

# R markdown and Shiny

“Processing large dataset with R”

# R markdown

- Rmarkdown is a comprehensive way of creating reports which involve R.
- R markdown is a format for *literate programming* documents. It is based on *markdown*, a markup language that is widely used to generate html pages.
- Literate programming weaves instructions, documentation and detailed comments in between machine executable code, producing a document that describes the program.
- Unlike a word processor, such as Microsoft Word, where what you see is what you get, with R markdown, you need to *compile* the document into the final report.

# How it works



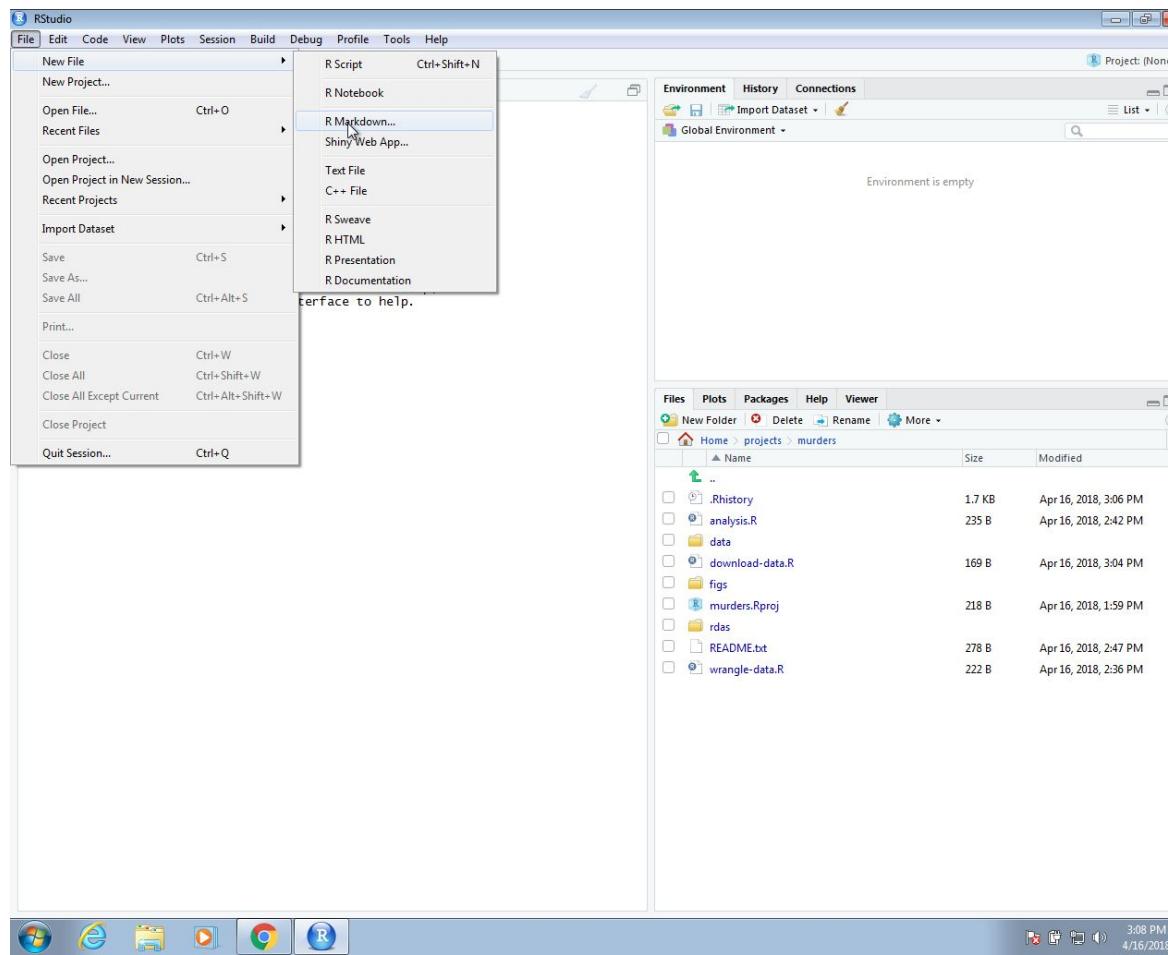
When you run `render`, R Markdown feeds the .Rmd file to [knitr](#), which executes all of the code chunks and creates a new markdown (.md) document which includes the code and it's output.

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.

# Let's start

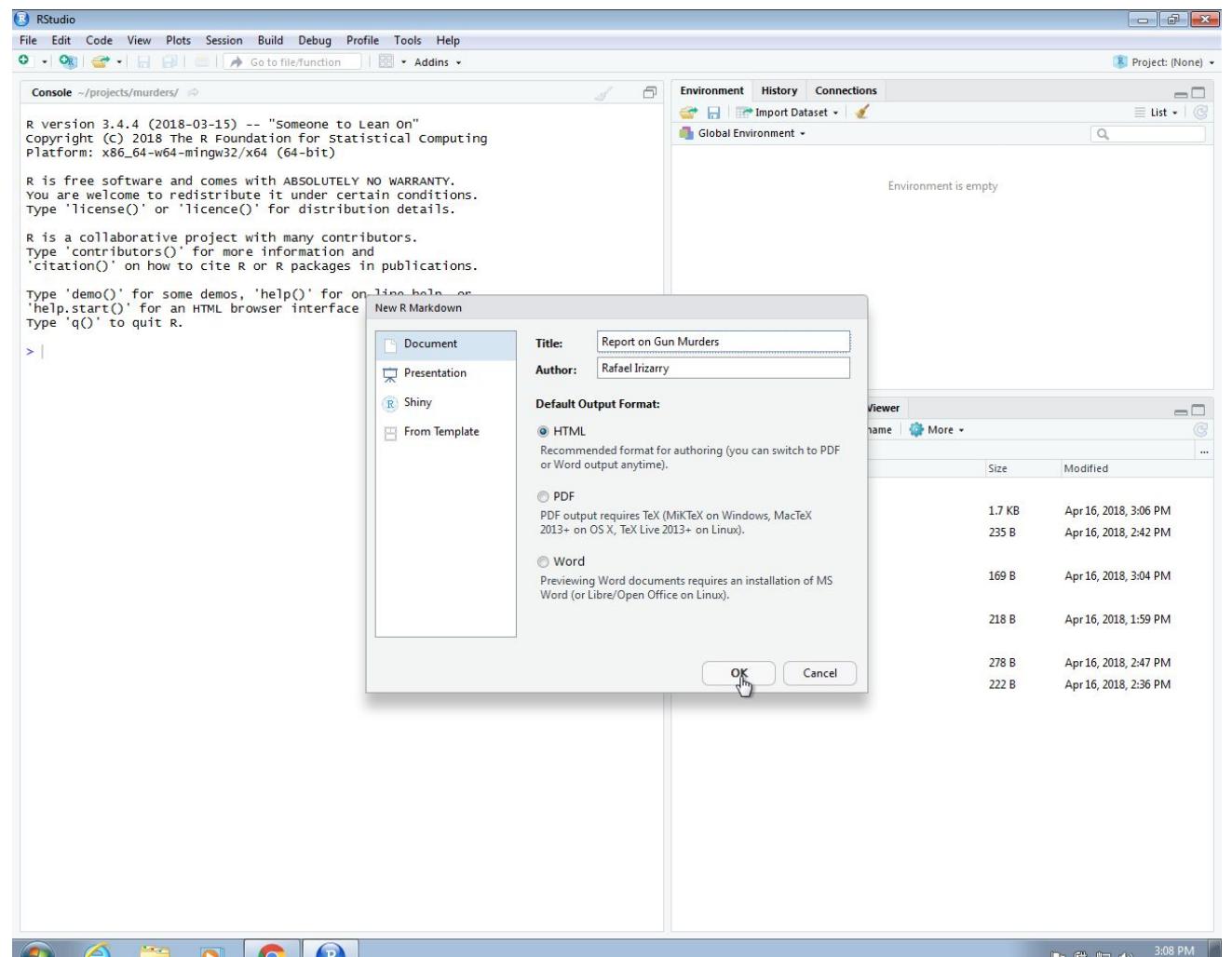
In RStudio, you can start an R markdown document by clicking on *File*, *New File*, the *R Markdown*, like this:



# Project initialization

You will then be asked to enter a title and author for your document. Here we hypothesized to prepare a report on gun murders so we give it an appropriate name. You can also decide what format you would like the final report to be in: HTML, PDF, or Microsoft Word.

