R Shiny

Intro to Data Science

Shawn Santo

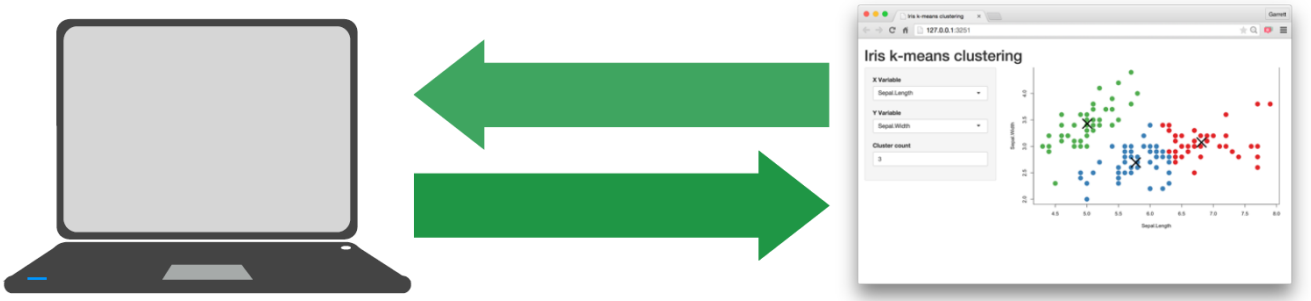
Announcements

- Homework 05 due today
- Submit your GitHub issues for peer review today
- Friday is a project workday

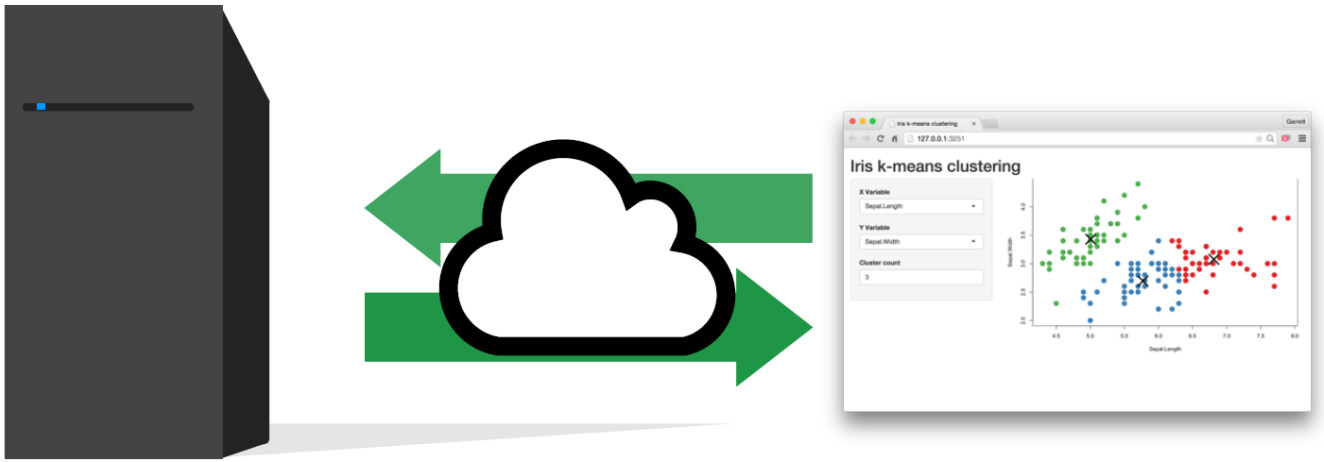
What is Shiny?

- Shiny is an R package.
- Build web-based apps with R in RStudio.
- Shiny can incorporate CSS themes and JavaScript actions.
- To see some apps in action, check out the Shiny gallery.

How does Shiny work?



How does Shiny work?



How does Shiny work?

