CT5102: Programming for Data Analytics

Lecture 12: R Shiny and Course Summary

Dr. Jim Duggan,

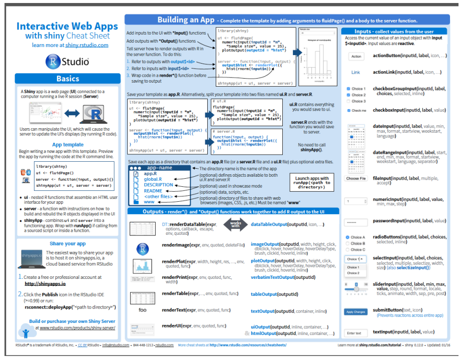
School of Engineering & Informatics

National University of Ireland Galway.

https://github.com/JimDuggan/EDAR

https://twitter.com/_jimduggan

RShiny



Interactive Web Apps with shiny Cheat Sheet
learn more at shiny.rstudio.com

RStudio

Basics

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 R code).

App template

Begin writing a new app with this template. Preview the app by running the code at the R command line.

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

- **ui** - nested R functions that assemble 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 `ui` and `server` into a functioning app. Wrap with **runApp()** if calling from a sourced script or inside a function.

Share your app

The easiest way to share your app is to host it on shinyapps.io, a cloud based service from RStudio

1. Create a free or professional account at **http://shinyapps.io**
2. Click the **Publish** icon in the RStudio IDE (>=0.99) or run: **rsconnect::deployApp**("<path to directory>")

Build or purchase your own Shiny Server at www.rstudio.com/products/shiny-server/

Building an App - Complete the template by adding arguments to fluidPage() and a body to the server function.

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)
```

Save your template as **app.R**. Alternatively, split your template into two files named **ui.R** and **server.R**.

```
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)
```

```
# ui.R
fluidPage(
  numericInput(inputId = "n",
    "Sample size", value = 25),
  plotOutput(outputId = "hist")
)
```

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

ui.R contains everything you would save to ui.

server.R ends with the function you would save to server.

No need to call **shinyApp()**.

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.

```
app-name        The directory name is the name of the app
  app.R
  global.R      (optional) defines objects available to both ui.R and server.R
  DESCRIPTION   (optional) used in showcase mode
  README        (optional) data, scripts, etc.
  <other files> (optional) directory of files to share with web
  www           browsers (images, CSS, .js, etc.) Must be named "www"
```

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

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

	works with
DT::**renderDataTable**(expr, options, callback, escape, env, quoted)	**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)
renderText(expr, env, quoted, func)	**textOutput**(outputId, container, inline)
renderUI(expr, env, quoted, func)	**uiOutput**(outputId, inline, container, ...) & **htmlOutput**(outputId, inline, container, ...)

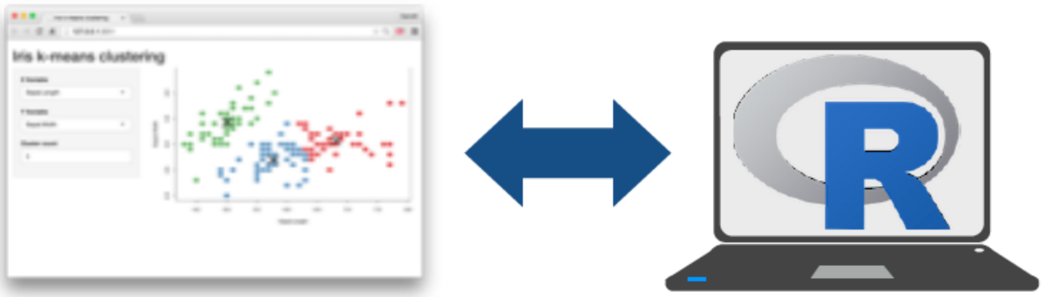
Inputs - collect values from the user

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

actionButton(inputId, label, icon, ...)
actionLink(inputId, label, icon, ...)
checkboxGroupInput(inputId, label, choices, selected, inline)
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)
fileInput(inputId, label, multiple, accept)
numericInput(inputId, label, value, min, max, step)
passwordInput(inputId, label, value)
radioButtons(inputId, label, choices, selected, inline)
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)
submitButton(text, icon) (Prevents reactions across entire app)
textInput(inputId, label, value)

RStudio® is a trademark of RStudio, Inc. • CC BY RStudio • info@rstudio.com • 844-448-1212 • rstudio.com More cheat sheets at http://www.rstudio.com/resources/cheatsheets/ Learn more at shiny.rstudio.com/tutorial • shiny 0.12.0 • Updated: 01/16

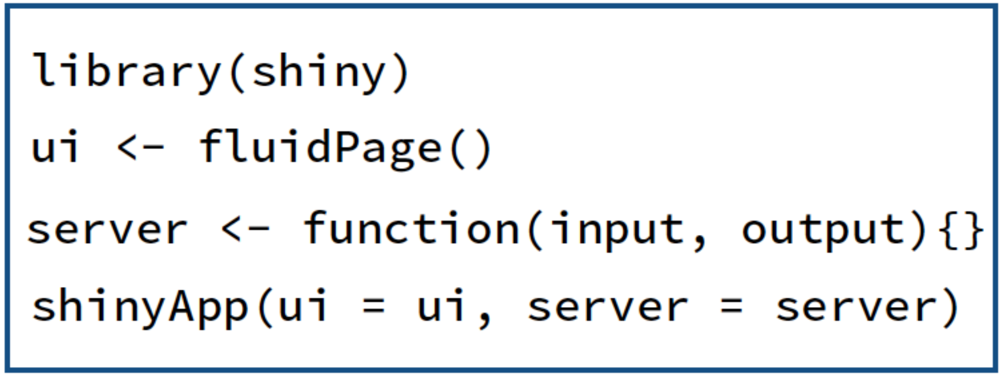
- A Shiny app is a web page (UI) connected to a computer running a live R session (Server)



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

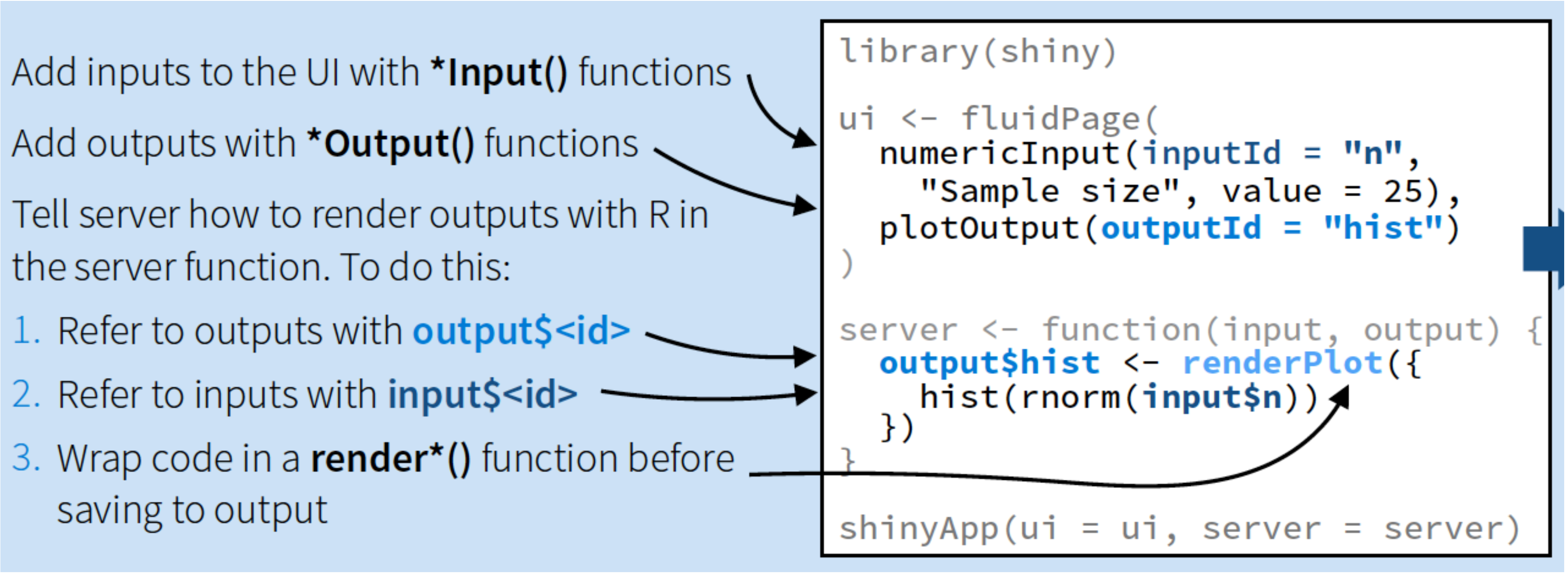
App template

- **ui** – nested R functions that represent an HTML user interface for your app
- **server** – a function with instructions on how to build an 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

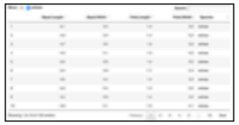
Tell server how to render outputs with R in the server function. To do this:

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

works with

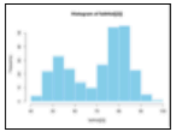


DT::**renderDataTable**(expr, options, callback, escape, env, quoted)

dataTableOutput(outputId, icon, …)

renderImage(expr, env, quoted, deleteFile)

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



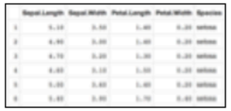
renderPlot(expr, width, height, res, …, env, quoted, func)

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

