# R for Environmental Data Analysis Workshop

Rocío Prieto González

Counting Whales

Counting.Whales@gmail.com

Crédits: Laurent Bouveret

# R for Environmental Data Analysis Workshop

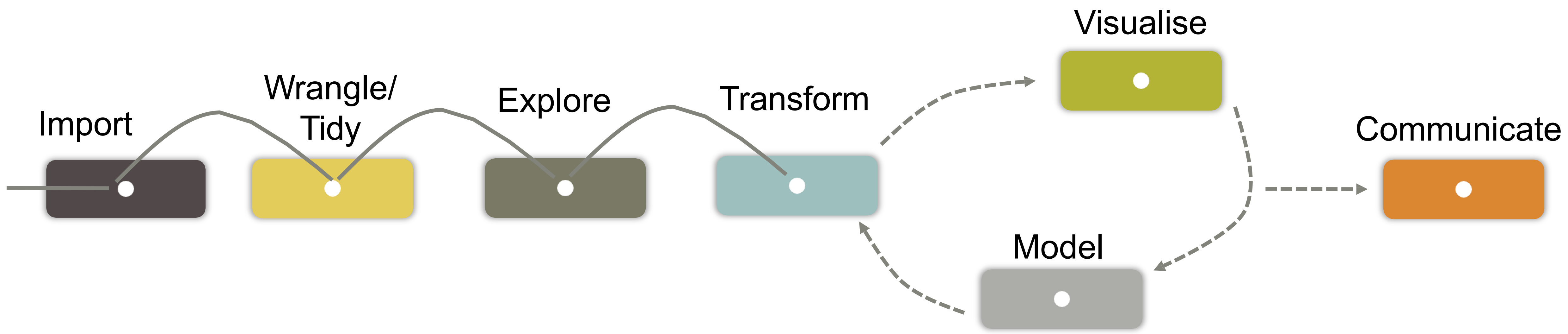
| | | |
|---|---|---|
| ✓ | **Day 1** | Getting started with data management in R |
| ✓ | **Day 2** | Survey design and an introduction to statistical analysis in R |
| | **Day 3** | Advanced sessions – interactive data with Shiny apps, and how to maximize your results using reactive programming |

# R for Environmental Data Analysis Workshop

**Day 3** Advanced sessions – interactive data with Shiny apps, and how to maximize your results using reactive programming

- Importance of data visualitation
- Understand the structure of a Shiny app
- Create a Shiny app
- Get help!!

## Importance of data visualitation



**Data visualisation is crucial in various steps of the data science process**

http://r4ds.had.co.nz/introduction.html

## Why interactive data visualization?

- More **engaging** and better user **experience**
- **Empowers** the user
- Telling a **story** through data
- Easier to understand and **remember**
- **Succinct** communication of data

Bokeh

Power BI

Shiny

D3

Qlik

tableau

Spotfire

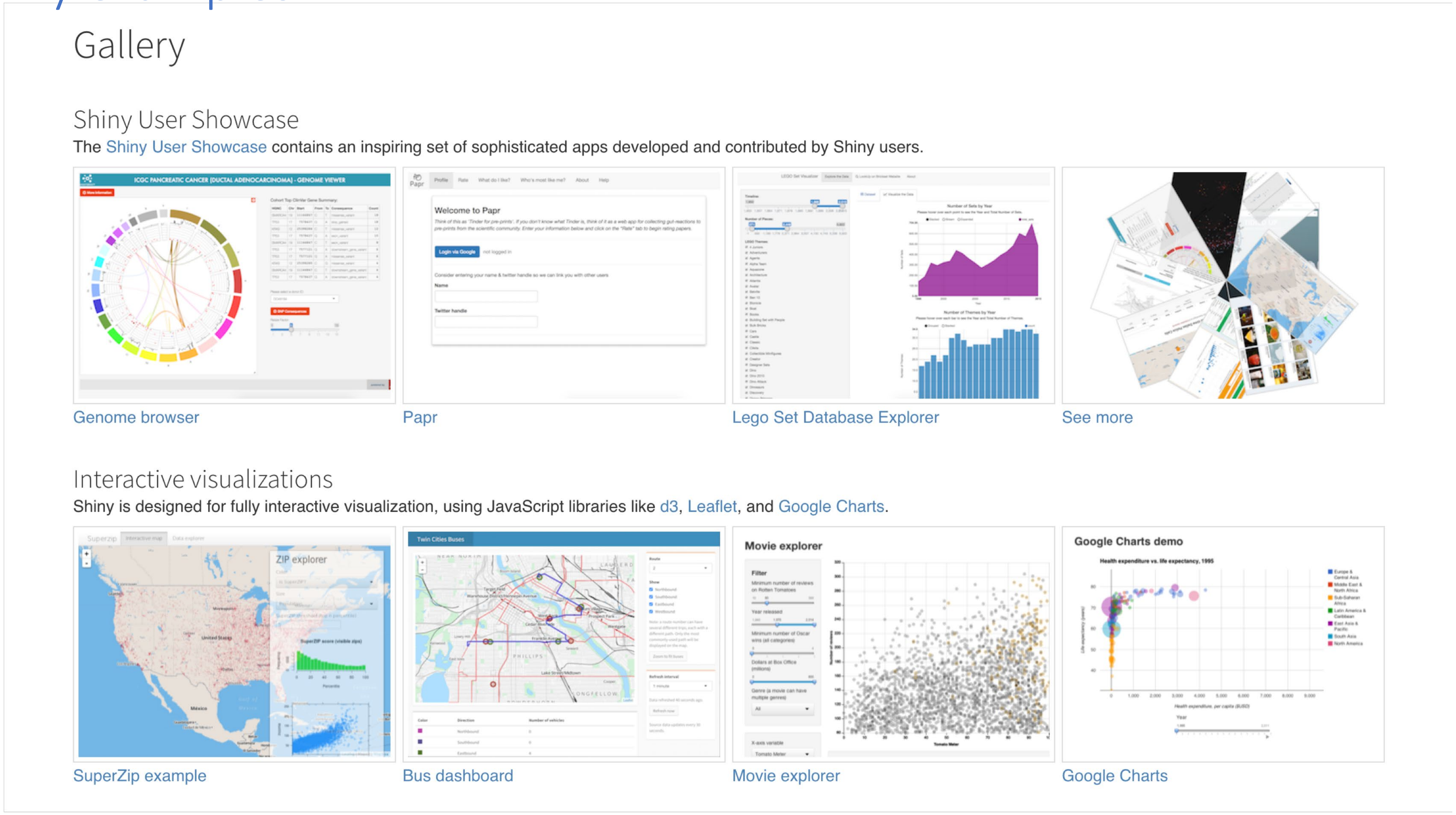
SISENSE

## Why R Shiny?

Web application framework for R that helps turn data analyses into interactive web applications

- **Open** source and **big community**
- Develop for the web using the **same language** as for data exploration and modelling
- No HTML, CSS, or JavaScript knowledge required!
- Pre-built widgets, allowing to build **elegant** and **powerful** applications with minimal effort

