Shiny

SOCIETY OF ACTUARIES
Antitrust Compliance Guidelines

Active participation in the Society of Actuaries is an important aspect of membership. While the positive contributions of professional societies and associations are well-recognized and encouraged, association activities are vulnerable to close antitrust scrutiny. By their very nature, associations bring together industry competitors and other market participants.

The United States antitrust laws aim to protect consumers by preserving the free economy and prohibiting anti-competitive business practices; they promote competition. There are both state and federal antitrust laws, although state antitrust laws closely follow federal law. The Sherman Act, is the primary U.S. antitrust law pertaining to association activities. The Sherman Act prohibits every contract, combination or conspiracy that places an unreasonable restraint on trade. There are, however, some activities that are illegal under all circumstances, such as price fixing, market allocation and collusive bidding.

There is no safe harbor under the antitrust law for professional association activities. Therefore, association meeting participants should refrain from discussing any activity that could potentially be construed as having an anti-competitive effect. Discussions relating to product or service pricing, market allocations, membership restrictions, product standardization or other conditions on trade could arguably be perceived as a restraint on trade and may expose the SOA and its members to antitrust enforcement procedures.

While participating in all SOA in person meetings, webinars, teleconferences or side discussions, you should avoid discussing competitively sensitive information with competitors and follow these guidelines:

- Do not discuss prices for services or products or anything else that might affect prices
- Do not discuss what you or other entities plan to do in a particular geographic or product markets or with particular customers.
- Do not speak on behalf of the SOA or any of its committees unless specifically authorized to do so.
- Do leave a meeting where any anticompetitive pricing or market allocation discussion occurs.
- Do alert SOA staff and/or legal counsel to any concerning discussions
- Do consult with legal counsel before raising any matter or making a statement that may involve competitively sensitive information.

Adherence to these guidelines involves not only avoidance of antitrust violations, but avoidance of behavior which might be so construed. These guidelines only provide an overview of prohibited activities. SOA legal counsel reviews meeting agenda and materials as deemed appropriate and any discussion that departs from the formal agenda should be scrutinized carefully. Antitrust compliance is everyone's responsibility; however, please seek legal counsel if you have any questions or concerns.

Presentation Disclaimer

Presentations are intended for educational purposes only and do not replace independent professional judgment. Statements of fact and opinions expressed are those of the participants individually and, unless expressly stated to the contrary, are not the opinion or position of the Society of Actuaries, its cosponsors or its committees. The Society of Actuaries does not endorse or approve, and assumes no responsibility for, the content, accuracy or completeness of the information presented. Attendees should note that the sessions are audio-recorded and may be published in various media, including print, audio and video formats without further notice.

Predictive Analytics Symposium

Ben Johnson

Session 31: Developing Web Applications With R Shiny

September 20, 2019

SOCIETY OF ACTUARIES

Agenda

- Introduction to Shiny
  - Example applications
- Components of an app
  - User interface
  - Server code
- How to host Shiny applications online
- Tips & tricks
- Demo of building a Shiny application

What is Shiny?

- Web application framework
- Allows R programmers to create interactive web apps
  - Can be programmed exclusively in R
  - Augment code with HTML, CSS, and JavaScript
- Shiny is an R package developed and maintained by RStudio

I'd rather be shiny.