```
'data.frame':   3 obs. of  2 variables:
 $ Sepal.Length: num  5.1 4.9 4.7
 $ Sepal.Width : num  3.5 3 3.2
```

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)

Choose a number

Write a title

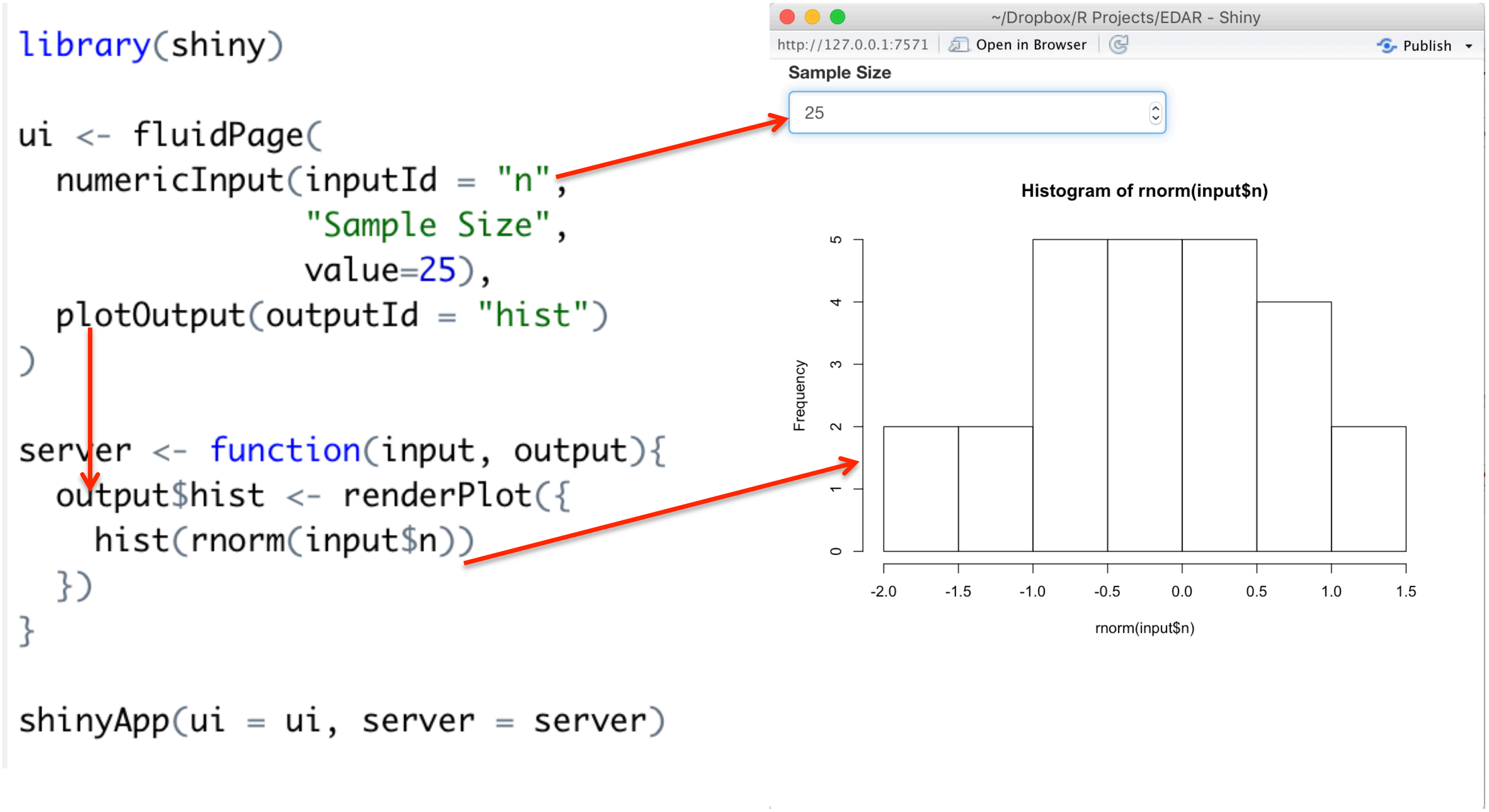
Histogram of Random Normal Values

renderUI(expr, env, quoted, func)

uiOutput(outputId, inline, container, …)

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

Example

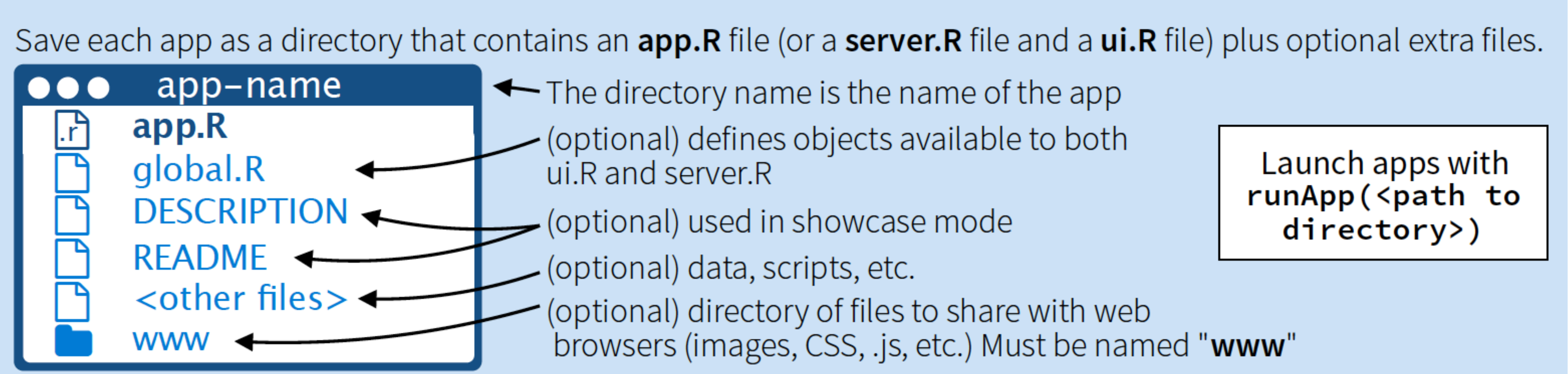


