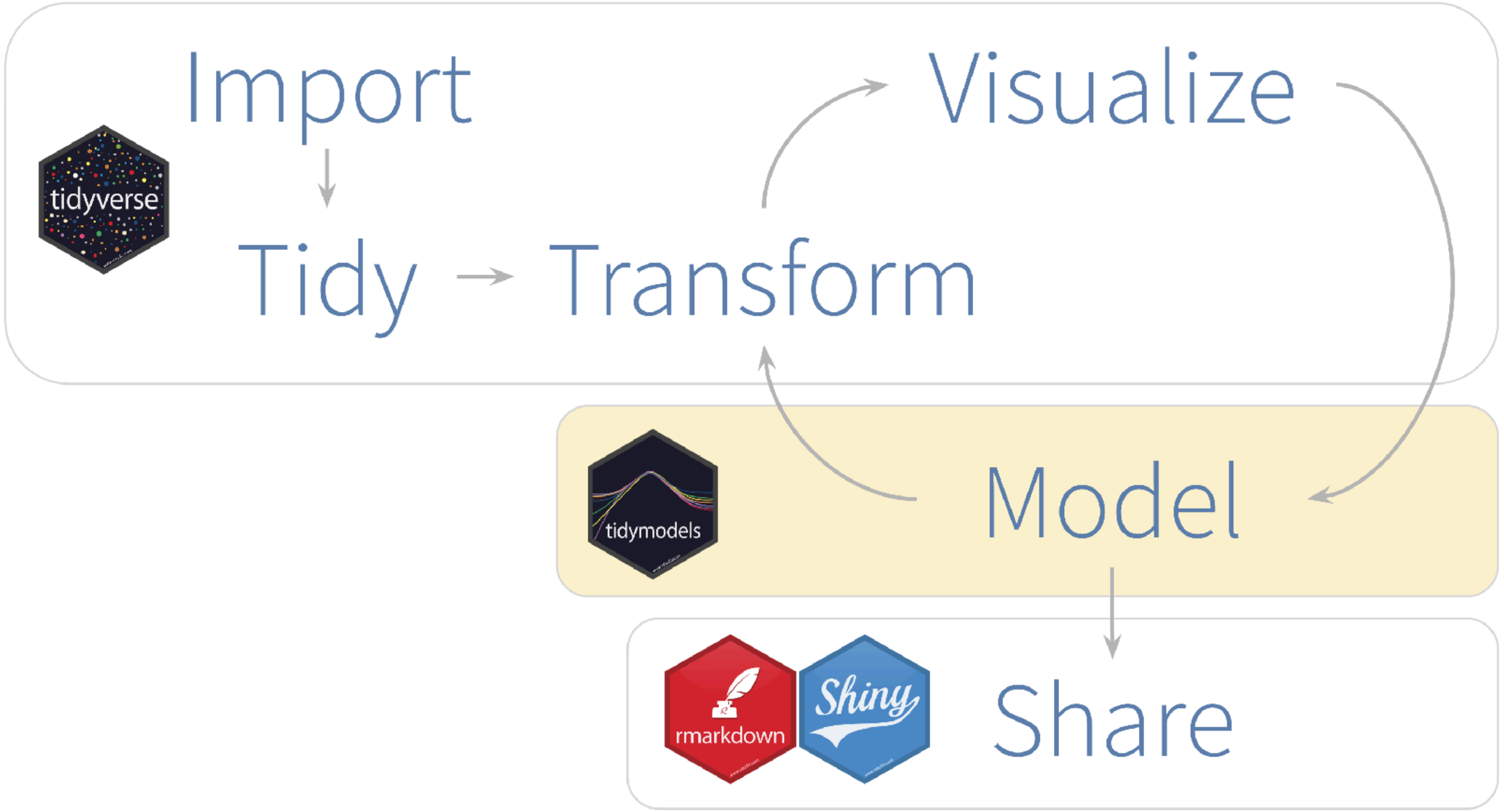
Why Shiny is just like sushi ...

Taryn Morris
taryn@ixioanalytics.com
16 April 2020

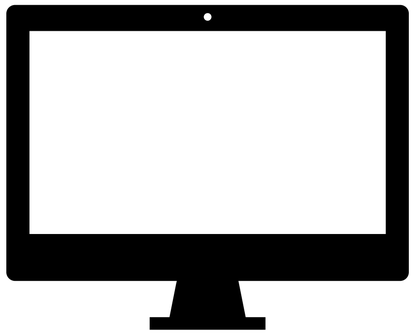
R Ladies CAPE TOWN

IXIO
PREDICT. BETTER. GROW.


Import
tidyverse
Tidy
Transform
Visualize
tidymodels
Model
rmarkdown
Shiny
Share

https://vac-lshtm.shinyapps.io/ncov_tracker/

https://alhill.shinyapps.io/COVID19seir/



DEMO

shiny.rstudio.com/

RStudio Products Resources Pricing About Us Blog

Shiny Apps for the Enterprise

Shiny Dashboard Demo

A dashboard built with Shiny.

Location tracker

Track locations over time with streaming data.

Download monitor

Streaming download rates visualized as a bubble chart.

Supply and Demand

Forecast demand to plan resource allocation.

Industry Specific Shiny Apps

Economic Dashboard

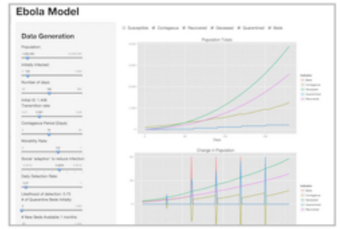
Economic forecasting with macroeconomic indicators.

ER Optimization

An app that models patient flow.

CDC Disease Monitor

Alert thresholds and automatic weekly updates.

