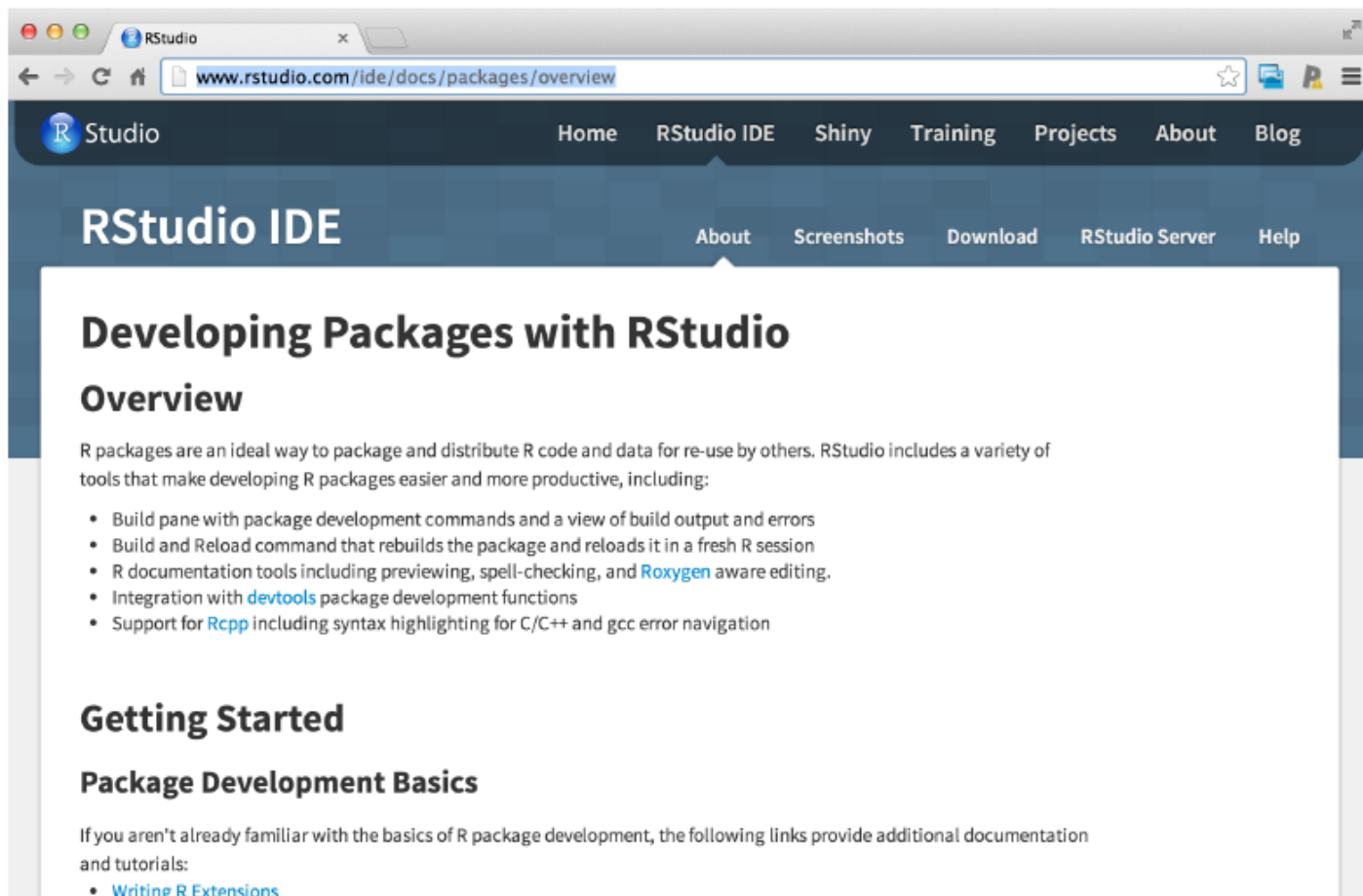


Building Data Products Overview

Building Data Products Content

- R packages
 - devtools
 - roxygen
 - testthat
- rCharts
- Slidify
- Shiny

R packages - for the engineers



The screenshot shows a web browser window with the URL www.rstudio.com/ide/docs/packages/overview. The page has a dark blue header with the RStudio logo and navigation links: Home, RStudio IDE, Shiny, Training, Projects, About, and Blog. Below this is a sub-header for 'RStudio IDE' with links: About, Screenshots, Download, RStudio Server, and Help. The main content area is titled 'Developing Packages with RStudio' and 'Overview'. It explains that R packages are an ideal way to package and distribute R code and data for re-use by others. It lists several tools included in RStudio for developing packages: a build pane, a build and reload command, R documentation tools (previewing, spell-checking, and Roxygen aware editing), integration with devtools, and support for Rcpp. Below this is a 'Getting Started' section titled 'Package Development Basics' which provides links to additional documentation and tutorials, including 'Writing R Extensions'.

Developing Packages with RStudio

Overview

R packages are an ideal way to package and distribute R code and data for re-use by others. RStudio includes a variety of tools that make developing R packages easier and more productive, including:

- Build pane with package development commands and a view of build output and errors
- Build and Reload command that rebuilds the package and reloads it in a fresh R session
- R documentation tools including previewing, spell-checking, and [Roxygen](#) aware editing.
- Integration with [devtools](#) package development functions
- Support for [Rcpp](#) including syntax highlighting for C/C++ and gcc error navigation

