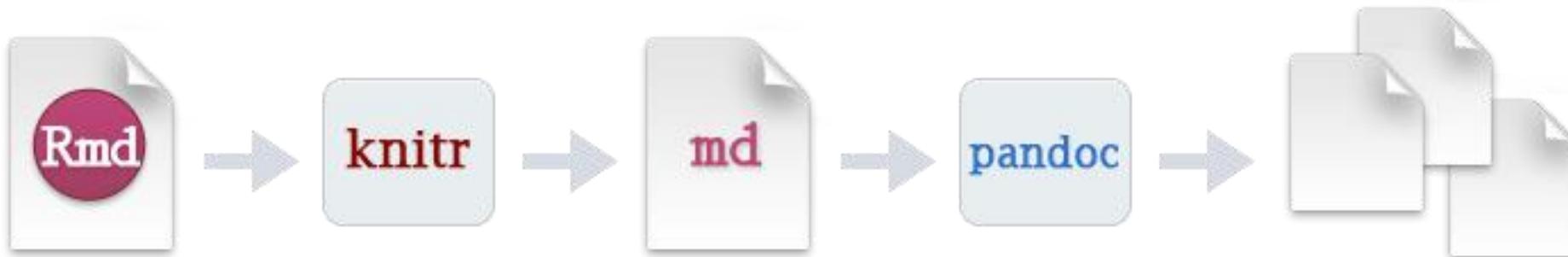


Notebooks and shiny apps

Using Markdown

How it works



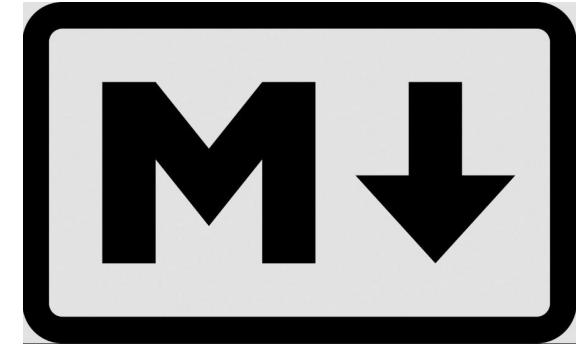
When you run `render`, R Markdown feeds the .Rmd file to [knitr](#), which executes all of the code chunks and creates a new markdown (.md) document which includes the code and its output.

The markdown file generated by knitr is then processed by [pandoc](#)  which is responsible for creating the finished format.

This may sound complicated, but R Markdown makes it extremely simple by encapsulating all of the above processing into a single `render` function.

Commentary sections use ‘Markdown’

- Simple markup language
- Designed to be nicely readable as plain text
- Compiles to properly formatted text
- Simple syntax



Markdown basics

- Headings

```
# Heading 1
```

```
## Heading 2
```

```
### Heading 3 etc.
```

Heading 1

Heading 2

- Lists (need a blank line first)

- * Bullet 1

- [Tab] * Sub-bullet 1

- * Bullet 2

1. Numbered 1

2. Numbered 2



Heads also give you navigation for your document, so they're worth using!

Markdown basics

- Emphasis

italics

italics

bold

__bold__

bold italics

__bold italics__

vol=width*depth*height

NOT bold (escaped)

- Other formatting

` `` `fixed width code etc` `` `

> quoted text

super^{script}[^]

sub~script~

***** or ----- page break

Needs blank line above and below

Markdown basics

- Tables

Name	Quest	Success
Simon	To teach R	Sometimes
Emma	To teach the world to sing	Always
Libby	To pass her GCSEs	Unknown

:--- Left Justified

:--: Centred

---: Right Justified

Markdown basics

$$e = mc^2$$

```
$e=mc^2$
```

- Markdown supports Latex equations.
 - `$equation$` is inline with text
 - `$$equation$$` is as a separate block

$$\sum_{i=1}^n X_i$$

```
$\sum_{i=1}^n X_i$
```

$$F_{i,j}$$

```
$F_{i,j}$
```

$$\sqrt{x^2 - 5y}$$

```
$\sqrt{x^2 - 5y}$
```

$$\sum_{i=1}^n \left(\frac{X_i}{Y_i} \right)$$

```
$\sum_{i=1}^n \left( \frac{X_i}{Y_i} \right)$
```

R code block details

Working directories

- Working directories
 - Working directory is automatically set to directory with Rmd file
 - That's why we immediately save
 - Designed so that data and code all go together
 - Can run `setwd` but get a warning, and only lasts for 1 block

Good code block practices

- Break code into short chunks
- All chunks are part of the same session
- Stop the block as soon as any output is generated

```
```{r}
tibble(x=1:5) -> some.data
some.data
```
```

| x | <int> |
|---|-------|
| | 1 |
| | 2 |
| | 3 |
| | 4 |
| | 5 |

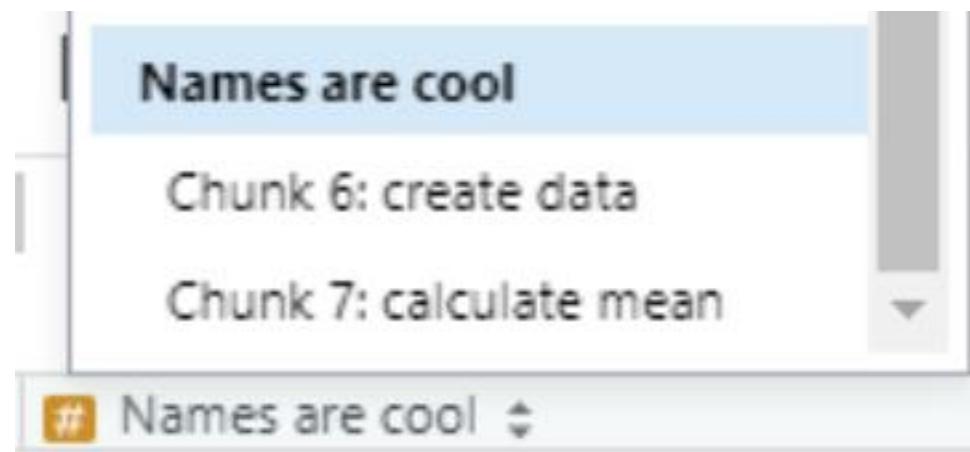
5 rows

```
```{r}
some.data %>% pull(x) %>% mean()
```
```

[1] 3

Good code block practices

- Name your chunks
- Name appears in the navigation along with headings you've created



Names are cool

```
```{r "create data"}  
tibble(x=1:5) -> some.data
some.data
```
```

```
```{r "calculate mean"}  
some.data %>% pull(x) %>% mean()
```
```

Displaying tibbles

- By default you don't see the text form of tibbles/dataframes
- You get a nice interactive table
 - Not in all output formats
- Buttons to see more columns/rows

```
```{r}
read_csv("Child_Variants.csv") -> child
child
```

R Console

CHR <dbl>	POS <dbl>	dbSNP <chr>	REF <chr>	ALT <chr>	QUAL <dbl>	GENE <chr>
1	69270	.	A	G	16	OR4F5
1	69511	rs75062661	A	G	200	OR4F5
1	69761	.	A	T	200	OR4F5
1	69897	rs75758884	T	C	59	OR4F5
1	877831	rs6672356	T	C	200	SAMD11
1	881627	rs2272757	G	A	200	NOC2L
1	887801	rs3828047	A	G	200	NOC2L
1	888639	rs3748596	T	C	200	NOC2L
1	888659	rs3748597	T	C	200	NOC2L
1	889158	rs13303056	G	C	200	NOC2L

1-10 of 25,822 rows | 1-7 of... Previous 1 2 3 4 5 6 ... 100 Next

# Displaying tibbles

- Although you only see 10 rows, all of the data goes into your document
- When rendered to HTML / PDF this can make your document BIG
- Use the `head()` function to only show a few example rows

```
```{r}
read_csv("Child_Variants.csv") -> child
head(child, n=20)
````
```

| CHR<br><dbl> | POS<br><dbl> | dbSNP<br><chr> | REF<br><chr> | ALT<br><chr> | QUAL<br><dbl> | GENE<br><chr> |
|--------------|--------------|----------------|--------------|--------------|---------------|---------------|
| 1            | 69270        | .              | A            | G            | 16            | OR4F5         |
| 1            | 69511        | rs75062661     | A            | G            | 200           | OR4F5         |
| 1            | 69761        | .              | A            | T            | 200           | OR4F5         |
| 1            | 69897        | rs75758884     | T            | C            | 59            | OR4F5         |
| 1            | 877831       | rs6672356      | T            | C            | 200           | SAMD11        |
| 1            | 881627       | rs2272757      | G            | A            | 200           | NOC2L         |
| 1            | 887801       | rs3828047      | A            | G            | 200           | NOC2L         |
| 1            | 888639       | rs3748596      | T            | C            | 200           | NOC2L         |
| 1            | 888659       | rs3748597      | T            | C            | 200           | NOC2L         |
| 1            | 889158       | rs13303056     | G            | C            | 200           | NOC2L         |

1-10 of 20 rows | 1-7 of 11 columns

Previous 1 2 Next

# Controlling warnings / errors / messages

```
```{r}
library(tidyverse)
```

Registered S3 methods overwritten by 'dbplyr':
 method from
 print.tbl_lazy
 print.tbl_sql

-- Attaching packages --
v ggplot2 3.3.2 v purrr 0.3.4
v tibble 3.0.4 v dplyr 1.0.2
v tidyr 1.1.2 v stringr 1.4.0
v readr 1.4.0 vforcats 0.5.0

-- Conflicts --
x dplyr::filter() masks stats::filter()
x dplyr::lag() masks stats::lag()
```

```
```{r}
read_tsv("small_file.txt") -> small
```

-- Column specification --
cols(
 Sample = col_character(),
 Length = col_double(),
 Category = col_character()
)
```

# Controlling warnings / errors / messages

- Can select which output you want to see using the block header

```
```{r "Block name", warnings=FALSE}
```

- Can remove

- Warnings {r warnings=FALSE}
- Errors {r error=TRUE} means that script doesn't stop on error
- Messages {r message=FALSE}
- Code {r echo=FALSE}
- Code + output {r include=FALSE}

Changing graphics options

- You can change the way that figures / graphs are displayed by changing R code block options

- Change the file format (default is PNG)

```
```{r dev="svg"}
```

- Change the size

```
```{r fig.height=5, fig.width=8}
```

- Change the alignment (only affected compiled document)

```
```{r fig.align="center"}
```

- Add a legend

```
```{r fig.cap="This is a great picture"}
```

Changing document appearance

Table of Contents

- If you have used headings in your document then you can auto-create a table of contents
- This can be a fixed set of links at the top of your document, or a floating table on the left
- This is set in the header section



Processing

Read the data

```
read_tsv("small_file.txt")
```

```
##  
## -- Column specification  
## cols(
```

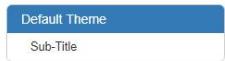
```
title: "Example Notebook"  
output:  
html_document:  
df_print: paged  
toc: yes  
toc_float: yes
```

Document themes

- HTML documents are based on the bootswatch theme collection (<https://bootswatch.com>)
- You can change the theme by adding to the header

```
title: "Themes"
output:
  html_document:
    df_print: paged
    toc: true
    toc_float: true
theme: yeti
highlight: kate
```

Document themes



Themes

Default Theme

This is a small document to show the effect of changing the themes.

Sub-Title

- Themes can change
- The overall appearance

of your document in a quick and easy fashion.



Themes

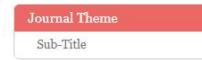
Cerulean Theme

This is a small document to show the effect of changing the themes.

Sub-Title

- Themes can change
- The overall appearance

of your document in a quick and easy fashion.



Themes

Journal Theme

This is a small document to show the effect of changing the themes.

Sub-Title

- Themes can change
- The overall appearance

of your document in a quick and easy fashion.



Themes

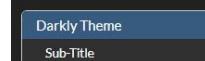
Flatly Theme

This is a small document to show the effect of changing the themes.

Sub-Title

- Themes can change
- The overall appearance

of your document in a quick and easy fashion.



Themes

Darkly Theme

This is a small document to show the effect of changing the themes.