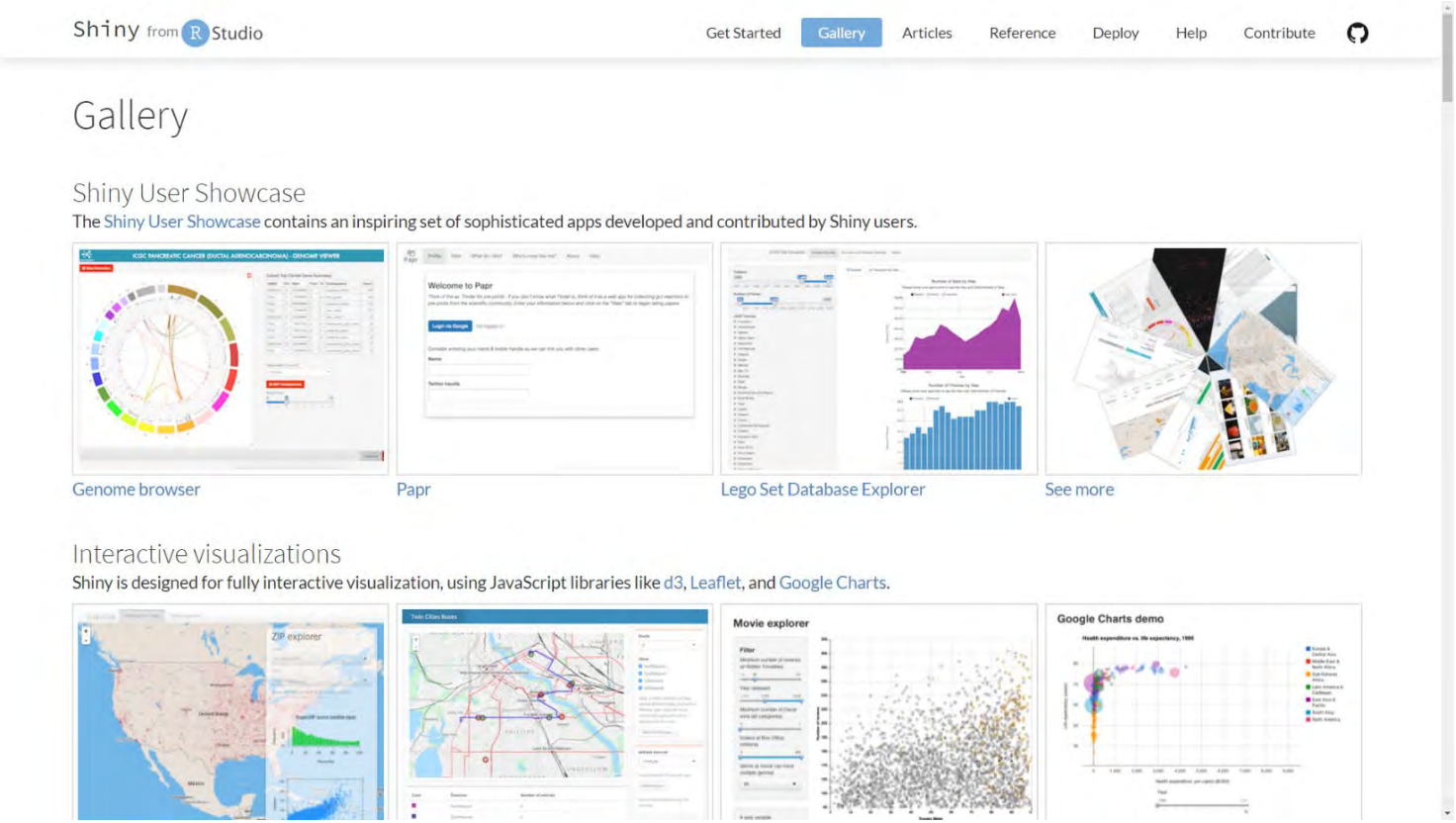
Tamatoa from Disney's Moana

Why use Shiny?

- You want to build web-apps but don't know the multitude of required programming languages
- Your analytics is already performed in R
- One person can accomplish what would otherwise have taken a full team of specialists
- It can be relatively inexpensive, or potentially free