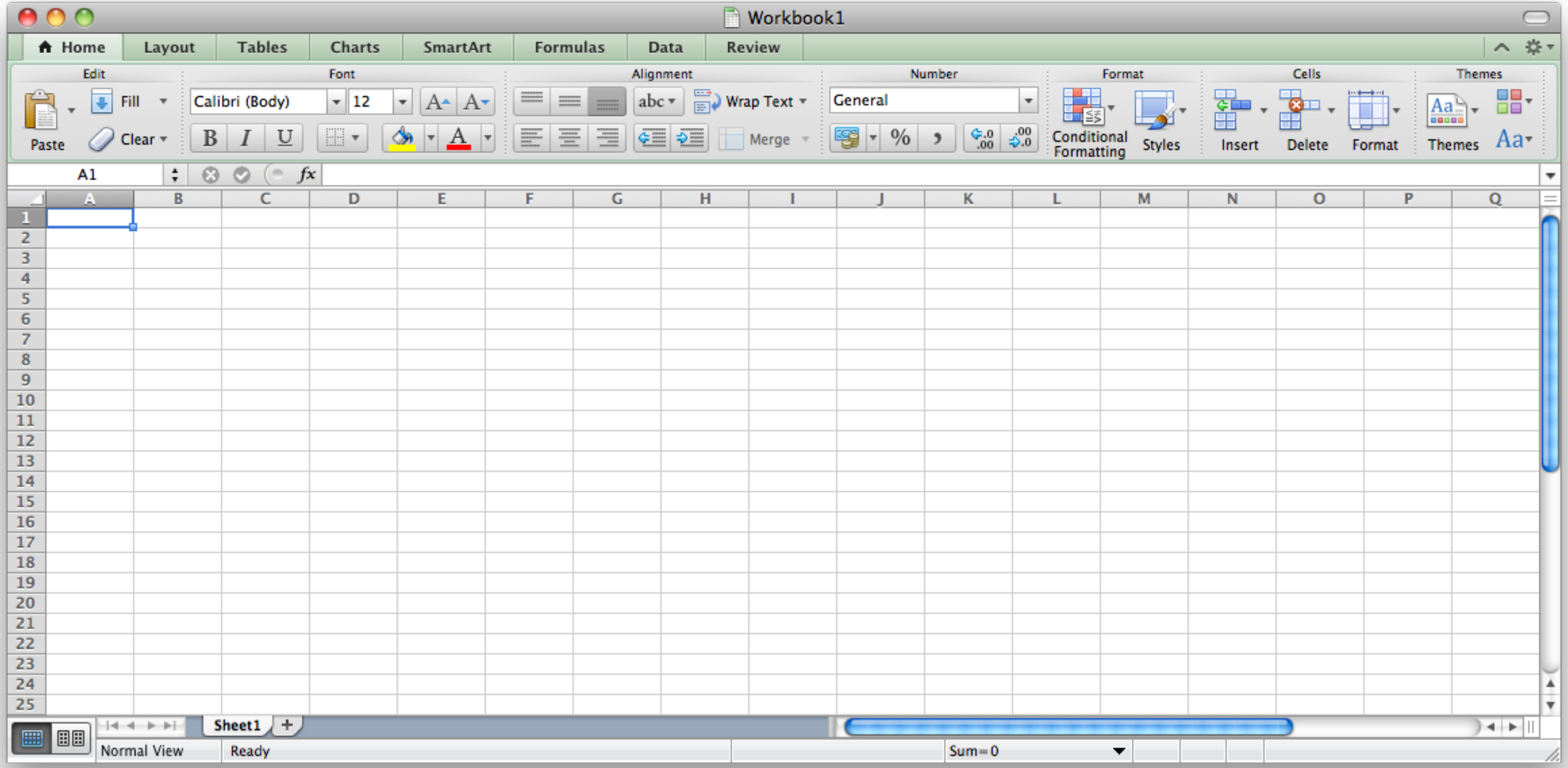


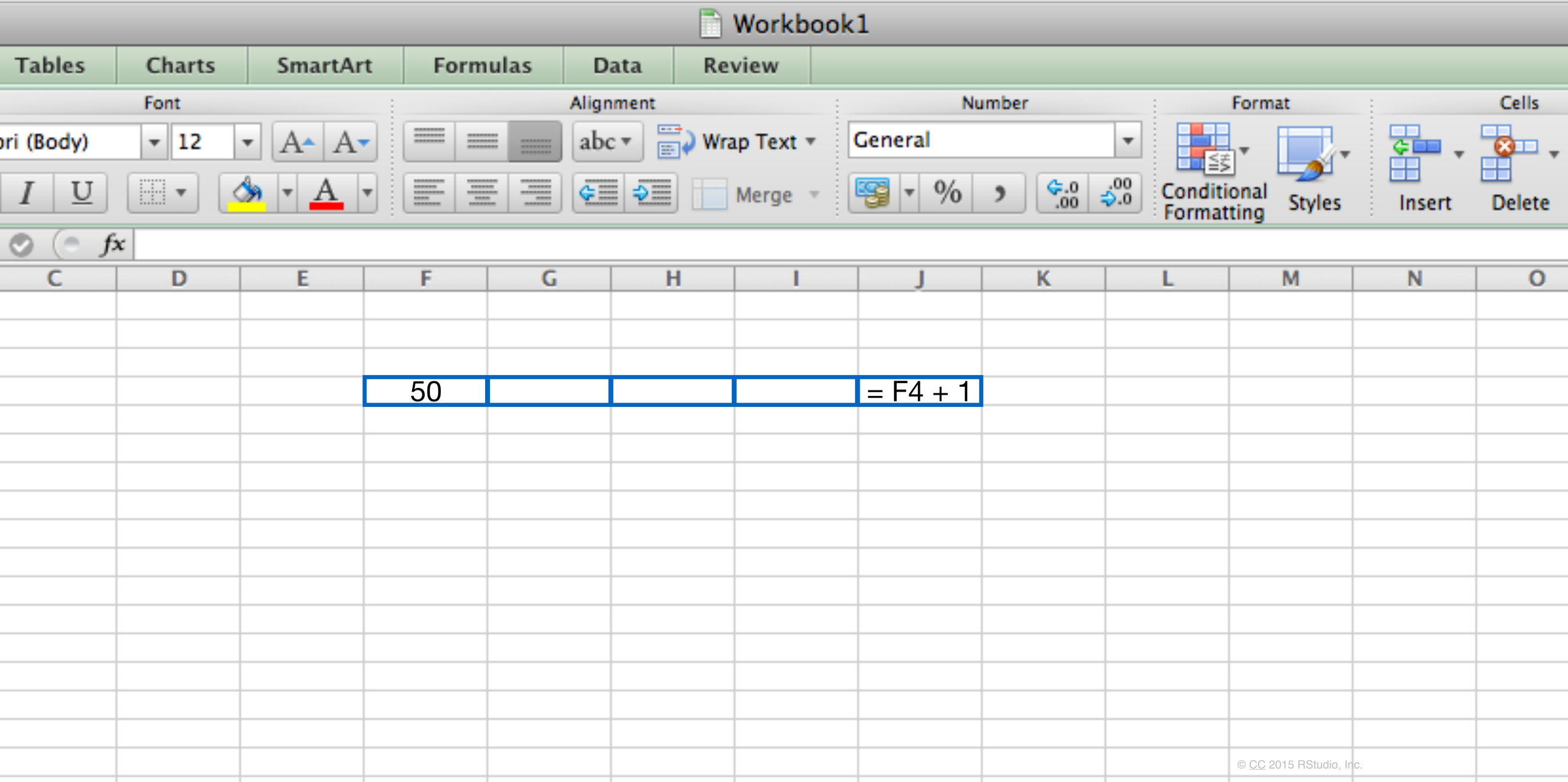
Ebola Model

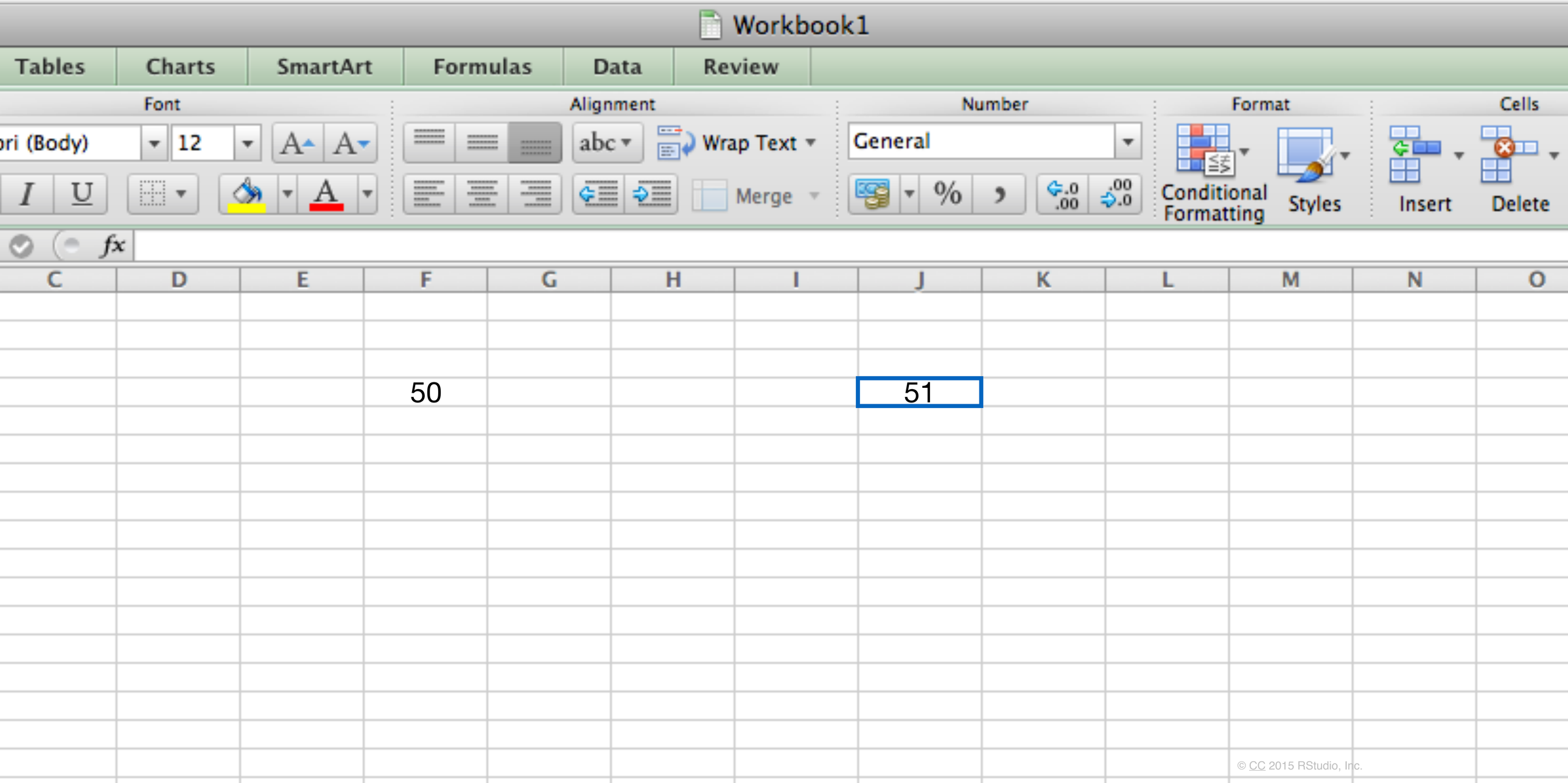
An epidemiological simulation.

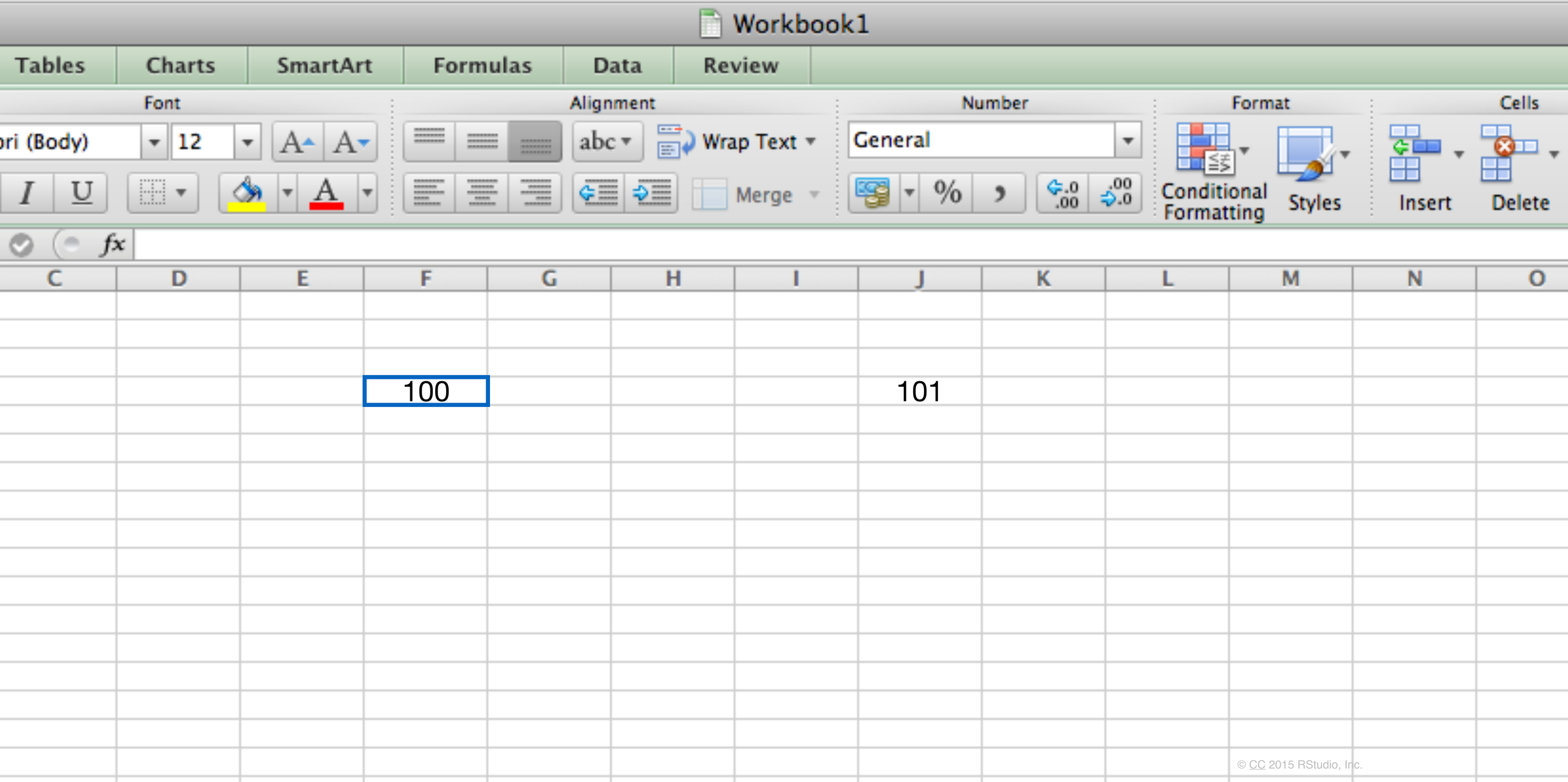
What is Reactivity?

