

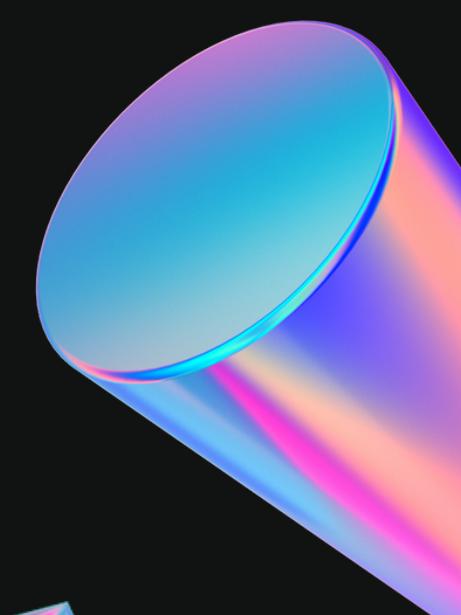
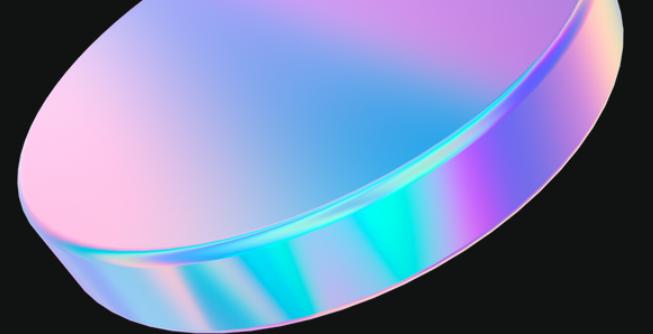
BRIGHT R SHINY OBJECTS

Achievable, shareable and understandable representations of data can help us communicate & collaborate more effectively.
Shiny apps give us an easy way to do this.

UQ Advanced R Course - 2021

Dr Bill Venables - Data 61 & CSIRO

Rhetta Chappell - RIDL, Griffith University





Rhetta Chappell

Data Scientist – RIDL at Griffith University

BACKGROUND

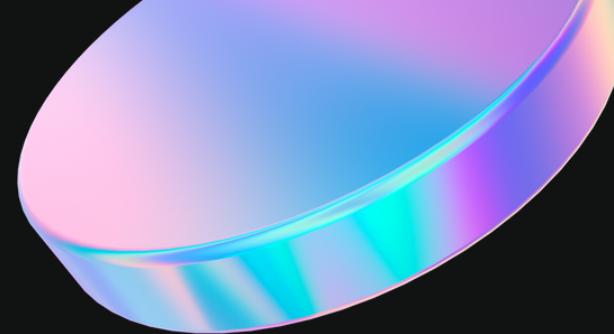
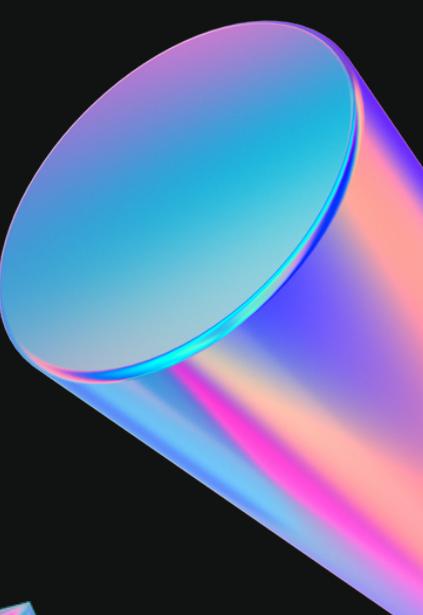
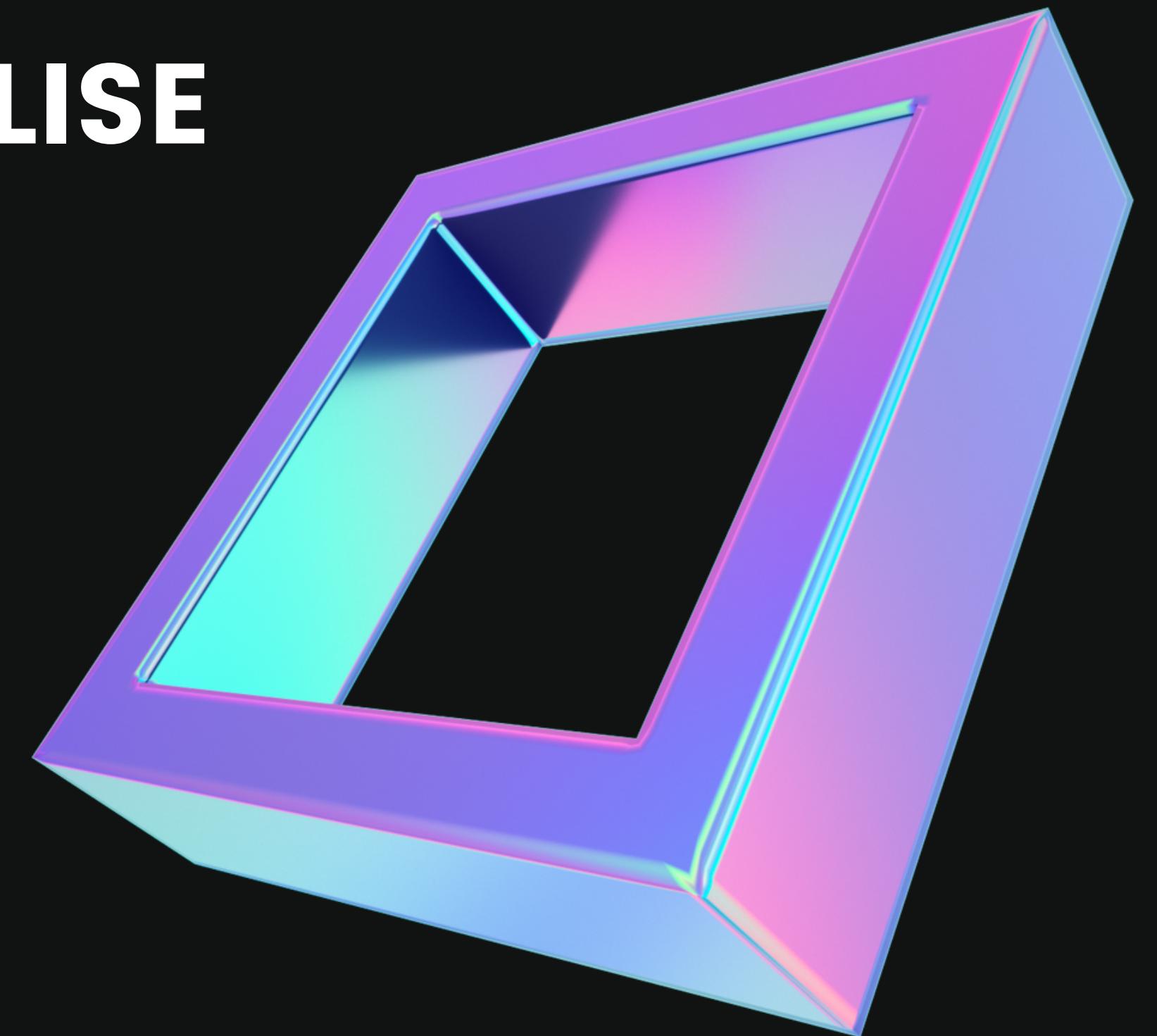
- art, design
- anthropology
- research
- data science

PROFESSIONAL INTERESTS

- data visualisation & storytelling
- democratising data – data collaboratives and coops
- actionable insights for social good
- behavioural economics & psychology

 @rhetta_chappell

**HOW DO YOU VISUALISE
YOUR DATA AND
COMMUNICATE
YOUR REASEARCH
FINDINGS?**



Top 7 reasons why I build Shiny apps:

1. Research translation (PhD students, sharing project outcomes)
2. Generate & share accessible & actionable insights (Community Orgs, Local Councils)
3. Stakeholder engagement
4. Policy advocacy
5. Civic engagement
6. Wireframing / prototyping more complex applications
7. Data exploration & analysis

The ShinyR the better...

Shiny app examples:

PRACTICAL:

<https://connect.thinkr.fr/hexmake/>

DATA SCIENCE & OPEN DATA:

https://nz-stefan.shinyapps.io/blog_explorer/

- network graph +30,000 blog articles on R

LIFE SCIENCES/PUBLIC INTEREST:

[https://vac-lshtm.shinyapps.io/ncov_tracker/?
_ga=2.90997338.1162613968.1610319694-609160277.1538646407](https://vac-lshtm.shinyapps.io/ncov_tracker/?_ga=2.90997338.1162613968.1610319694-609160277.1538646407)

INDUSTRY:

[https://phillyo.shinyapps.io/intelligentsia/?
_ga=2.168124897.1162613968.1610319694-609160277.1538646407](https://phillyo.shinyapps.io/intelligentsia/?_ga=2.168124897.1162613968.1610319694-609160277.1538646407)

ENTERPRISE:

<https://demo.appsilon.ai/apps/shiny-enterprise-demo/>

POLITICS:

<https://regionalinnovationdatalab.shinyapps.io/qldvotes/>

why shiny?