← We tend to like this stuff

Shiny examples

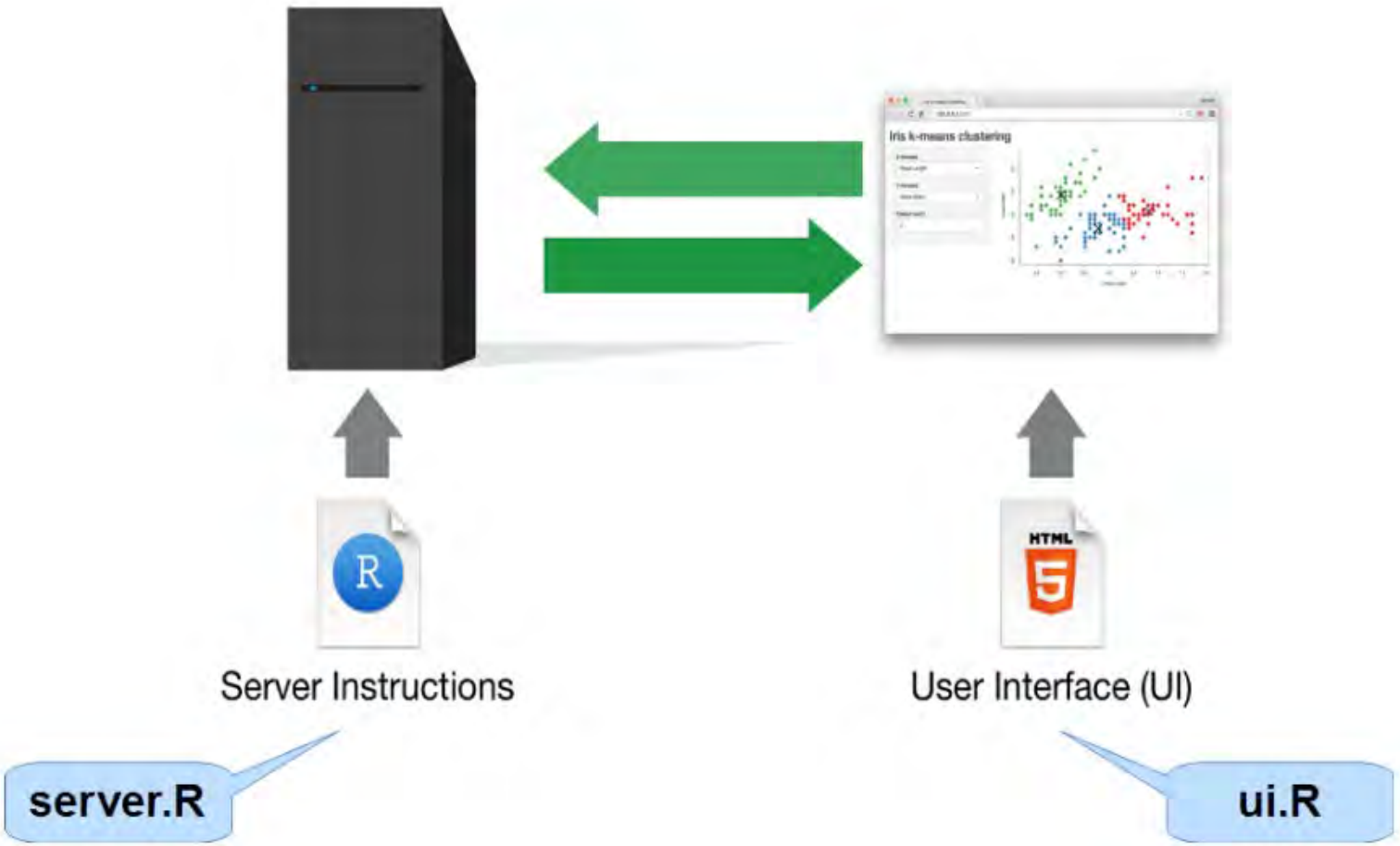


- https://shiny.rstudio.com/gallery/
- https://www.rstudio.com/products/shiny/shiny-user-showcase/

How does Shiny work?

- Employs an active R session to host the application
- The R session communicates with the browser in reactive context
  - Reactive context means that the session is actively listening for changes in user inputs
  - The R session reacts to changes in inputs by updating any corresponding outputs

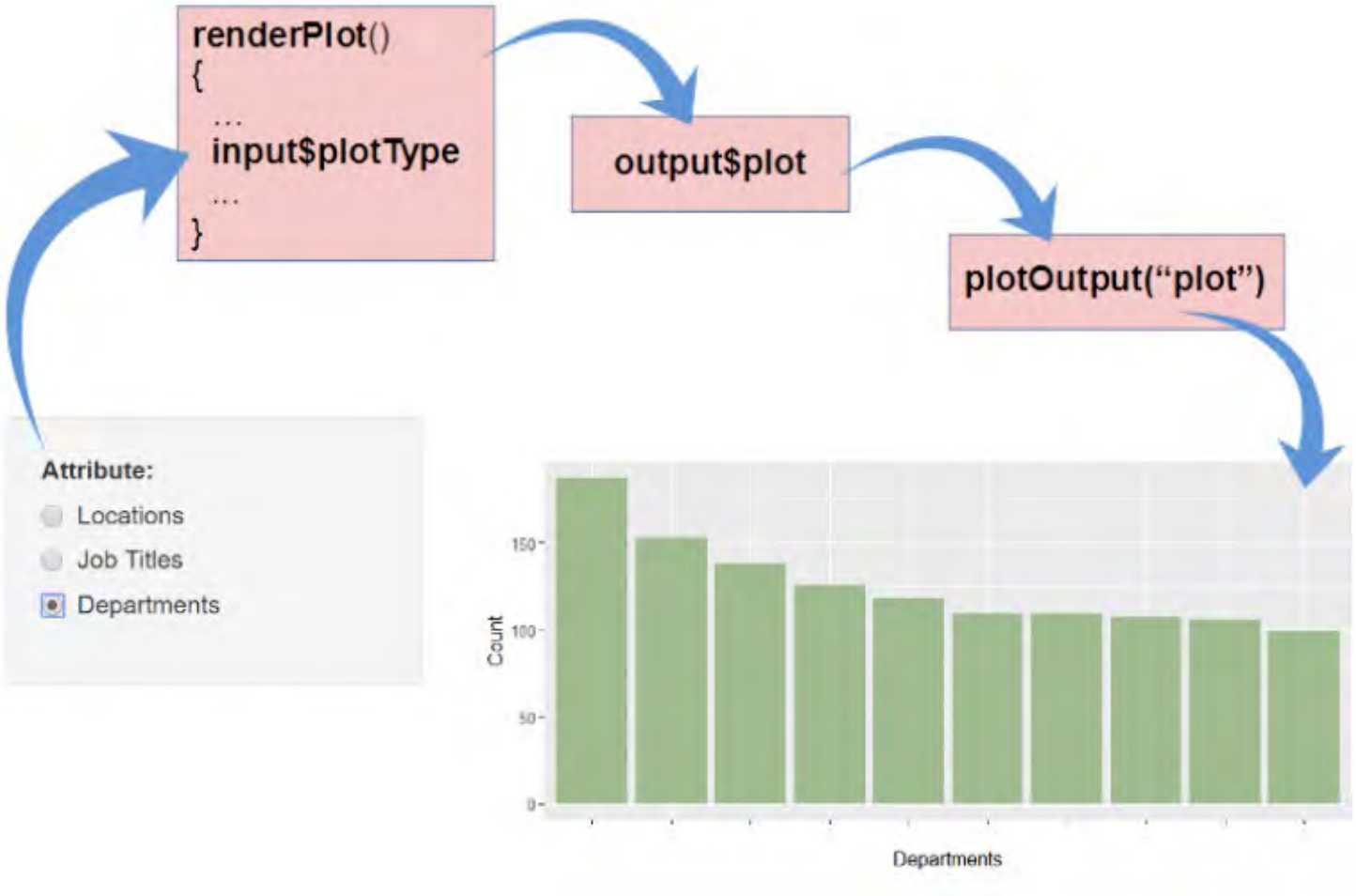
How does Shiny work?



