Lecture 26: Nov 12, 2018

# Shiny

- *Lists*
- *Interactivity and Shiny*
- *Components*
  - *New Shiny App* and *Sketching Apps*
  - *UI* and *Server*
- *Debugging*
- *Resources*

James Balamuta
STAT 385 @ UIUC

# Announcements

- **Group Proposal** due **Friday, November 16th** at **11:59 PM**
- **hw08** due **Friday, November 16th** at **6:00 PM**
- **Quiz 12** covers Week 11 contents @ CBTF.
  - Window: Nov 13th - Nov 15th
  - Sign up: https://cbtf.engr.illinois.edu/sched
- Want to review your homework or quiz grades?
  **Schedule an appointment.**

# Last Time

- **Regular Expressions ("regex")**
  - Find, extract, or replace patterns in strings.
- **Using Regex**
  - Metacharacters provide ways to generate dynamic patterns to match inside a string.
  - Be wary of *greedy* vs. *lazy* capture
- **Resources**
  - Cheatsheets, regex pattern builders, and more!

# Lecture Objectives

- **Create** and **modify** a list data structure.
- **Describe** the relationship between user interfaces and server.
- **Apply** the principles of reactivity.
- **Design** a user interface to explore data.

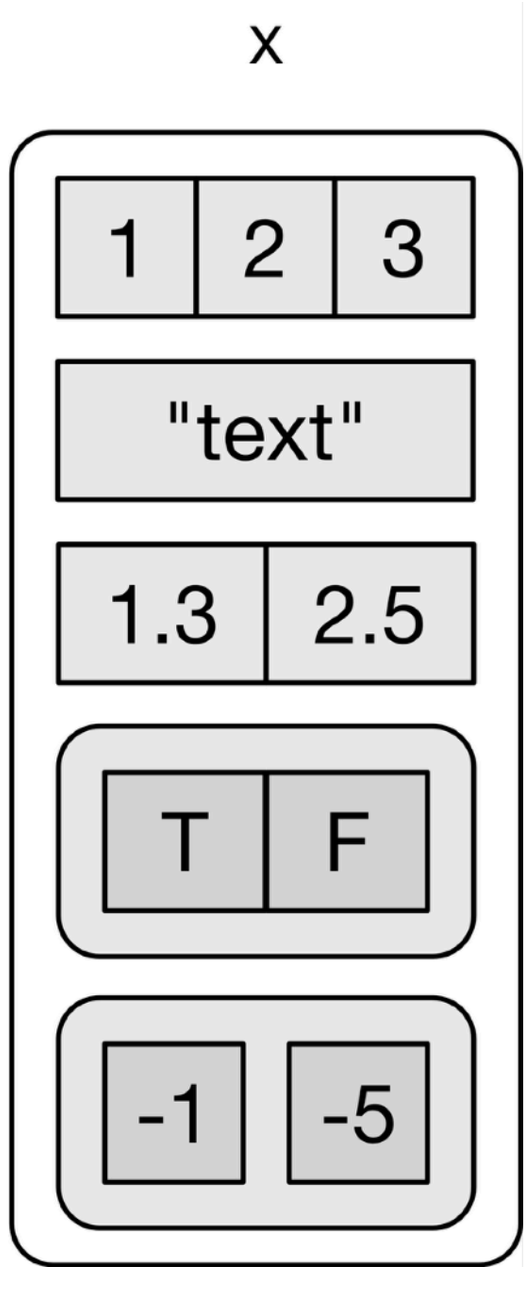
# Lists

Previously

# List

… **1 D**imensional *Heterogenous (mixed)* data …

```
x = list(c(1, 2, 3),
         "text",
          c(1.3, 2.5),
          list(c(TRUE, FALSE)),
          list(c(-1), c(-5))
)

length(x)
# [1] 5

dim(x)
# NULL
```



# Mixed Return Type

… returning multiple structures / types…

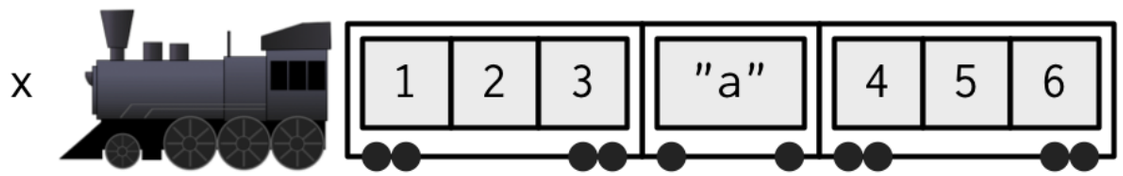
```
# Define a function that merges
# inputs
return_list = function(a, b, c) {
  list(element1 = a, toad = b, c)
}

# Combine different types
out =
  return_list(1:3, c("a", "b"),
                   c(2 + 3i, 4 - 1i))

out
# $element1
# [1] 1 2 3
# $toad
# [1] "a" "b"
# [[3]]
# [1] 2+3i 4-1i
```

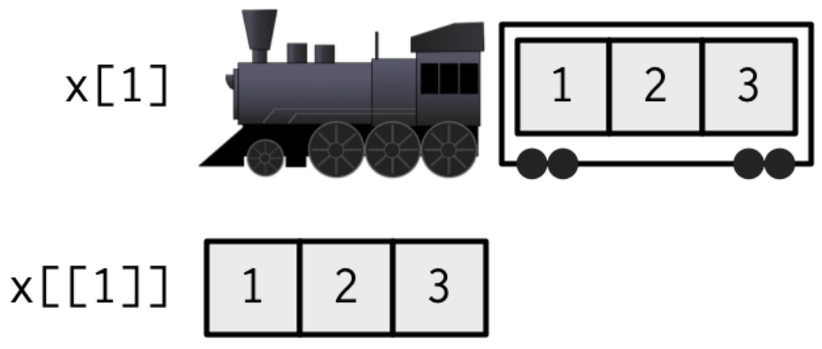
**Named Elements**

**Position Index**

# Preserving vs. Simplifying

… subset differences with trains …




**List**

**Vector**

```
# Define a list with different elements
x = list(1:3, "a", 4:6)

# View list
x
# [[1]]
# [1] 1 2 3
# [[2]]
# [1] "a"
# [[3]]
# [1] 4 5 6
```

```
# Single brackets retain the
# list structure around item
x[1]
# [[1]]
# [1] 1 2 3
```

```
# Double bracket remove the
# list structure around item
x[[1]]
# [1] 1 2 3
```

Source

**Double brackets** or **$name**
gives an *atomic vector*

**Single brackets**
gives a *list with vectors inside*

# Subset Rules

… general rules for subset behaviors …

| | **Simplifying** | **Preserving** |
|---|---|---|
| **Vectors** | x[[1]] | x[1] |
| **Lists** | x[[1]]<br>x$name | x[1] |
| **Array** | x[1, ] **or** x[, 1] | x[1, , drop = F] **or**<br>x[, 1, drop = F] |
| **Data Frames** | x[, 1] **or** x[[1]] | x[, 1, drop = F] **or** x[1] |

# Interactivity

# Main Workflow

... generates a static text report ...

# How can we allow for a **quicker iteration of ideas?**

# Explorable Explanations

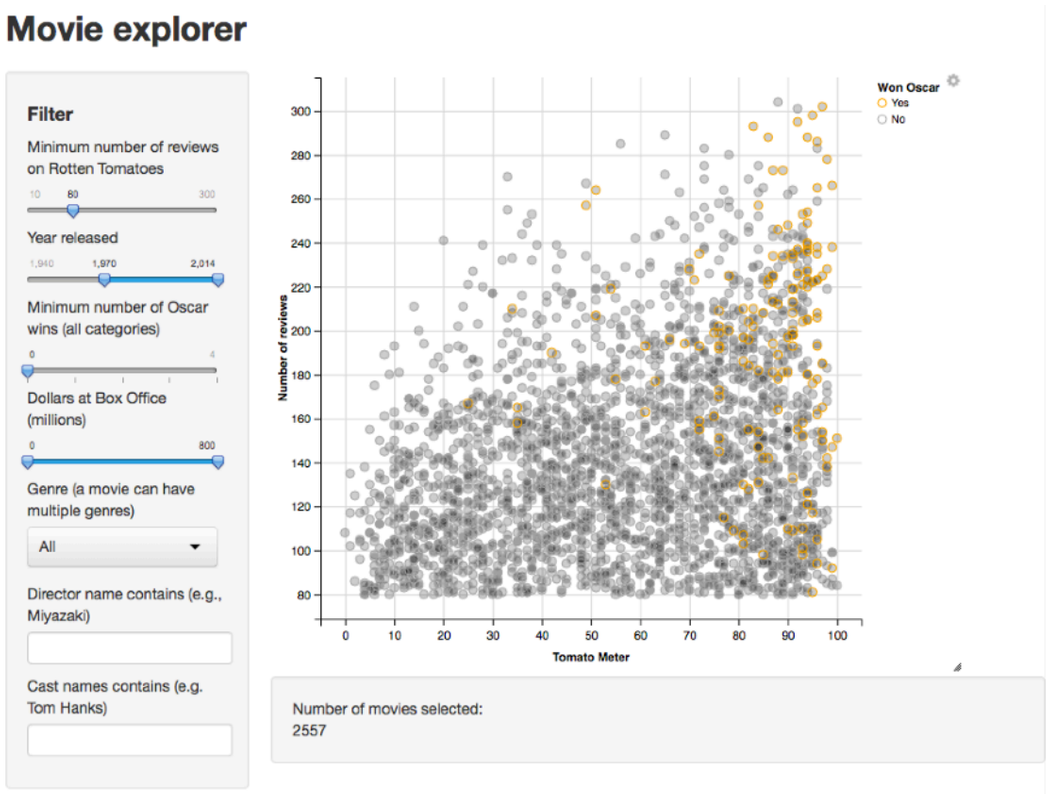
… facilitating interactivity …

Send **Output Result** from *R* to *Web Browser*

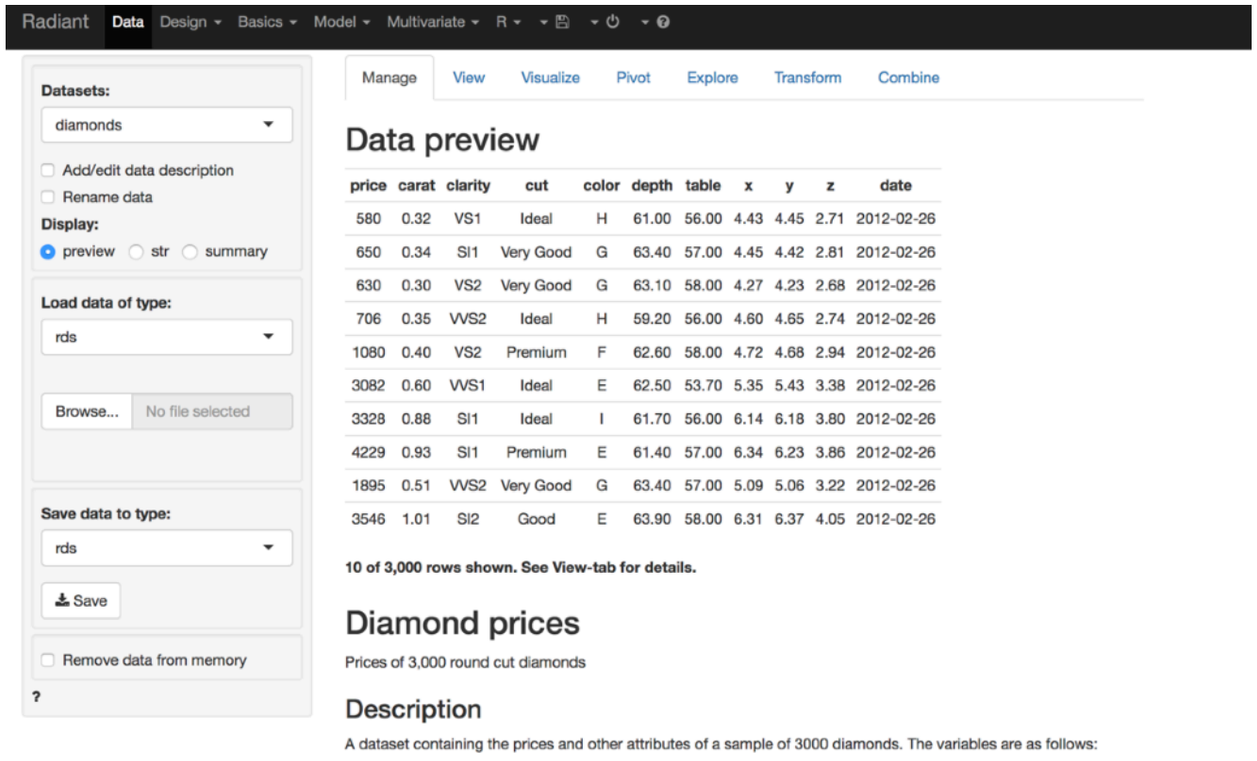
Send **User Input** from *Web Browser* to *R*