# Shiny examples



Gallery

Shiny User Showcase

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

Genome browser · Papr · Lego Set Database Explorer · See more

Interactive visualizations

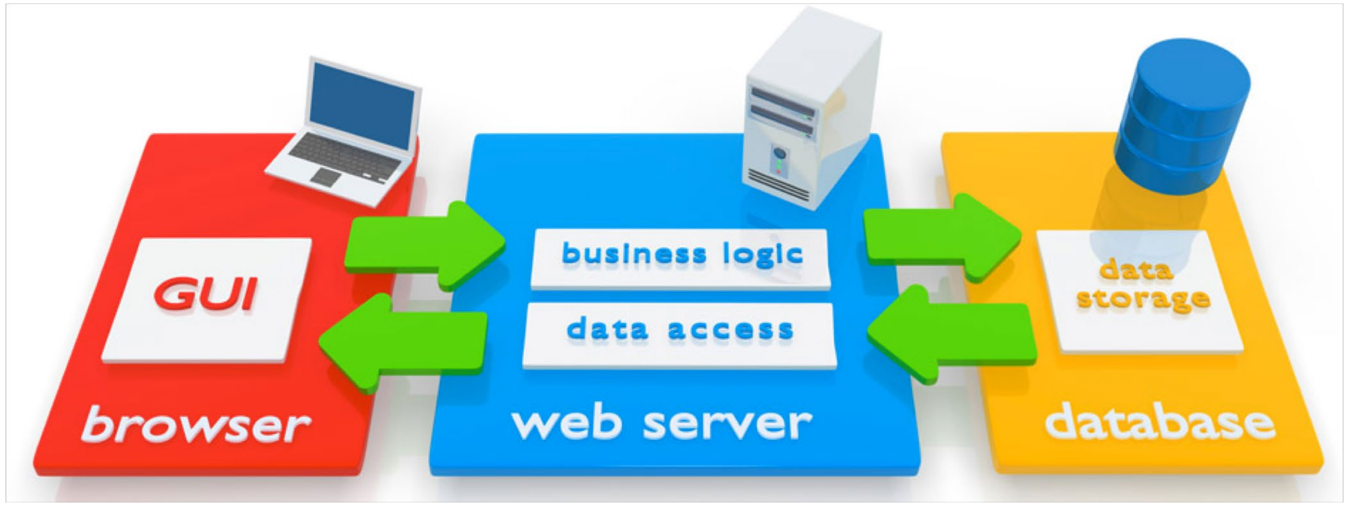
Shiny is designed for fully interactive visualization, using JavaScript libraries like d3, Leaflet, and Google Charts.

SuperZip example · Bus dashboard · Movie explorer · Google Charts

http://shiny.rstudio.com/gallery/

## Webapps

- **Web app:** application program stored on a remote server and accessed via a web browser.



**Front-end**

- Producing HTML, CSS and JavaScript for a website or web application so that a user can see and interact with them directly

**Back-end**

- Creates the logical back-end and core computational logic of a website

https://bookdown.org/weicheng/shinyTutorial/web-development.html

# Understand the structure of a Shiny app

## Building a shiny app

**A Shiny app is composed of 2 parts:**

User Interface (UI)

- ✓ **UI:** web document that the user gets to see
- ✓ **Server:** set of instructions that tell the web page what to show when the user interacts with the page

Server Instructions

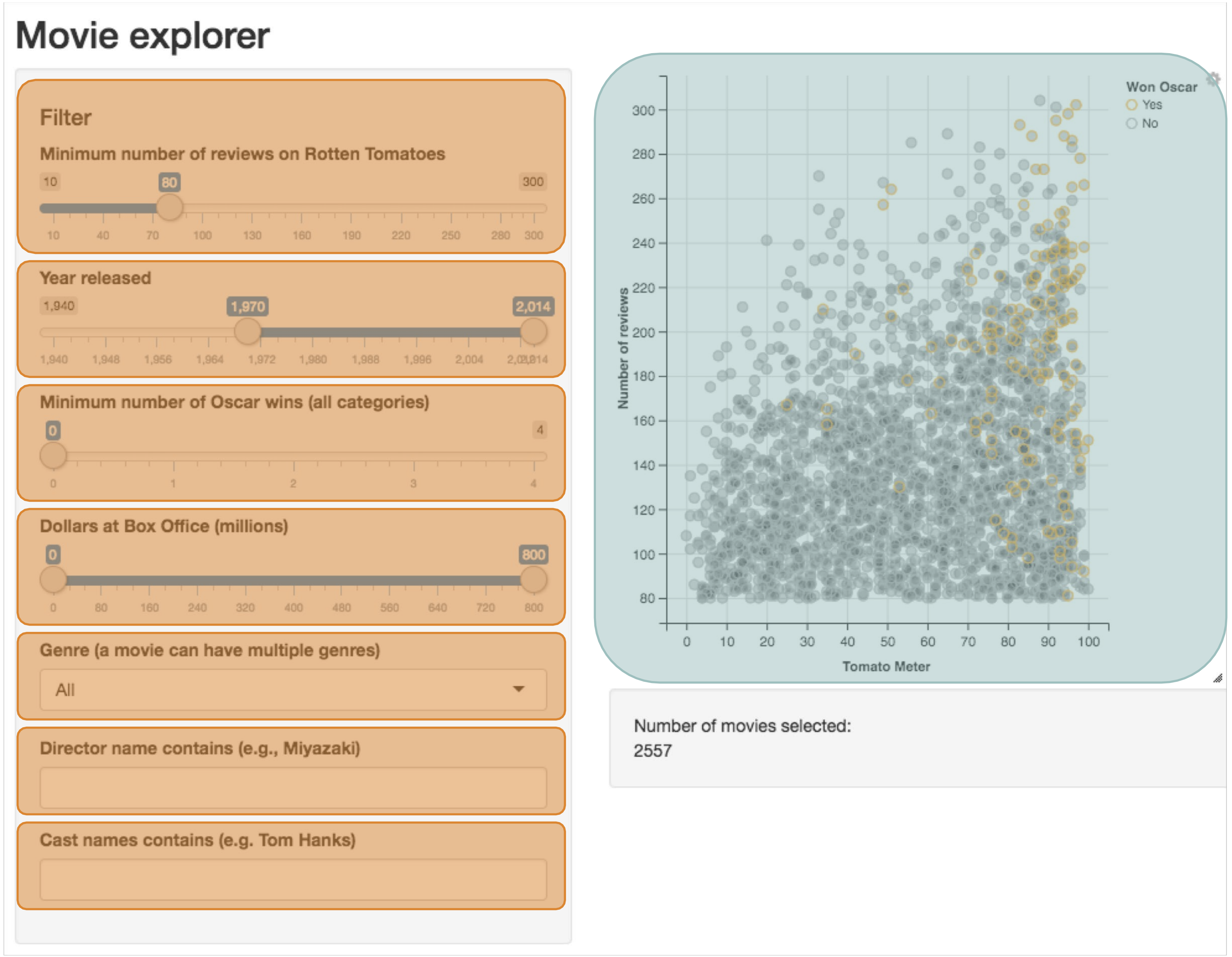
**How does a Shiny app work?**

- ➢ The *Server* keeps monitoring the *UI*. Whenever there is a change in the *UI*, the *Server* will follow some instructions (run some R code) accordingly and update the *UI* (concept of **reactivity**)

https://deanattali.com/blog/building-shiny-apps-tutorial/

# Shiny components

- **Shiny apps are built around inputs and outputs**



## Movie explorer

Filter

Minimum number of reviews on Rotten Tomatoes: 80 (range 10–300)

Year released: 1,970 – 2,014 (range 1,940–2,014)

Minimum number of Oscar wins (all categories): 0 (range 0–4)

Dollars at Box Office (millions): 0 – 800 (range 0–800)

Genre (a movie can have multiple genres): All

Director name contains (e.g., Miyazaki)

Cast names contains (e.g. Tom Hanks)

