

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

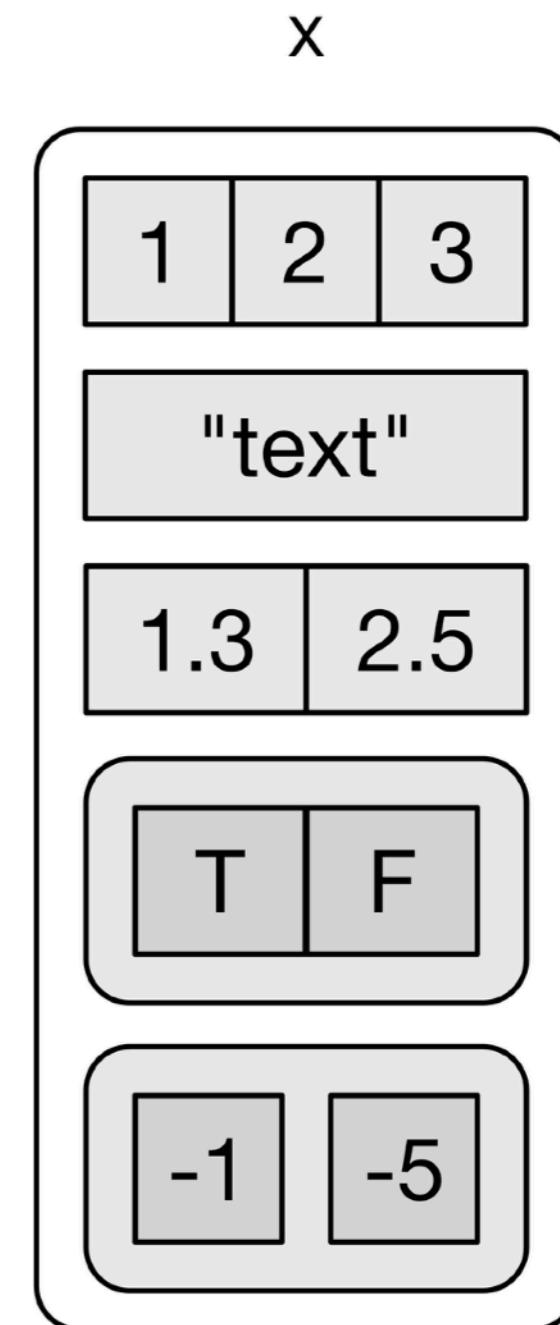
List

... 1 Dimensional 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...

**Named
Elements**

**Position
Index**

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

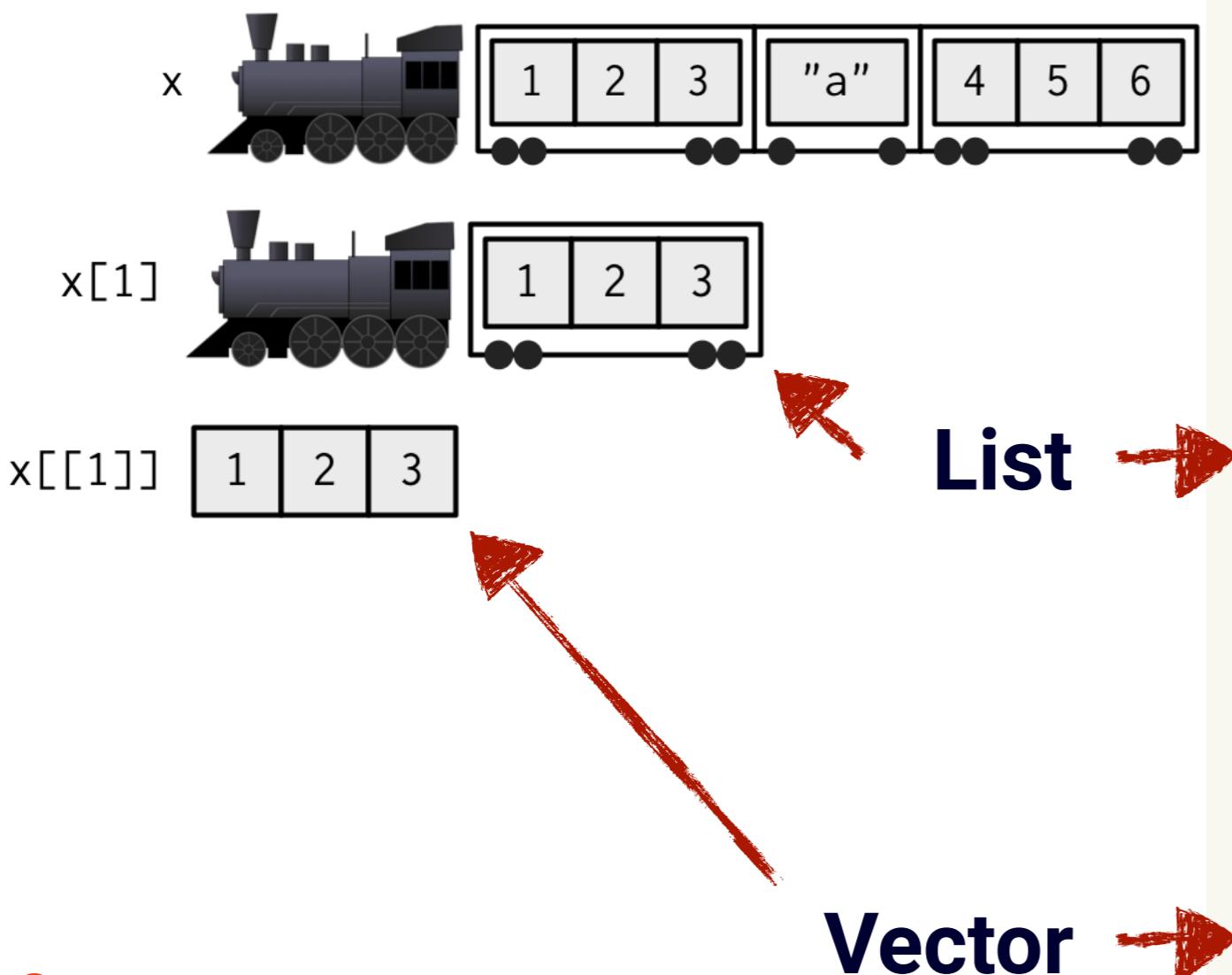
Combine different types
out =

```
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
```

Preserving vs. Simplifying

... subset differences with trains ...



```
# 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
```

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	<code>x[[1]]</code>	<code>x[1]</code>
Lists	<code>x[[1]]</code> <code>x\$name</code>	<code>x[1]</code>
Array	<code>x[1,]</code> or <code>x[, 1]</code>	<code>x[1, , drop = F]</code> or <code>x[, 1, drop = F]</code>
Data Frames	<code>x[, 1]</code> or <code>x[[1]]</code>	<code>x[, 1, drop = F]</code> or <code>x[1]</code>

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

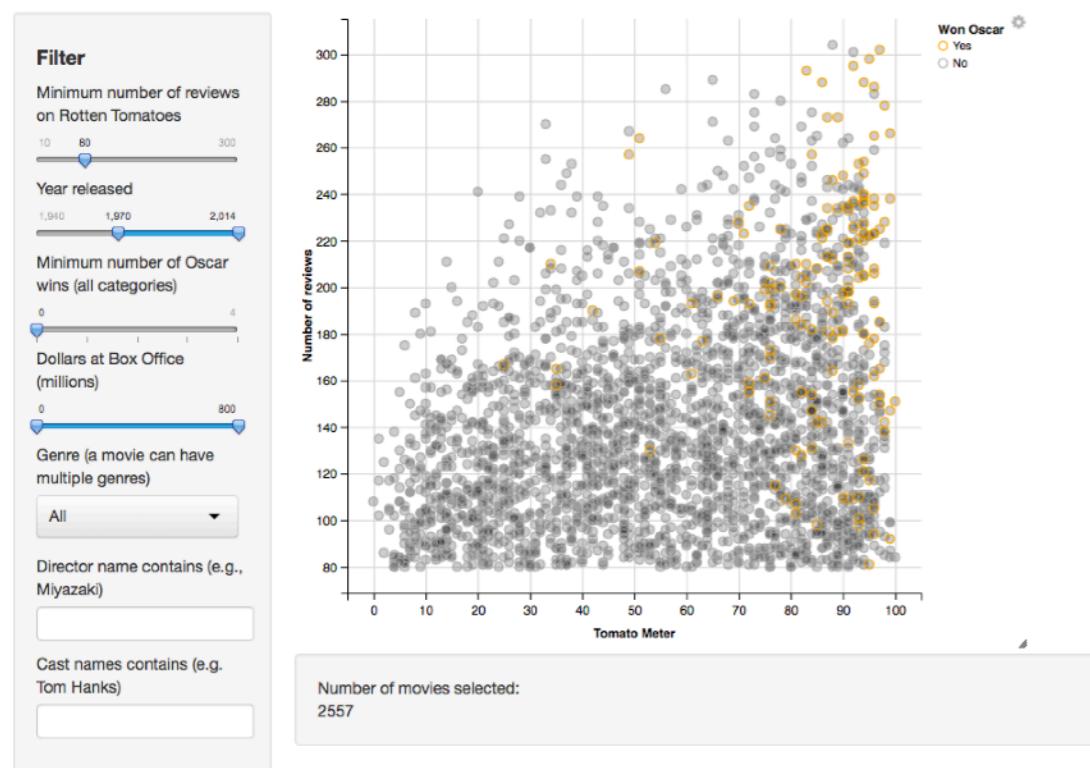


Send **User Input** from Web Browser to *R*

Shiny

... facilitating *R* to Web Browser interactions since '12 ...

Movie explorer



[Shiny Gallery: Movie explorer](#)

The figure shows the Radiant interface, a web-based RStudio add-on for business analytics. The top navigation bar includes tabs for Radiant, Data, Design, Basics, Model, Multivariate, R, and Help. The main area is titled 'Data preview' and shows the first 10 rows of the 'diamonds' dataset. The columns listed are price, carat, clarity, cut, color, depth, table, x, y, z, and date. Each row contains specific data points for these variables. Below the preview, there are sections for managing datasets, loading data from files, saving data, and removing data from memory. A note at the bottom states '10 of 3,000 rows shown. See View-tab for details.'

Diamond prices

Prices of 3,000 round cut diamonds

Description

A dataset containing the prices and other attributes of a sample of 3000 diamonds. The variables are as follows:

[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.