Later, we can easily change this, but here we select html as it is the preferred format for debugging purposes:



# Project initialization

This will generate a template file:

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:** Untitled1 x, showing R Markdown code. The code includes a header for a report on gun murders, setup instructions for knitr, and examples of R code chunks and plots.
- Console:** Shows the R version 3.4.4 startup message and the command to run the R Markdown file.
- Environment:** Global Environment tab, showing an empty environment.
- Files:** A file browser showing the contents of a 'murders' project folder. The folder contains .Rhistory, analysis.R, data, download-data.R, figs, murders.Rproj, rdas, README.txt, and wrangle-data.R files.
- Bottom Bar:** Taskbar with icons for Windows, Internet Explorer, File Explorer, Media Player, Google Chrome, and RStudio. System tray showing battery, volume, and date/time (3:08 PM, 4/16/2018).

# The header

At the top you see:

```
---
```

```
title: "Report on Gun Murders"
author: "Rafael Irizarry"
date: "April 16, 2018"
output: html_document
```

```
--
```

The things between the --- is the header. We actually don't need a header, but it is often useful. You can define many other things in the header than what is included in the template.

The one parameter that we will highlight is `output`. By changing this to, say, `pdf_document`, we can control the type of output that is produced when we compile.

# R code chunks

In various places in the document, we see something like this:

```
```{r}
summary(pressure)
```
```

These are the code **chunks**. When you compile the document, the R code inside the chunk, in this case `summary(pressure)`, will be evaluated and **the result included** in that position in the final document. To add your own R chunks, you can type the characters above quickly with the key binding **command-option-i** on the Mac and **Ctrl-Alt-i** on Windows.

We recommend getting into the habit of **adding a label** to the R code chunks. This will be very useful when debugging, among other situations. You do this by adding a descriptive word like this:

```
```{r - summary pressure}
summary(pressure)
```
```

# Chunk Options

- include = FALSE prevents **code and results** from appearing in the finished file. R Markdown still runs the code in the chunk, and the results can be used by other chunks.
- echo = FALSE prevents **code, but not the results** from appearing in the finished file. This is a useful way to embed figures.
- the eval option disables the **execution** of the code.
- message = FALSE prevents **messages** that are generated by code from appearing in the finished file.
- warning = FALSE prevents **warnings** that are generated by code from appearing in the finished.
- fig.cap = "..." adds a **caption** to graphical results.

See the [R Markdown Reference Guide](#)  for a complete list of knitr chunk options

# Code Languages

.Rmd file executes code in bash and python.

The screenshot shows an RStudio interface with the following details:

- Left Panel (Code Editor):** Displays the R Markdown file `4-languages.Rmd` with the following content:

```
1 ---  
2 title: "Simple Language Demos"  
3 output: html_document  
4 ---  
5  
6 You can write code in languages other than R with R Markdown, e.g.  
7  
8 ## Bash  
9  
10 ```{bash}  
11 ls *.Rmd  
12 ```  
13  
14 ## Python  
15  
16 ```{python}  
17 x = 'hello, python world!'  
18 print(x.split(' '))  
19 ```  
20  
21
```
- Right Panel (Viewer):** Shows the generated HTML output:
  - Section: Simple Language Demos**

You can write code in languages other than R with R Markdown, e.g.
  - Bash**

```
ls *.Rmd  
## 1-example.Rmd  
## 2-chunks.Rmd  
## 3-inline.Rmd  
## 4-languages.Rmd
```
  - Python**

```
x = 'hello, python world!'  
print(x.split(' '))  
## ['hello,', 'python', 'world!']
```

- Python
- SQL
- Bash
- Rcpp
- Stan
- JavaScript
- CSS

# Tables

By default, R Markdown displays data frames and matrixes as they would be in the R terminal (in a monospaced font). If you prefer that data be displayed with additional formatting you can use the `knitr::kable` function

The screenshot shows the RStudio interface with two panes. The left pane displays an R Markdown file named "6-tables.Rmd". The code includes a YAML front matter section specifying a title and output type, followed by a section about various packages for creating tables and an example of using the knitr::kable function.

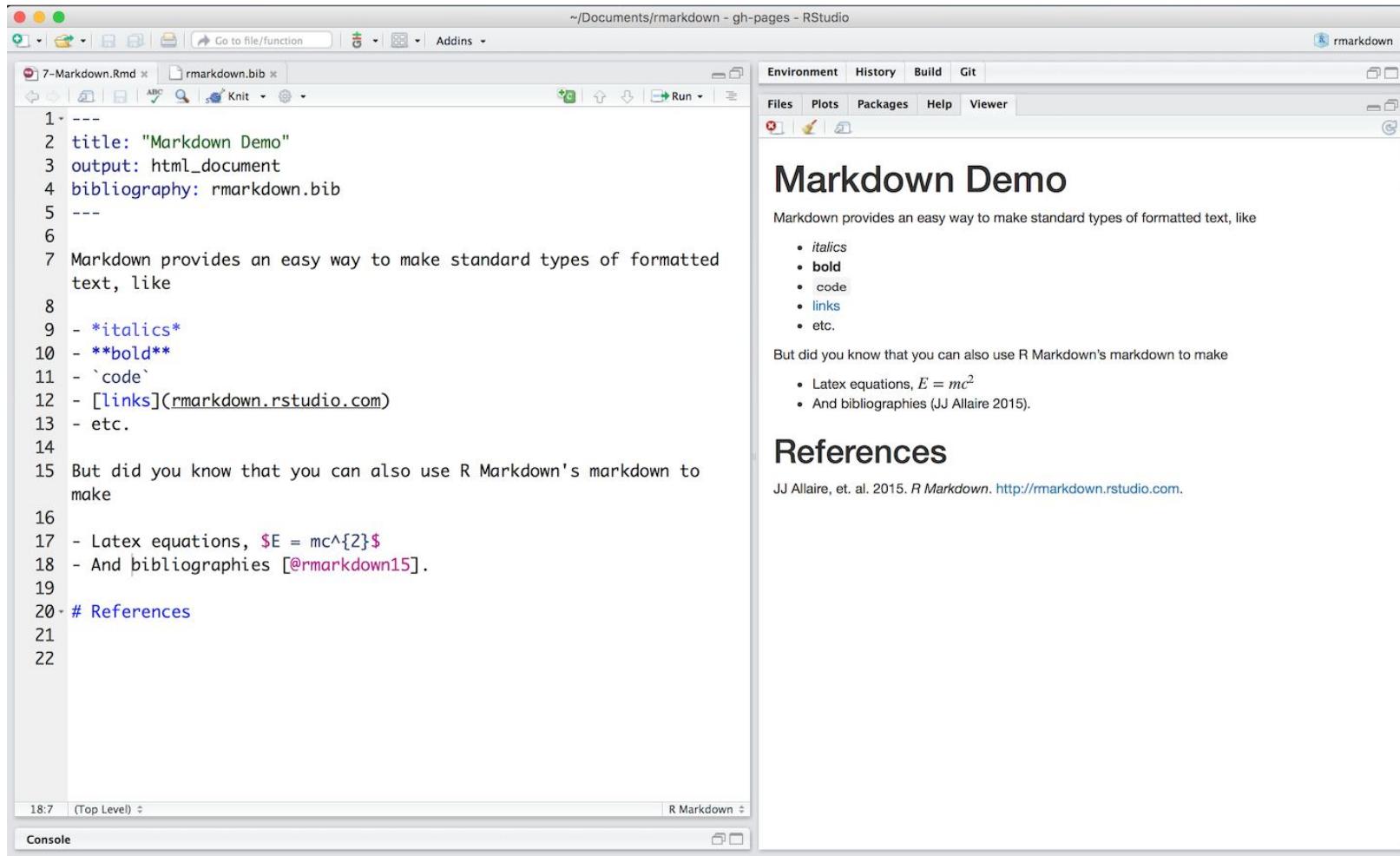
```
1---  
2 title: "Table options"  
3 output: html_document  
4---  
5  
6 Several packages support making beautiful tables with R, such as  
7  
8 * [xtable](https://cran.r-project.org/web/packages/xtable/)  
9 * [stargazer](https://cran.r-project.org/web/packages/stargazer/)  
10 * [pander](http://rapporter.github.io/pander/)  
11 * [tables](https://cran.r-project.org/web/packages/tables/)  
12 * [ascii](http://eusebe.github.io/ascii/)  
13 * etc.  
14  
15 It is also very easy to make tables with knitr's `kable` function:  
16  
17```{r echo = FALSE, results = 'asis'}  
18 library(knitr)  
19 kable(mtcars[1:5, ], caption = "A knitr kable.")  
20```  
21
```