# Shiny

## … facilitating *R* to Web Browser interactions since '12 …

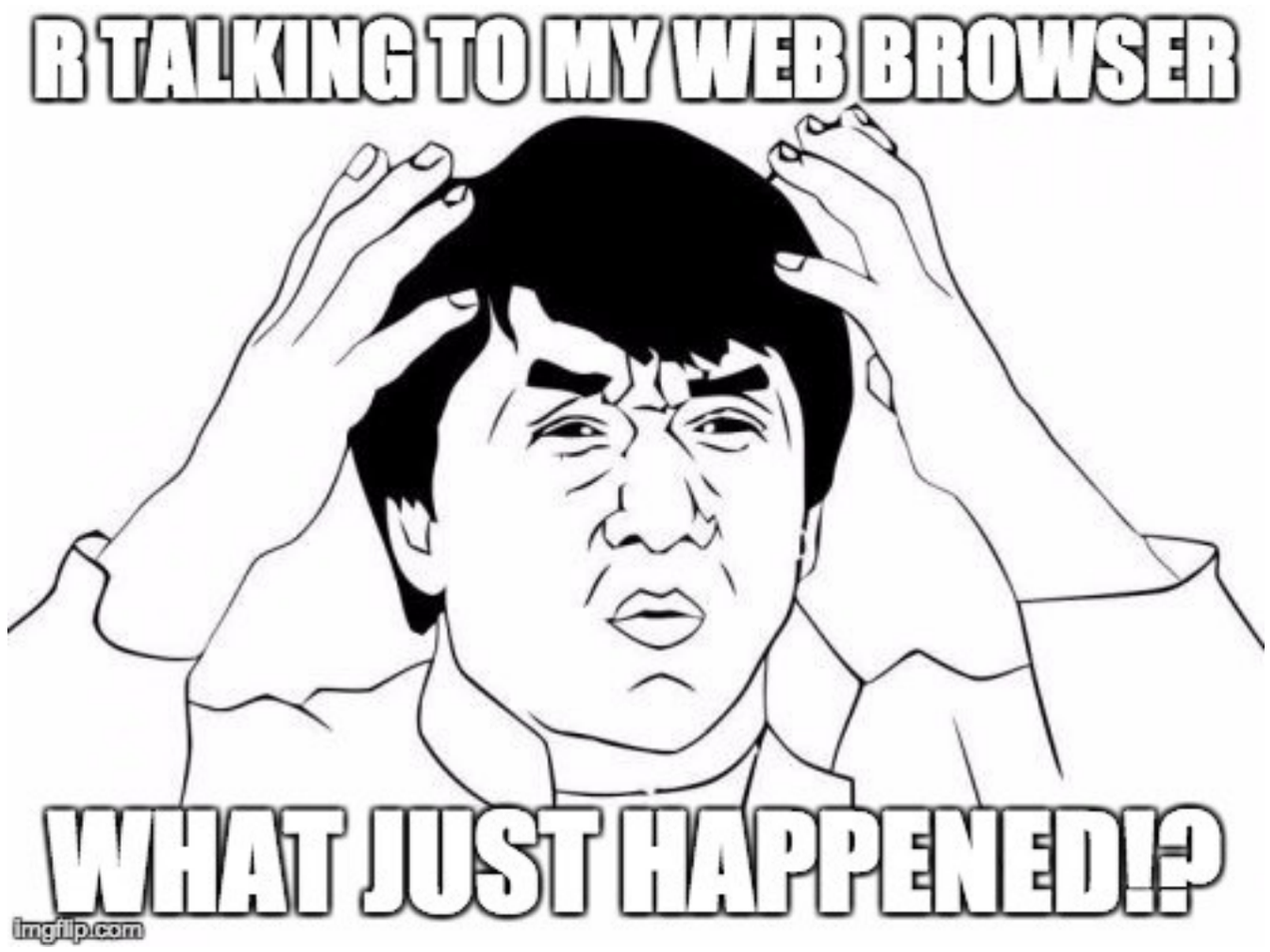



Shiny Gallery: Movie explorer




Radiant by Vincent Nijs is an open-source platform-independent browser-based interface for business analytics in R.

# Lions, Tigers, and Bears… Oh my!



# Components

# Backbone of Shiny

… UI and Server components …


**UI** is responsible for providing the User Interface (UI) or frontend for the shiny application.

**Server** is responsible for providing the backend logic behind each change that occurs due to an action on the frontend.

# Barebones

... minimum required for shiny ...

```
# Backend logic
ui = fluidPage(
      # Make a page layout
)

# Backend logic
server = function(input, output) {
   # Retrieve values from input
   # Save values to output.
   # Both are lists that are
   # accessed with dollar signs
   # e.g. input$ and output$
}

# Launch the App
shinyApp(ui = ui, server = server)
```

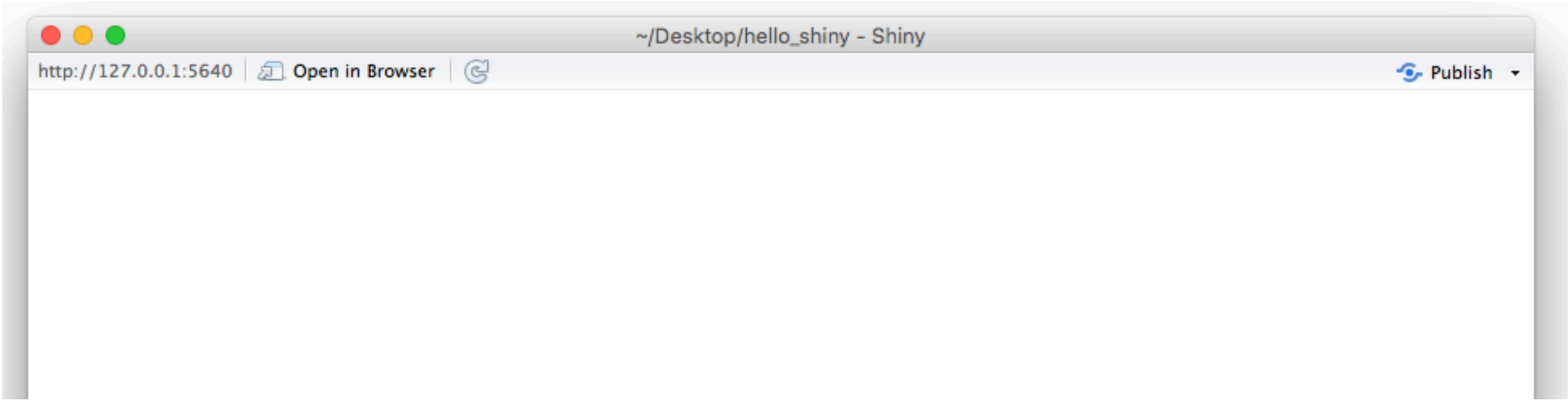
Specify a UI template

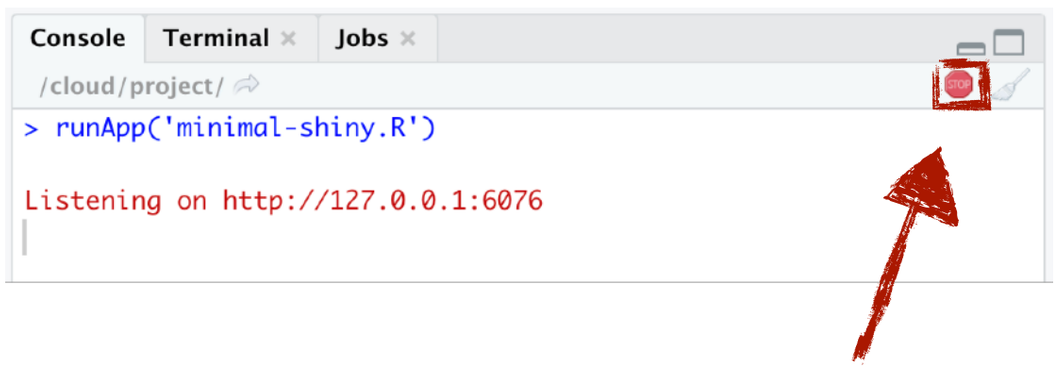
Define a Function with two pre-named arguments

Call function with above values to launch app

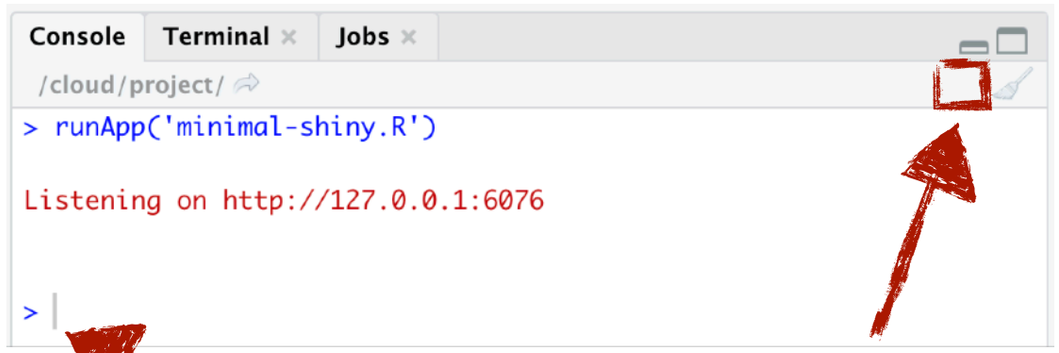
# Output

## … a master piece …




**Stop App**

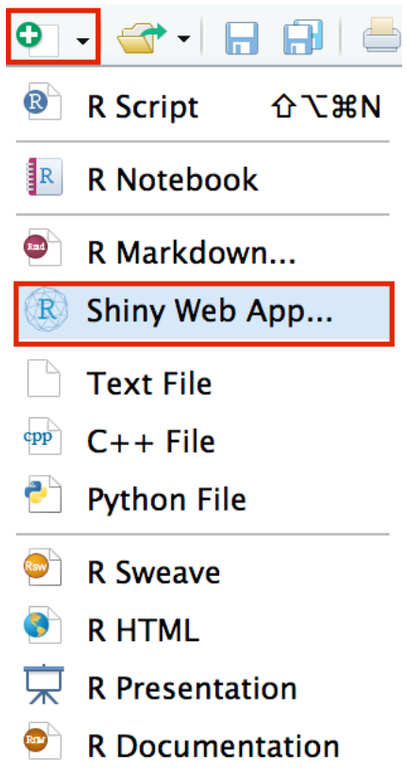


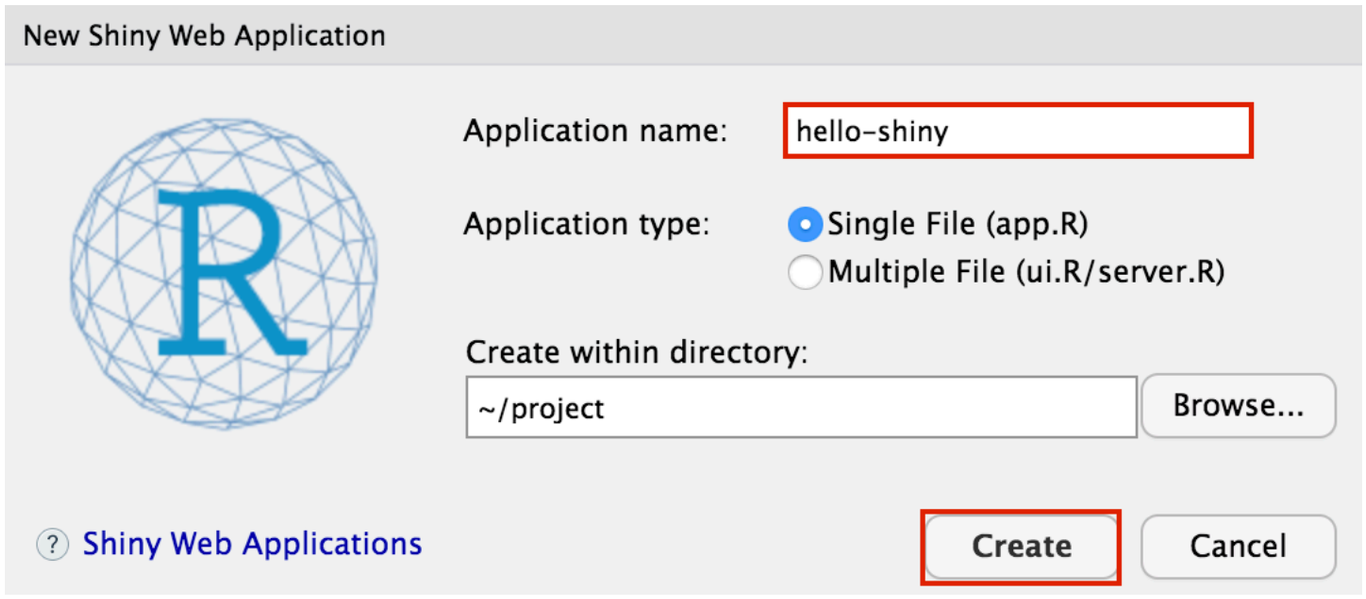

**Normal R session**

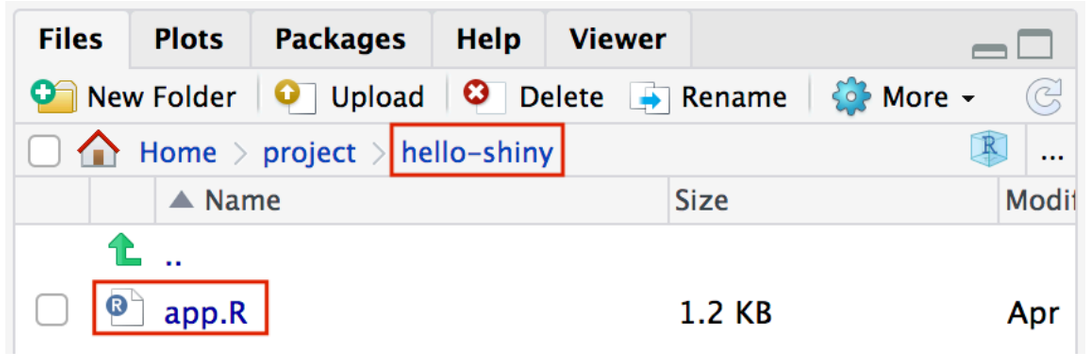
# New Shiny App

# Making a Shiny App

## … setting up RStudio to host a shiny development environment …



# Source + Live App

… note the "Run App" in source…

## Source Panel

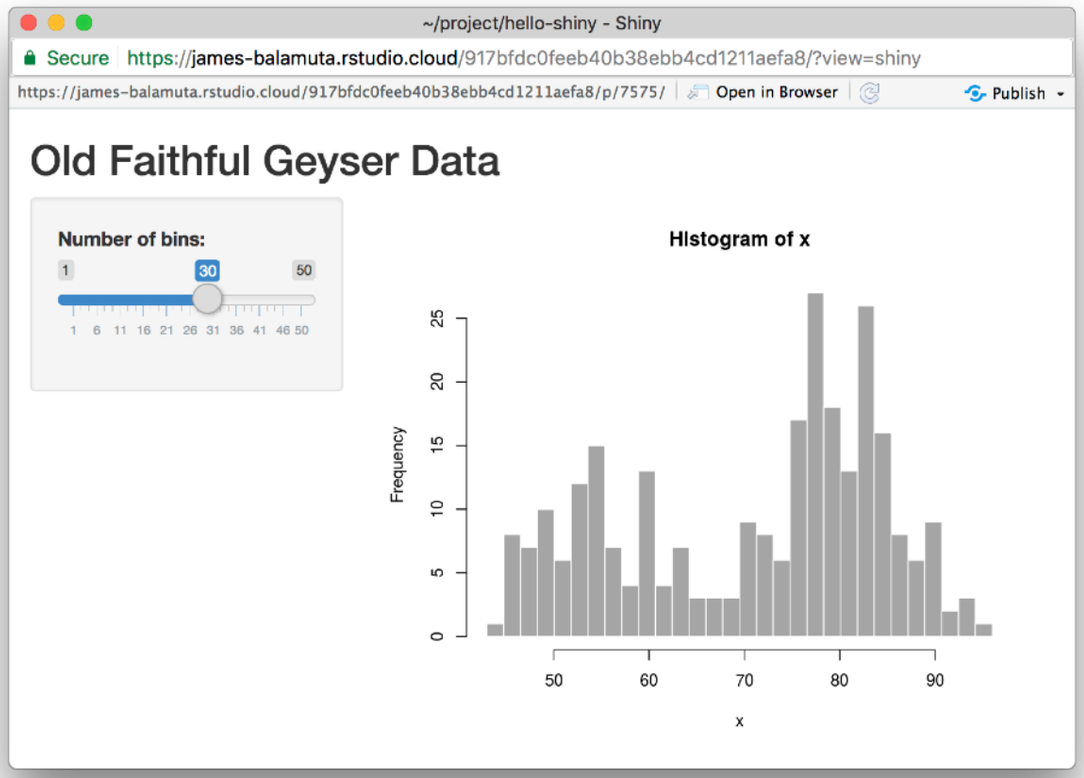
app.R

Run App

```
1  #
2  # This is a Shiny web application. You can run the application by clicking
3  # the 'Run App' button above.
4  #
5  # Find out more about building applications with Shiny here:
6  #
7  #    http://shiny.rstudio.com/
8  #
9
10 library(shiny)
11
12 # Define UI for application that draws a histogram
13 ui <- fluidPage(
14
15    # Application title
16    titlePanel("Old Faithful Geyser Data"),
17
18    # Sidebar with a slider input for number of bins
19    sidebarLayout(
```