Getting Started

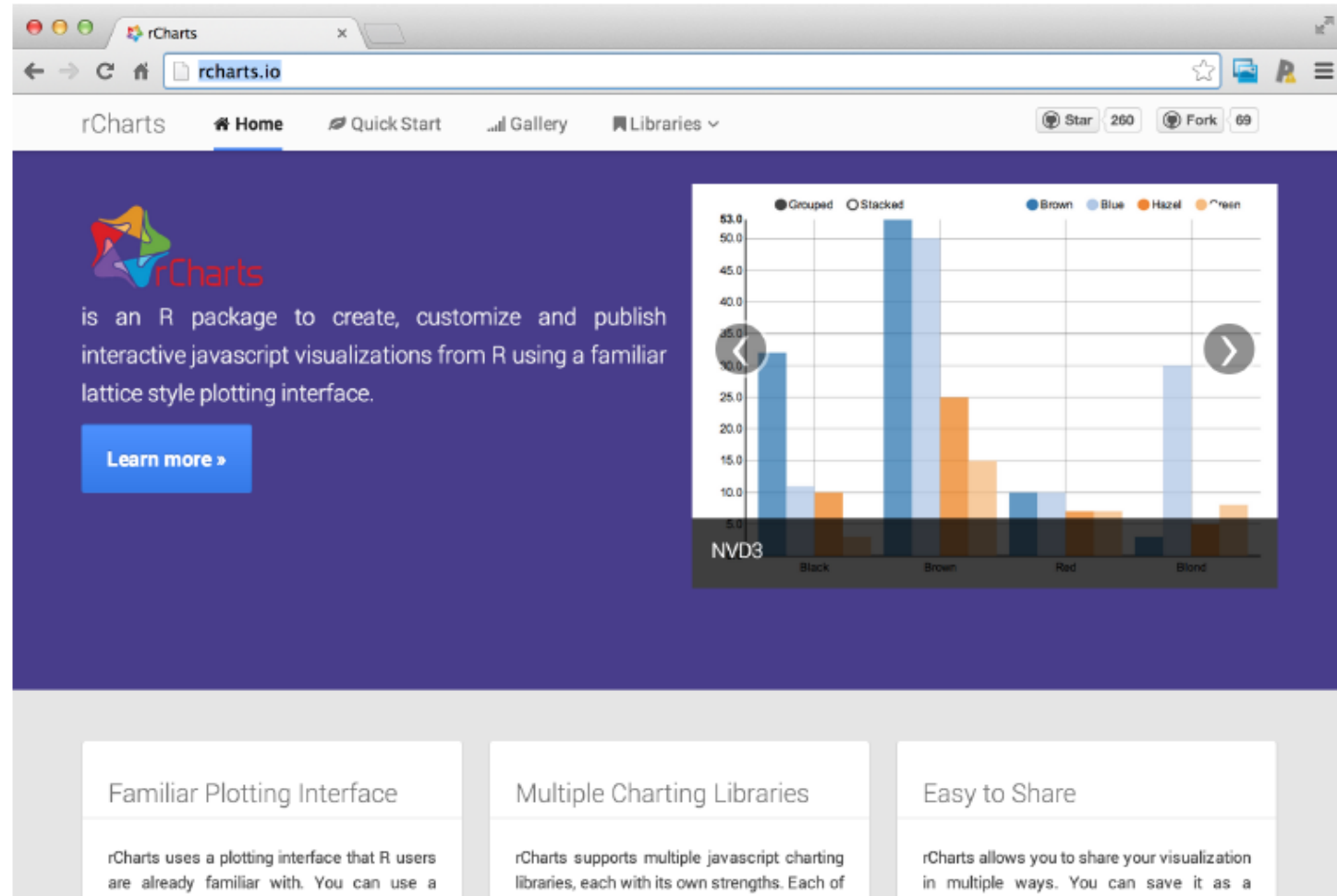
Package Development Basics

If you aren't already familiar with the basics of R package development, the following links provide additional documentation and tutorials:

- [Writing R Extensions](#)

<http://cran.r-project.org/web/packages/> <http://www.rstudio.com/ide/docs/packages/overview>

rCharts - for marketing



The screenshot shows the rCharts website in a web browser. The browser's address bar displays 'rcharts.io'. The website's navigation bar includes links for 'Home', 'Quick Start', 'Gallery', and 'Libraries', along with GitHub statistics: 260 stars and 69 forks. The main content area has a dark blue background. On the left, the rCharts logo is followed by the text: 'is an R package to create, customize and publish interactive javascript visualizations from R using a familiar lattice style plotting interface.' Below this is a blue 'Learn more >' button. On the right, a bar chart titled 'NVD3' is displayed. The chart has a legend with 'Grouped' (dark grey circle) and 'Stacked' (light grey circle) options, and a color legend with 'Brown', 'Blue', 'Hazel', and 'Green'. The x-axis is labeled with 'Black', 'Brown', 'Red', and 'Blond'. The y-axis ranges from 0.0 to 55.0. The chart shows grouped bars for each category, with the 'Brown' category having the highest values. Below the main content area, there are three white boxes with grey borders. The first box is titled 'Familiar Plotting Interface' and contains the text: 'rCharts uses a plotting interface that R users are already familiar with. You can use a'. The second box is titled 'Multiple Charting Libraries' and contains the text: 'rCharts supports multiple javascript charting libraries, each with its own strengths. Each of'. The third box is titled 'Easy to Share' and contains the text: 'rCharts allows you to share your visualization in multiple ways. You can save it as a'.

rCharts

Home Quick Start Gallery Libraries

Star 260 Fork 69

rCharts

is an R package to create, customize and publish interactive javascript visualizations from R using a familiar lattice style plotting interface.

[Learn more >](#)

NVD3

Grouped Stacked

Brown Blue Hazel Green

Black Brown Red Blond

55.0 50.0 45.0 40.0 35.0 30.0 25.0 20.0 15.0 10.0 5.0 0.0

Familiar Plotting Interface

rCharts uses a plotting interface that R users are already familiar with. You can use a

Multiple Charting Libraries

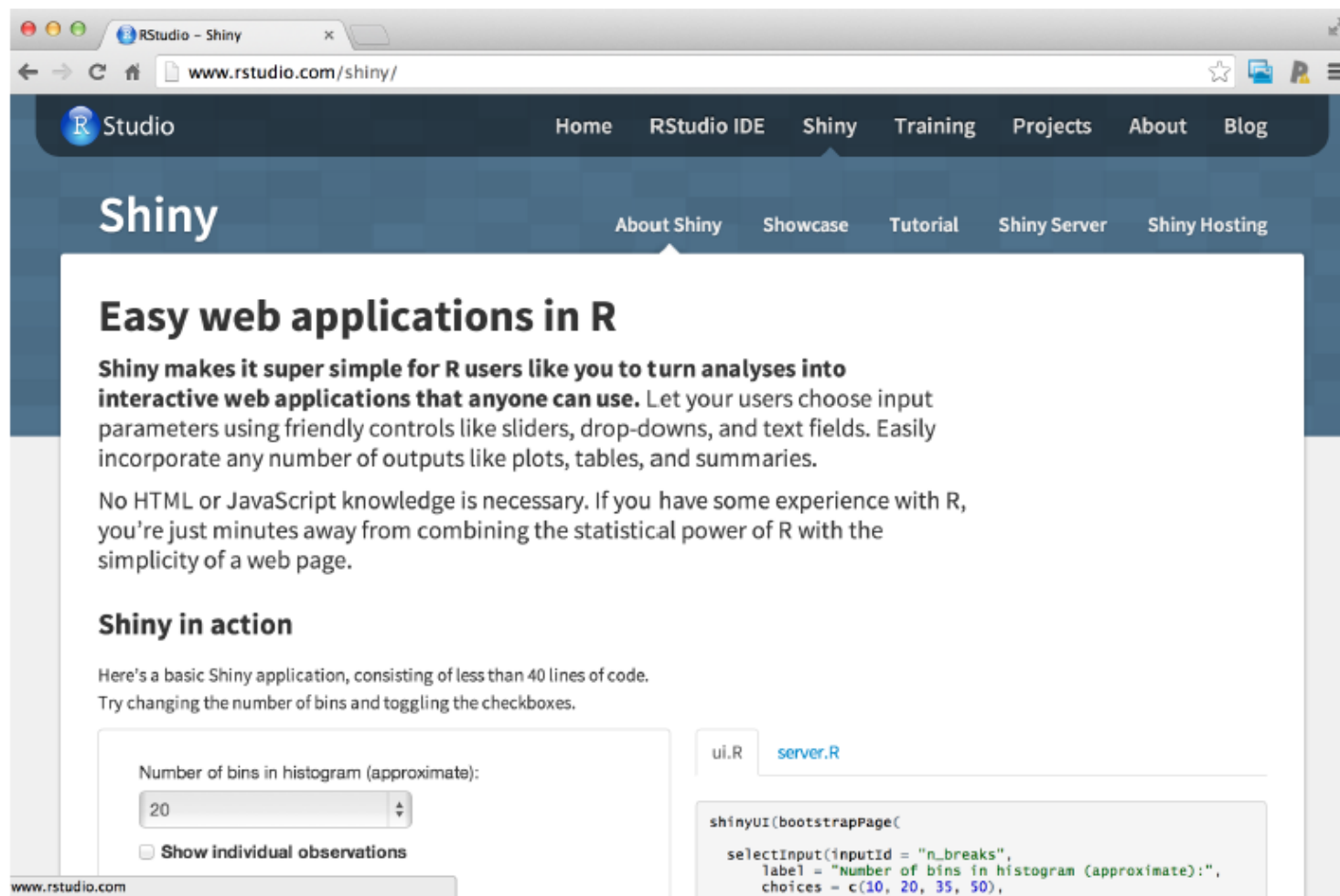
rCharts supports multiple javascript charting libraries, each with its own strengths. Each of