```r
library(shiny)

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)
```

Sample Size: 25

Histogram of rnorm(input$n)

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.

The directory name is the name of the app

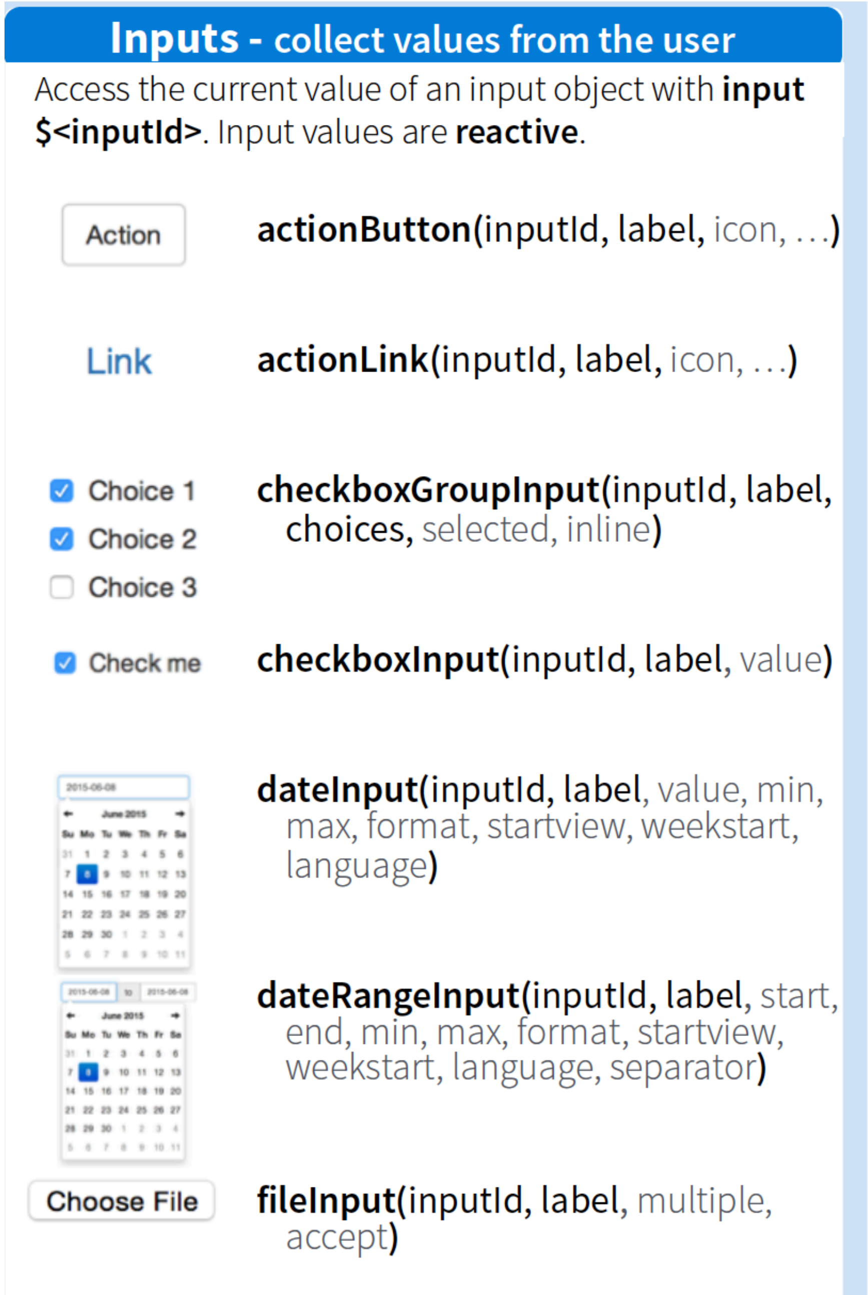
(optional) defines objects available to both ui.R and server.R

(optional) used in showcase mode

(optional) data, scripts, etc.

(optional) directory of files to share with web browsers (images, CSS, .js, etc.) Must be named "**www**"

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

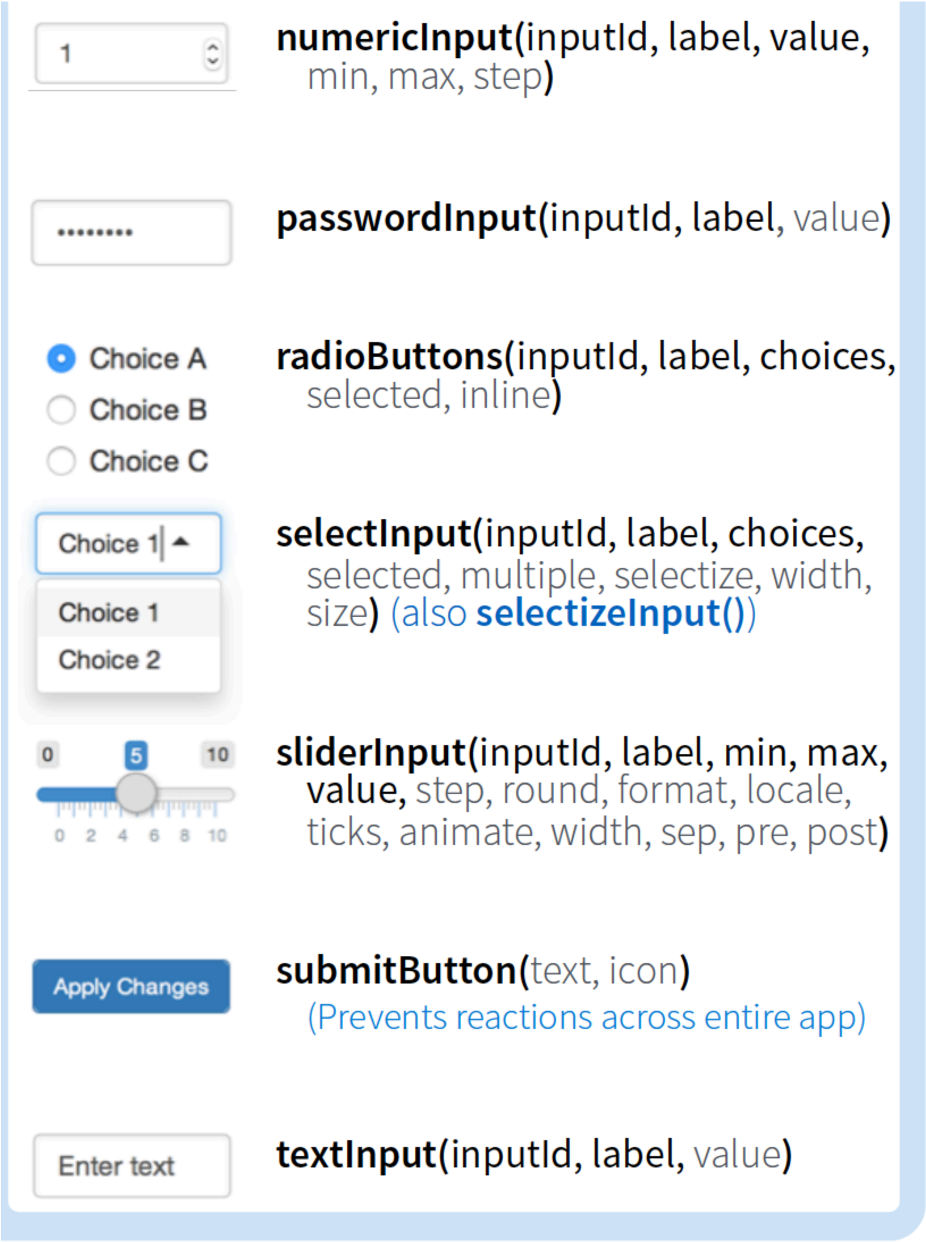
checkboxGroupInput(inputId, label, choices, selected, inline)

☑ 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

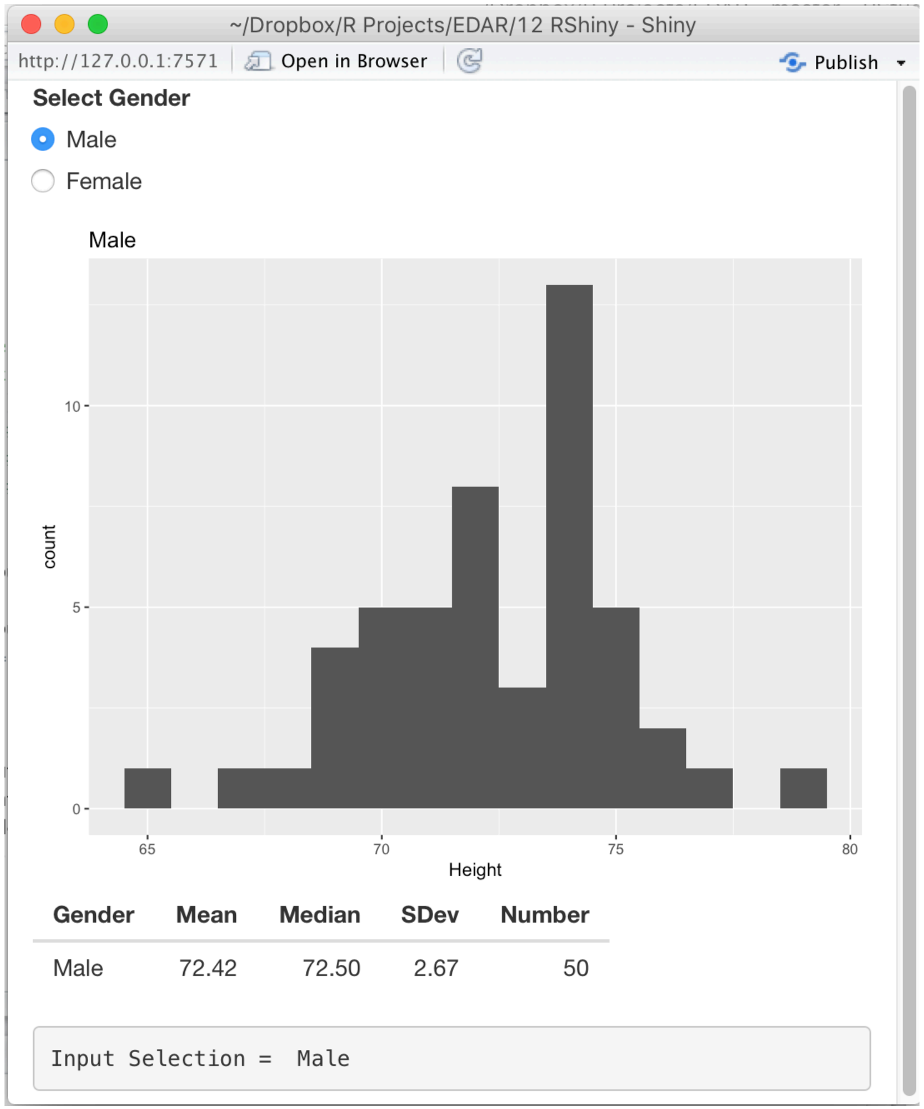
selectInput(inputId, label, choices, selected, multiple, selectize, width, size) (also **selectizeInput()**)

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…



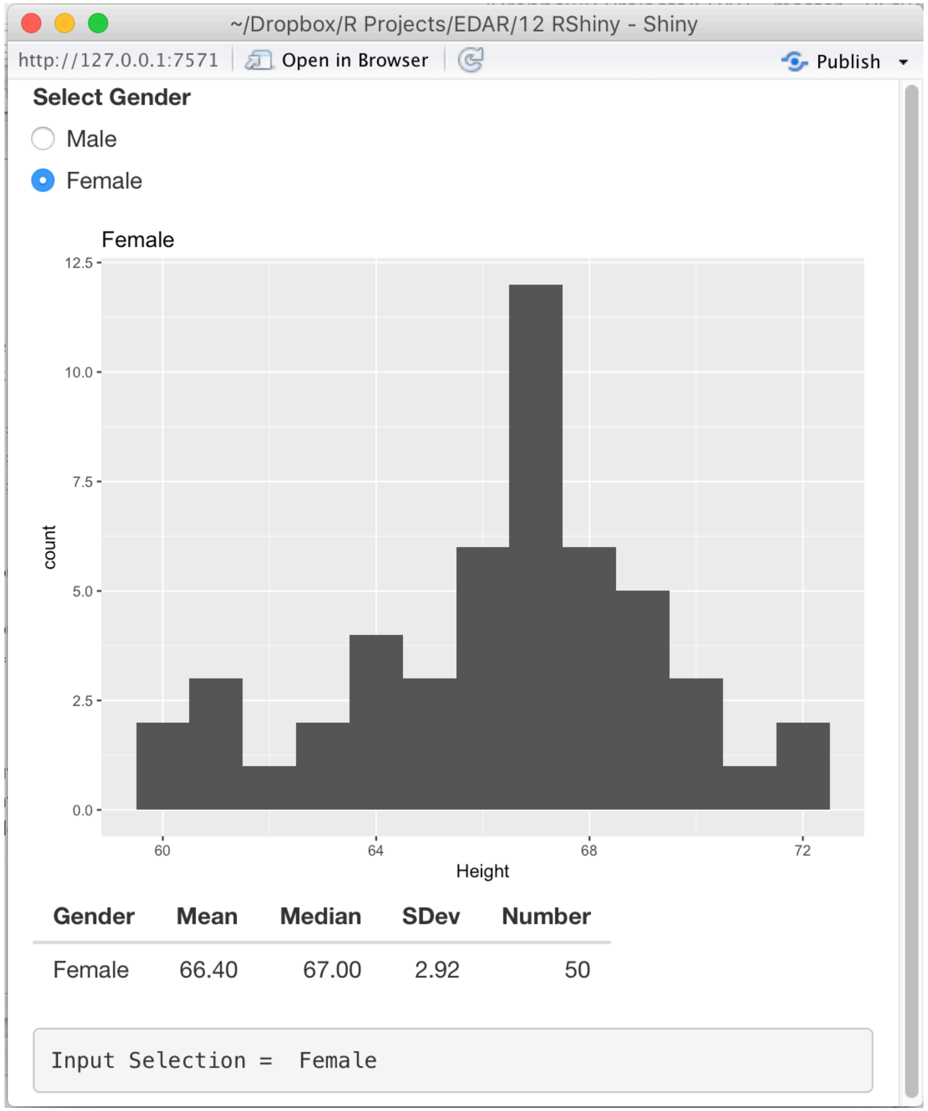
Select Gender

● Male

○ Female

Gender	Mean	Median	SDev	Number
Male	72.42	72.50	2.67	50

