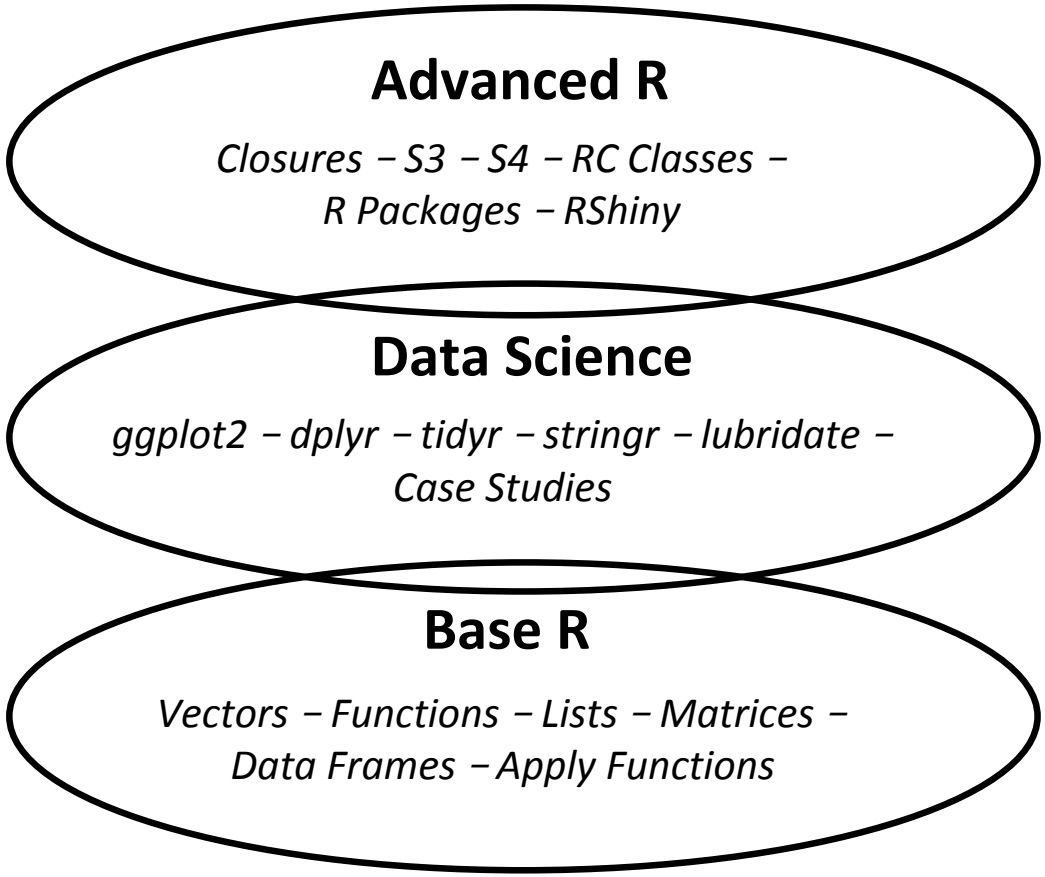
CT5102: Programming for Data Analytics

Lecture 12: Course Summary

Dr. Jim Duggan,
School of Engineering & Informatics
National University of Ireland Galway.
https://github.com/JimDuggan/CT5102

Lecture Overview

- Closures
- RMarkdown
- Rshiny
- Course summary



(1) Closures

"An object is data with functions. A closure is a function with data." John D. Cook.

- Anonymous functions can be used to create closures, functions written by functions
- Closures get their name because the enclose the environment of the parent function and can then access all its variables
- "Stateful functions (closures) are best used in moderation. As soon as your function starts managing the state of multiple variables, it's better to switch to R6" (Wickham)