Easy to Share

rCharts allows you to share your visualization in multiple ways. You can save it as a

<http://rcharts.io/> <http://ramnathv.github.io/rChartsNYT/>

Shiny - for your users

A screenshot of a web browser displaying the RStudio Shiny website. The browser's address bar shows 'www.rstudio.com/shiny/'. The website has a dark blue header with navigation links: 'Studio', 'Home', 'RStudio IDE', 'Shiny', 'Training', 'Projects', 'About', and 'Blog'. Below this is a secondary navigation bar with 'About Shiny', 'Showcase', 'Tutorial', 'Shiny Server', and 'Shiny Hosting'. The main content area has a large heading 'Easy web applications in R' followed by a paragraph explaining that Shiny makes it simple for R users to create interactive web applications. It then states that no HTML or JavaScript knowledge is necessary. Below this is a section titled 'Shiny in action' with a brief description of a basic application. At the bottom, there is a live demo of a histogram application with a slider for 'Number of bins in histogram (approximate)' set to 20, a checkbox for 'Show individual observations', and a code editor showing the corresponding R code for the UI and server logic.

www.rstudio.com

RStudio

Home RStudio IDE Shiny Training Projects About Blog

Shiny

About Shiny Showcase Tutorial Shiny Server Shiny Hosting

Easy web applications in R

Shiny makes it super simple for R users like you to turn analyses into **interactive web applications that anyone can use**. Let your users choose input parameters using friendly controls like sliders, drop-downs, and text fields. Easily incorporate any number of outputs like plots, tables, and summaries.

No HTML or JavaScript knowledge is necessary. If you have some experience with R, you're just minutes away from combining the statistical power of R with the simplicity of a web page.

Shiny in action

Here's a basic Shiny application, consisting of less than 40 lines of code. Try changing the number of bins and toggling the checkboxes.

Number of bins in histogram (approximate):

20

☐ Show individual observations

ui.R server.R

```
shinyUI(bootstrapPage(  
  selectInput(inputId = "n_breaks",  
    label = "Number of bins in histogram (approximate):",  
    choices = c(10, 20, 35, 50),
```

<http://www.rstudio.com/shiny/> <http://www.rstudio.com/shiny/showcase/>

What is Shiny?

- Shiny is a platform for creating interactive R programs embedded into a web page.
- Suppose that you create a prediction algorithm, with shiny you can very easily create web input form that calls R and thus your prediction algorithm and displays the results.
- Using Shiny, the time to create simple, yet powerful, web-based interactive data products in R is minimized.
- Shiny is made by the fine folks at R Studio. . . . - However, it lacks the flexibility of full featured (and more complex) solutions.

Some mild prerequisites

Shiny doesn't really require it, but as with all web programming, a little knowledge of html, css and js is very helpful


- html gives a web page structure and sectioning as well as markup instructions
- css gives the style
- js for interactivity

There are too many tutorials online to count for getting basic proficiency in these topics to count.

Shiny uses [bootstrap](#) (no relation to the statistics bootstrap) style, which (to me) seems to look nice and renders well on mobile platforms



What else is out there?

- Creating any solution requiring fairly deep knowledge of web client/server programming
 - OpenCPU by Jerome Ooms, is a really neat project providing an API for calling R from web documents
 - And he even hosts an OpenCPU server, but you can create your own
- 

Context

You created a novel prediction algorithm to predict risk for developing diabetes.

- You want to create a web site so that users can input the relevant predictors and obtain their prediction.

Your prediction algorithm

- You're hoping patients and caregivers will be able to enter their data and, if needed, take preventative measures.
- `diabetesRisk <- function(glucose) glucose/200`

Getting started

- Make sure you have the latest release of R installed
- If on windows, make sure that you have Rtools
- `install.packages("shiny")`
- `library(shiny)`
- Great tutorial at <http://rstudio.github.io/shiny/tutorial/>
- Basically, this lecture is walking through that tutorial offering some of my insights
- Note, some of the proposed interactive plotting uses of Shiny could be handled by the very simple manipulate function [rstudio manipulate](#)