What does Shiny code look like?

- User interface component
  - R code to build the structure and layout of the app
  - This dictates what the end user sees
- Server component
  - R code to process data and other objects
  - Defines relationships between inputs and outputs
- Supplementary files
  - Global file
  - www folder
  - Any additional scripts

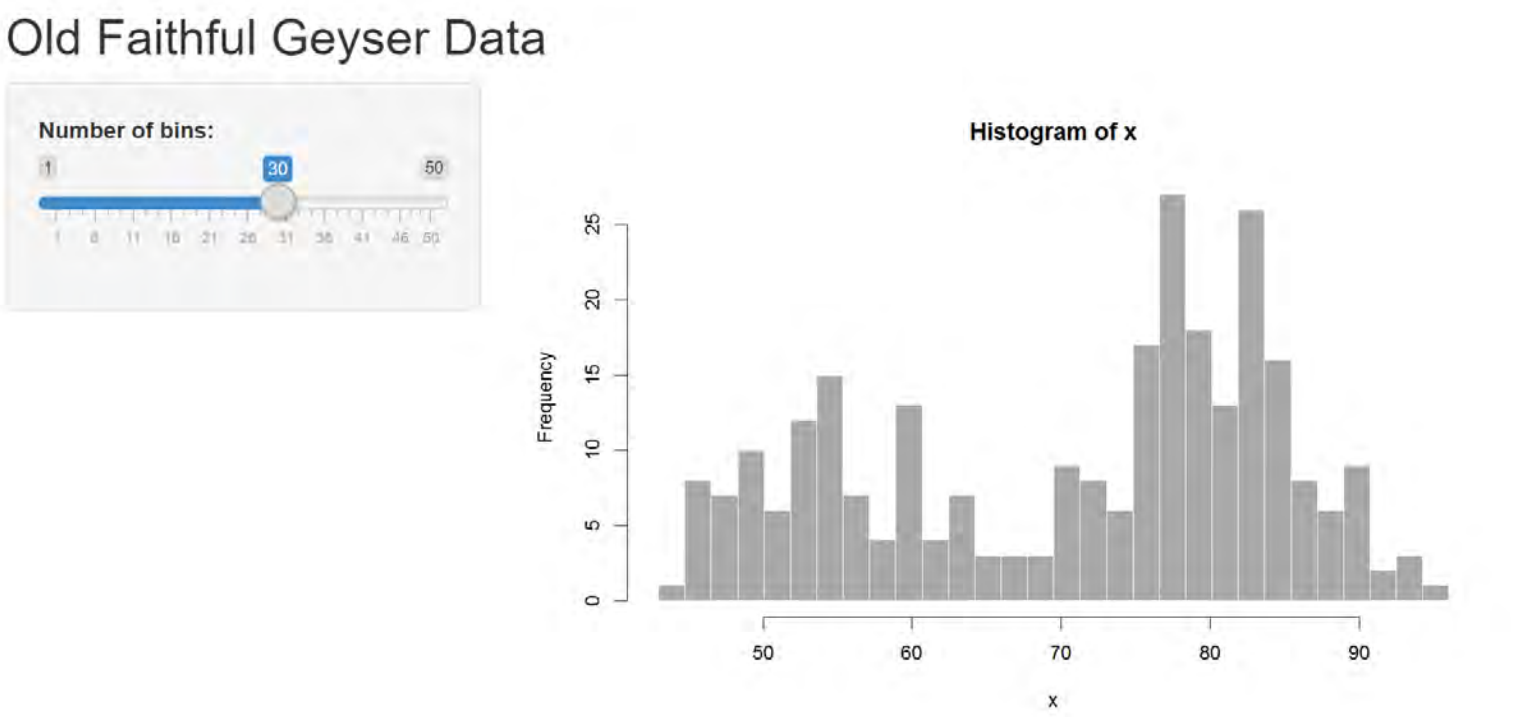
Basic reactive flow chart example



User interface

SOCIETY OF ACTUARIES

What the UI looks like in browser



UI code using Shiny

```
ui <- fluidPage(

   # Application title
   titlePanel("Old Faithful Geyser Data"),

   # Sidebar with a slider input for number of bins
   sidebarLayout(
      sidebarPanel(
         sliderInput("bins",
                     "Number of bins:",
                     min = 1,
                     max = 50,
                     value = 30)
      ),

      # Show a plot of the generated distribution
      mainPanel(
         plotOutput("distPlot")
      )
   )
)
```

UI code using HTML

```
<div class="container-fluid">
  <h2>Old Faithful Geyser Data</h2>
  <div class="row">
    <div class="col-sm-4">
      <form class="well">