- You're already familiar with R
- No web development skills are required
 - Easy to understand interactivity through reactivity
- Quickly prototype and let users interact with your data and your analysis:
 - easily embed into .Rmd, blogdown and websites
 - highly customisable - basic to beautiful ui
 - free to host and share - shinyapps.io, RStudio Connect
 - lots of great packages
 - (shinydashboardPlus, shiny.demantic, shinydashboard, plumbR, leaflet, plotly, DT, data.table)
 - great for stakeholder engagement & collaboration

How does Shiny work?

- R Shiny is an R package which translates R code into html code to be understood by your web browser
 - R code = more concise and intelligible for R users
 - `install.packages("shiny")`

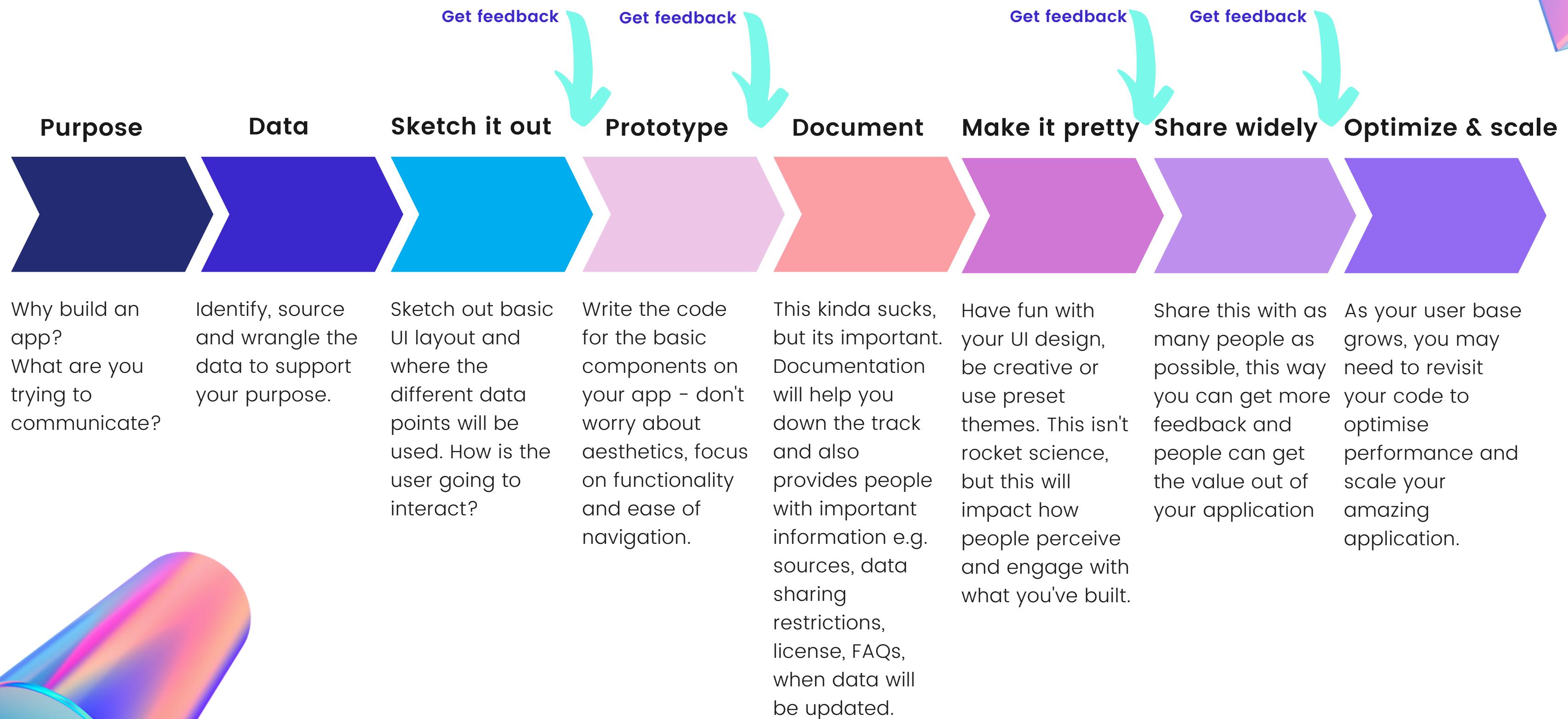
```
library(shiny)
sliderInput("obs", "Number of observations:",
  min = 0, max = 1000, value = 500
)
```

```
<div class="form-group shiny-input-container">
  <label class="control-label" for="obs">Number of observations<br>
  <input class="js-range-slider" id="obs" data-min="0" data-max="100" data-grid="true" data-grid-num="10" data-grid-snap="false" data-prettify-enabled="true" data-keyboard="true" data-keyboard="true" type="text">
</div>
```