46:2 (Top Level) R Script

## Live App




Console Terminal Jobs

/cloud/project/

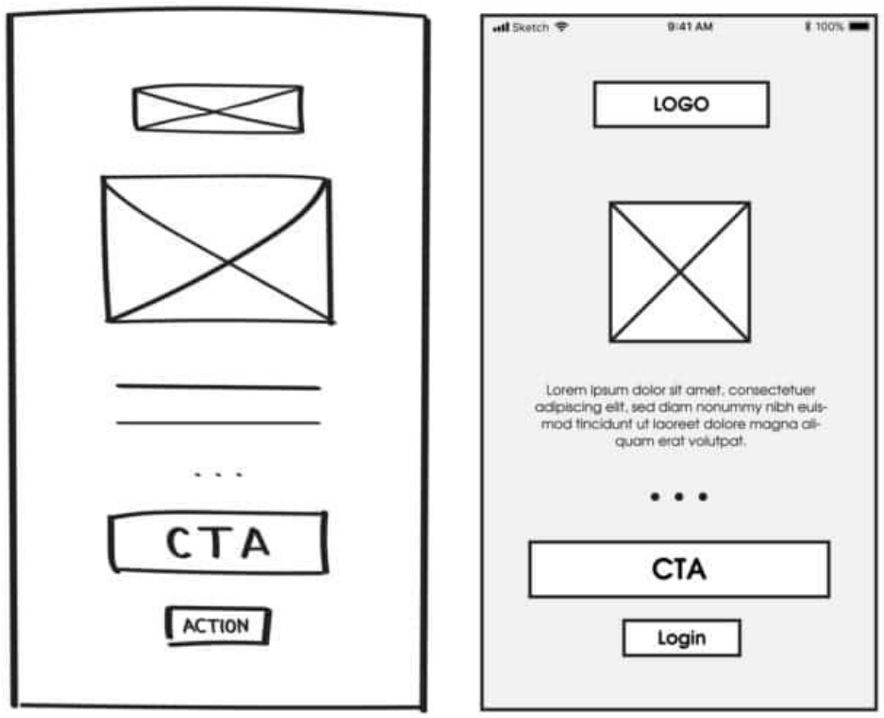
```
> shiny::runApp('demo')
Loading required package: shiny

Listening on http://127.0.0.1:6076
```

## Stop App

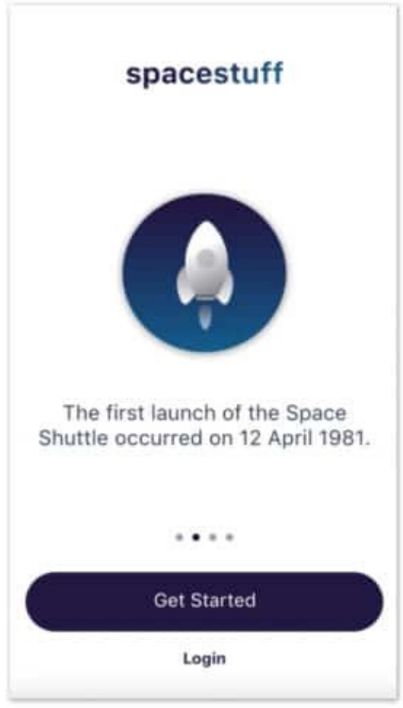
# Sketching Apps

**Definition:**
*Prototypes* are filters that traverse a design space and are manifestations of ideas. (Lim et al. ACM TOCHI, 15 (2), 2008.)

## Prototypes

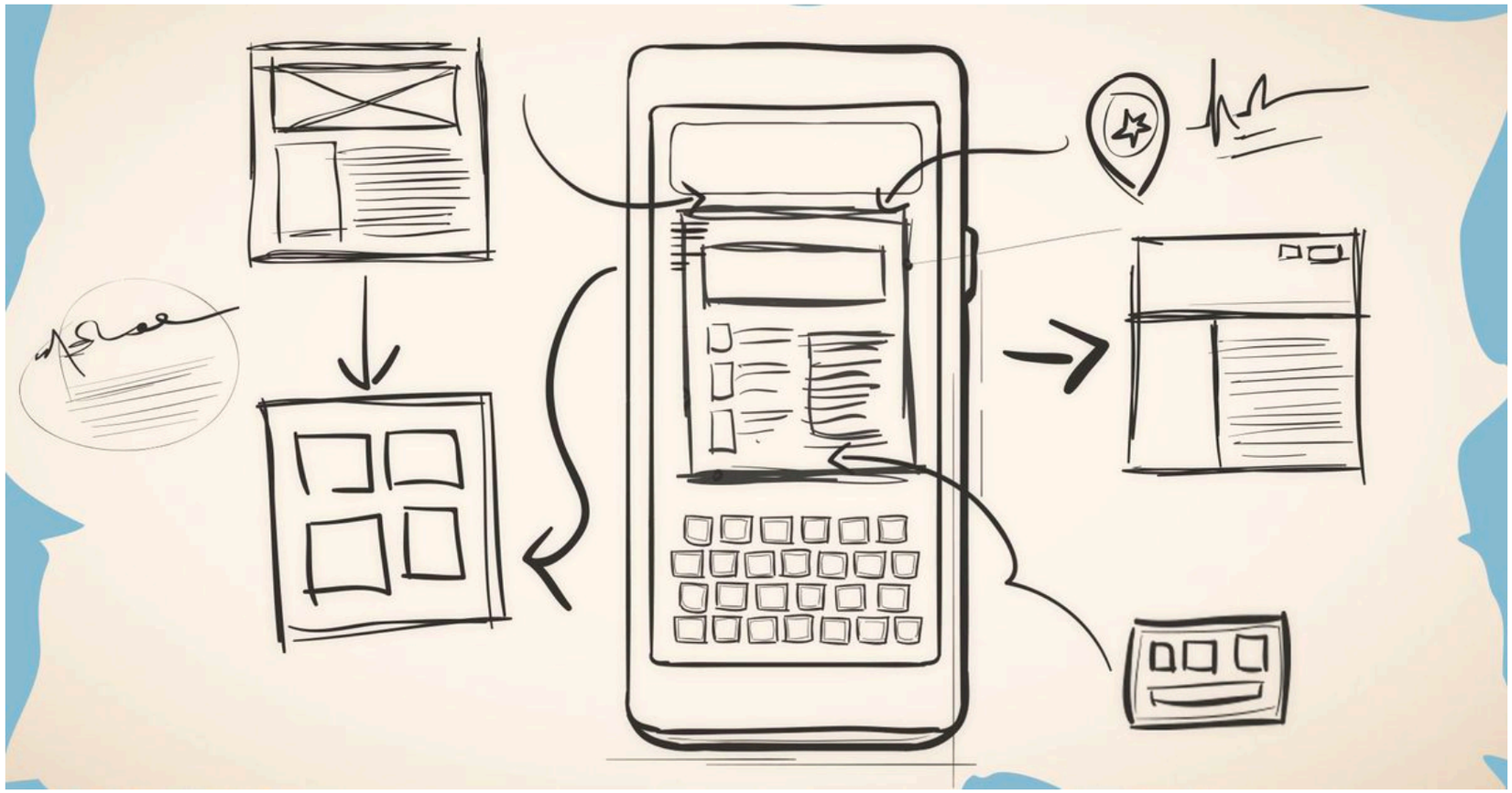



## Final Product




Source

**Definition:**
*Low Fidelity Prototypes* are prototypes that trade **realism** for **iterability** by using paper sketches and story boards to convey the desired purpose.



Source

**Definition:**
*High Fidelity Prototypes* are prototypes that focus on a user experience that is as **realistic** as possible by implementing the idea.

Sketch


Interactive

Source

# Cost of a Prototype

… when to go with one prototype over another …

**Early Design**

Low-fidelity

Medium-fidelity

High-fidelity

Quick Iteration

Extended Use Cases

**Late Design**

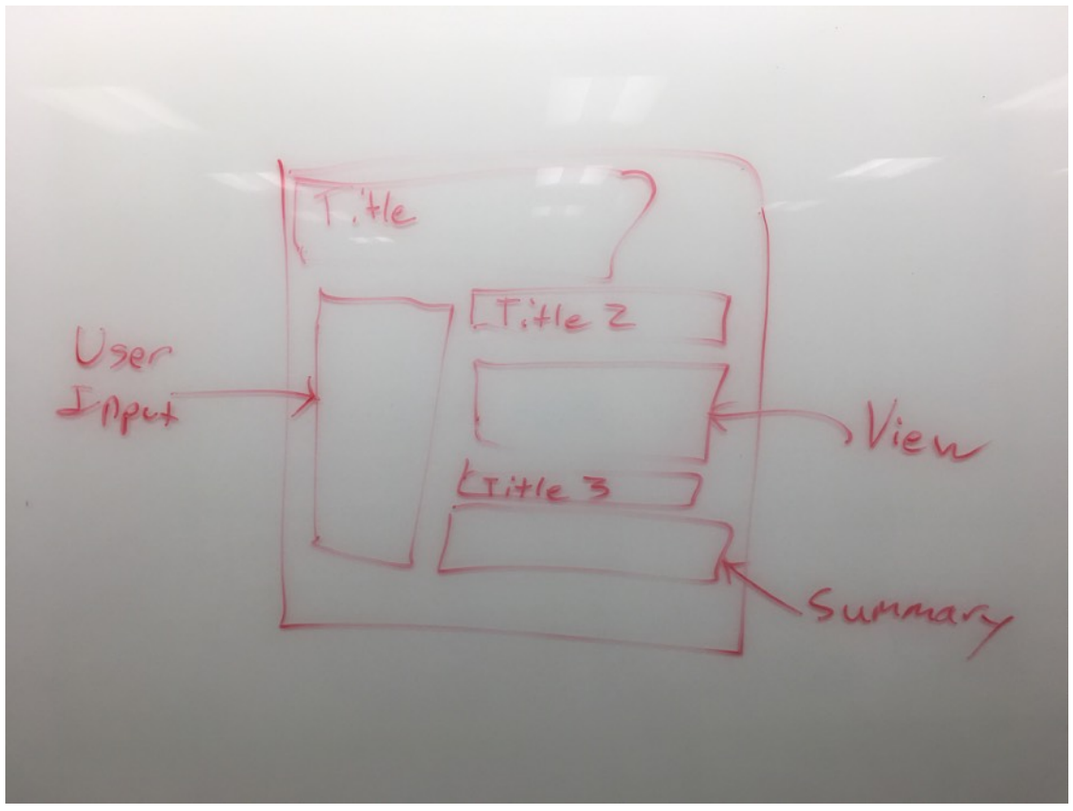
# Your Turn

**Scenario:** Imagine you want to show how often highway accidents occurs throughout different states and counties.

**Goal:** Sketch as many ideas as possible to enable the data to be displayed and interacted with.

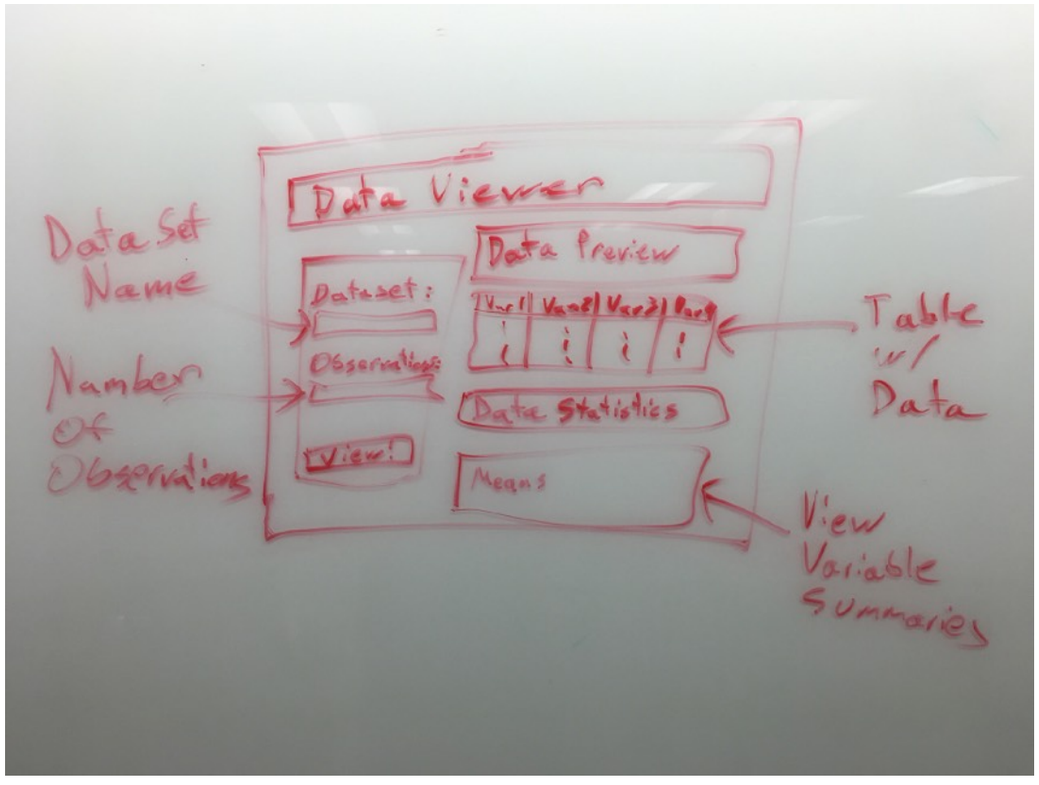
# Thinking about an Idea

## … sample sketches for a Data Viewer …




**Iteration 1**

(Scribble Sketch)