Sub-Title

- Themes can change
- The overall appearance

of your document in a quick and easy fashion.



Themes

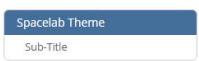
Readable Theme

This is a small document to show the effect of changing the themes.

Sub-Title

- Themes can change
- The overall appearance

of your document in a quick and easy fashion.



Themes

Spacelab Theme

This is a small document to show the effect of changing the themes.

Sub-Title

- Themes can change
- The overall appearance

of your document in a quick and easy fashion.



Themes

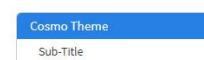
United Theme

This is a small document to show the effect of changing the themes.

Sub-Title

- Themes can change
- The overall appearance

of your document in a quick and easy fashion.



Themes

Cosmo Theme

This is a small document to show the effect of changing the themes.

Sub-Title

- Themes can change
- The overall appearance

of your document in a quick and easy fashion.

(there are more than this)

Highlighting themes

- Similarly to the document themes you can also change the colouring / style used to highlight R code in your document

```
title: "Themes"
output:
  html_document:
    df_print: paged
    toc: true
    toc_float: true
    theme: yeti
highlight: kate
```

Highlighting themes

Haddock

```
library(tidyverse)

starwars %>%
  filter(height>150) %>%
  arrange(desc(birth_year)) %>%
  filter(gender=="male")
```

Monochrome

```
library(tidyverse)

starwars %>%
  filter(height>150) %>%
  arrange(desc(birth_year)) %>%
  filter(gender=="male")
```

Tango

```
library(tidyverse)

starwars %>%
  filter(height>150) %>%
  arrange(desc(birth_year)) %>%
  filter(gender=="male")
```

Zenburn

```
library(tidyverse)

starwars %>%
  filter(height>150) %>%
  arrange(desc(birth_year)) %>%
  filter(gender=="male")
```

Kate

```
library(tidyverse)

starwars %>%
  filter(height>150) %>%
  arrange(desc(birth_year)) %>%
  filter(gender=="male")
```

Pygments

```
library(tidyverse)

starwars %>%
  filter(height>150) %>%
  arrange(desc(birth_year)) %>%
  filter(gender=="male")
```

Espresso

```
library(tidyverse)

starwars %>%
  filter(height>150) %>%
  arrange(desc(birth_year)) %>%
  filter(gender=="male")
```

Textmate

```
library(tidyverse)

starwars %>%
  filter(height>150) %>%
  arrange(desc(birth_year)) %>%
  filter(gender=="male")
```

Tibble / DataFrame display options

- Rather than text output you see an interactive HTML version of tibbles
 - This will vary by output document type
- A few options exist for how they are displayed these are set in the header, and are specific to the HTML output type:

```
html_document:  
  df_print: paged
```

Tibble / DataFrame display options

	Option value	Text or HTML	Fits to space	Restricts rows	Paging controls
Only works on data frames	default	text	no	no	no
	kable	HTML	no	no	no
	tibble	text	yes	yes	no
This is the default	paged	HTML	yes	yes	yes

Tibble / DataFrame display options

Tibble

```
## # A tibble: 20 x 5
##   Plant Type Treatment conc uptake
##   <chr> <chr>    <chr> <dbl>  <dbl>
## 1 Qn1  Quebec nonchilled  95   16
## 2 Qn1  Quebec nonchilled 175  30.4
## 3 Qn1  Quebec nonchilled 250  34.8
## 4 Qn1  Quebec nonchilled 350  37.2
## 5 Qn1  Quebec nonchilled 500  35.3
## 6 Qn1  Quebec nonchilled 675  39.2
## 7 Qn1  Quebec nonchilled 1000 39.7
## 8 Qn2  Quebec nonchilled  95   13.6
## 9 Qn2  Quebec nonchilled 175  27.3
## 10 Qn2  Quebec nonchilled 250  37.1
## 11 Qn2  Quebec nonchilled 350  41.8
## 12 Qn2  Quebec nonchilled 500  40.6
## 13 Qn2  Quebec nonchilled 675  41.4
## 14 Qn2  Quebec nonchilled 1000 44.3
## 15 Qn3  Quebec nonchilled  95   16.2
## 16 Qn3  Quebec nonchilled 175  32.4
## 17 Qn3  Quebec nonchilled 250  40.3
## 18 Qn3  Quebec nonchilled 350  42.1
## 19 Qn3  Quebec nonchilled 500  42.9
## 20 Qn3  Quebec nonchilled 675  43.9
```

Kable

Plant	Type	Treatment	conc	uptake
Qn1	Quebec	nonchilled	95	16.0
Qn1	Quebec	nonchilled	175	30.4
Qn1	Quebec	nonchilled	250	34.8
Qn1	Quebec	nonchilled	350	37.2
Qn1	Quebec	nonchilled	500	35.3
Qn1	Quebec	nonchilled	675	39.2
Qn1	Quebec	nonchilled	1000	39.7
Qn2	Quebec	nonchilled	95	13.6
Qn2	Quebec	nonchilled	175	27.3
Qn2	Quebec	nonchilled	250	37.1
Qn2	Quebec	nonchilled	350	41.8
Qn2	Quebec	nonchilled	500	40.6
Qn2	Quebec	nonchilled	675	41.4
Qn2	Quebec	nonchilled	1000	44.3
Qn3	Quebec	nonchilled	95	16.2
Qn3	Quebec	nonchilled	175	32.4
Qn3	Quebec	nonchilled	250	40.3
Qn3	Quebec	nonchilled	350	42.1
Qn3	Quebec	nonchilled	500	42.9
Qn3	Quebec	nonchilled	675	43.9

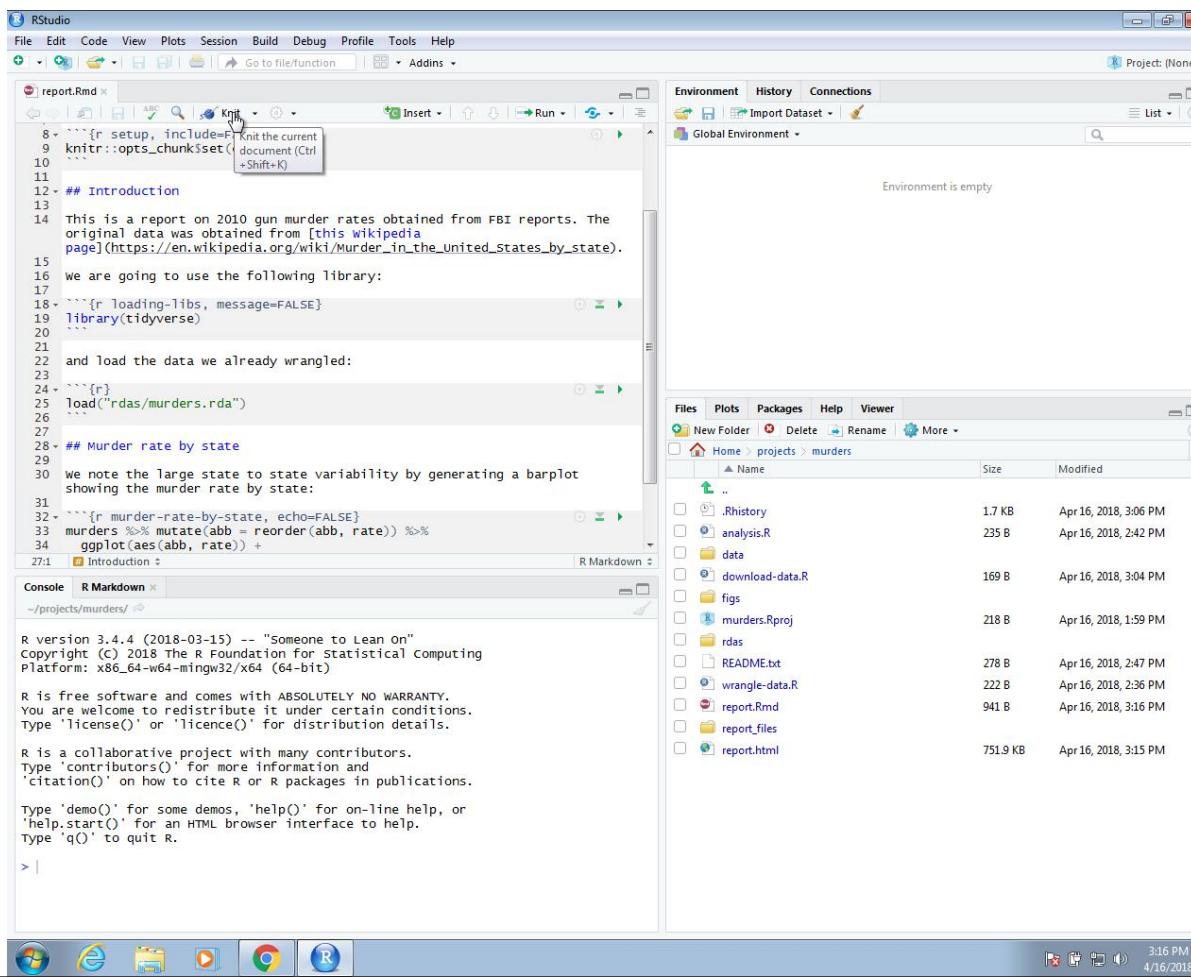
Paged

Plant	Type	Treatment	conc	uptake
<chr>	<chr>	<chr>	<dbl>	<dbl>
Qn1	Quebec	nonchilled	95	16.0
Qn1	Quebec	nonchilled	175	30.4
Qn1	Quebec	nonchilled	250	34.8
Qn1	Quebec	nonchilled	350	37.2
Qn1	Quebec	nonchilled	500	35.3
Qn1	Quebec	nonchilled	675	39.2
Qn1	Quebec	nonchilled	1000	39.7
Qn2	Quebec	nonchilled	95	13.6
Qn2	Quebec	nonchilled	175	27.3
Qn2	Quebec	nonchilled	250	37.1
Qn2	Quebec	nonchilled	350	41.8
Qn2	Quebec	nonchilled	500	40.6
Qn2	Quebec	nonchilled	675	41.4
Qn2	Quebec	nonchilled	1000	44.3
Qn3	Quebec	nonchilled	95	16.2
Qn3	Quebec	nonchilled	175	32.4
Qn3	Quebec	nonchilled	250	40.3
Qn3	Quebec	nonchilled	350	42.1
Qn3	Quebec	nonchilled	500	42.9
Qn3	Quebec	nonchilled	675	43.9

1-10 of 20 rows Previous 1 2 Next

knitR

We use the **knitr** package to compile R markdown documents. The specific function used to compile is the **knit** function, which takes a filename as input. RStudio provides a button that makes it easier to compile the document.