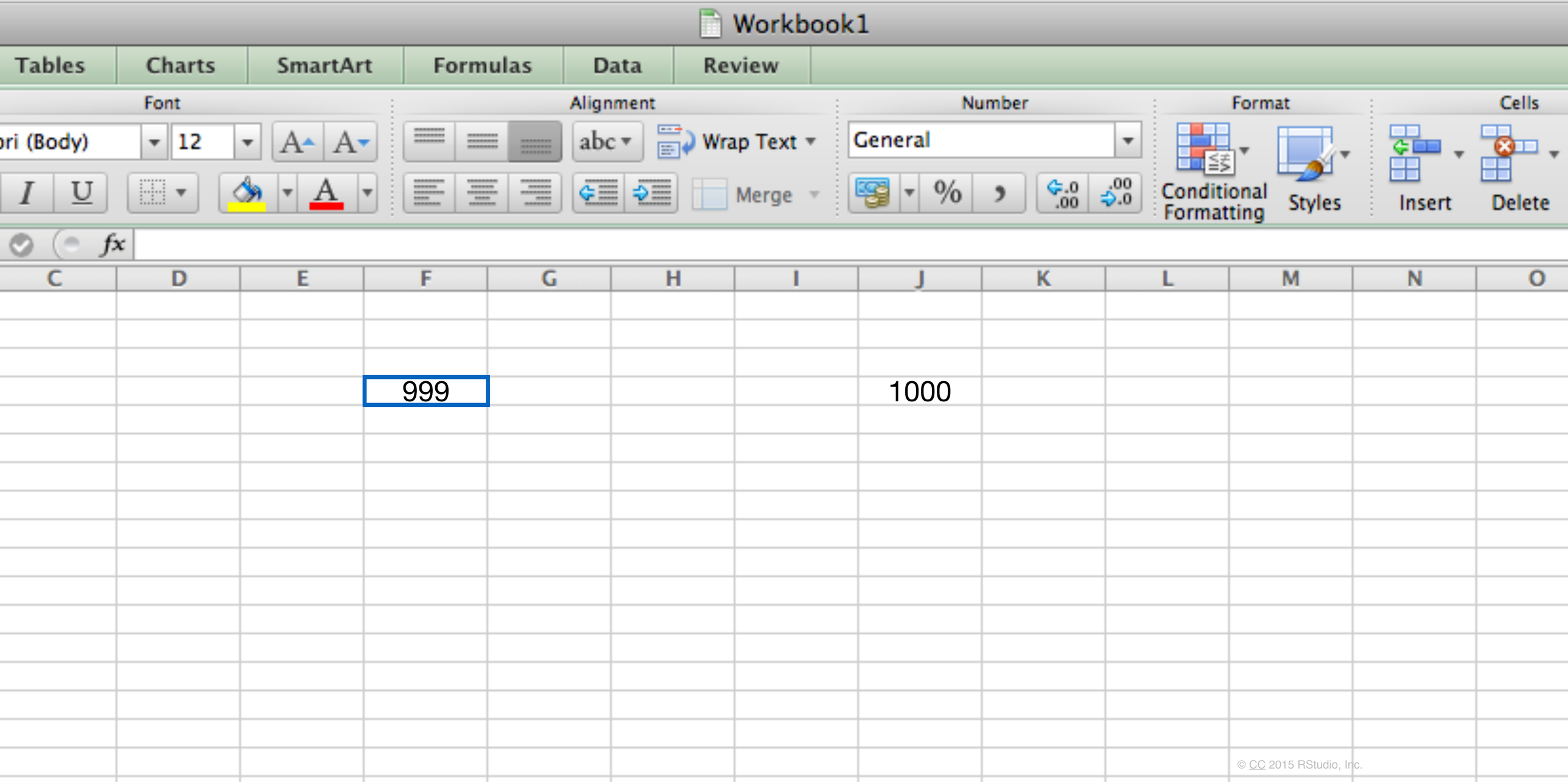
Think Excel.





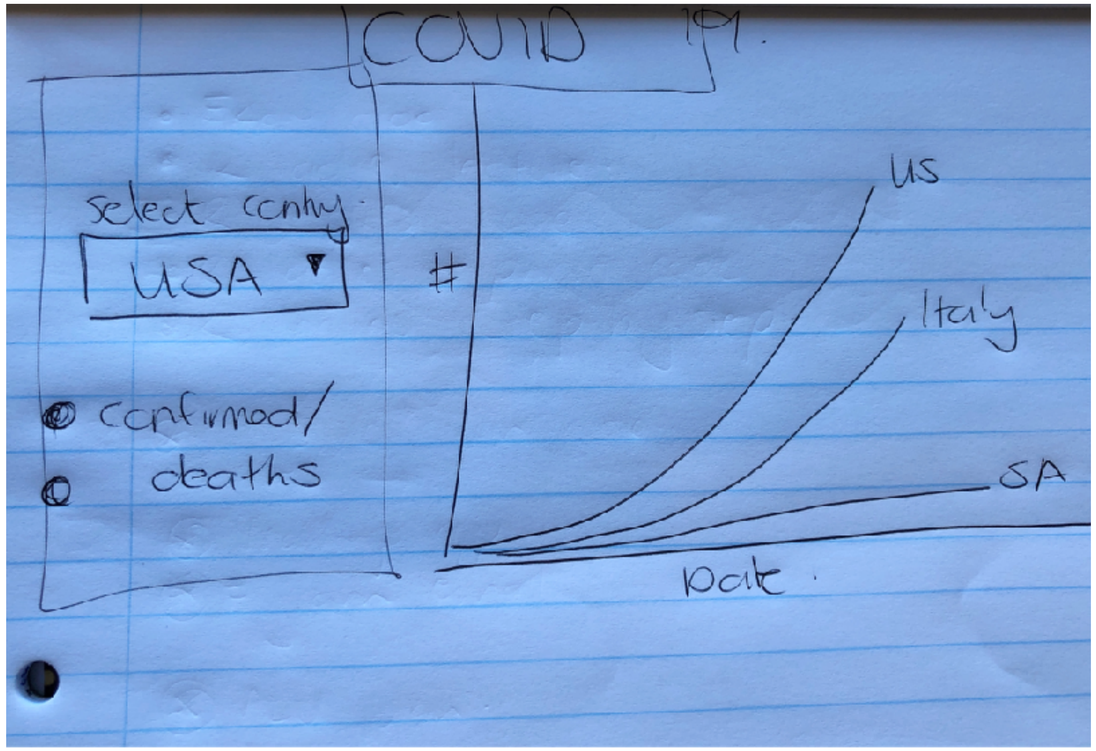






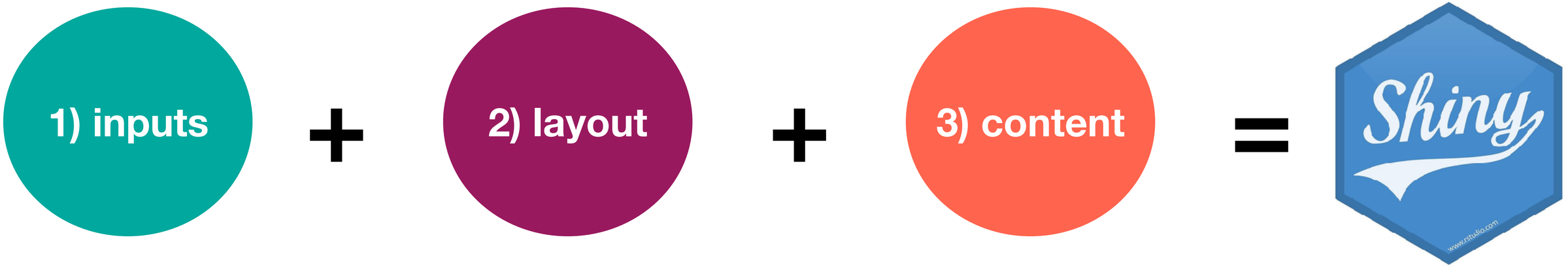
Where to start:

1) inputs



2) layout

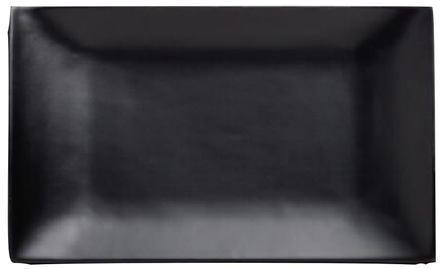
3) content



Why Shiny is just like sushi ...

