Introduction to Shiny

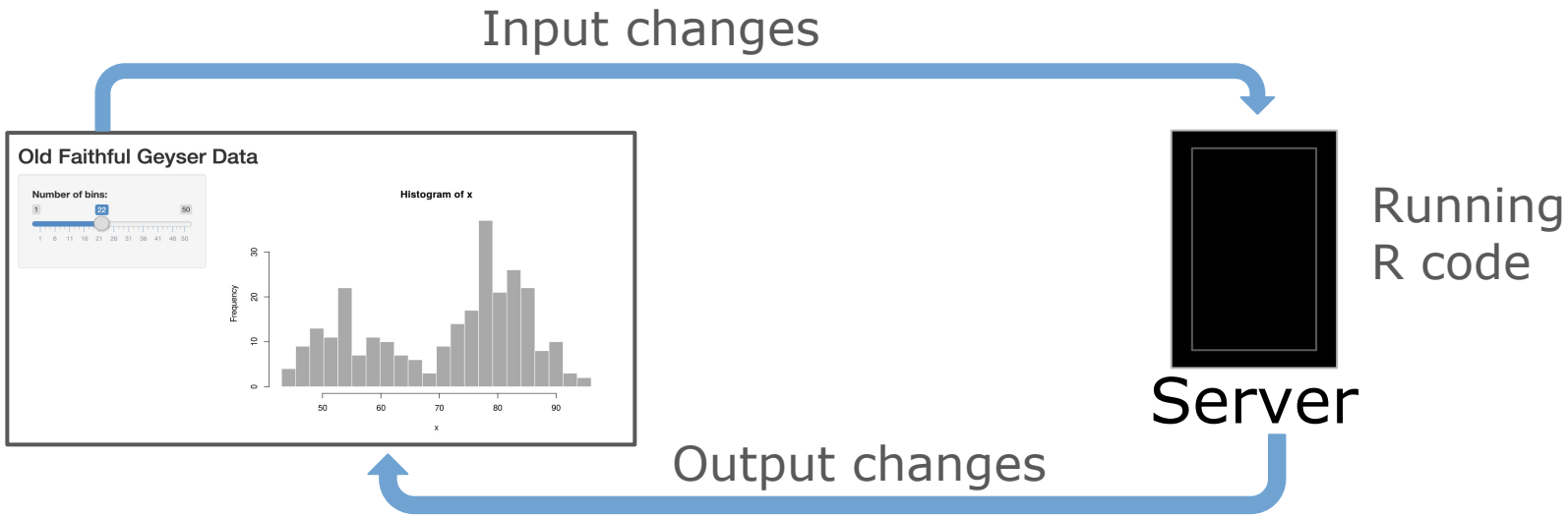
What we'll cover today

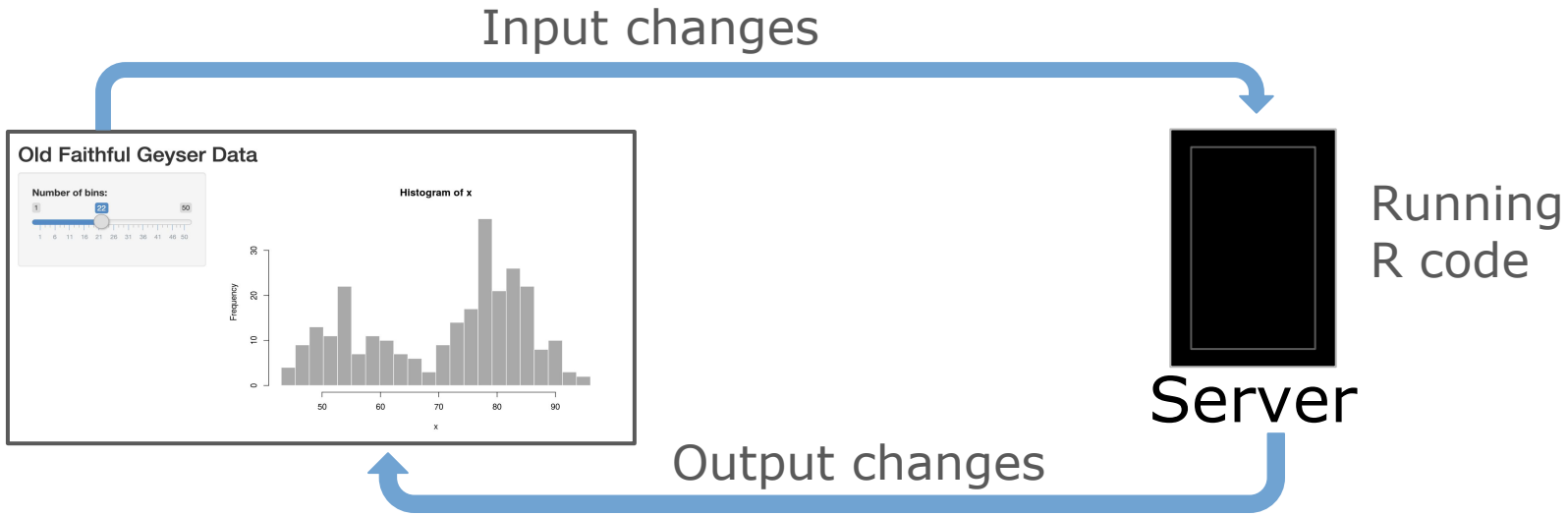
- Part 1. Getting Started
  - What is it, Basic components, Reactivity
- Part 2. To infinity and beyond
  - aesthetics, layouts, sharing your application

Getting Started

What is Shiny?

- Shiny is a framework for creating web applications in R
- No html, css, or javascript knowledge required
- A quick way to build cool things with big impact


Input changes
Old Faithful Geyser Data
Number of bins:
1
22
50
1 6 11 16 21 26 31 36 41 46 50
Histogram of x
Frequency
0
10
20
30
50
60
70
80
90
x
Server
Running
R code
Output changes



UI

What do I see / touch?

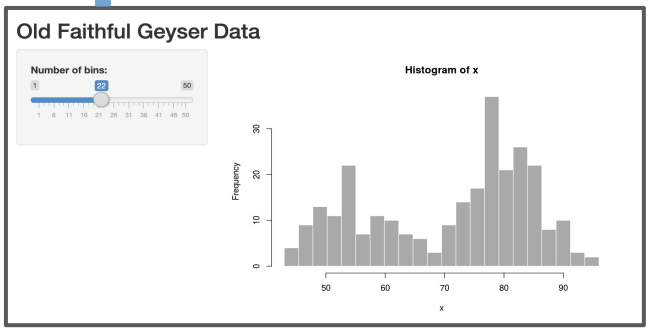
Server

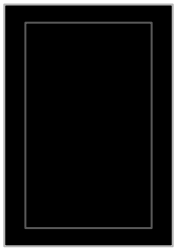
What does the app DO?

Shiny Template

```
library(shiny)

ui <- fluidPage(

)

server <- function(input, output) {

}

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

Input changes





Server

Running
R code

Output changes

UI

- Title
- Slider
- graph

Server

- Receive the slider input
- Create a ggplot

The Shiny UI

UI

- Title
- Slider
- graph

```
ui <- fluidPage(
    titlePanel("Old Faithful Geyser Data"),
    sidebarLayout(
        sidebarPanel(
            sliderInput("bins",
                        "Number of bins:",
                        min = 1,
                        max = 50,
                        value = 30)
        ),
        mainPanel(
           plotOutput("distPlot")
        )
    )
)
```

The Shiny UI

UI

- **Title**
- Slider
- graph

```
ui <- fluidPage(
    titlePanel("Old Faithful Geyser Data"),
    sidebarLayout(
        sidebarPanel(
            sliderInput("bins",
                        "Number of bins:",
                        min = 1,
                        max = 50,
                        value = 30)
        ),
        mainPanel(
           plotOutput("distPlot")
        )
    )
)
```

The Shiny UI

UI

- Title
- **Slider**
- graph