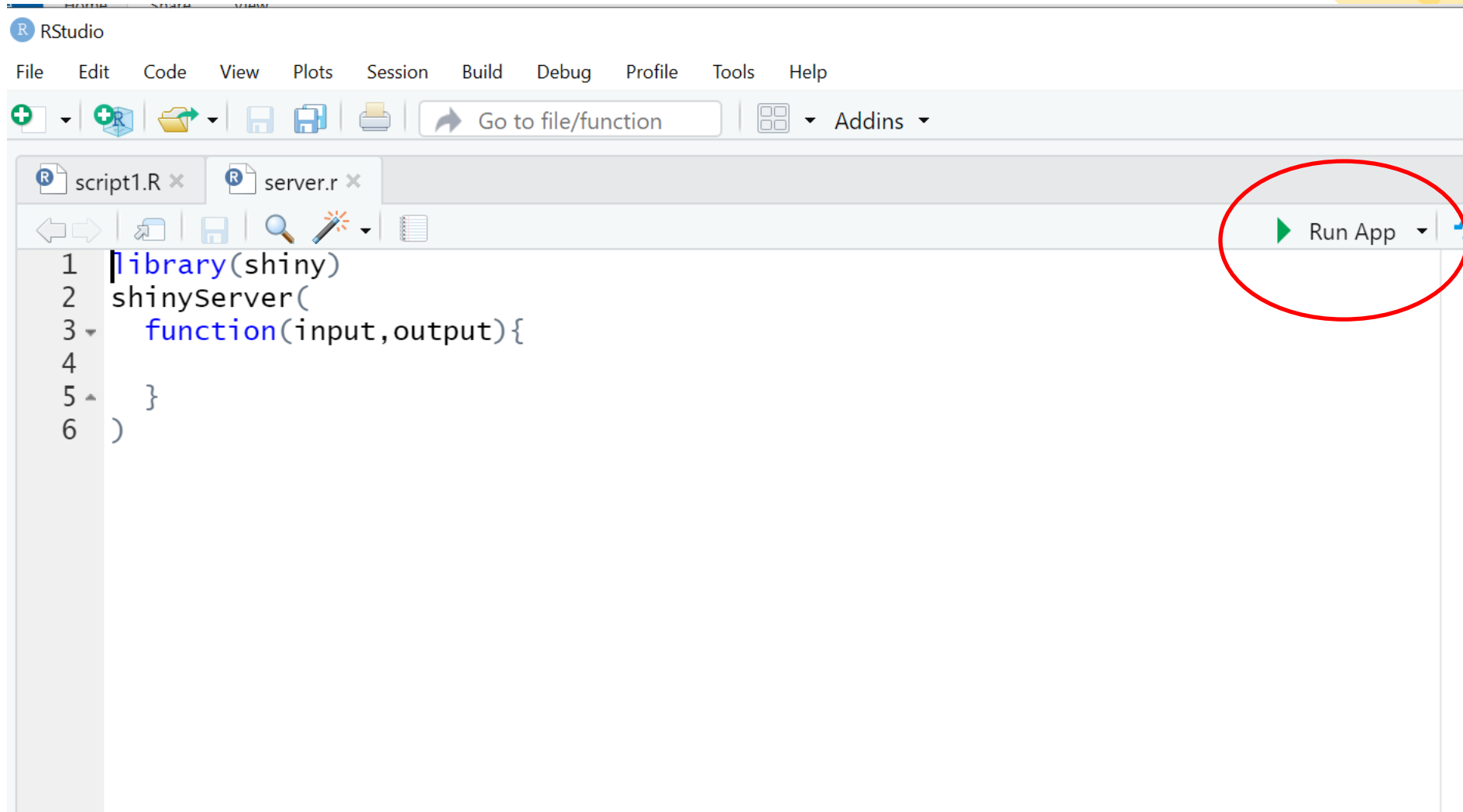
A Shiny project

- A shiny project is a directory containing at least two parts:
 - One named ui.R (for user interface) controls how it looks.
 - One named server.R that controls what it does.

To run it

- In R, change to the directories with these files and type `runApp()` or put the path to the directory as an argument. It should open an browser window with the app running.

To run it



HTML Content

shiny function	HTML5 equivalent	creates
<code>p</code>	<code><p></code>	A paragraph of text
<code>h1</code>	<code><h1></code>	A first level header
<code>h2</code>	<code><h2></code>	A second level header
<code>h3</code>	<code><h3></code>	A third level header
<code>h4</code>	<code><h4></code>	A fourth level header
<code>h5</code>	<code><h5></code>	A fifth level header
<code>h6</code>	<code><h6></code>	A sixth level header
<code>a</code>	<code><a></code>	A hyper link
<code>br</code>	<code>
</code>	A line break (e.g. a blank line)
<code>div</code>	<code><div></code>	A division of text with a uniform style
<code>span</code>	<code></code>	An in-line division of text with a uniform style
<code>pre</code>	<code><pre></code>	Text 'as is' in a fixed width font
<code><u>code</u></code>	<code><code></code>	A formatted block of code
<code>img</code>	<code></code>	An image
<code>strong</code>	<code></code>	Bold text
<code>em</code>	<code></code>	Italicized text
<code>HTML</code>		Directly passes a character string as HTML code

Basic widgets

Buttons

Action

Submit

Single checkbox

☒ Choice A

Checkbox group

☒ Choice 1

☐ Choice 2

☐ Choice 3

Date input

2014-01-01

Date range

2017-06-21

to

2017-06-21

File input

Browse...

No file selected

Help text

Note: help text isn't a true widget, but it provides an easy way to add text to accompany other widgets.

Numeric input

1

Radio buttons

☒ Choice 1

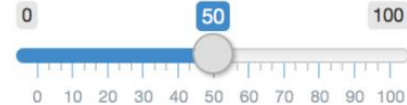
☐ Choice 2

☐ Choice 3

Select box

Choice 1

Sliders



Text input

Enter text...

The standard Shiny widgets are:

function

actionButton

checkboxGroupInput

checkboxInput

dateInput

dateRangeInput

fileInput

helpText

numericInput

radioButtons

selectInput

sliderInput

submitButton

textInput

widget

Action Button

A group of check boxes

A single check box

A calendar to aid date selection

A pair of calendars for selecting a date range

A file upload control wizard

Help text that can be added to an input form

A field to enter numbers

A set of radio buttons

A box with choices to select from

A slider bar

A submit button

A field to enter text

Display reactive output

Two steps

You can create reactive output with a two step process.

1. Add an R object to your user interface.
2. Tell Shiny how to build the object in the server function. The object will be reactive if the code that builds it calls a widget value.

Step 1: Add an R object to the UI

Output function	Creates
dataTableOutput	DataTable
htmlOutput	raw HTML
imageOutput	image
plotOutput	plot
tableOutput	table
<u>textOutput</u>	text
uiOutput	raw HTML
verbatimTextOutput	text

Step 2: Provide R code to build the object.

render function

renderDataTable

renderImage

renderPlot

renderPrint

renderTable

renderText

renderUI

creates

DataTable

images (saved as a link to a source file)

plots

any printed output

data frame, matrix, other table like structures

character strings

a Shiny tag object or HTML

Images

```
img(src = "my_image.png", height = 72, width = 72)
```


```
ui <- fluidPage(  
  titlePanel("My Shiny App"),  
  sidebarLayout(  
    sidebarPanel(),  
    mainPanel(  
      img(src = "rstudio.png", height = 140, width = 400)  
    )  
  )  
)
```

My Shiny App





Image example

- Let's build an example with an image
 - How about we create a histogram of data
 - Put a slider on so that the user has to guess the mean
- 

ui.R

```
library(shiny)
shinyUI(pageWithSidebar(
  headerPanel("Data science FTW!"),
  sidebarPanel(
    h3('Sidebar text') ← h3("Third level title")
  ),
  mainPanel(
    h3('Main Panel text')
  )
))
```

server.r

```
library(shiny)
shinyServer(
  function(input, output) {
  }
)
```



Output

Hello Shiny!

Sidebar text

Main Panel text



R functions for HTML markup

ui.R

```
shinyUI(pageWithSidebar(  
  headerPanel("Illustrating markup"),  
  sidebarPanel(  
    h1('Sidebar panel'),  
    h1('H1 text'), —————> h1("First level title")  
    h2('H2 Text'),  
    h3('H3 Text'), —————> h3("Third level title")  
    h4('H4 Text')  
  
  ),  
  mainPanel(  
    h3('Main Panel text'),  
    code('some code'),  
    p('some ordinary text')  
  )  
))
```



Illustrating markup

Sidebar panel

H1 text

H2 Text

H3 Text

H4 Text

Main Panel text

`some code`

some ordinary text



ui.R

```
shinyUI(pageWithSidebar(  
  headerPanel("Illustrating inputs"),  
  sidebarPanel(  
    numericInput('id1', 'Numeric input, labeled id1', 0, min = 0, max = 10, step = 1),  
    checkboxGroupInput("id2", "Checkbox",  
      c("Value 1" = "1",  
        "Value 2" = "2",  
        "Value 3" = "3")),  
    dateInput("date", "Date:")  
  ),  
  mainPanel(  
  
  )  
))
```

Illustrating inputs

Illustrating inputs

Numeric input, label id1

Checkbox

☐ Value 1

☐ Value 2

☐ Value 3

Date:

Let's build our prediction function

ui.R

```
shinyUI(  
  pageWithSidebar(  
    # Application title  
    headerPanel("Diabetes prediction"),  
  
    sidebarPanel(  
      numericInput('glucose', 'Glucose mg/dl', 90, min = 50, max = 200, step = 5),  
      submitButton('Submit')  
    ),  
    mainPanel(  
      h3('Results of prediction'),  
      h4('You entered'),  
      verbatimTextOutput("inputValue"),  
      h4('Which resulted in a prediction of '),  
      verbatimTextOutput("prediction")  
    )  
  )  
)
```

Server. R

```
diabetesRisk <- function(glucose) glucose / 200

shinyServer(
  function(input, output) {
    output$inputValue <- renderPrint({input$glucose})
    output$prediction <- renderPrint({diabetesRisk(input$glucose)})
  }
)
```

The result

Diabetes prediction

Glucose mg/dl

Submit

Results of prediction

You entered

```
[1] 120
```

Which resulted in a prediction of

```
[1] 0.6
```

ui.R

```
shinyUI(pageWithSidebar(  
  headerPanel("Example plot"),  
  sidebarPanel(  
    sliderInput('mu', 'Guess at the mean', value = 70, min = 62, max = 74, step = 0.05,  
  ),  
  mainPanel(  
    plotOutput('newHist')  
  )  
))
```

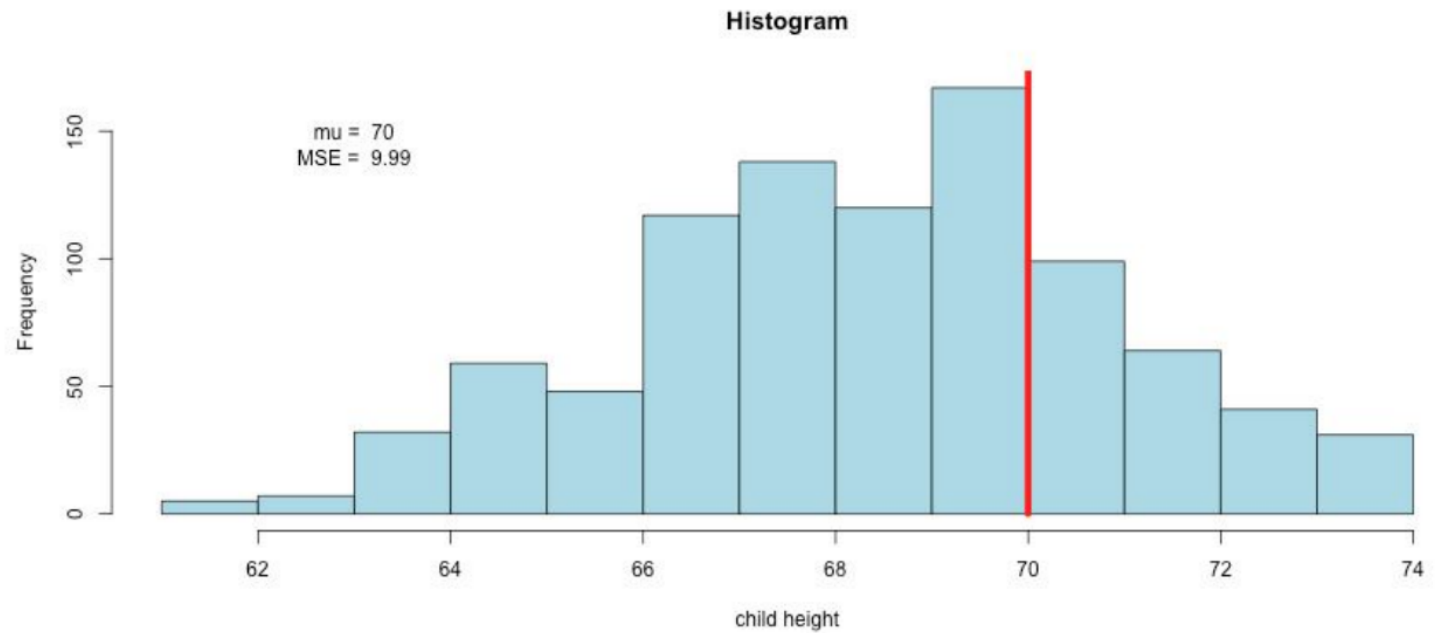
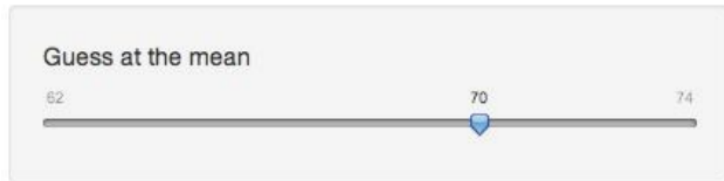

server.R

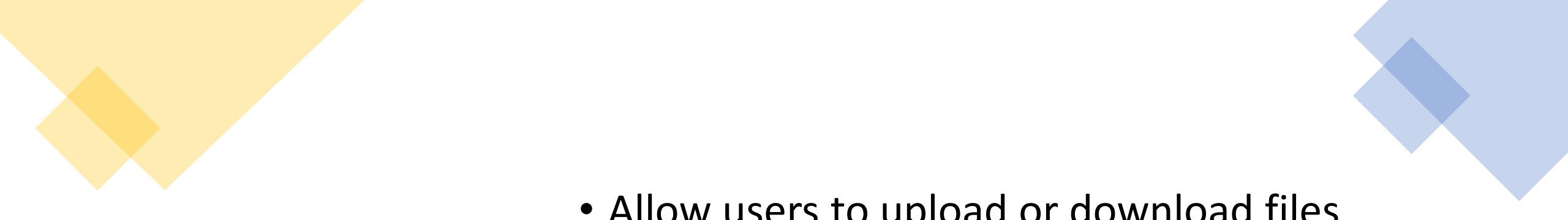
```
library(UsingR)
data(galton)

shinyServer(
  function(input, output) {
    output$newHist <- renderPlot({
      hist(galton$child, xlab='child height', col='lightblue',main='Histogram')
      mu <- input$mu
      lines(c(mu, mu), c(0, 200),col="red",lwd=5)
      mse <- mean((galton$child - mu)^2)
      text(63, 150, paste("mu = ", mu))
      text(63, 140, paste("MSE = ", round(mse, 2)))
    })
  }
)
```


The output

Example plot



- 
- Allow users to upload or download files
 - Have tabbed main panels Have editable data tables