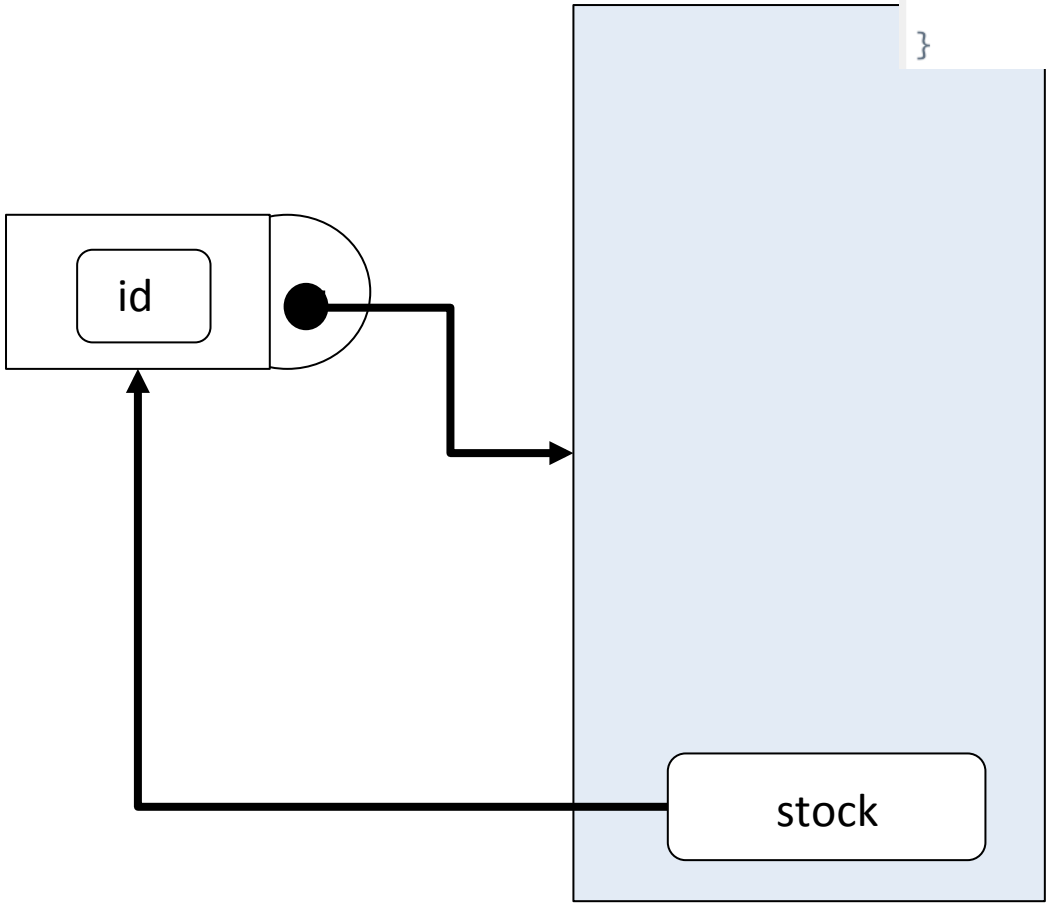
Closure Example

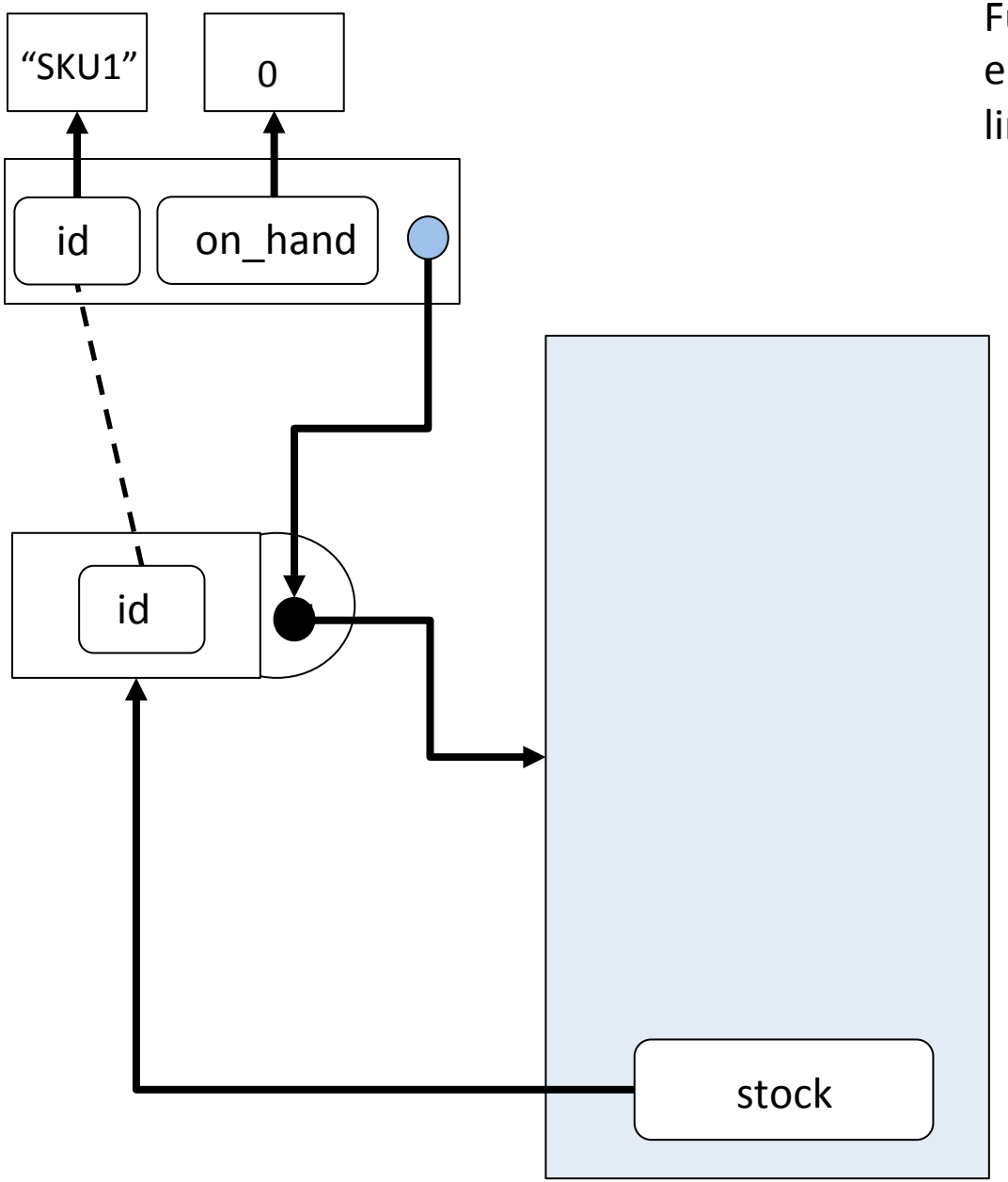
```
stock <- function(id){
  id <- id
  on_hand <- 0
  list(increment=function(a)on_hand<<-on_hand+a,
       decrement=function(a)on_hand<<-on_hand-a,
       on_hand=function()on_hand,
       id=function()id,
       set_id=function(id)id<<-id,
       get_state=function()list(id=id,
                                on_hand=on_hand))
}

s1 <- stock("SKU1")

s1$increment(100)
s1$on_hand()
```

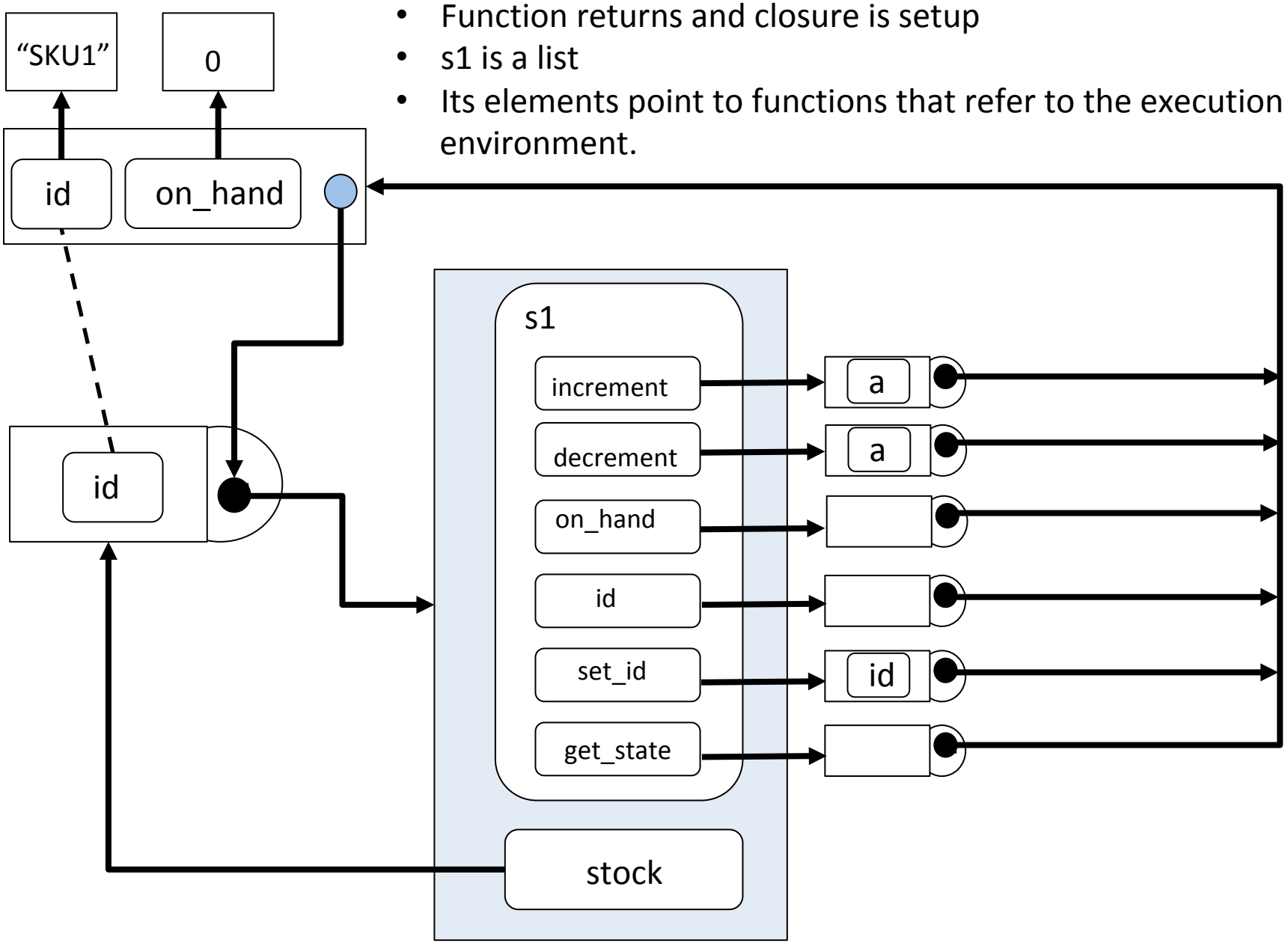
```
stock <- function(id){
  id <- id
  on_hand <- 0
  list(increment=function(a)on_hand<<-on_hand+a,
       decrement=function(a)on_hand<<-on_hand-a,
       on_hand=function()on_hand,
       id=function()id,
       set_id=function(id)id<<-id,
       get_state=function()list(id=id,
                                on_hand=on_hand))
}
```