Example

```
--- title: "Report on Gun Murders"
```

```
author: "Rafael Irizarry"
```

```
date: ``r format(Sys.Date())``"
```

```
output: github_document
```

```
---
```

```
```{r setup, include=FALSE}
```

```
knitr::opts_chunk$set(echo = TRUE)
```

```
```
```

```
## Introduction
```

This is a report on 2010 gun murder rates obtained from FBI reports. The original data was obtained from [this Wikipedia page](https://en.wikipedia.org/wiki/Murder_in_the_United_States_by_state). We are going to use the following library:

```
```{r loading-libs, message=FALSE}
```

```
library(tidyverse)
```

```
```
```

and load the data we already wrangled:

```
```{r} load("rdas/murders.rda")
```

```
```
```

```
## Murder rate by state
```

We note the large state to state variability by generating a barplot showing the murder rate by state:

```
```{r murder-rate-by-state, echo=FALSE}
```

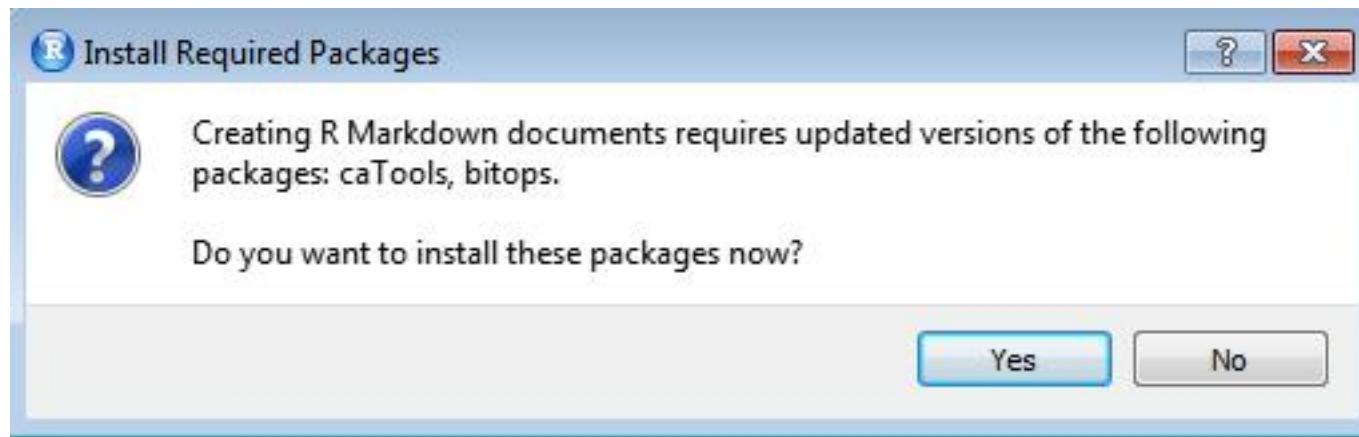
```
murders %>% mutate(abb = reorder(abb, rate)) %>%
```

```
ggplot(aes(abb, rate)) + geom_bar(width = 0.5, stat = "identity", color = "black") + coord_flip()
```

```
```
```

Knit button

The first time you click on the *Knit* button, a dialog box may appear asking you to install packages you need:



Once you have installed the packages, clicking the *Knit* will compile your R markdown file and the resulting document will pop-up:

This produces an html document which you can see in your working directory. To view it, open a terminal and list the files. You can open the file in a browser and use this to present your analysis. You can also produce a PDF or Microsoft document by changing:
output: html_document to output: pdf_document or output: word_document.

github_document

We can also produce documents that render on GitHub using output: github_document like this:

The screenshot shows the RStudio interface with the following components:

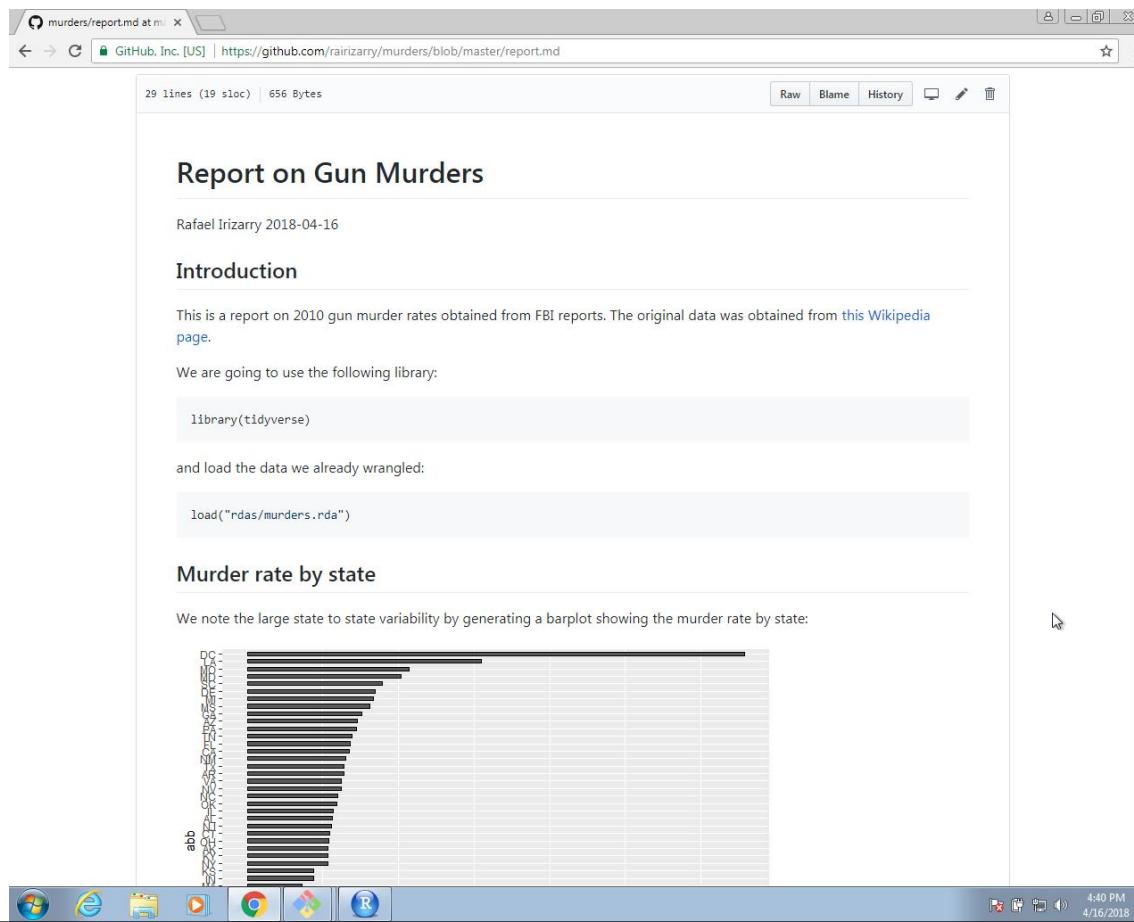
- Top Bar:** RStudio, File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, Help.
- Left Panel:** A code editor window titled "report.Rmd*". It contains R code for generating a report on gun murders. The code includes a YAML header, setup chunks, and a library loading chunk. It also includes a section for wrangling data from an RDA file and a section for analyzing murder rates by state.
- Middle Panel:** An "Environment" panel showing the Global Environment is empty.
- Right Panel:** A "Files" panel showing the project structure and files. The project is named "murders". Inside, there are subfolders ".data", "download-data.R", "figs", "murders.Rproj", "rdatas", "README.txt", "wrangle-data.R", and "report.Rmd". The "report.Rmd" file is listed with a size of 941 B and modified on Apr 16, 2018, 3:16 PM. The "report.html" file is also listed with a size of 752 KB and modified on Apr 16, 2018, 3:16 PM.
- Bottom Panel:** A "Console" tab showing the R environment details and a "R Markdown" tab showing the rendered HTML output.

The rendered output in the "R Markdown" tab includes:

- R version 3.4.4 (2018-03-15) -- "Someone to Lean On"
- Copyright (C) 2018 The R Foundation for Statistical Computing
- Platform: x86_64-w64-mingw32/x64 (64-bit)
- R is free software and comes with ABSOLUTELY NO WARRANTY.
- You are welcome to redistribute it under certain conditions.
- Type 'license()' or 'licence()' for distribution details.
- R is a collaborative project with many contributors.
- Type 'contributors()' for more information and
- 'citation()' on how to cite R or R packages in publications.
- Type 'demo()' for some demos, 'help()' for on-line help, or
- 'help.start()' for an HTML browser interface to help.
- Type 'q()' to quit R.

Web report

This will produce a markdown file, with suffix **md**, that renders in GitHub. Because we have uploaded these files to GitHub, you can click on the **md** file and you will see the report as a webpage:



Slide presentation

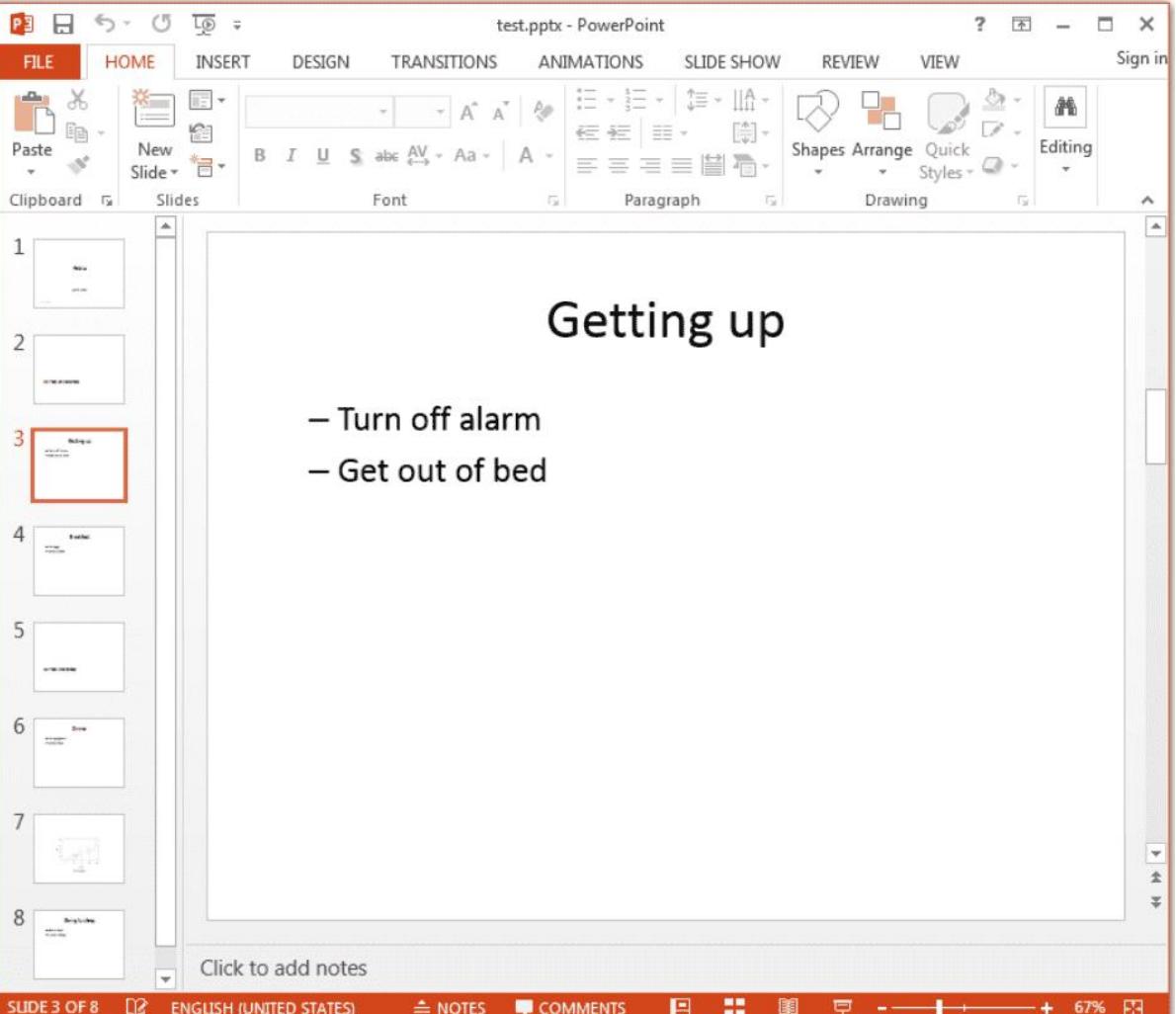
R Markdown renders to four presentation formats:

- [beamer presentation](#) - PDF presentations with beamer
- [ioslides presentation](#) - HTML presentations with ioslides
- [slidy presentation](#) - HTML presentations with slidy
- [powerpoint presentation](#) - PowerPoint presentation
- [revealjs::revealjs presentation](#) - HTML presentations with reveal.js

Each format will intuitively divide your content into slides, with a new slide beginning at each first or second level header.

Power Point

```
--- title: "Habits"  
author: John Doe  
date: March 22, 2005  
output: powerpoint_presentation  
---  
  
# In the morning  
## Getting up  
- Turn off alarm  
- Get out of bed  
## Breakfast  
- Eat eggs  
- Drink coffee  
# In the evening  
## Dinner  
- Eat spaghetti  
- Drink wine  
---  
```{r, cars, fig.cap="A scatterplot.", echo=FALSE}  
plot(cars)
```  
  
## Going to sleep  
- Get in bed  
- Count sheep
```



The screenshot shows a Microsoft PowerPoint presentation window titled "test.pptx - PowerPoint". The ribbon menu is visible at the top, with the "HOME" tab selected. The left sidebar displays a list of slides numbered 1 through 8. Slide 3 is currently selected and contains the following content:

Getting up

- Turn off alarm
- Get out of bed

At the bottom of the slide, there is a note placeholder: "Click to add notes". The status bar at the bottom of the screen shows "SLIDE 3 OF 8", "ENGLISH (UNITED STATES)", and a zoom level of "67%".