        <div class="form-group shiny-input-container">
          <label class="control-label" for="bins">Number of bins:</la
bel>
          <input class="js-range-slider" id="bins" data-min="1" data-
max="50" data-from="30" data-step="1" data-grid="true" data-grid-num=
"9.8" data-grid-snap="false" data-prettify-separator="," data-prettif
y-enabled="true" data-keyboard="true" data-data-type="number"/>
        </div>
      </form>
    </div>
    <div class="col-sm-8">
      <div id="distPlot" class="shiny-plot-output" style="width: 100%
 ; height: 400px"></div>
    </div>
  </div>
</div>
```

Shiny input controls

Buttons

Action

Submit

`actionButton()`
`submitButton()`

Single checkbox

Choice A

`checkboxInput()`

Checkbox group

Choice 1
Choice 2
Choice 3

`checkboxGroupInput()`

Date input

2014-01-01

`dateInput()`

Date range

2014-01-24 to 2014-01-24

`dateRangeInput()`

File input

Choose File No file chosen

`fileInput()`

Numeric input

1

`numericInput()`

Password Input

`passwordInput()`

Radio buttons

Choice 1
Choice 2
Choice 3

`radioButtons()`

Select box

Choice 1

`selectInput()`

Sliders

`sliderInput()`

Text input

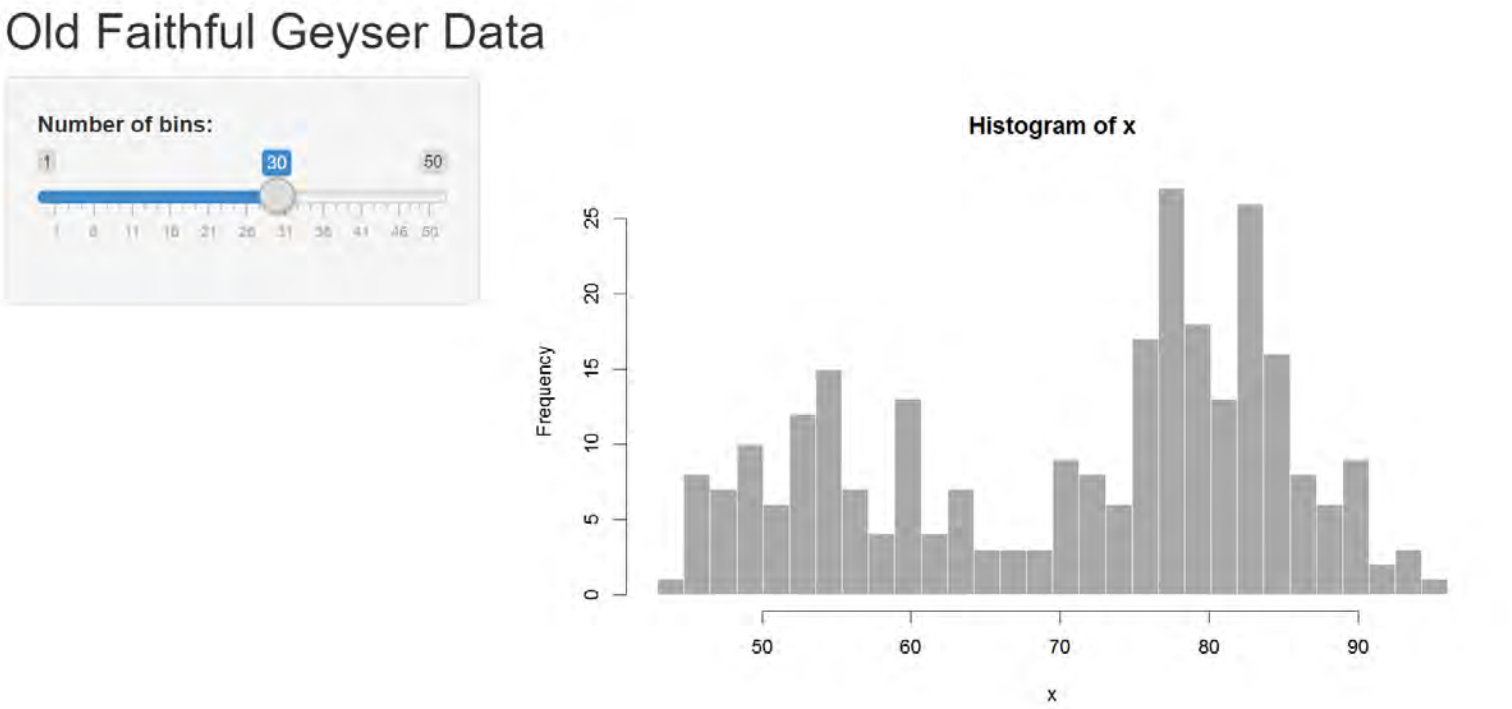
Enter text...