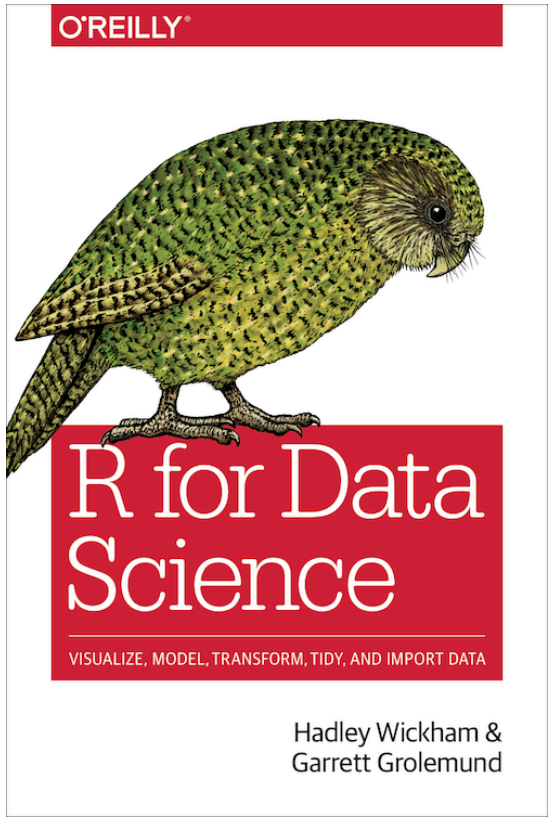
Function gets called and execution environment is created (first two lines of code)

- Function returns and closure is setup
- s1 is a list
- Its elements point to functions that refer to the execution environment.



(2) R Markdown
(Wickham and Grolemund)

- R Markdown provides a unified authoring framework for data science, combining your code, results and prose commentary
- R Markdown documents are fully reproducible and support many output formats (pdfs, slideshows, and more).



Why use R Markdown?

- **For communicating to decision makers**, who want to focus on conclusions, not the code behind the analysis.
- **For collaborating with other data scientists**, who are interested in your conclusions, and how you reached them
- As an environment in which to do data science, where you capture not only what you did, but what you were thinking

R Markdown, loaded by R Studio

- Contains three important types of content:
  - An (optional) YAML header surrounded by ---
  - Chunks of R Code, surrounded by ```
  - Text mixed with simple text formatting like # heading and _italics_

Example

````
---
title: "Diamond Sizes"
date: 2017-08-25
output:
  html_document: default
---

Here is an example of using **R Markdown**.

```{r setup, include=FALSE}
library(ggplot2)
library(dplyr)
```

```{r, echo=FALSE}
smaller <- diamonds %>%
  filter(carat <= 2.5)
```
````

````
We have data about `r nrow(diamonds)`
diamonds in our data set.
Only **`r nrow(diamonds) - nrow(smaller)`**
are larger than 2.5 carats.

The distribution of the remainder is show below:

```{r, echo=FALSE}
smaller %>%
  ggplot(aes(carat)) +
  geom_freqpoly(binwidth=0.01)

```
````

https://yihui.name/knitr/options/

“knit” to HTML

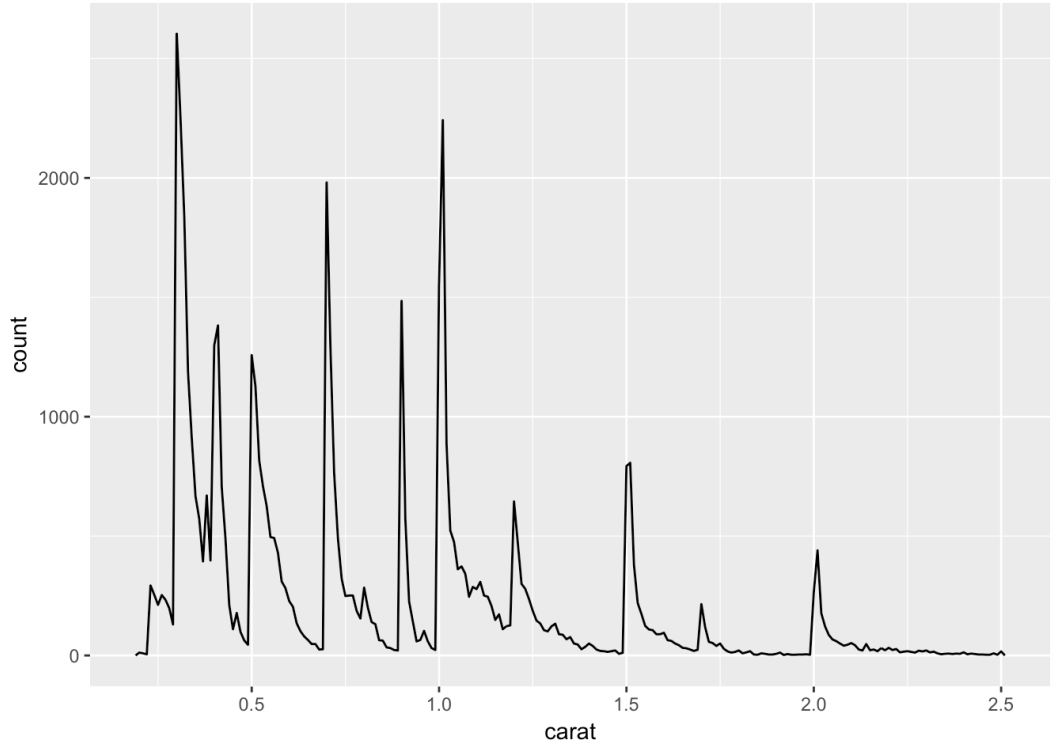
Diamond Sizes

2017-08-25

Here is an example of using **R Markdown**.

We have data about 53940 diamonds in our data set. Only **126** are larger than 2.5 carats.

The distribution of the remainder is show below:



“knit” to PDF

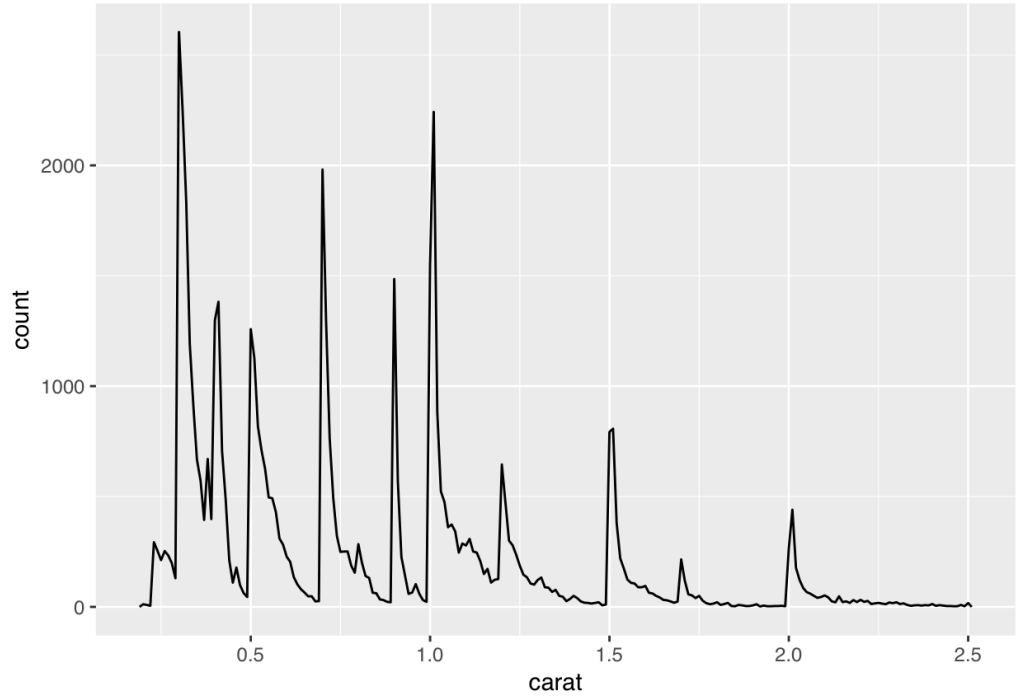
Diamond Sizes

2017-08-25

Here is an example of using **R Markdown**.

We have data about 53940 diamonds in our data set. Only **126** are larger than 2.5 carats.

The distribution of the remainder is show below:



Text Formatting with Markdown

- Text Formatting
  - *italic* or _italic_
  - **bold** or _bold_
  - `code`
- Headings
  - # First Level header
  - ## Second Level header
  - ### Third Level header
- Lists

  * Bulleted list item 1

  1 Numbered list item 1

- Links

  [linked phrase]http://example.com
- Images

  ![caption text](path/to/imp.png)

Inserting Chunks (Cmd/Ctrl-Alt-I)

This table summarizes what types of output each option suppresses…

Option	Run code	Show Code	Output	Plots	Messages	Warnings
eval = FALSE	X		X	X	X	X
include = FALSE		X	X	X	X	X
echo = FALSE		X				
results= “hide”			X			
fig.show=“hide”				X		
message=FALSE					X	
Warning=FALSE						X

Table

```
---
title: "Table Test"
output: html_document
---
```

```
```{r}
mtcars[1:5,1:10]
```
```

```
```{r}
knitr::kable(
  mtcars[1:5,1:10],
  caption="A knitr kable"
)

```
```

Table Test

```
mtcars[1:5,1:10]
```

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

```
knitr::kable(
  mtcars[1:5,1:10],
  caption="A knitr kable"
)
```

A knitr kable

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

Inline Code

```
There are `r nrow(mtcars)` records in *mtcars*
```

Table Test

```
mtcars[1:5,1:10]
```

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

```
knitr::kable(
  mtcars[1:5,1:10],
  caption="A knitr kable"
)
```

A knitr kable

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

There are 32 records in *mtcars*

YAML Header

- "Yet Another Markup Language"
- Useful features
  - Parameters
  - Bibliographies

Example

```
---
title: "Parameter Test"
bibliography: ref.bib
params:
  my_class: suv
  my_time: !r lubridate::now()

output:
  html_document: default
  pdf_document: default
---

The time is now `r params$my_time`

The reference is [@paper1]
```

````
```{r setup, include=FALSE}
library(ggplot2)
library(dplyr)

class <- mpg %>% filter(class == params$my_class)

```
````

````
```{r, message=FALSE}
ggplot(class,aes(x=displ,y=hwy))+
  geom_point()+
  geom_smooth(se=F)

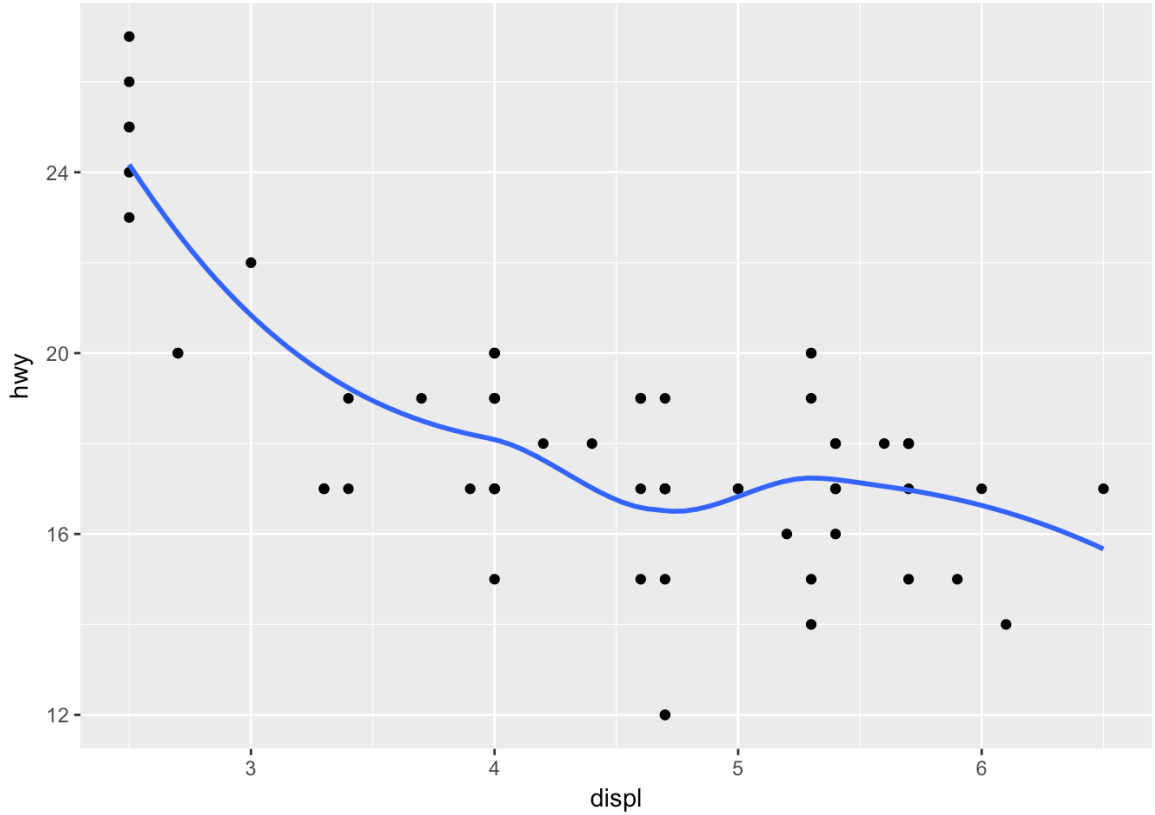
```
````

Parameter Test

The time is now 2017-11-15 19:41:58

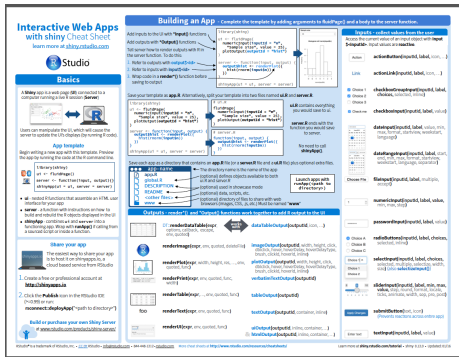
The reference is (Koppeschaar et al. 2017)

```
ggplot(class,aes(x=displ,y=hwy))+
  geom_point()+
  geom_smooth(se=F)
```



Koppeschaar, E. Carl, Vittoria Colizza, Caroline Guerrisi, Clément Turbelin, Jim Duggan, John W. Edmunds, Charlotte Kjelsø, et al. 2017. "Influenzanet: Citizens Among 10 Countries Collaborating to Monitor Influenza in Europe." *JMIR Public Health Surveill* 3 (3): e66. http://publichealth.jmir.org/2017/3/e66/.