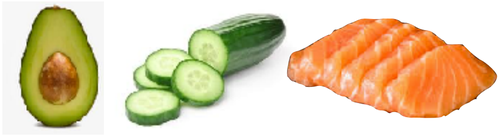
1) inputs + 2) layout + 3) content = Shiny

+



+

=



What's in an app?

```
library(shiny)

ui <- fluidPage()

Put a slider input 1:100

Have a heading followed by
a graph and then a table

server <- function(input, output) {}

Make an awesome
gg-plot scatterplot

shinyApp(ui = ui, server = server)
```

User interface
controls the options and layout of app

1) inputs

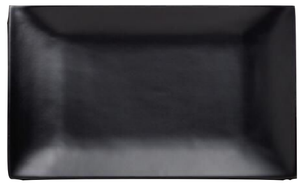
2) layout

Server function
contains recipes needed to build visuals

3) content

What's in an app?

```
library(shiny)

ui <- fluidPage()
```



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

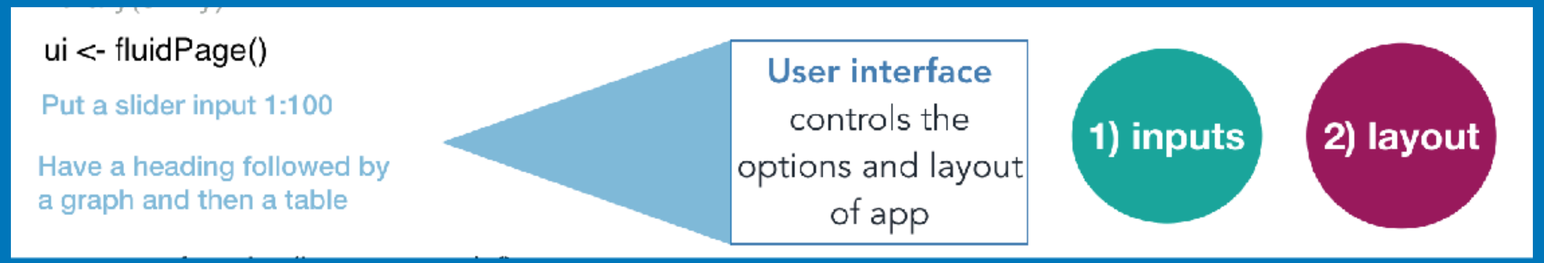
```
shinyApp(ui = ui, server = server)
```

1) inputs

2) layout

3) content

USER INTERFACE (UI)



UI

Add elements to your app as arguments to `fluidPage()`

```
ui <- fluidPage(
  # *Input() functions,
  # *Output() functions
)
```

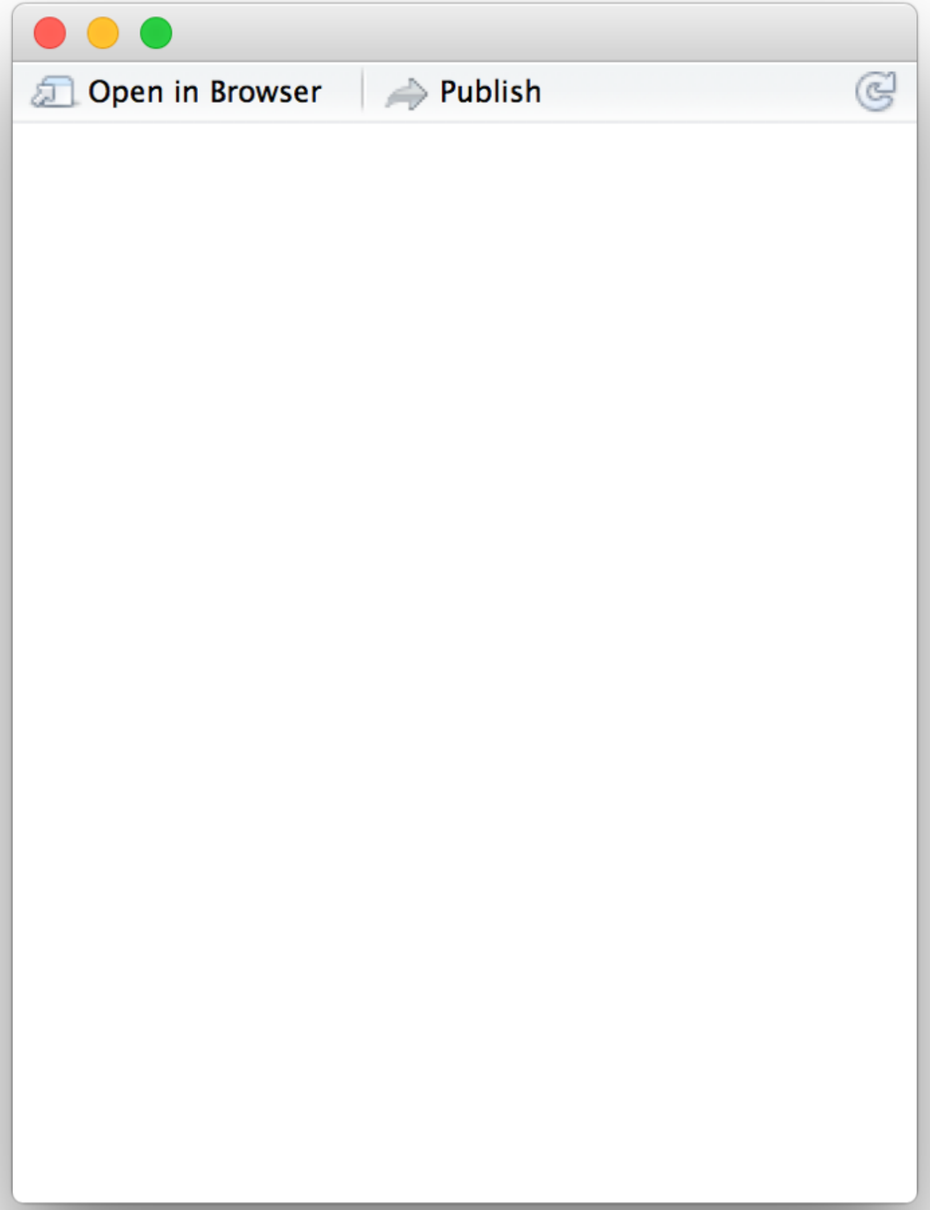
USER INTERFACE (UI)

Inputs

Create an input with an input function.

UI

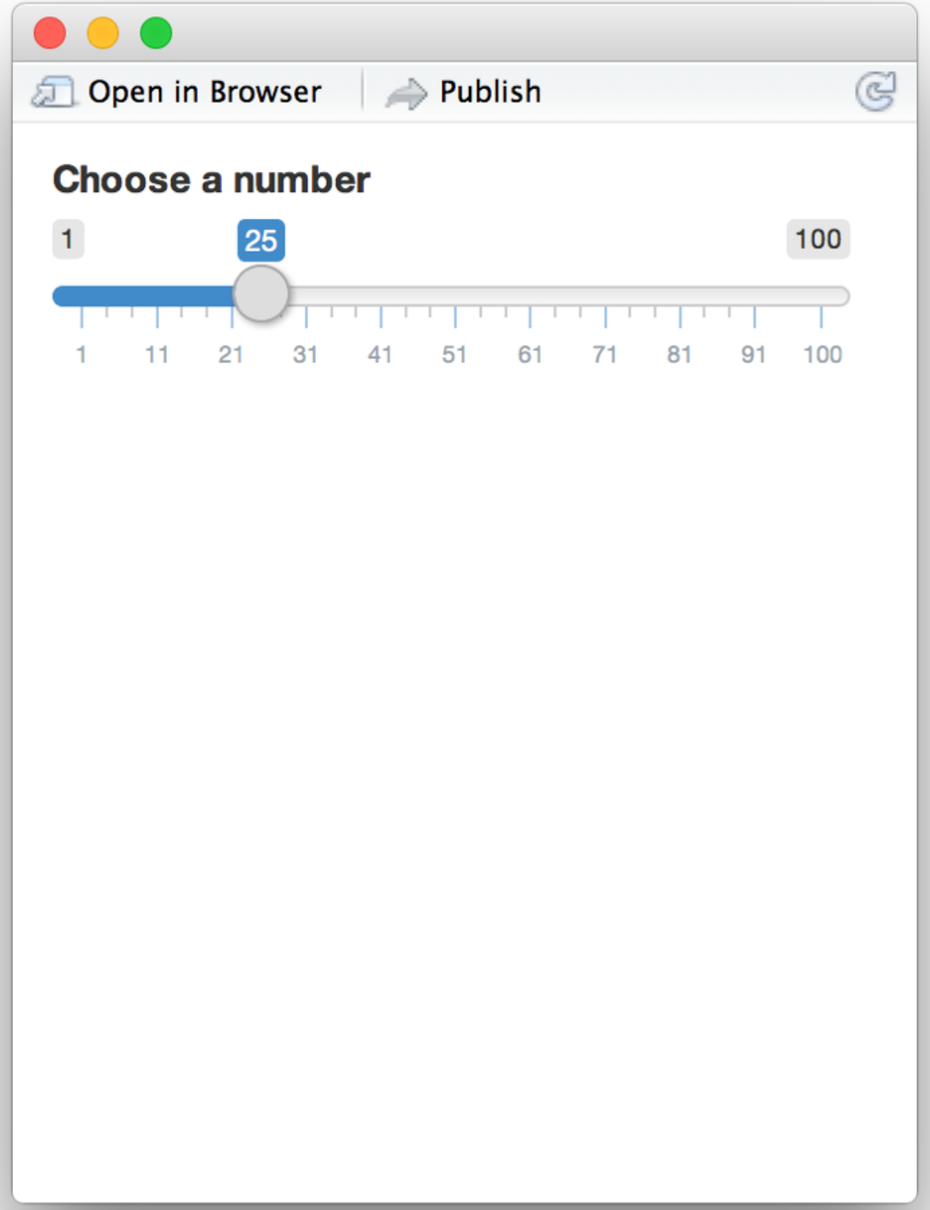
```
library(shiny)
ui <- fluidPage(




)

server <- function(input, output) {}

shinyApp(server = server, ui = ui)
```



Create an input with an input function.