Getting started

Instead of coding in an `Rmd` file we will code in a script, `.R`, file. To get started

- open RStudio;
- if not installed, run `install.packages("shiny")`;
- go to `File` > `New File` > `Shiny Web App`;
- enter your application's name;
- keep option Single File (app.R) selected;
- enter the directory of where the application should be saved;
- file `app.R` should open, click Run App to see the result.

Skeleton Shiny app

```
 # Load package shiny
library(shiny)
# Define UI for application
ui <- fluidPage(


)

# Define server logic
server <- function(input, output)


}

# Build and run the application
shinyApp(ui = ui, server = server)
```

- Function `fluidPage()` creates a dynamic HTML user interface you see when you look at an RShiny app. Convention is to save this as an object named `ui`.
- Function `server()` is user-defined and contains R commands your computer or external server need to run the app.
- Function `shinyApp()` builds the app based on the user interface and server pair of code.

Available examples

Package `shiny` comes with some example apps. Enter any of the following in your Console to see the Shiny app in action along with the code.

```
runExample("01_hello")          # a histogram
runExample("02_text")           # tables and data frames
runExample("03_reactivity")     # a reactive expression
runExample("04_mpg")            # global variables
runExample("05_sliders")        # slider bars
runExample("06_tabsets")        # tabbed panels
runExample("07_widgets")        # help text and submit buttons
runExample("08_html")           # Shiny app built from HTML
runExample("09_upload")         # file upload wizard
runExample("10_download")       # file download wizard
runExample("11_timer")          # an automated timer
```

Going forward

We're going to build an app that allows users to explore Brexit (EU referendum) poll outcomes for 127 polls from January 2016 to the referendum date on June 23, 2016. The data is available in package `dslabs` and can be loaded with `data("brexit_polls")`. We'll build this app in two steps.

1. We'll first focus on constructing the UI, the widgets available, and layout design our app.

2. Next, we'll learn how to connect our input widgets and outputs with code in the server function.

Goal app

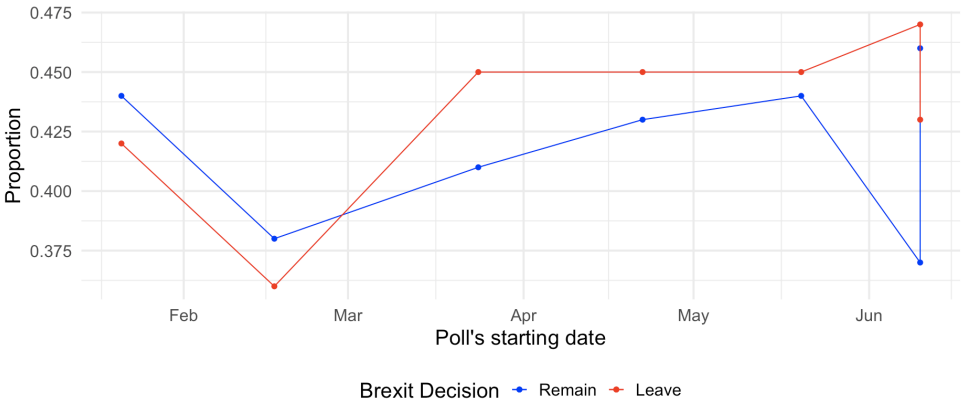
Brexit Polls Explorer

Enter poll start range

2016-01-08 to 2016-06-23

Select pollster

BMG Research



Show 10 entries Search:

	startdate	enddate	pollster	poll_type	samplesize	remain	leave	undecided	spread
1	2016-01-21	2016-01-25	BMG Research	Online	1511	0.44	0.42	0.14	0.02
2	2016-02-17	2016-02-23	BMG Research	Online	1517	0.38	0.36	0.25	0.02
3	2016-03-24	2016-03-29	BMG Research	Online	1518	0.41	0.45	0.14	-0.04
4	2016-04-22	2016-04-26	BMG Research	Online	2001	0.43	0.45	0.13	-0.02
5	2016-05-20	2016-05-25	BMG Research	Online	1638	0.44	0.45	0.12	-0.01
6	2016-06-10	2016-06-15	BMG Research	Online	1468	0.37	0.47	0.16	-0.1
7	2016-06-10	2016-06-15	BMG Research	Telephone	1064	0.46	0.43	0.11	0.03