```
Input Selection =  Male
```



Select Gender

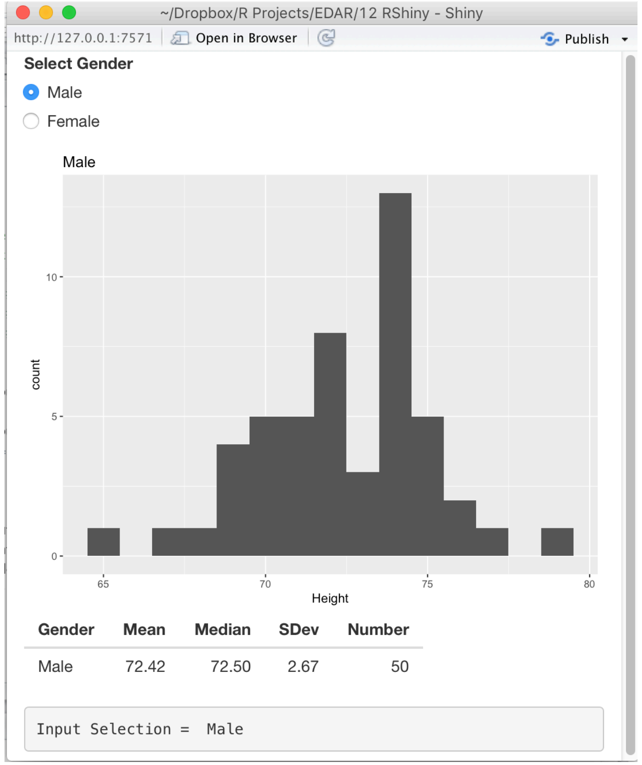
○ Male

● Female

Gender	Mean	Median	SDev	Number
Female	66.40	67.00	2.92	50