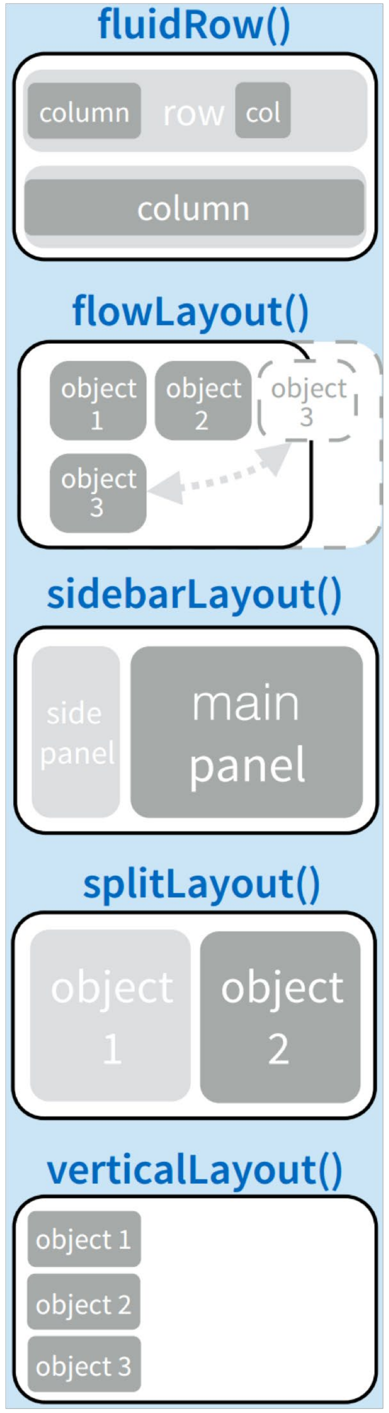
Number of movies selected:
2557

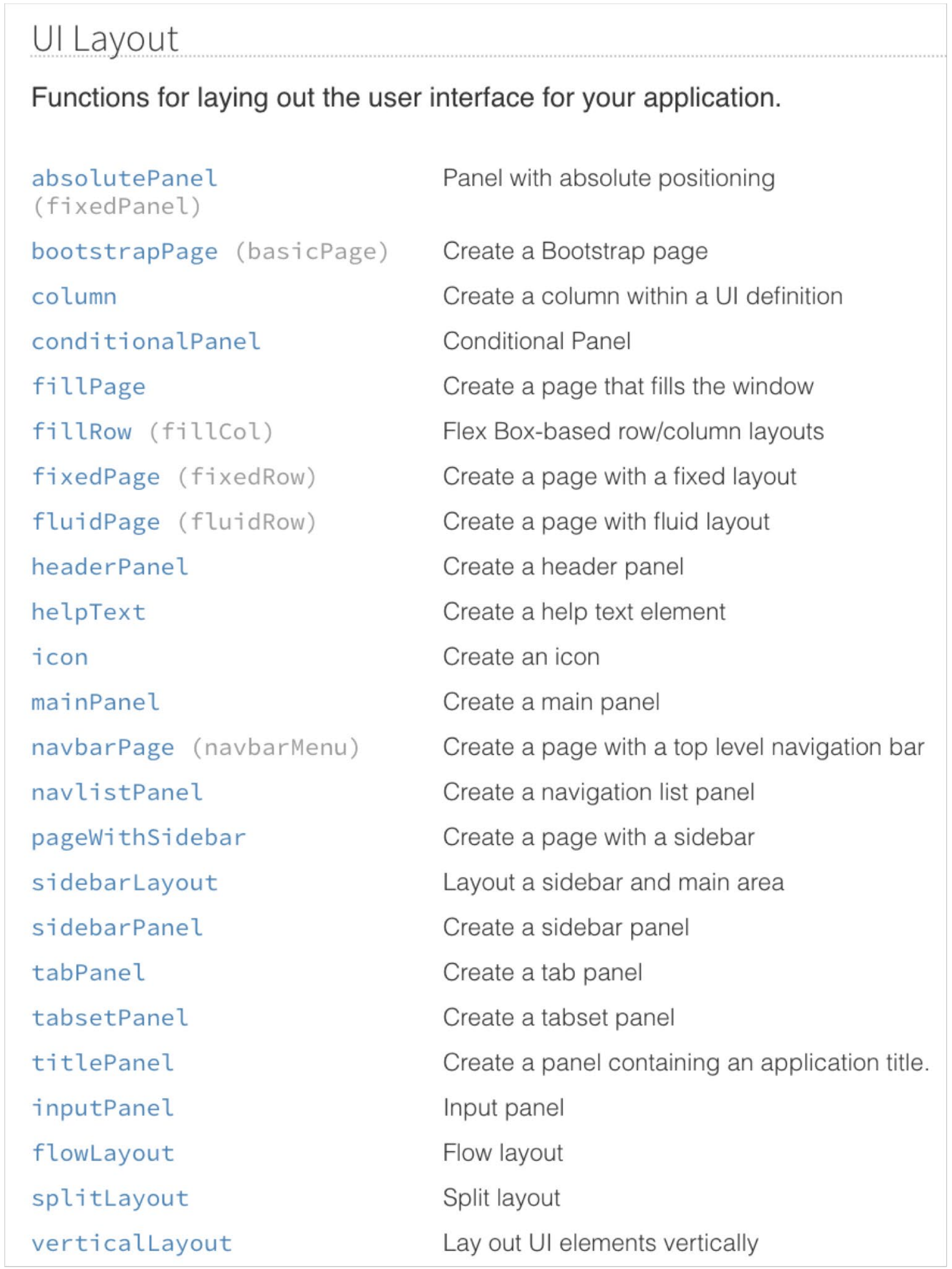
https://shiny.rstudio.com/gallery/movie-explorer.html

# ui.R

## 1. Layout

- **Panels:** group elements together into a single 'panel'.
- **Layouts:** organize panels and elements





UI Layout

Functions for laying out the user interface for your application.