Showing 1 to 7 of 7 entries Previous 1 Next

User interface

User interface: inputs

Input widgets

Buttons

Action

Submit

`actionButton()`
`submitButton()`

Single checkbox

Choice A

`checkboxInput()`

Checkbox group

Choice 1
Choice 2
Choice 3

`checkboxGroupInput()`

Date input

2014-01-01

`dateInput()`

Date range

2014-01-24 to 2014-01-24

`dateRangeInput()`

File input

Choose File No file chosen

`fileInput()`

Numeric input

1

`numericInput()`

Password Input

••••••••••

`passwordInput()`

Radio buttons

Choice 1
Choice 2
Choice 3

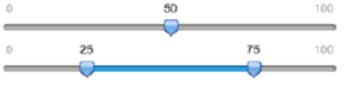
`radioButtons()`

Select box

Choice 1

`selectInput()`

Sliders



`sliderInput()`

Text input

Enter text...

`textInput()`

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)

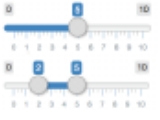
Choose File — **fileInput**(inputId, label, multiple, accept)

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)

Adding an input widget

Most input widgets are set-up as `*Input(inputId, label, ...)` or `*Button(inputId, label, ...)`, where * is replaced with the widget's name.

For example, to create a slider widget we can write

```
sliderInput(inputId = "bins", label = "Number of bins:",
            min = 1, max = 50, value = 30)
```

Typically, the first two widget function argument names are not specified since most widgets first take an `inputId` and `label`. Argument `inputId` is where you specify a name for the widget (this is not seen by the user); argument `label` is the label that will appear in your app (this will be seen by the user).

What do these widget functions return?

If you run

```
sliderInput(inputId = "bins", label = "Number of bins:",
            min = 1, max = 50, value = 30)
```

in you console, you get

```
<div class="form-group shiny-input-container">
  <label class="control-label" for="bins">Number of bins:</label>
  <input class="js-range-slider" id="bins" data-min="1" data-max="50" dat
</div>
```

Some HTML!

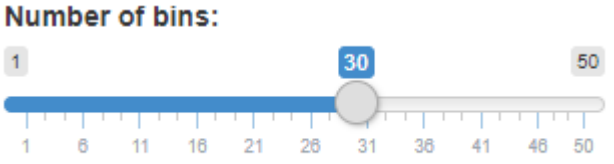
Assortment of input widgets

```
ui <- fluidPage(
  # add slider
  sliderInput("bins", "Number of bins:",
              min = 1, max = 50, value = 30)
)

server <- function(input, output) {

}

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



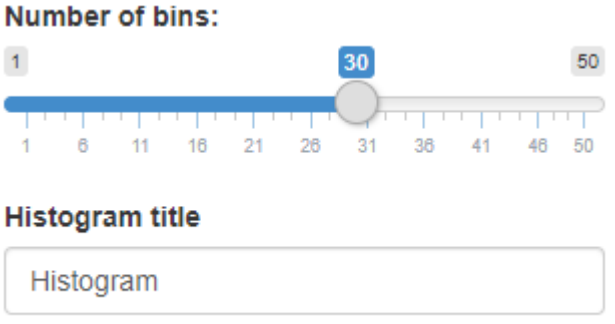
Assortment of input widgets