```
Input Selection =  Female
```

```r
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")
  })
}
```



Select Gender

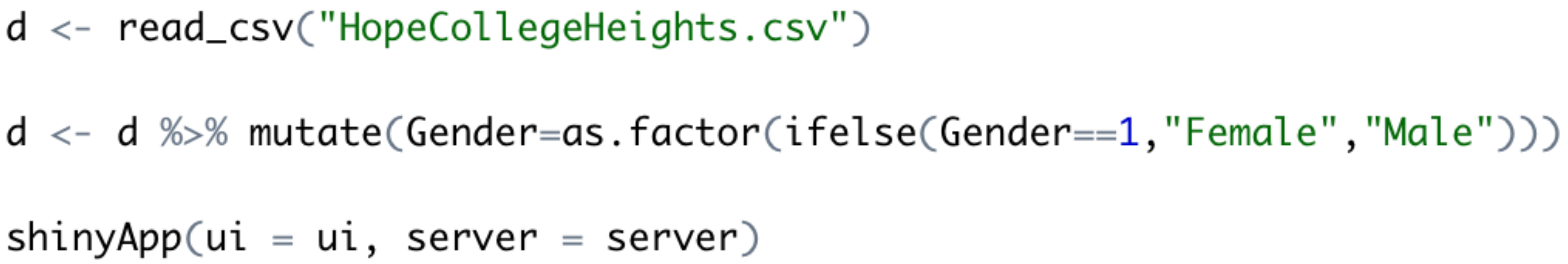
● Male

○ Female

Gender	Mean	Median	SDev	Number
Male	72.42	72.50	2.67	50

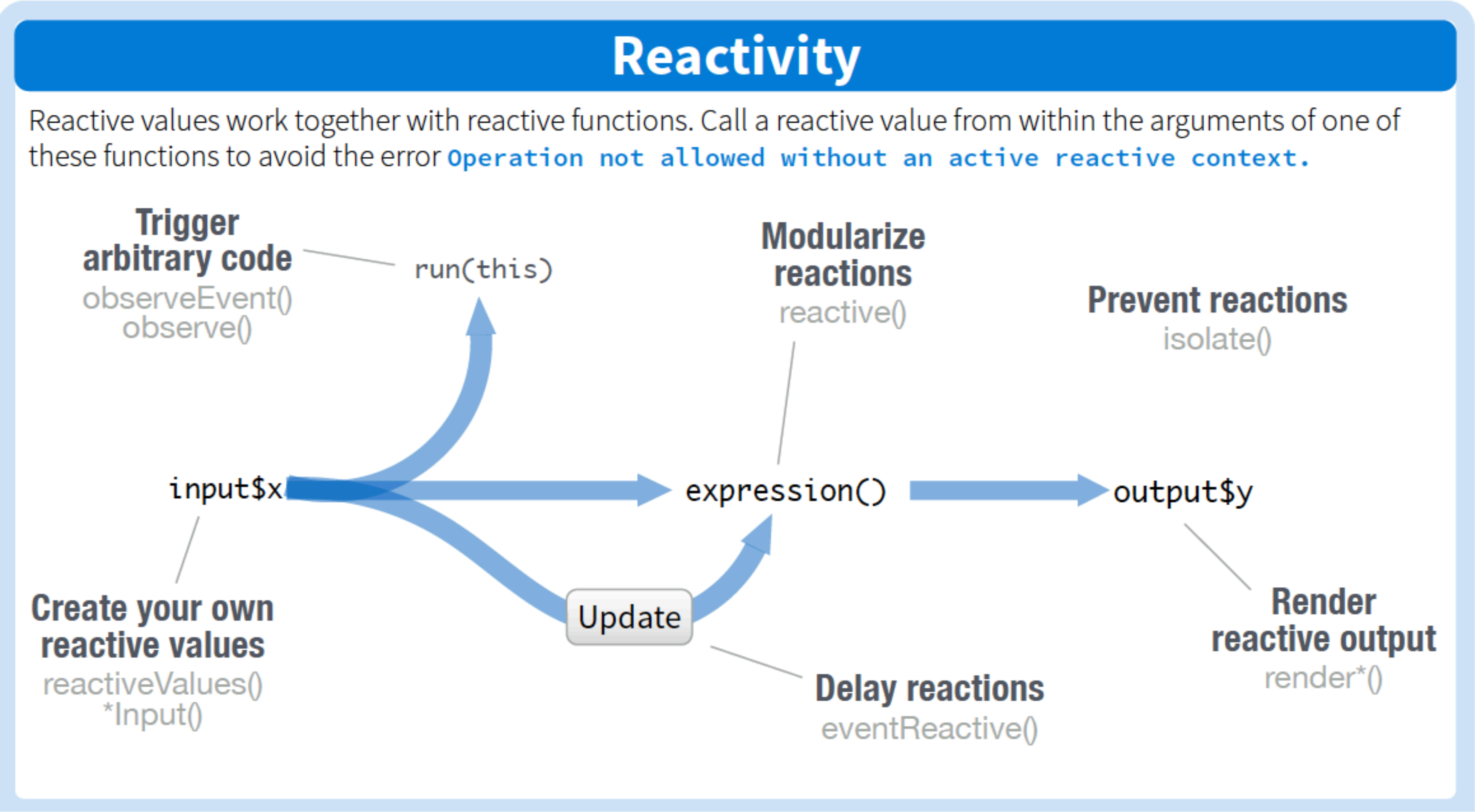
```
Input Selection =  Male
```

Setup code



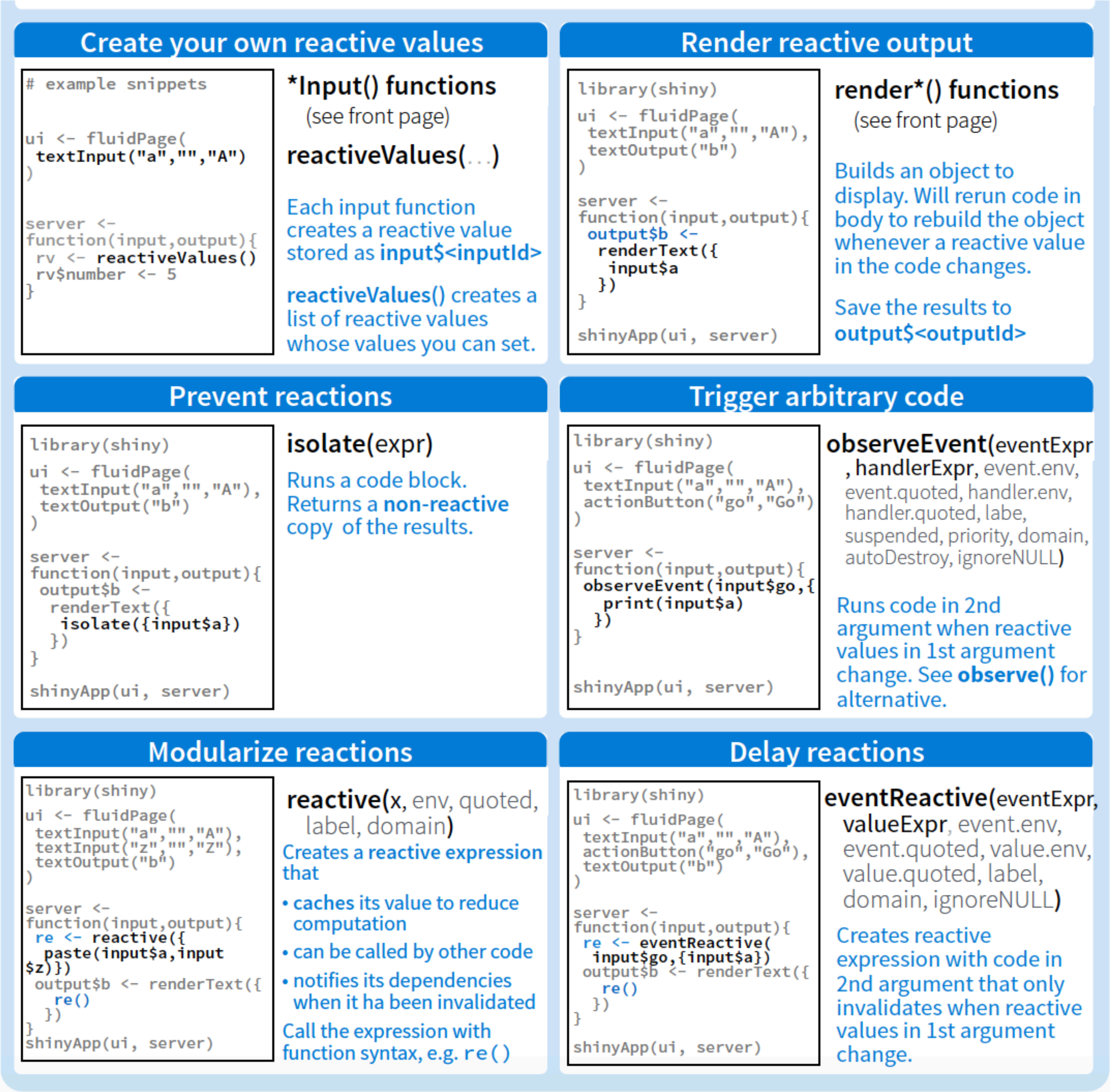
```
d <- read_csv("HopeCollegeHeights.csv")