Dashboard

Dashboards are a useful way to communicate large amounts of information visually and quickly. Create one with the `flexdashboard::flex_dashboard` output format

The screenshot shows the RStudio interface with an R Markdown file open on the left and its corresponding dashboard output on the right.

Code View (Left):

```
1 ---  
2 title: "Review Dashboard"  
3 output:  
4   flexdashboard::flex_dashboard:  
5     orientation: columns  
6 ---  
7  
8 `r include = FALSE`  
9 library(viridis)  
10 library(ggplot2)  
11 library(marmap)  
12 ``  
13  
14 # Intro {.sidebar}  
15  
16 This dashboard covers several topics:  
17  
18 * The marmap package  
19 * The viridis package  
20 * Miscellaneous material  
21  
22 # Marmap Package  
23  
24 ## Column 1  
25  
26 ### Florida  
27  
28 `r echo = FALSE`  
29 data(florida)  
30 autoplot(florida)
```

Dashboard View (Right):

The dashboard has a blue header bar with tabs: "Review Dashboard", "Marmap Package", "Viridis Package", and "Miscellaneous".

- Review Dashboard:** A sidebar section titled "This dashboard covers several topics:" with a bulleted list:
 - The marmap package
 - The viridis package
 - Miscellaneous material
- Marmap Package:** A section titled "Florida" containing text about the marmap package and a contour plot of the Florida dataset.
- Viridis Package:** A section titled "Hawaii" containing a contour plot of the Hawaii dataset.
- Miscellaneous:** A section titled "Alaska" containing a contour plot of the Alaska dataset.

Flexdashboard makes it easy to organize your content into a visual layout:

- Each Level 1 Header (#) begins a new page in the dashboard.
- Each Level 2 Header (##) begins a new column.
- Each Level 3 Header (###) begins a new box.

Interactive documents

Markdown documents are a perfect platform for interactive content. To make your documents interactive, add:

1. Interactive JavaScript visualizations based on [htmlwidgets](#) or
2. Reactive components made with [Shiny](#)

The image shows a screenshot of the RStudio interface. On the left, the code editor displays an R Markdown file named "13-htmlwidget.Rmd". The code includes a YAML header and an R chunk that uses the leaflet package to create an interactive map of Mount Eden in Auckland, NZ. The right side of the interface shows the resulting "Interactive Map" output. The map features a green area representing Mount Eden, with a blue marker and a red info bubble labeled "Maunga Whau". The map also shows surrounding streets like Glenfell Place, Summer Road, and Owens Road, along with other landmarks like Wilkes Lookout and Mount Eden. A snippet of the R code used to generate the map is visible above the map area.

```
1 ---  
2 title: "Interactive Map"  
3 output: html_document  
4 ---  
5  
6 ````{r include = FALSE}  
7 library(leaflet)  
8 library(dplyr)  
9 ````  
10  
11 Use the leaflet map below to explore the actual Maunga Whau  
volcano in Auckland, NZ.  
12  
13 ````{r}  
14 leaflet() %>%  
15   setView(lng=174.764, lat=-36.877, zoom = 16) %>%  
16   addTiles() %>%  
17   addMarkers(lng=174.764, lat=-36.877, popup="Maunga Whau")  
18 ````
```

Shiny apps

Shiny

Shiny is an R package that makes it easy to build interactive web apps straight from R. You can host standalone apps on a webpage or embed them in [R Markdown](#) documents or build [dashboards](#).

You can also extend your Shiny apps with [CSS themes](#), [htmlwidgets](#), and [JavaScript actions](#).

web apps was for R users:

In the past

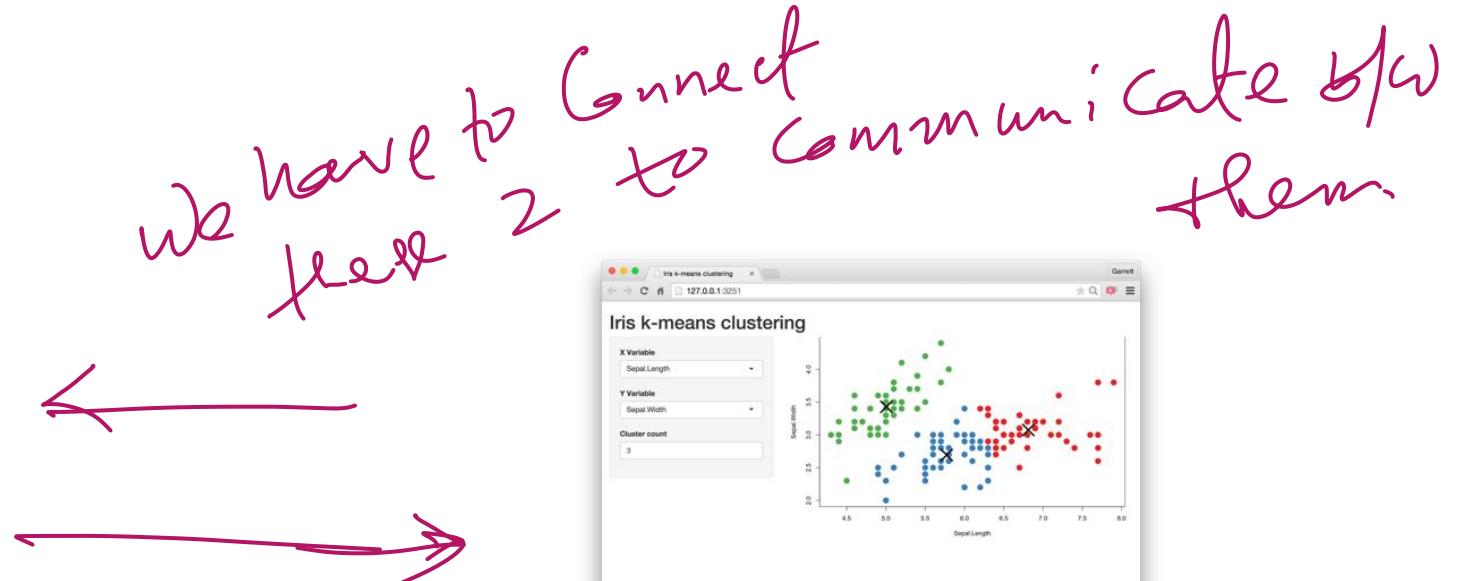
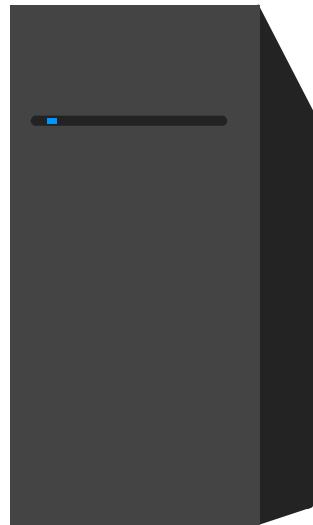
- You need a deep knowledge of web technologies like HTML, CSS and JavaScript.
- Making complex interactive apps requires careful analysis of interaction flows to make sure that when an input changes, only the related outputs are updated.

With shiny

- Providing a carefully curated set of user interface (UI for short) functions that generate the HTML, CSS, and JavaScript that you need for common tasks.

- Introducing a new style of programming called **reactive programming** which automatically tracks the dependencies of a code block.

Every Shiny app is maintained by a computer running R



Server Instructions

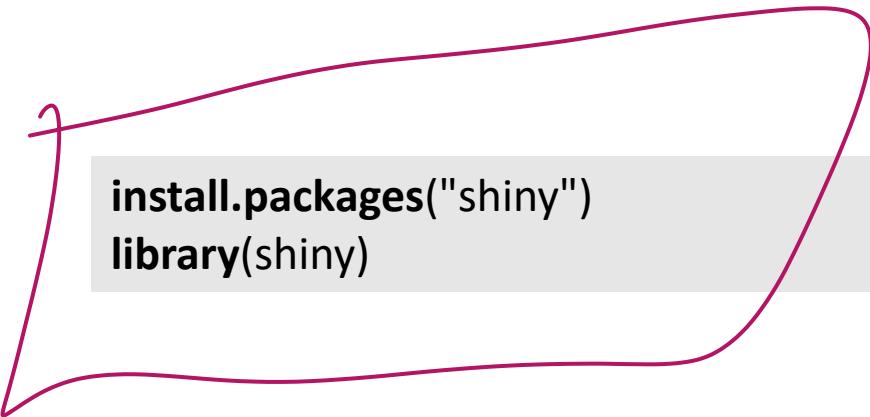


User Interface (UI)

Introduction

Two key components of every Shiny app:

- the **UI** (short for user interface) which defines how your app *looks*,
- the **server function** which defines how your app *works*. Shiny uses reactive programming to automatically update outputs when inputs change so we'll finish by learning the third important component of Shiny apps: reactive expressions.



```
install.packages("shiny")
library(shiny)
```

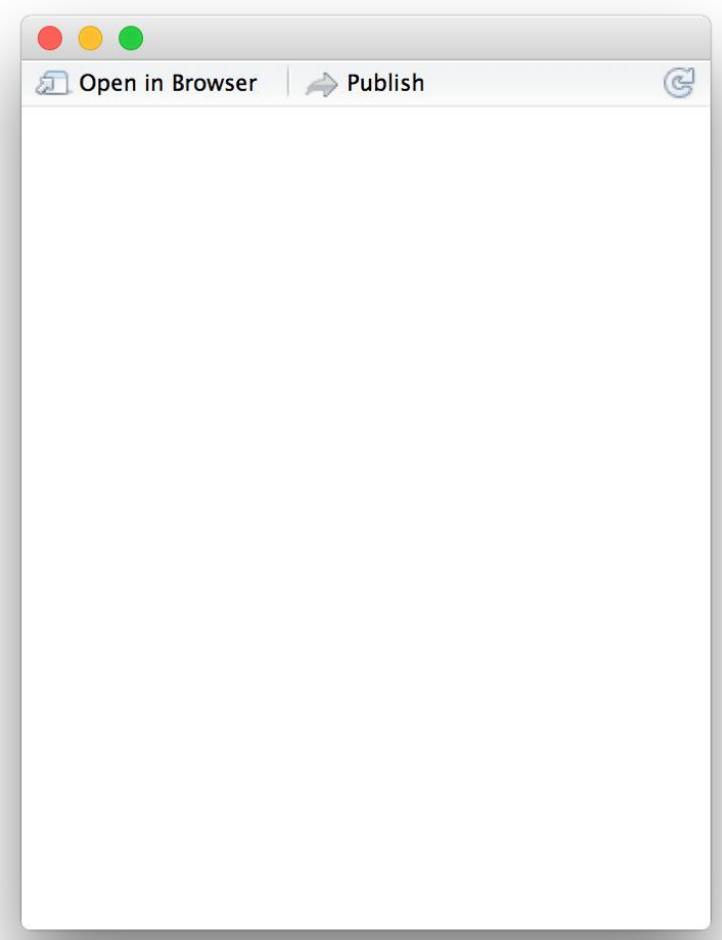
Build your app

Add elements to your app as arguments to
flui dPage()