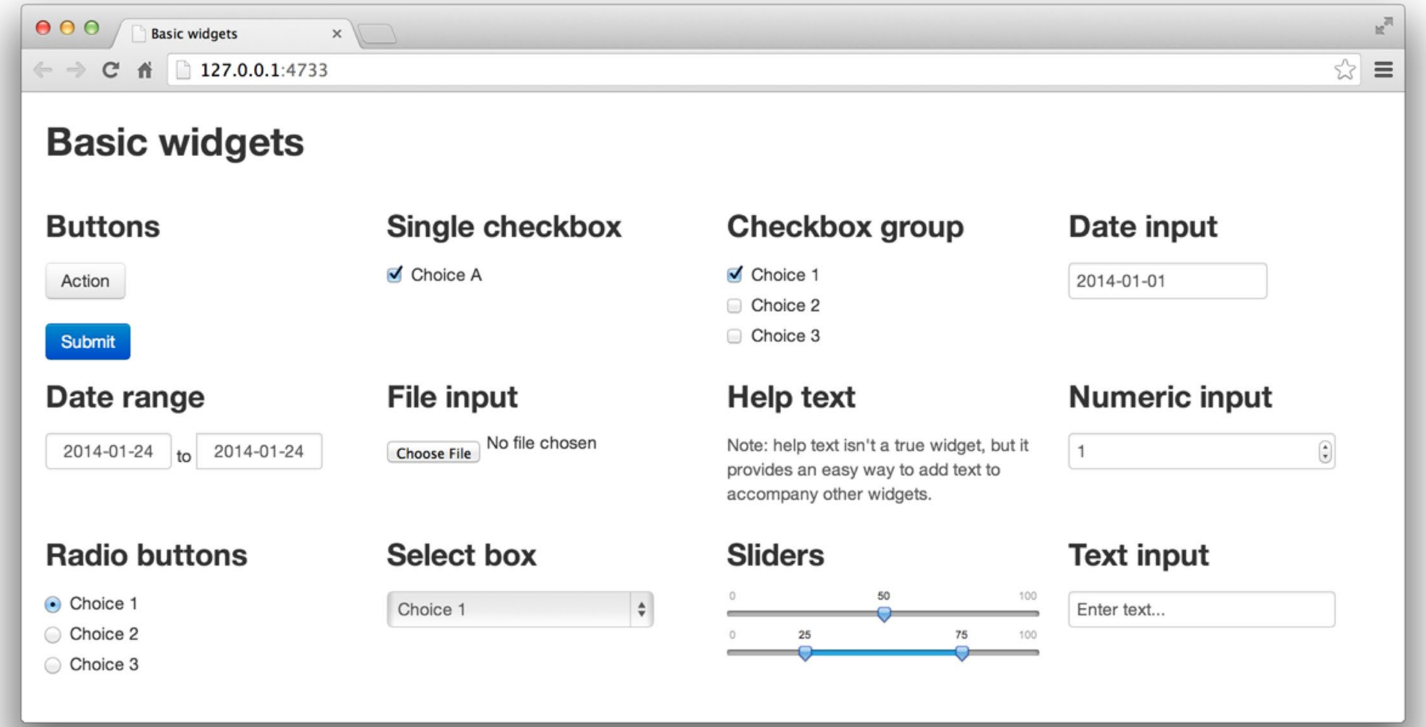
| | |
|---|---|
| `absolutePanel` (`fixedPanel`) | Panel with absolute positioning |
| `bootstrapPage` (`basicPage`) | Create a Bootstrap page |
| `column` | Create a column within a UI definition |
| `conditionalPanel` | Conditional Panel |
| `fillPage` | Create a page that fills the window |
| `fillRow` (`fillCol`) | Flex Box-based row/column layouts |
| `fixedPage` (`fixedRow`) | Create a page with a fixed layout |
| `fluidPage` (`fluidRow`) | Create a page with fluid layout |
| `headerPanel` | Create a header panel |
| `helpText` | Create a help text element |
| `icon` | Create an icon |
| `mainPanel` | Create a main panel |
| `navbarPage` (`navbarMenu`) | Create a page with a top level navigation bar |
| `navlistPanel` | Create a navigation list panel |
| `pageWithSidebar` | Create a page with a sidebar |
| `sidebarLayout` | Layout a sidebar and main area |
| `sidebarPanel` | Create a sidebar panel |
| `tabPanel` | Create a tab panel |
| `tabsetPanel` | Create a tabset panel |
| `titlePanel` | Create a panel containing an application title. |
| `inputPanel` | Input panel |
| `flowLayout` | Flow layout |
| `splitLayout` | Split layout |
| `verticalLayout` | Lay out UI elements vertically |

https://shiny.rstudio.com/reference/shiny/1.0.5/
https://bookdown.org/weicheng/shinyTutorial/ui.html

## ui.R

## 2. Input

- Create with an ***Input()** functions
- Adding widgets/inputs
  - **Name (id)**
  - **Label**
  - Called using ***input$name***



```
sliderInput(inputId = “myid”, label = “this will show in the UI”, …)
```

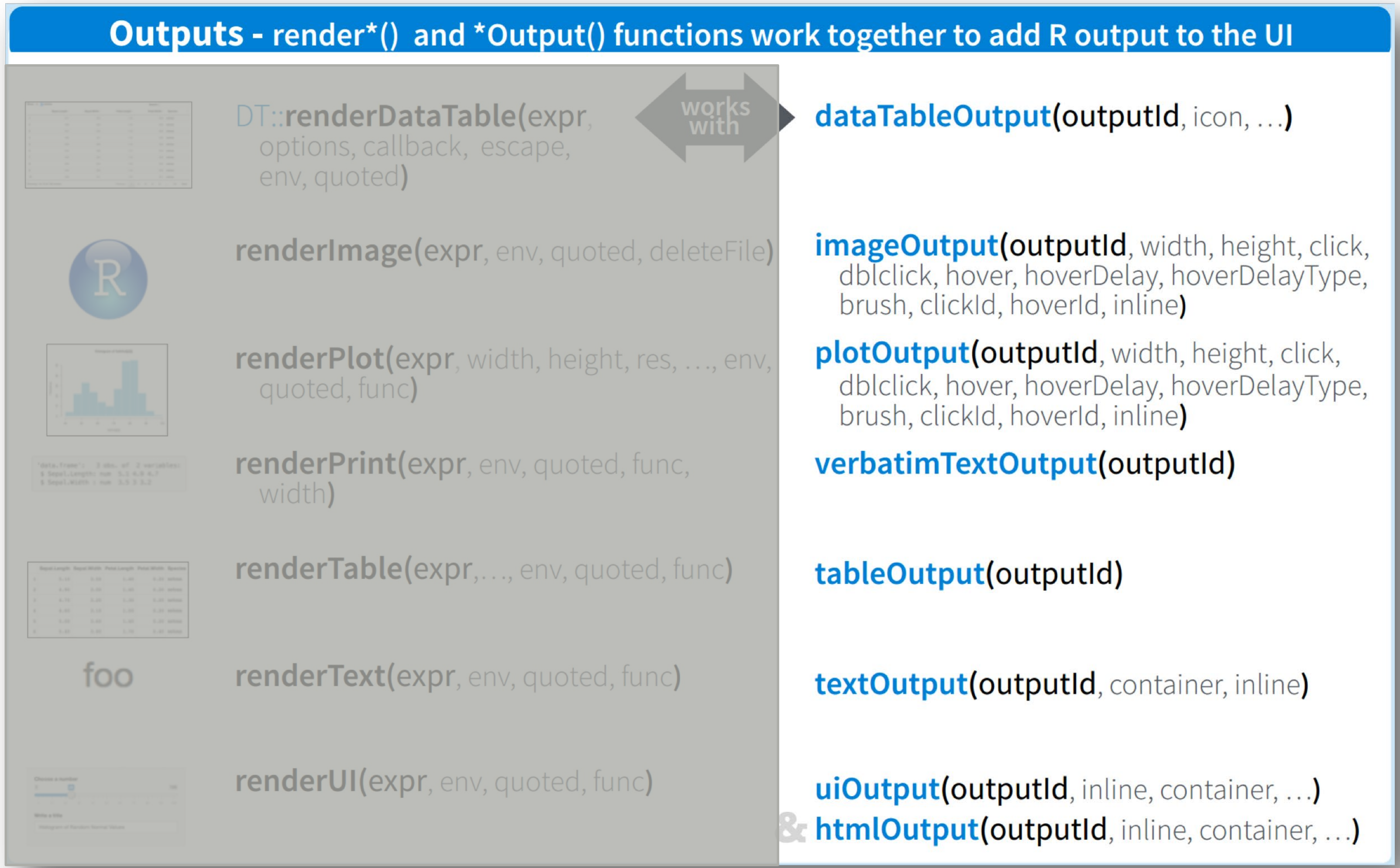
https://shiny.rstudio.com/reference/shiny/1.0.5/
https://bookdown.org/weicheng/shinyTutorial/ui.html

## ui.R

### 3. Output

- Create with ***Output()** functions
- ***Output()** adds a space in the ui.R for an R object

```
plotOutput("histogram")
```