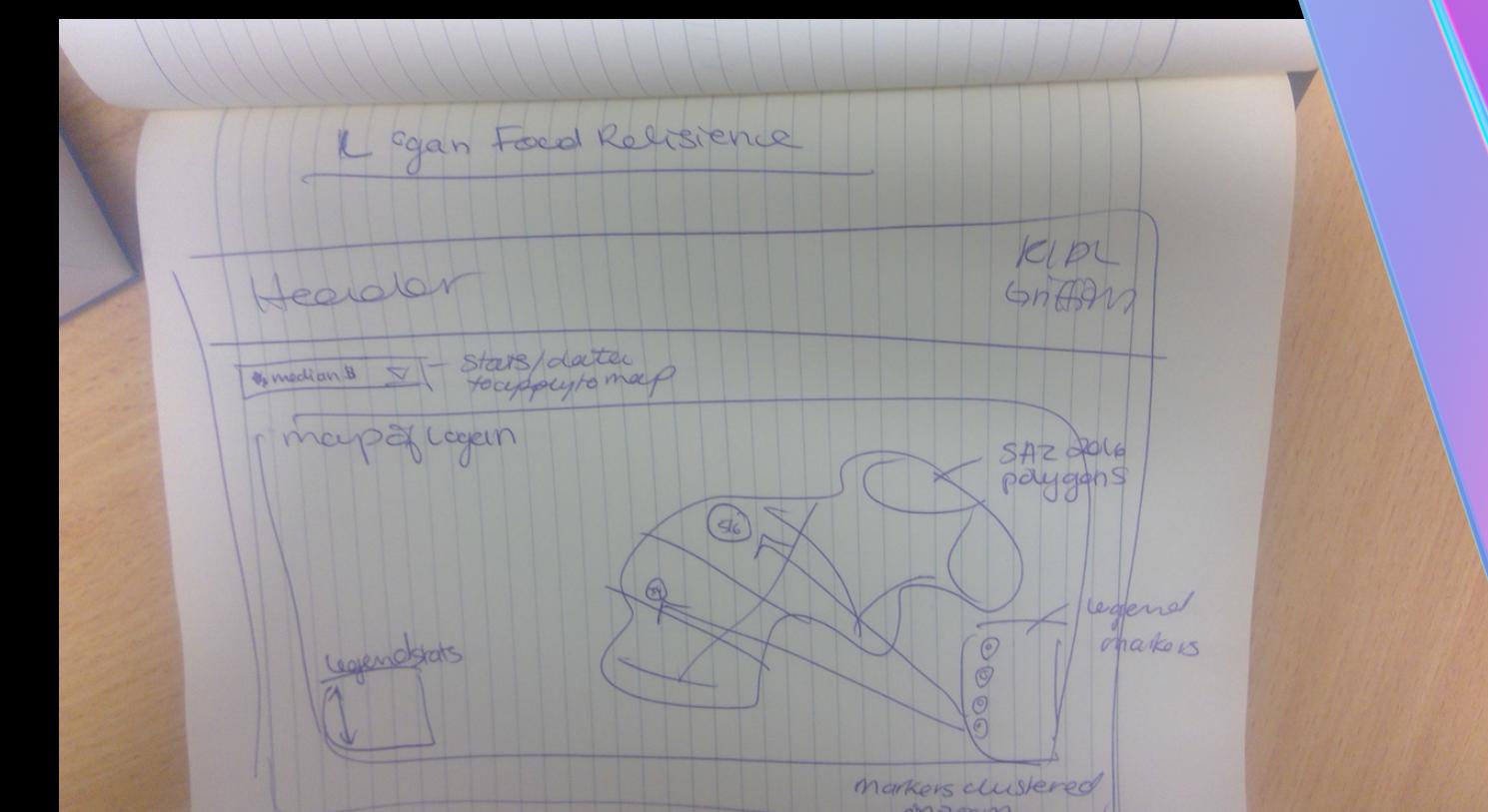
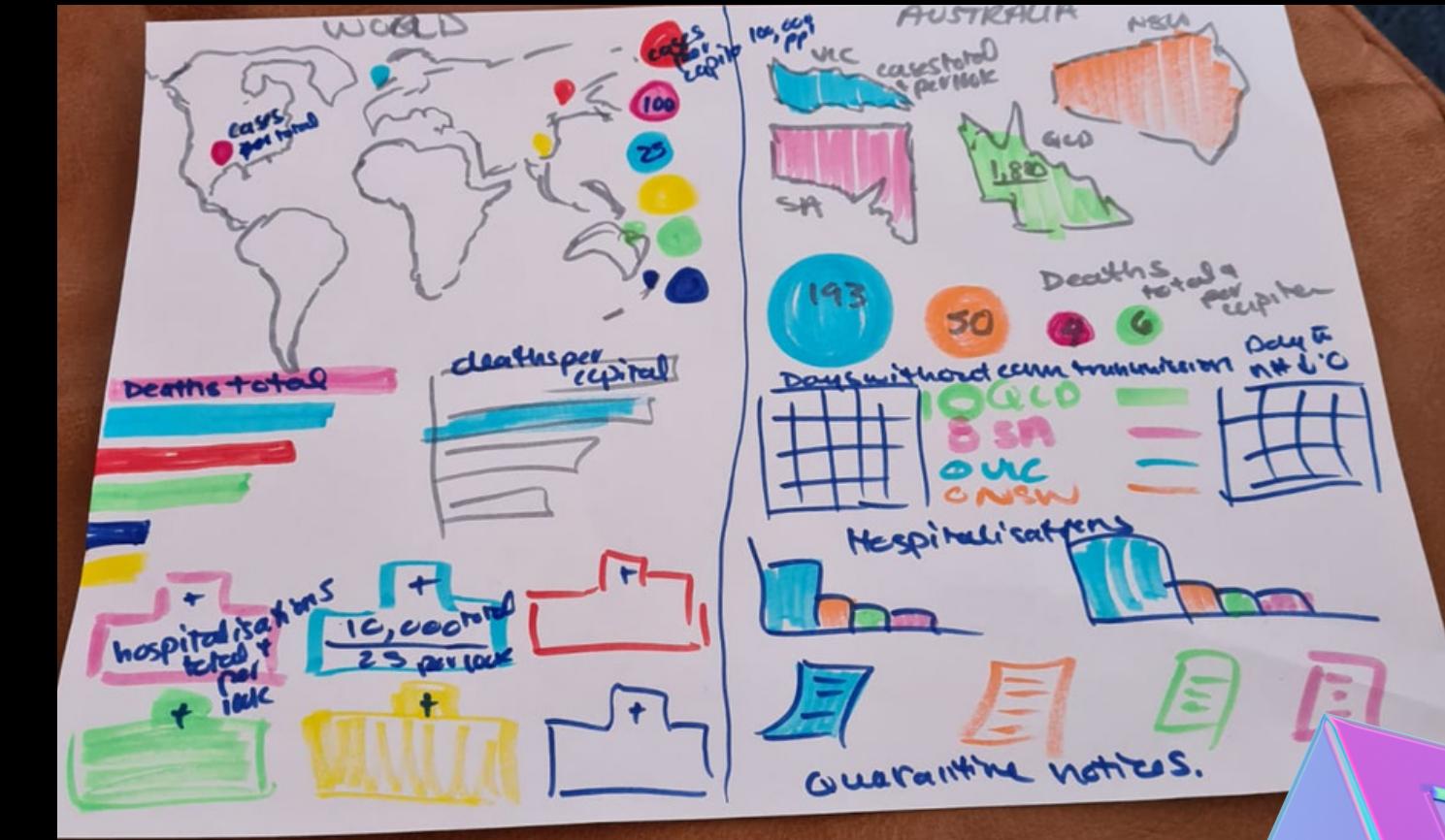
MY SHINY WORK FLOW

this is what works for me, you might be different



TIP #1 - SKETCH IT OUT

- Sketch out app ui and basic reactive components (slider, dropdowns, upload)
 - doesn't need to be pretty (look at mine)
 - Walk client/ stakeholder through your sketch
 - ask for feedback, rework your sketch and then start coding



● **UI**

- front end interface
- simpler (in the beginning)
 - all users get same version of HTML
- defines how your app looks

● **SERVER**

- back end logic
- more complex (in the beginning)
 - each user gets own version of app
- defines how your app works

2 key components of every Shiny app:

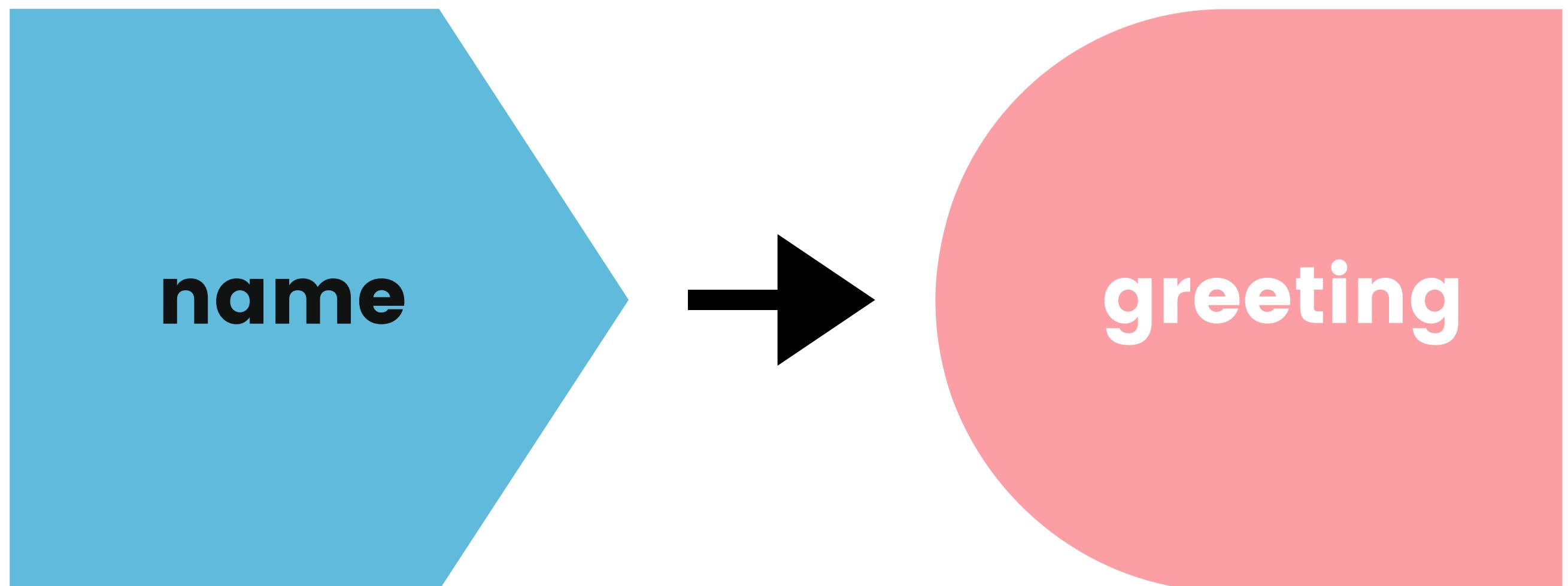
```
library(shiny)
#1
ui <- fluidPage(
)

#2
server <- function(input, output, session) {

}
shinyApp(ui, server)
```

Reactivity

Shiny uses a reactive programming model.



The **reactive graph** contains one symbol for every **input** and **output**, and we connect an **input** to an **output** whenever the **output** accesses the **input**.