```
ui <- fluidPage(
  # add slider
  sliderInput("bins", "Number of bins:",
              min = 1, max = 50, value = 30),
  # text box input
  textInput("title", "Histogram title",
            value = "Histogram")
)

server <- function(input, output) {

}

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



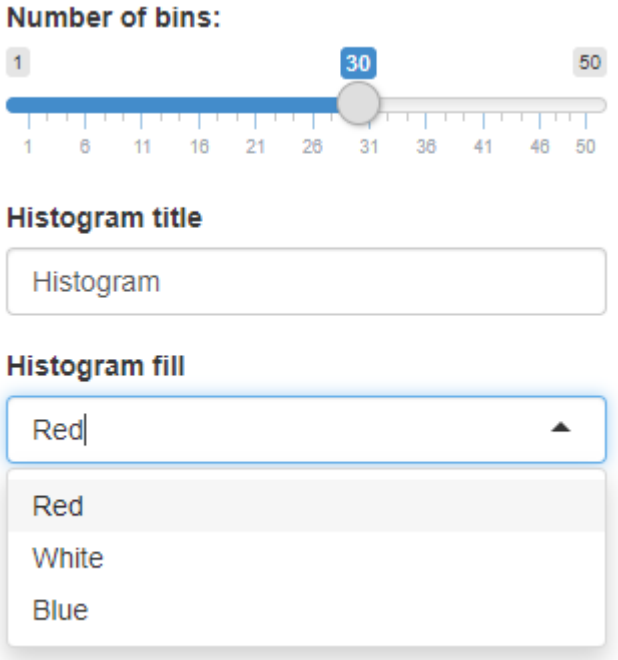
Assortment of input widgets

```
ui <- fluidPage(
  # add slider
  sliderInput("bins", "Number of bins:",
              min = 1, max = 50, value = 30),
  # text box input
  textInput("title", "Histogram title",
            value = "Histogram"),
  # combo box
  selectInput("color", "Histogram fill",
              choices = c("Red", "White", "Bl
              selected = "Red")
)

server <- function(input, output) {

}

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



Continue to add as many additional widgets as you want/need.

User interface: outputs