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

Barebones

... minimum required for shiny ...

```
# 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$  
}
```

Specify a UI template

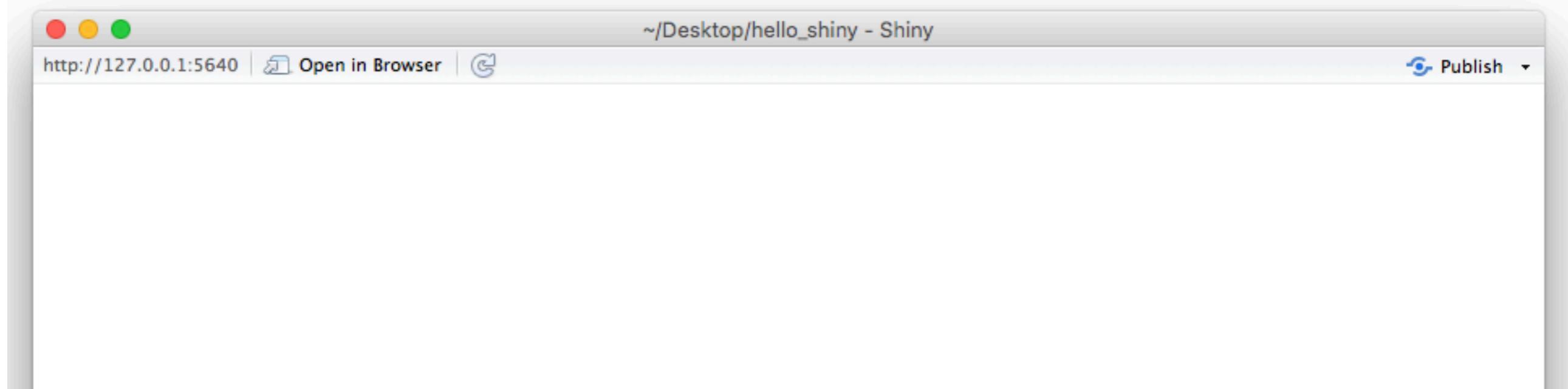
Define a Function with two pre-named arguments

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

Call function with above values to launch app

Output

... a master piece ...



```
Console Terminal × Jobs ×  
/cloud/project/ ↵  
> runApp('minimal-shiny.R')  
  
Listening on http://127.0.0.1:6076
```



Stop App

```
Console Terminal × Jobs ×  
/cloud/project/ ↵  
> runApp('minimal-shiny.R')  
  
Listening on http://127.0.0.1:6076  
  
>
```

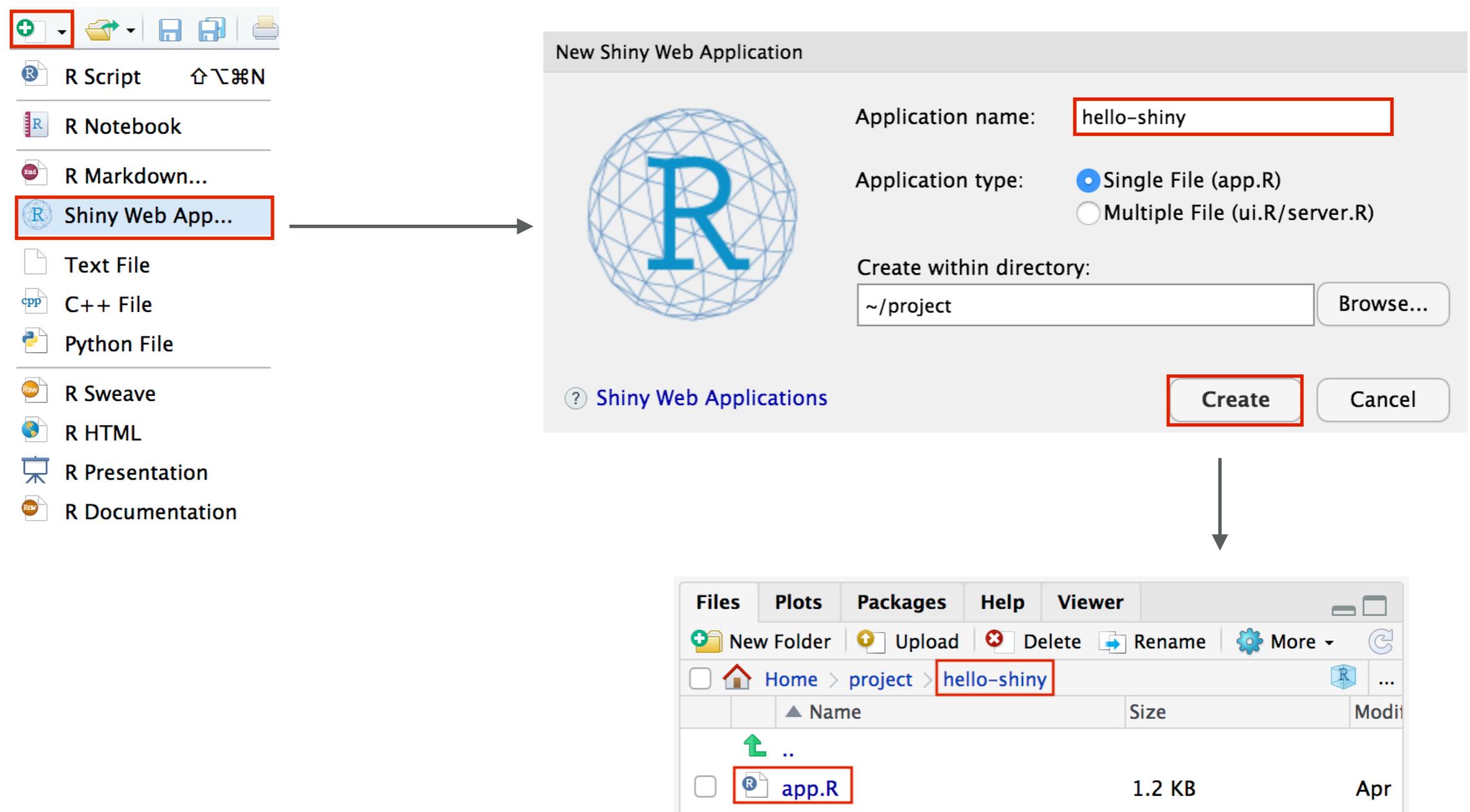


Normal R session

New Shiny App

Making a Shiny App

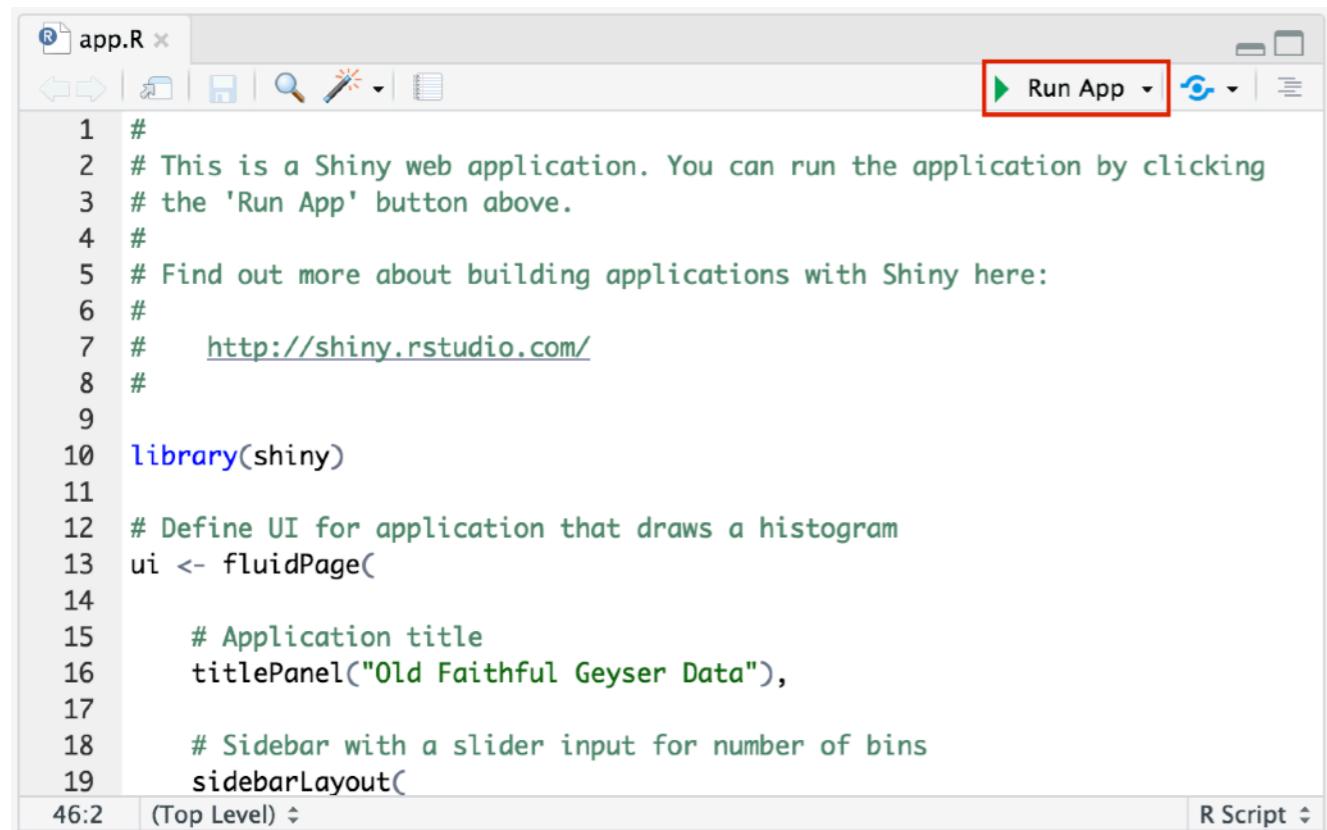
... setting up RStudio to host a shiny development environment ...



Source + Live App

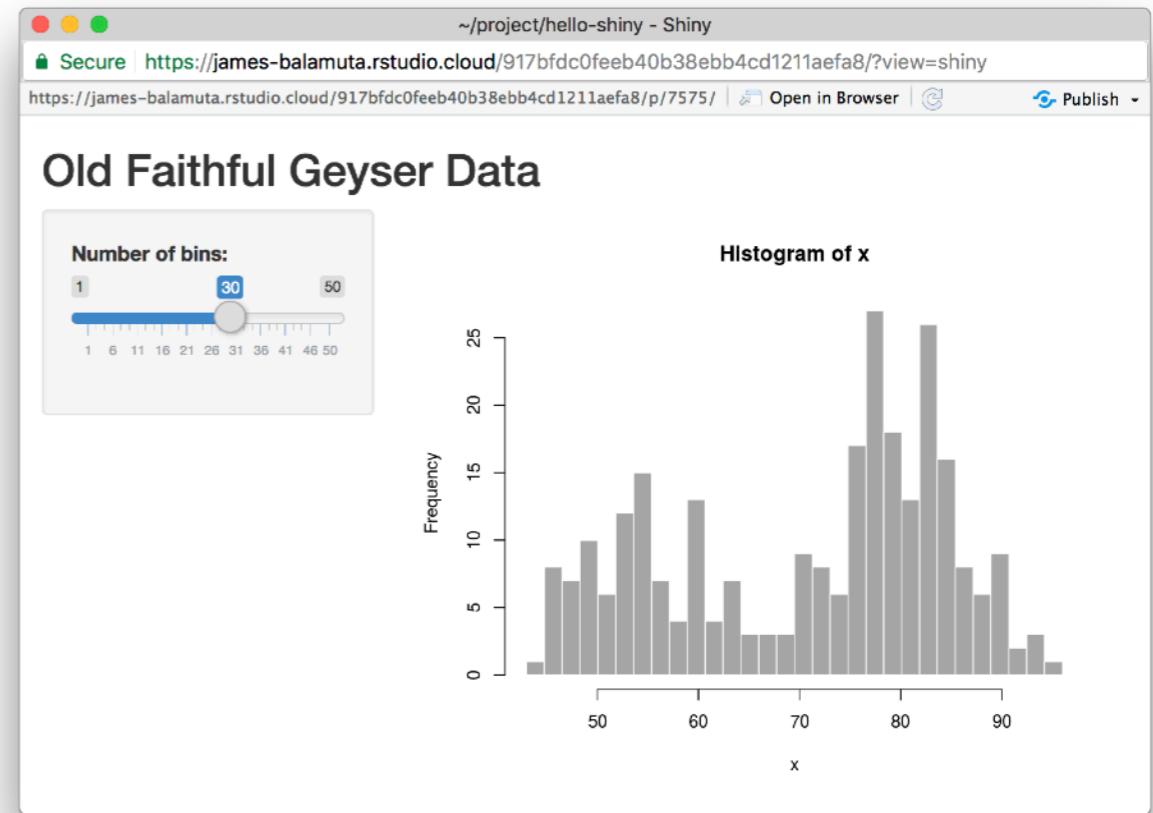
... note the "Run App" in source...