UI

```
library(shiny)
ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100)
)

server <- function(input, output) {}

shinyApp(server = server, ui = ui)
```



UI

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

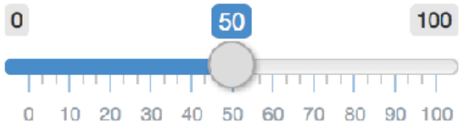
Choice 2

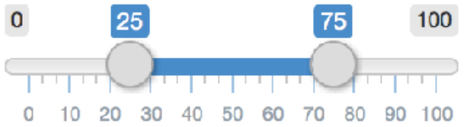
Choice 3

Select box

Choice 1

Sliders





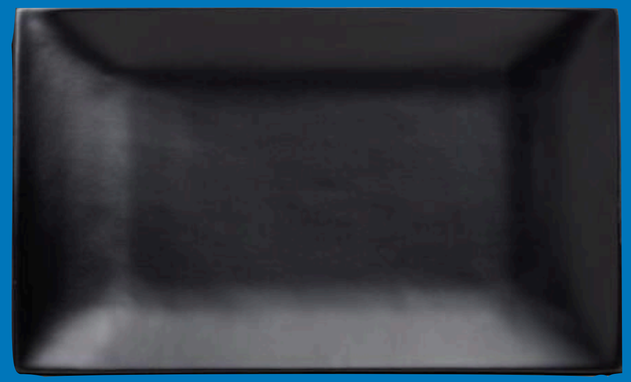
Text input

Enter text...

https://shiny.rstudio.com/tutorial/written-tutorial/lesson3/

USER INTERFACE (UI)

Outputs



Need to specify the type of output

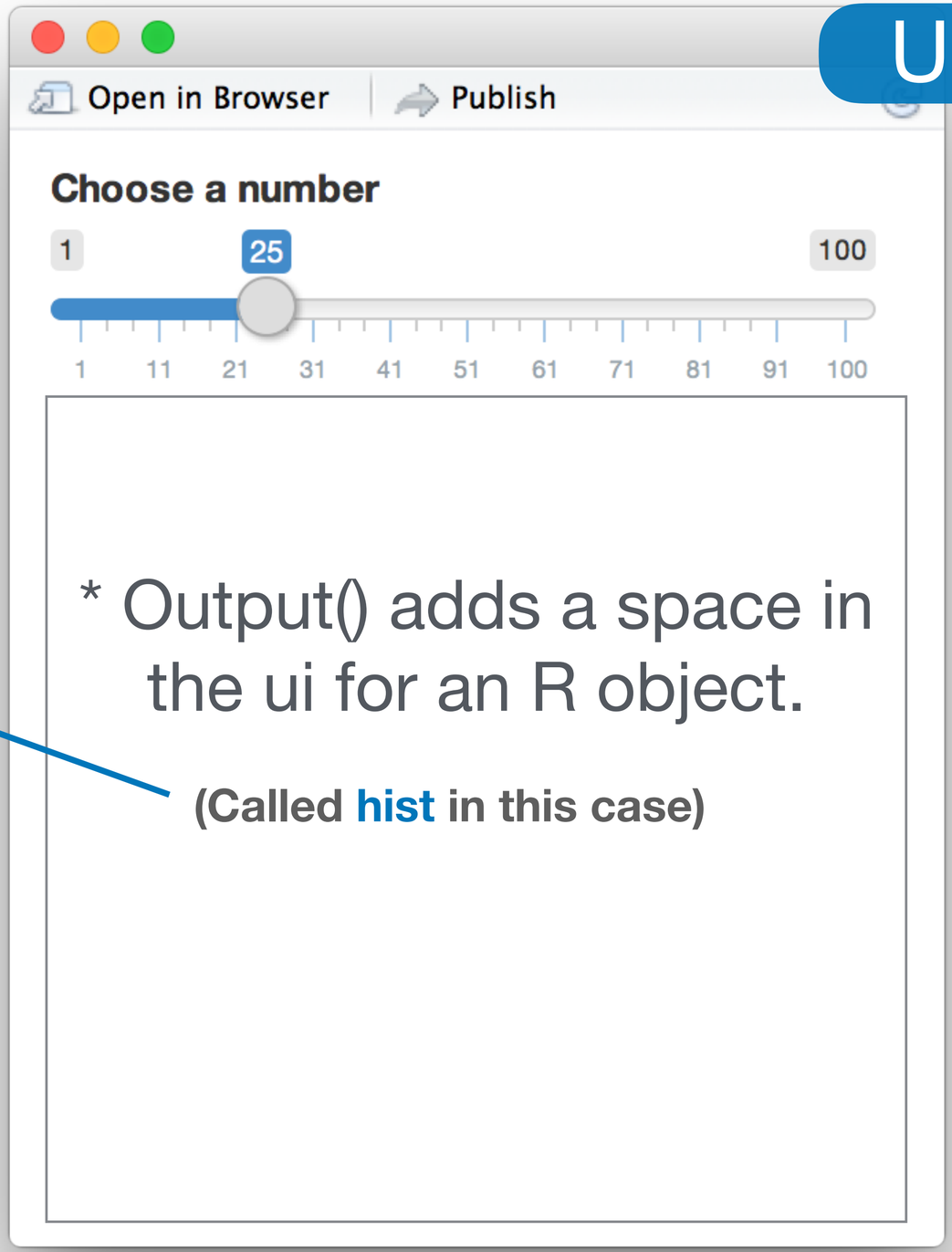
Function	Inserts
`dataTableOutput()`	an interactive table
`htmlOutput()`	raw HTML
`imageOutput()`	image
`plotOutput()`	plot
`tableOutput()`	table
`textOutput()`	text
`uiOutput()`	a Shiny UI element
`verbatimTextOutput()`	text
`plotlyOutput ()`	`plotlyOutput ()`

UI

```
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist")
)

server <- function(input, output) {}

shinyApp(ui = ui, server = server)
```

Comma between arguments

```
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist")
)

server <- function(input, output) {}

shinyApp(ui = ui, server = server)
```



Tell the

SERVER

How to assemble
inputs into the outputs

In 3 steps ...