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

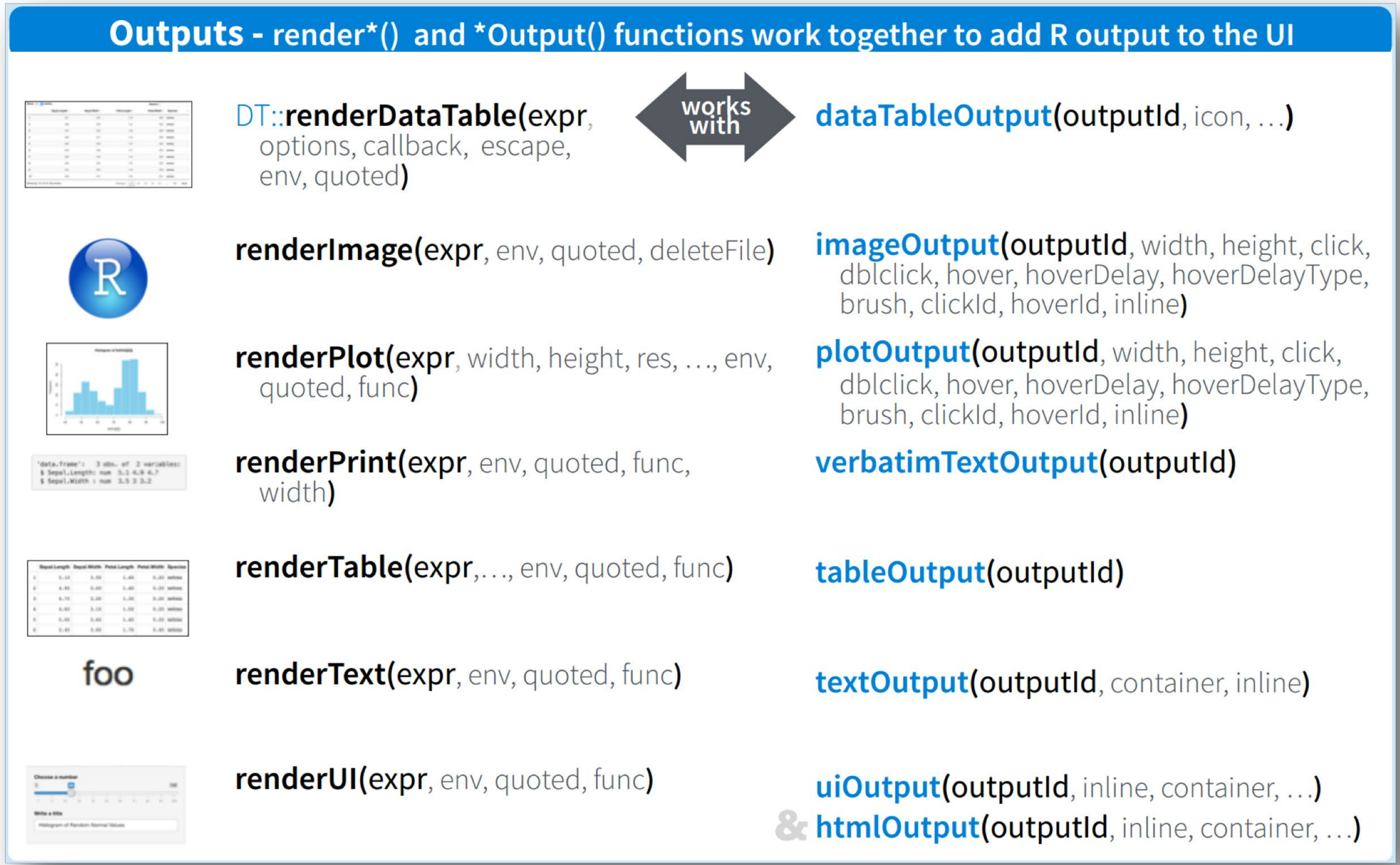
works with

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

# server.R

- Tells the server **how** to render logic using **render*()**
- Works together with ***output()**

```
Output$histogram = 
renderPlot({
    hist(rnorm(100))
})
```



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

works with
foo

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

## Reactivity

**Reactivity** is what enables your outputs to react to changes in inputs. If x is a reactive variable, this *means that when the value of a variable x changes, then anything that relies on x gets re-evaluated*

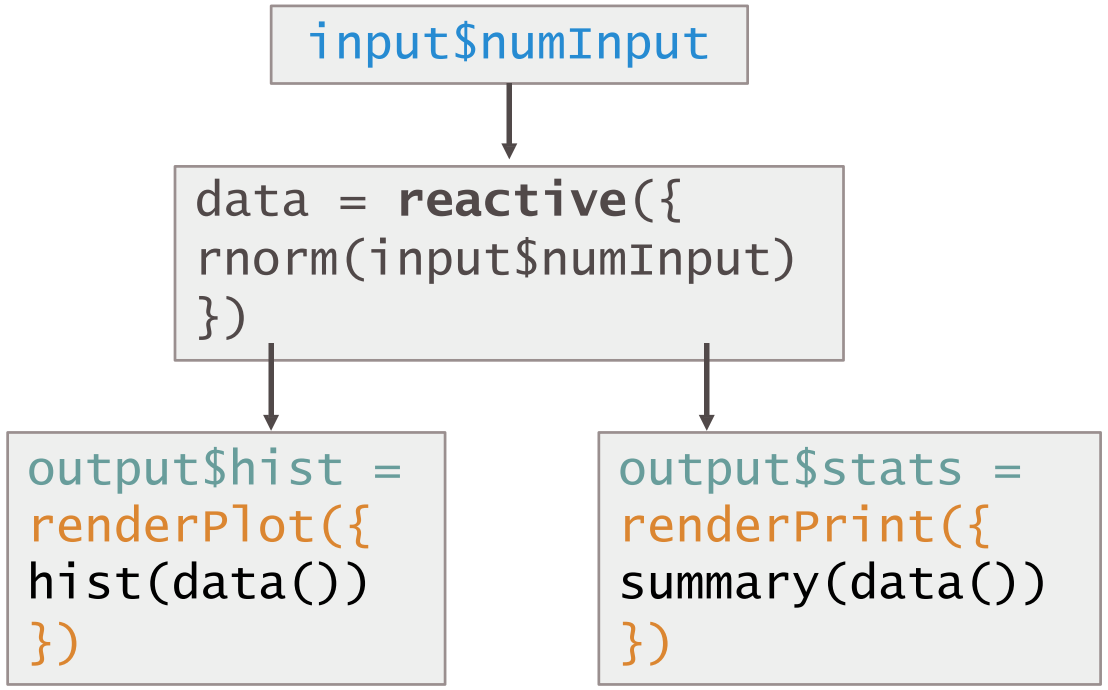
- In Shiny all **inputs** are automatically **reactive**.
- Reactive variables can only be used inside reactive contexts. Any **render*** function is a reactive context
- Besides render*(), there are other 2 common reactive contexts: **reactive({ })** and **observe({ })**

Whenever you want to print a reactive variable for debug you need to remember to wrap it in an observe({ }):
**observe({ print(input$x )})**

http://rstudio.github.io/shiny/tutorial/#reactivity
https://deanattali.com/blog/building-shiny-apps-tutorial/#10-reactivity-101

## Reactivity

- Modularize code with **reactive()**
- **Reactive expression**
  1. Call a reactive expression like a function. ex. data()
  2. Reactive expression cache their values to avoid unnecessary computation



```
input$numInput
```

```
data = reactive({
rnorm(input$numInput)
})
```

```
output$hist =
renderPlot({
hist(data())
})
```

```
output$stats =
renderPrint({
summary(data())
})
```

http://rstudio.github.io/shiny/tutorial/#reactivity
https://deanattali.com/blog/building-shiny-apps-tutorial/#10-reactivity-101

## Summary

User Interface (UI)

Server Instructions

- Create some inputs
- Create a space for output

- Choose a reactive function that will accept our input
- Assign the reactive function to an output that will update the user interface

## Share your shiny app

Two main options:

1) Share the files with users that have R and know how to use it. (e.g. email zip file directly, use GitHub to share files, run from a web-link - runURl("<the weblink>")).