`textInput()`

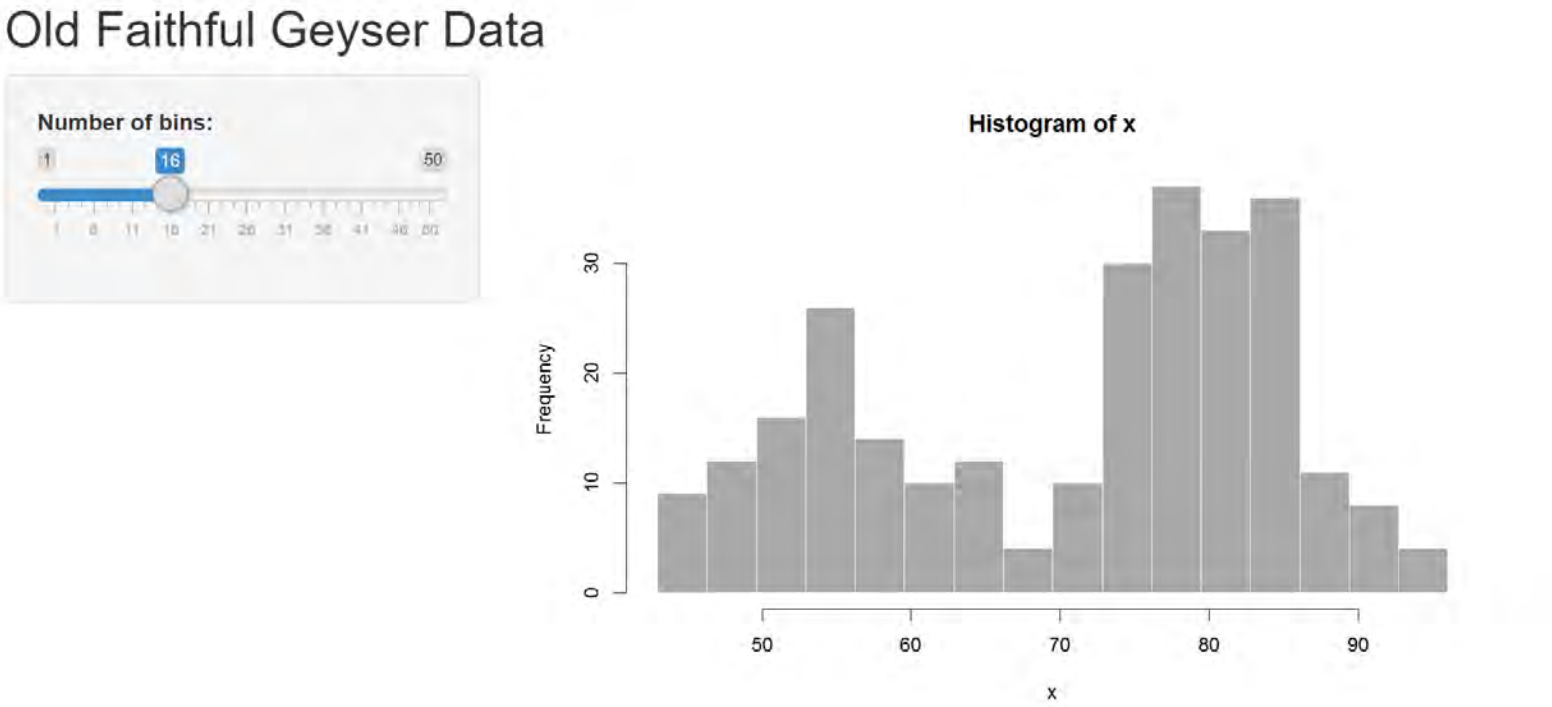
Server

SOCIETY OF ACTUARIES

Shiny is interactive and reactive



Shiny is interactive and reactive



Server code using Shiny

```
# Define server logic required to draw a histogram
server <- function(input, output) {

   output$distPlot <- renderPlot({
      # generate bins based on input$bins from ui.R
      x    <- faithful[, 2]
      bins <- seq(min(x), max(x), length.out = input$bins + 1)

      # draw the histogram with the specified number of bins
      hist(x, breaks = bins, col = 'darkgray', border = 'white')
   })
}

# Run the application
shinyApp(ui = ui, server = server)
```

Input, Output, and Session

- Think of these as reactive lists
- Input – contains values for each user control
  - Values can refresh when the user interacts with browser