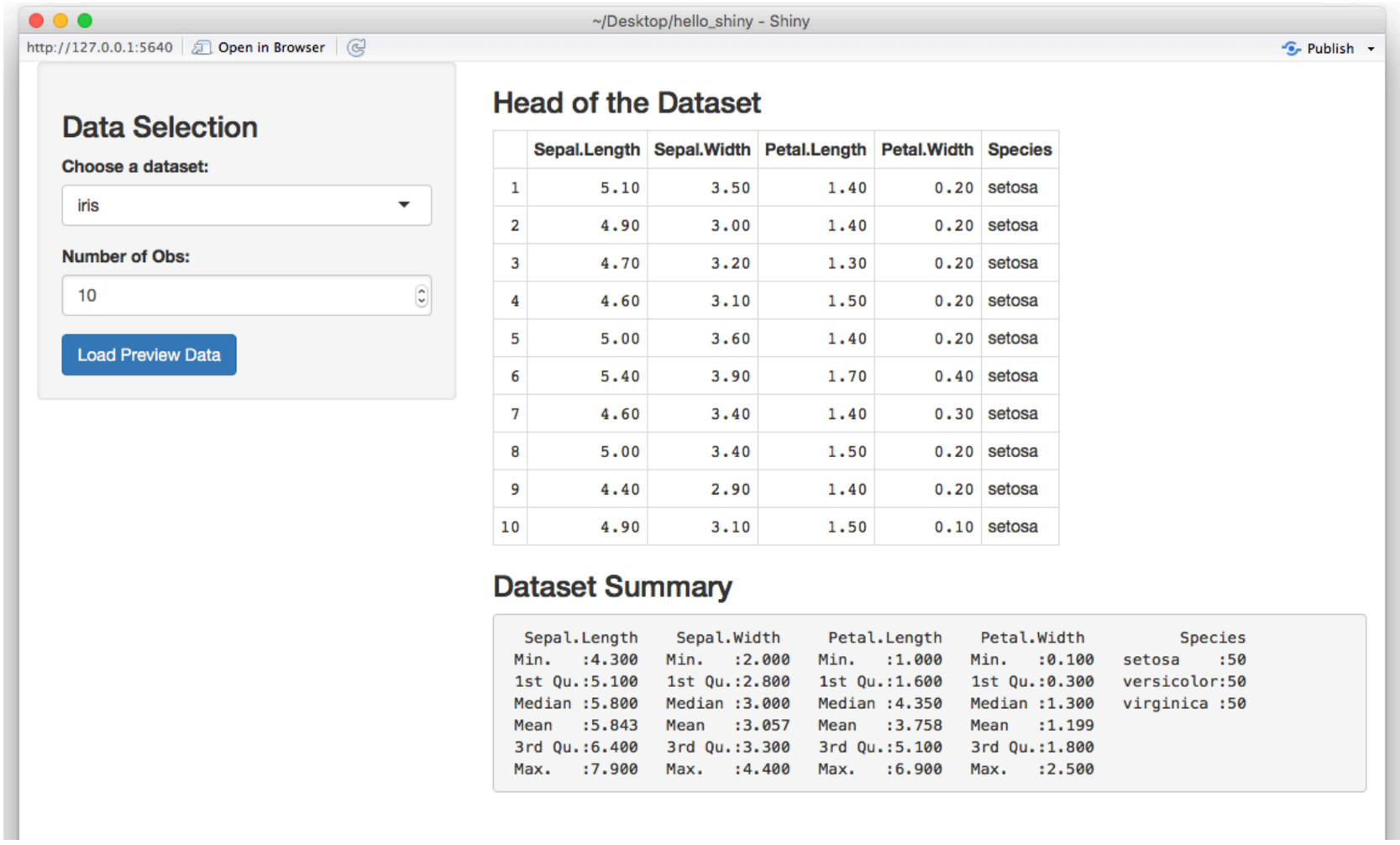



**Iteration 2**

(Paper Prototype)

# Goal

## … aim of this lecture …




| | Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
|---|---|---|---|---|---|
| 1 | 5.10 | 3.50 | 1.40 | 0.20 | setosa |
| 2 | 4.90 | 3.00 | 1.40 | 0.20 | setosa |
| 3 | 4.70 | 3.20 | 1.30 | 0.20 | setosa |
| 4 | 4.60 | 3.10 | 1.50 | 0.20 | setosa |
| 5 | 5.00 | 3.60 | 1.40 | 0.20 | setosa |
| 6 | 5.40 | 3.90 | 1.70 | 0.40 | setosa |
| 7 | 4.60 | 3.40 | 1.40 | 0.30 | setosa |
| 8 | 5.00 | 3.40 | 1.50 | 0.20 | setosa |
| 9 | 4.40 | 2.90 | 1.40 | 0.20 | setosa |
| 10 | 4.90 | 3.10 | 1.50 | 0.10 | setosa |

```
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa    :50
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   versicolor:50
 Median :5.800   Median :3.000   Median :4.350   Median :1.300   virginica :50
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
```

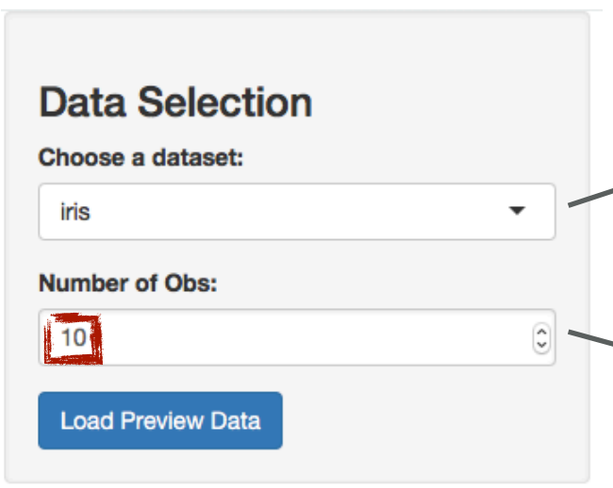
# Behind the Scenes

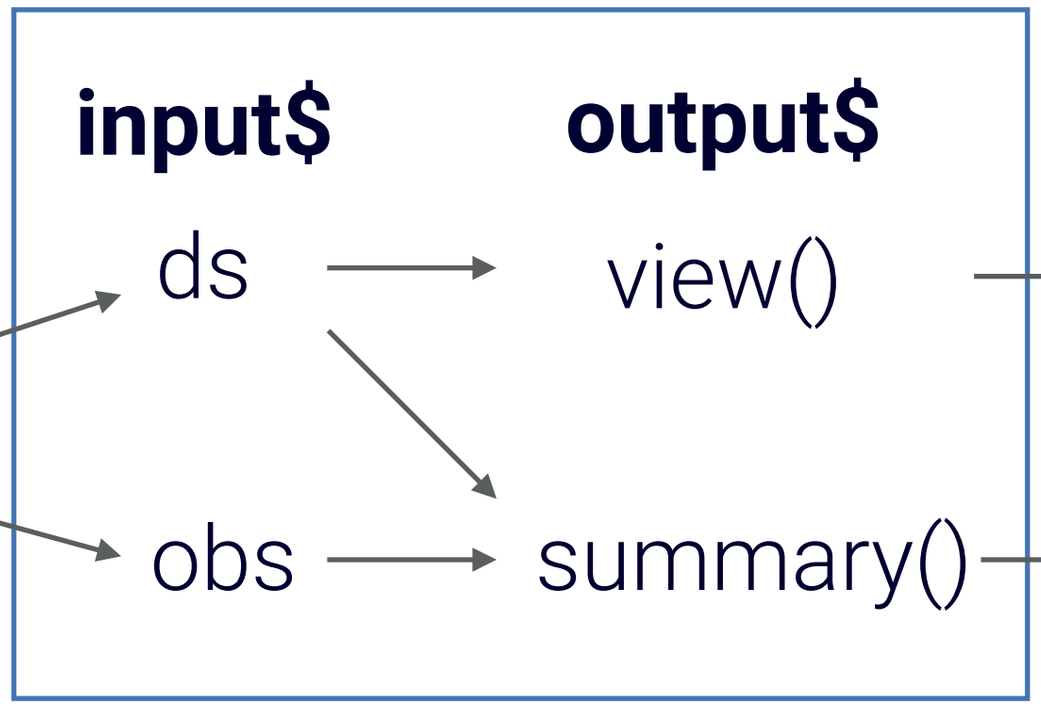
… breakdown of components …

## *Server* Logic

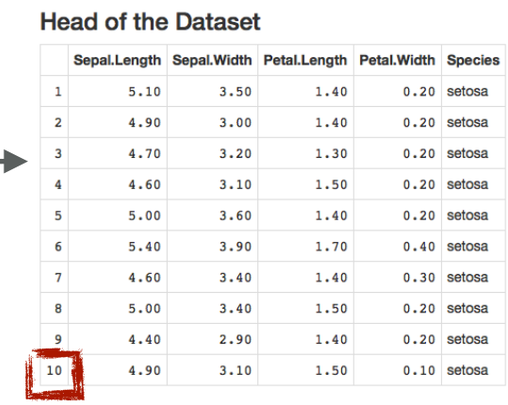
*hidden*

## *UI* Input




## *UI* Display



**Head of the Dataset**

| | Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
|---|---|---|---|---|---|
| 1 | 5.10 | 3.50 | 1.40 | 0.20 | setosa |
| 2 | 4.90 | 3.00 | 1.40 | 0.20 | setosa |
| 3 | 4.70 | 3.20 | 1.30 | 0.20 | setosa |
| 4 | 4.60 | 3.10 | 1.50 | 0.20 | setosa |
| 5 | 5.00 | 3.60 | 1.40 | 0.20 | setosa |
| 6 | 5.40 | 3.90 | 1.70 | 0.40 | setosa |
| 7 | 4.60 | 3.40 | 1.40 | 0.30 | setosa |
| 8 | 5.00 | 3.40 | 1.50 | 0.20 | setosa |
| 9 | 4.40 | 2.90 | 1.40 | 0.20 | setosa |
| 10 | 4.90 | 3.10 | 1.50 | 0.10 | setosa |

**Dataset Summary**

| Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
|---|---|---|---|---|
| Min. :4.300 | Min. :2.000 | Min. :1.000 | Min. :0.100 | setosa :50 |
| 1st Qu.:5.100 | 1st Qu.:2.800 | 1st Qu.:1.600 | 1st Qu.:0.300 | versicolor:50 |
| Median :5.800 | Median :3.000 | Median :4.350 | Median :1.300 | virginica :50 |
| Mean :5.843 | Mean :3.057 | Mean :3.758 | Mean :1.199 | |
| 3rd Qu.:6.400 | 3rd Qu.:3.300 | 3rd Qu.:5.100 | 3rd Qu.:1.800 | |
| Max. :7.900 | Max. :4.400 | Max. :6.900 | Max. :2.500 | |

# Strategy

… overall strategy …

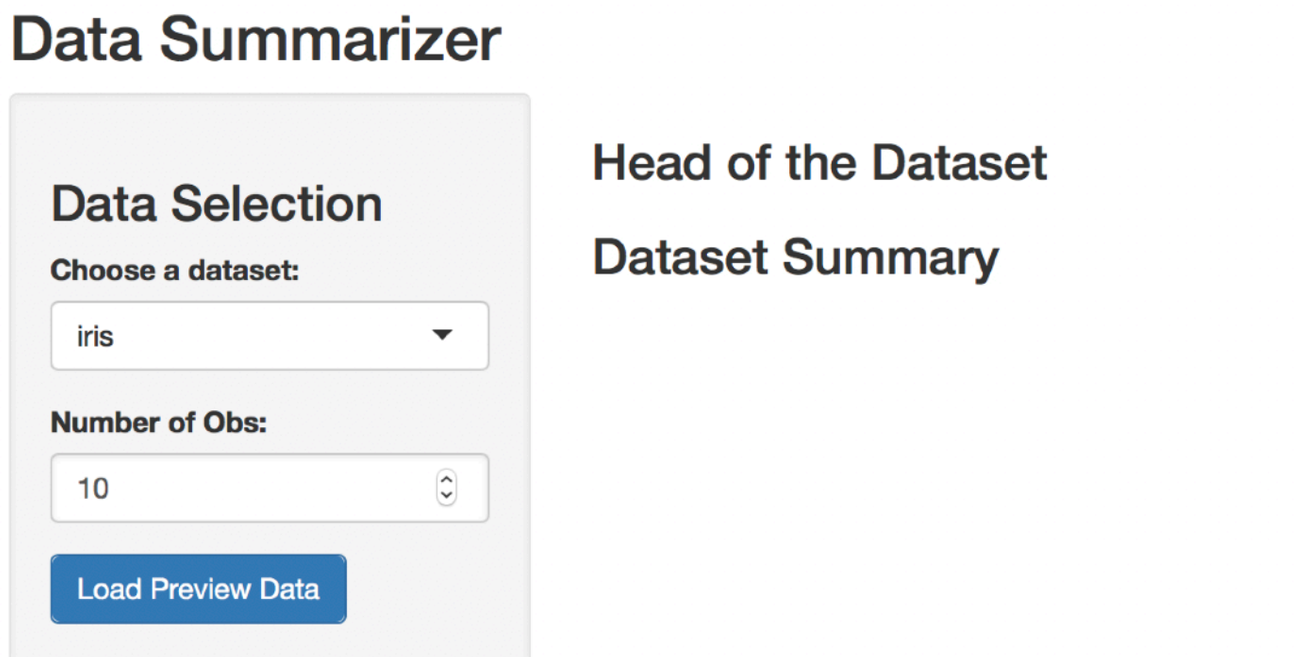
1. Establish the *layouts* for the **UI**
2. Allow user input in **UI** via *Control Widgets*
3. Implement *reactive* logic in **Server** to update areas in the **UI**.

# UI

# UI Strategy

… creating the interface …

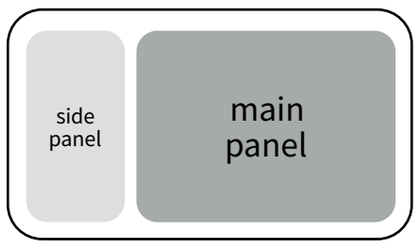
1. Create a layout that allows us to implement a **side bar** and a **main panel**
2. Side bar must have two **inputs** to access user responses.
3. Main panel must have two regions for **outputs** that *R* can send results to.