Source Panel



```
app.R x
1 # This is a Shiny web application. You can run the application by clicking
2 # the 'Run App' button above.
3 #
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(
```

Live App



```
Console Terminal x Jobs x
/cloud/project/
> shiny::runApp('demo')
Loading required package: shiny
Listening on http://127.0.0.1:6076
```

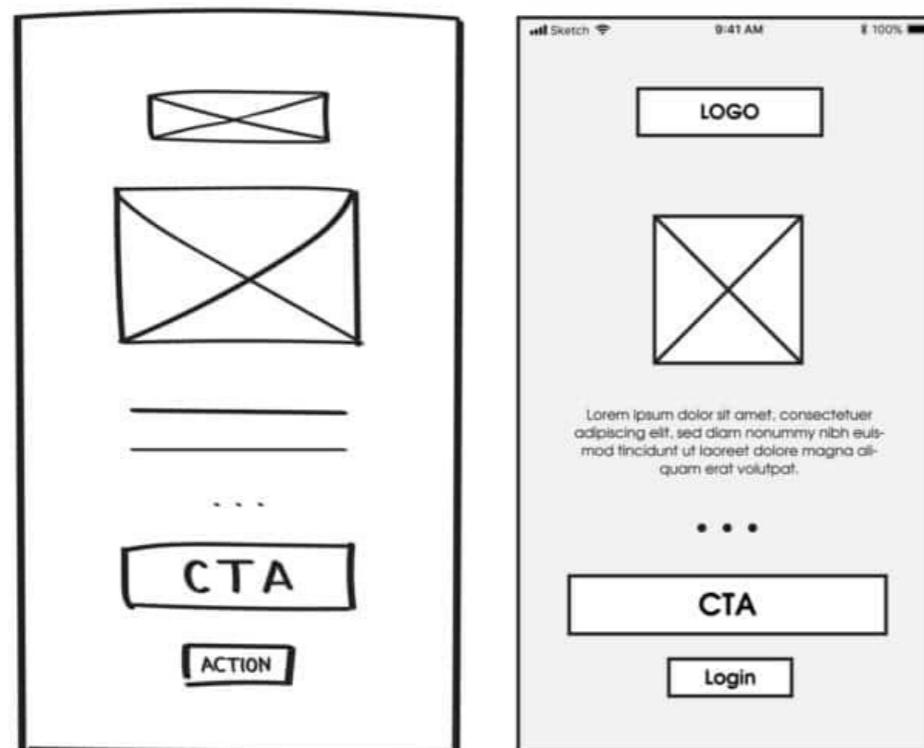
Stop App

Sketching Apps

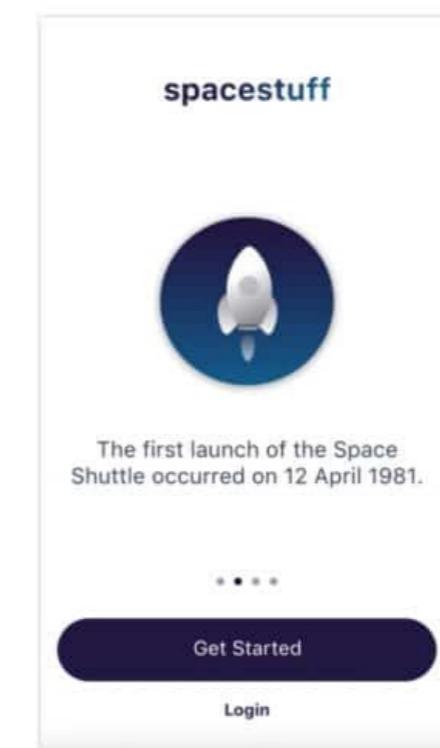
Definition:

Prototypes are filters that traverse a design space and are manifestations of ideas. ([Lim et al. ACM TOCHI, 15 \(2\), 2008.](#))

Prototypes



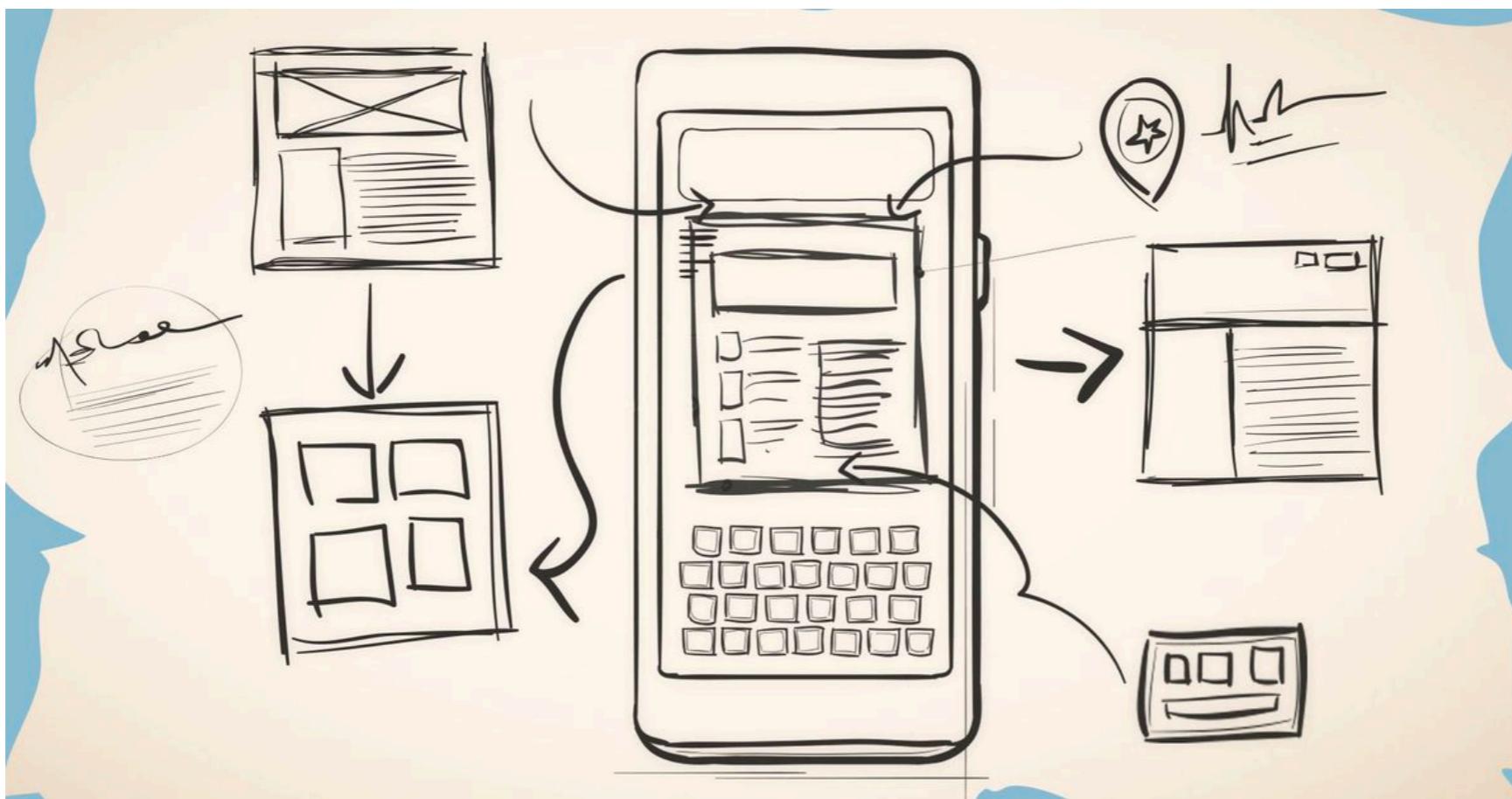
Final Product



[Source](#)

Definition:

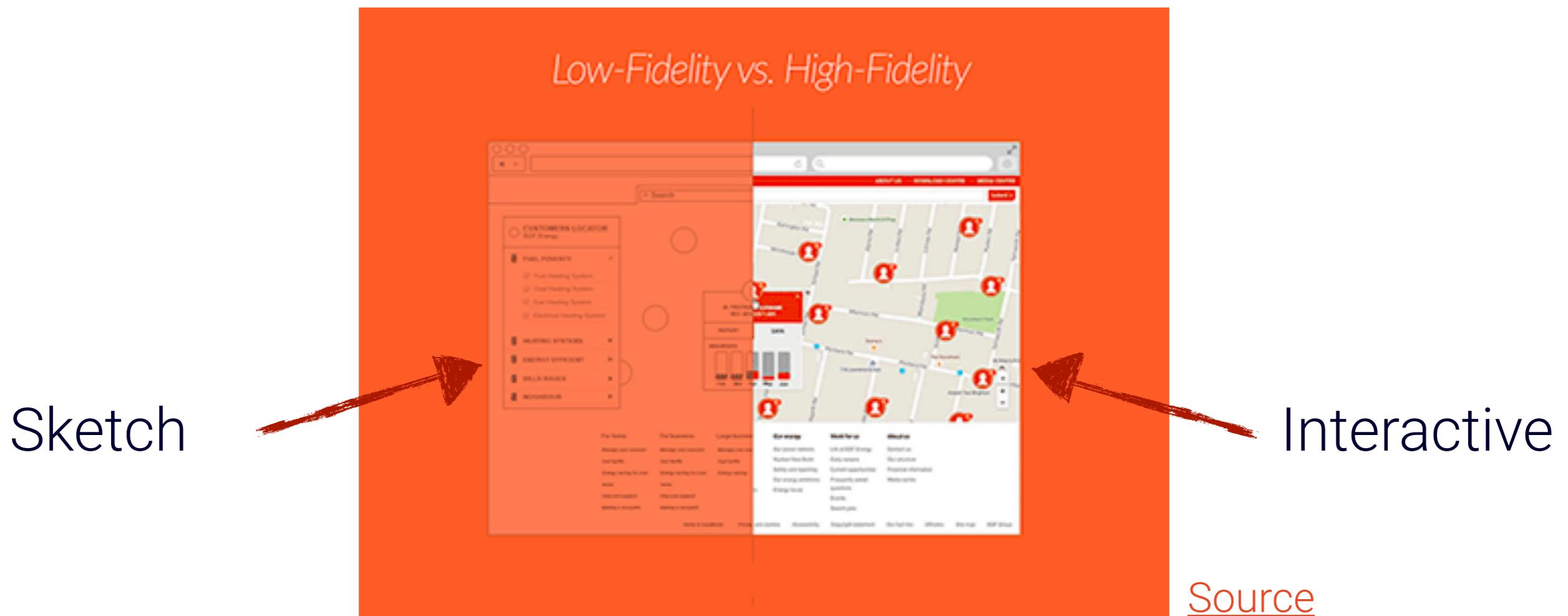
Low Fidelity Prototypes are prototypes that trade **realism** for **iterability** by using paper sketches and story boards to convey the desired purpose.