The right pane shows the generated HTML output titled "Table options". It lists several packages for creating tables and shows a table generated by the knitr::kable function. The table has columns for mpg, cyl, disp, hp, drat, wt, qsec, vs, am, gear, and carb, with data for five cars from the mtcars dataset.

|                   | mpg  | cyl | disp | hp  | drat | wt    | qsec  | vs | am | gear | carb |
|-------------------|------|-----|------|-----|------|-------|-------|----|----|------|------|
| Mazda RX4         | 21.0 | 6   | 160  | 110 | 3.90 | 2.620 | 16.46 | 0  | 1  | 4    | 4    |
| Mazda RX4 Wag     | 21.0 | 6   | 160  | 110 | 3.90 | 2.875 | 17.02 | 0  | 1  | 4    | 4    |
| Datsun 710        | 22.8 | 4   | 108  | 93  | 3.85 | 2.320 | 18.61 | 1  | 1  | 4    | 1    |
| Hornet 4 Drive    | 21.4 | 6   | 258  | 110 | 3.08 | 3.215 | 19.44 | 1  | 0  | 3    | 1    |
| Hornet Sportabout | 18.7 | 8   | 360  | 175 | 3.15 | 3.440 | 17.02 | 0  | 0  | 3    | 2    |

# Markdown Basics

Format the text in your R Markdown file with [Pandoc's Markdown](#), a set of markup annotations for plain text files. When you render your file, Pandoc transforms the marked up text into formatted text in your final file format



The screenshot shows the RStudio interface. On the left, the code editor displays an R Markdown file named "7-Markdown.Rmd". The code includes YAML front matter and various Markdown syntax examples. On the right, the preview pane shows the rendered output. The title "Markdown Demo" is displayed, followed by a paragraph about Markdown's ease of use for formatted text. It lists several features: italics, bold, code, links, and etc. Below this, it notes that R Markdown can also produce LaTeX equations and bibliographies. The "References" section is shown, listing "JJ Allaire, et. al. 2015. *R Markdown*. <http://rmarkdown.rstudio.com>". The bottom status bar indicates "18:7 (Top Level)" and "R Markdown".

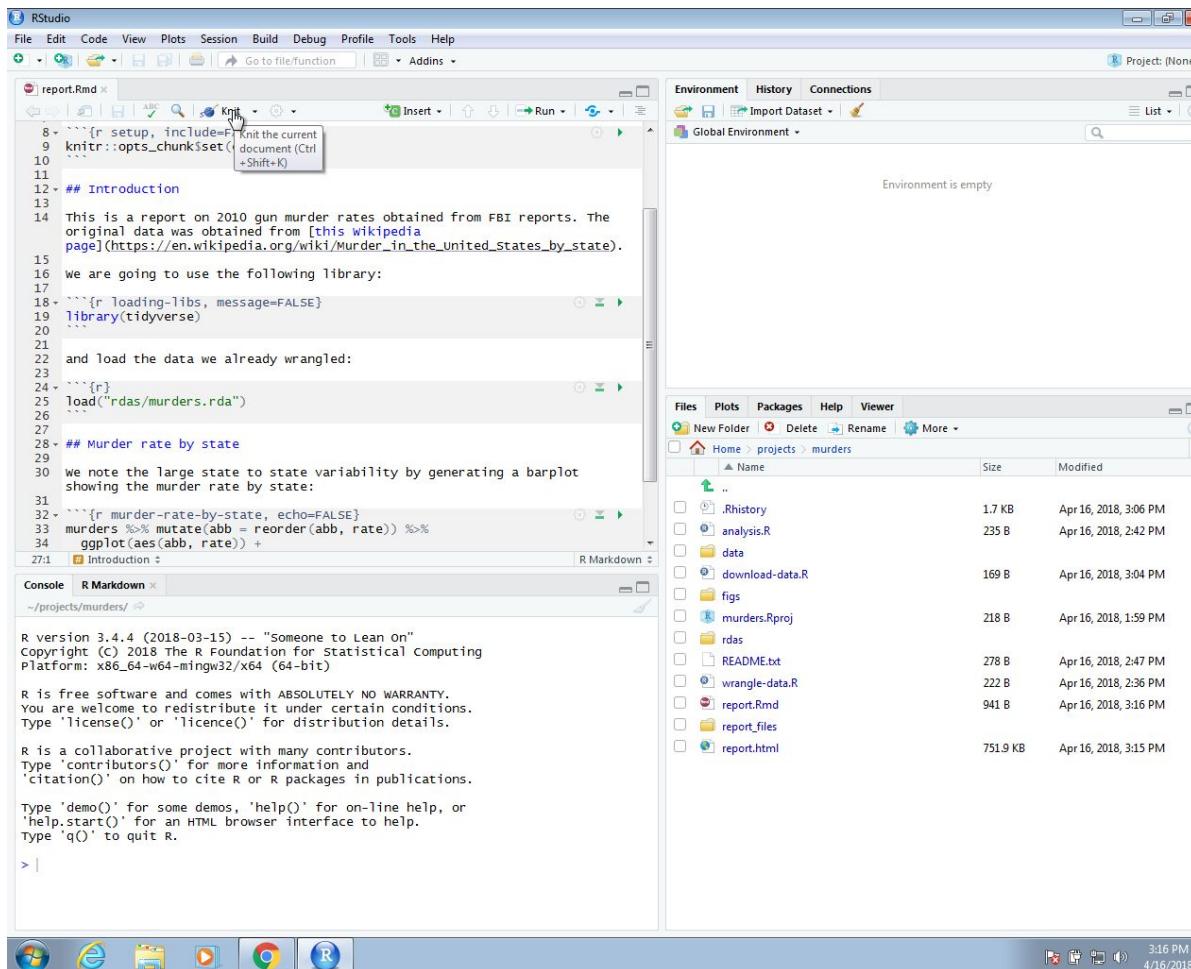
- Headers
- Lists
- Links
- Images
- Block quotes
- Latex equations
- Horizontal rules
- Tables
- Footnotes
- Bibliographies and Citations
- Slide breaks
- Italicized text
- Bold text
- Superscripts
- Subscripts
- Strikethrough text

# Markdown Basics

- the # allows to organize the document in sections, subsections, ... The more you add #, the more you go down into the subsections, subsubsections, ...
- the \* allows to highlight some part of the text: if you use one \*tyty\* it produces *tyty*. If you use \*\*tyty\*\*, the text **tyty** will be in bold.
- the > allows to create some “citation / remark box”: > The citation that I would like to highlight!
- It is possible to provide online code using single quote. As an exemple: ls().