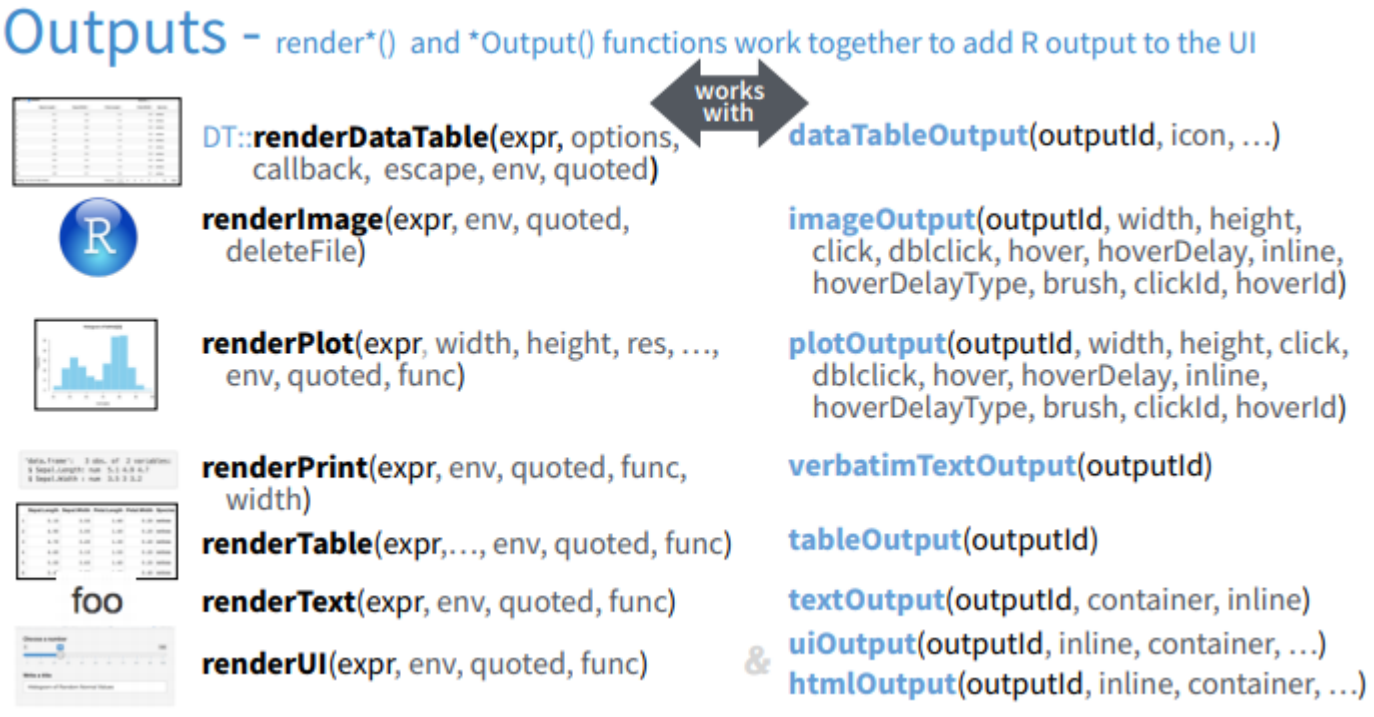
Output functions

Inputs are added with `*Input()`. Similarly, outputs in Shiny are added with `*Output()`.

Output function	Creates
`dataTableOutput()`	data table
`htmlOutput()`	raw HTML
`imageOutput()`	image
`plotOutput()`	plot
`tableOutput()`	table
`textOutput()`	text
`uiOutput()`	raw HTML
`verbatimTextOutput()`	text

Output function details

Focus on the second column for now. We'll learn about the `render*()` functions when we write our server function.



The first argument for each output function is `outputId`. This argument is where you specify a name for the output (this is not seen by the user). This name will serve as reference for code in function `server()`.

Output function

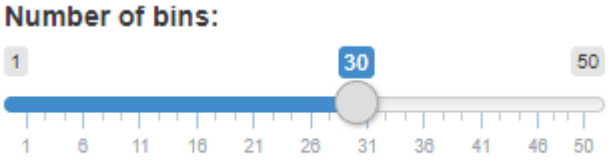
```
ui <- fluidPage(
  # add slider
  sliderInput("bins", "Number of bins:",
              min = 1, max = 50,
              value = 30),