Source

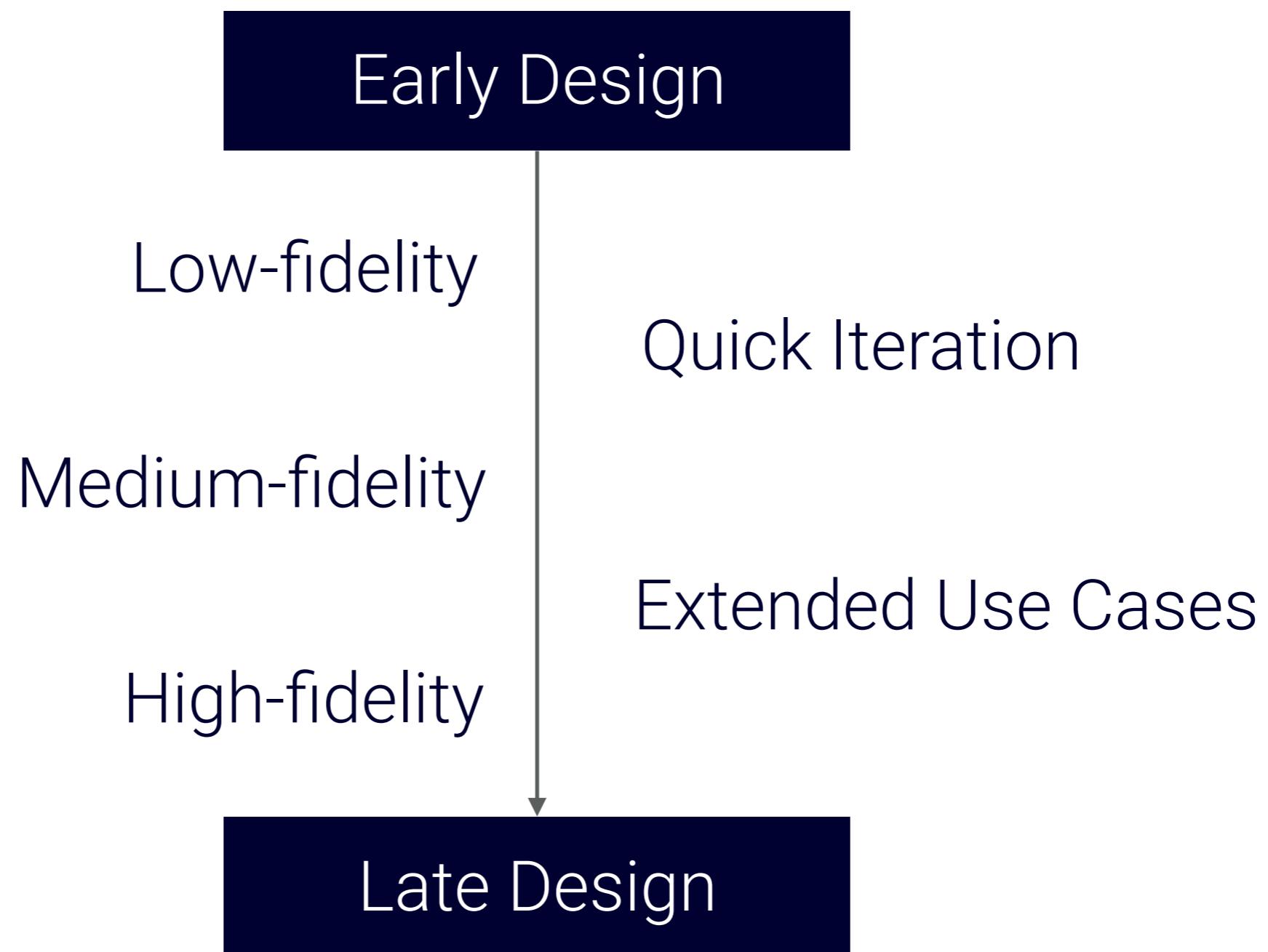
Definition:

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



Cost of a Prototype

... when to go with one prototype over another ...



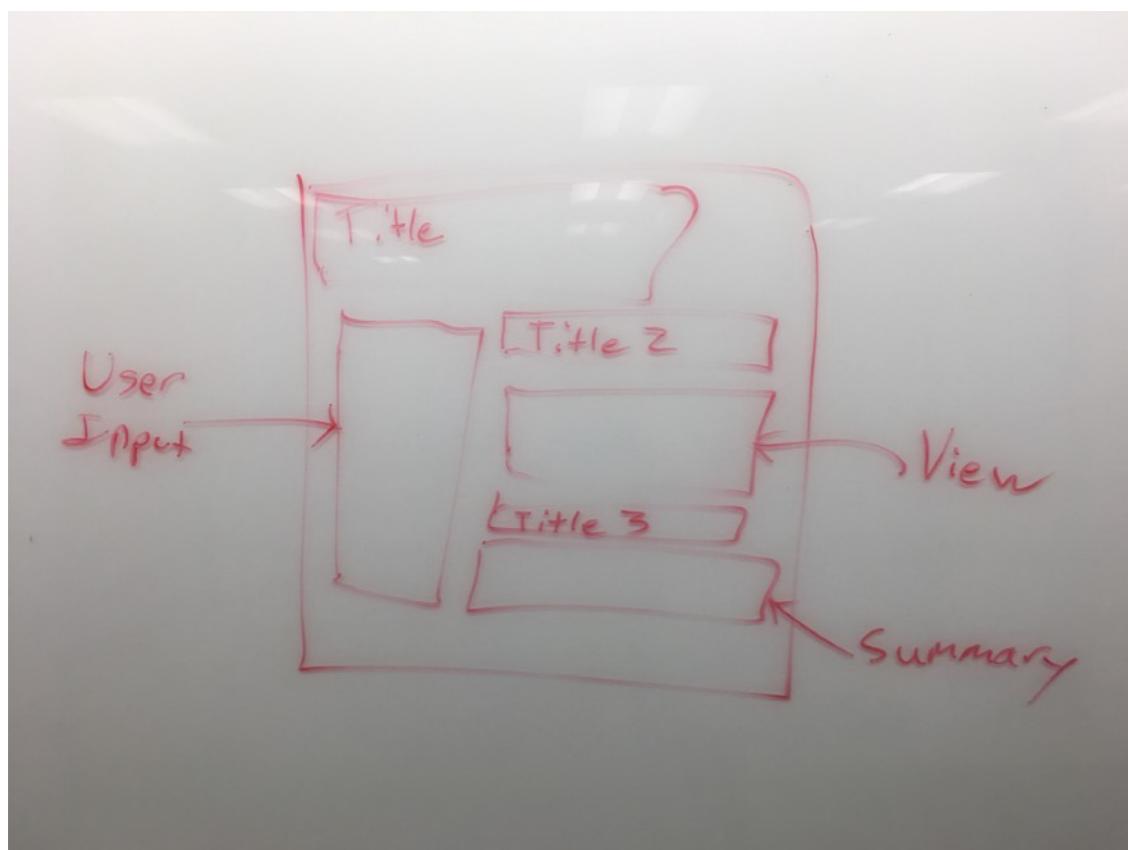
Your Turn

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

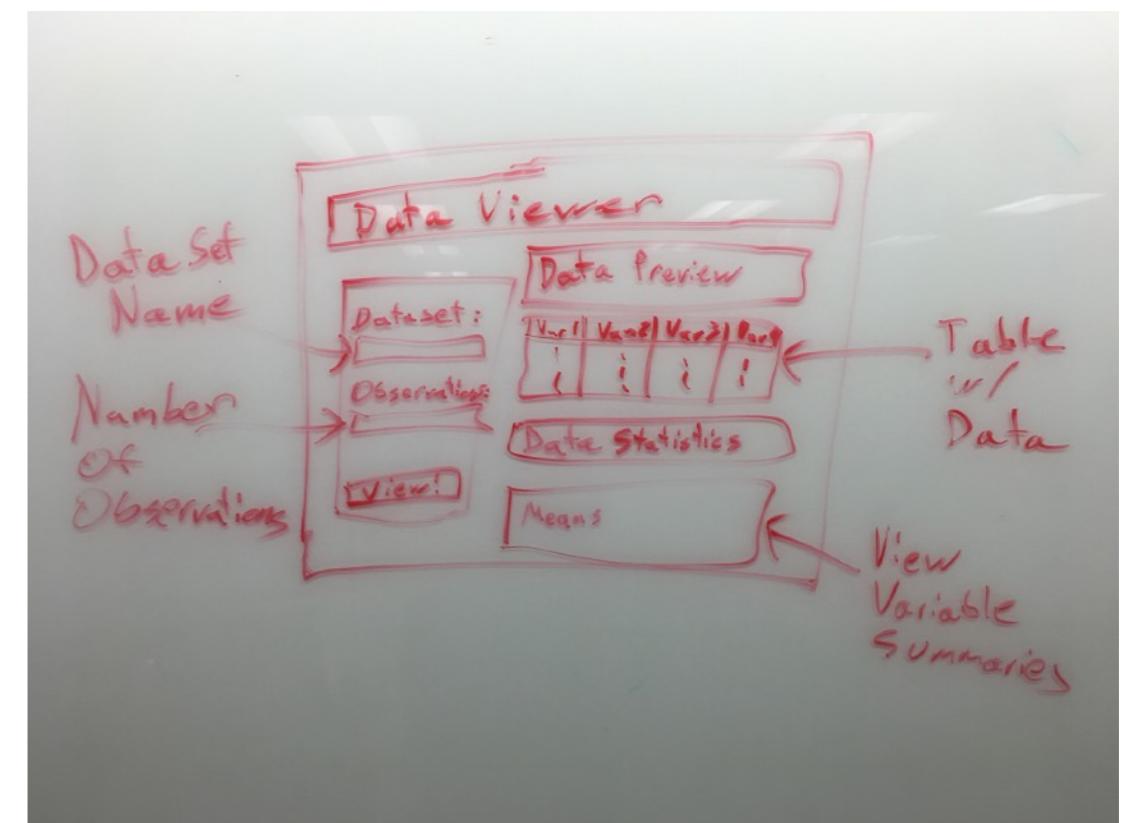
Goal: Sketch as many ideas as possible to enable the data to be displayed and interacted with.

Thinking about an Idea

... sample sketches for a Data Viewer ...



Iteration 1
(Scribble Sketch)



Iteration 2
(Paper Prototype)

Goal

... aim of this lecture ...

The screenshot shows a Shiny application window titled "~/Desktop/hello_shiny - Shiny" at the URL "http://127.0.0.1:5640".

Data Selection: A sidebar on the left contains a dropdown menu labeled "Choose a dataset" with "iris" selected, and a dropdown menu labeled "Number of Obs" with "10" selected. Below these is a blue button labeled "Load Preview Data".

Head of the Dataset: A table titled "Head of the Dataset" showing the first 10 rows of the iris dataset.