Server

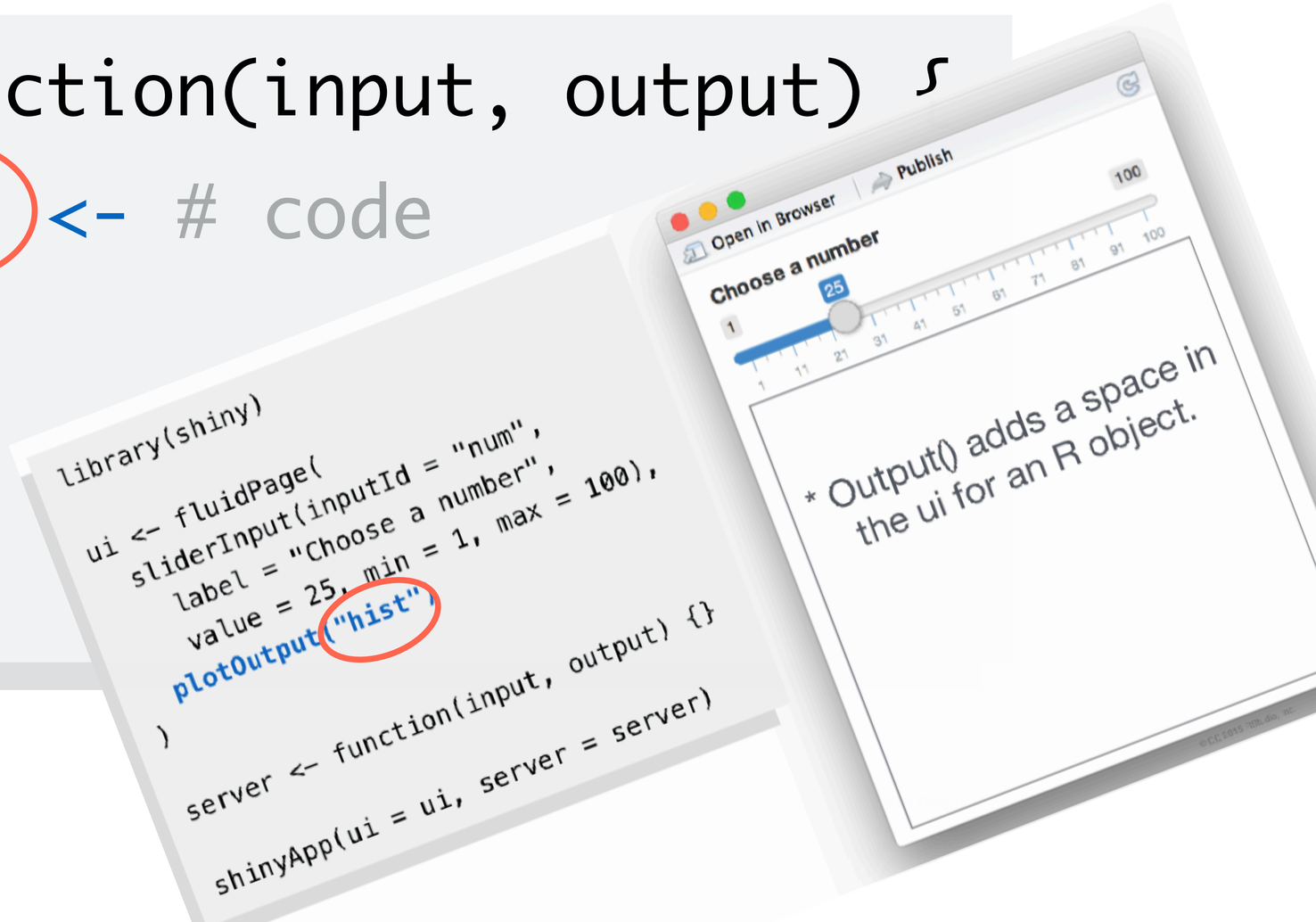
Server

1 Save objects to display to output$

```
server <- function(input, output) {
  output$hist <- # code



}
```

1 Save objects to display to output$

```
server <- function(input, output) {
  output$hist <- # code

}
```



Server

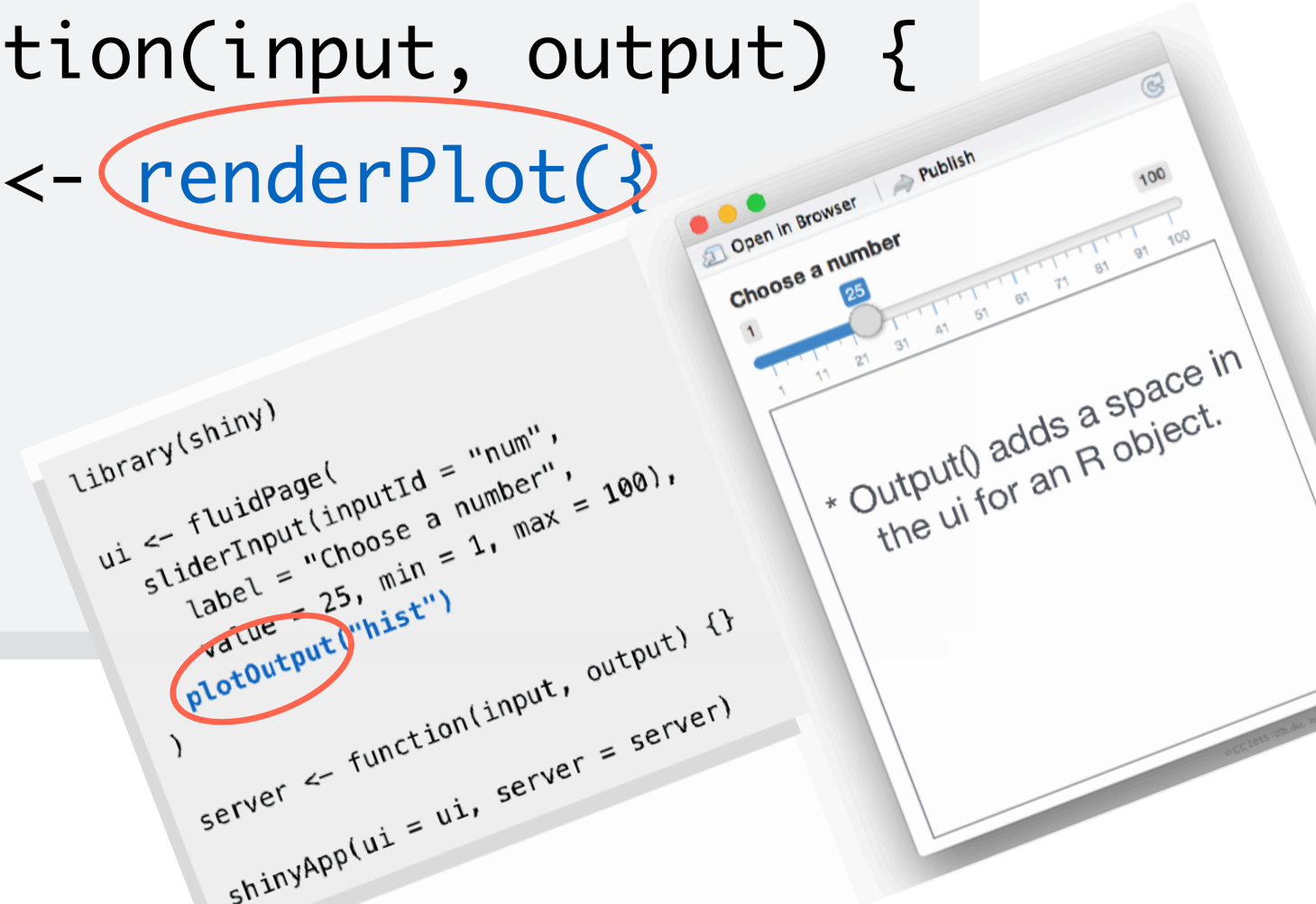
2 Build objects to display with **render*()**

```
server <- function(input, output) {
  output$hist <- renderPlot({


  })
}
```

Server

2 Build objects to display with **render*()**

```
server <- function(input, output) {
  output$hist <- renderPlot({


  })
}
```



Server

2 Use the **render*()** function that creates the type of output you wish to make.

function	creates
`renderDataTable()`	An interactive table (from a data frame, matrix, or other table-like structure)
`renderImage()`	An image (saved as a link to a source file)
`renderPlot()`	A plot
`renderPrint()`	A code block of printed output
`renderTable()`	A table (from a data frame, matrix, or other table-like structure)
`renderText()`	A character string
`renderUI()`	a Shiny UI element

Server

3 Build reactive output with linked inputs - “input$…”

```
server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}
```

```
library(shiny)
ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100)
)

server <- function(input, output) {}

shinyApp(server = server, ui = ui)
```

Server

3 Build reactive output with linked inputs - “input$…”

```
renderPlot({ hist(rnorm(input$num})
```

type of object to build

code block that builds the object

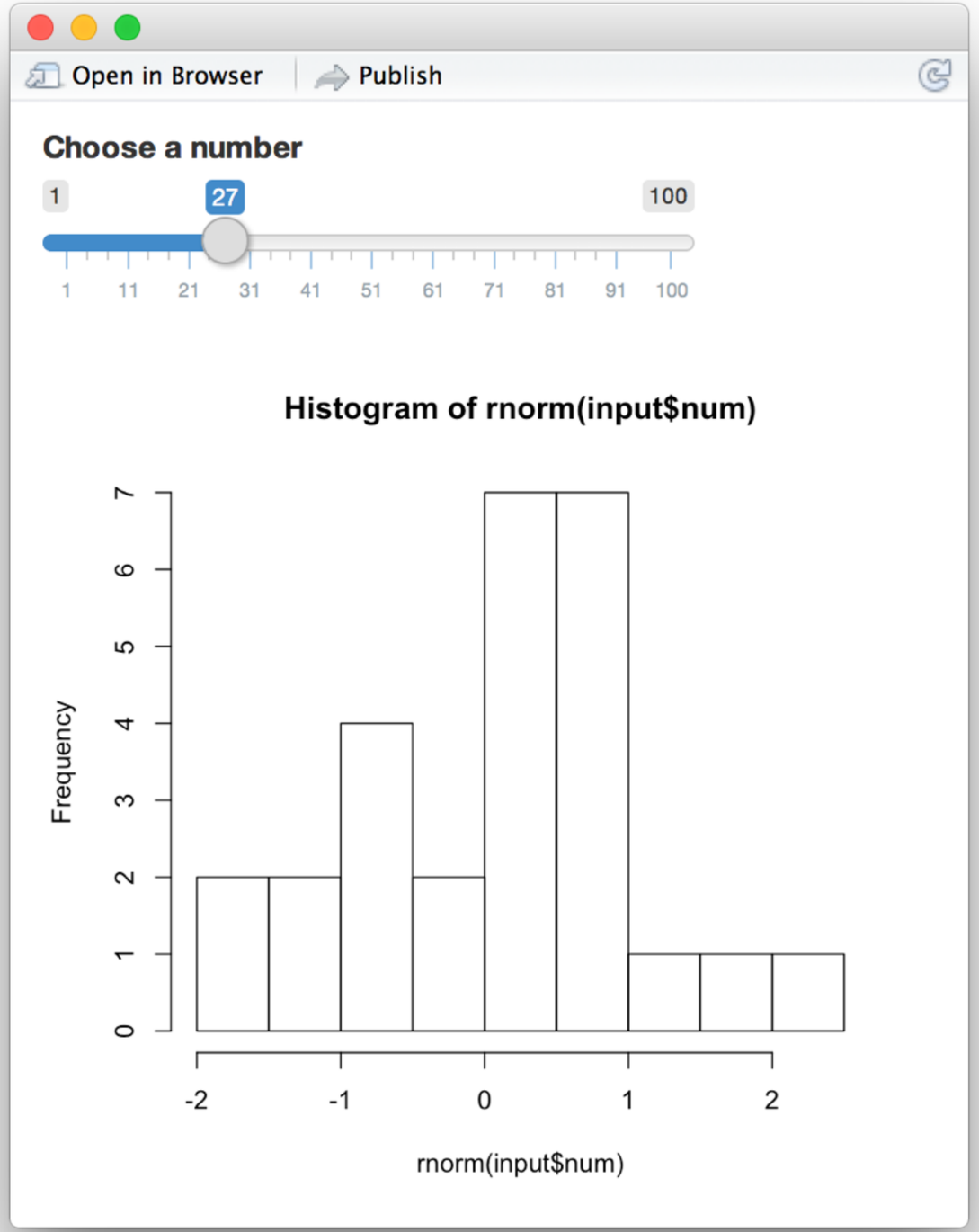
Server

Reactivity 101

Reactivity automatically occurs whenever you use an input value to render an output object

```
function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
})
```

```
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist")
)

server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}

shinyApp(ui = ui, server = server)
```