# 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](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}
```

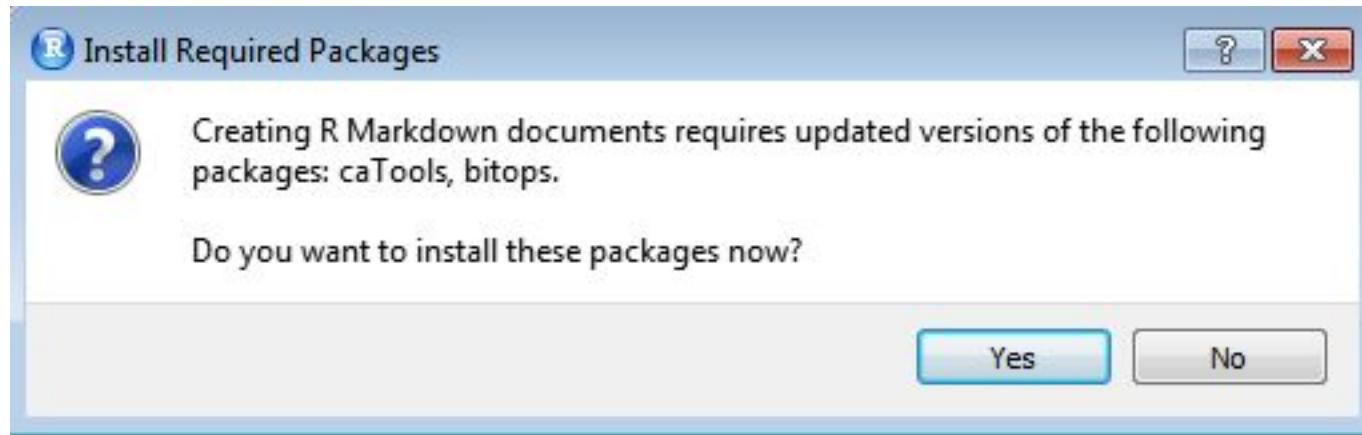
```
murders %>% mutate(abb = reorder(abb, rate)) %>%
```

```
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:



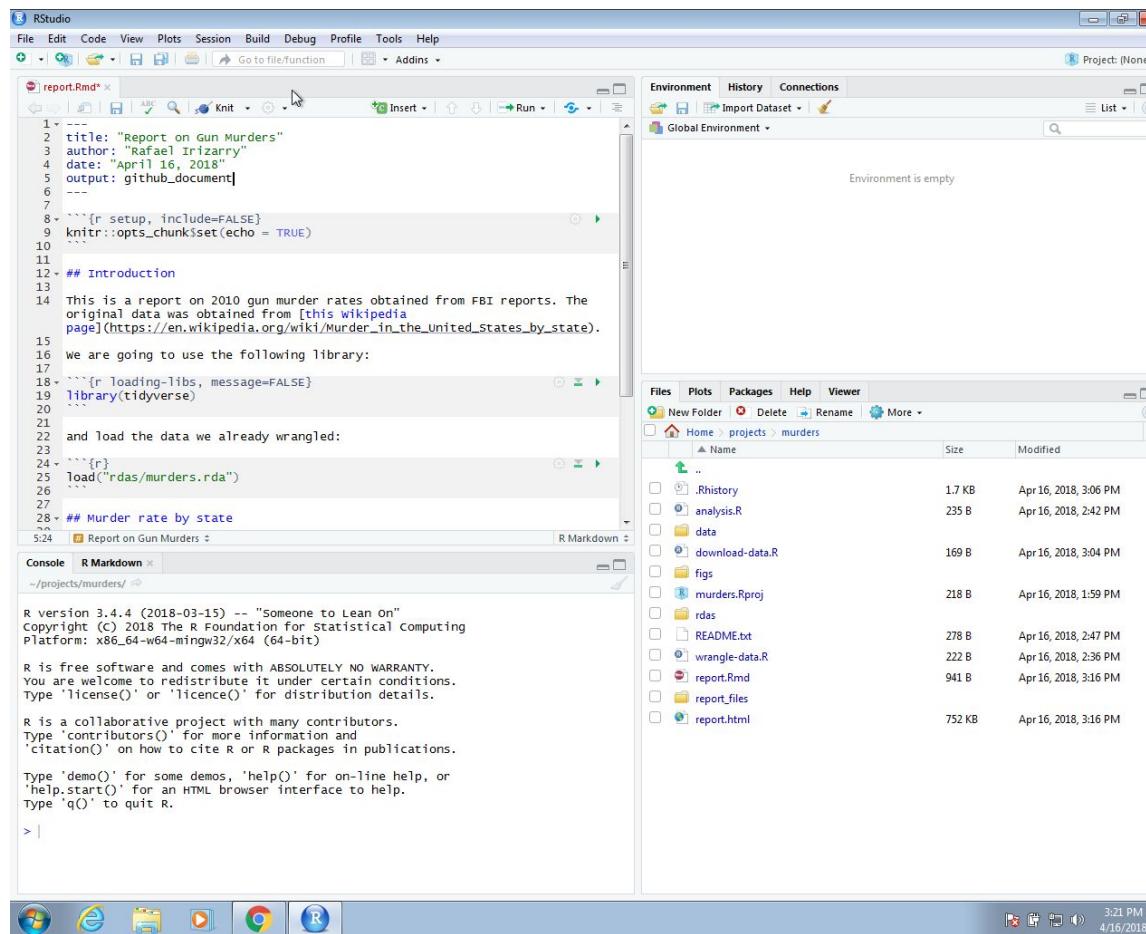
Once you have installed the packages, clicking the *Knit* will **compile** your R markdown file and the resulting document will pop-up:

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 an R Markdown file open. The code editor contains the following R Markdown code:

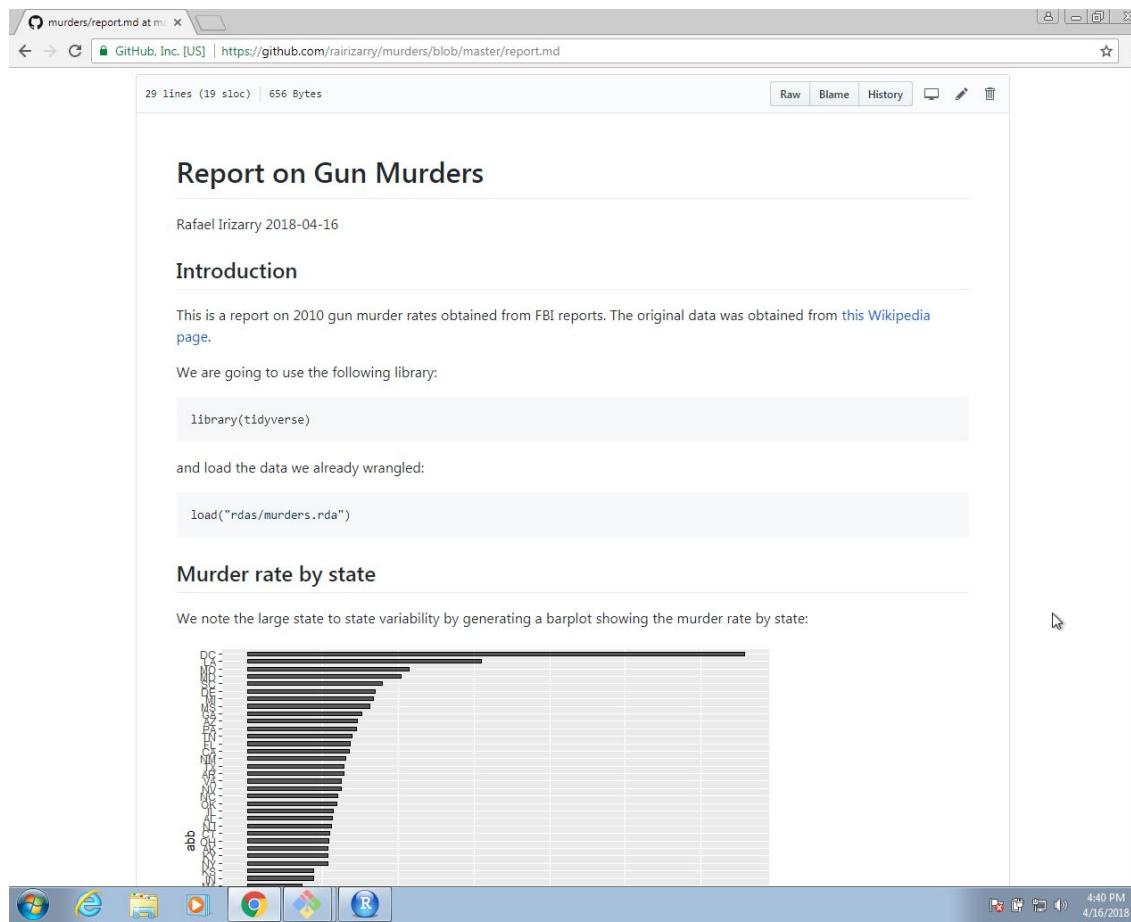
```
1 ---  
2 title: "Report on Gun Murders"  
3 author: "Rafael Irizarry"  
4 date: "April 16, 2018"  
5 output: github_document  
---  
8 ``{r setup, include=FALSE}  
9 knitr::opts_chunk$set(echo = TRUE)  
10  
11  
12 ## Introduction  
13  
14 This is a report on 2010 gun murder rates obtained from FBI reports. The  
15 original data was obtained from [this wikipedia  
16 page](https://en.wikipedia.org/wiki/Murder_in_the_United_States_by_state).  
17  
18 ``{r loading-libs, message=FALSE}  
19 library(tidyverse)  
20  
21 and Load the data we already wrangled:  
22  
23 ``{r}  
24 load("rdas/murders.rda")  
25  
26  
27  
28 ## Murder rate by state  
5:24 Report on Gun Murders
```