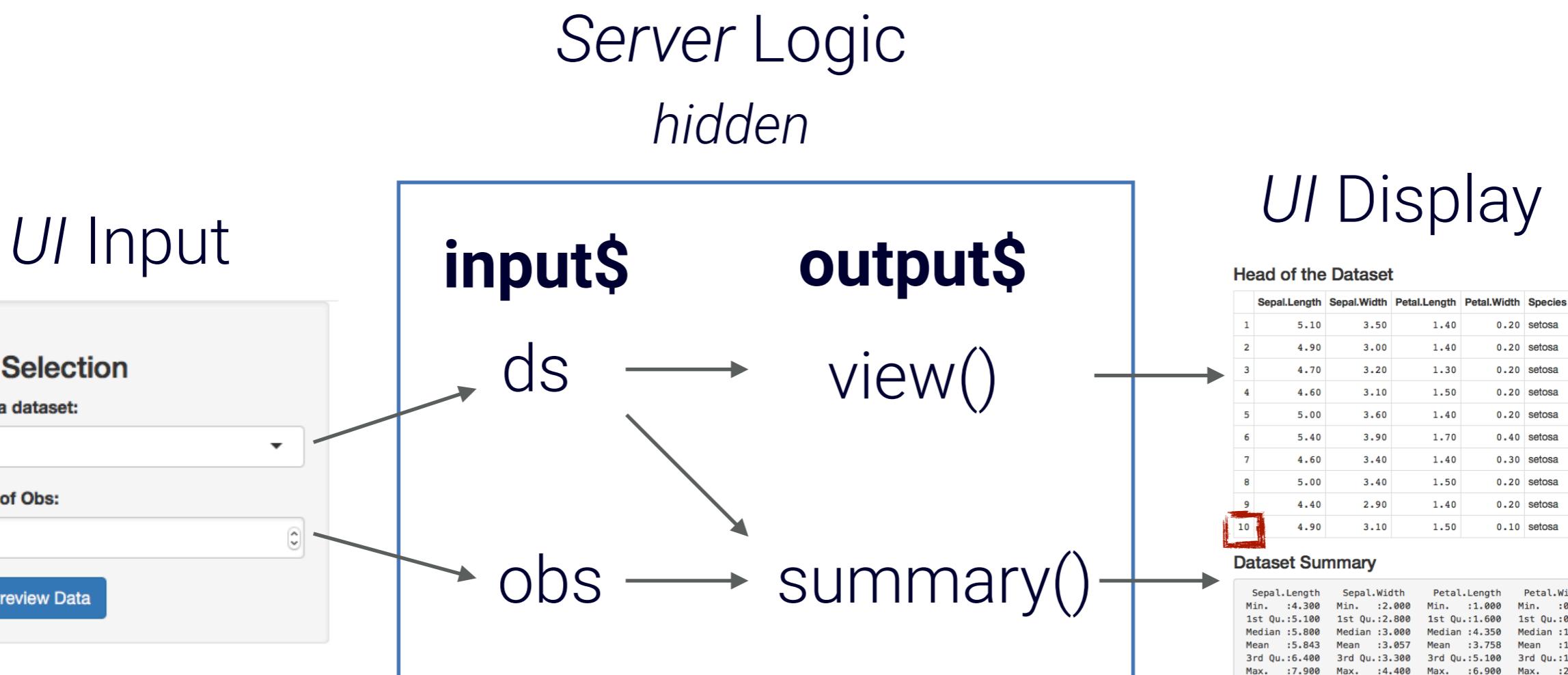
	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.10	3.50	1.40	0.20	setosa
2	4.90	3.00	1.40	0.20	setosa
3	4.70	3.20	1.30	0.20	setosa
4	4.60	3.10	1.50	0.20	setosa
5	5.00	3.60	1.40	0.20	setosa
6	5.40	3.90	1.70	0.40	setosa
7	4.60	3.40	1.40	0.30	setosa
8	5.00	3.40	1.50	0.20	setosa
9	4.40	2.90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

Dataset Summary: A table titled "Dataset Summary" showing statistical summaries for each column.

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 ...



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.

Data Summarizer

Data Selection

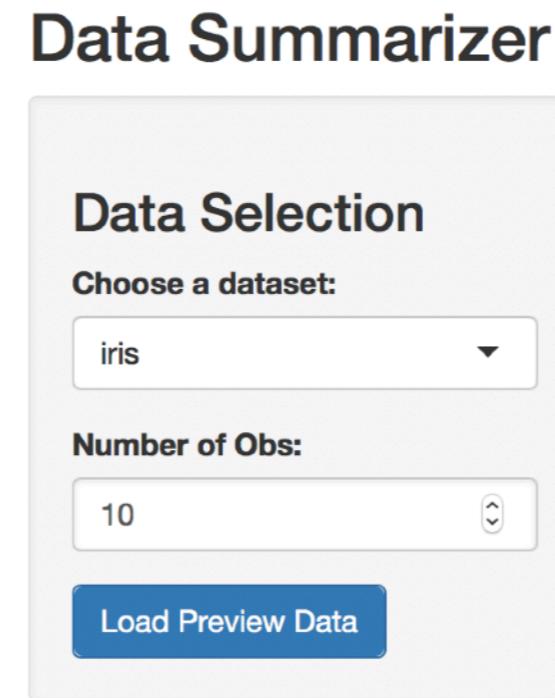
Choose a dataset:

iris

Number of Obs:

10

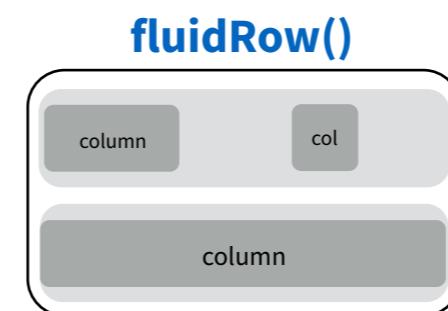
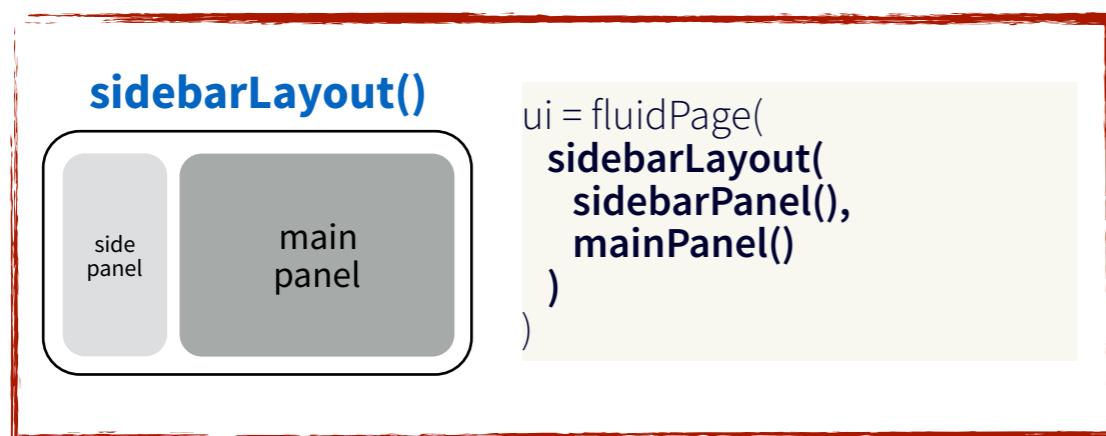
Load Preview Data



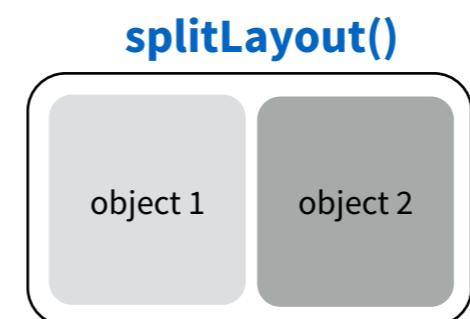
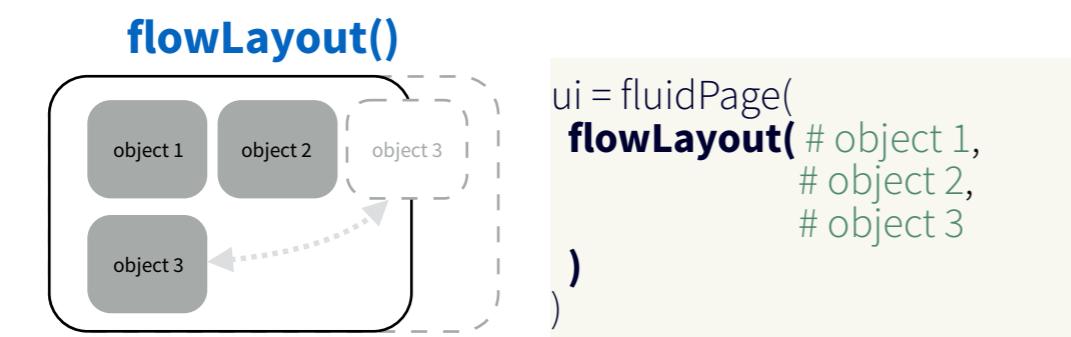
Head of the Dataset
Dataset Summary

Layouts

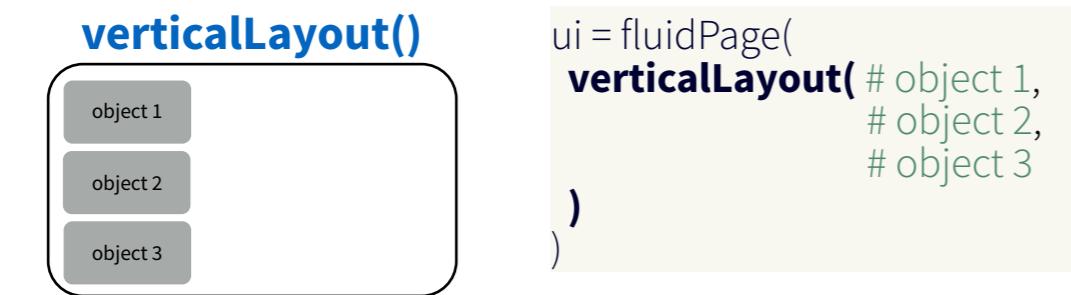
... applying styling to UI ...



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



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



Pagination Layouts

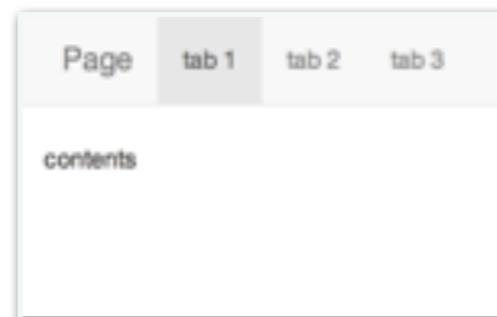
... splitting content up ...



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



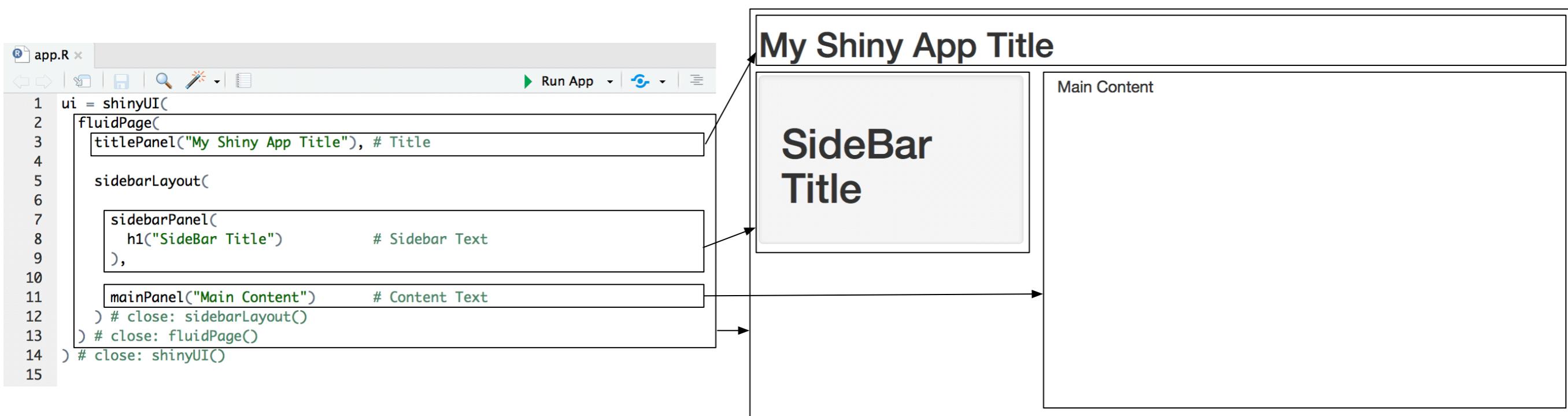
```
ui = fluidPage(navlistPanel(  
  tabPanel("tab 1", "contents"),  
  tabPanel("tab 2", "contents"),  
  tabPanel("tab 3", "contents")))
```



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

UI Implemented

... using a layout in practice ...