- Output – contains visuals that depend on inputs
  - Visuals are created by R code in the server
- Session – contains additional client information
  - E.g. pixelratio, namespace, user, group
  - Some aspects of the session object only exists if the user accesses app from internet

Hosting your application

SOCIETY OF ACTUARIES

Shinyapps.io

Pros

- Quick and easy
- No server maintenance
- Security managed by RStudio
- Can be free

Cons

- Limited control over configuration
- Limited user authentication
- No local storage persistency
- If free, time of usage is limited

Shiny Server Pro

- Lots of control
- Deploy app behind firewalls
  - Helpful in testing
- Enterprise-grade user access and security features

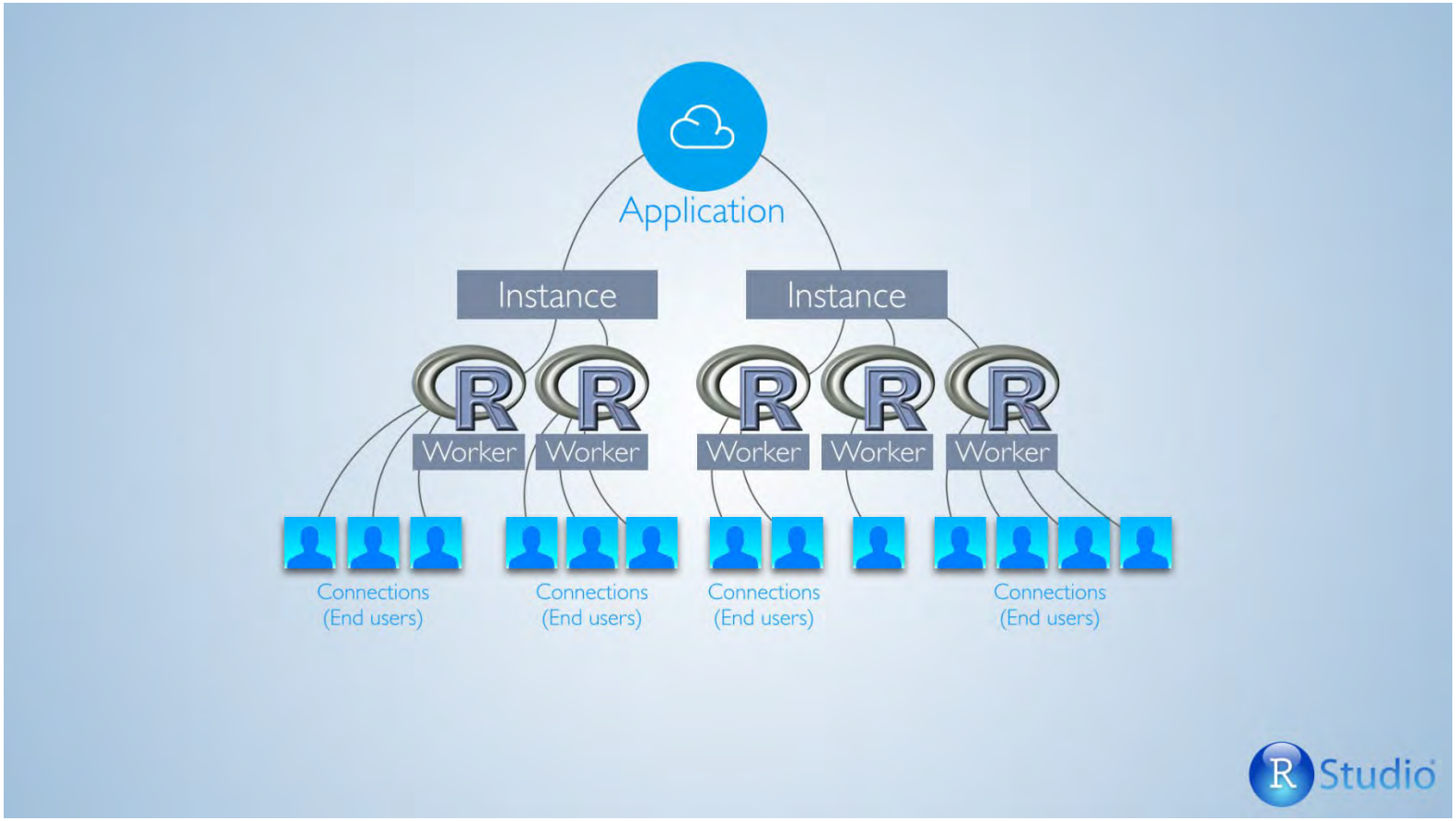
- Non-negligible cost
- Requires technical know-how to maintain