For more on reactivity: <https://shiny.rstudio.com/articles/reactivity-overview.html>

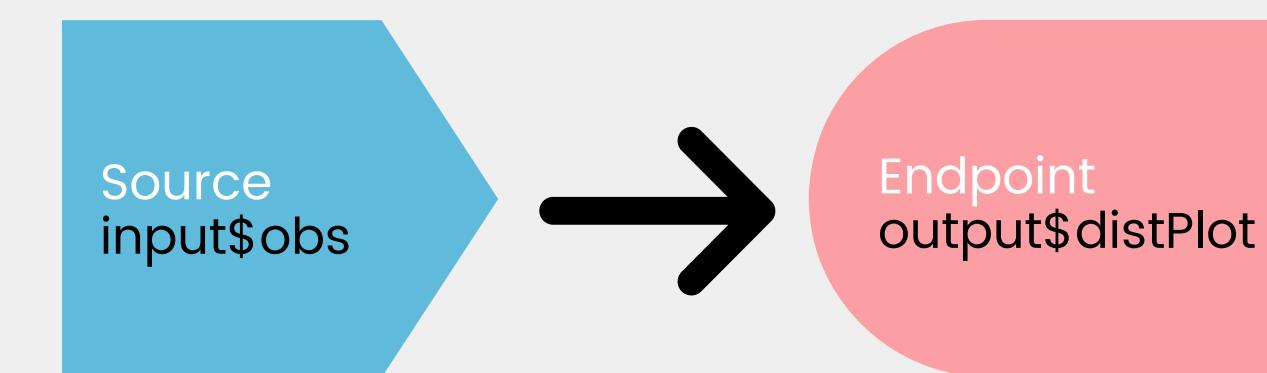
Reactive programming model

- 3 types of objects in reactive programming



- simple app structure = source and endpoint

```
server <- function(input, output) {  
  
  output$distPlot <- renderPlot({  
  
    #generate an r norm distribution plot  
    hist(rnorm(input$obs))  
  })  
}
```



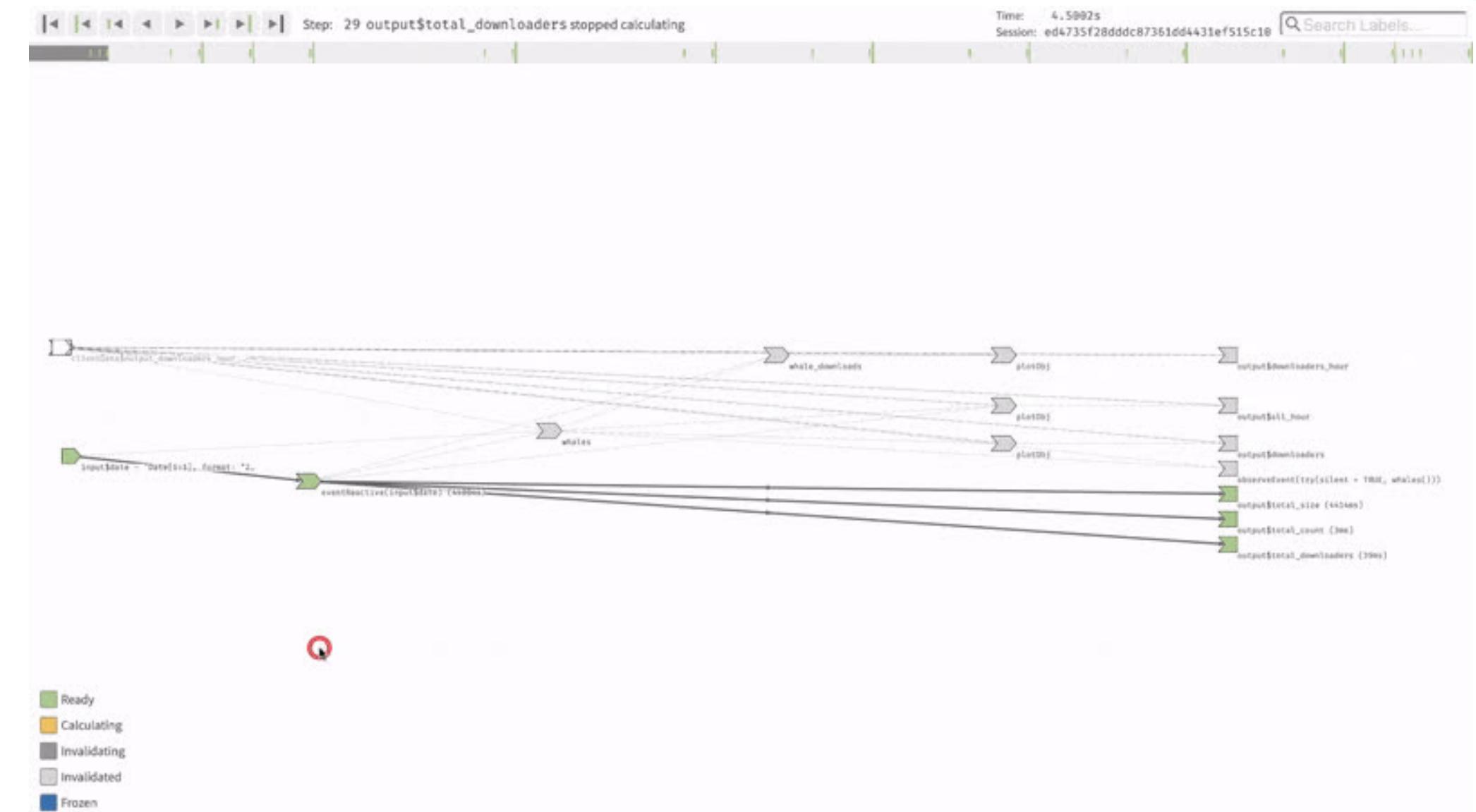
https://gallery.shinyapps.io/01_hello/?_ga=2.120228746.360158209.1611709044-609160277.1538646407

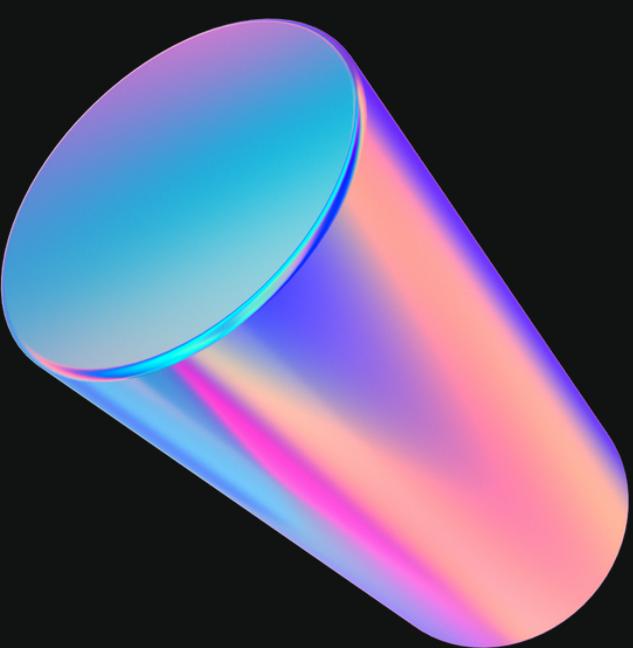
For more complex apps:

<https://cloud.r-project.org/package=reactlog>

Display the reactivity dependency graph of your Shiny applications:

- `install.packages("reactlog")`
- don't want a "reactive spaghetti mess"
 - modularize your code - best practice, but more advanced





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

input parameter

- The input argument is a list-like object that contains all the input data sent from the browser, named according to the input ID.



```
ui <- fluidPage(  
  numericInput( id = "count",  
    label = "Number of values",  
    value = 100) #end numeric input  
)#end fluidPage
```

- you can access the value of that input with `input$count`
 - It will initially contain the value 100, and it will be automatically updated as the user changes the value in the browser.
- Unlike a typical list, input objects are read-only.
 - To read from an input, you must be in a reactive context - more on this later

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

output & input parameters

- Note that the ID is quoted in the UI, but not in the server.

```
ui <- fluidPage(  
  numericInput(id = "count", ↗  
    label = "Number of values",  
    value = 10), # end numeric input  
  
  textOutput(id = "message") # end text output  
  
) #end fluidPage
```

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

```
  ↗  
  output$message <- renderText({  
    paste0("You selected ", input$count, "!") #end RenderText
```

```
}
```



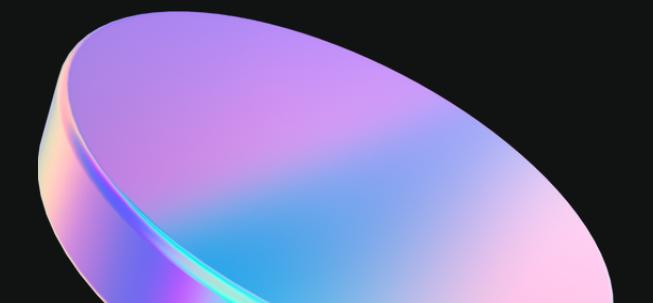
```
server <- function(input, output, session) { }
```

output parameter

- also a list-like object named according to the output ID
- used for sending output instead of receiving input
- always use the `output` object together with a `render` function

```
ui <- fluidPage(  
  textOutput(id = "message")  
)#end fluidPage
```

```
server <- function(input, output, session) {  
  output$message <- renderText("Advanced R 2021!")#end fluidPage  
}
```



TIP #2 – SHINY IS LAZY

- check that your UI and server functions are using the same identifiers
 - error messages can be cryptic or more often non-existent

```
ui <- fluidPage(  
  textOutput( id = "greeting")  
 )#end fluidPage
```

```
server <- function(input, output, session) {  
  output$greetnig <- renderText({ paste0("Hello ", input$name, "!")  
    }) #end RenderText  
}
```

IMPERITIVE & DECLARATIVE PROGRAMMING

COMMANDS

IMPERITIVE – R SCRIPTS

In imperative programming, you issue a specific command and it's carried out immediately.