Other things Shiny can do

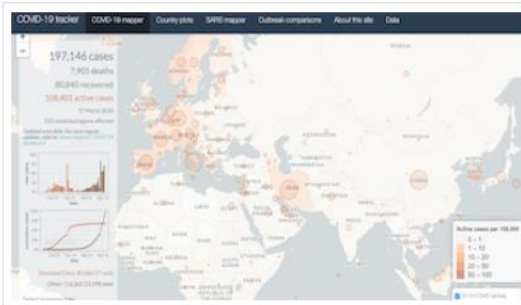
- Have a dynamic UI User defined inputs and outputs
 - Put a submit button so that Shiny only executes complex code after user hits submit
- 

https://shiny.rstudio.com/gallery/

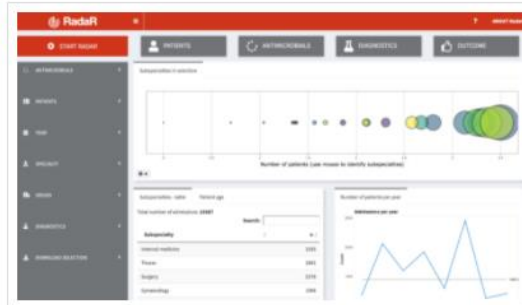


Compare the cycling speed of cities

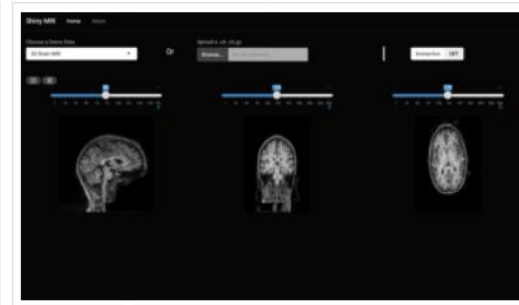
Life sciences



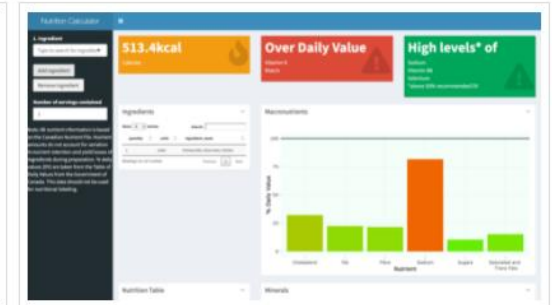
COVID-19 tracker



Exploring large hospital data for better use of antimicrobials



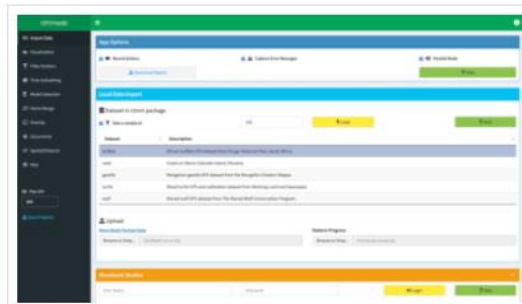
ShinyMRI - View MRI images in Shiny



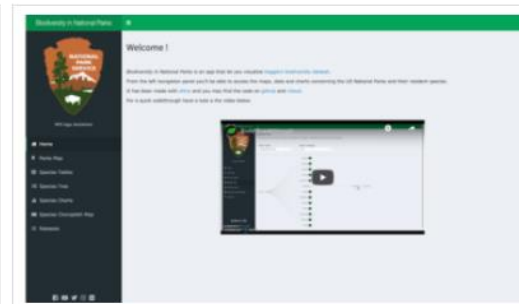
Nutrition Calculator - calculate nutrition for recipes



A/B Testing Sample Size Calculator



ctmmweb, a web app to analysis Animal tracking data



Visualizing Biodiversity in National Parks data



ExPanD: Explore Your Data Interactively

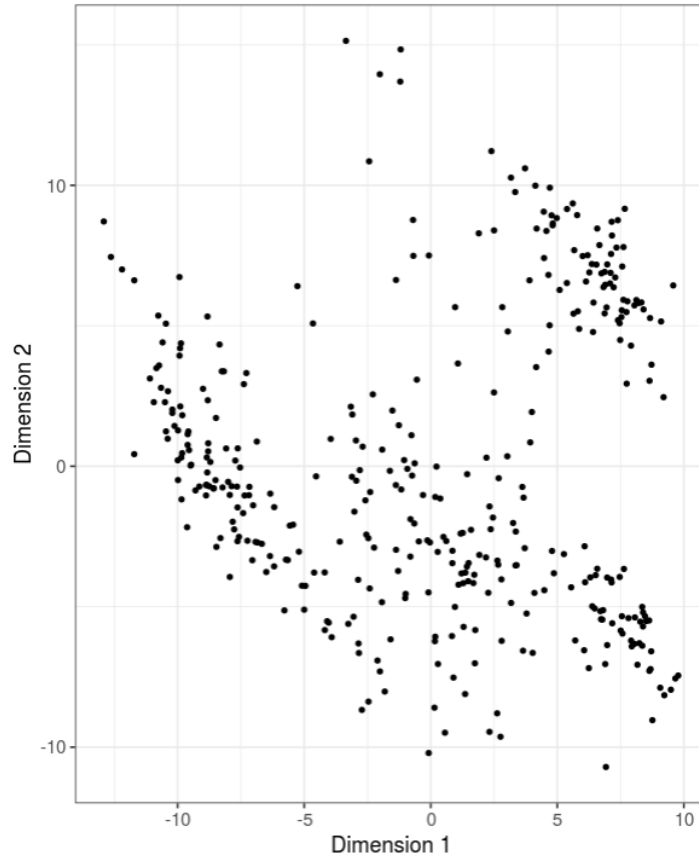
<https://shiny.rstudio.com/gallery/>

← → ↻ kevinrue.shinyapps.io/isee-shiny-contest/?_ga=2.162676988.1701018618.1623957089-1226216896.1615911563