RStudio Connect

- Publishing platform for Shiny apps, markdown docs, dashboards, etc.
- Push-button publishing from RStudio IDE
- Ideal for sharing internally

- Very non-negligible cost
  - Can range from $15k-80k annually

How multiple users connect



Source: http://shiny.rstudio.com/articles/scaling-and-tuning.html

Why scaling optimization matters

- R is single-threaded
  - If two users share an R process, they compete for computational resources
  - Can result in the app becoming nonresponsive while it waits for other computations to finish
- When two users share an R process, if the application crashes then it will affect both users
- Limiting the number of users per R process also has negative consequences
  - Increases the time required to initialize the app
  - Will use more RAM on your server

Tips, tricks, and neat features

SOCIETY OF ACTUARIES

Practical considerations

- Who is your intended audience?
  - How many users do you anticipate
  - Is your app external or internal

- Will the app serve as a demonstrative web page or an analytics tool
- Will Shiny fit into your workflow?
  - Do you need to save files to feed into external systems
  - Will users need to download from your server, or can files be written to the server

Practical considerations (continued)

- Do you need an audit trail
  - Build logging into your code
  - Build a report generating capability
- Can users start fresh every time they visit your application or does it need to “remember” their previous visits
  - Bookmarking in Shiny is great for this
- What user experience is sufficient for your use case
  - Creating a pleasant experience takes a lot of work

Downloading versus writing to disk

- Examples of writing to disk:
  - write.csv
  - saveRDS
- Examples of downloading
  - downloadHandler

- If your Shiny app is hosted online, writing to disk will save files to the server.
- You can transmit data to the end user by employing a downloadHandler
  - The user's internet browser handles this action

Bookmarking in Shiny

- Bookmarking allows you to save the active state of the shiny application so that it can be revisited later with identical input values
- enableBookmarking(store = “url”)
  - store = “server” saves the state data to the server
- bookmarkButton

Source: http://clipart-library.com

Use modularized code

- Shiny modules are essentially mini-applications
  - Modules have their own UI and server code
  - They use reactive context
- Using Shiny modules will streamline future updates
  - Changes to the current code can be made in one place
  - You can build new features quickly when they have commonalities with existing features
- Your code will be much easier to read when separated into distinct scripts

Use a global CSS file

- You can style your Shiny applications with CSS in three different ways
  - Inline CSS
  - In the HTML header
  - In a separate file
- Try to resist using inline CSS whenever possible
  - CSS in a single file will help increase development speed and be easier to maintain
- Implement your CSS code with the “addClass” function

Example Shiny-related packages

- shinyWidgets
  - Many additional input controls
- shinyBS
  - Useful for adding special alerts, buttons, modals, etc.
- shinyJS
  - Adds JavaScript functionality like show, hide, disable, etc.
- shinydashboard
  - Dashboard layout applications
- DT
  - Data tables

The shinyJS package

- This is my favorite helper package for Shiny applications
- One method for creating conditional UI is to use functions like “hide” or “show” to display the UI at the appropriate time
  - Those UI elements will technically exists but not be seen by the user
  - Contrast this to functions like “insertUI” which create UI elements dynamically
- https://deanattali.com/shinyjs/

Shiny resources

SOCIETY OF ACTUARIES

- https://www.rstudio.com/products/shiny/shiny-user-showcase/
- https://shiny.rstudio.com/gallery/
- https://shiny.rstudio.com/tutorial/
  - Garrett Grolemund's video tutorial
  - Also includes written tutorials
  - Additional videos from RStudio's conferences related to shiny development
- https://deanattali.com/blog/building-shiny-apps-tutorial/
  - Dean Attali is unaffiliated with RStudio but well known in the Shiny community for his tutorials and Shiny-related R packages

Demo in R

SOCIETY OF ACTUARIES