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



Reactivity

Reactive values work together with reactive functions. Call a reactive value from within the arguments of one of these functions to avoid the error `Operation not allowed without an active reactive context.`



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

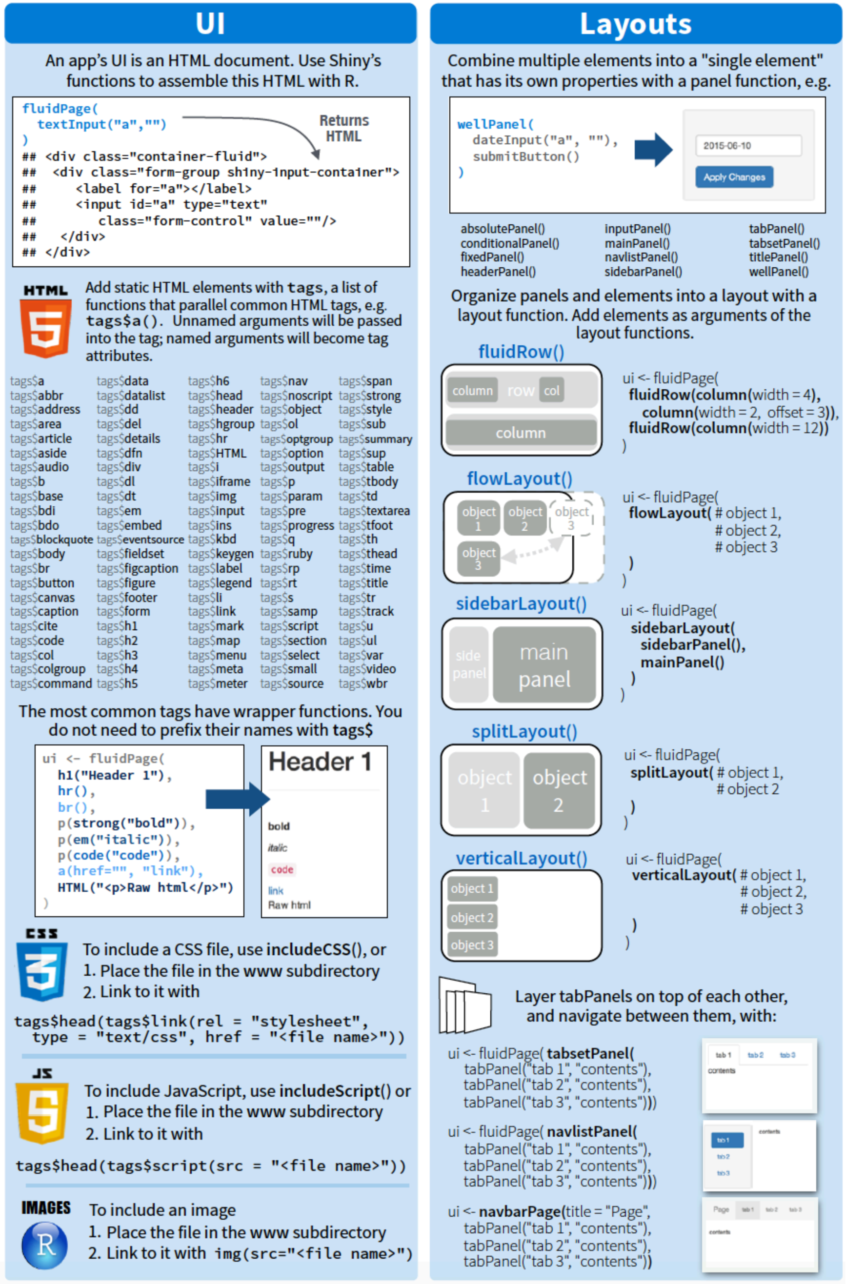
Call the expression with function syntax, e.g. `re()`

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$a
tags$abbr
tags$address
tags$area
tags$article
tags$aside
tags$audio
tags$b
tags$base
tags$bdi
tags$bdo
tags$blockquote
tags$body
tags$br
tags$button
tags$canvas
tags$caption
tags$cite
tags$code
tags$col
tags$colgroup
tags$command

tags$data
tags$datalist
tags$dd
tags$del
tags$details
tags$dfn
tags$div
tags$dl
tags$dt
tags$em
tags$embed
tags$eventsource
tags$fieldset
tags$figcaption
tags$figure
tags$footer
tags$form
tags$h1
tags$h2
tags$h3
tags$h4
tags$h5

tags$h6
tags$head
tags$header
tags$hgroup
tags$hr
tags$HTML
tags$i
tags$iframe
tags$img
tags$input
tags$ins
tags$kbd
tags$keygen
tags$label
tags$legend
tags$li
tags$link
tags$mark
tags$map
tags$menu
tags$meta
tags$meter

tags$nav
tags$noscript
tags$object
tags$ol
tags$optgroup
tags$option
tags$output
tags$p
tags$param
tags$pre
tags$progress
tags$q
tags$ruby
tags$rp
tags$rt
tags$s
tags$samp
tags$script
tags$section
tags$select
tags$small
tags$source

tags$span
tags$strong
tags$style
tags$sub
tags$summary
tags$sup
tags$table
tags$tbody
tags$td
tags$textarea
tags$tfoot
tags$th
tags$thead
tags$time
tags$title
tags$tr
tags$track
tags$u
tags$ul
tags$var
tags$video
tags$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()
conditionalPanel()
fixedPanel()
headerPanel()

inputPanel()
mainPanel()
navlistPanel()
sidebarPanel()

tabPanel()
tabsetPanel()
titlePanel()
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
  )
)
```

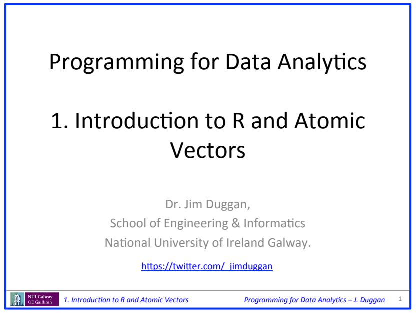
Layer tabPanels on top of each other, and navigate between them, with:

```
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 Overview



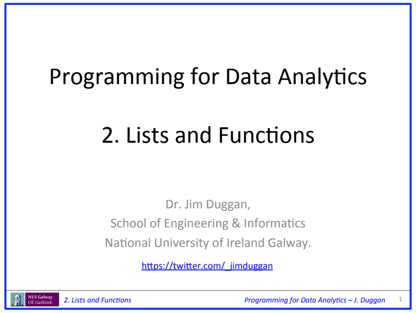
Programming for Data Analytics

1. Introduction to R and Atomic Vectors

Dr. Jim Duggan,
School of Engineering & Informatics
National University of Ireland Galway.

https://twitter.com/_jimduggan

NUI Galway OÉ Gaillimh 1. Introduction to R and Atomic Vectors Programming for Data Analytics – J. Duggan 1



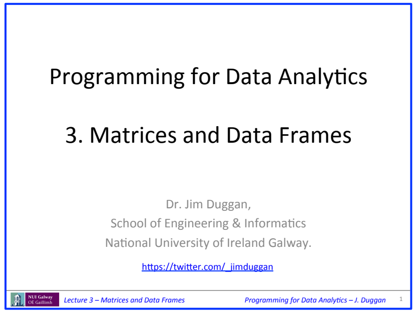
Programming for Data Analytics

2. Lists and Functions

Dr. Jim Duggan,
School of Engineering & Informatics
National University of Ireland Galway.

https://twitter.com/_jimduggan

NUI Galway OÉ Gaillimh 2. Lists and Functions Programming for Data Analytics – J. Duggan 1



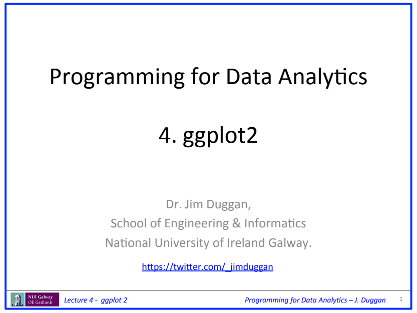
Programming for Data Analytics

3. Matrices and Data Frames

Dr. Jim Duggan,
School of Engineering & Informatics
National University of Ireland Galway.

https://twitter.com/_jimduggan

NUI Galway OÉ Gaillimh Lecture 3 – Matrices and Data Frames Programming for Data Analytics – J. Duggan 1



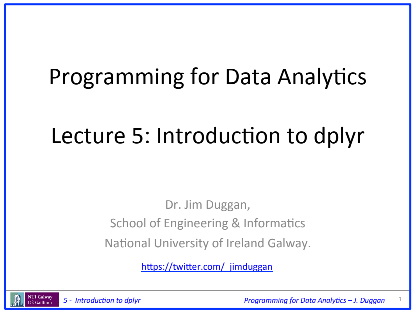
Programming for Data Analytics

4. ggplot2

Dr. Jim Duggan,
School of Engineering & Informatics
National University of Ireland Galway.

https://twitter.com/_jimduggan

NUI Galway OÉ Gaillimh Lecture 4 - ggplot 2 Programming for Data Analytics – J. Duggan 1



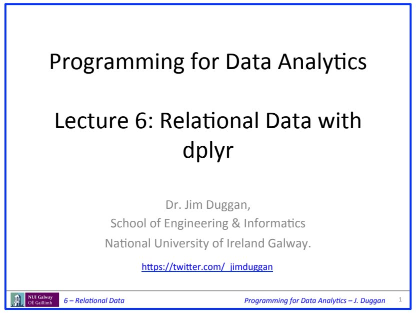
Programming for Data Analytics

Lecture 5: Introduction to dplyr

Dr. Jim Duggan,
School of Engineering & Informatics
National University of Ireland Galway.

https://twitter.com/_jimduggan

NUI Galway OÉ Gaillimh 5 - Introduction to dplyr Programming for Data Analytics – J. Duggan 1



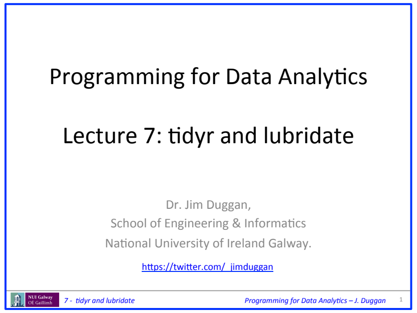
Programming for Data Analytics

Lecture 6: Relational Data with dplyr

Dr. Jim Duggan,
School of Engineering & Informatics
National University of Ireland Galway.

https://twitter.com/_jimduggan

NUI Galway OÉ Gaillimh 6 – Relational Data Programming for Data Analytics – J. Duggan 1



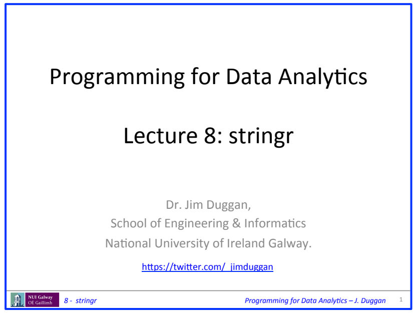
Programming for Data Analytics

Lecture 7: tidyr and lubridate

Dr. Jim Duggan,
School of Engineering & Informatics
National University of Ireland Galway.

https://twitter.com/_jimduggan

NUI Galway OÉ Gaillimh 7 - tidyr and lubridate Programming for Data Analytics – J. Duggan 1



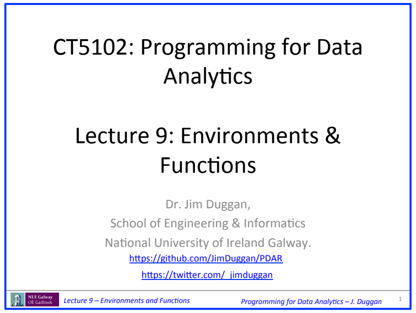
Programming for Data Analytics

Lecture 8: stringr

Dr. Jim Duggan,
School of Engineering & Informatics
National University of Ireland Galway.

https://twitter.com/_jimduggan

NUI Galway OÉ Gaillimh 8 - stringr Programming for Data Analytics – J. Duggan 1

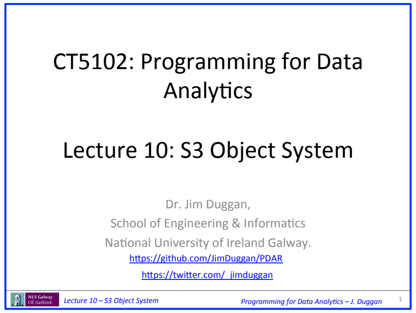


CT5102: Programming for Data Analytics

Lecture 9: Environments & Functions

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

NUI Galway OÉ Gaillimh Lecture 9 – Environments and Functions Programming for Data Analytics – J. Duggan 1