# Layouts

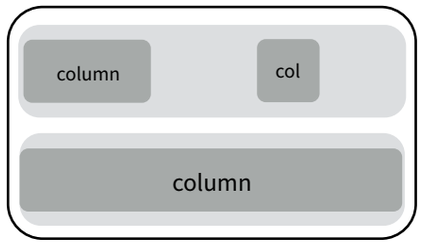
## … applying styling to UI …

### sidebarLayout()



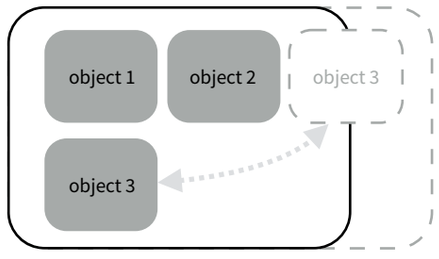

```
ui = fluidPage(
  sidebarLayout(
    sidebarPanel(),
    mainPanel()
  )
)
```

### fluidRow()



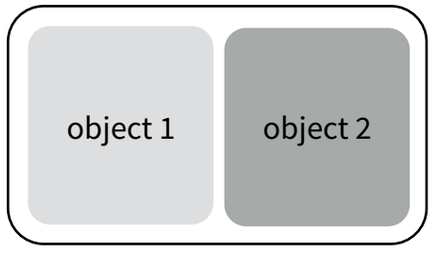

```
ui = fluidPage(
  fluidRow(column(width = 4),
     column(width = 2,  offset = 3)),
  fluidRow(column(width = 12))
)
```

### flowLayout()



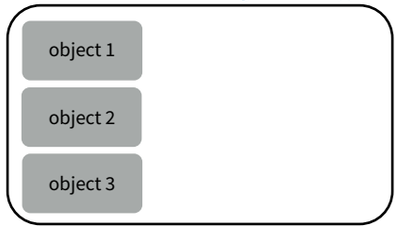

```
ui = fluidPage(
  flowLayout( # object 1,
              # object 2,
              # object 3
  )
)
```

### splitLayout()




```
ui = fluidPage(
  splitLayout( # object 1,
               # object 2
  )
)
```

### verticalLayout()




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

# Pagination Layouts

## … splitting content up …


```
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"))
```

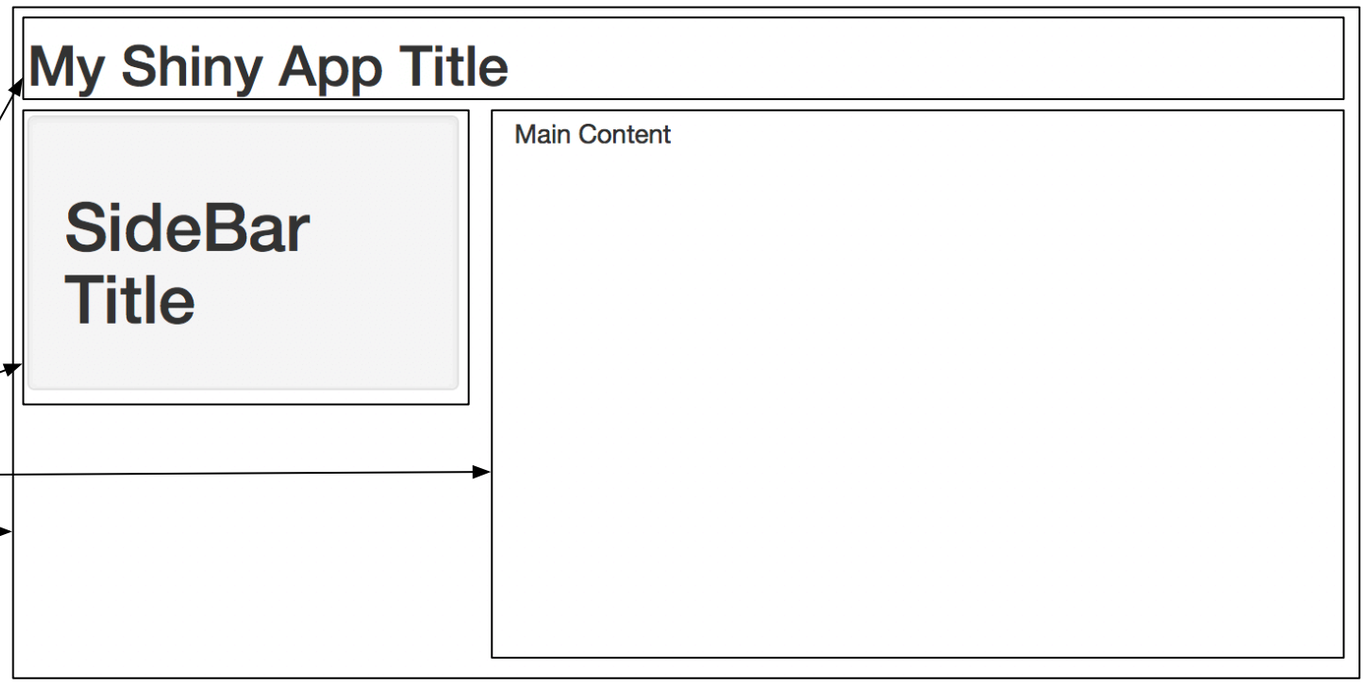
# UI Implemented

## … using a layout in practice …

app.R

Run App

```
ui = shinyUI(
  fluidPage(
    titlePanel("My Shiny App Title"), # Title

    sidebarLayout(

      sidebarPanel(
        h1("SideBar Title")           # Sidebar Text
      ),

      mainPanel("Main Content")       # Content Text
    ) # close: sidebarLayout()
  ) # close: fluidPage()
) # close: shinyUI()
```



# UI Object Rendered

… underlying *HTML* code produced …

```
# UI Object Constructed Previously (in R)
ui

<!-- HTML Output Generated from ui Object —>
<div class="container-fluid">
  <h2>My Shiny App Title</h2>
  <div class="row">
    <div class="col-sm-4">
      <form class="well">
        <h1>SideBar Title</h1>
      </form>
    </div>
    <div class="col-sm-8">Main Content</div>
  </div>
</div>
```

# HTML in Shiny

… how to provide *style* to the app …

| Function | HTML | Rmd | Description |
|---|---|---|---|
| strong() |  | ** ** | **Bold Text** |
| em() |  | _ _ | *Italicize Text* |
| a() | <a></a> | [title](url) | Makes a hyperlink |
| p() |  | Two newlines | Text Paragraph |
| h1() | <h1></h1> | # (h1), ## (h2), ### (h3) | **Header Text** |
| br() | <br /> | Two newlines | Creates a page break |
| code() | <code></code> | ` code here ` | Code formated block |
| html() | - | - | Embed own HTML Code |

More Shiny Tags (about 110 of them!)

Customization of UI with HTML

# Control Widgets

## … allowing the user to input values …

```
checkboxInput(inputId = "checkbox",
              label = "Choice A",
              value = TRUE)
```

```
checkboxGroupInput(inputId = "checkGroup",
                   label =  h3("Checkbox group"),
                   choices = list("Choice 1" = 1,
                                  "Choice 2" = 2,
                                  "Choice 3" = 3),
                   selected = 1)
```

```
actionButton(inputId = "action",
             label = "Action"),
br(), br(),
submitButton(text = "Submit")
```

```
dateInput(inputId = "date",
          label = h3("Date input"),
          value = "2014-01-01")
```

```
dateRangeInput(inputId = "dates",
               label = h3("Date range"))
```

```
numericInput(inputId = "num",
             label = h3("Numeric input"),
             value = 1)
```

```
fileInput(inputId = "file",
          label = h3("File input"))
```

```
helpText("Note: help text isn't a true widget,",
         "but it provides an easy way to add text to",
         "accompany other widgets.")
```

```
radioButtons(inputId = "radio",
             label = h3("Radio buttons"),
             choices = list("Choice 1" = 1,
                            "Choice 2" = 2,
                            "Choice 3" = 3),
             selected = 1)
```

```
textInput(inputId = "text",
          label = h3("Text input"),
          value = "Enter text...")
```

~/Google Drive/Courses/STAT385/fa2017/static/assets/lectures/20-Shiny/hello_shiny - Shiny

http://127.0.0.1:6008 | Open in Browser | Publish

**Basic widgets**

**Buttons**: Action, Submit

**Single checkbox**: Choice A

**Checkbox group**: Choice 1, Choice 2, Choice 3

**Date input**: 2014-01-01

**Date range**: 2017-11-08 to 2017-11-08

**File input**: Browse... No file selected

**Help text**: Note: help text isn't a true widget, but it provides an easy way to add text to accompany other widgets.

**Numeric input**: 1

**Radio buttons**: Choice 1, Choice 2, Choice 3

**Select box**: Choice 1

**Sliders**: 0 50 100; 0 10 20 30 40 50 60 70 80 90 100; 0 25 75 100; 0 10 20 30 40 50 60 70 80 90 100

**Text input**: Enter text...

```
selectInput(inputId = "select",
            label = h3("Select box"),
            choices = list("Choice 1" = 1,
                           "Choice 2" = 2,
                           "Choice 3" = 3),
            selected = 1)
```

```
sliderInput(inputId = "slider1", label = h3("Sliders"),
            min = 0, max = 100, value = 50),
sliderInput(inputId = "slider2", label = "",
            min = 0, max = 100, value = c(25, 75))
```

Based off Shiny Widgets Gallery and Shiny Written Tutorial 3

# Control Widget Requirements

## … requirement per input field …

### Variable Name for Input Field

The name of the element in the *input* list in the **server** component that stores the user input.

### Descriptor for Users

Provides an explanation of the input field in the **UI**.

### Default Values

Initial input field value when app is opened.

```
numericInput(inputId = "obs", label = "Number of Obs:", value = 10)
```

All widgets generally require:

1. **inputId**
2. **label**
3. some input-specific *default values*

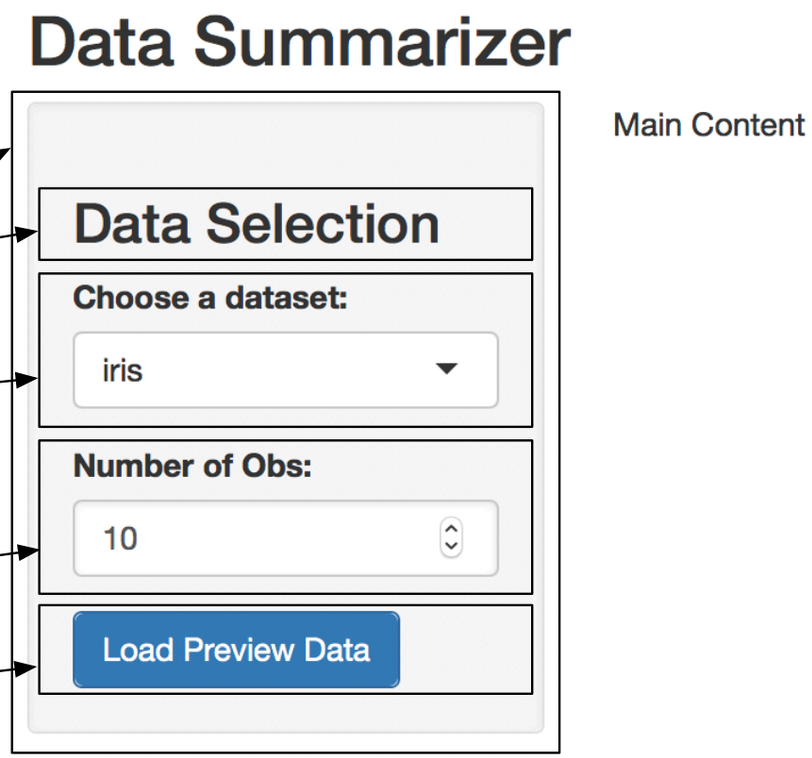
# Adding Widgets to Code

… formulating ways the user can interact with the app …

app.R

Run App

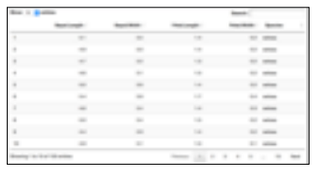
```
ui = shinyUI(
  fluidPage(
    titlePanel("Data Summarizer"),        # Title
    sidebarLayout(
      sidebarPanel(
        h3("Data Selection"),             # Note the ,

        # Dropdown
        selectInput("ds",                 # Name
                    "Choose a dataset:",  # Label
                    choices = c("iris", "Spam", "mtcars")),

        numericInput("obs",               # Name
                     "Number of Obs:",    # Label
                     10),                 # Default Value

        submitButton("Load Preview Data") # Update data
      ), # close: sidebarPanel
      mainPanel("Main Content")           # Content
    ) # close: sidebarLayout()
)) # close: shinyUI(fluidPage(
```



# Output Areas

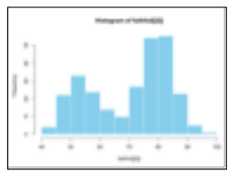
## … defining output …

## UI



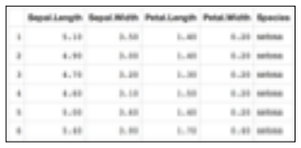
**dataTableOutput**(outputId, icon, …)

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



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


**verbatimTextOutput**(outputId)



**tableOutput**(outputId)


**textOutput**(outputId, container, inline)


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

# Output Area Requirements

## … requirement per output field …

### Variable Name for Output Area

The name of the element in the *output* list in the **server** component that displays the values.

```
tableOutput(outputId = "view-data")
```

All output elements generally require:

1. **outputId**
2. some output-specific *values*

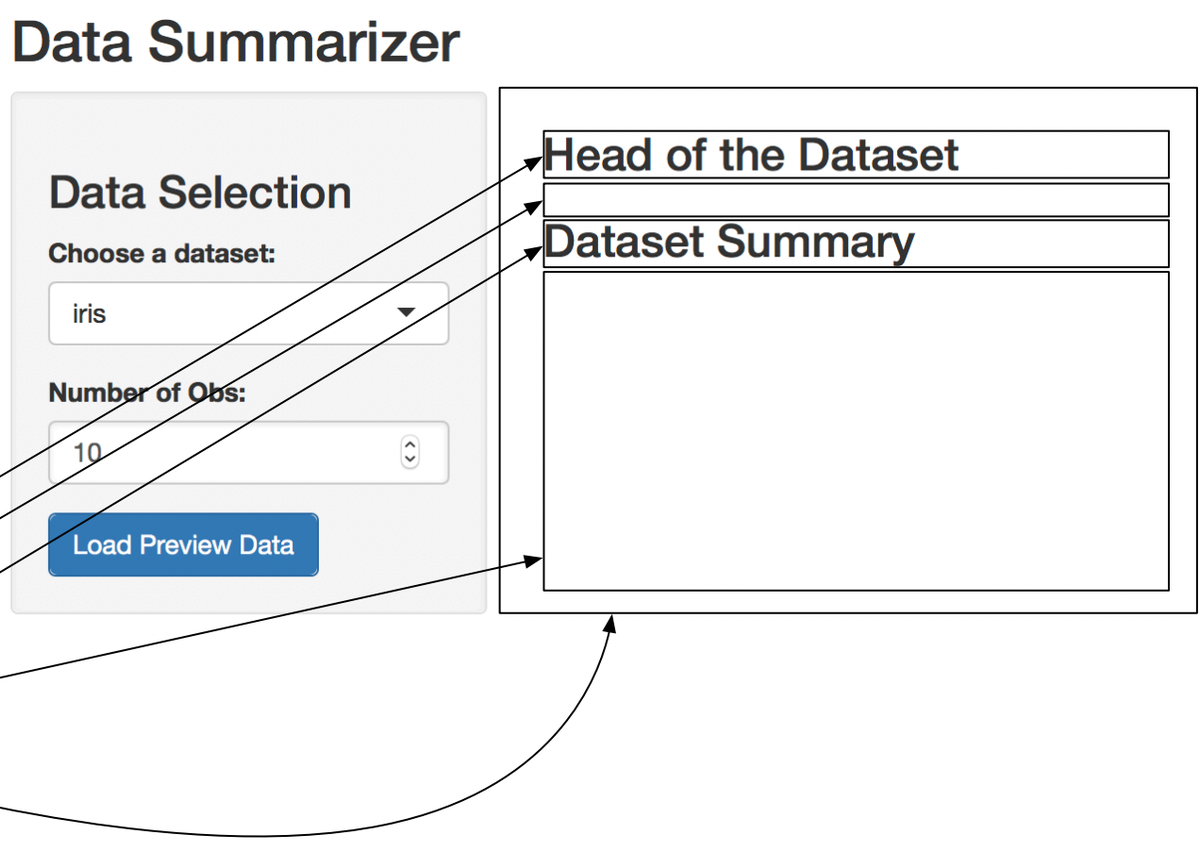
# Output UI Areas Defined

## … setting up UI output areas …

app.R

Reload App

```
1  ui = shinyUI(
2    fluidPage(
3      titlePanel("Data Summarizer"),        # Title
4      sidebarLayout(
5        sidebarPanel(
6          h3("Data Selection"),             # Note the ,
7
8          # Dropdown
9          selectInput("ds",                 # Name
10                     "Choose a dataset:",  # Label
11                     choices = c("iris", "Spam", "mtcars")),
12
13         numericInput("obs",               # Name
14                      "Number of Obs:",    # Label
15                      10),                 # Default Value
16
17         submitButton("Load Preview Data") # Update data
18       ), # close: sidebarPanel
19       mainPanel(
20         h3("Head of the Dataset"),     # HTML
21         tableOutput("view"),           # Table View
22
23         h3("Dataset Summary"),         # HTML
24         verbatimTextOutput("summary") # Output Asis
25       ) # close: mainPanel()
26     ) # close: sidebarLayout()
27 )) # close: shinyUI(fluidPage(
28
```

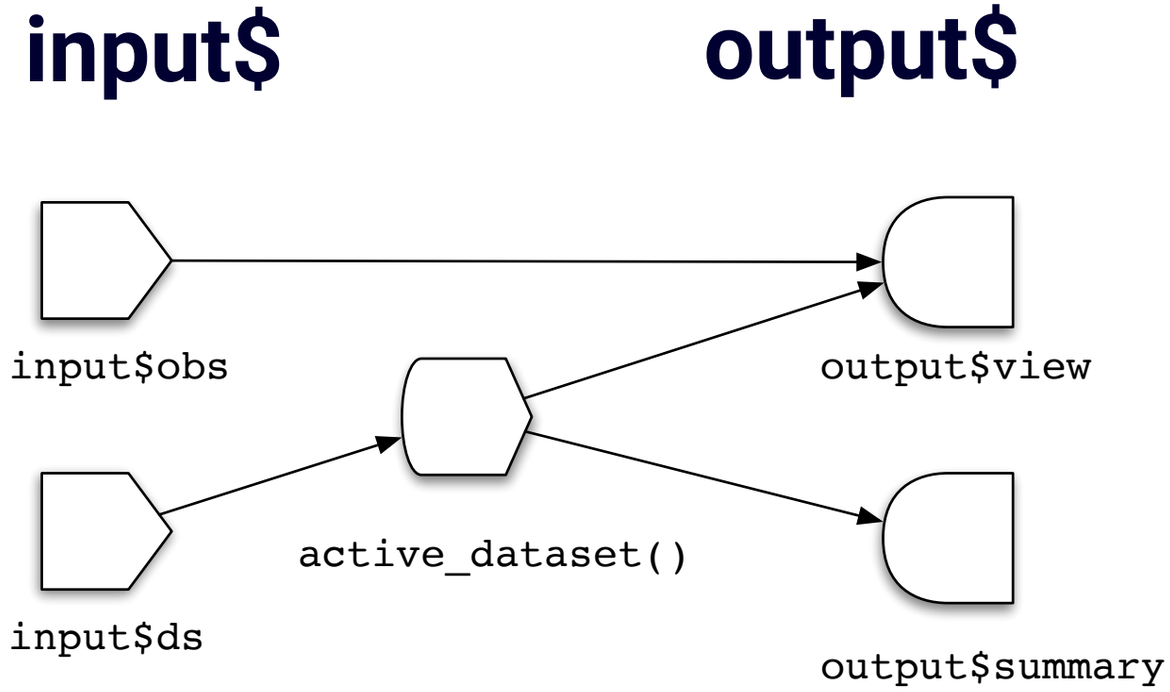


# Server

# Server Strategy

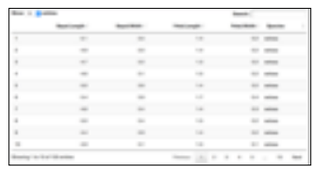
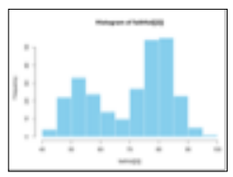
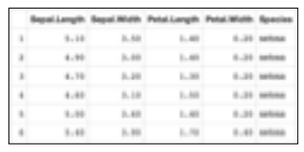
… implementing logic …

1. Retrieve the *data set name* and *number of observations* from **input$ds** and **input$obs**
2. Create output recipes **output$summary** and **output$view** for the similarly named render areas in the UI.



# Output Recipes

## … designating the output recipe for the server …

| **Server** | works with | UI |
|---|---|---|
| DT::**renderDataTable**(expr, options, callback, escape, env, quoted) | |  **dataTableOutput**(outputId, icon, …) |
| **renderImage**(expr, env, quoted, deleteFile) | |  **imageOutput**(outputId, width, height, click, dblclick, hover, hoverDelay, inline, hoverDelayType, brush, clickId, hoverId) |
| **renderPlot**(expr, width, height, res, …, env, quoted, func) | |  **plotOutput**(outputId, width, height, click, dblclick hover, hoverDelay, inline, hoverDelayType, brush clickId, hoverId) |
| **renderPrint**(expr, env, quoted, func, width) | |  **verbatimTextOutput**(outputId) |
| **renderTable**(expr,…, env, quoted, func) | |  **tableOutput**(outputId) |
| **renderText**(expr, env, quoted, func) | | foo **textOutput**(outputId, container, inline) |
| **renderUI**(expr, env, quoted, func) | |  **uiOutput**(outputId, inline, container, …) **htmlOutput**(outputId, inline, container, …) |

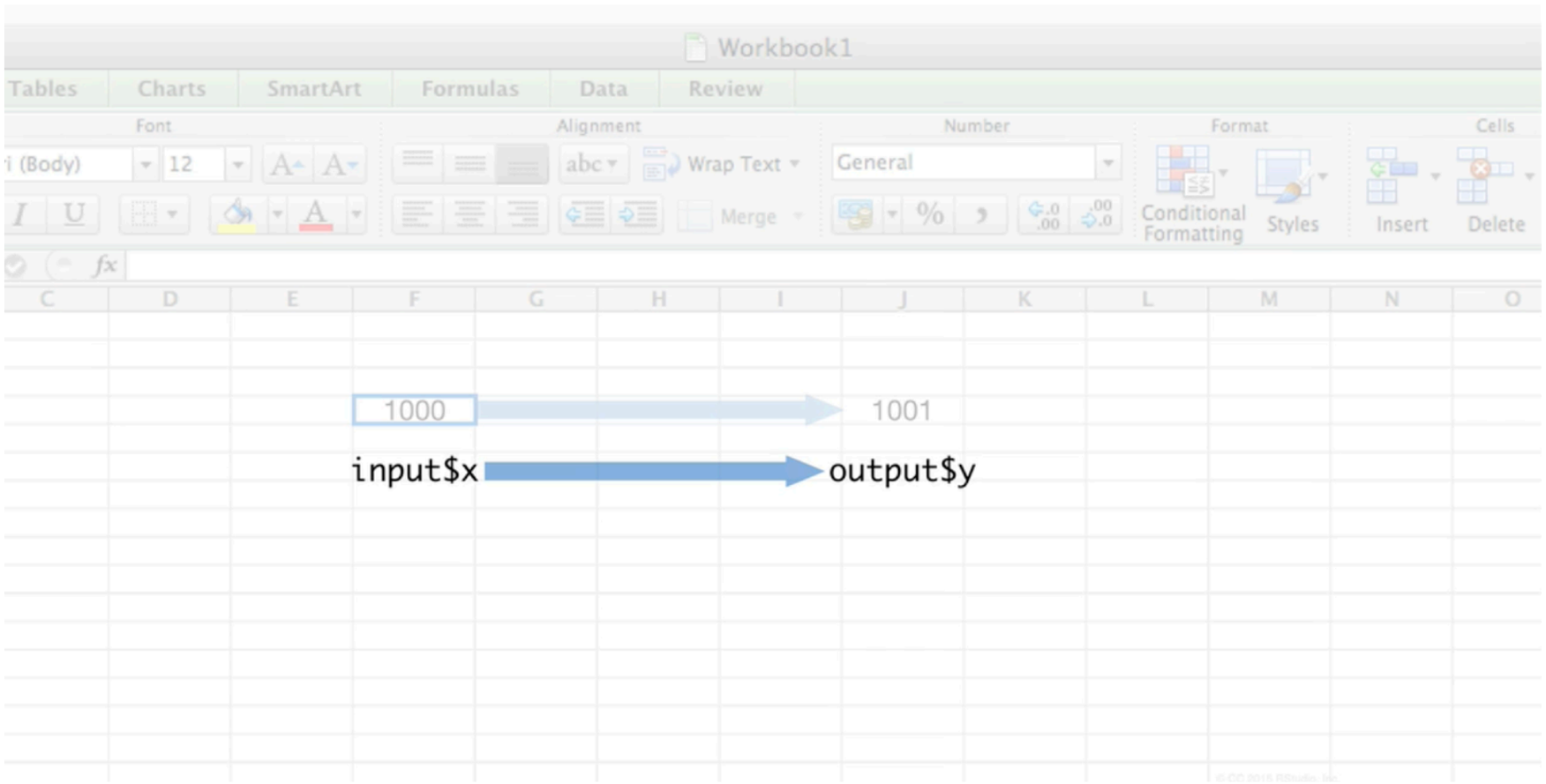
# Connection

## … relation between UI and Server components …

```
library("shiny")
ui = fluidPage(
       # Dropdown Menu with fixed Choices
       selectInput(inputId = "ds",                    # Server ID
                   label = "Choose a dataset:",   # UI Label
                   choices = c("iris", "Spam", "mtcars")),
       verbatimTextOutput("summary")              # Output Asis
) # close: fluidPage(


server = function(input, output) {

  output$summary = renderPrint({   # Summary Render

    summary(get(input$ds))

  }) # close: renderPrint()

}
```

# Reactivity in Excel

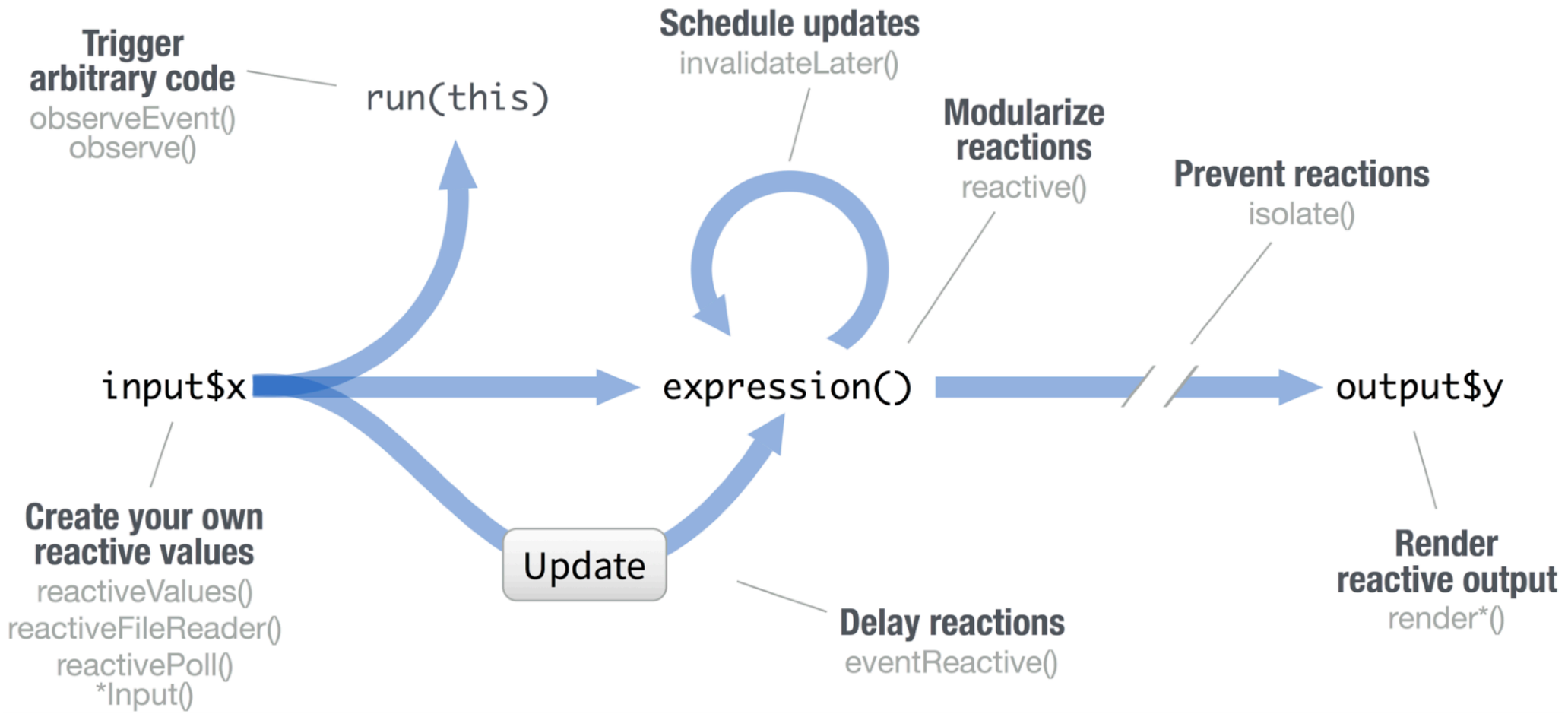
## … formulaic updates …




[Reactivity Explanation]