"Make me a Greek salad."

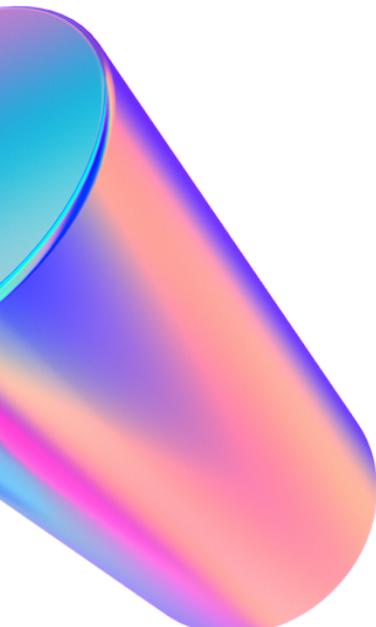
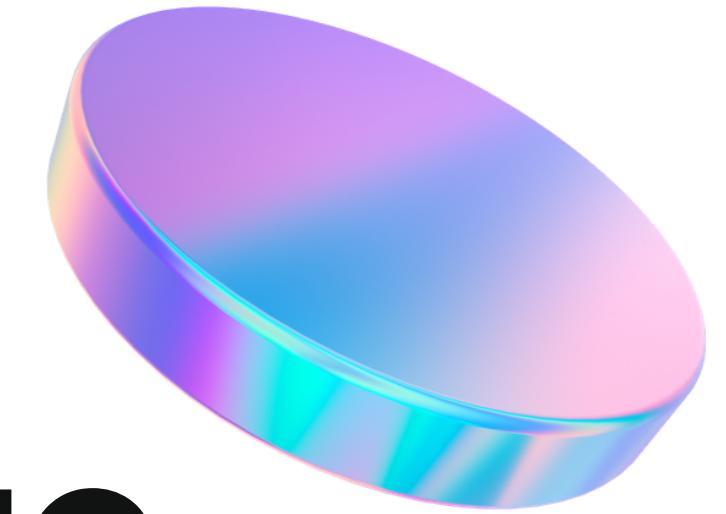
RECIPES

DECLARATIVE – R SHINY

In declarative programming, you express higher-level goals or describe important constraints, and rely on someone else to decide how and/or when to translate that into action.

- can sometimes be tricky to frame what you want this way

"Ensure there is a Greek salad in the fridge whenever I look inside it".



"RECIPES INSTEAD OF COMMANDS"

Copy, Paste & Run:

```
ui <- fluidPage(  
 textInput(id = "name",  
            label = "What's your name?"), #end textInput  
  → textOutput("greeting")  
 ) # end fluidPage
```

```
server <- function(input, output, session) {  
  → output$greeting <- renderText({  
    paste0("Hello ", input$name, "!") #end RenderText
```

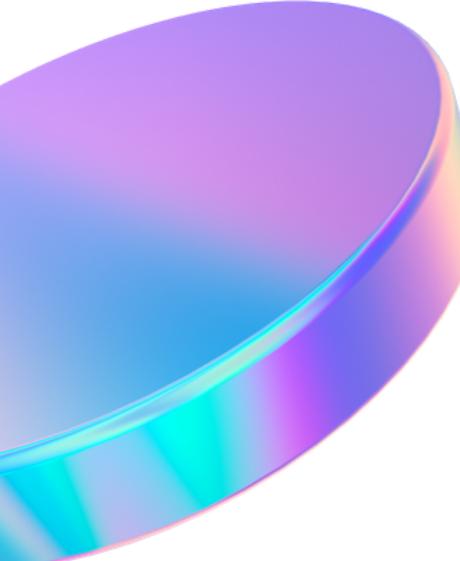


The image displays two side-by-side screenshots of a Shiny application. Both screenshots show a 'What's your name?' label above a text input field containing either 'J' or 'Joe'. Below each input field is an output text area. In the left screenshot, the output is 'Hello J!', and in the right screenshot, the output is 'Hello Joe!'. This visual comparison illustrates how the application generates personalized greetings based on the user's input.

Order of execution

- Not top to bottom
- Only executes as needed
 - user interaction
 - unique sessions





TIP #2 - BE KIND TO YOUR FUTURE SELF

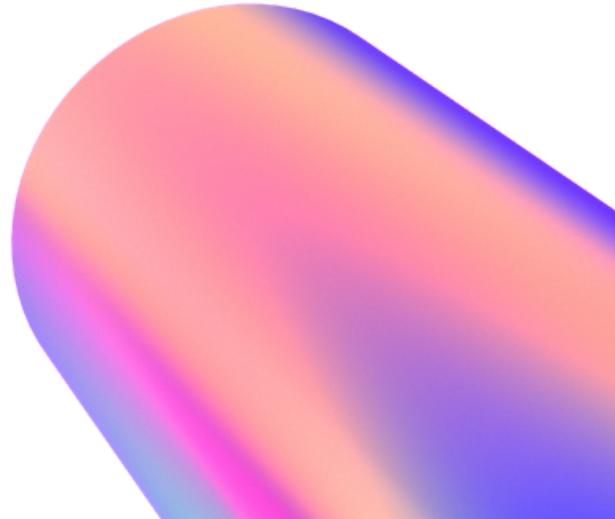
COMMENT LIKE CRAZY, AND USE A CONSISTENT NAMING CONVENTION

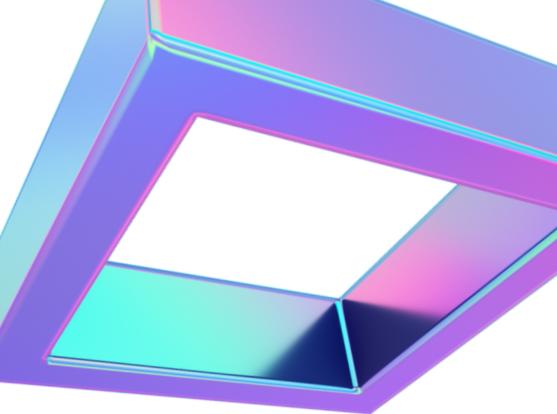
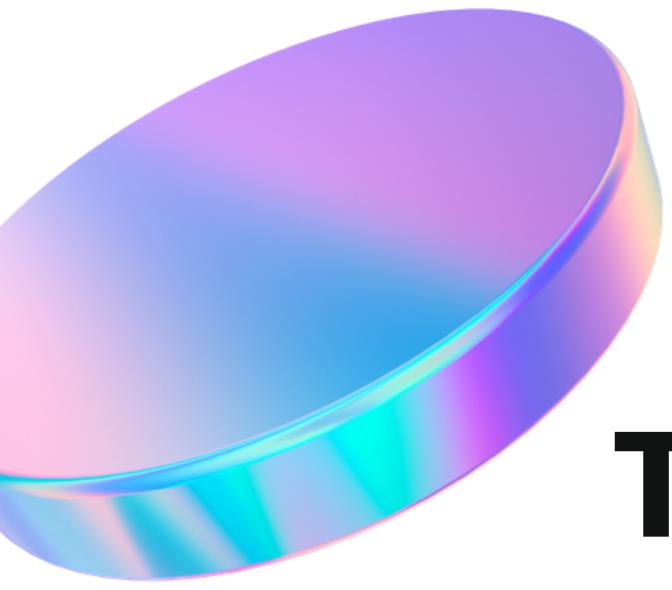
```
library(shiny)
library(shinydashboardPlus)

ui <- fluidPage(
  #drop down menu - user selects child safety service centre (CSSC)
  shinydashboardPlus::pickerInput( inputId = 'cssc_input1',
    label = 'Select a CSSC to update the graphs & map:',
    width = "100%",
    choices = unique(df_cssc$CSSC), # end choices
    options = list(`style` = 'btn-primary') #end options
  ) #end pickerInput
) #end fluidPage

server <- function(input, output, session) {

} #end server
```