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

Inputs

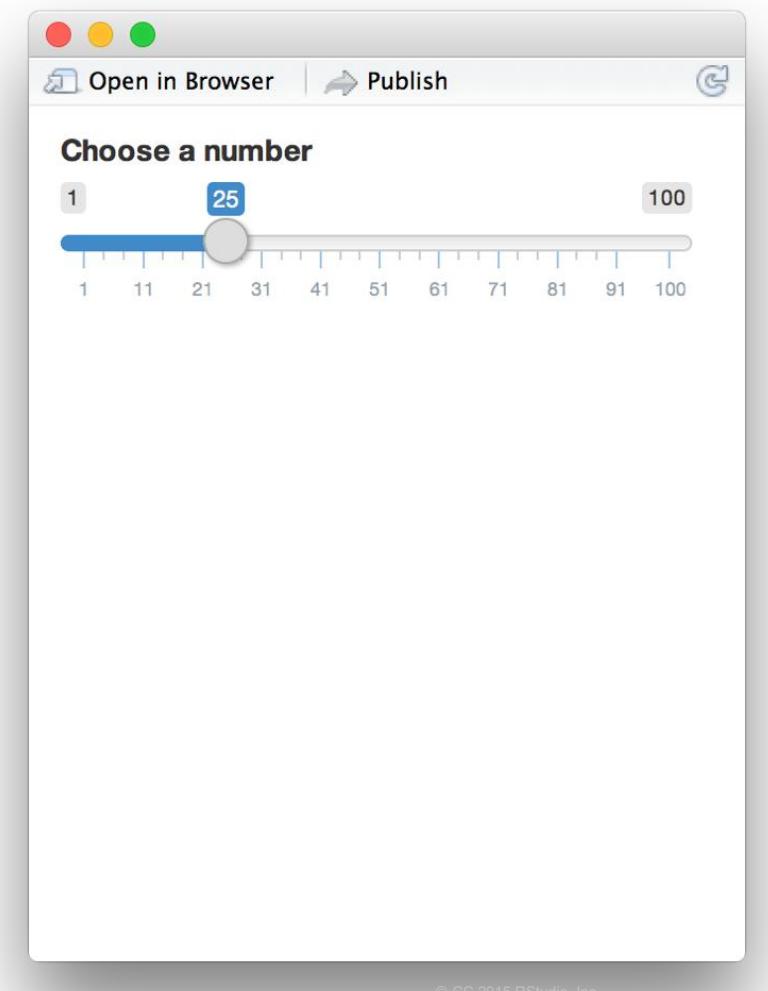
```
library(shiny)
ui <- fluidPage(
  )
server <- function(input, output) {}
shinyApp(server = server, ui = ui)
```



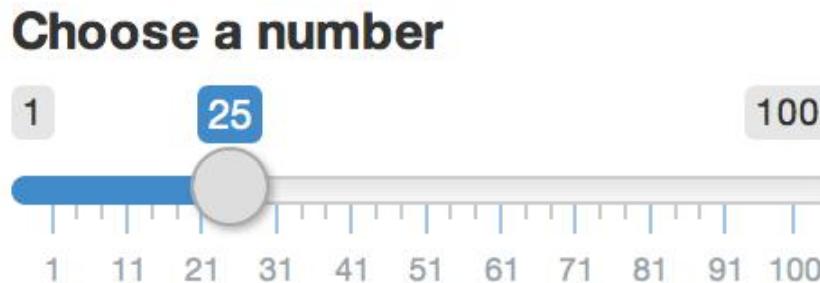
Inputs

```
{  
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)}
```

inputId
User Interface



Syntax



```
sliderInput(inputId = "num", label = "Choose a number", ...)
```

input name
(for internal use)

Notice:
Id not ID

label to
display

input specific
arguments

?sliderInput

Other inputs

Buttons

Action

Submit

`actionButton()`
`submitButton()`

Single checkbox

Choice A

Checkbox group

- Choice 1
- Choice 2
- Choice 3

Date input

2014-01-01

Date range

2014-01-24 to 2014-01-24

`dateRangeInput()`

File input

Choose File No file chosen

`fileInput()`

Numeric input

1

Password Input

.....

`passwordInput()`

Radio buttons

- Choice 1
- Choice 2
- Choice 3

`radioButtons()`

Select box

Choice 1

▼

`selectInput()`

Sliders



`sliderInput()`

Text input

Enter text...

`textInput()`

Outputs

To display output, add it to `fluidPage()` with an
* `Output()` function

plot `Output("hist")`

the type of output
to display

Outputs

| Function | Inserts |
|-----------------------------------|----------------------|
| <code>dataTableOutput()</code> | an interactive table |
| <code>htmlOutput()</code> | raw HTML |
| <code>imageOutput()</code> | image |
| <code>plotOutput()</code> | plot |
| <code>tableOutput()</code> | table |
| <code>textOutput()</code> | text |
| <code>uiOutput()</code> | a Shiny UI element |
| <code>verbatimTextOutput()</code> | text |

Outputs

```
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

how to assemble inputs into outputs ?

1

Server function

1

Save objects to display to output\$

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

2

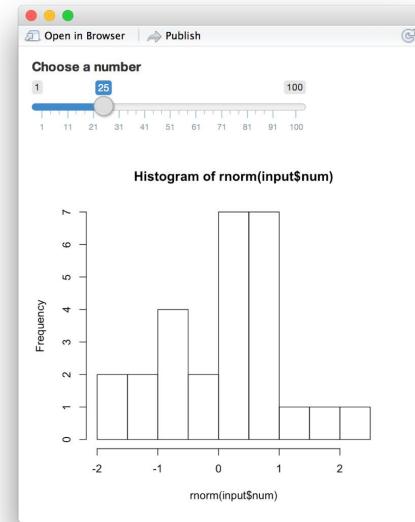
Build objects to display with **render***()

```
server <- function(input, output) {  
  output$hist <- renderPlot({  
    hist(rnorm(100))  
  })  
}
```

3

Access **input** values with input\$

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



Render

`textOutput()`

`renderText()`

Output function in UI code pairs with a render function in the server code.

render^{*}()

Builds reactive output to display in UI

```
renderPlot( { hist(rnorm(100)) } )
```

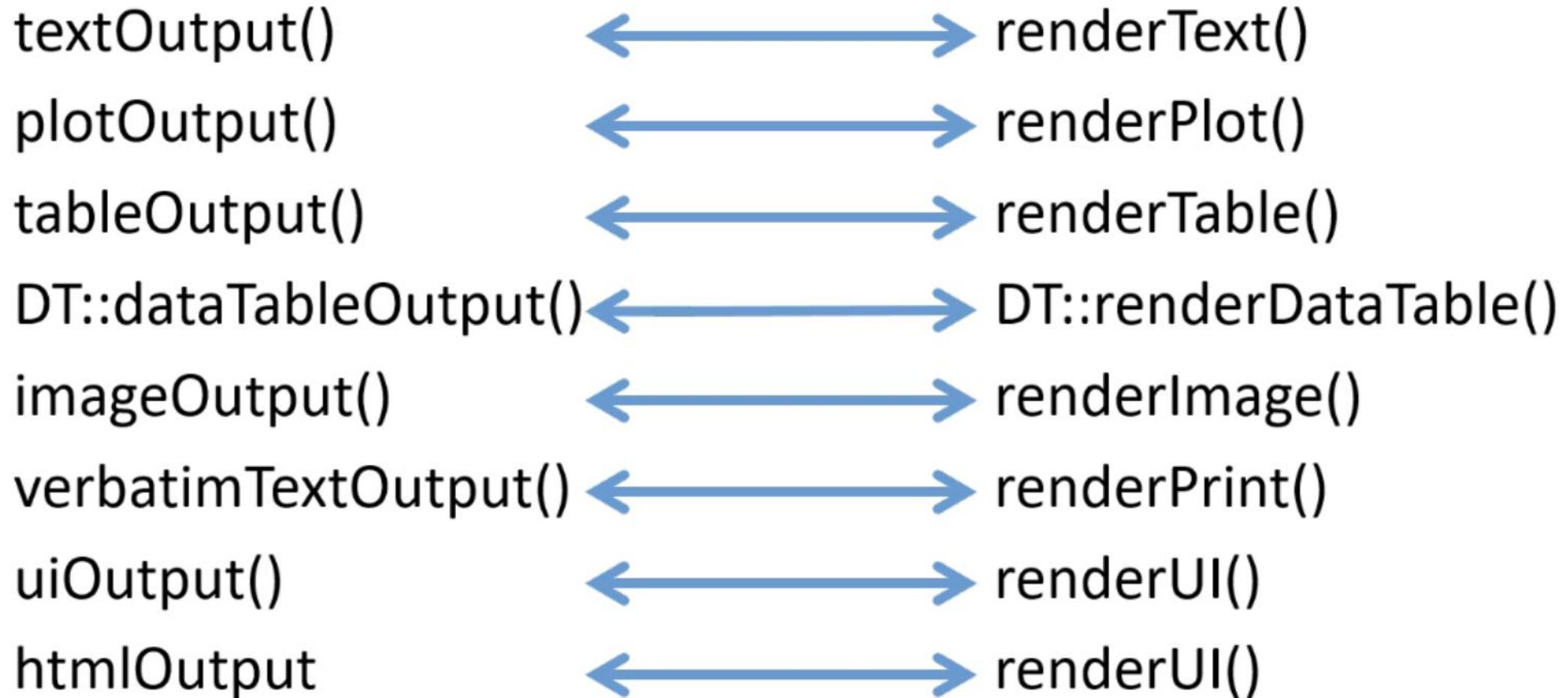
type of object to build

code block that builds the object

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

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

Outputs and render functions

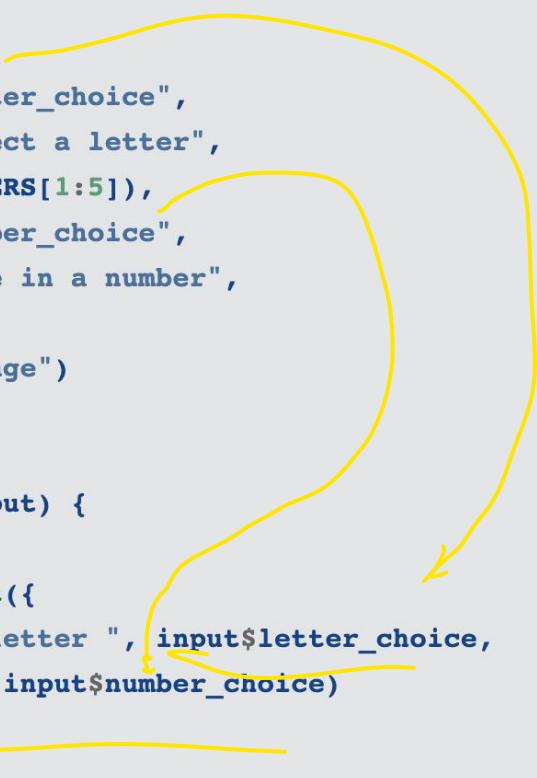


Simple app code