# Types of Reactivity

## … how reactivity comes into play …



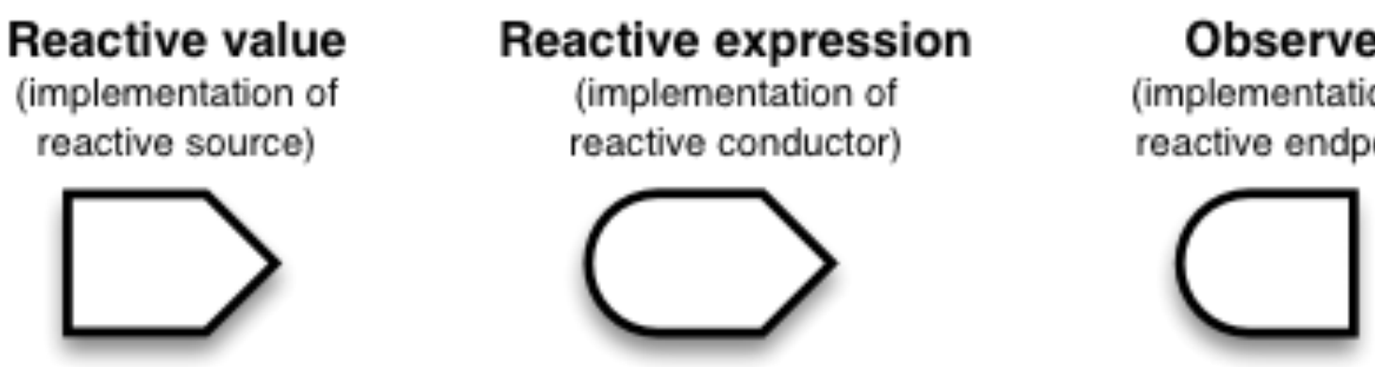

Source

# Reactivity

… how input and output variables work …

**“For every action, there is an equal and opposite reaction.”**

–Issac Newton

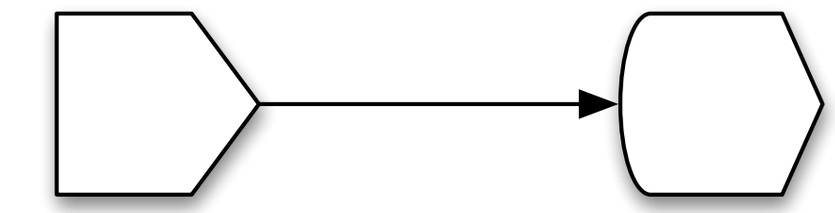



- Reactive Sources (**Reactive Values**)
  - UI Widget Inputs contained in **input$** or **reactiveValues()**
- Reactive Conductors (**Reactive Expressions**)
  - Server catches for UI elements **reactive({})**
- Reactive Endpoints (**Observers**)
  - Recipes for UI Output contained in **output$** and **observer({})** in Server

```
# Server component
server = function(input, output) {

  # Reactive conductor
  active_dataset =
   eventReactive(input$preview,
    {
     switch(input$ds,
                "iris" = iris,
                "Spam" = Spam,
                "mtcars" = mtcars
                )
    }
  ) # close: eventReactive()

  # Output recipes omitted
  # e.g. summary() and view()
}
```

# Event Reactive

… caching values …




Convert character value
to **data.frame** object

```
server = function(input, output) {
  # Reactive conductor
  active_dataset =
    eventReactive(input$preview, {
        switch(input$ds,
                    "iris" = iris,
                    "Spam" = Spam,
                     "mtcars" = mtcars
                   )
  }) # close: eventReactive()

  # Summary Render
  output$summary = renderPrint({
    summary(active_dataset())
  }) # close: renderPrint()

  # Table Render
  output$view = renderTable({
    head(active_dataset(),
              n = input$obs)
  }) # close: renderPrint()
}
```

# Server Implementation

… reactive conductor and observers …

The reactive conductor is a function call!

# Variable Mapping

## … how reactivity maps between UI and Server…

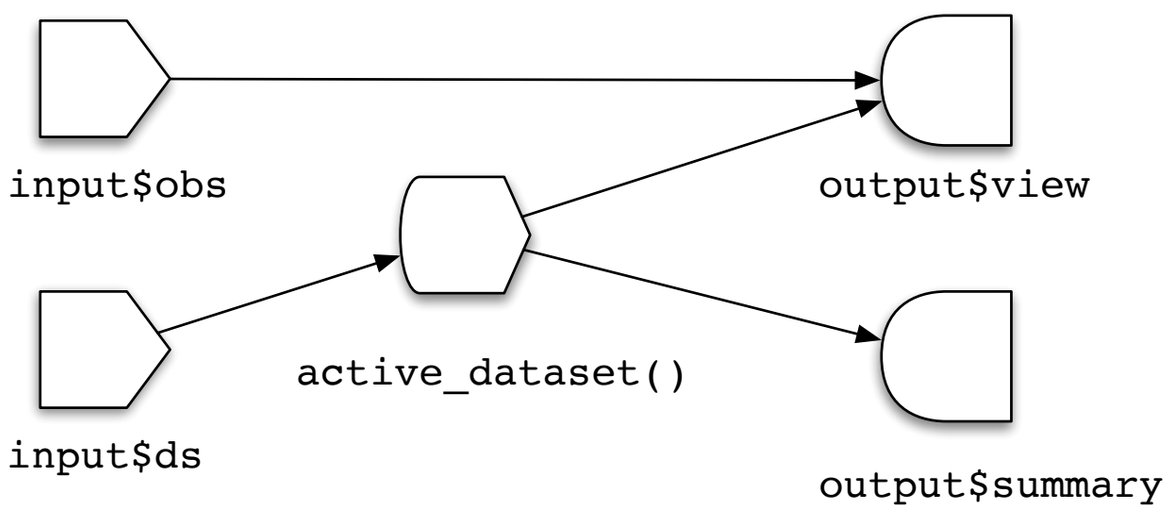
app.R

Run App

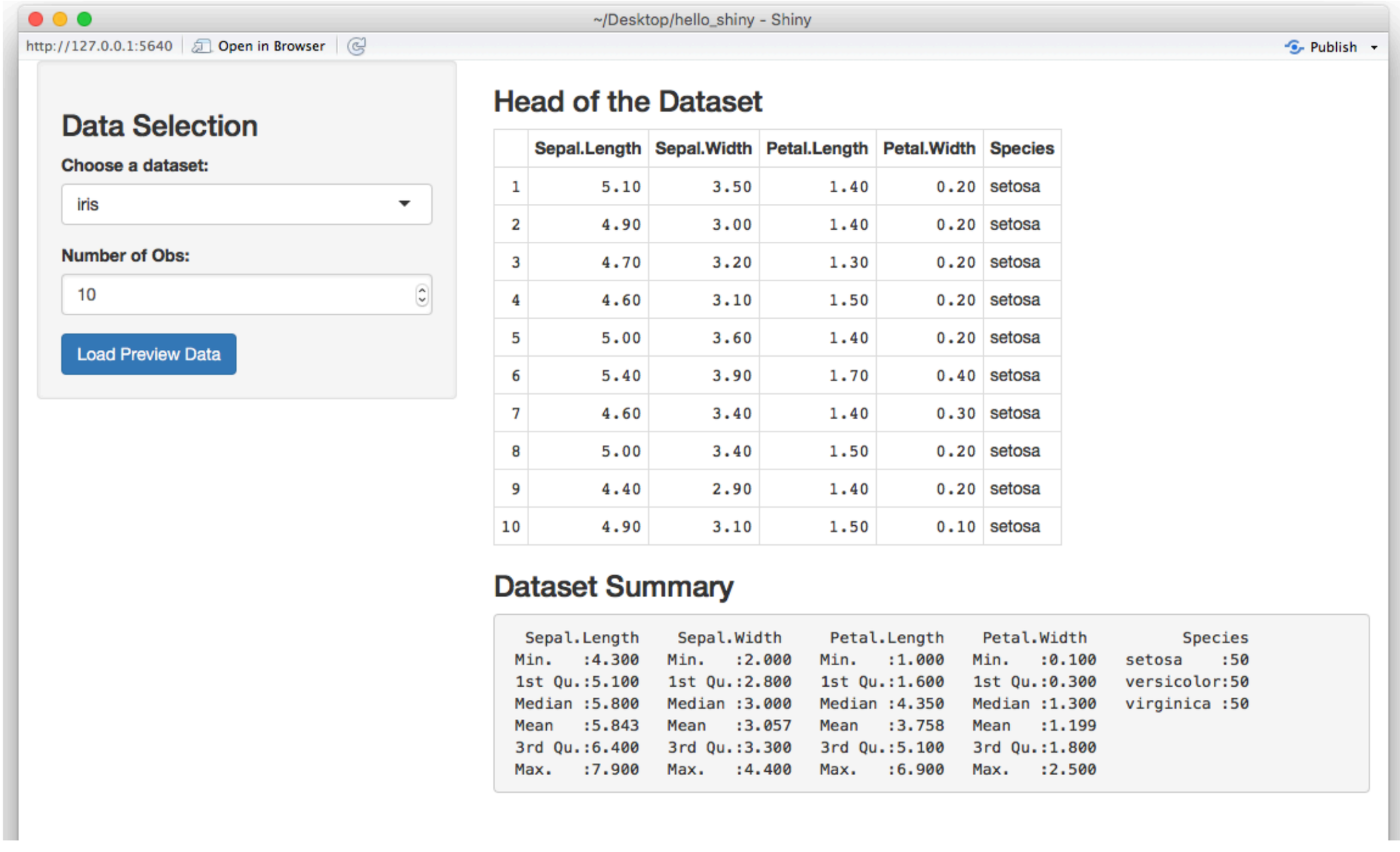
```r
ui = shinyUI(
  fluidPage(
    titlePanel("Data Summarizer"),        # Title
    sidebarLayout(
      sidebarPanel(
        h3("Data Selection"),             # Note the ,

        # Dropdown
        selectInput("ds",                 # Name
                    "Choose a dataset:",  # Label
                    choices = c("iris", "Spam", "mtcars")),

        numericInput("obs",               # Name
                     "Number of Obs:",    # Label
                     10),                 # Default Value

        submitButton("Load Preview Data") # Update data
      ), # close: sidebarPanel()
      mainPanel(
        h3("Head of the Dataset"),    # HTML
        tableOutput("view"),          # Table View

        h3("Dataset Summary"),        # HTML
        verbatimTextOutput("summary") # Output Asis
      ) # close: mainPanel()
    ) # close: sidebarLayout()
)) # close: shinyUI(fluidPage(
```

```r
server = shinyServer(function(input, output) {

  active_dataset = reactive({      # Reactive Conductor
    if(input$ds == "iris") {
      iris
    } else if (input$ds == "Spam") {
      Spam
    } else {
      mtcars
    }
  }) # close: reactive()

  output$summary = renderPrint({   # Summary Render
    summary(active_dataset())
  }) # close: renderPrint()

  output$view = renderTable({      # Table Render
    head(active_dataset(),
         n = input$obs)
  }) # close: renderPrint()

}) # close: shinyServer()
```

Reactive Source
Reactive Endpoint
Reactive Conductor



# Final App




| | Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
|---|---|---|---|---|---|
| 1 | 5.10 | 3.50 | 1.40 | 0.20 | setosa |
| 2 | 4.90 | 3.00 | 1.40 | 0.20 | setosa |
| 3 | 4.70 | 3.20 | 1.30 | 0.20 | setosa |
| 4 | 4.60 | 3.10 | 1.50 | 0.20 | setosa |
| 5 | 5.00 | 3.60 | 1.40 | 0.20 | setosa |
| 6 | 5.40 | 3.90 | 1.70 | 0.40 | setosa |
| 7 | 4.60 | 3.40 | 1.40 | 0.30 | setosa |
| 8 | 5.00 | 3.40 | 1.50 | 0.20 | setosa |
| 9 | 4.40 | 2.90 | 1.40 | 0.20 | setosa |
| 10 | 4.90 | 3.10 | 1.50 | 0.10 | setosa |

```
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species
Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa    :50
1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   versicolor:50
Median :5.800   Median :3.000   Median :4.350   Median :1.300   virginica :50
Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
```

# Debugging

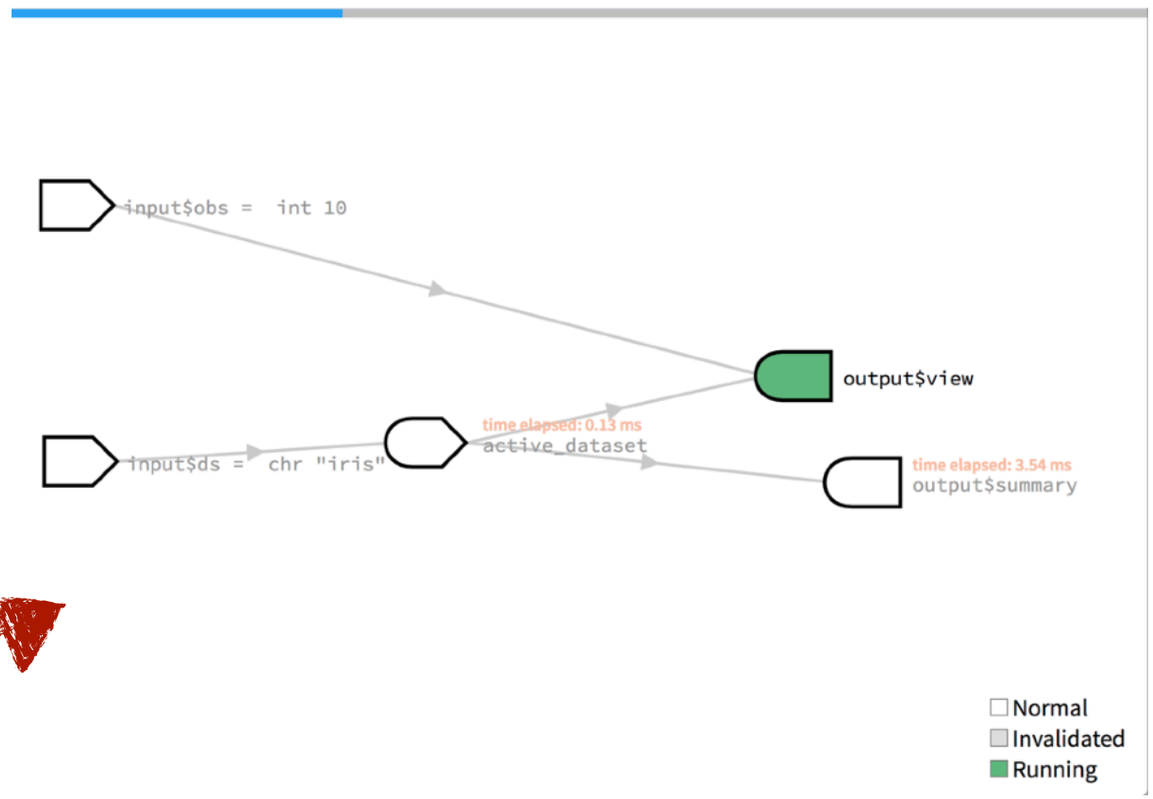
Use a *Version Control System*

**A**lways **B**e **C**ommitting

```
# Load shiny package
library("shiny")

# Enable recording of actions
options(
   shiny.reactlog = TRUE
)

# Launch the application
runApp()

# Stop the App and, then run
showReactLog()
```