```
ui <- fluidPage(
    titlePanel("Old Faithful Geyser Data"),
    sidebarLayout(
        sidebarPanel(
            sliderInput("bins",
                        "Number of bins:",
                        min = 1,
                        max = 50,
                        value = 30)
        ),
        mainPanel(
           plotOutput("distPlot")
        )
    )
)
```

The Shiny UI

UI

- Title
- Slider
- **graph**

```
ui <- fluidPage(
    titlePanel("Old Faithful Geyser Data"),
    sidebarLayout(
        sidebarPanel(
            sliderInput("bins",
                        "Number of bins:",
                        min = 1,
                        max = 50,
                        value = 30)
        ),
        mainPanel(
           plotOutput("distPlot")
        )
    )
)
```

The Shiny UI

UI
- Title
- Slider
- Graph
- **Layout**

```
ui <- fluidPage(
    titlePanel("Old Faithful Geyser Data"),
    sidebarLayout(
        sidebarPanel(
            sliderInput("bins",
                        "Number of bins:",
                        min = 1,
                        max = 50,
                        value = 30)
        ),
        mainPanel(
           plotOutput("distPlot")
        )
    )
)
```

The Shiny UI: Input Controls

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

The Shiny UI: Input Controls

```
sliderInput(<inputID>,"Number of bins:",
                        min =  1,
                        max =  50,
                        value =  30)
```

The Shiny UI: Input Controls

```
sliderInput(<inputID>,<Readable Text>,
                          min =  1,
                          max =  50,
                          value =  30)
```

The Shiny UI: Input Controls

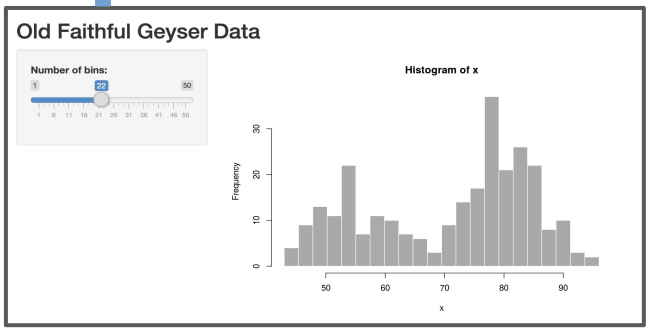
```
sliderInput(<inputID>,<Readable Text>,
                          min =  1,
                          max =  50,
                         <value>)
```

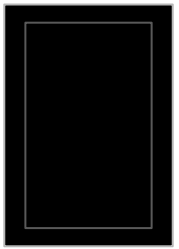
The Shiny UI: Input Controls

```
*Input(<inputID>,<Readable Text>,
                              <value>, <other>)
```

There are a variety of input control options, that may have additional argument options. Check out the options in the shiny widgets gallery.

Input changes





Server

Running
R code

Output changes

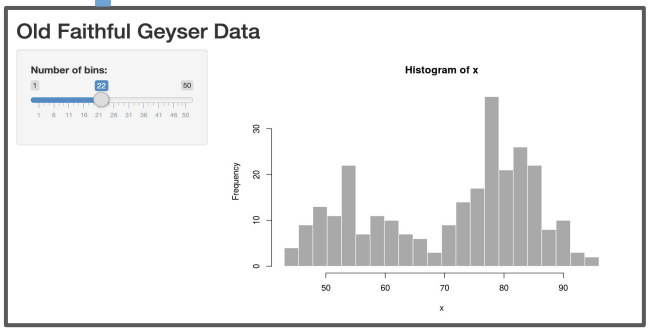
UI

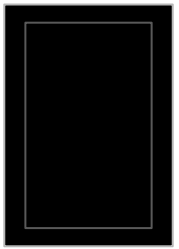
```
sliderInput("bins", …)
```

input$bins

Server

Input changes





Running R code

Server

Output changes

UI

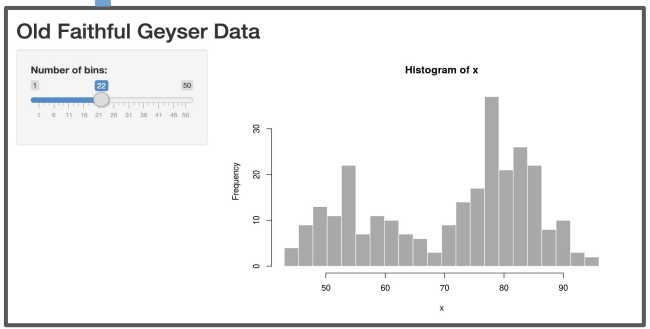
```
sliderInput("bins", …)
```

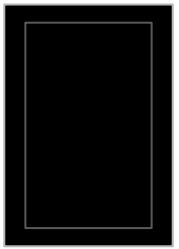
```
plotOutput("distPlot")
```