```
library(shiny)

ui <- fluidPage(
  titlePanel("Simple app"),
  radioButtons(inputId = "letter_choice",
    label = "select a letter",
    choices = LETTERS[1:5]),
  numericInput(inputId = "number_choice",
    label = "type in a number",
    value = 4),
  textOutput(outputId = "message")
)

server <- function(input, output) {
  output$message <- renderText({
    paste0("You selected the letter ", input$letter_choice,
      " and the number ", input$number_choice)
  })
}
shinyApp(ui, server)
```



Simple app

select a letter

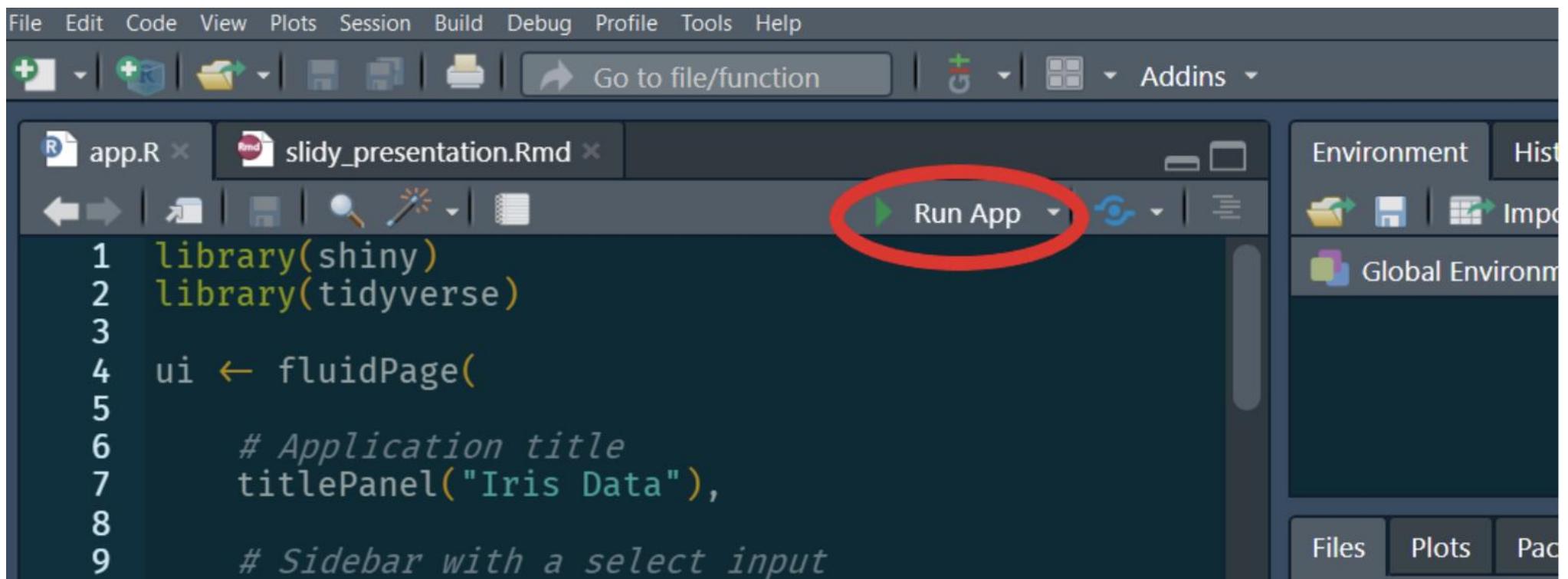
- A
- B
- C
- D
- E

type in a number

4

You selected the letter A and the number 4

How to run a shiny app?



Exercise 1

Create a Shiny application from scratch that looks (and functions) like the one below.

- There should be 2 textInput fields for entering user names, and one numericInput for entering height
- The message at the bottom should display the first and last names entered by the user, along with their height
- Add some radio buttons to allow users to select age brackets
- Change the height selector to a slider
- Feel free to add extra info fields

Enter first name

Enter last name

Enter approximate height (cm)

Hello Peter Rabbit
You are approximately 30cm tall.

Two inputs

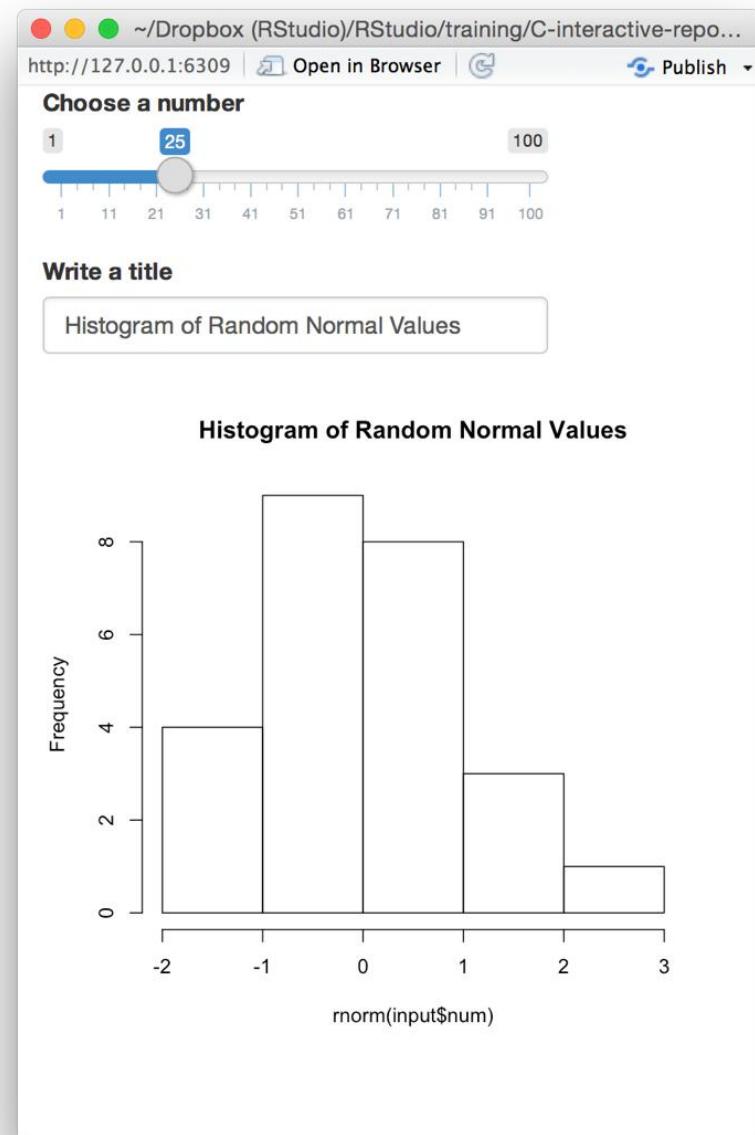
```
# 01-two-inputs

library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  textInput(inputId = "title",
    label = "Write a title",
    value = "Histogram of Random Normal Values"),
  plotOutput("hist")
)

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

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



Two outputs

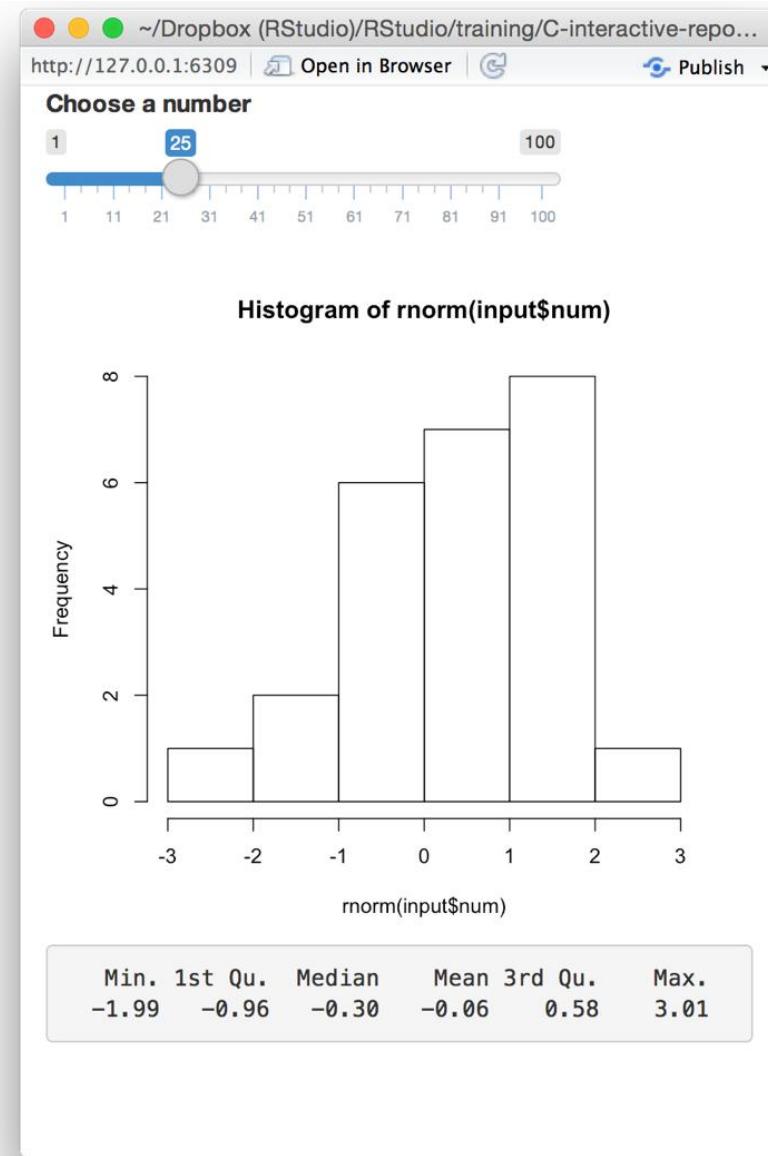
```
# 02- two- outputs

library(shiny)

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

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

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



Side bar layouts - syntax

```
sidebarLayout(  
  sidebarPanel,  
  mainPanel,  
  position = c("left", "right"),  
  fluid = TRUE  
)  
  
sidebarPanel(..., width = 4)  
mainPanel(..., width = 8)
```

Side bar layouts - example

```
ui <- fluidPage(  
  
  titlePanel("Exercise 1.1"),  
  sidebarLayout(  
    sidebarPanel(  
  
     textInput(inputId = "user_name",  
                label = "Enter name"),  
      numericInput(inputId = "number_chosen",  
                  label = NULL, value = 15),  
      sliderInput(inputId = "slider",  
                  label = "Choose another number",  
                  min = 10, max = 100, value = 50),  
    ),  
    mainPanel(  
  
      textOutput(outputId = "supplied_name"),  
      textOutput(outputId = "info"),  
      plotOutput(outputId = "density_plot", height = "300px", width = "400px")  
    )  
)
```

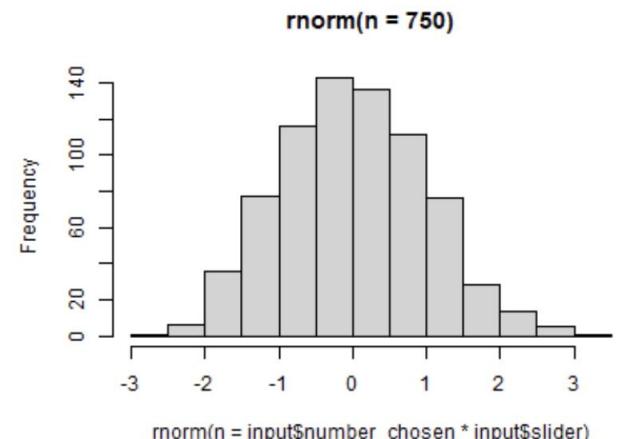
Enter name

15

Choose another number



Hello
Your two numbers are 15 and 50



Layout examples

The 'objects' referred to within the layouts could be single elements such as a plot, table, image or piece of text, or a panel containing multiple elements. Panels can be nested within panels.

[sidebarLayout\(\)](#)[fluidRow\(\)](#)[flowLayout](#)[splitLayout](#)[verticalLayout](#)[panels](#)[tabPanels](#)

sidebarLayout()

Contains a sidebarPanel and a mainPanel

sidebarPanel()

The sidebar panel generally contains input fields

Enter text here

Choose file

Browse...

No file sel

mainPanel()

This is a classic simple layout that is often used to keep user input components in the sidebar separate from the outputs in the mainPanel, though there are no restrictions on the placement of elements. Inputs can be located in the main panel and outputs in the sidebar, or some in one, and some in the other.

Single elements can be added one by one in the main panel or they could be nested inside other panels

The width of the sidebar can be adjusted by setting the width argument, by default it is 4.

The widths of the sidebarPanel and mainPanel should add up to 12, as in the fluidRow examples in the next tab.

Importing files – fileInput()

Unlike other inputs, fileInput() returns a data frame with 4 columns

```
ui <- fluidPage(  
  
  titlePanel("Uploading files"),  
  fileInput(inputId = "upload",  
            label = "Upload..."),  
  tableOutput("files")  
)  
server <- function(input, output) {  
  
  output$files <- renderTable(input$upload)  
}
```

Uploading files

Upload...

Browse... adsl.csv
Upload complete

| name | size | type | datapath |
|----------|-------|--------------------------|--|
| adsl.csv | 79519 | application/vnd.ms-excel | C:\Users\biggins\AppData\Local\Temp\RtmpDzJGd\file1f333e03adsl.csv |

default max file size is 5MB

```
# allow file up to 10MB to be uploaded  
options(shiny.maxRequestSize = 10 * 1024^2)
```

Importing files – fileInput()

```
library(shiny)
library(readr)

options(shiny.maxRequestSize = 10 * 1024^2)

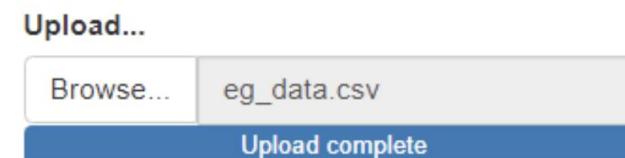
ui <- fluidPage(
  titlePanel("Uploading files"),
  fileInput(inputId = "upload",
            label = "Upload..."),
  tableOutput("files")
)

server <- function(input, output) {

  output$files <- renderTable({
    req(input$upload)
    dataset <- read_csv(input$upload$datapath)
    head(dataset)
  })
}

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

Uploading files



| Uniprot id | Gene | Protein names | Gene names | Pep Count | log2_fc
naive primed |
|------------|-------|---|------------------------------------|-----------|-------------------------|
| O00574 | CXCR6 | C-X-C chemokine receptor type 6 (CXC-R6)
(CXCR-6)
(CDw186) (G-protein coupled receptor STRL33) (G-protein coupled receptor bonzo)
(CD antigen CD186) | CXCR6
BONZO
STRL33
TYMSTR | 2.00 | 5.72 |

Importing files – fileInput()

fileInput() summary

fileInput() returns a data frame of 4 columns

```
input$id$datapath  
options(shiny.maxRequestSize = 10 * 1024^2)  
req(input$id)
```

Access the filepath.