2) Share it as a web page.

- **shinyapps.io** – a server maintained by Rstudio

shinyapps.io by RStudio

http://docs.rstudio.com/shinyapps.io/getting-started.html

- **Shiny Server** – build your own server

Shiny Server

https://shiny.rstudio.com/articles/shiny-server.html

- **Docker container** – deploy and share anywhere

docker

rocker/shiny ☆

## Create a Shiny app

Elkhorn Marine Conservancy
Antigua, W.I.

https://www.emcantigua.org/

https://shiny.posit.co/r/getstarted/shiny-basics/lesson1/

## What is R shiny?

A web application framework for R that allows you to turn your data into an interactive web App.

## How does it work?

Shiny apps have two components:

- user interface script (controls layout and appearance by converting R code into HTML)

- server script (contains the code needed to build the app)

## Do I need to know how to code in HTML?

No knowledge of HTML, JavaScript or CSS is required. However, you can make your apps more interesting if you know some HTML.

## Where can I get more information?

There are some excellent tutorials and lots of links to more information at http://shiny.rstudio.com/

## Getting Started

```
> install.packages("shiny")
> library(shiny)
> runApp("my_app")
```

As a minimum, every shiny app has two R scripts (named ui and server) saved within the same directory (e.g. “my_app”).

**ui.R**

```
shinyUI(fluidPage(
))
```

The fluidPage function allows the display to adjust automatically to the browser dimensions.

**server.R**

```
shinyServer(function(input, output) {
 })
```

The unnamed function contains the code needed to run the app.

Other scripts (written in standard R code), libraries and data can be called by the server script if they are needed, using standard R commands above the shinyServer function.

## Getting Started

```
> install.packages("shiny")
> library(shiny)
> runApp("my_app")
```

As a minimum, every shiny app has two R scripts (named ui and server) saved within the same directory (e.g. “my_app”).

**ui.R**

```
shinyUI(fluidPage(
))
```

The fluidPage function allows the display to adjust automatically to the browser dimensions.

**server.R**

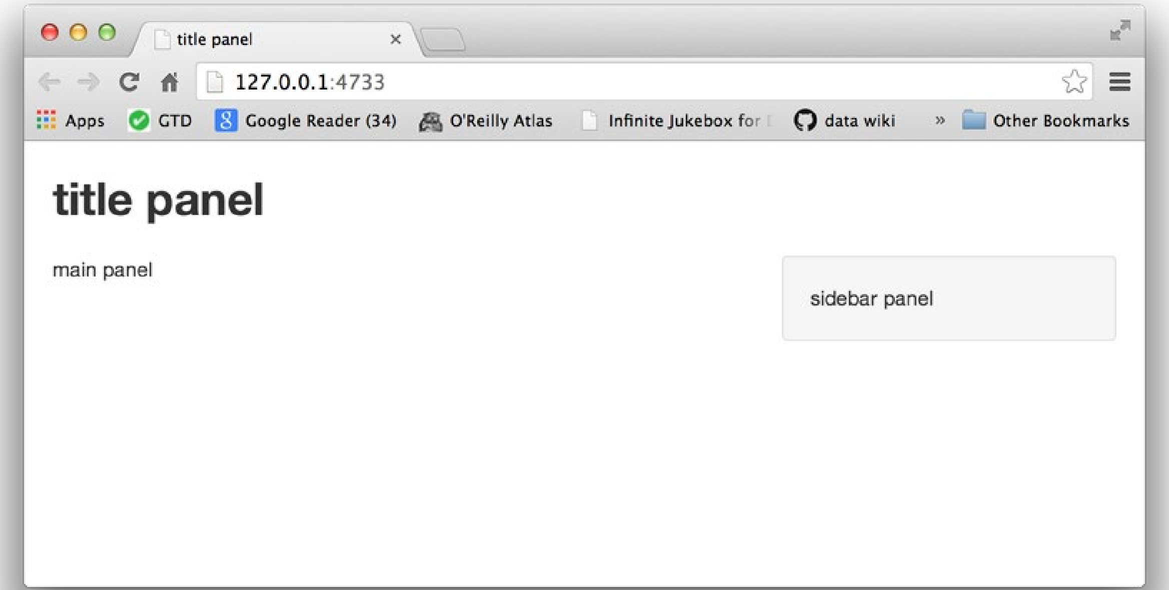
```
shinyServer(function(input, output) {
 })
```

The unnamed function contains the code needed to run the app.

Other scripts (written in standard R code), libraries and data can be called by the server script if they are needed, using standard R commands above the shinyServer function.

# The User Interface (ui.R)

```
# ui.R

shinyUI(fluidPage(
        titlePanel("title panel"),

        sidebarLayout(position = "right",
                sidebarPanel( "sidebar panel"),
                mainPanel("main panel")
        )
))
```



title panel

main panel

sidebar panel

A series of nested shiny functions control the layout of the content.

sidebarLayout is a function with two compulsory arguments, both themselves functions (sidebarPanel and mainPanel). An optional argument controls the position of the panels.

Text is written as a character string inside quotation marks.

# Shiny User Interface Functions

| | |
|---|---|
| absolutePanel **(fixedPanel)** | Panel with absolute positioning |
| bootstrapPage **(basicPage)** | Create a Bootstrap page |
| column | Create a column within a UI definition |
| conditionalPanel | Conditional Panel |
| fixedPage **(fixedRow)** | Create a page with a fixed layout |
| fluidPage **(fluidRow)** | Create a page with fluid layout |
| headerPanel | Create a header panel |
| helpText | Create a help text element |
| icon | Create an icon |
| mainPanel | Create a main panel |
| navbarPage **(navbarMenu)** | Create a page with a top level navigation bar |
| navlistPanel | Create a navigation list panel |
| pageWithSidebar | Create a page with a sidebar |
| sidebarLayout | Layout a sidebar and main area |
| sidebarPanel | Create a sidebar panel |
| tabPanel | Create a tab panel |
| tabsetPanel | Create a tabset panel |
| titlePanel | Create a panel containing an application title. |
| inputPanel | Input panel |
| flowLayout | Flow layout |
| splitLayout | Split layout |
| verticalLayout | Lay out UI elements vertically |
| wellPanel | Create a well panel |
| withMathJax | Load the MathJax library and typeset math expressions |

## Formatting Text

To make things look a bit more interesting there are lots of shiny commands that can alter the text colour, style and size. Alternatively, if you are familiar with HTML, you can write HTML code directly inside HTML("").

| shiny function | HTML5 equivalent | Creates |
|---|---|---|
| p | `<p>` | A paragraph of text |
| h1 | `<h1>` | A first level header |
| h2 | `<h2>` | A second level header |
| h3 | `<h3>` | A third level header |
| h4 | `<h4>` | A fourth level header |
| h5 | `<h5>` | A fifth level header |
| h6 | `<h6>` | A sixth level header |
| a | `<a>` | A hyper link |
| br | `<br>` | A line break (e.g. a blank line) |
| div | `<div>` | A division of text with a uniform style |
| span | `<span>` | An in-line division of text with a uniform style |
| pre | `<pre>` | Text 'as is' in a fixed width font |
| code | `<code>` | A formatted block of code |
| img | `<img>` | An image |
| strong | `<strong>` | Bold text |
| em | `<em>` | Italicized text |
| HTML | | Directly passes a character string as HTML code |