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 ...

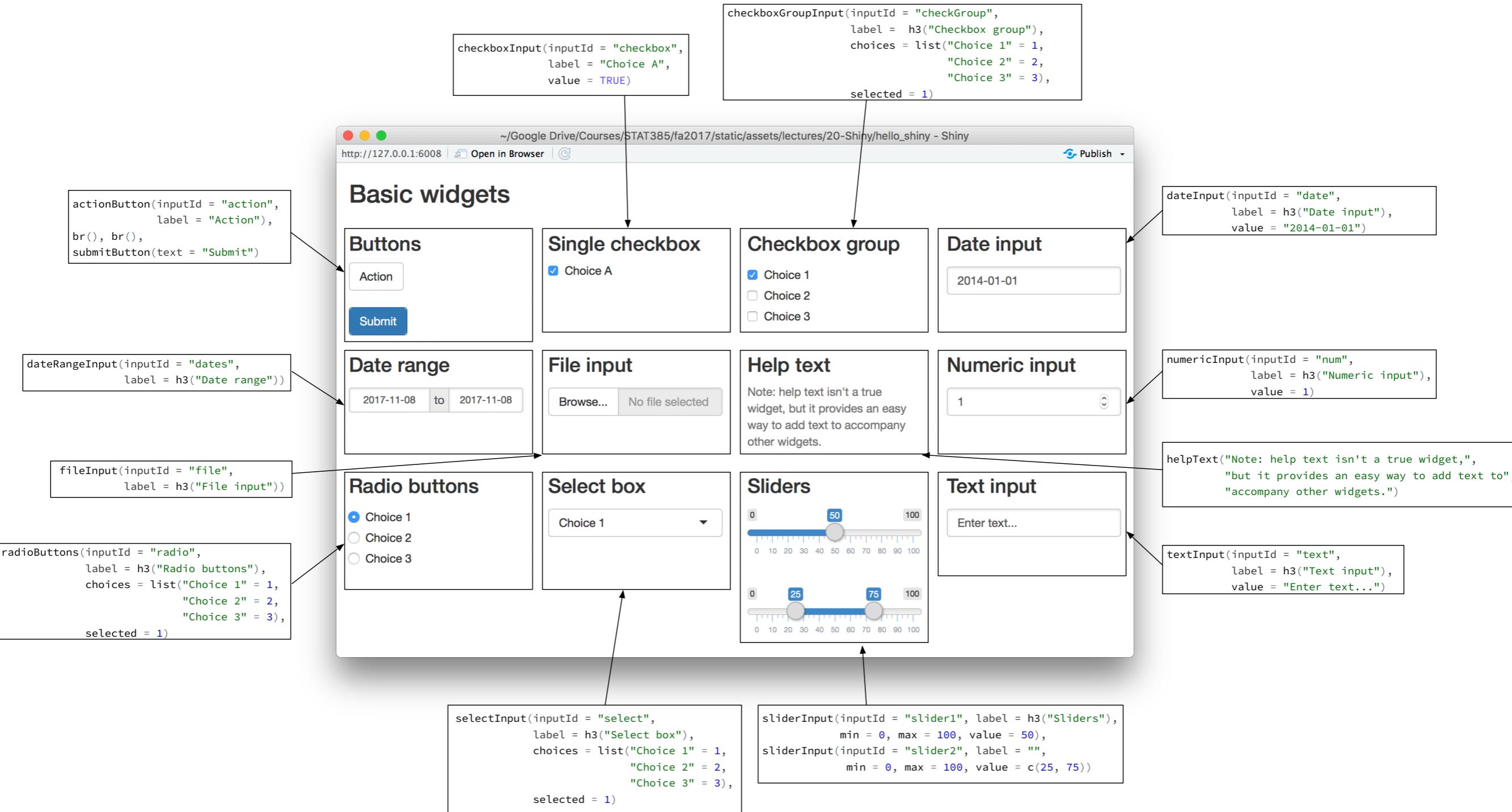
Function	HTML	Rmd	Description
strong()		** **	Bold Text
em()		_ _	<i>Italicize Text</i>
a()	<a>	[title](url)	<u>Makes a hyperlink</u>
p()	<p></p>	Two newlines	Text Paragraph
h1()	<h1></h1>	# (h1), ## (h2), ### (h3)	Header Text
br()	 	Two newlines	Creates a page break
code()	<code></code>	` code here `	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 ...



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 ...

The image shows the RStudio environment with an open file named 'app.R'. The code defines a Shiny user interface (UI) for a 'Data Summarizer' application. The UI consists of a sidebar panel titled 'Data Selection' containing a dropdown menu for dataset selection and a numeric input for the number of observations. A main panel titled 'Main Content' contains a button to load preview data. Arrows point from the corresponding code blocks in 'app.R' to their respective components in the 'Data Summarizer' app interface.

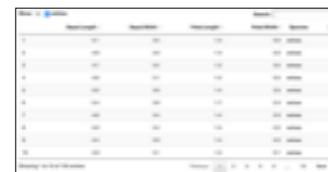
```
1 ui = shinyUI(
2   fluidPage(
3     titlePanel("Data Summarizer"),           # Title
4     sidebarLayout(
5       sidebarPanel(
6         h3("Data Selection"),                 # Note the ,
7
8         # Dropdown
9         selectInput("ds",                  # Name
10            "Choose a dataset:",          # 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("Main Content")          # Content
20    ) # close: sidebarLayout()
21 )) # close: shinyUI(fluidPage)
```

Main Content

Output Areas

... defining output ...

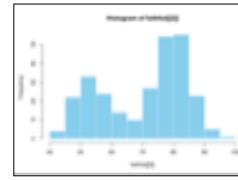
UI



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)

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

verbatimTextOutput(outputId)



tableOutput(outputId)

foo

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 ...

The diagram illustrates the mapping between the R code in the 'app.R' file and the resulting shiny application interface.

app.R Code:

```
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 Aasis
25    ) # close: mainPanel()
26  ) # close: sidebarLayout()
27) # close: shinyUI(fluidPage(
```

shiny Application Interface:

The shiny application has a title 'Data Summarizer'. The 'Data Selection' sidebar panel contains a dropdown menu set to 'iris' and a numeric input field set to 10, with a 'Load Preview Data' button. The 'mainPanel' area displays two output sections: 'Head of the Dataset' (tableOutput) and 'Dataset Summary' (verbatimTextOutput).

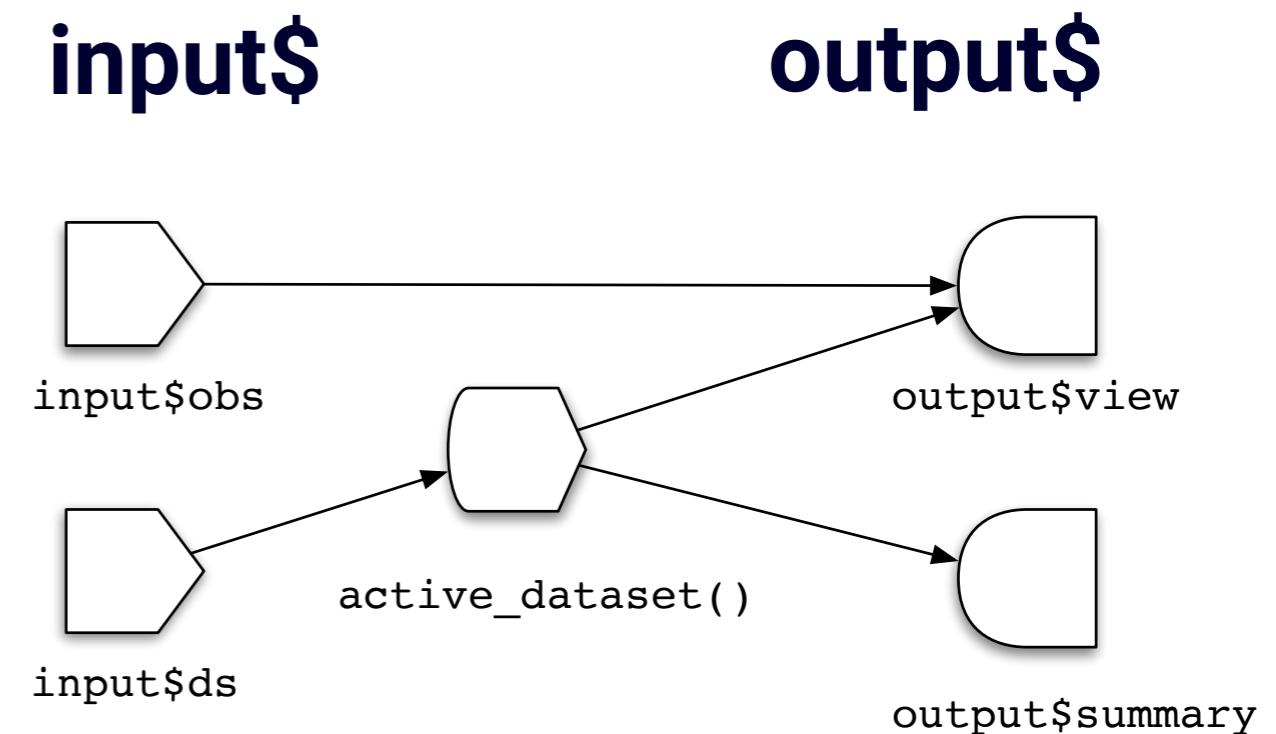
Arrows point from specific lines of code in the R script to their corresponding UI components in the application interface, showing how each part of the code defines a UI element.

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

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

renderImage(expr, env, quoted, deleteFile)

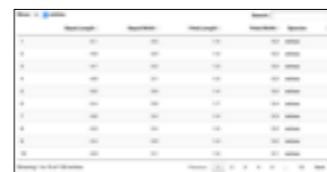
renderPlot(expr, width, height, res, ..., env, quoted, func)

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

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

renderText(expr, env, quoted, func)

renderUI(expr, env, quoted, func)

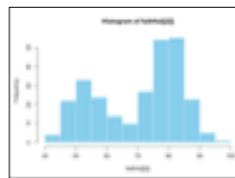


UI

dataTableOutput(outputId, icon, ...)



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



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

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

	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	3.00	1.40	0.20	setosa
6	5.40	3.90	1.70	0.40	versicolor

verbatimTextOutput(outputId)

tableOutput(outputId)

foo



textOutput(outputId, container, inline)

uiOutput(outputId, inline, container, ...)

htmlOutput(outputId, inline, container, ...)