The R Markdown tab in the bottom-left corner is selected. The bottom-right panel shows the project file structure:

Name	Size	Modified
..		
.Rhistory	1.7 KB	Apr 16, 2018, 3:06 PM
analysis.R	235 B	Apr 16, 2018, 2:42 PM
data	169 B	Apr 16, 2018, 3:04 PM
download-data.R		
figs		
murders.Rproj	218 B	Apr 16, 2018, 1:59 PM
rdas	278 B	Apr 16, 2018, 2:47 PM
README.txt	278 B	Apr 16, 2018, 2:47 PM
wrangle-data.R	222 B	Apr 16, 2018, 2:36 PM
report.Rmd	941 B	Apr 16, 2018, 3:16 PM
report_files		
report.html	752 KB	Apr 16, 2018, 3:16 PM

# 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 window with the title bar "test.pptx - PowerPoint". The ribbon menu is visible with tabs FILE, HOME, INSERT, DESIGN, TRANSITIONS, ANIMATIONS, SLIDE SHOW, REVIEW, and VIEW. The HOME tab is selected. The left sidebar shows a list of slides numbered 1 to 8. Slide 3 is currently selected and has a red border around its thumbnail. The main content area displays the slide content:

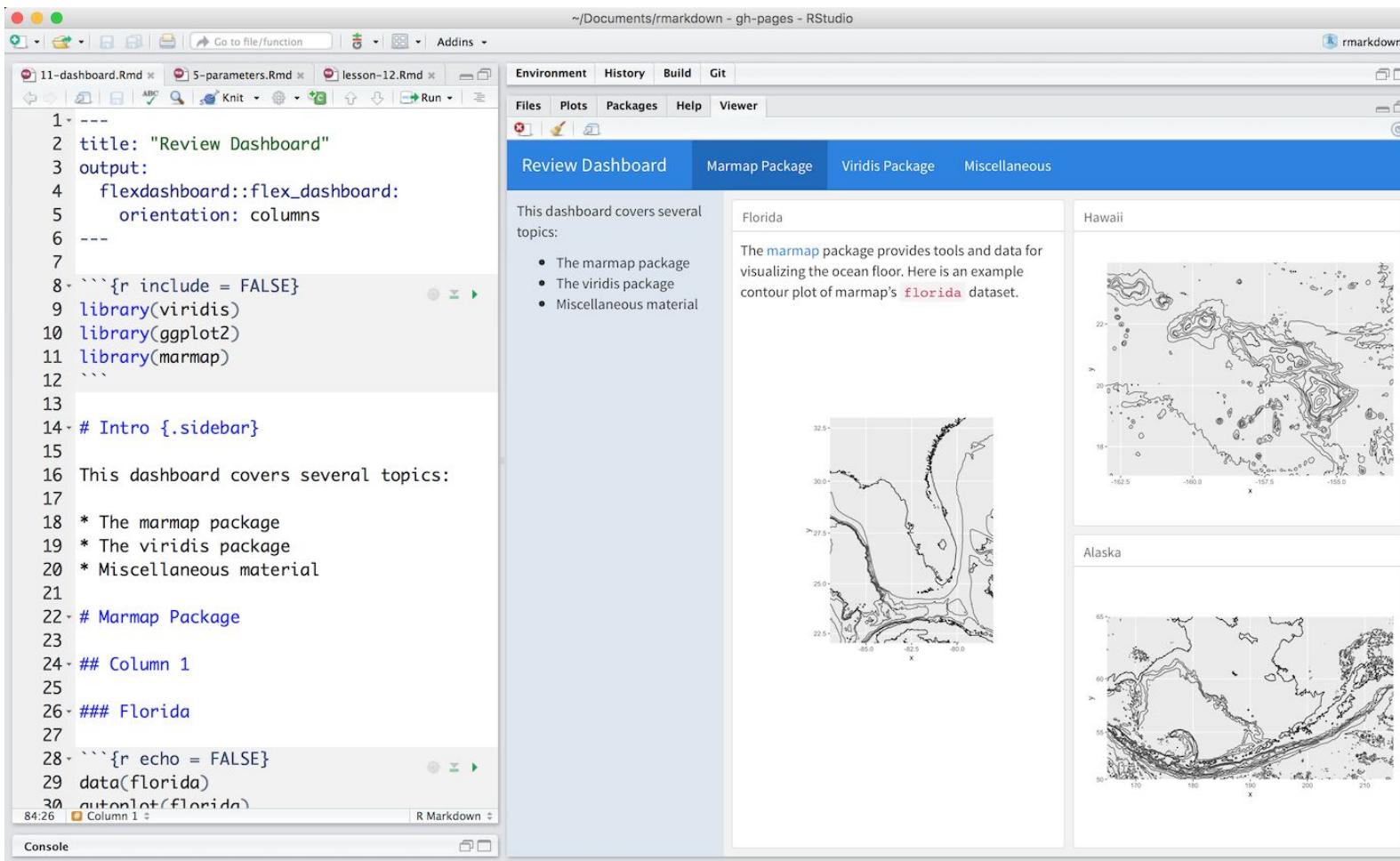
**Getting up**

- 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 indicates "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 an RStudio interface with an R Markdown file open on the left and its corresponding flexdashboard output on the right.

**Code (R Markdown):**

```
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)
```

**Output (flexdashboard):**

- Header:** Review Dashboard, Marmap Package, Viridis Package, Miscellaneous
- Section:** Florida
- Description:** The `marmap` package provides tools and data for visualizing the ocean floor. Here is an example contour plot of `marmap`'s `florida` dataset.
- Figure:** A contour plot of the Florida coastline and surrounding ocean floor.
- Section:** Hawaii
- Figure:** A contour plot of the Hawaiian Islands and surrounding ocean floor.
- Section:** Alaska
- Figure:** A contour plot of the Alaskan coastline and surrounding ocean floor.

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 screenshot shows the RStudio interface with an R Markdown document titled "13-htmlwidget.Rmd". The code chunk at the top defines a title, output type, and imports leaflet and dplyr libraries. The main content of the chunk uses the leaflet package to create a map of Mount Eden in Auckland, NZ. The resulting map is displayed in the "Viewer" pane, titled "Interactive Map". It includes a marker for "Maunga Whau" and a callout box. The map is overlaid on a street map of the surrounding 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 ````  
19  
20
```

Interactive Map

Use the leaflet map below to explore the actual Maunga Whau volcano in Auckland, NZ.