(3) RShiny



- A Shiny app is a web page (UI) connected to a computer running a live R session (Server)

- Users can manipulate the UI which will cause the server to update the UI's displays (by running code)

App template

- **ui** – nested R functions that represent an HTML user interface for your app
- **server** – a function with instructions on how to build and rebuild the R objects displayed in the UI
- **shinyApp** – combines the **ui** and **server** into a functioning app.

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

Building an App

Add inputs to the UI with ***Input()** functions

Add outputs with ***Output()** functions

Tell server how to render outputs with R in the server function. To do this:

1. Refer to outputs with **output$<id>**
2. Refer to inputs with **input$<id>**
3. Wrap code in a **render*()** function before saving to output

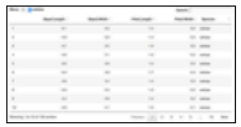
```
library(shiny)

ui <- fluidPage(
  numericInput(inputId = "n",
    "Sample size", value = 25),
  plotOutput(outputId = "hist")
)

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

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

Outputs - render*() and *Output() functions work together to add R output to the UI



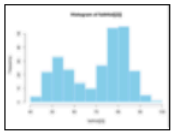
DT::**renderDataTable**(expr, options, callback, escape, env, quoted)

works with

dataTableOutput(outputId, icon, …)

renderImage(expr, env, quoted, deleteFile)

imageOutput(outputId, width, height, click, dblclick, hover, hoverDelay, hoverDelayType, brush, clickId, hoverId, inline)

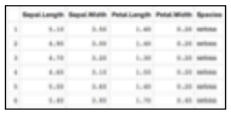


renderPlot(expr, width, height, res, …, env, quoted, func)

plotOutput(outputId, width, height, click, dblclick, hover, hoverDelay, hoverDelayType, brush, clickId, hoverId, inline)

renderPrint(expr, env, quoted, func, width)

verbatimTextOutput(outputId)



renderTable(expr,…, env, quoted, func)

tableOutput(outputId)

foo

renderText(expr, env, quoted, func)

textOutput(outputId, container, inline)

renderUI(expr, env, quoted, func)

uiOutput(outputId, inline, container, …)
& **htmlOutput**(outputId, inline, container, …)

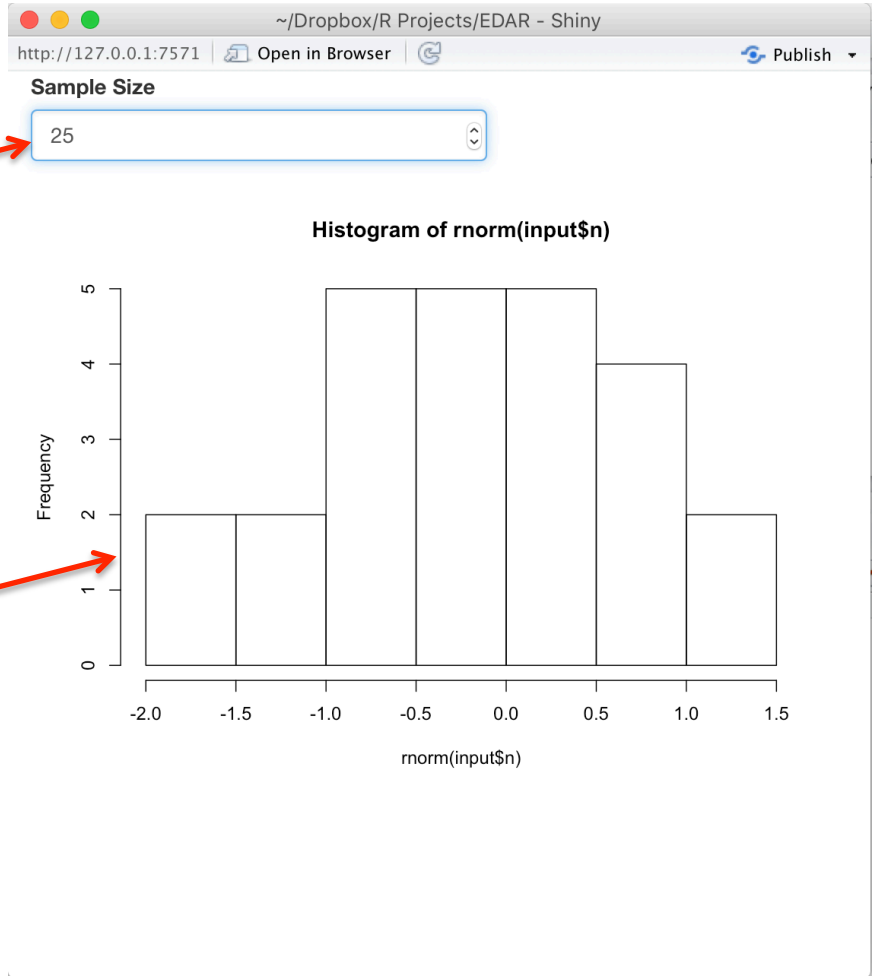
Example

```
library(shiny)

ui <- fluidPage(
  numericInput(inputId = "n",
               "Sample Size",
               value=25),
  plotOutput(outputId = "hist")
)

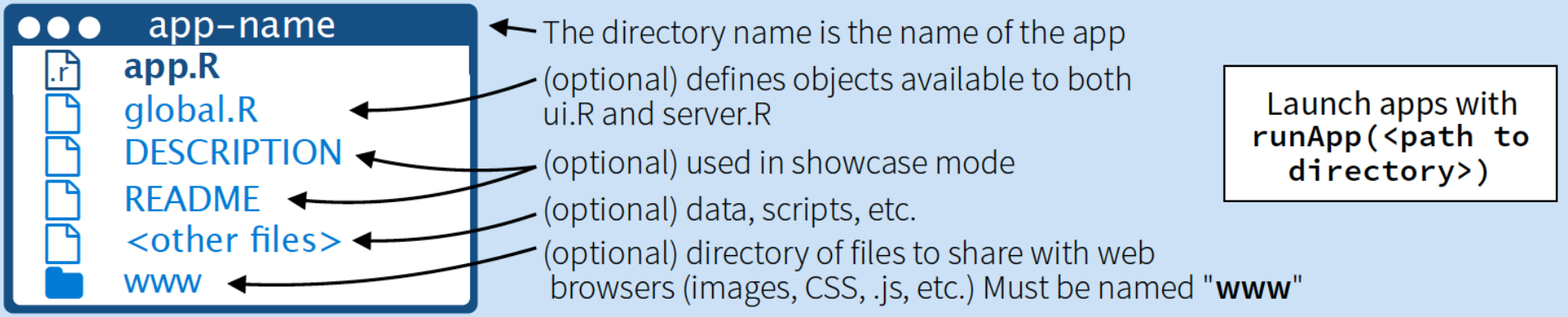
server <- function(input, output){
  output$hist <- renderPlot({
    hist(rnorm(input$n))
  })
}

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



File Organisation

Save each app as a directory that contains an **app.R** file (or a **server.R** file and a **ui.R** file) plus optional extra files.