Connection

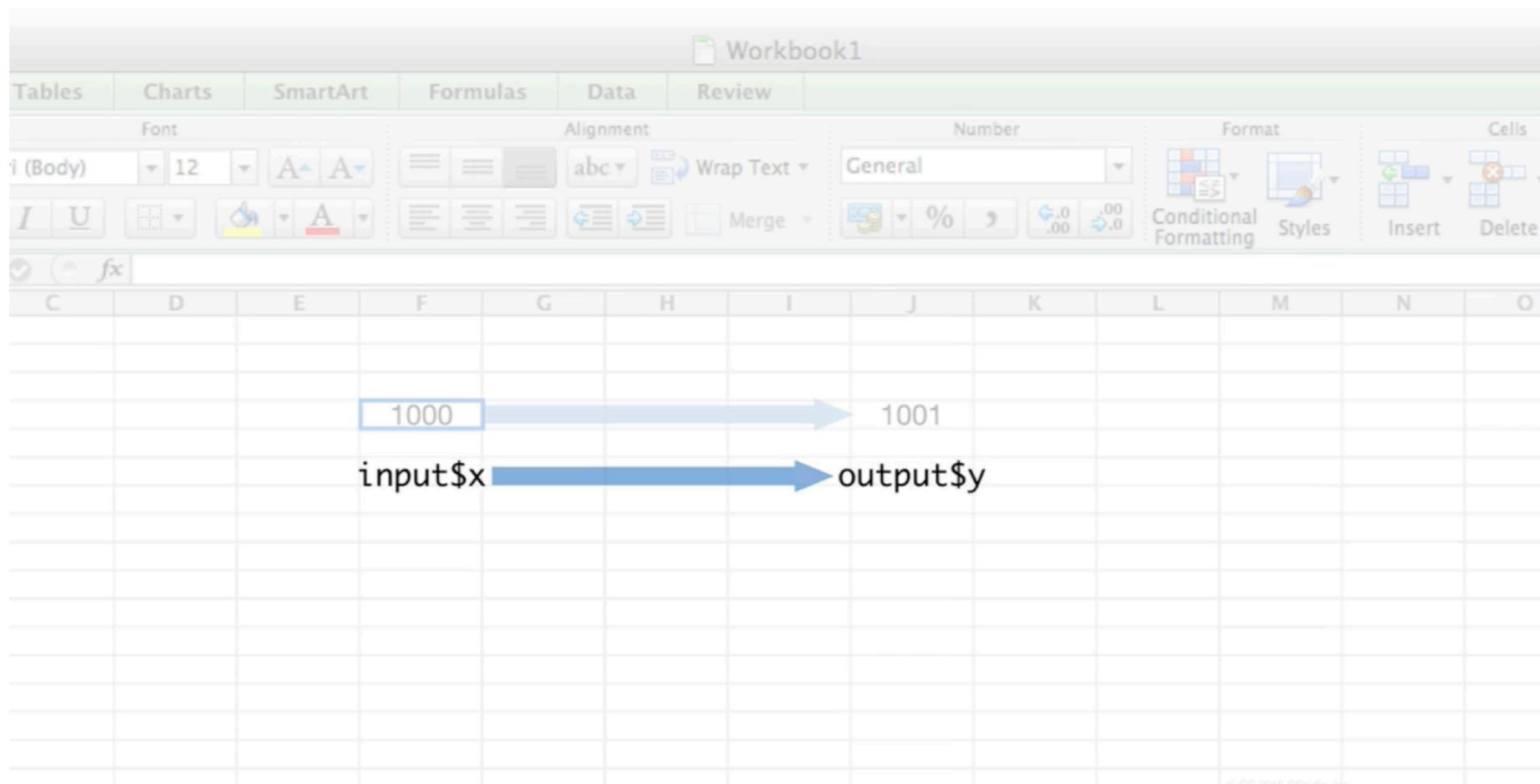
... relation between UI and Server components ...

```
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

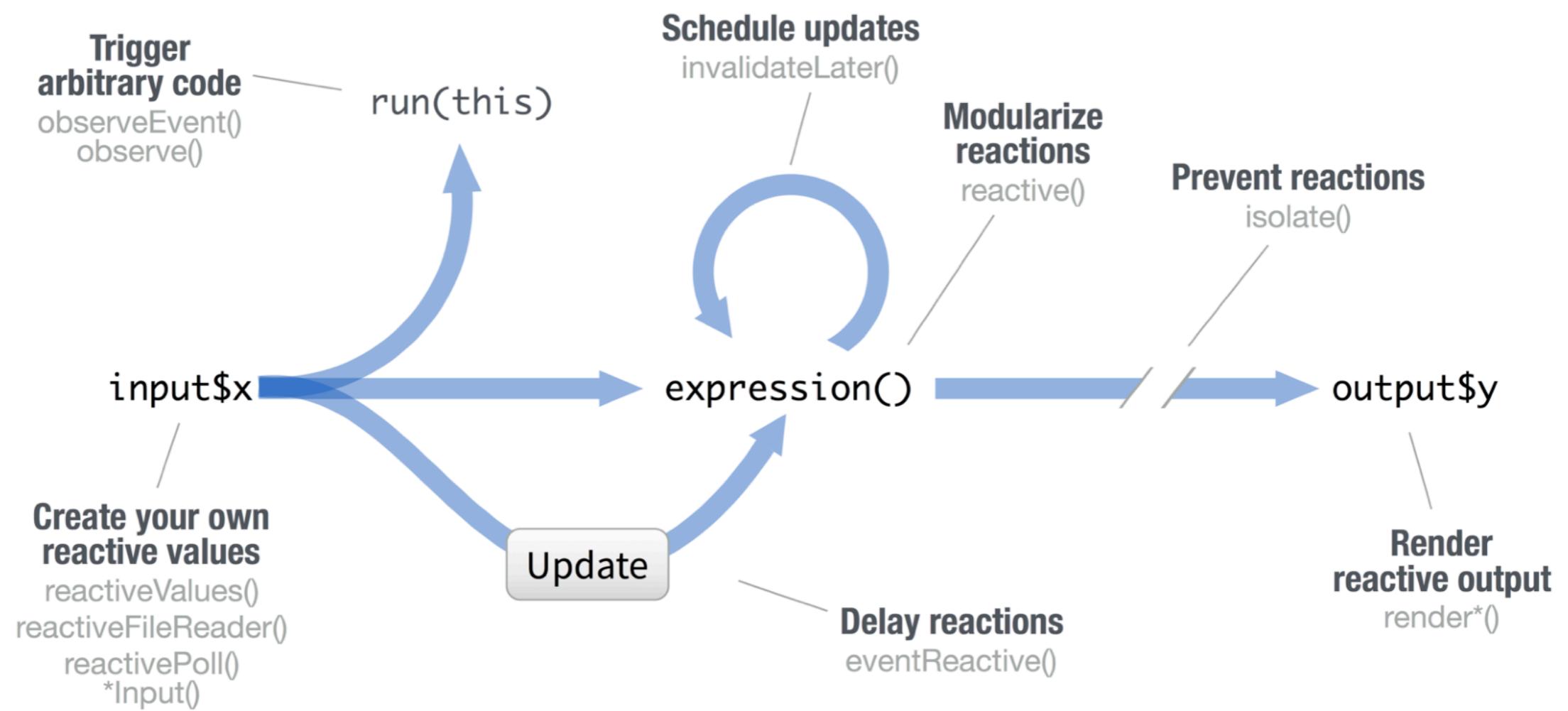
... formulaic updates ...



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 value
(implementation of reactive source)



Reactive expression
(implementation of reactive conductor)



Observer
(implementation of reactive endpoint)



- 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

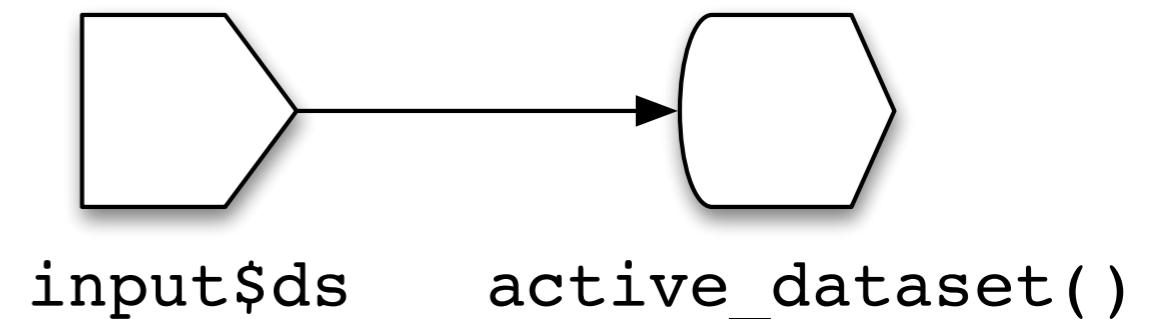
```
# Server component
server = function(input, output) {

# 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...

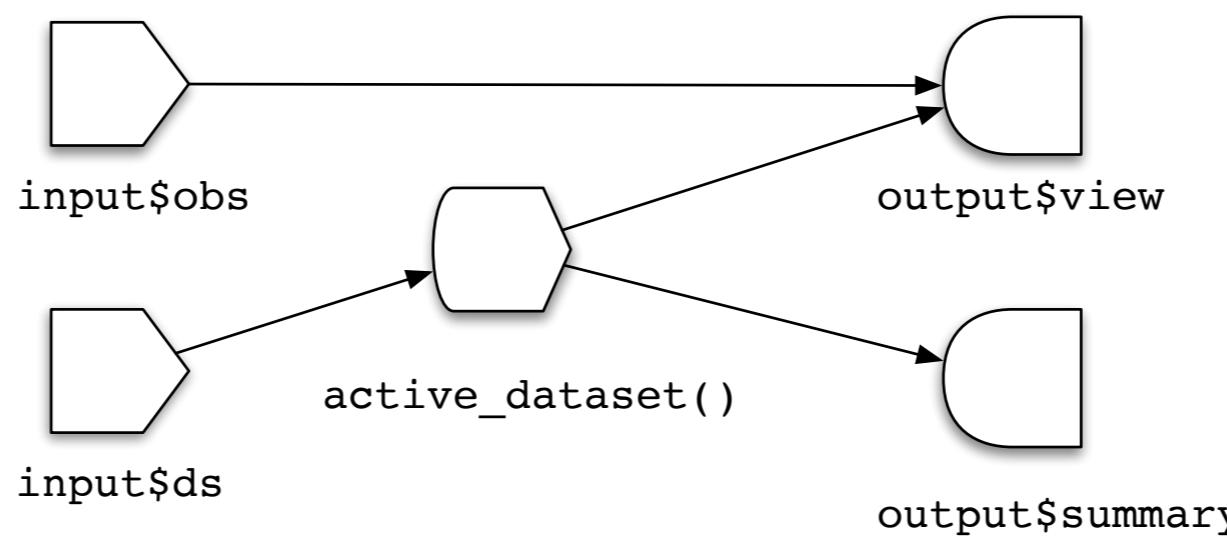
The screenshot shows the RStudio interface with the file 'app.R' open. The code defines a Shiny application with UI and server components. Annotations highlight reactive variables: 'ds' and 'obs' in the UI, and 'active_dataset()', 'output\$view', 'output\$summary', and 'active_dataset()' in the server component. A legend at the bottom identifies the colors: yellow for Reactive Source, blue for Reactive Endpoint, and red for Reactive Conductor.

```
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 As Is
25      ) # close: mainPanel()
26    ) # close: sidebarLayout()
27  ) # close: shinyUI(fluidPage(
28
```

```
31 server = shinyServer(function(input, output) {
32 
33   active_dataset = reactive{           # Reactive Conductor
34     if(input$ds == "iris") {
35       iris
36     } else if (input$ds == "Spam") {
37       Spam
38     } else {
39       mtcars
40     }
41   } # close: reactive()
42 
43   output$summary = renderPrint{        # Summary Render
44     summary(active_dataset())
45   } # close: renderPrint()
46 
47   output$view = renderTable{          # Table Render
48     head(active_dataset(),
49       n = input$obs)
50   } # close: renderPrint()
51 
52 }) # close: shinyServer()
53
```