```
leaflet() %>%  
  setView(lng=174.764, lat=-36.877, zoom = 16) %>%  
  addTiles() %>%  
  addMarkers(lng=174.764, lat=-36.877, popup="Maunga Whau")
```

The map shows the Mount Eden volcanic crater in Auckland, New Zealand. The crater is green and labeled "Mount Eden 196 m". A blue marker indicates the location of "Maunga Whau". Other landmarks shown include "Reserve", "View Road", "Glenfell Place", "Doris Urquhart", "Albury Avenue", "Owens Road", "Wiers Road", "Oaklands Road", "Rautang Road", "Balger Road", "Sumner Road", "Wilkes Lookout", "Bellevue Road", "Explanatory Road", "Shenstone Road", "Valley Road", "Taitia Street", "Coles Avenue", and "Lovelock Avenue". A callout box points to the "Maunga Whau" marker.

# 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:

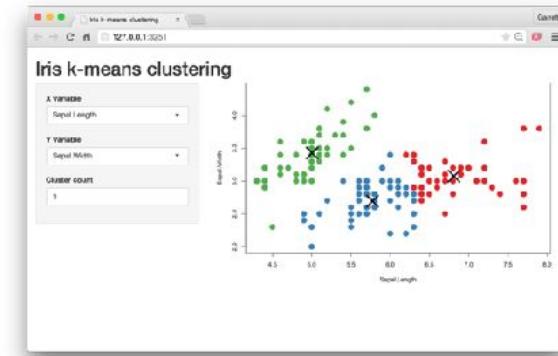
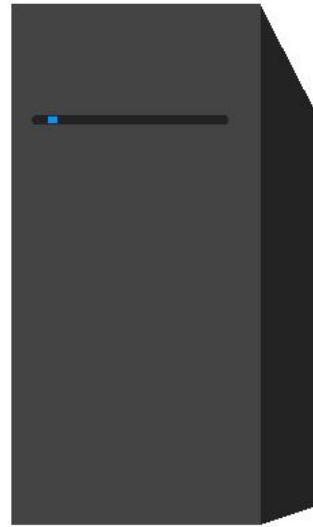
In the past

- 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 chunk.

# 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 off the chapter by learning the third important component of Shiny apps: reactive expressions.

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

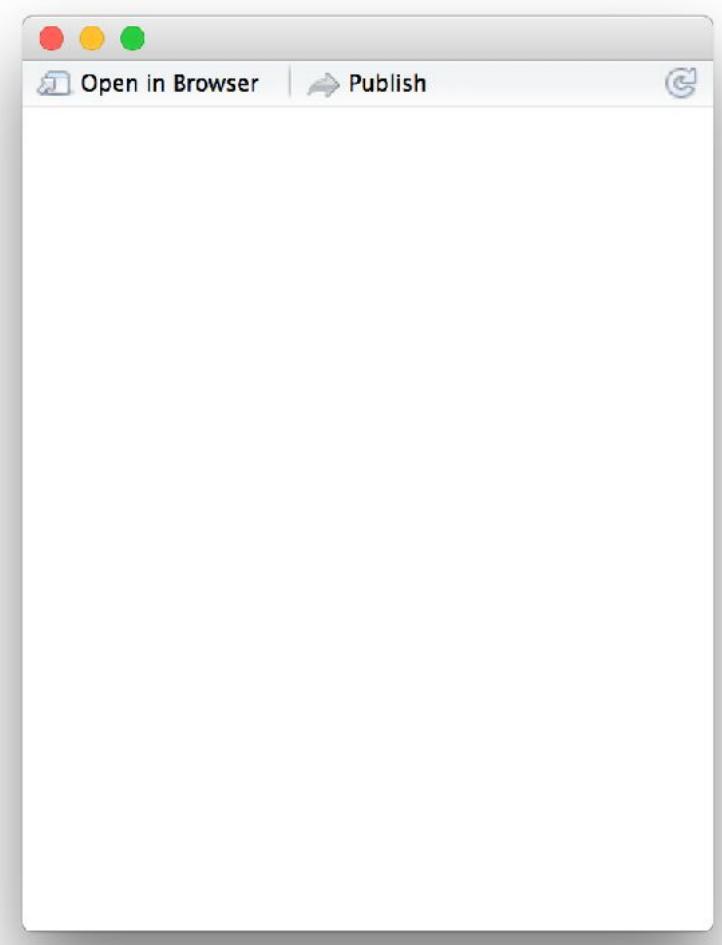
# Build your app

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

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

# Inputs

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

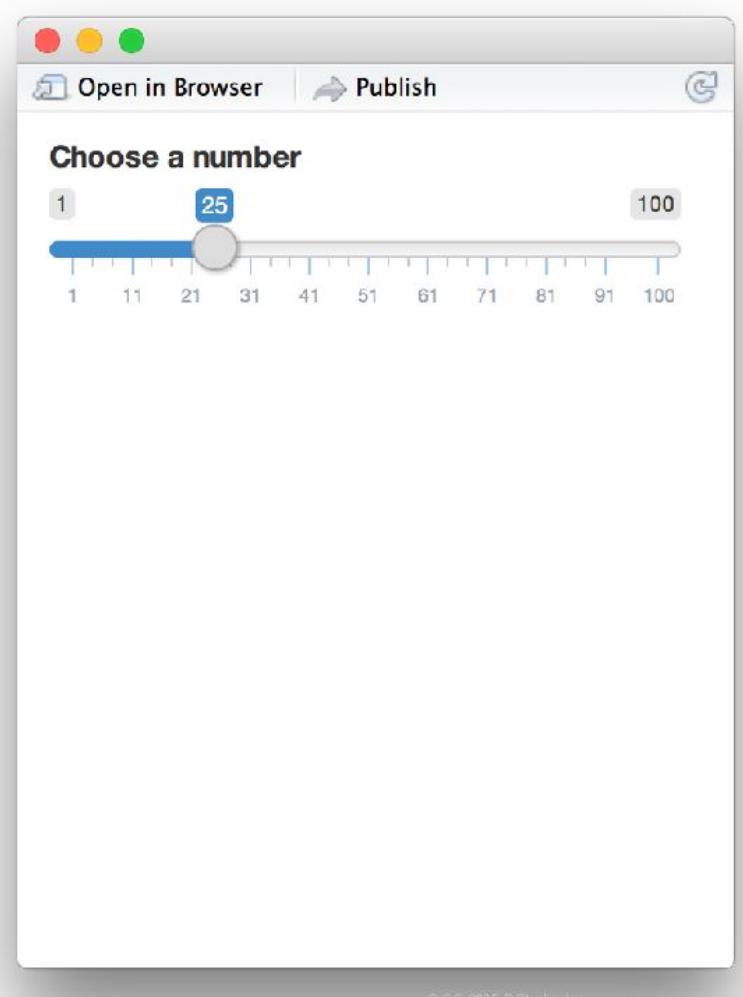


# Inputs

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

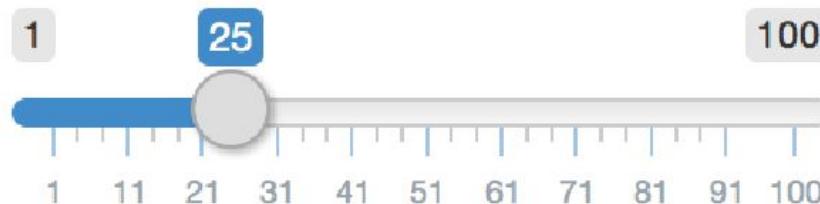
server <- function(input, output) {}

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



# Syntax

Choose a number



```
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()`

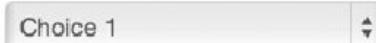
## Radio buttons

- Choice 1
- Choice 2
- Choice 3

`radioButtons()`

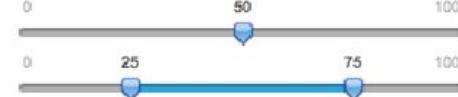
## Select box

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 ?