Update

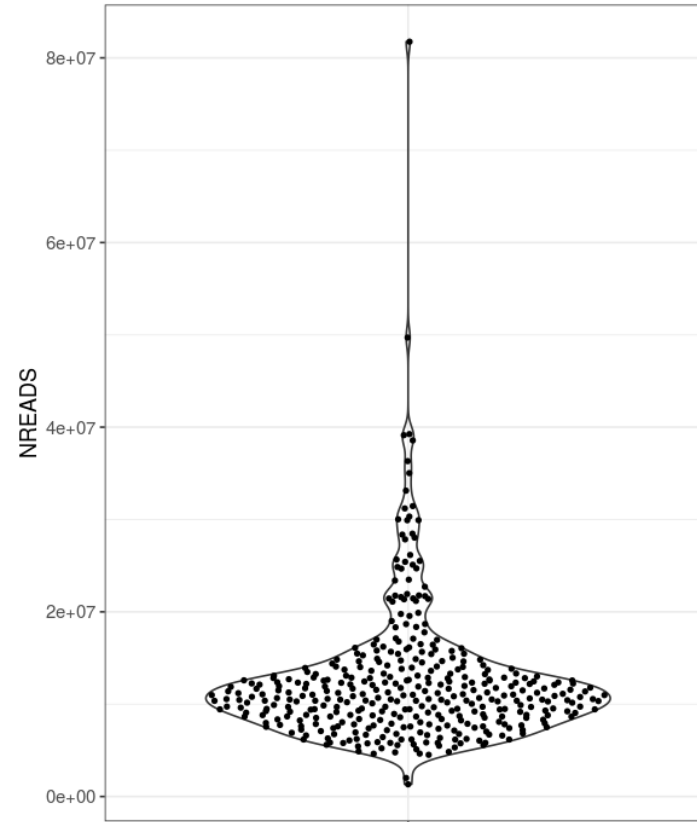
(1) PCA



Data parameters

Visual parameters

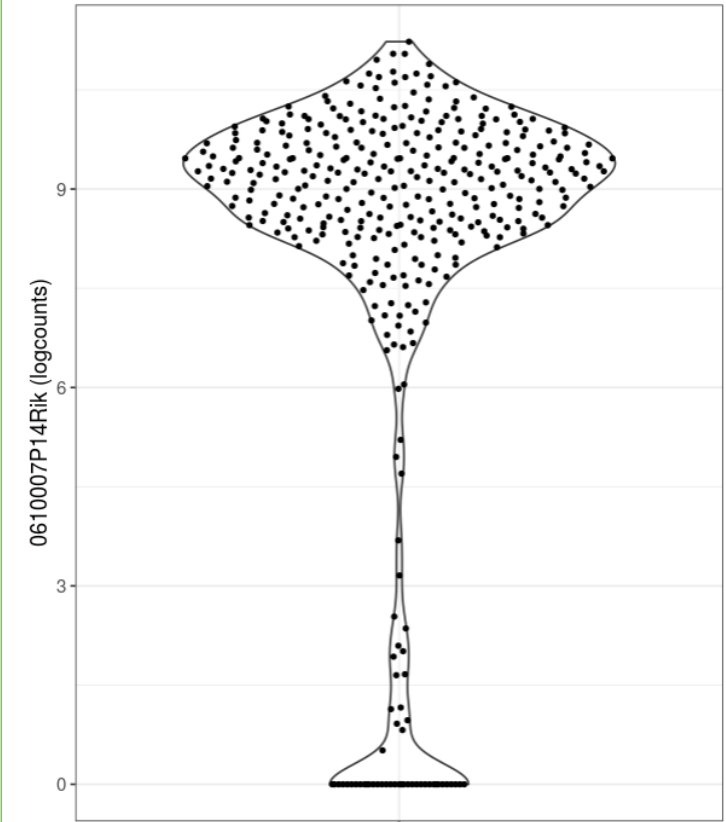
NREADS



Data parameters

Visual parameters

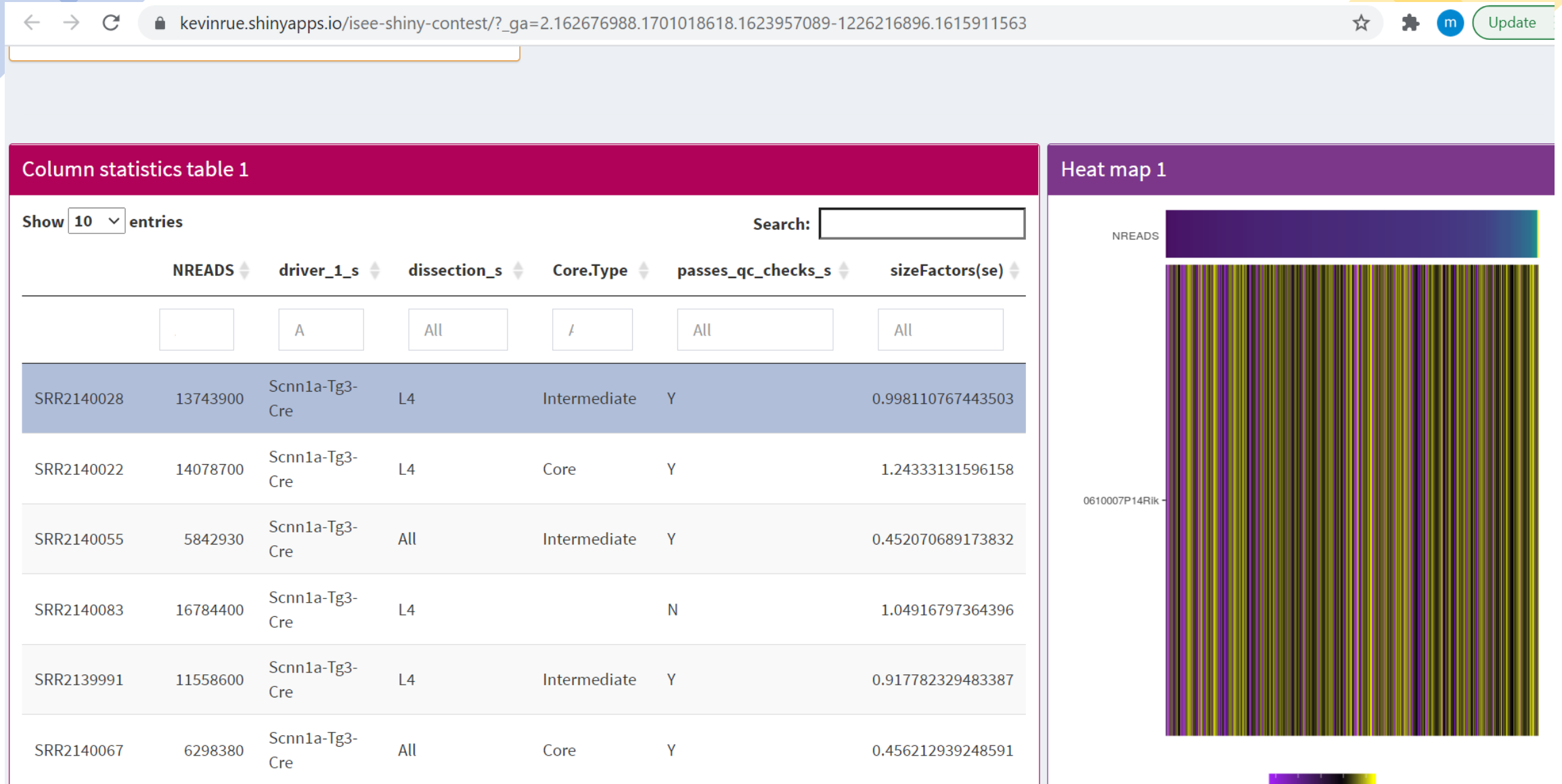
0610007P14Rik



Data parameters

Visual parameters

https://shiny.rstudio.com/gallery/





Manipulate

Data Products





Manipulate

- Suppose that you want to create a quick interactive graphic
 - You have to do it *now*
 - The intended users also use Rstudio
- `manipulate` is a really cool solution that is often all you need to quickly make interactive graphics

Documentation

- Manipulate is well documented at the Rstudio web site here
 - <http://www.rstudio.com/ide/docs/advanced/manipulate>
- From there, try this `library(manipulate) manipulate(plot(1:x), x = slider(1, 100))`
- You can create a slider, checkbox, or picker (drop down) and have more than one