Launch apps with `runApp(<path to directory>)`

Inputs - collect values from the user

Access the current value of an input object with **input $<inputId>**. Input values are **reactive**.

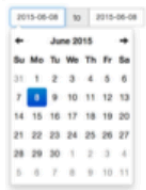
Action — **actionButton**(inputId, label, icon, …)

Link — **actionLink**(inputId, label, icon, …)

Choice 1
Choice 2
Choice 3 — **checkboxGroupInput**(inputId, label, choices, selected, inline)

Check me — **checkboxInput**(inputId, label, value)

dateInput(inputId, label, value, min, max, format, startview, weekstart, language)



dateRangeInput(inputId, label, start, end, min, max, format, startview, weekstart, language, separator)

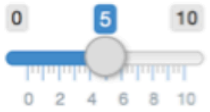
Choose File — **fileInput**(inputId, label, multiple, accept)

1 — **numericInput**(inputId, label, value, min, max, step)

••••••••• — **passwordInput**(inputId, label, value)

Choice A
Choice B
Choice C — **radioButtons**(inputId, label, choices, selected, inline)

Choice 1
Choice 1
Choice 2 — **selectInput**(inputId, label, choices, selected, multiple, selectize, width, size) (also **selectizeInput()**)

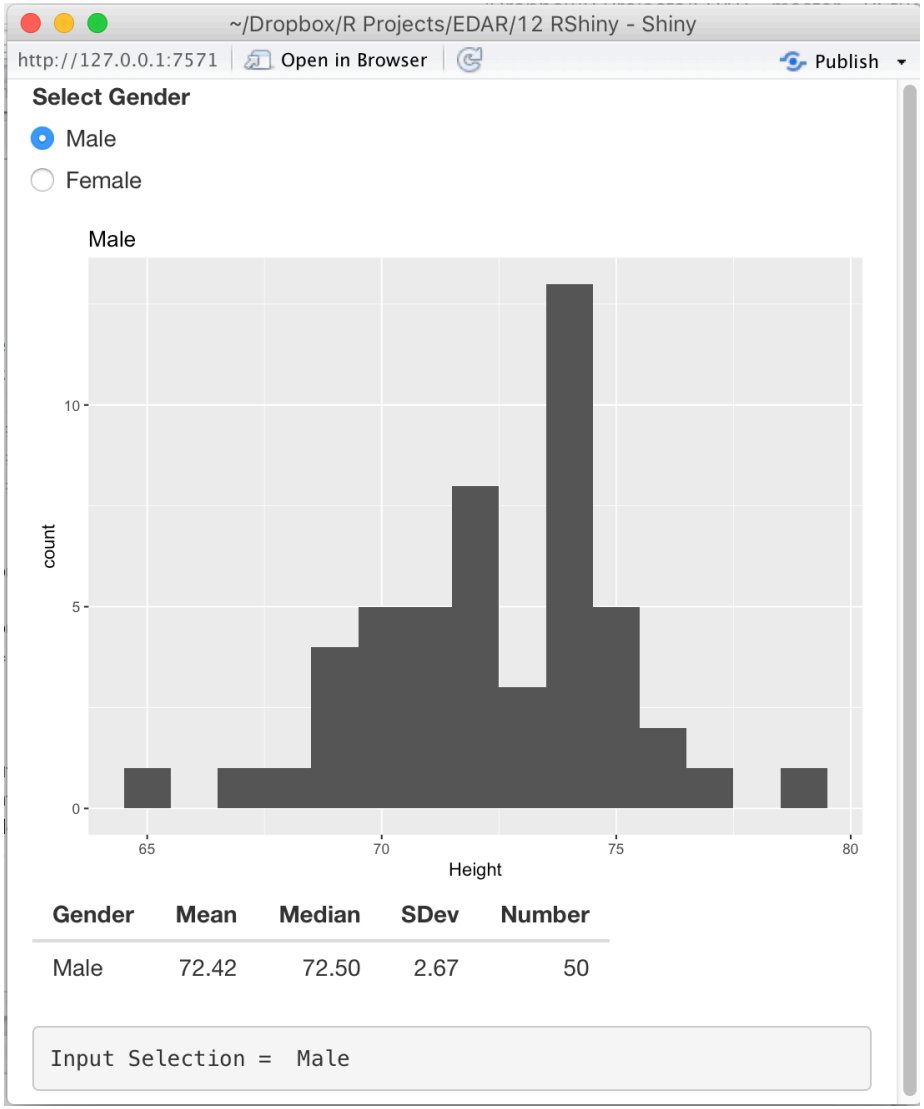


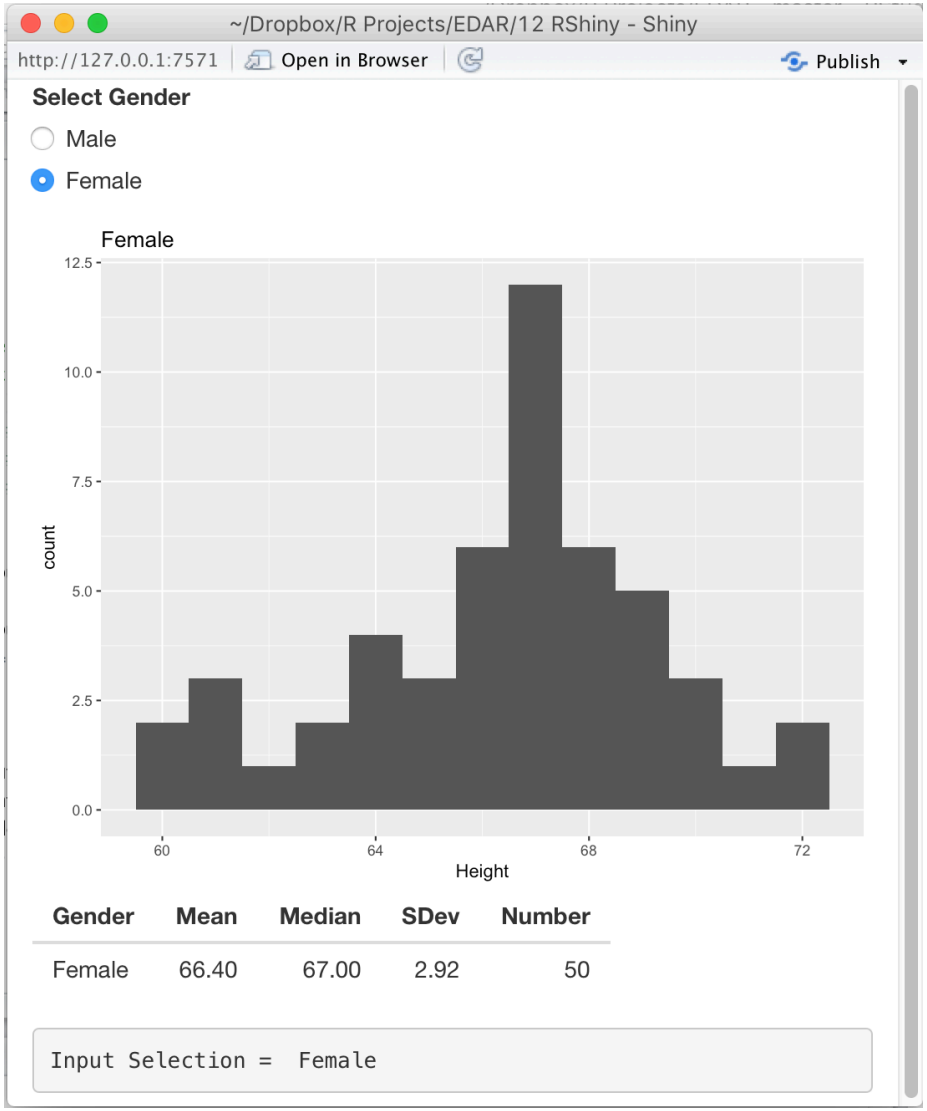
sliderInput(inputId, label, min, max, value, step, round, format, locale, ticks, animate, width, sep, pre, post)