  plotOutput(outputId = "hist")
)

server <- function(input, output) {

}

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



Our code `plotOutput(outputId = "hist")` allocates space for a plot. We haven't created anything yet, hence no plot is visible.

What do these output functions return?

If you run

```
plotOutput(outputId = "hist")
```

in you console, you get

```
<div id="hist" class="shiny-plot-output" style="width: 100% ; height: 400
```

Some HTML!

User interface review

- Build the user interface inside function `fluidPage()` and save it as an object named `ui`.
- Function `fluidPage()` scales its components in realtime to fill all available browser width - dynamic HTML user interface.
- Build inputs with `*Input(inputId, label, ...)`.
- Build outputs with `*Output(outputId, ...)`.
- Separate multiple inputs and outputs with commas.
- Run your app after each added input or output to minimize complications later on.

Exercise: build UI

We're going to build an app that allows users to explore Brexit (EU referendum) poll outcomes for 127 polls from January 2016 to the referendum date on June 23, 2016.

Objectives:

- Use a `sidebarLayout()` with a `sidebarPanel()` and `mainPanel()`
- Add a shiny theme with package `shinythemes`
- Create two input widgets: `dateRangeInput()` and `selectInput()`
- Create two outputs: `plotOutput()` and `dataTableOutput()`

UI preview

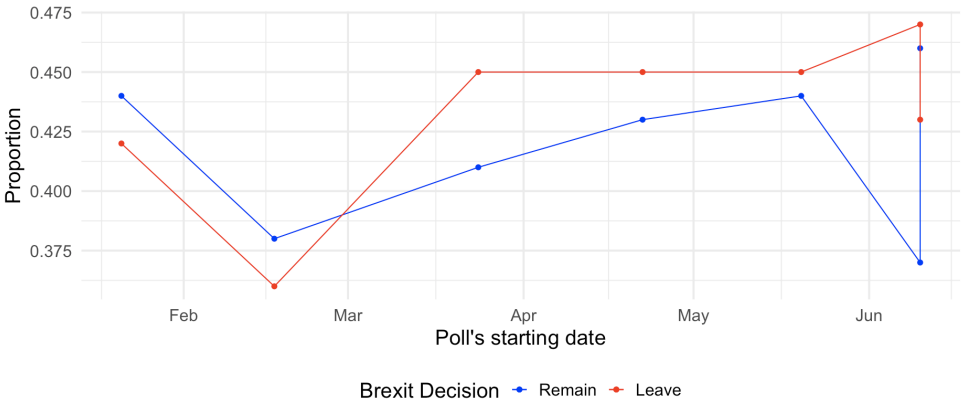
Brexit Polls Explorer

Enter poll start range

2016-01-08 to 2016-06-23

Select pollster

BMG Research



Show 10 entries Search:

	startdate	enddate	pollster	poll_type	samplesize	remain	leave	undecided	spread
1	2016-01-21	2016-01-25	BMG Research	Online	1511	0.44	0.42	0.14	0.02
2	2016-02-17	2016-02-23	BMG Research	Online	1517	0.38	0.36	0.25	0.02
3	2016-03-24	2016-03-29	BMG Research	Online	1518	0.41	0.45	0.14	-0.04
4	2016-04-22	2016-04-26	BMG Research	Online	2001	0.43	0.45	0.13	-0.02
5	2016-05-20	2016-05-25	BMG Research	Online	1638	0.44	0.45	0.12	-0.01
6	2016-06-10	2016-06-15	BMG Research	Online	1468	0.37	0.47	0.16	-0.1
7	2016-06-10	2016-06-15	BMG Research	Telephone	1064	0.46	0.43	0.11	0.03

Showing 1 to 7 of 7 entries Previous 1 Next

Code for UI object is in the presentation notes. Hit P.

Beyond the UI

You have a user interface built. Why does it not do anything? How do I create the plot and table in the image on the previous slide?

You need to give R commands that react when inputs are provided or are changed. These reactions are seen by updates to the outputs. Take a look at https://shiny.rstudio.com/gallery/tabsets.html. As you change inputs, look at what is highlighted in function `server()`.

This is where function `server()`, that you create, will come into play.

Server

Function `server()`

```
server <- function(input, output) {


}
```

This function plays a special role in the Shiny process; it builds a list-like object named `output` that contains all of the code needed to update the R objects in your app. Each R object needs to have its own entry in the list.

You can create an entry by defining a new element for output within the server function. The element name should match the name of the reactive element that you created in the user interface. This is where the `inputId`s and `outputId`s that you define in widgets and outputs come into play.