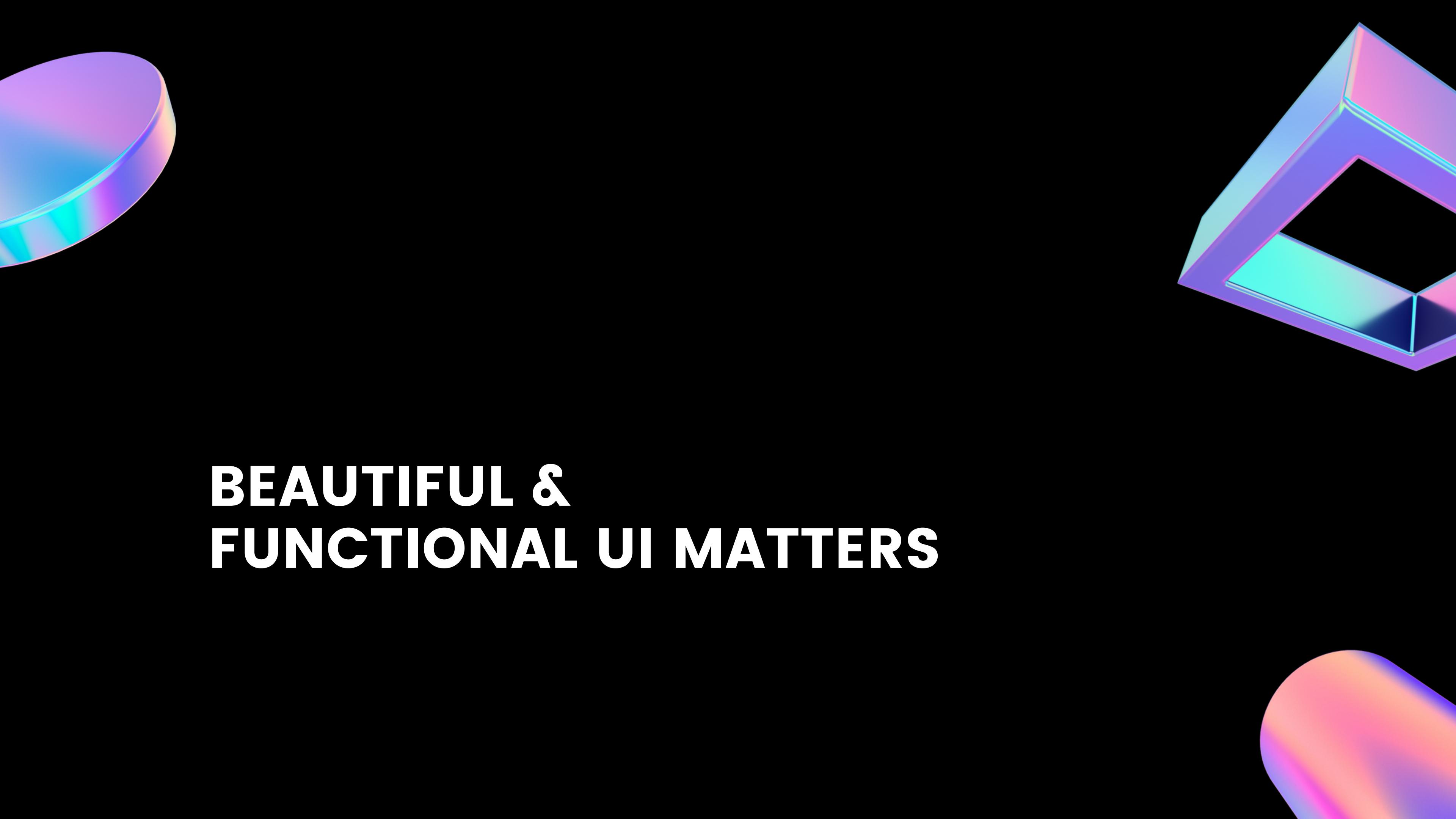




TIP #2 - EXAMPLE

```
481   column(6,  
482     withSpinner(  
483       leafletOutput("map4",height = "340px", width="100%")  
484     )#end spinner  
485   )#end column  
486 )# end fluidrow  
487 ) # end FluidPage  
488
```

```
column(6,  
  withSpinner(  
    leafletOutput("map4",height = "340px", width="100%")  
  )  
)
```

The background features abstract, semi-transparent 3D geometric shapes in shades of purple, blue, and pink, floating against a black background.

**BEAUTIFUL &
FUNCTIONAL UI MATTERS**

“

Design is not just what it looks like and
feels like. Design is how it works.

— Steve Jobs

UI PRINCIPLES

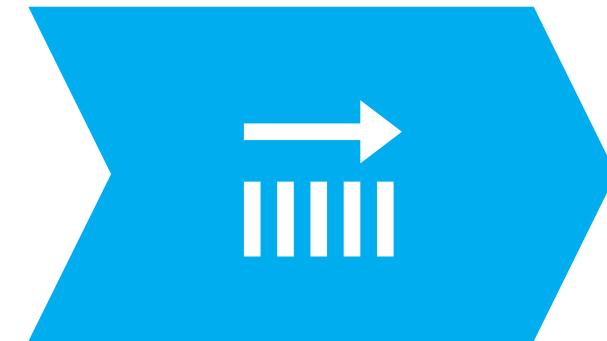
consider your audience's accessibility needs

KISS



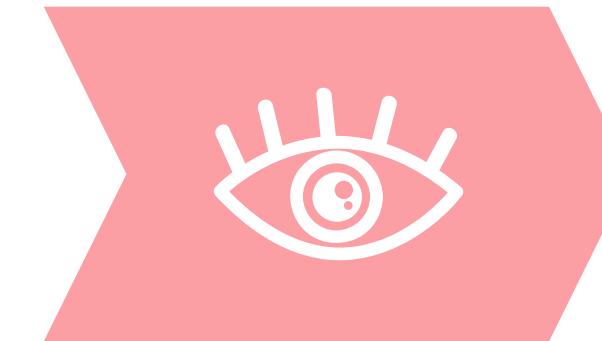
Keep it simple, stupid

**Be
consistent**



Colour, buttons,
widgets, flows, layout,
arrangement

**Readability &
scanability**



Highlight key
elements (info,
navigation) using
layout, contrast, light,
space, font and size

**Reduce
frustration**



Provide users with
feedback, don't let
them feel lost (status
bar, loaders,
messages, updates)

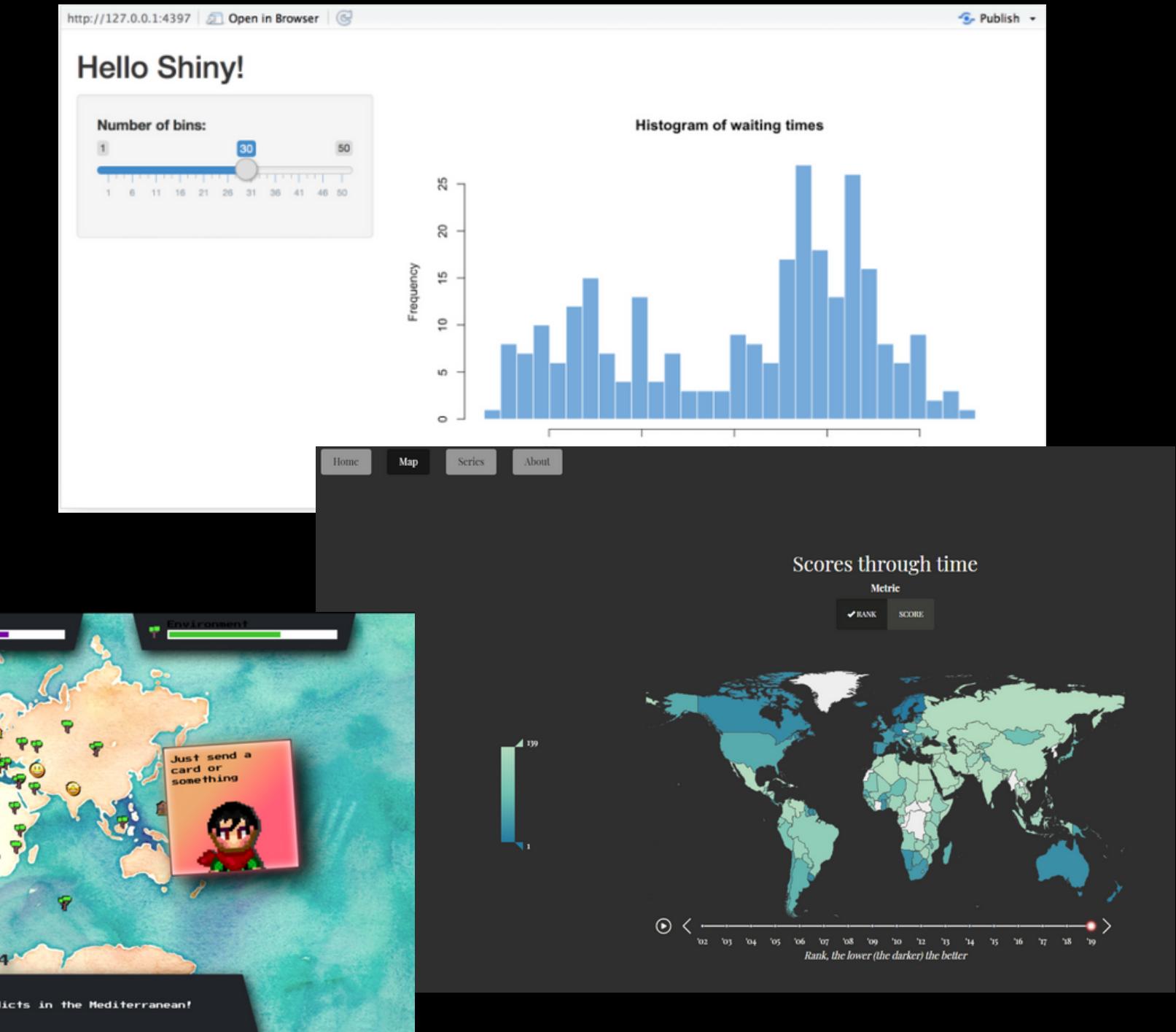
Iterate



Be ready and willing
to adjust your UI to
suit your users, SEEK
OUT feedback, stay
up to date on best
practices and new
packages

Don't make boring Shiny apps!