Apply Changes — **submitButton**(text, icon)
(Prevents reactions across entire app)

Enter text — **textInput**(inputId, label, value)

Additional Example…



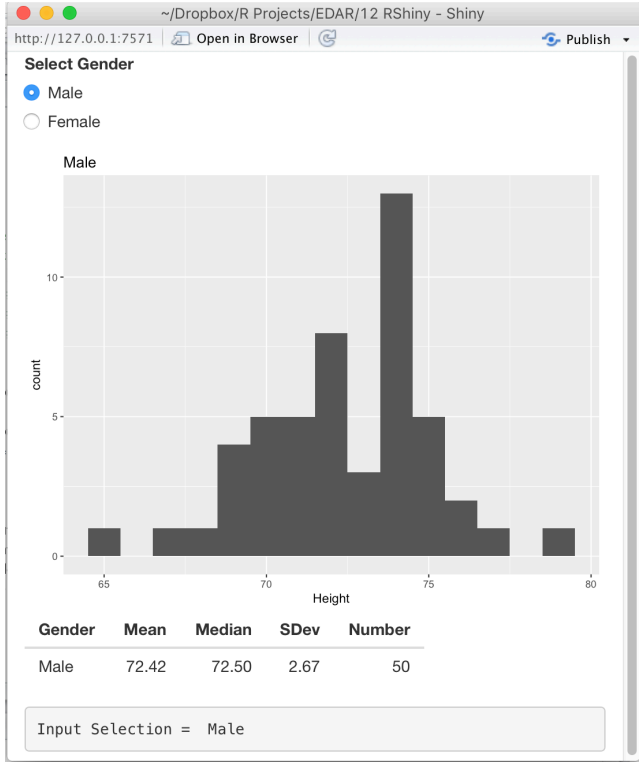


```
ui <- fluidPage(
  radioButtons(inputId = "gender",
               label   = "Select Gender",
               c("Male","Female")),
  plotOutput("hist"),          # to plot a chart
  tableOutput("tab"),          # to generate a table
  verbatimTextOutput("text")   # to generate some text
)

server <- function (input, output){
  output$hist <- renderPlot({
    ggplot(filter(d,Gender==input$gender), aes(Height)) +
      geom_histogram(binwidth = 1) + ggtitle(input$gender)
    })

  output$tab <- renderTable({
    sd <- filter(d,Gender==input$gender) %>% group_by(Gender) %>%
      summarise(Mean=mean(Height),
                Median=median(Height),
                SDev=sd(Height),
                Number=n()
      )
  })

  output$text <- renderPrint({
    cat("Input Selection = ", input$gender,"\n")
  })
}
```



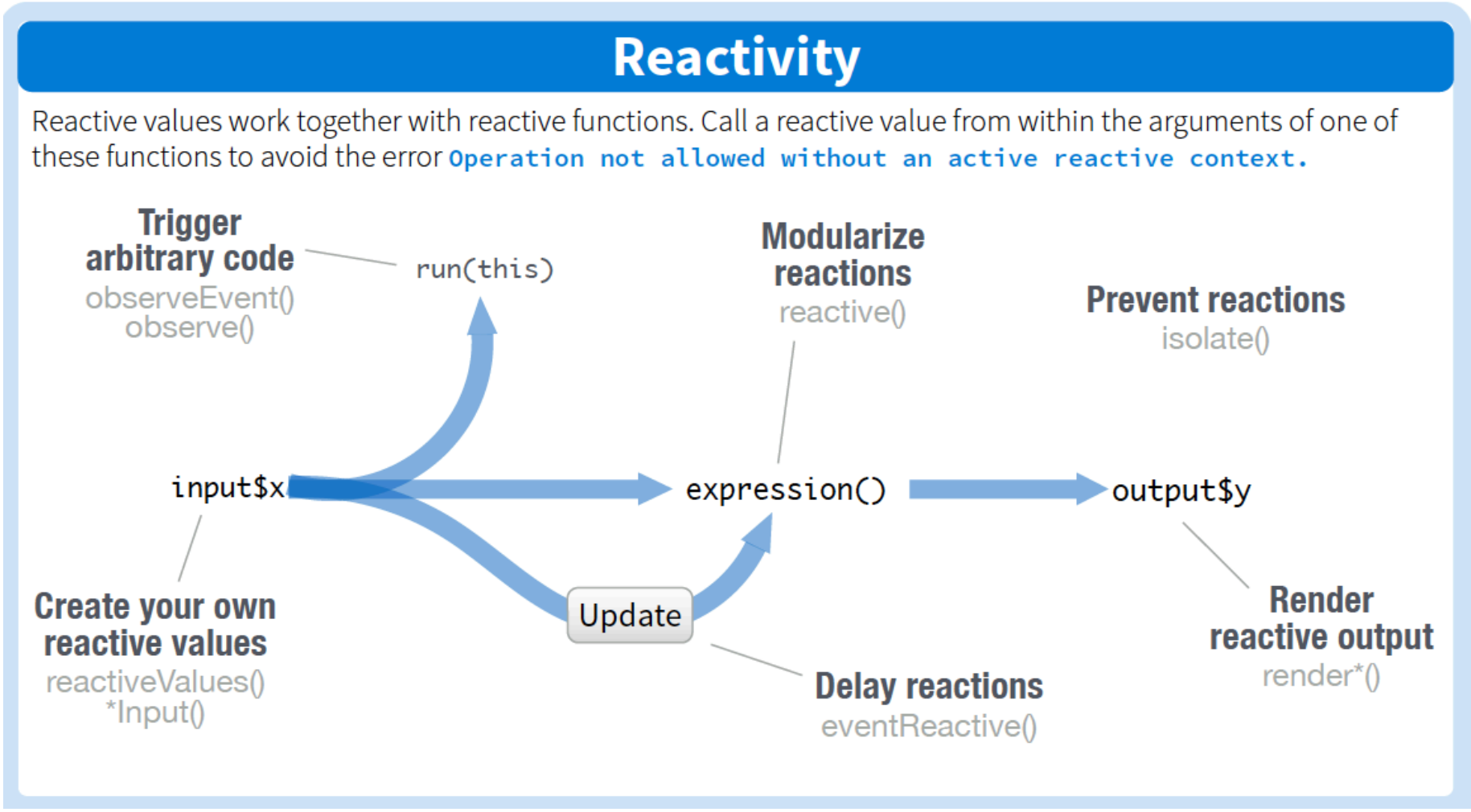
Setup code

```
d <- read_csv("HopeCollegeHeights.csv")

d <- d %>% mutate(Gender=as.factor(ifelse(Gender==1,"Female","Male")))

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

More complex structures… supports Action Buttons and Shared State



Create your own reactive values

```
# example snippets


ui <- fluidPage(
 textInput("a","","A")
)


server <-
function(input,output){
 rv <- reactiveValues()
 rv$number <- 5
}
```

***Input() functions**
(see front page)

reactiveValues(...)

Each input function creates a reactive value stored as **input$<inputId>**

reactiveValues() creates a list of reactive values whose values you can set.

Render reactive output