How to learn

- If you build it … you will learn
- RStudio Shiny website has guided learning modules (slow but thorough)
- Learn as you need … LMGTFY
- Take an example off the web, play around with it, change input types, elements etc …

Credit to RStudio for slide material

Ways to make your

app

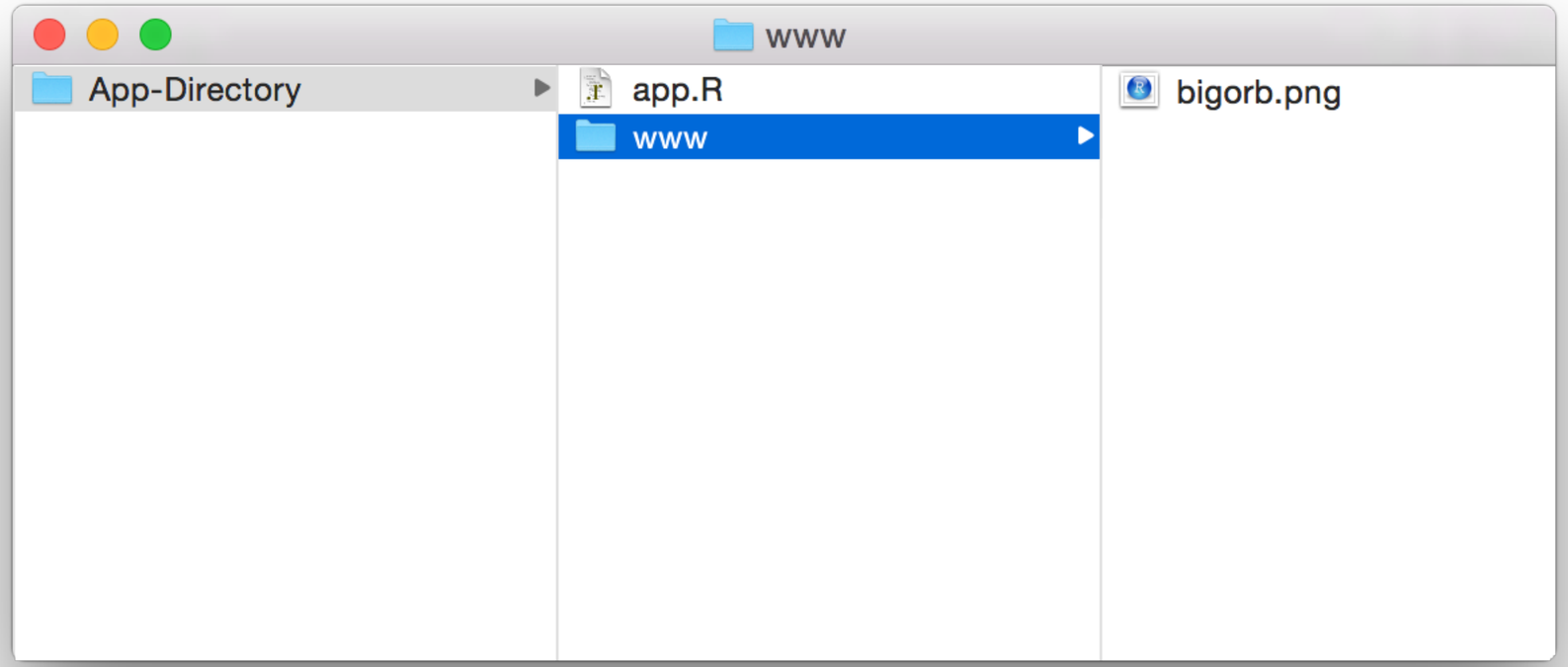
More … pretty

Add some text - or images?

Function	Creates
`a()`	A Hyperlink
`br()`	A line break
`code()`	Text formatted like computer code
`em()`	Italicized (emphasized) text
`h1(),h2(),h3(),h4(),h5(),h6()`	Headers (First level to sixth)
`hr()`	A horizontal rule (line)
`img()`	An image
`p()`	A new paragraph
`strong()`	Bold (strong) text

Adding Images

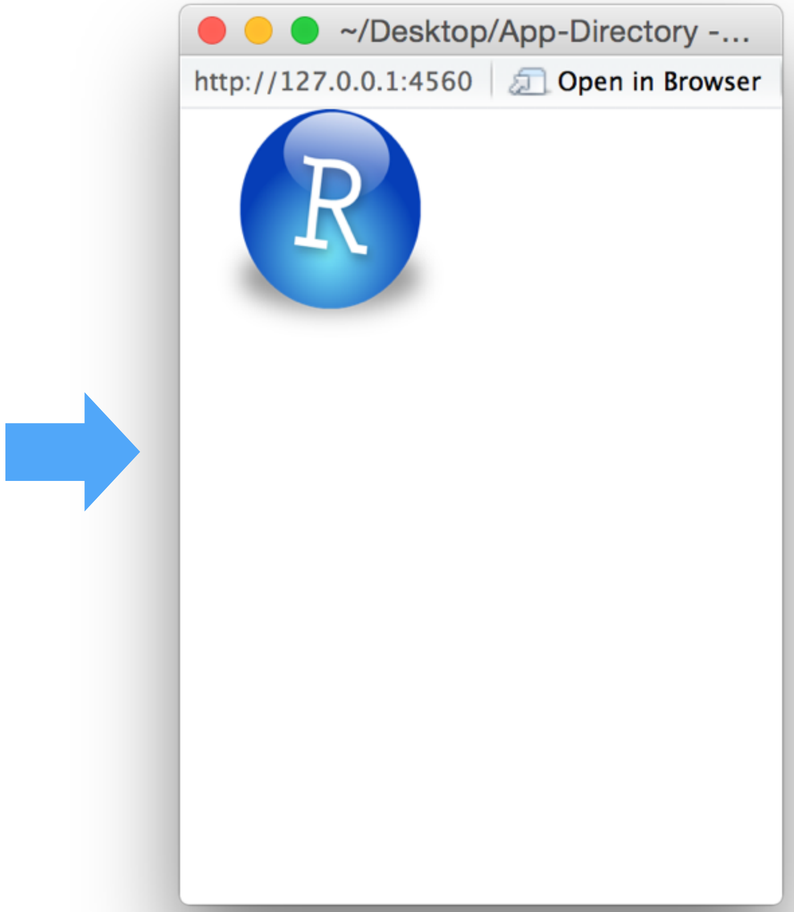
To add an image from a file, save the file in a subdirectory named **www**



img()

Add an img from a file in www

```
fluidPage(
        img(height = 100,
            width = 100,
            src = "bigorb.png")
)
```

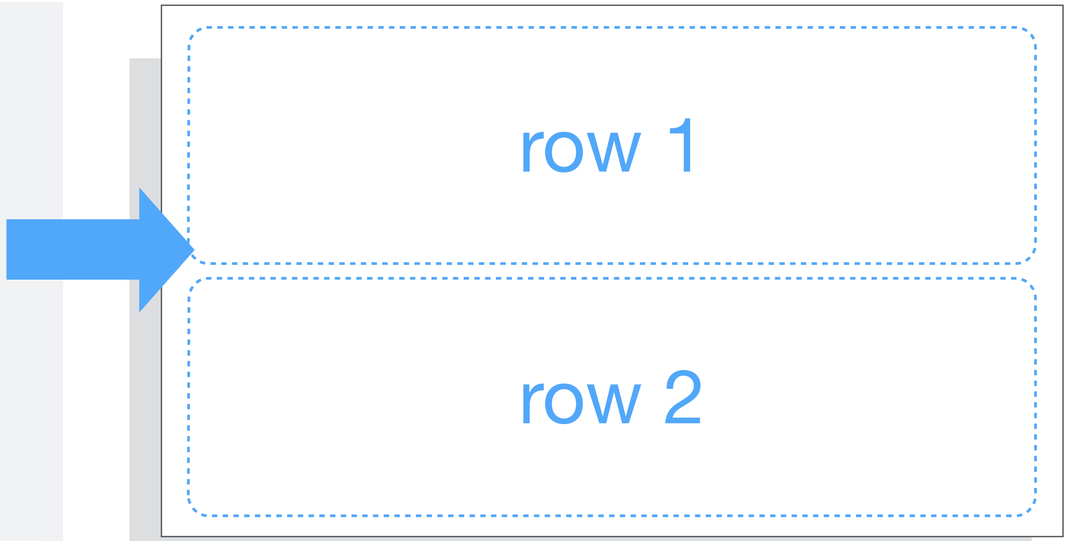


Change the layout

fluidRow()

`fluidRow()` adds rows to the grid. Each new row goes below the previous rows.

```
ui <- fluidPage(
  fluidRow(),
  fluidRow()
)
```

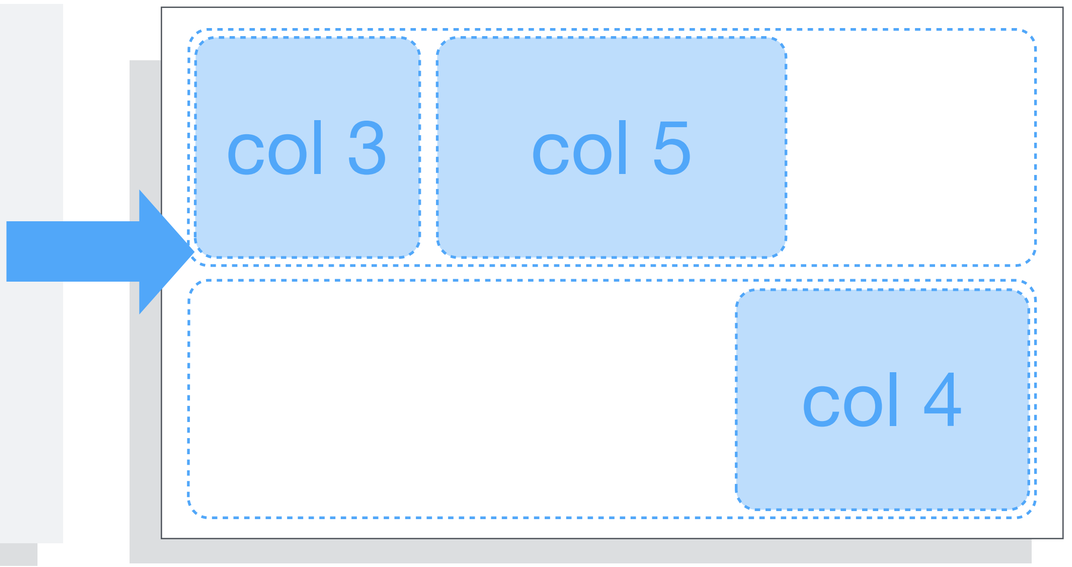


column()

`column()` adds columns within a row. Each new column goes to the left of the previous column.