Steps to create the `server()` function

1. Save objects to display to `output$<outputId>`, where `<outputId>` is the name given from function `*Output()`.

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

2. Build these `output$<outputId>` objects with the family of functions `render*()`.

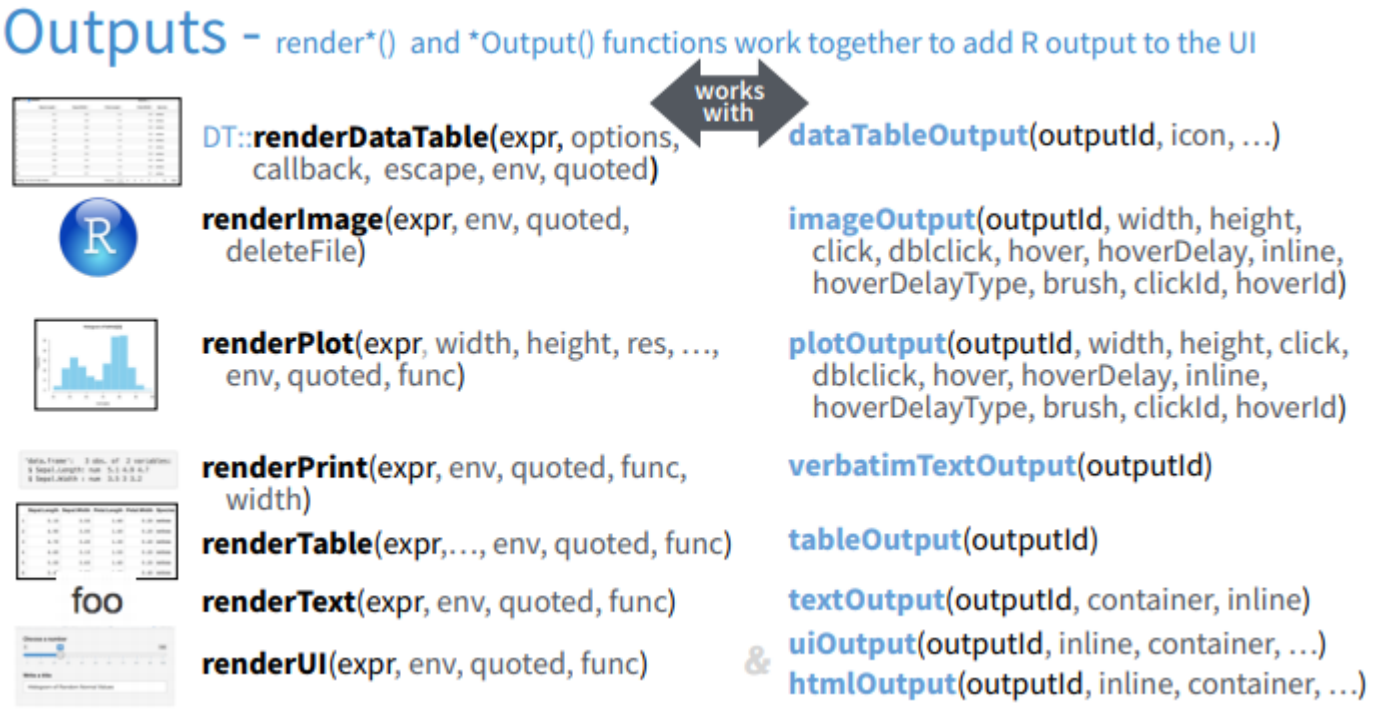
```
server <- function(input, output) {
  output$hist <- renderPlot({
    # code to build your object
    # in this case, code to create
    # the histogram
  })
}
```

3. Access your input values with `input$<inputId>`, where `<inputID>` is the name your provided in function `*Input()`.

Render functions

Render function	Creates a reactive
`renderDataTable()`	data table
`renderImage()`	image
`renderPlot()`	plot
`renderPrint()`	version of the given function that captures print output
`renderTable()`	table
`renderText()`	version of the given function to turn its result into a character vector.
`renderUI()`	HTML

Render and Output connection



Each `render*()` function only requires a single argument: an R expression surrounded by braces, `{ }`. The expression can be one simple line of code, or it can involve many.

Reactivity

Assuming a well-built Shiny app, every time the user moves the slider, selects a value in a combo box, selects a new radio button option, outputs will automatically get updated when inputs change.

This is known as reactivity. Reactivity automatically occurs whenever you use an input value to render an output object.

Function `server()` review

- The server function does the work in terms of building and rebuilding R objects that ultimately get displayed to the user in the user interface.
- Save output you build to `output$<outputId>`.
- Build output with a `render*()` function.
- Access inputs with `input$<inputId>`.
- Multiple outputs can be placed in the server function.
- Reactivity happens automatically when you use inputs to build rendered outputs.