Increase max upload size (to 10MB).

Check a file has been chosen.

File validity

```
fileInput(inputId, label,  
         accept = ".csv")  
  
req(input$file)  
ext <- tools::file_ext(input$file$name)  
validate(need(ext == "csv",  
            "Please upload a csv file"))  
read_csv(input$file$datapath)
```

Use the accept parameter in fileInput() to limit files shown in browser.

The accept argument is only a suggestion to the file browser.

Use further checks to validate file type.

Pre-processing datasets

Data can be imported, manipulated if necessary and saved as R data objects (.Rds or .RData). This can be done in a pre-processing script to allow the data to simply be loaded within the Shiny app.

Saving and loading one file

```
# in pre-processing script
tree_data <- read_csv("tree_measurements.csv")
saveRDS(tree_data, file = "data/tree_data.Rds")

# in app.R script
tree_data <- readRDS("data/tree_data.Rds")
```

Saving a named list

```
tree_list <- list(
  "beech" = beech_trees,
  "oak" = oak_trees)
saveRDS(tree_list, file = "data/tree_list.Rds")

# in app.R script
tree_data <- readRDS("data/tree_list.Rdata")
tree_data$beech # to access the beech dataset
tree_data$oak # to access the oak dataset
```

Saving more than one dataset in .Rdata

```
# in pre-processing script
beech_trees <- read_csv("beech_tree_measurements.csv")
oak_trees <- read_csv("oak_tree_measurements.csv")
save(beech_trees, oak_trees, file = "data/tree_data.Rdata")

# in app.R script
load("data/tree_data.Rdata") # loads the 2 datasets beech_trees and oak_trees
```

Exercise 2

Uploading a file.

Create an app that looks like this one. It should:

- allow a csv file to be chosen and uploaded
- display the first few rows of data in a table
- have an option to change the number of rows displayed

Uploading a file

select file

Browse... eg_data2.csv

Upload complete

select number of rows to display

7

| Uniprot id | Gene | Gene names | Pep Count | log2_fc naive primed | q-value |
|------------|--------|------------------------------------|-----------|----------------------|---------|
| O00574 | CXCR6 | CXCR6
BONZO
STRL33
TYMSTR | 2.00 | 5.72 | 0.00 |
| P19397 | CD53 | CD53
MOX44
TSPAN25 | 3.00 | 5.41 | 0.00 |
| P09564 | CD7 | CD7 | 2.00 | 5.37 | 0.00 |
| P05187 | ALPP | ALPP
PLAP | 13.00 | 5.22 | 0.00 |
| Q07075 | ENPEP | ENPEP | 40.00 | 5.15 | 0.00 |
| P22732 | SLC2A5 | SLC2A5
GLUT5 | 9.00 | 5.08 | 0.00 |
| P15509 | CSF2RA | CSF2RA
CSF2R
CSF2RY | 14.00 | 5.03 | 0.00 |

Reactivity

reactive()

Builds a reactive object (reactive expression)

```
data <- reactive( { rnorm(input$num) })
```

object will respond to *every reactive value in the code*

code used to build (and rebuild) object

Two outputs with reactive()

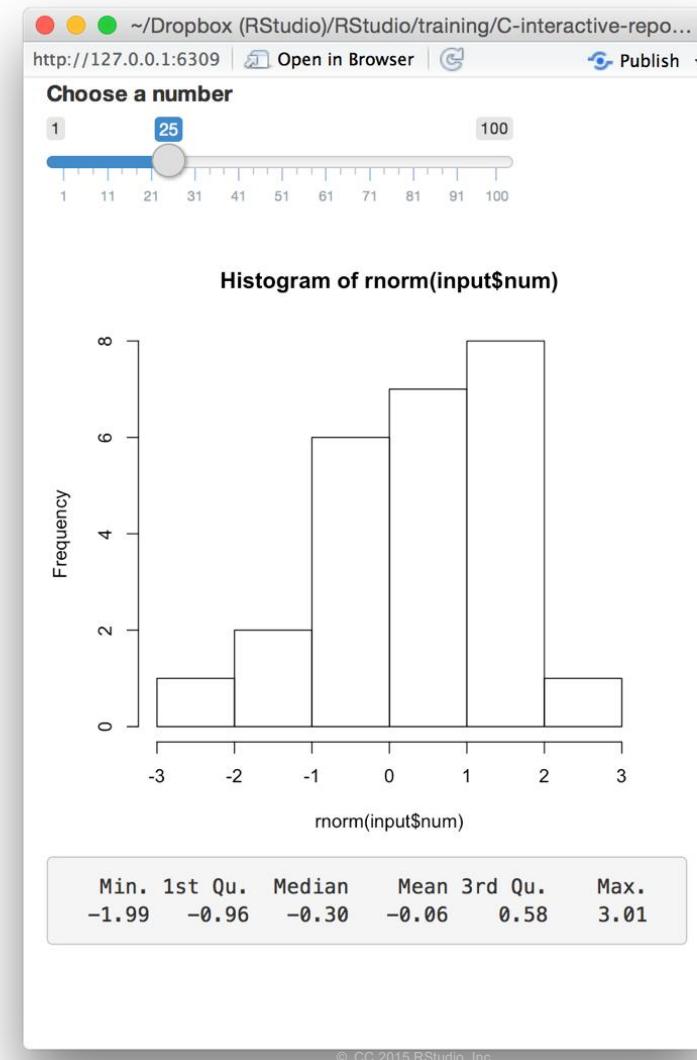
```
# 03-reactive

library(shiny)

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

server <- function(input, output) {
  data <- reactive({
    rnorm(input$num)
  })
  output$hist <- renderPlot({
    hist(data())
  })
  output$stats <- renderPrint({
    summary(data())
  })
}

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



Can we prevent the title field from updating the plot?

isolate()

Returns the result as a non-reactive value

```
isolate( { rnorm(input$num) } )
```

object will NOT respond to
any reactive value in the code

code used to build object

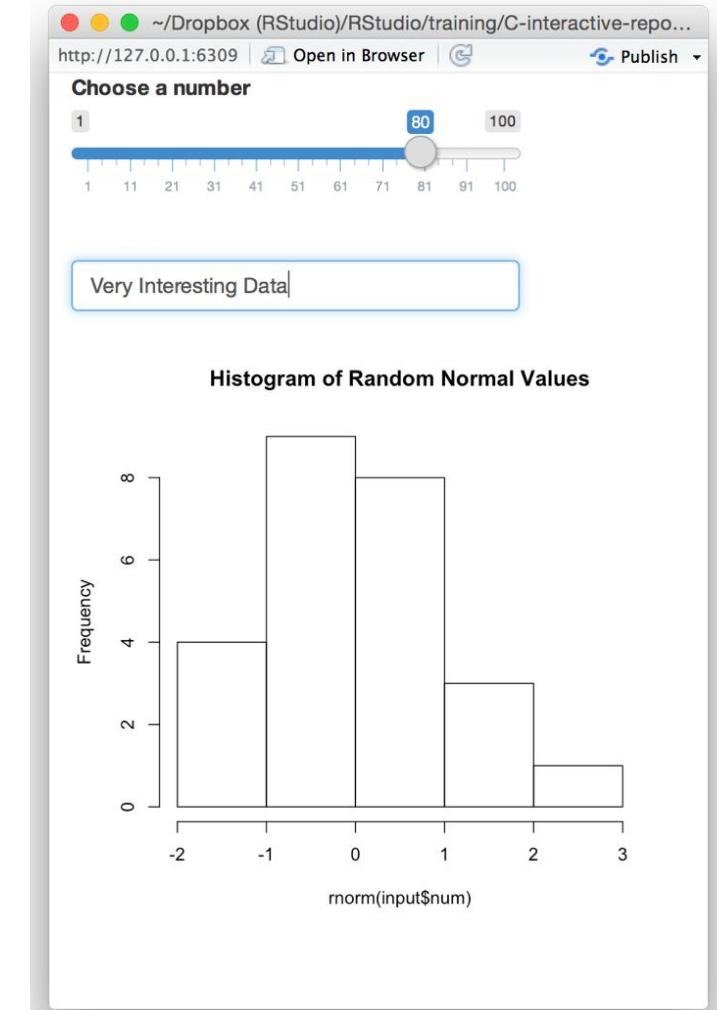
Can we prevent the title field from updating the plot?

```
# 04-isolate
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  textInput(inputId = "title",
    label = "Write a title",
    value = "Histogram of Random Normal Values"),
  plotOutput("hist")
)

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

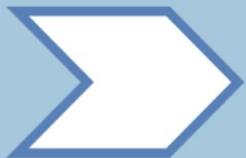
shinyApp(ui = ui, server = server)
```



Reactive expressions and observers

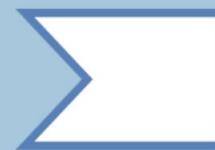
Both store expressions that can be executed.

Reactive expression



Returns values
Callable
not side effects
Lazy
Caches values

Observer

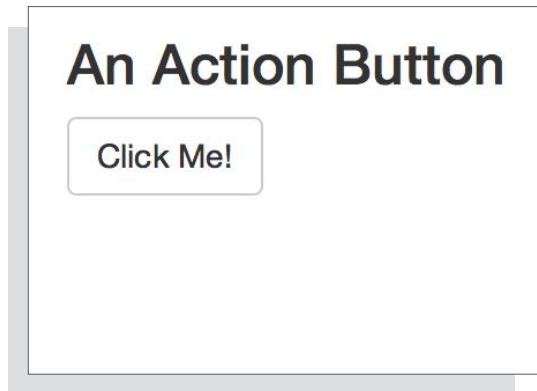


Doesn't return values
Not callable
Side effects
Eager
N/A

Reactive expressions need an observer as a descendent in order to execute.

`reactive()` – calculating values without side effects
`observe()` – performing actions with side effects

Action buttons



input name
(for internal use)

label to
display

```
actionButton(inputId = "go", label = "Click Me!")
```

Notice:
Id not ID

observeEvent()

Triggers code to run on server