- Shiny app UI
 - Bootstrap open source, responsive grid system and library of shiny components
 - <https://shiny.rstudio.com/articles/layout-guide.html>
- Many people don't bother to customise their UI
 - ways to customise your UI:
 - HTML, CSS style sheets, JavaScript, SASS, preset themes



UI packages I'm loving at the moment:

Layout/Widgets/Themes/UX:

- shinydashboard
- shinydashboardPlus
 - <https://github.com/RinteRface/shinydashboardPlus>
- shiny.semantic
- shinythemes
 - <https://rstudio.github.io/shinythemes/>
- bslib
 - <https://github.com/rstudio/bslib>

Maps:

- leaflet

Data Vis:

- plotly, ggplot ([ggplotly](#))
- r2d3 - D3 visualizations (advanced)

Displaying tables, upload / download data:

- data.table
- DT

The screenshot shows a shiny dashboard application titled "Theme demo". The main interface features several input components: a sliderInput with values 30 and 70, a selectizeInput with value "AL", a selectizeInput(multiple=T) with value "NULL", and a dateInput with value "2020-10-06". Below these is a dateRangeInput with values "2020-10-06" and "2020-10-13". A text box displays the R code used to bind these inputs to variables: "List of 5 \$ sliderInput : int [1:2] 30 70 \$ selectizeInput : chr "AL" \$ selectizeMultiInput: NULL \$ dateInput : Date[1:1], format: "2020-10-06" \$ dateRangeInput : Date[1:2], format: "2020-10-06" ...". To the right of the main interface is a "Theme customizer" sidebar. It includes sections for "Basic colors" (Background color: #202123, Foreground color: #B8BCC2), "Accent colors", "Fonts", "Options", and "Spacing". At the bottom of the sidebar, there are six colored buttons labeled "Primary", "Secondary (default)", "Success", "Info", "Warning", and "Danger".

TIP #3 – MAKE YOUR DOCUMENTS SHINE

INTEGRATE YOUR SHINY SKILLS IN YOUR .RMD DOCUMENTS & PRESENTATIONS

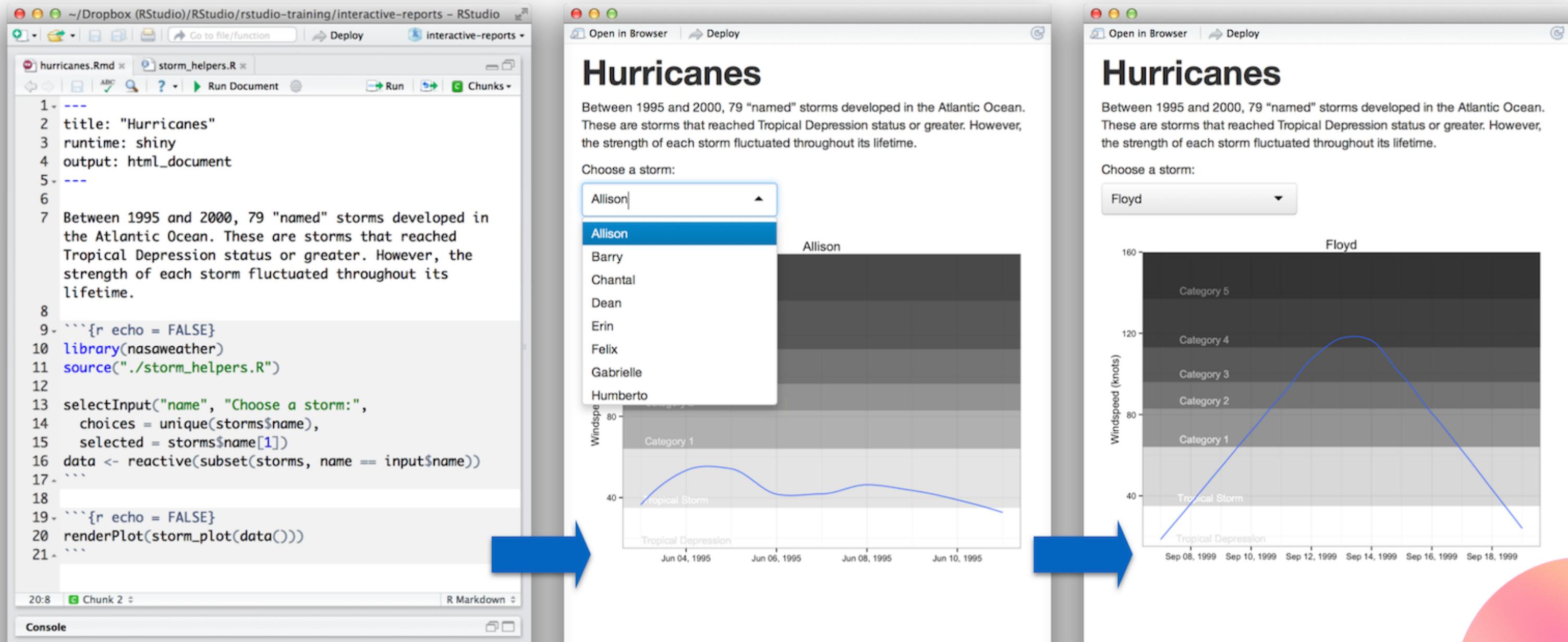
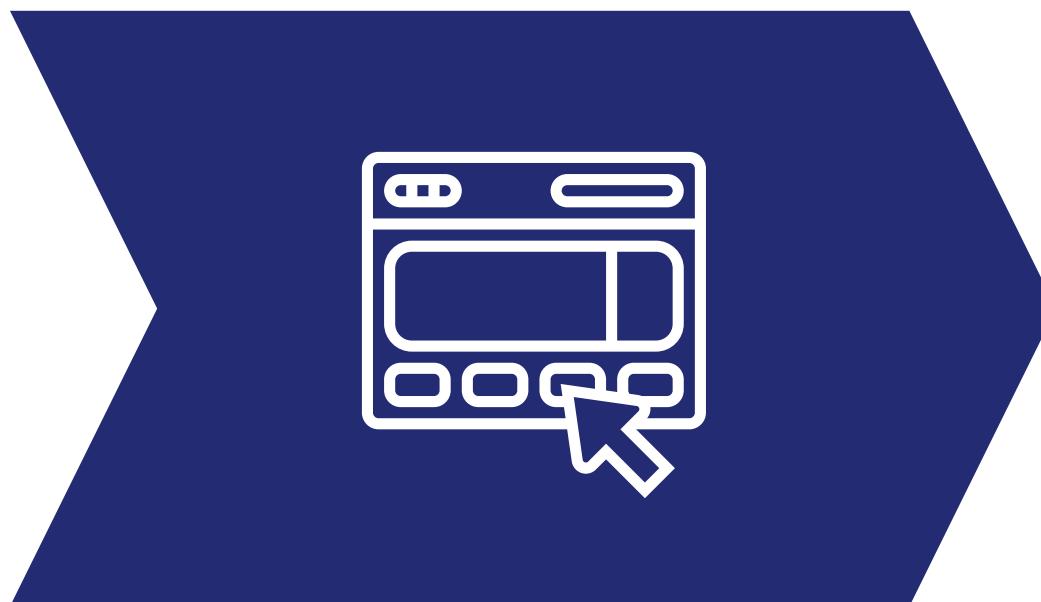


Image: <https://blog.rstudio.com/2014/06/19/interactive-documents-an-incredibly-easy-way-to-use-shiny/>