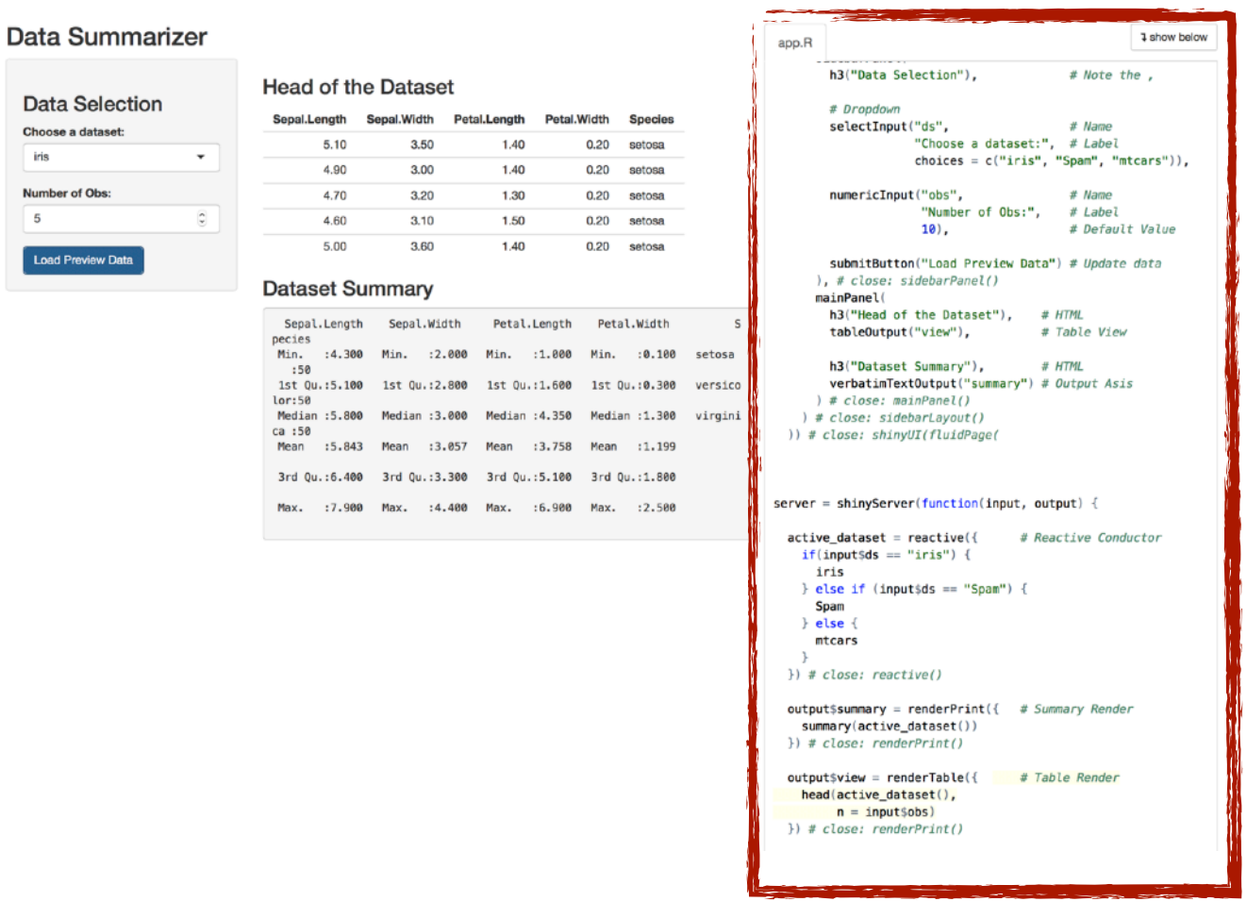
# Reactive Object Graphs

… see how reactivity acts …



# Showcase Mode

… each reaction updates the **app.R** code …




```
# Load shiny package
library("shiny")

# Launch the application
# with code being shown
# inline
runApp(
  display.mode = "showcase"
)
```

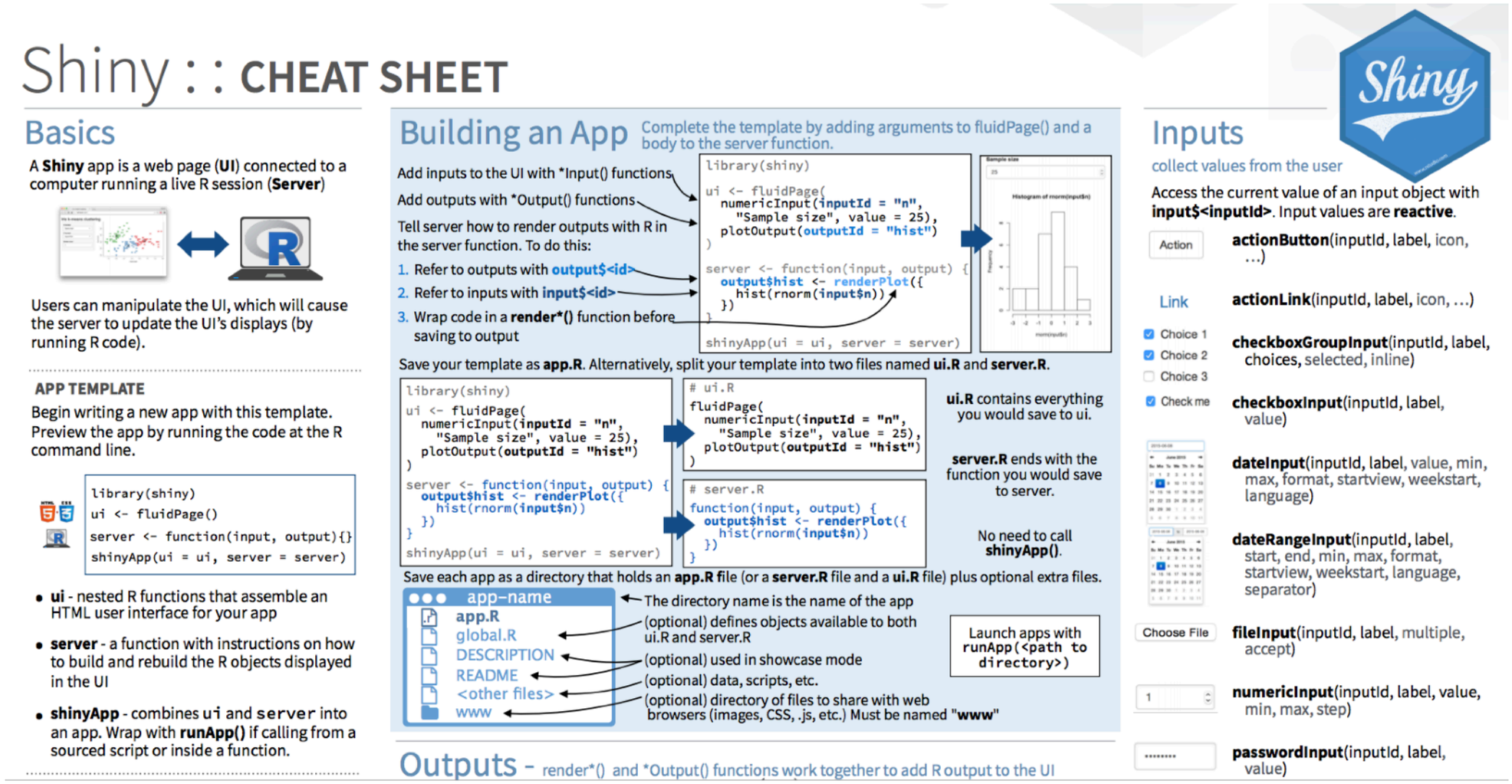
# Recap

- **Lists**
  - Provide the ability to return mixed data types
  - Act as a "dictionary"
- **Shiny**
  - Allows for R to communicate with a web browser
  - Lower the barrier of entry for sharing new algorithms with other users

# Resources

# Cheatsheet for Shiny

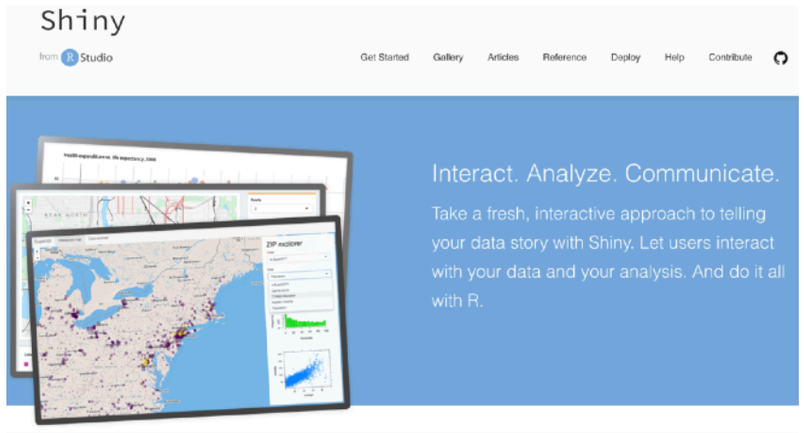
## … shiny overview …



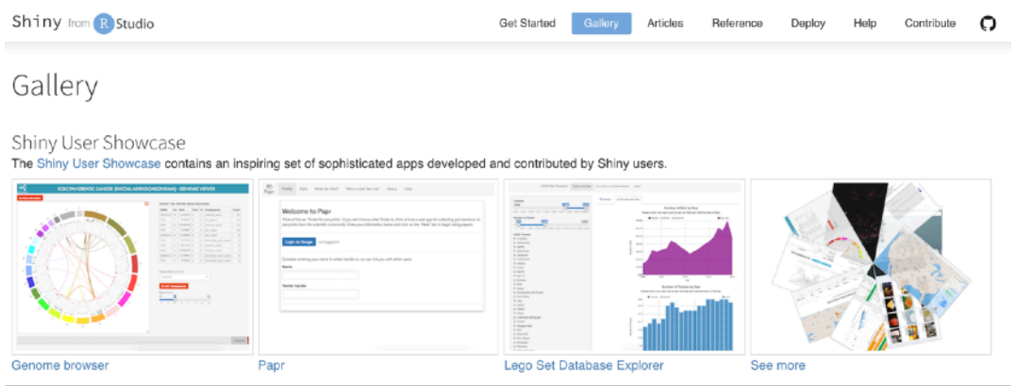

https://github.com/rstudio/cheatsheets/raw/master/shiny.pdf

# Shiny Websites

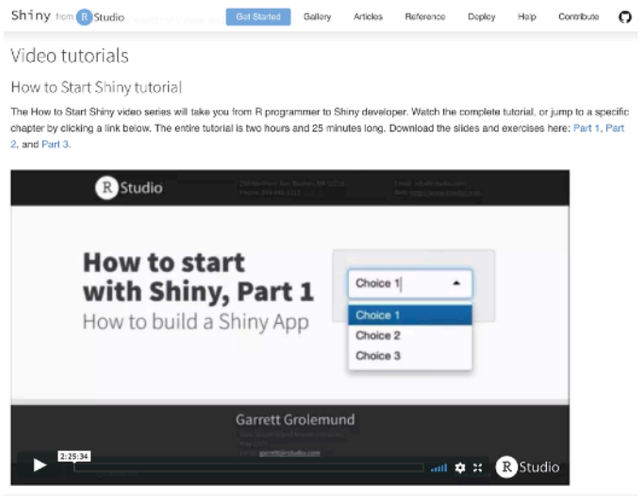
## … official shiny websites backed by RStudio …



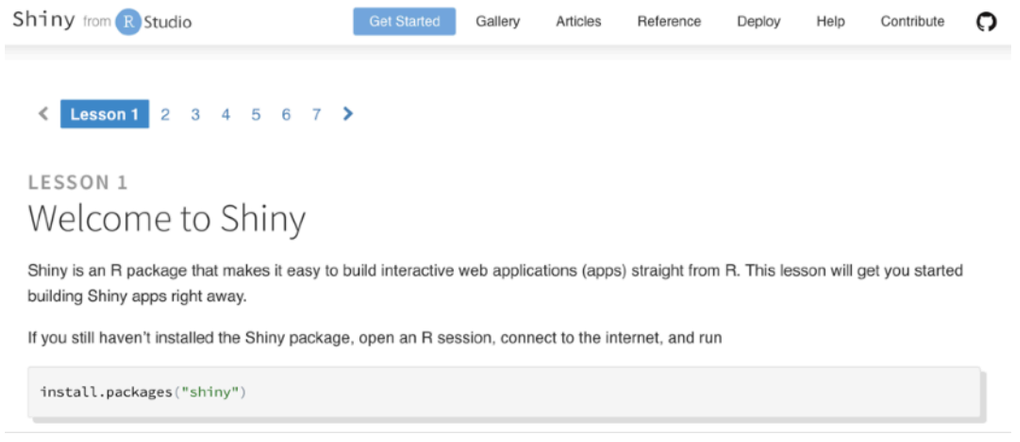

Shiny's Website




Shiny App Gallery




Video Tutorials




Written Tutorials

# Acknowledgements

# Acknowledgements

- Style of the RStudio Cheatsheet for Shiny
- More Widgets Shiny App
- Joe Cheng's "Ladder of Enlightenment" for Shiny
- Shiny: Reactivity Overview