Exercise: server function

We're going to build an app that allows users to explore Brexit (EU referendum) poll outcomes for 127 polls from January 2016 to the referendum date on June 23, 2016.

Objectives:

- Use `renderPlot()` to display a reactive plot that depends on the date range and pollster inputs
- Use `DT::renderDataTable()` to display a reactive table that depends on the date range and pollster inputs. The table should be sorted by start date.

Preview

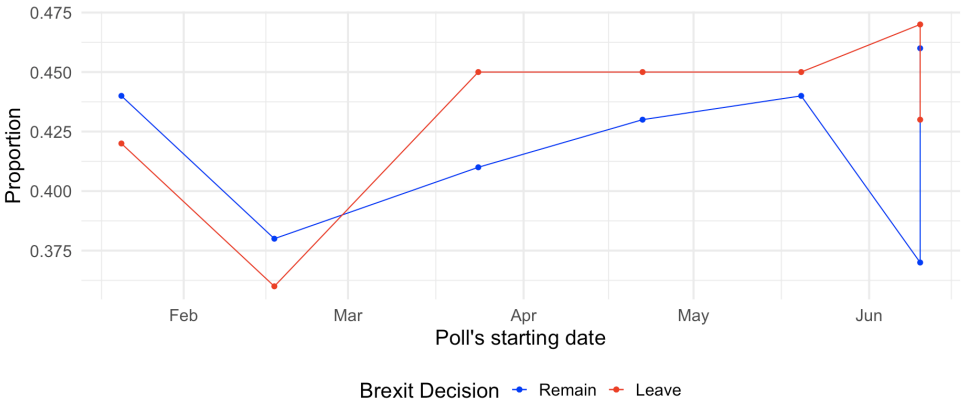
Brexit Polls Explorer

Enter poll start range

2016-01-08 to 2016-06-23

Select pollster

BMG Research



Show 10 entries Search:

	startdate	enddate	pollster	poll_type	samplesize	remain	leave	undecided	spread
1	2016-01-21	2016-01-25	BMG Research	Online	1511	0.44	0.42	0.14	0.02
2	2016-02-17	2016-02-23	BMG Research	Online	1517	0.38	0.36	0.25	0.02
3	2016-03-24	2016-03-29	BMG Research	Online	1518	0.41	0.45	0.14	-0.04
4	2016-04-22	2016-04-26	BMG Research	Online	2001	0.43	0.45	0.13	-0.02
5	2016-05-20	2016-05-25	BMG Research	Online	1638	0.44	0.45	0.12	-0.01
6	2016-06-10	2016-06-15	BMG Research	Online	1468	0.37	0.47	0.16	-0.1
7	2016-06-10	2016-06-15	BMG Research	Telephone	1064	0.46	0.43	0.11	0.03

Showing 1 to 7 of 7 entries Previous 1 Next

Code for only the server function is in the presentation notes. Hit P.

Share your app

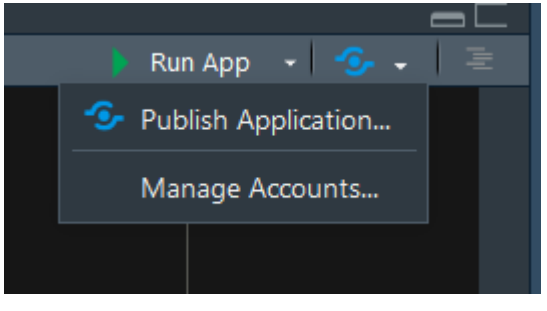
Upload it to shinyapps.io

Distribute your app

1. Create a free account at https://www.shinyapps.io/.

2. Build your Shiny app.

3. Publish your app.



What you get with a free account:

- 5 active applications
- 25 hours per month of active use

References

- Shiny. (2021). Shiny.rstudio.com. https://shiny.rstudio.com/