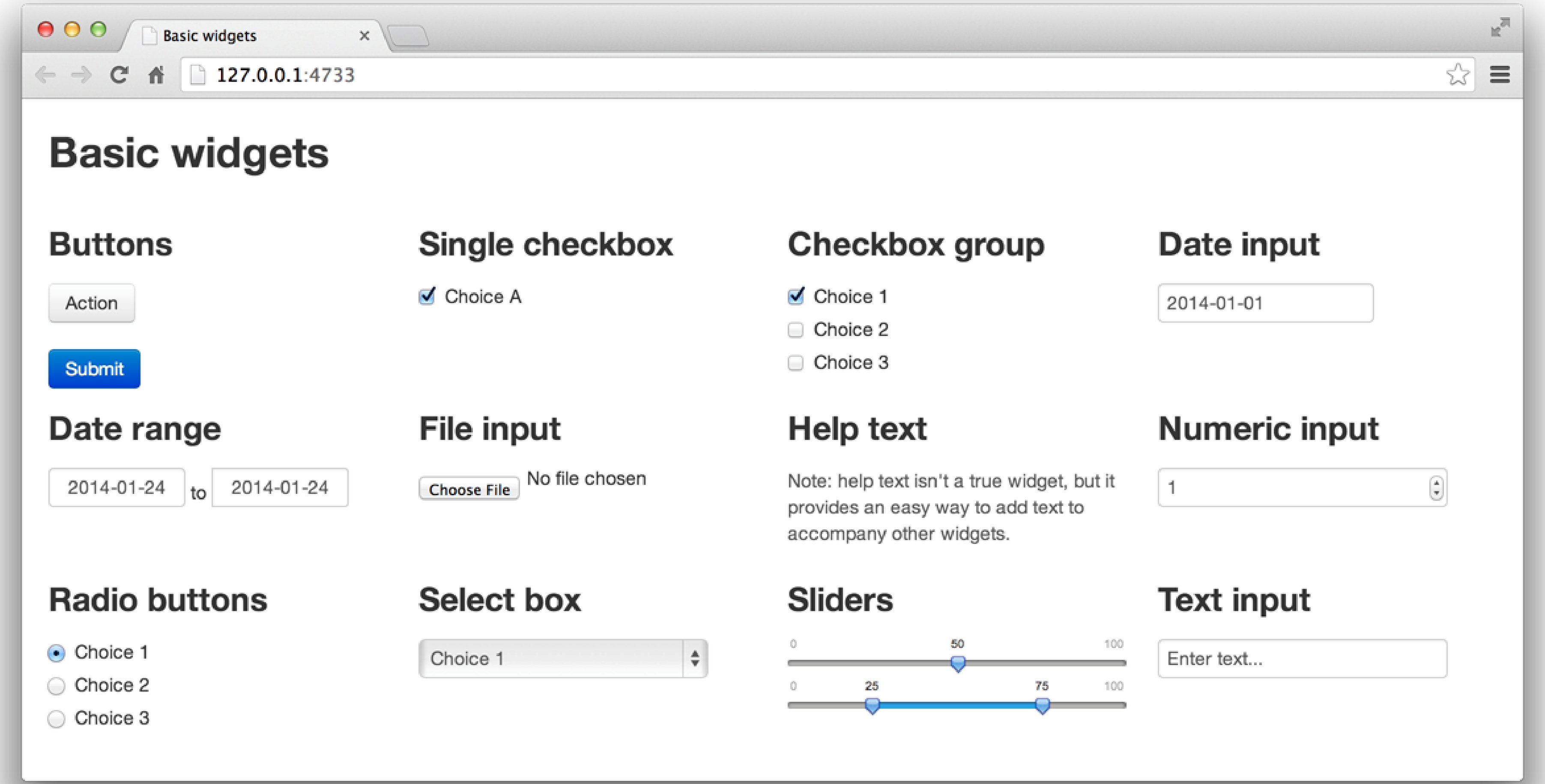
## Adding Images

```
# ui.R

shinyUI(fluidPage(
            titlePanel("My Shiny App"),

            sidebarLayout(
                        sidebarPanel(),
                        mainPanel(
                        img(src = “my_image.png", height = 400, width = 400)
                        )
            )
))
```

The image file must be placed inside a folder named www within the same directory as the ui and server scripts. Any file placed in the www folder will be shared with the web browser.

# Adding Widgets

Widgets are interactive web elements. A series of inbuild shiny functions allows widgets to be easily added to webpages. Each function has several arguments. It must have at least a name (used to access its value, but not visible) and a label (visible on the page). Both are character strings. Other arguments are widget specific.



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

## Adding Widgets

```
# ui.R

shinyUI(fluidPage(
        titlePanel("Basic widgets"),

        fluidRow(
                column(3,
                        h3("Buttons"),
                        actionButton("action", label = "Action"),
                        br(),
                        br(),
                        submitButton("Submit")),

                column(3,
                        h3("Single checkbox"),
                        checkboxInput("checkbox", label = "Choice A", value = TRUE)),

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

                column(3,
                        dateInput("date",
                        label = h3("Date input"),
                        value = "2014-01-01"))
        )
))
```

Action button

Submit button

Single checkbox (whether pre-selected)

Checkbox group (list choices and which pre-selected)

Date input

# Adding Reactive Output

Widgets allow users to input their choices. Reactive output is a response to those choices.
Programming it into an app is a two-step process:

- Add an R object to the user interface to indicate where it should be displayed.
- Add instructions on how to build the object in the server

```
# ui.R

shinyUI(fluidPage(
  titlePanel("My Shiny App"),

    sidebarLayout(
      sidebarPanel(
        sliderInput(“value",
        label = “Value of interest:",
        min = 0, max = 100, value = 50)
      ),

    mainPanel(
      textOutput("text1")
    )
  )
))
```

| **Output function** | **creates** |
|---|---|
| htmlOutput | raw HTML |
| imageOutput | image |
| plotOutput | plot |
| tableOutput | table |
| textOutput | text |

Output functions have one argument, which is a name (character string). In this case “text1”. This is used in the server to identify the output.

## Adding Reactive Output

The unnamed function inside shinyServer contains all the code that needs to be updated when a user accesses the app. All R output objects used in the ui need to be defined in server using the prefix output$ followed by the name of the object.
e.g. output$text1