# Server function

1

Save objects to display to output\$

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

2

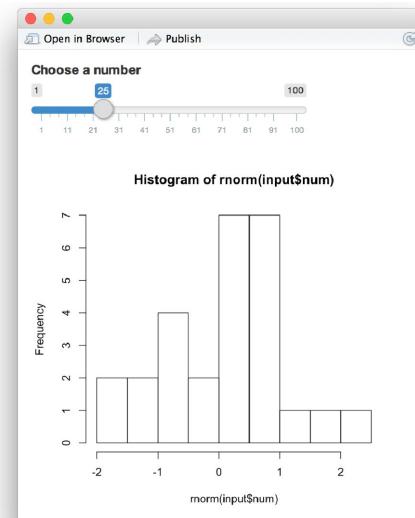
Build objects to display with **render\***()

```
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

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

# Two inputs

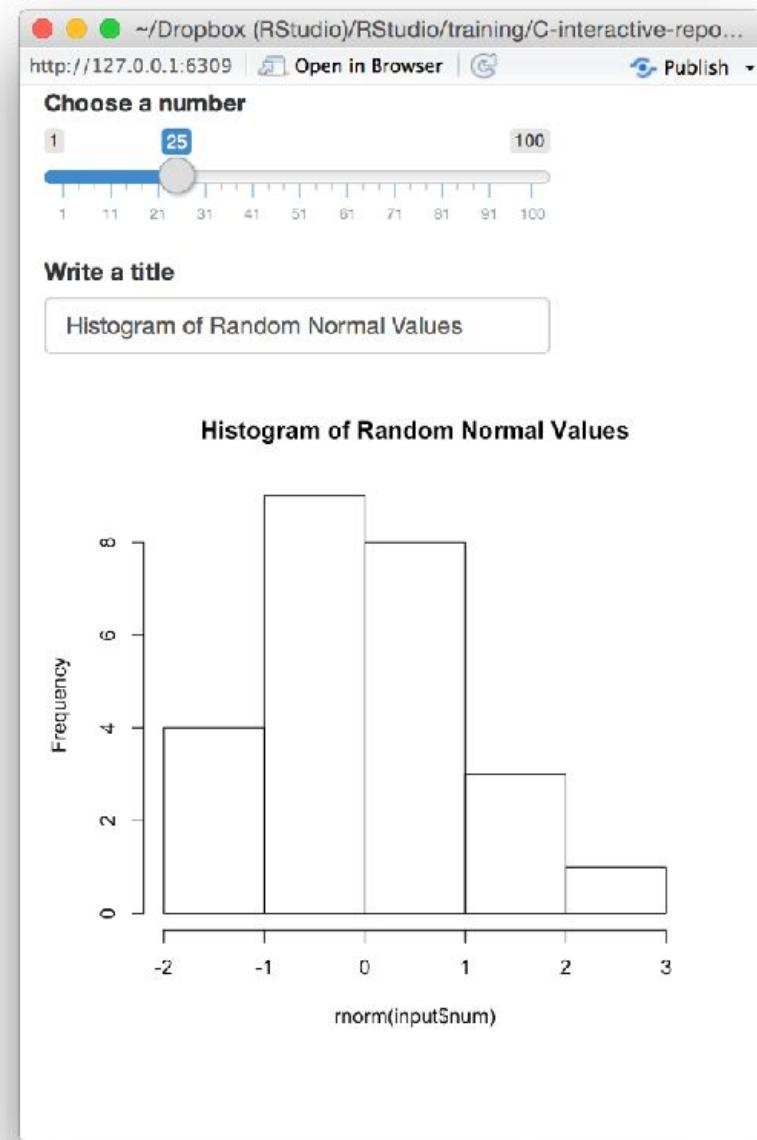
```
# 01-two-inputs

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

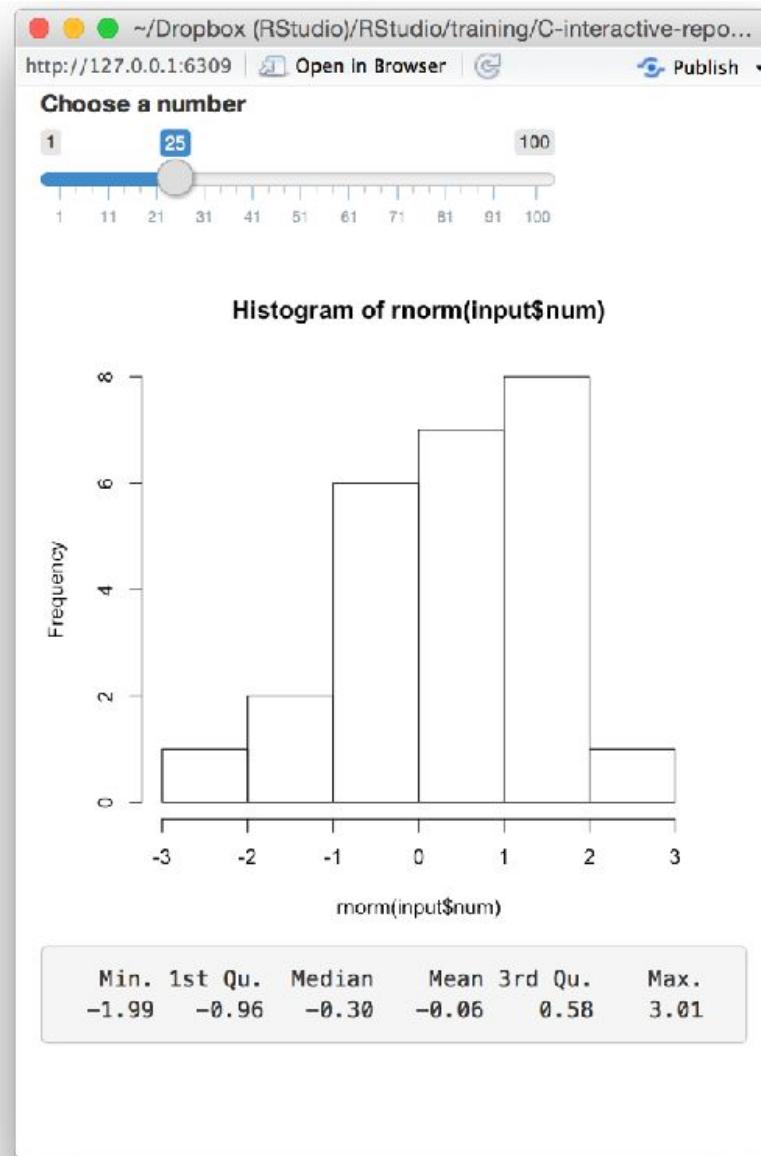
```
# 02- two-outputs

library(shiny)

ui <- fluidPage(
  sliderInput(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)
```



# 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()

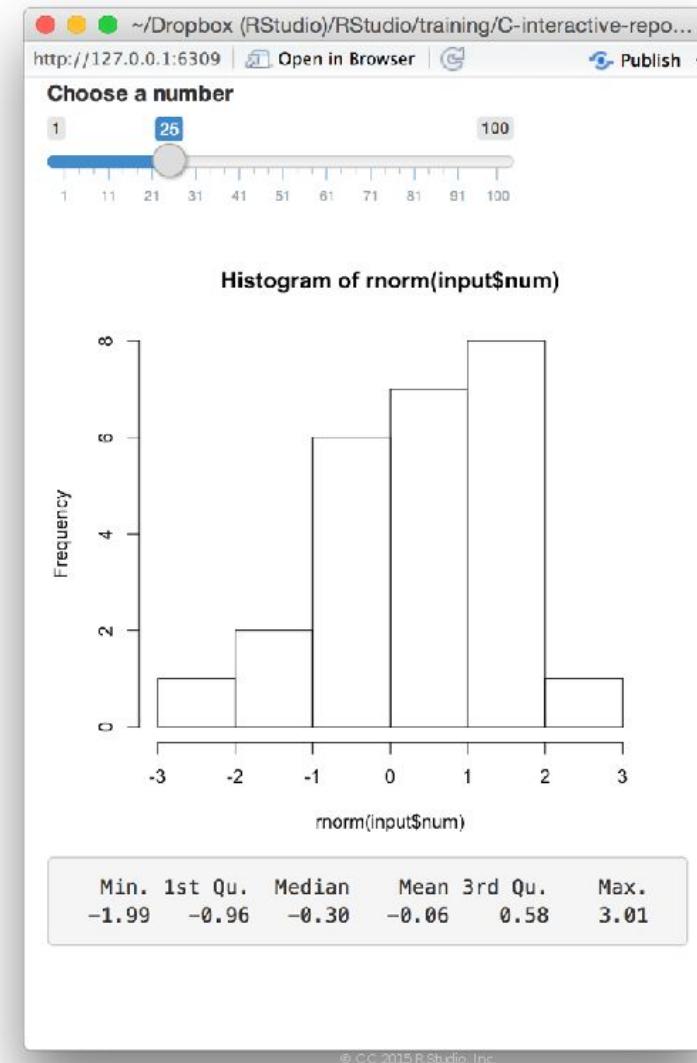
```
# 03-reactive

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

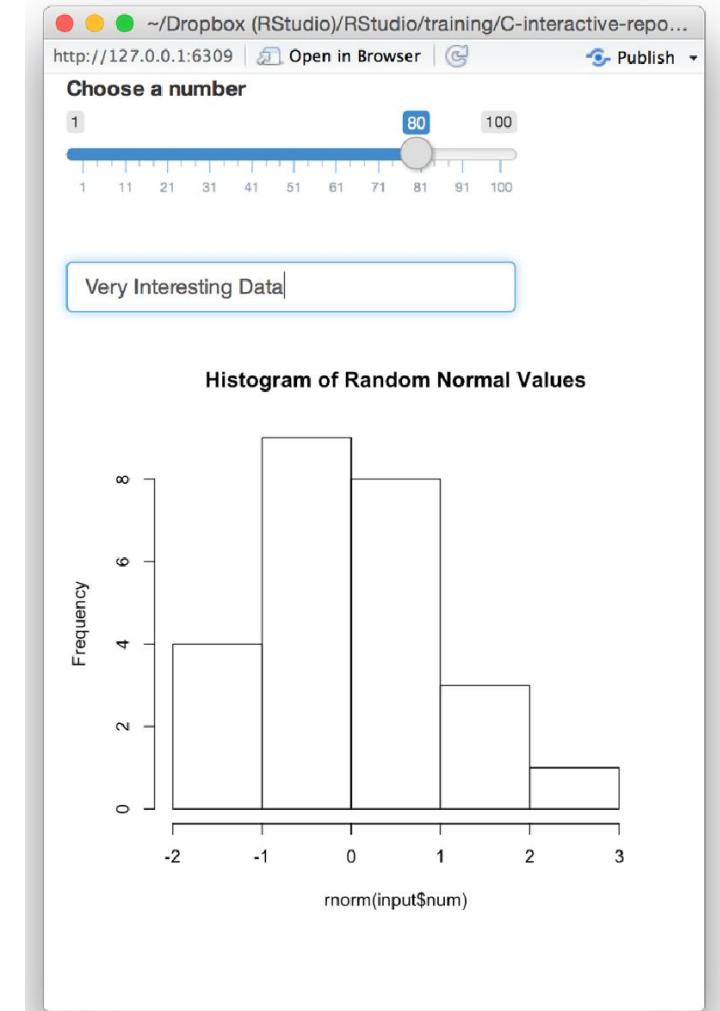
# Can we prevent the title field from updating the plot?