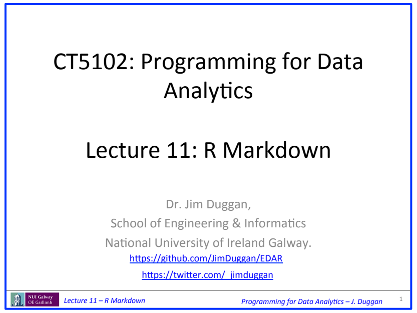


CT5102: Programming for Data Analytics

Lecture 10: S3 Object System

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

NUI Galway OÉ Gaillimh Lecture 10 – S3 Object System Programming for Data Analytics – J. Duggan 1

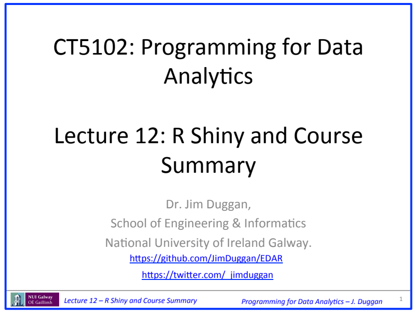


CT5102: Programming for Data Analytics

Lecture 11: R Markdown

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

NUI Galway OÉ Gaillimh Lecture 11 – R Markdown Programming for Data Analytics – J. Duggan 1



CT5102: Programming for Data Analytics

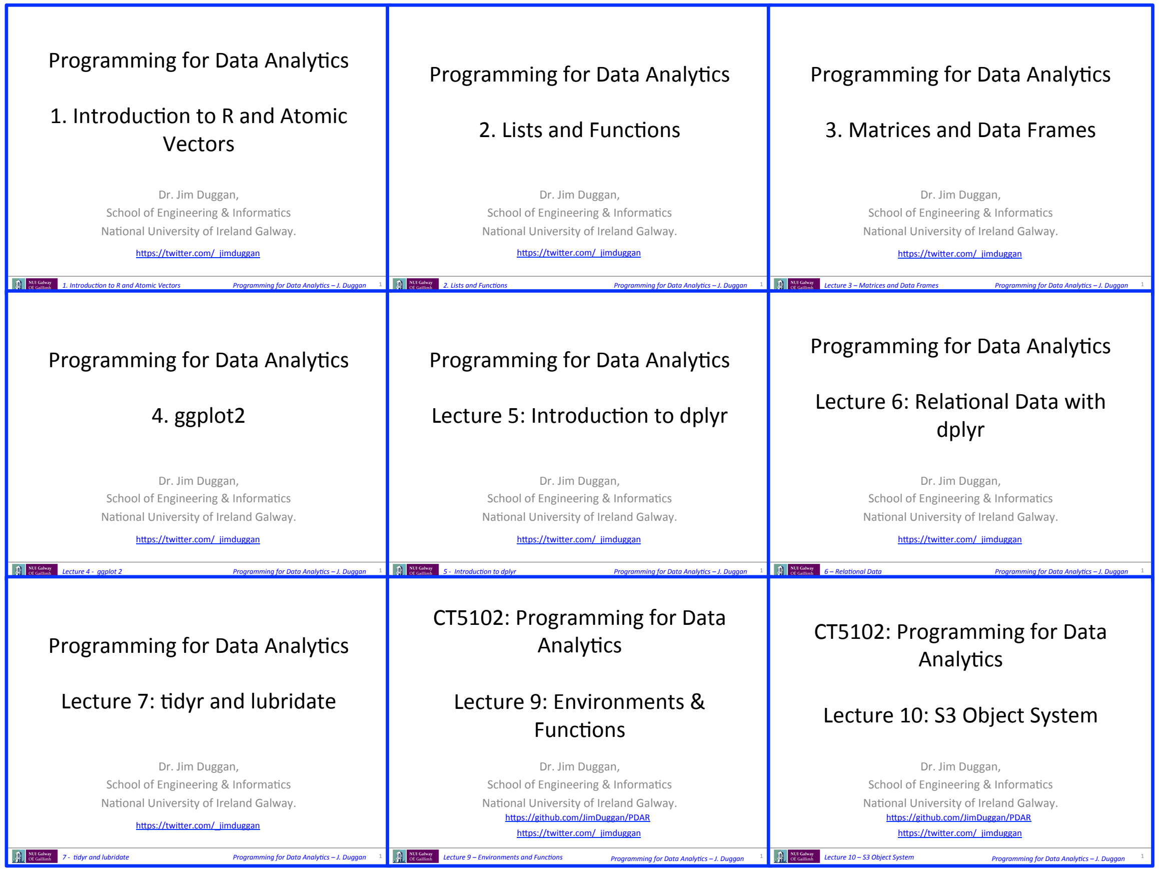
Lecture 12: R Shiny and Course Summary

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

NUI Galway OÉ Gaillimh Lecture 12 – R Shiny and Course Summary Programming for Data Analytics – J. Duggan 1

Exam Revision Summary

- 4 questions, do any 3
- Base R:
  - Vectors, Lists, Data Frames
  - Functions
  - Environments, Closures
  - S3
- tidyverse:
  - dplyr, tidyr, ggplot2
- Topics not covered:
  - stringr
  - lubridate
  - RMarkdown
  - RShiny



Programming for Data Analytics

1. Introduction to R and Atomic Vectors

Dr. Jim Duggan,
School of Engineering & Informatics
National University of Ireland Galway.

https://twitter.com/_jimduggan

NUI Galway OÉ Gaillimh 1. Introduction to R and Atomic Vectors Programming for Data Analytics – J. Duggan 1

Programming for Data Analytics

2. Lists and Functions

Dr. Jim Duggan,
School of Engineering & Informatics
National University of Ireland Galway.

https://twitter.com/_jimduggan

NUI Galway OÉ Gaillimh 2. Lists and Functions Programming for Data Analytics – J. Duggan 1

Programming for Data Analytics

3. Matrices and Data Frames

Dr. Jim Duggan,
School of Engineering & Informatics
National University of Ireland Galway.

https://twitter.com/_jimduggan

NUI Galway OÉ Gaillimh Lecture 3 – Matrices and Data Frames Programming for Data Analytics – J. Duggan 1

Programming for Data Analytics

4. ggplot2

Dr. Jim Duggan,
School of Engineering & Informatics
National University of Ireland Galway.

https://twitter.com/_jimduggan

NUI Galway OÉ Gaillimh Lecture 4 - ggplot 2 Programming for Data Analytics – J. Duggan 1

Programming for Data Analytics

Lecture 5: Introduction to dplyr

Dr. Jim Duggan,
School of Engineering & Informatics
National University of Ireland Galway.

https://twitter.com/_jimduggan

NUI Galway OÉ Gaillimh 5 - Introduction to dplyr Programming for Data Analytics – J. Duggan 1

Programming for Data Analytics

Lecture 6: Relational Data with dplyr

Dr. Jim Duggan,
School of Engineering & Informatics
National University of Ireland Galway.

https://twitter.com/_jimduggan

NUI Galway OÉ Gaillimh 6 – Relational Data Programming for Data Analytics – J. Duggan 1

Programming for Data Analytics

Lecture 7: tidyr and lubridate

Dr. Jim Duggan,
School of Engineering & Informatics
National University of Ireland Galway.

https://twitter.com/_jimduggan

NUI Galway OÉ Gaillimh 7 - tidyr and lubridate Programming for Data Analytics – J. Duggan 1

CT5102: Programming for Data Analytics

Lecture 9: Environments & Functions

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

NUI Galway OÉ Gaillimh Lecture 9 – Environments and Functions Programming for Data Analytics – J. Duggan 1

CT5102: Programming for Data Analytics

Lecture 10: S3 Object System

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

NUI Galway OÉ Gaillimh Lecture 10 – S3 Object System Programming for Data Analytics – J. Duggan 1