Example

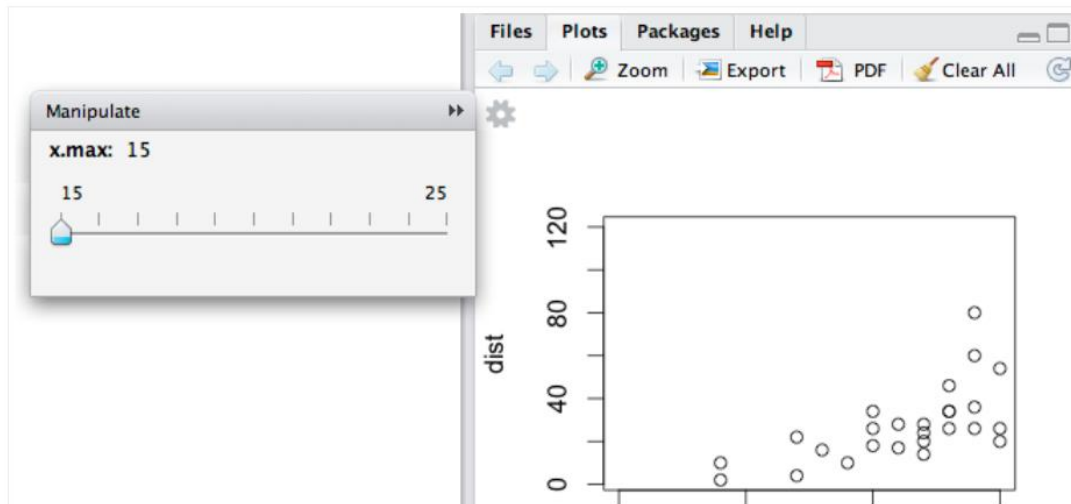
```
library(manipulate)
manipulate(plot(1:x), x = slider(1, 100))
```

Slider Control

The `slider` control enables manipulation of plot variables along a numeric range. For example:

```
manipulate(
  plot(cars, xlim=c(0,x.max)),
  x.max=slider(15,25))
```

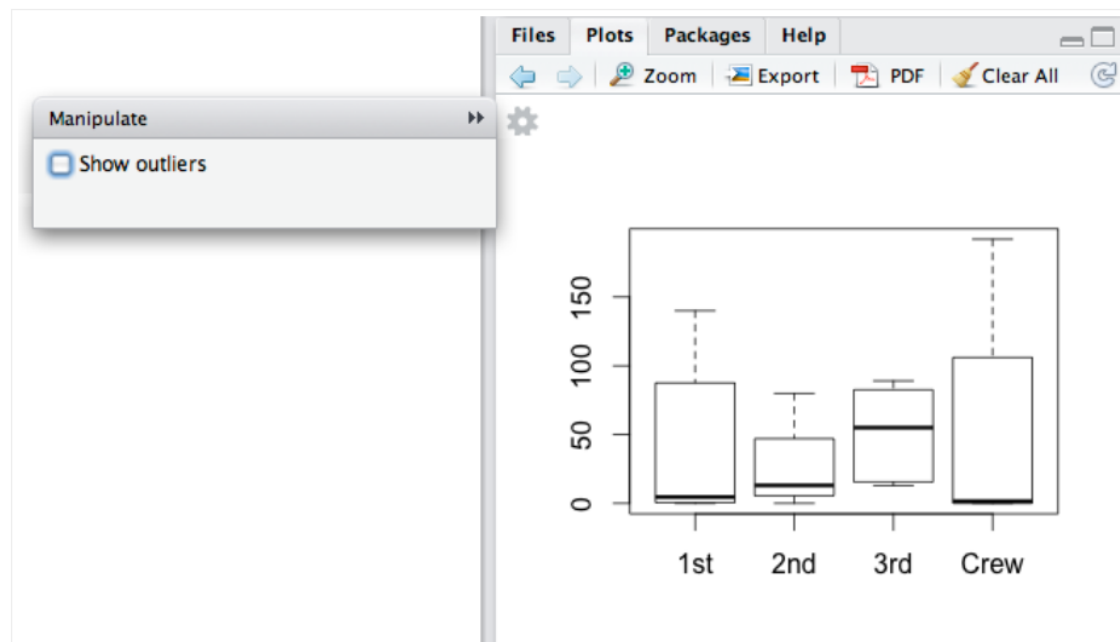
Results in this plot and manipulator:



The checkbox control enables manipulation of logical plot variables. For example:

```
manipulate(  
  boxplot(Freq ~ Class, data = Titanic, outline = outline),  
  outline = checkbox(FALSE, "Show outliers"))
```

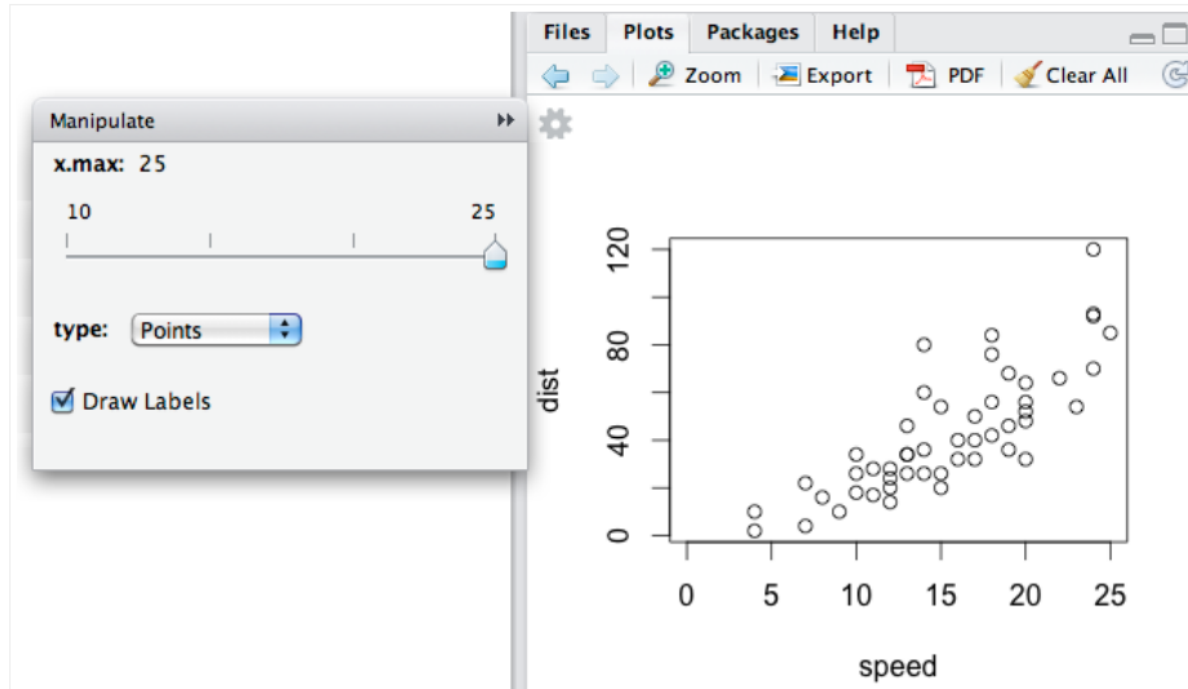
Results in this plot and manipulator:



Multiple controls can be combined within a single manipulator. For example:

```
manipulate(  
  plot(cars, xlim = c(0, x.max), type = type, ann = label),  
  x.max = slider(10, 25, step=5, initial = 25),  
  type = picker("Points" = "p", "Line" = "l", "Step" = "s"),  
  label = checkbox(TRUE, "Draw Labels"))
```

Results in this plot and manipulator:



Example

```
library(manipulate)
myHist <- function(mu){
  hist(galton$child,col="blue",breaks=100)
  lines(c(mu, mu), c(0, 150),col="red",lwd=5)
  mse <- mean((galton$child - mu)^2)
  text(63, 150, paste("mu = ", mu))
  text(63, 140, paste("MSE = ", round(mse, 2)))
}
manipulate(myHist(mu), mu = slider(62, 74, step = 0.5))
```