```
library(shiny)
ui <- fluidPage(
 textInput("a","","A"),
 textOutput("b")
)

server <-
function(input,output){
 output$b <-
  renderText({
   input$a
  })
}

shinyApp(ui, server)
```

render*() functions
(see front page)

Builds an object to display. Will rerun code in body to rebuild the object whenever a reactive value in the code changes.

Save the results to **output$<outputId>**

Prevent reactions

```
library(shiny)
ui <- fluidPage(
 textInput("a","","A"),
 textOutput("b")
)

server <-
function(input,output){
 output$b <-
  renderText({
   isolate({input$a})
  })
}

shinyApp(ui, server)
```

isolate(expr**)**

Runs a code block. Returns a **non-reactive** copy of the results.

Trigger arbitrary code

```
library(shiny)
ui <- fluidPage(
 textInput("a","","A"),
 actionButton("go","Go")
)

server <-
function(input,output){
 observeEvent(input$go,{
  print(input$a)
 })
}


shinyApp(ui, server)
```

observeEvent(eventExpr**,** handlerExpr, event.env, event.quoted, handler.env, handler.quoted, labe, suspended, priority, domain, autoDestroy, ignoreNULL**)**

Runs code in 2nd argument when reactive values in 1st argument change. See **observe()** for alternative.

Modularize reactions

```
library(shiny)
ui <- fluidPage(
 textInput("a","","A"),
 textInput("z","","Z"),
 textOutput("b")
)

server <-
function(input,output){
 re <- reactive({
  paste(input$a,input
$z)})
 output$b <- renderText({
   re()
  })
}
shinyApp(ui, server)
```

reactive(x, env, quoted, label, domain**)**

Creates a reactive expression that

- caches its value to reduce computation
- can be called by other code
- notifies its dependencies when it ha been invalidated

Call the expression with function syntax, e.g. `re()`

Delay reactions

```
library(shiny)
ui <- fluidPage(
 textInput("a","","A"),
 actionButton("go","Go"),
 textOutput("b")
)

server <-
function(input,output){
 re <- eventReactive(
  input$go,{input$a})
 output$b <- renderText({
   re()
  })
}

shinyApp(ui, server)
```

eventReactive(eventExpr, valueExpr, event.env, event.quoted, value.env, value.quoted, label, domain, ignoreNULL**)**

Creates reactive expression with code in 2nd argument that only invalidates when reactive values in 1st argument change.

UI

An app's UI is an HTML document. Use Shiny's functions to assemble this HTML with R.

```
fluidPage(
  textInput("a","")
)
## <div class="container-fluid">
##  <div class="form-group shiny-input-container">
##     <label for="a"></label>
##     <input id="a" type="text"
##        class="form-control" value=""/>
##   </div>
## </div>
```

Returns HTML

HTML

Add static HTML elements with `tags`, a list of functions that parallel common HTML tags, e.g. `tags$a()`. Unnamed arguments will be passed into the tag; named arguments will become tag attributes.

tags$a	tags$data	tags$h6	tags$nav	tags$span
tags$abbr	tags$datalist	tags$head	tags$noscript	tags$strong
tags$address	tags$dd	tags$header	tags$object	tags$style
tags$area	tags$del	tags$hgroup	tags$ol	tags$sub
tags$article	tags$details	tags$hr	tags$optgroup	tags$summary
tags$aside	tags$dfn	tags$HTML	tags$option	tags$sup
tags$audio	tags$div	tags$i	tags$output	tags$table
tags$b	tags$dl	tags$iframe	tags$p	tags$tbody
tags$base	tags$dt	tags$img	tags$param	tags$td
tags$bdi	tags$em	tags$input	tags$pre	tags$textarea
tags$bdo	tags$embed	tags$ins	tags$progress	tags$tfoot
tags$blockquote	tags$eventsource	tags$kbd	tags$q	tags$th
tags$body	tags$fieldset	tags$keygen	tags$ruby	tags$thead
tags$br	tags$figcaption	tags$label	tags$rp	tags$time
tags$button	tags$figure	tags$legend	tags$rt	tags$title
tags$canvas	tags$footer	tags$li	tags$s	tags$tr
tags$caption	tags$form	tags$link	tags$samp	tags$track
tags$cite	tags$h1	tags$mark	tags$script	tags$u
tags$code	tags$h2	tags$map	tags$section	tags$ul
tags$col	tags$h3	tags$menu	tags$select	tags$var
tags$colgroup	tags$h4	tags$meta	tags$small	tags$video
tags$command	tags$h5	tags$meter	tags$source	tags$wbr

The most common tags have wrapper functions. You do not need to prefix their names with tags$

```
ui <- fluidPage(
  h1("Header 1"),
  hr(),
  br(),
  p(strong("bold")),
  p(em("italic")),
  p(code("code")),
  a(href="", "link"),
  HTML("<p>Raw html</p>")
)
```

CSS

To include a CSS file, use **includeCSS()**, or
1. Place the file in the www subdirectory
2. Link to it with

```
tags$head(tags$link(rel = "stylesheet",
  type = "text/css", href = "<file name>"))
```

JS

To include JavaScript, use **includeScript()** or
1. Place the file in the www subdirectory
2. Link to it with

```
tags$head(tags$script(src = "<file name>"))
```

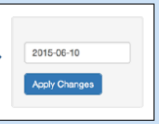
IMAGES

To include an image
1. Place the file in the www subdirectory
2. Link to it with `img(src="<file name>")`

Layouts

Combine multiple elements into a "single element" that has its own properties with a panel function, e.g.

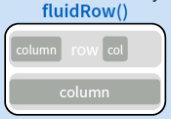
```
wellPanel(
  dateInput("a", ""),
  submitButton()
)
```



absolutePanel()	inputPanel()	tabPanel()
conditionalPanel()	mainPanel()	tabsetPanel()
fixedPanel()	navlistPanel()	titlePanel()
headerPanel()	sidebarPanel()	wellPanel()

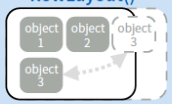
Organize panels and elements into a layout with a layout function. Add elements as arguments of the layout functions.

fluidRow()



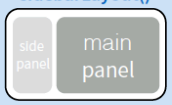

```
ui <- fluidPage(
  fluidRow(column(width = 4),
    column(width = 2, offset = 3)),
  fluidRow(column(width = 12))
)
```

flowLayout()



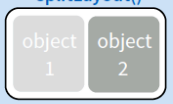

```
ui <- fluidPage(
  flowLayout( # object 1,
              # object 2,
              # object 3
  )
)
```

sidebarLayout()



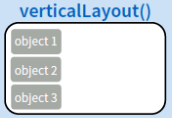

```
ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(),
    mainPanel()
  )
)
```

splitLayout()




```
ui <- fluidPage(
  splitLayout( # object 1,
               # object 2
  )
)
```

verticalLayout()




```
ui <- fluidPage(
  verticalLayout( # object 1,
                  # object 2,
                  # object 3
  )
)
```

Layer tabPanels on top of each other, and navigate between them, with:

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

```
ui <- fluidPage( navlistPanel(
  tabPanel("tab 1", "contents"),
  tabPanel("tab 2", "contents"),
  tabPanel("tab 3", "contents")))
```

```
ui <- navbarPage(title = "Page",
  tabPanel("tab 1", "contents"),
  tabPanel("tab 2", "contents"),
  tabPanel("tab 3", "contents"))
```

Course Summary