SCALING SHINY APPS

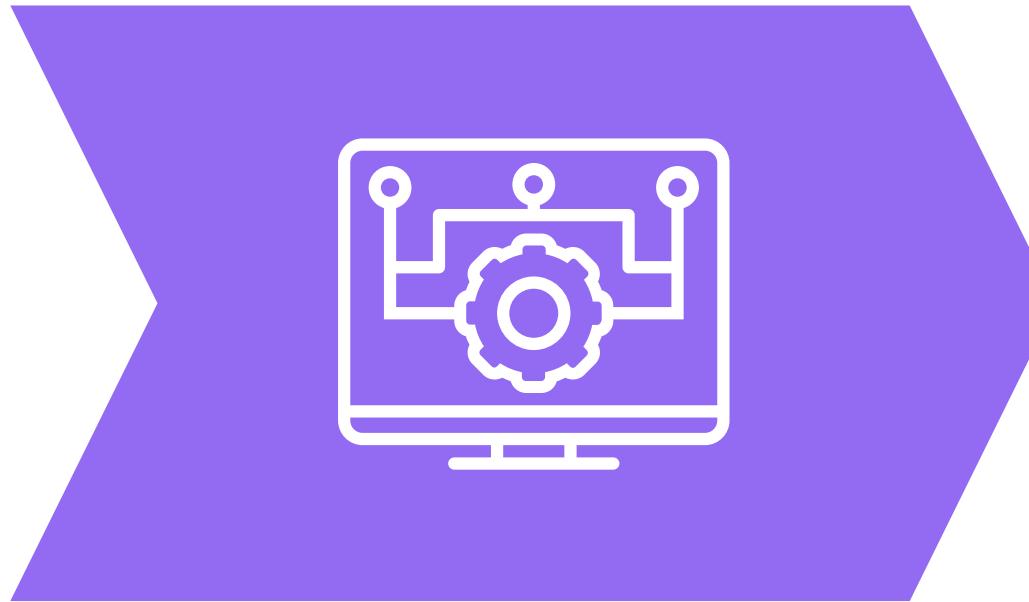
scale vertically first (more users 1 machine), then horizontally (across multiple machines)

Leverage the front end (ui)



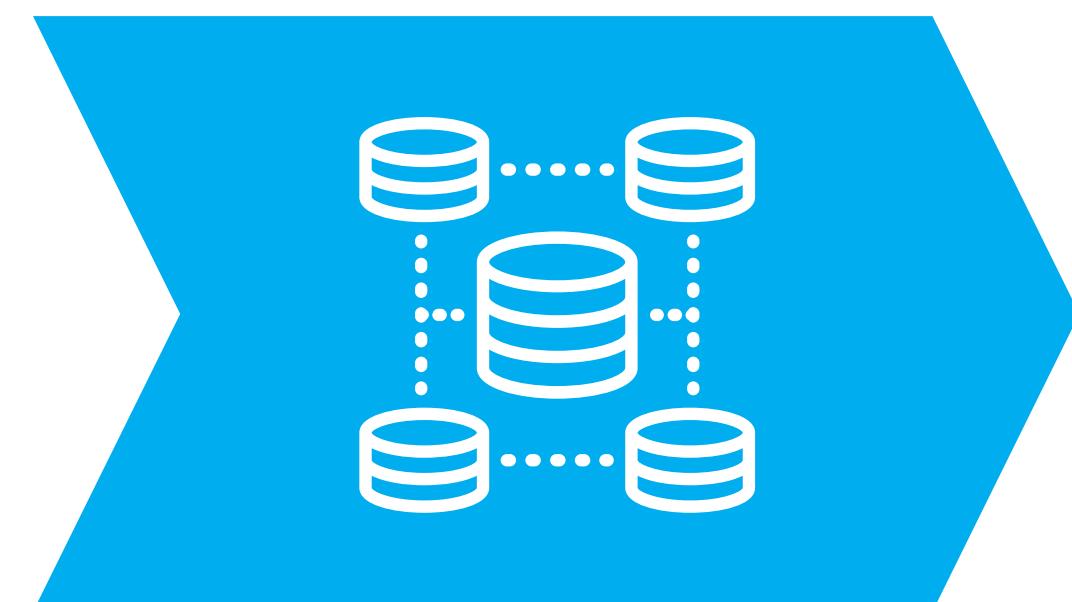
Modularise code, render outputs in ui not server, JavaScript (package by Dean Atali [ShinyJS](#))

Extract computations



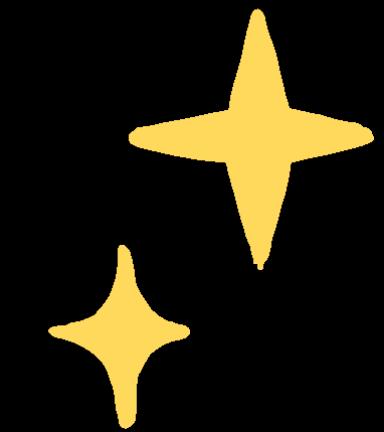
Remote API - load only what's needed (package called [Plumbr](#) - generate rest API)

Use a database & organise architecture



SHINY DOCUMENTS & PRESENTATIONS

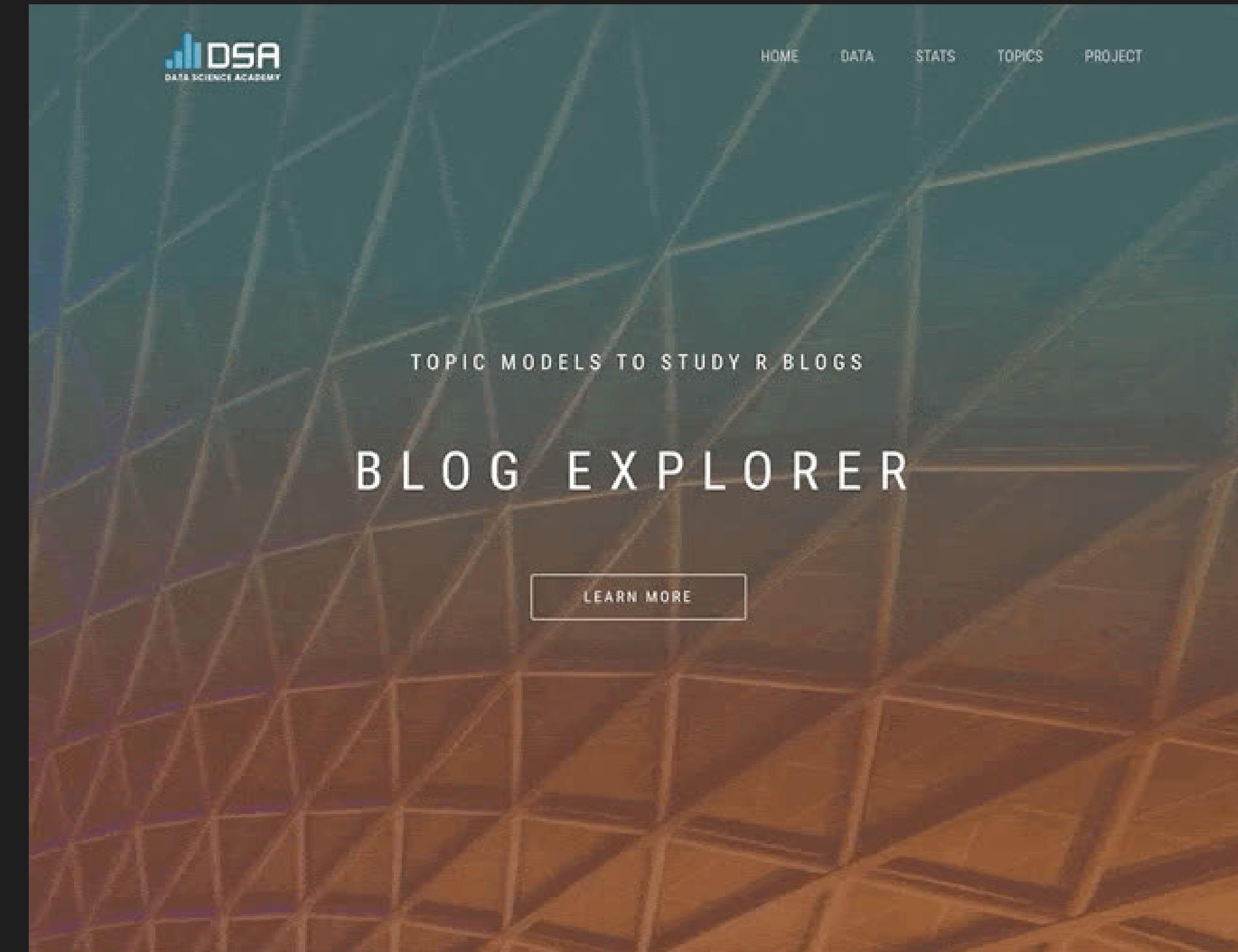
See the .Rmd documents in the
`Shiny/Code_Slides/` folder for code
examples



THANK you

get in touch:

@rhetta_chappell
r.chappell@griffith.edu.au
ridl.com.au



<https://nz-stefan.shinyapps.io/blog-explorer/>