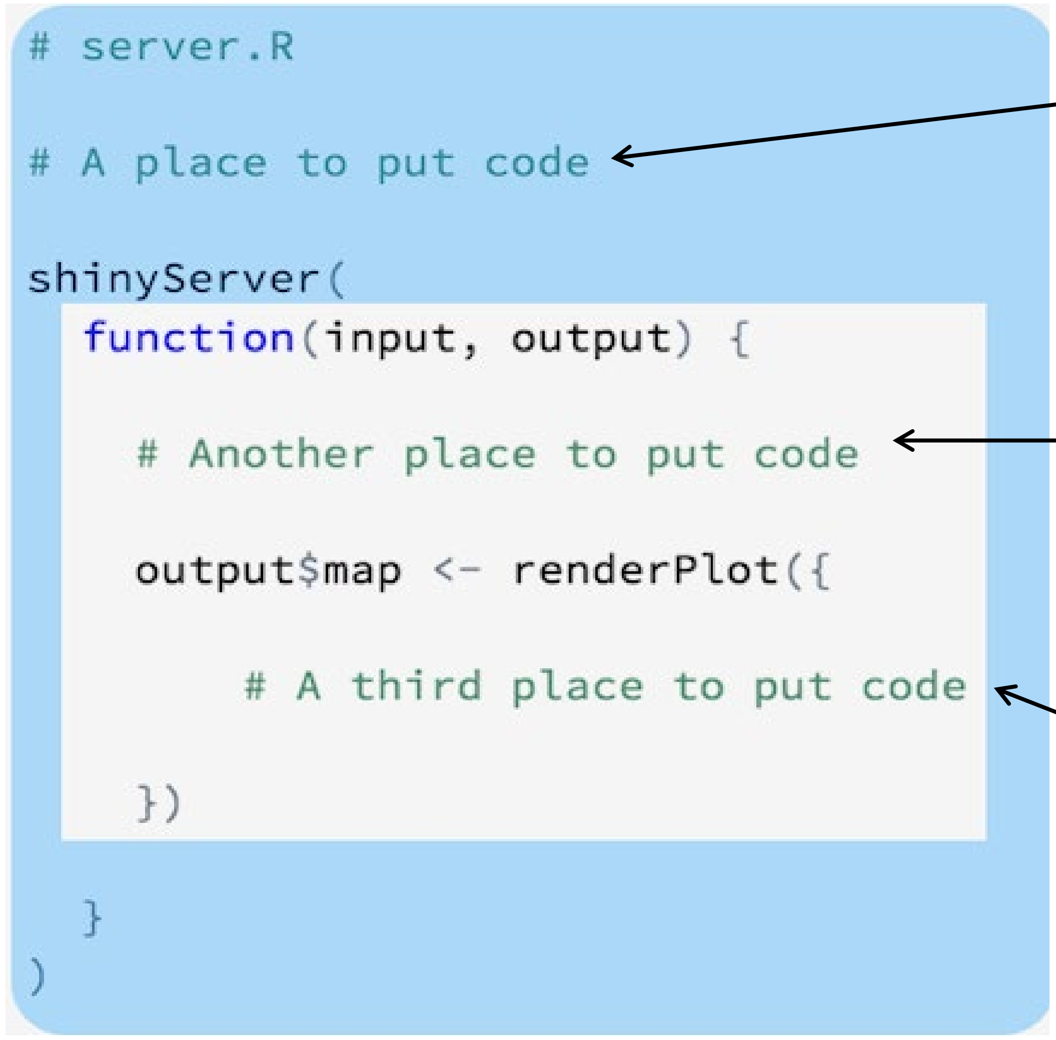
The element should be defined using one of the shiny render* functions – this should be chosen to reflect the type of output. Each render function takes a single argument (R expression) surrounded by braces.

```
# server.R

shinyServer(function(input, output) {

      output$text1 <- renderText({
         "You have selected" , input$value)
      })
   }
)
```

| **render function** | **creates** |
|---|---|
| renderImage | images (saved as a link to a source file) |
| renderPlot | plots |
| renderPrint | any printed output |
| renderTable | data frame, matrix, other table like structures |
| renderText | character strings |
| renderUI | a Shiny tag object or HTML |

## The Server

The server is where all the code is located for execution of the app. How frequently code is run depends upon where it is placed within the server script.



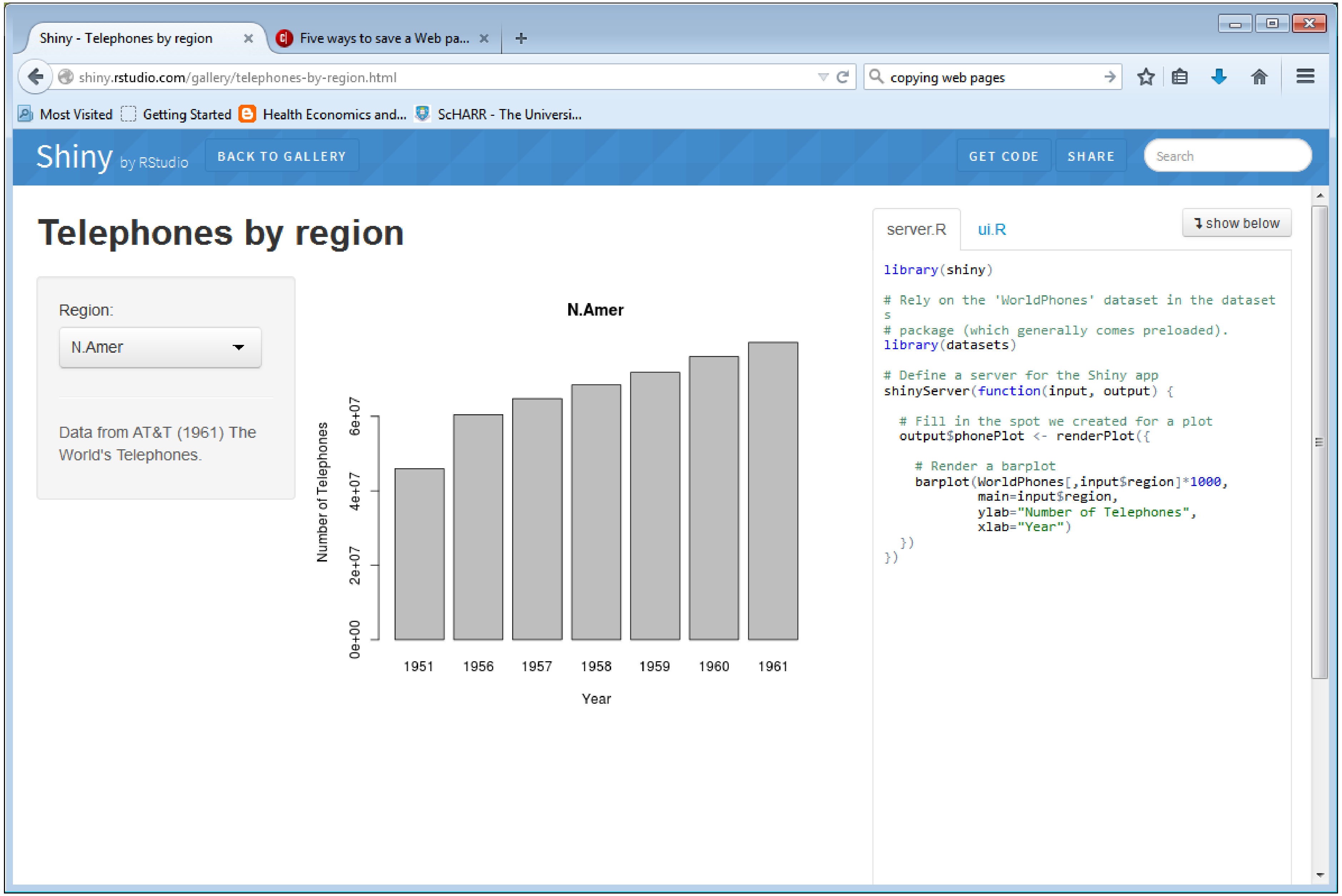
```
# server.R

# A place to put code

shinyServer(
  function(input, output) {

    # Another place to put code

    output$map <- renderPlot({

        # A third place to put code

    })

  }
)
```

Code placed here will be run once when the app is launched. shinyServer will be executed and the unnamed function saved. Good place to source other scripts, read data sets and load libraries.

Code placed here will run once each time a new user visits the app. The unnamed function will run once and save a distinct set of reactive objects for each user. Good place to put objects that record a user's session.

Code placed here will run every time the user changes the widget that this particular output depends upon. Important to not put unnecessary code here as it will slow down the entire app.

# Example of a shiny app with code



Telephones by region

Data from AT&T (1961) The World's Telephones.

```
library(shiny)

# Rely on the 'WorldPhones' dataset in the dataset
s
# package (which generally comes preloaded).
library(datasets)

# Define a server for the Shiny app
shinyServer(function(input, output) {

  # Fill in the spot we created for a plot
  output$phonePlot <- renderPlot({

    # Render a barplot
    barplot(WorldPhones[,input$region]*1000,
            main=input$region,
            ylab="Number of Telephones",
            xlab="Year")
  })
})
```

Thank you for your attention!!

Any questions?

Crédits: Laurent Bouveret