Legend:

- Yellow line: Reactive Source
- Blue line: Reactive Endpoint
- Red line: Reactive Conductor



Final App

http://127.0.0.1:5640 | Open in Browser |

~/Desktop/hello_shiny - Shiny

Publish ▾

Data Selection

Choose a dataset:

iris

Number of Obs:

10

Load Preview Data

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	

Debugging

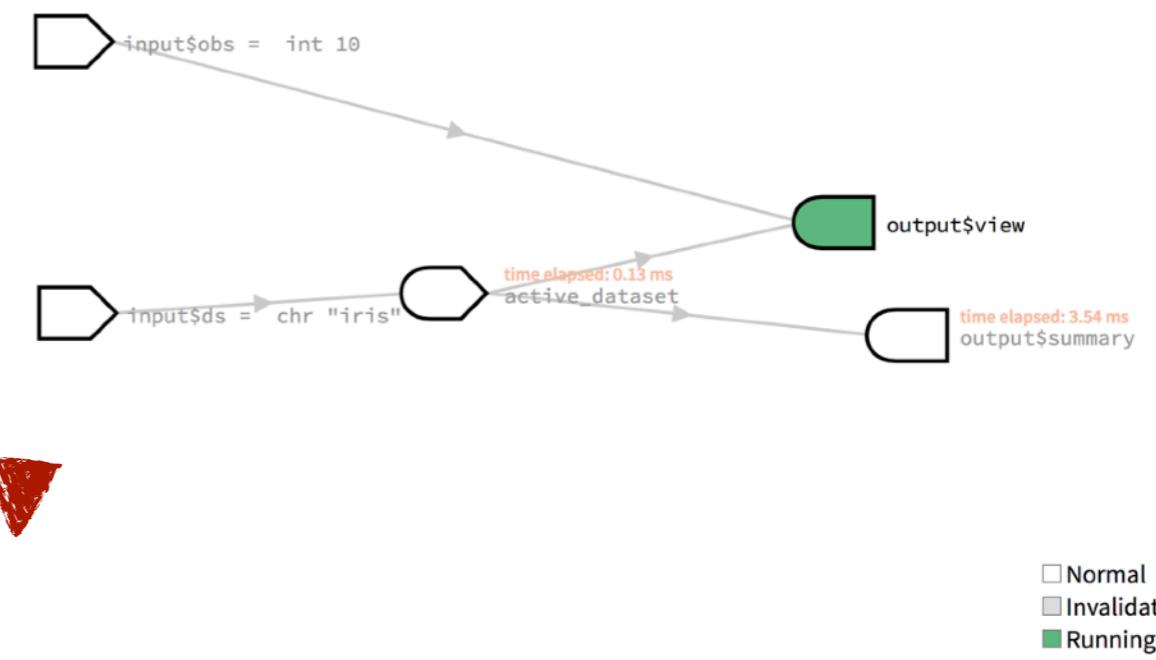
Use a Version Control System

Always Be Committing

```
# 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 ...

Data Summarizer

Data Selection

Choose a dataset:
iris

Number of Obs:
5

Load Preview Data

Head of the Dataset

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.10	3.50	1.40	0.20	setosa
4.90	3.00	1.40	0.20	setosa
4.70	3.20	1.30	0.20	setosa
4.60	3.10	1.50	0.20	setosa
5.00	3.60	1.40	0.20	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
1st Qu.:5.000	1st Qu.:2.800	1st Qu.:1.600	1st Qu.:0.800	versicolor
Median :5.800	Median :3.000	Median :4.350	Median :1.300	virginica
Mean :5.843	Mean :3.057	Mean :4.375	Mean :1.500	
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	

```
app.R
```

```
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 Axis  
) # close: mainPanel()  
) # close: sidebarLayout()  
) # close: shinyUI(fluidPage())  
  
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()
```

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 ...

Shiny :: CHEAT SHEET

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 an app. Wrap with **runApp()** if calling from a sourced script or inside a function.

Building an App

Complete the template by adding arguments to `fluidPage()` and a body to the `server` function.

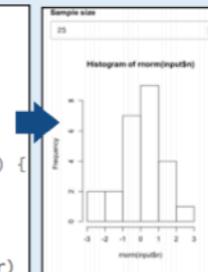
Add inputs to the UI with `*Input()` functions
Add outputs with `*Output()` functions
Tell server how to render outputs with R in the server function. To do this:
1. Refer to outputs with `output$<id>`
2. Refer to inputs with `input$<id>`
3. Wrap code in a `render*()` function before saving to output

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)
```

Save each app as a directory that holds 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"



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()**.

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, ...)`

checkbox1
checkbox2
checkbox3
checkbox4
`checkboxGroupInput(inputId, label, choices, selected, inline)`

checkboxInput
`checkboxInput(inputId, label, value)`

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

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

fileInput
`fileInput(inputId, label, multiple, accept)`

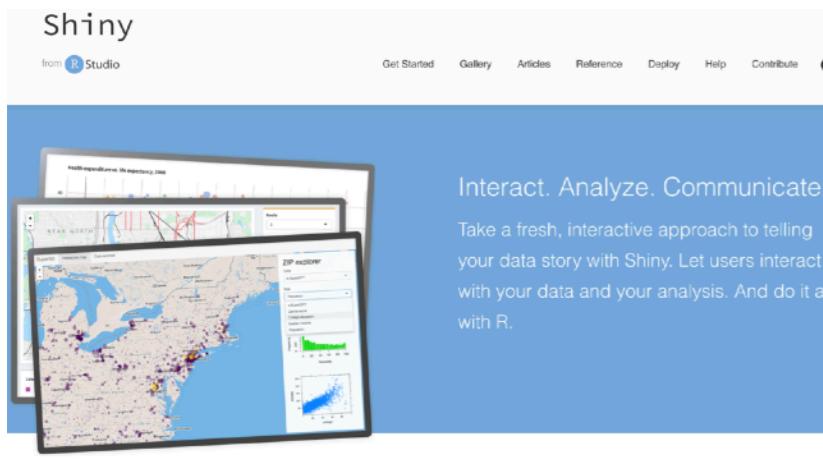
numericInput
`numericInput(inputId, label, value, min, max, step)`

passwordInput
`passwordInput(inputId, label, value)`

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

Shiny Websites

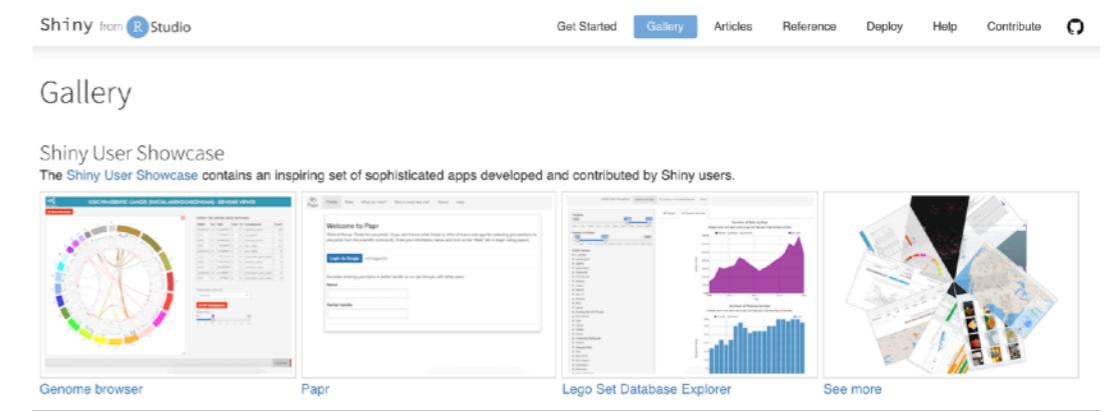
... official shiny websites backed by RStudio ...



Interact. Analyze. Communicate.

Take a fresh, interactive approach to telling your data story with Shiny. Let users interact with your data and your analysis. And do it all with R.

[Shiny's Website](#)



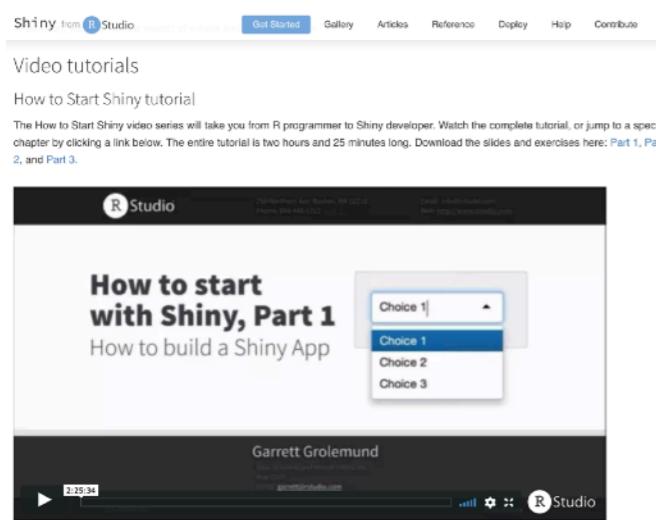
Gallery

Shiny User Showcase

The Shiny User Showcase contains an inspiring set of sophisticated apps developed and contributed by Shiny users.

Genome browser Paper Lego Set Database Explorer See more

[Shiny App Gallery](#)



Video tutorials

How to Start Shiny tutorial

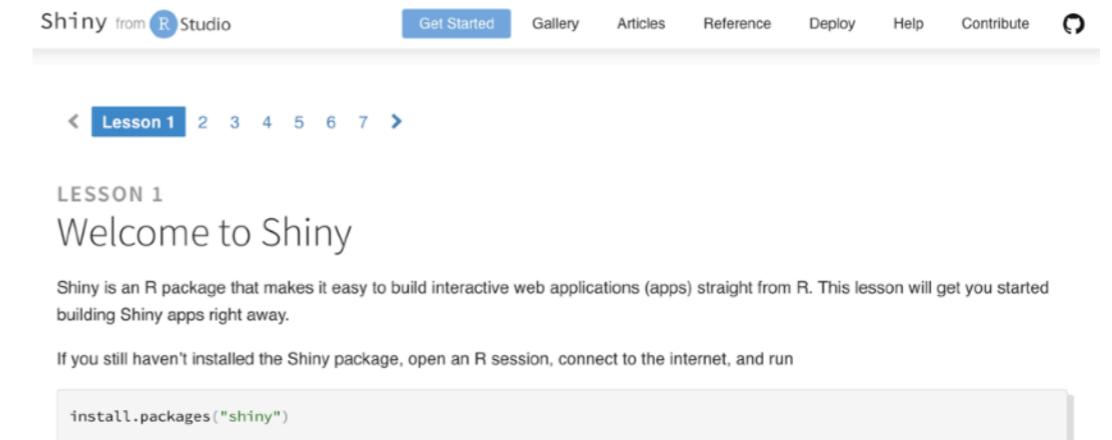
The How to Start Shiny video series will take you from R programmer to Shiny developer. Watch the complete tutorial, or jump to a specific chapter by clicking a link below. The entire tutorial is two hours and 25 minutes long. Download the slides and exercises here: [Part 1](#), [Part 2](#), and [Part 3](#).

How to start with Shiny, Part 1

How to build a Shiny App

Garrett Grolemund

[Video Tutorials](#)



Lesson 1 2 3 4 5 6 7

LESSON 1

Welcome to Shiny

Shiny is an R package that makes it easy to build interactive web applications (apps) straight from R. This lesson will get you started building Shiny apps right away.

If you still haven't installed the Shiny package, open an R session, connect to the internet, and run

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

[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](#)