```
# 04- isolate
library(shiny)

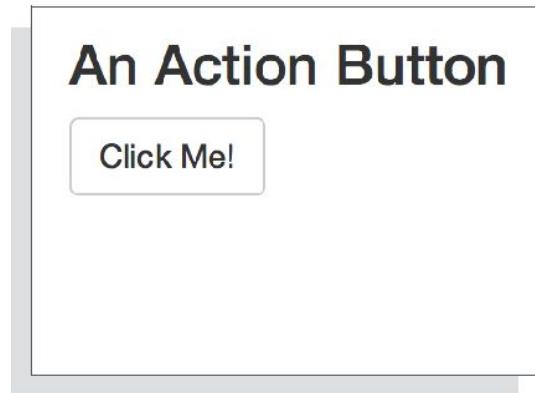
ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  textInput(inputId = "title",
    label = "Write a title",
    value = "Histogram of Random Normal Values"),
  plotOutput("hist")
)

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

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



# 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

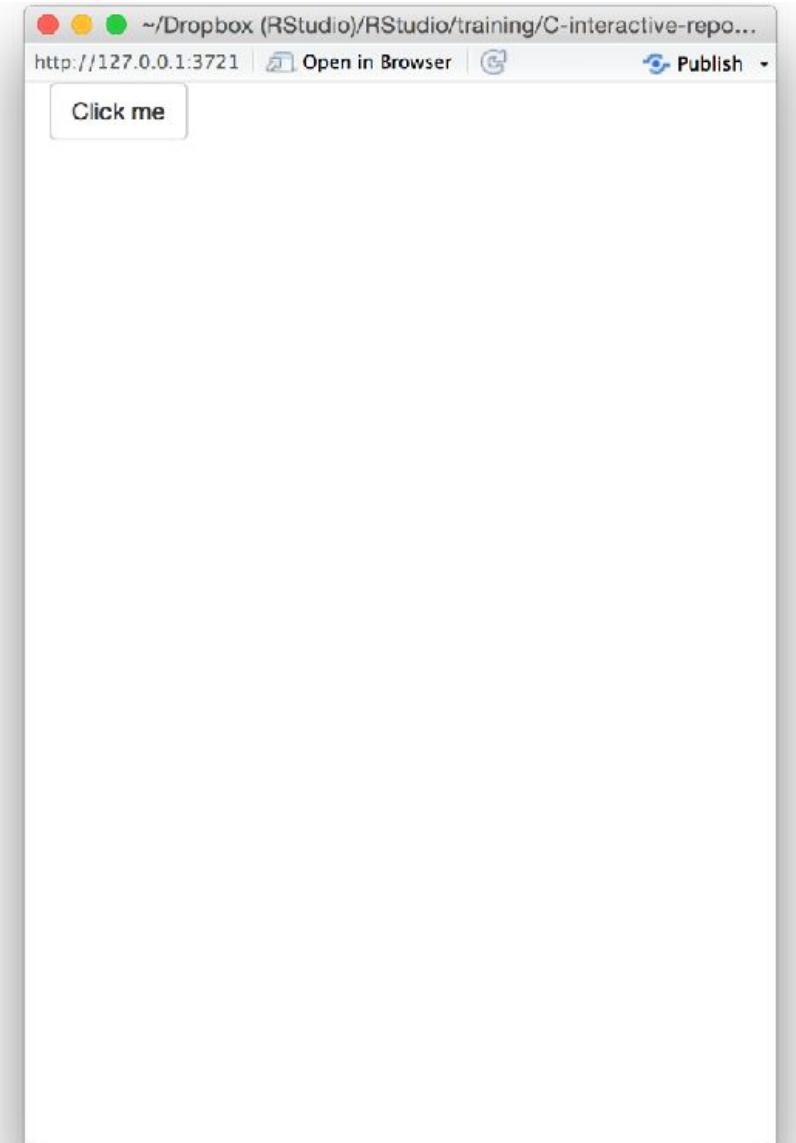
```
# 05- actionButton

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 = "unif", label = "Uniform"),
  plotOutput("hist")
)

server <- function(input, output) {

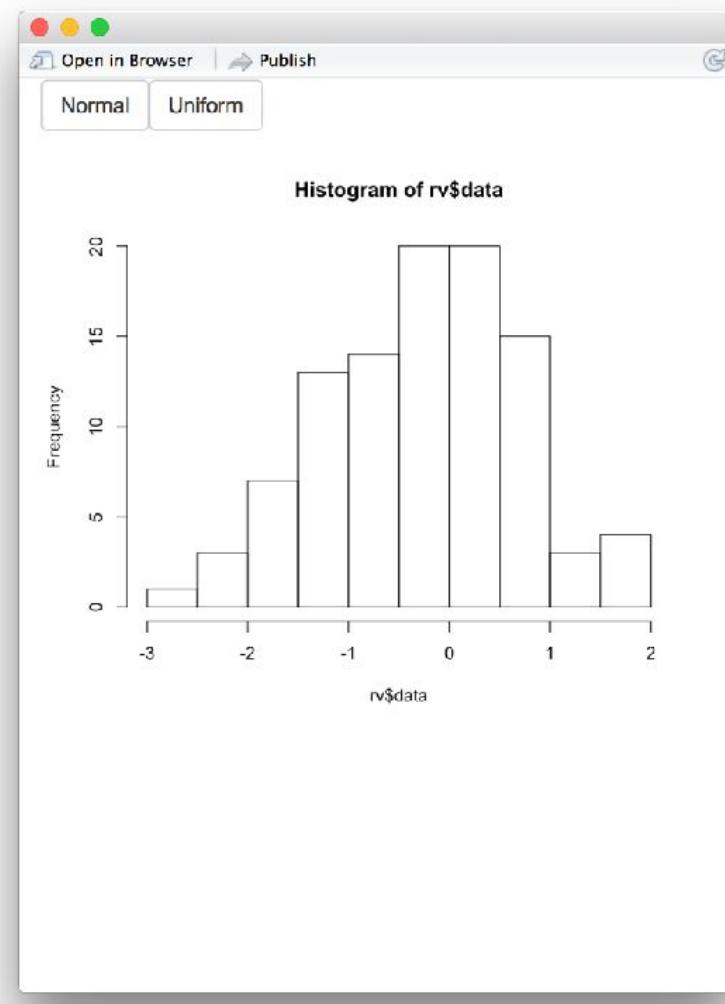
  rv <- reactiveValues(data = rnorm(100))

  observeEvent(input$norm, { rv$data <- rnorm(100) })
  observeEvent(input$unif, { rv$data <- runif(100) })

  output$hist <- renderPlot({
    hist(rv$data)
  })
}

shinyApp(ui = ui, server = server)

```



# Summary