```
observeEvent ( input $clicks, { print (input $clicks) } )
```

reactive value(s) to respond to

(observer invalidates ONLY when this value changes)

code block to run whenever observer is invalidated

note: observer treats this code as if it has been isolated with isolate()

Action buttons

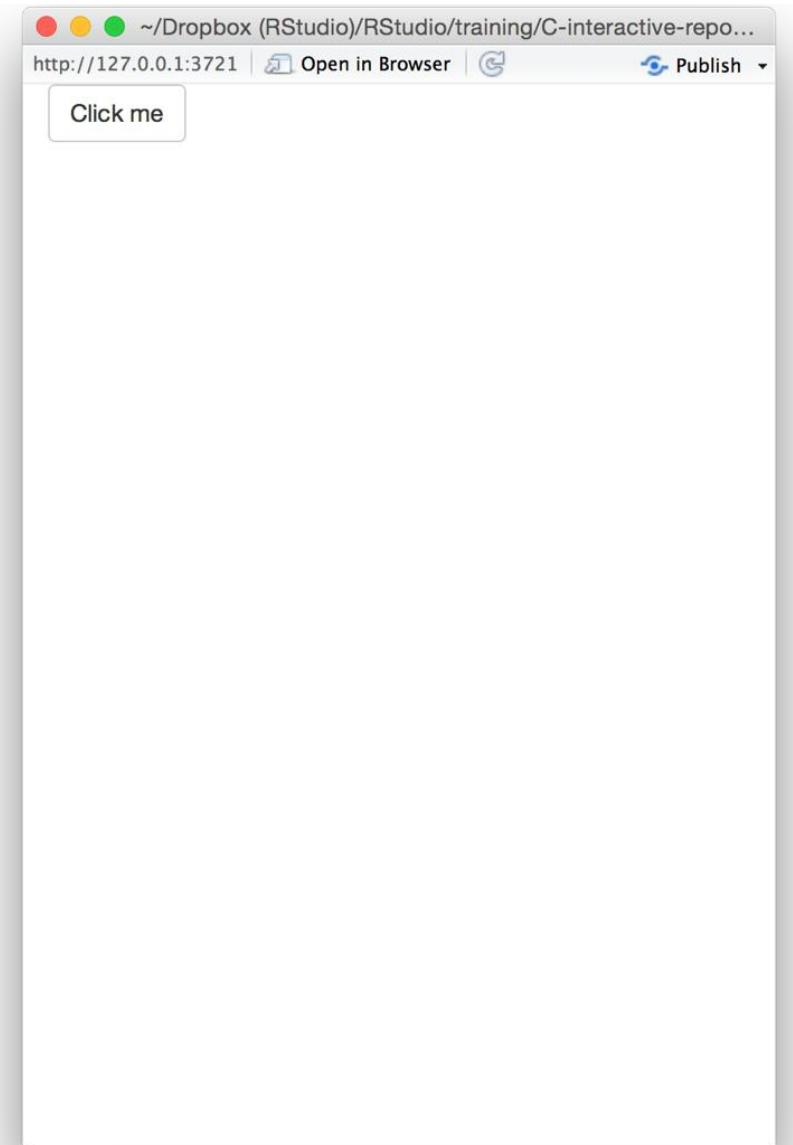
```
# 05- actionButton

library(shiny)

ui <- fluidPage(
  actionButton(inputId = "clicks",
  label = "Click me")
)

server <- function(input, output) {
  observeEvent(input$clicks, {
    print(as.numeric(input$clicks))
  })
}

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



reactiveValues()

Creates a list of reactive values to manipulate programmatically

```
rv <- reactiveValues(data = rnorm(100))
```

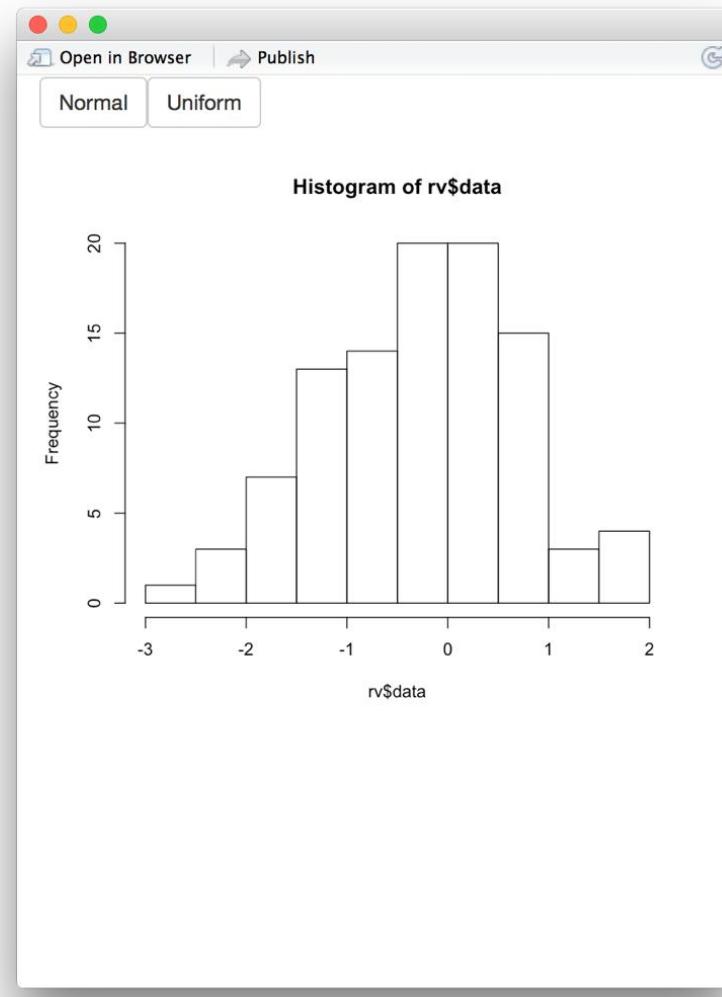
(optional) elements
to add to the list

```
# 08-reactiveValues
```

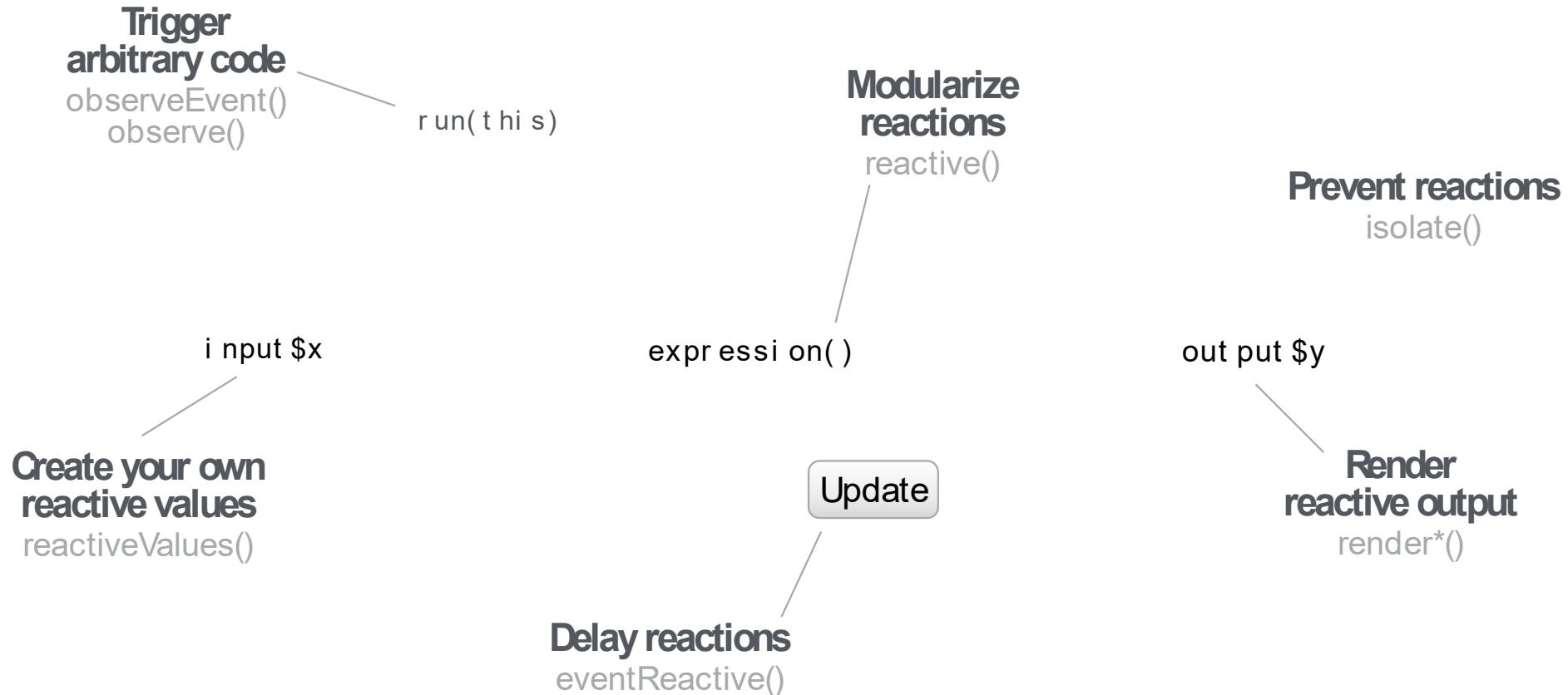
```
library(shiny)
```

```
ui <- fluidPage(  
  actionButton(inputId = "norm", label = "Normal"),  
  actionButton(inputId = "uniform", label = "Uniform"),  
  plotOutput("hist"))  
  
server <- function(input, output) {  
  
  rv <- reactiveValues(data = rnorm(100))  
  
  observeEvent(input$norm, { rv$data <- rnorm(100) })  
  observeEvent(input$uniform, { rv$data <- runif(100) })  
  
  output$hist <- renderPlot({  
    hist(rv$data)  
  })  
}
```

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



Summary



Exercise 3

Create an app that looks like this one (minus these instructions), using the sidebarLayout and the iris dataset

1. In the sidebarPanel, create radio buttons to select between sepal length, sepal width, petal length and petal width
2. In the main panel, create a boxplot showing the attribute selected from the radio buttons for the 3 different species.
3. Add a textInput to allow the user to name the plot
4. Show the iris data in table format below the plot
5. In the sidebar, allow the user to select the number of rows to be shown

[Optional]

1. Move the plot and table in to separate tabs
2. Add another tab with a table showing the mean and standard deviation for the selected attribute

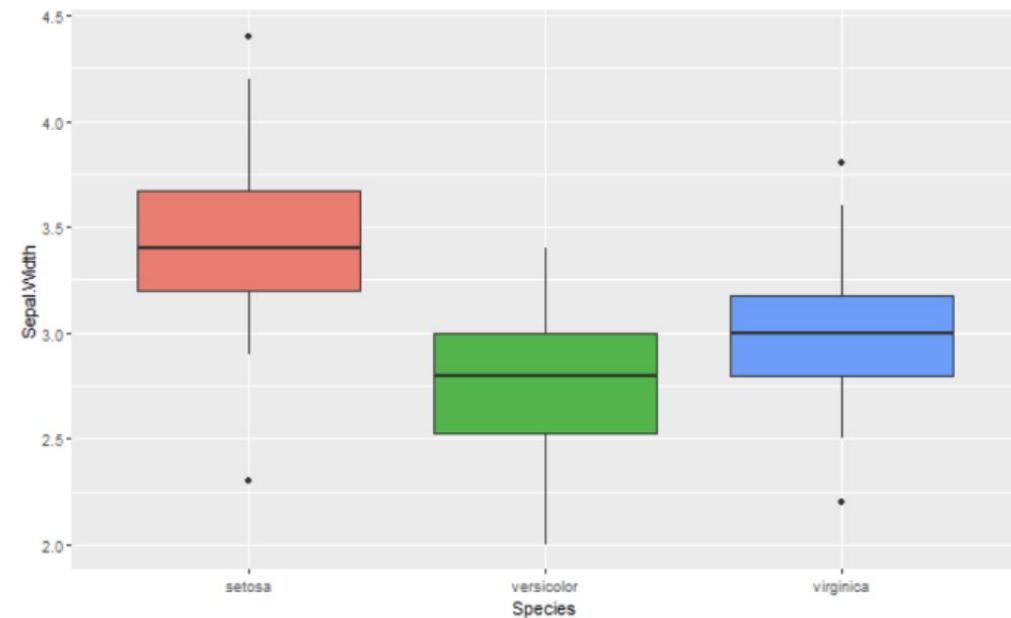
The app

Select attribute to plot

Sepal.Length
 Sepal.Width
 Petal.Length
 Petal.Width

Enter plot name

Number of rows in table



| Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
|--------------|-------------|--------------|-------------|---------|
| 5.10 | 3.50 | 1.40 | 0.20 | setosa |
| 4.90 | 3.00 | 1.40 | 0.20 | setosa |

Exercise 4

Create an app that looks like this one, it should:

- Import the advs.csv file.
- Provide drop down lists of variables to plot on x and y axes.
- Filter data based on the selected subject and parameter.