Specify the **width** and **offset** of each column out of 12

```
ui <- fluidPage(
  fluidRow(
   column(3),
   column(5)),
  fluidRow(
   column(4, offset = 8)
)
```



To place an element in the grid, call it as an argument of a layout function

```
fluidPage(
  fluidRow(
    column(3),
    column(5, sliderInput(…))
  ),
  fluidRow(
    column(4, offset = 8,
      plotOutput("hist")
    )
  )
)
```

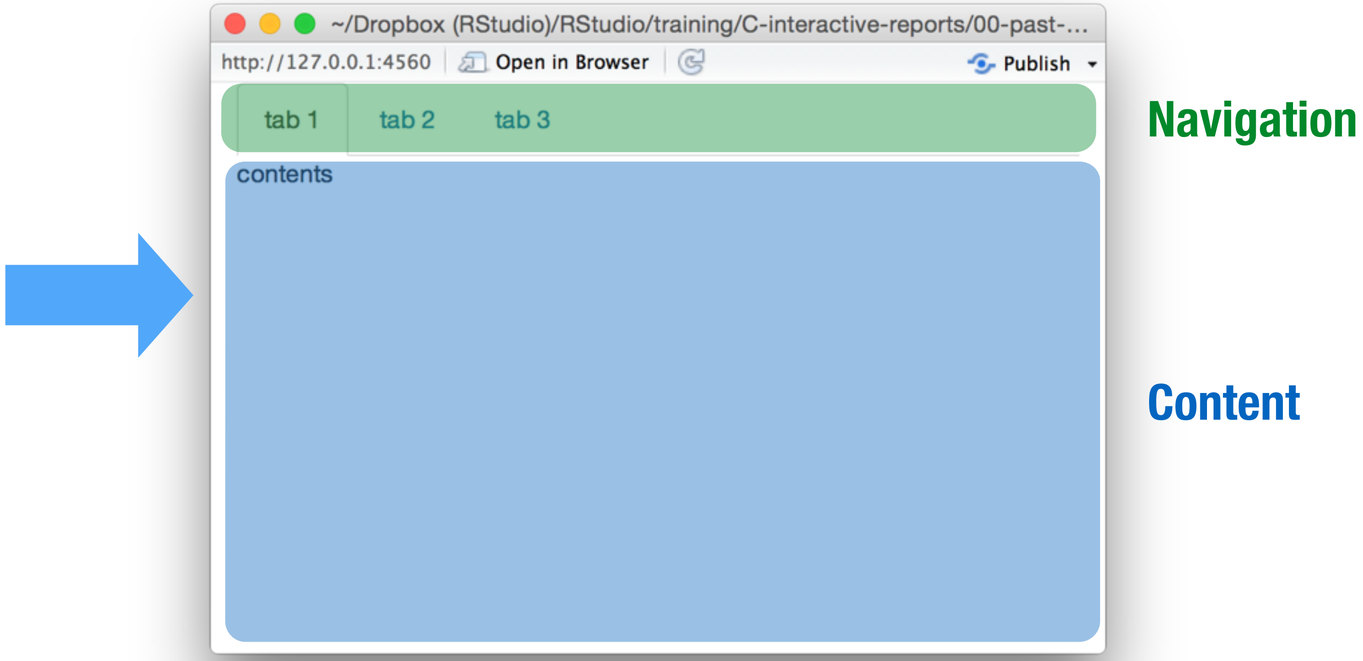
You could also add tabs if you have a whole lot of visuals

tabsetPanel()

`tabsetPanel()` combines tabs into a single *panel*.

Use *tabs* to navigate between tabs.

```
fluidPage(
  tabsetPanel(
   tabPanel("tab 1", "contents"),
   tabPanel("tab 2", "contents"),
   tabPanel("tab 3", "contents")
  )
)
```



absolutePanel()

Panel position set rigidly (absolutely), not fluidly

conditionalPanel()

A JavaScript expression determines whether panel is visible or not.

fixedPanel()

Panel is fixed to browser window and does not scroll with the page

headerPanel()

Panel for the app's title, used with pageWithSidebar()

inputPanel()

Panel with grey background, suitable for grouping inputs

mainPanel()

Panel for displaying output, used with pageWithSidebar()

navlistPanel()

Panel for displaying multiple stacked tabPanels(). Uses sidebar navigation

sidebarPanel()

Panel for displaying a sidebar of inputs, used with pageWithSidebar()

tabPanel()

Stackable panel. Used with navlistPanel() and tabsetPanel()

tabsetPanel()

Panel for displaying multiple stacked tabPanels(). Uses tab navigation

titlePanel()

Panel for the app's title, used with pageWithSidebar()

wellPanel()

Panel with grey background.

Share

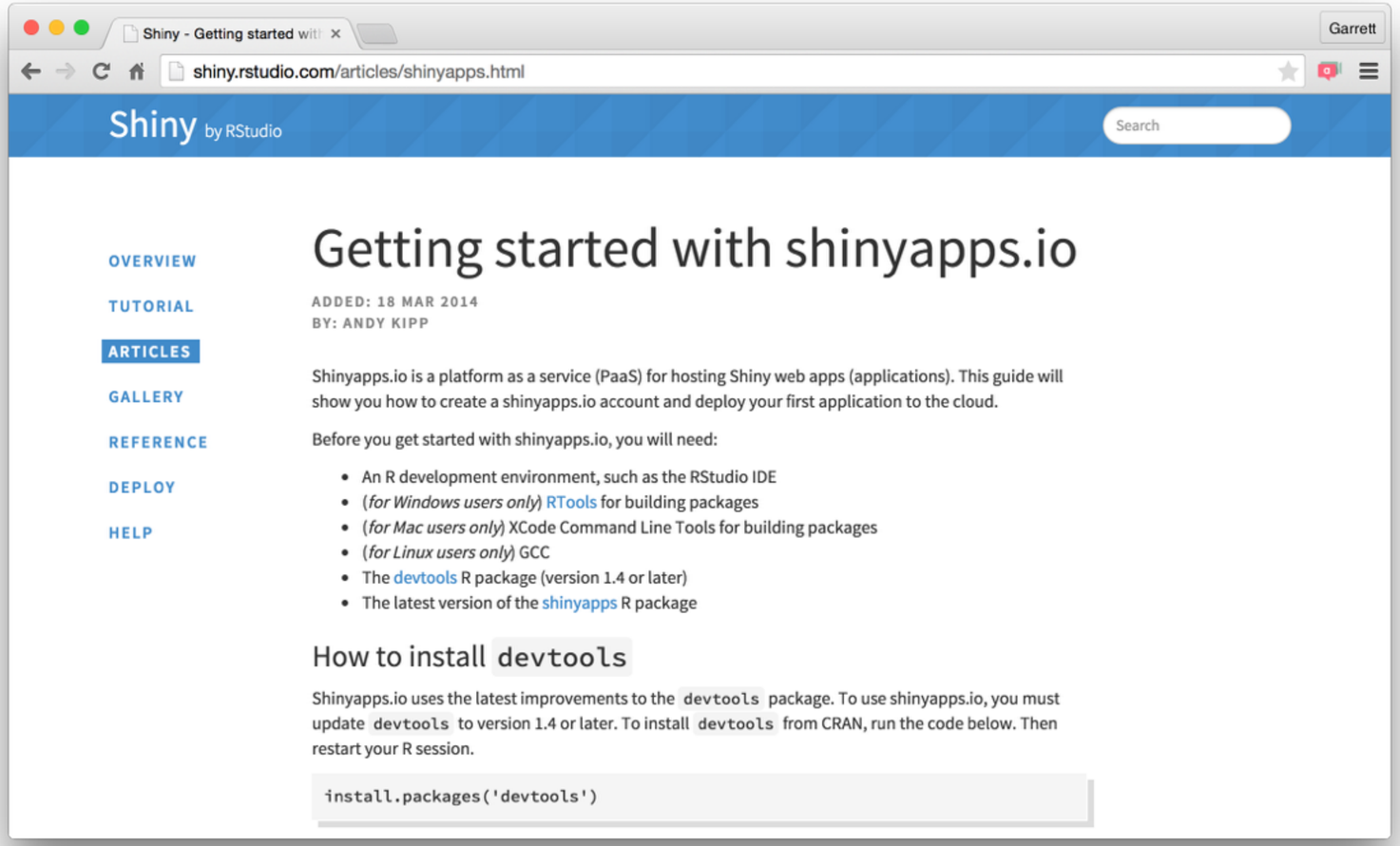
your app

Use

shinyapps.io

Getting started guide

shiny.rstudio.com/articles/shinyapps.html



Credit to RStudio for slide material