```
input$bins
```

Server

Input changes





Server

Running
R code

Output changes

UI

```
sliderInput("bins", …)
```

```
plotOutput("distPlot")
```

```
input$bins
```

Server

?

The Shiny UI: Output controls

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

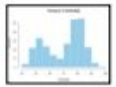
DT::**renderDataTable**(expr, options, callback, escape, env, quoted)

works with

dataTableOutput(outputId, icon, …)

renderImage(expr, env, quoted, deleteFile)

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

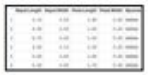


renderPlot(expr, width, height, res, …, env, quoted, func)

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)

foo

renderText(expr, env, quoted, func)

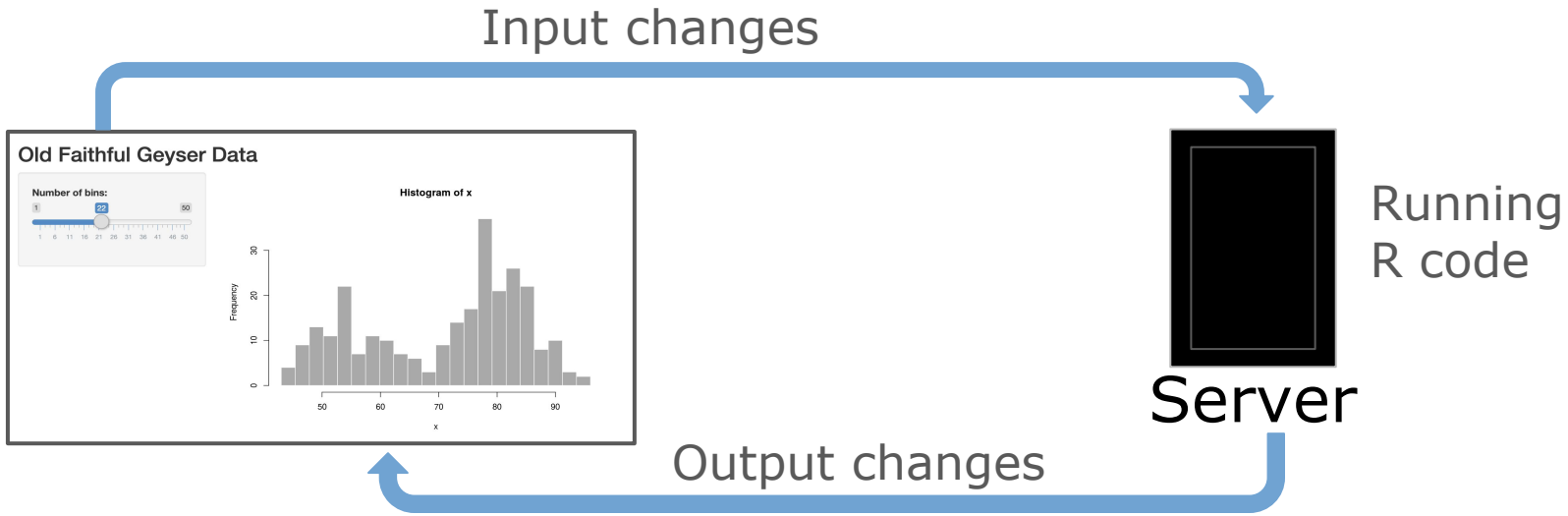
textOutput(outputId, container, inline)

renderUI(expr, env, quoted, func)

uiOutput(outputId, inline, container, …)

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

Output controls have a render*() pair



UI

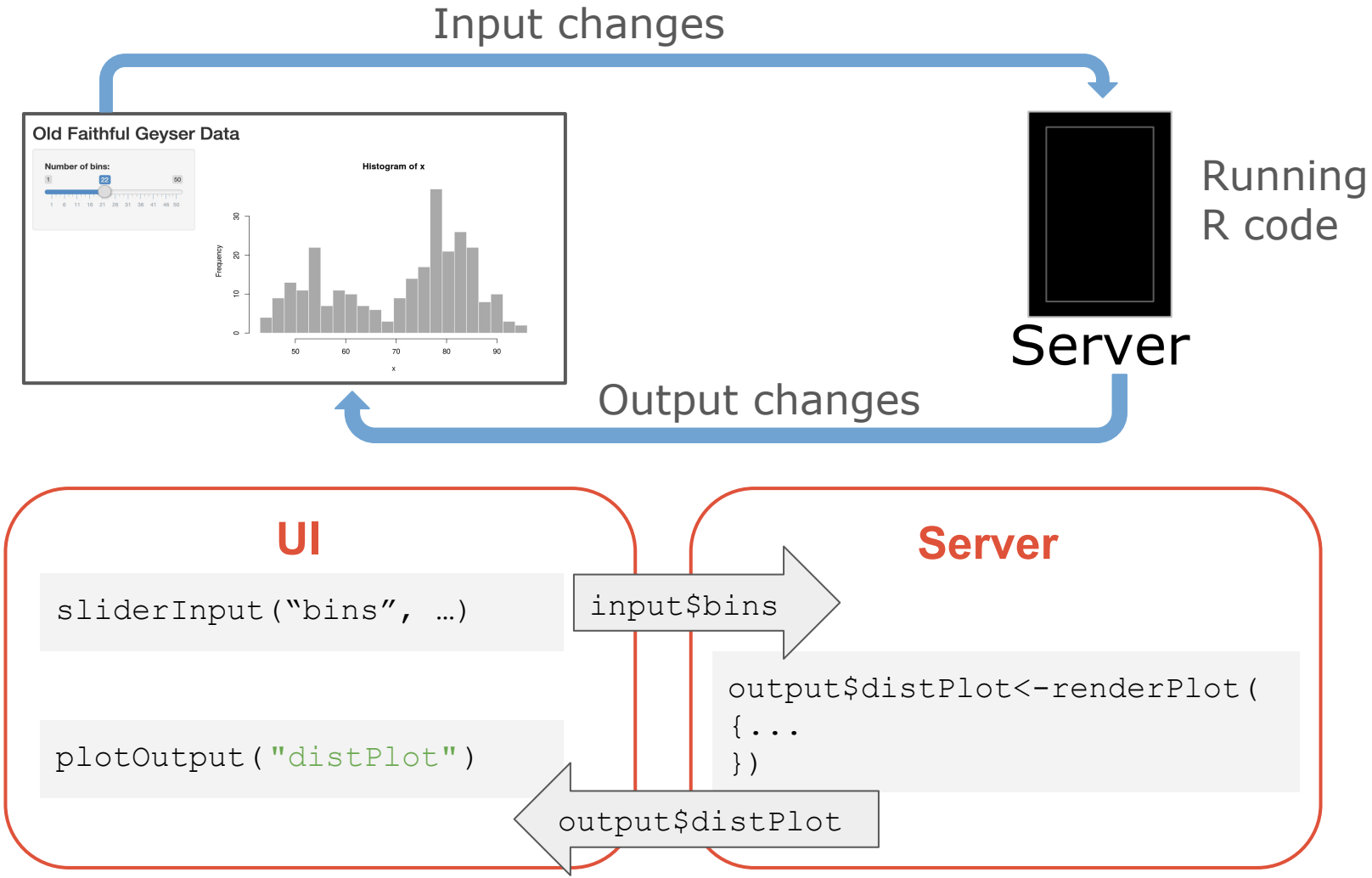
```
sliderInput("bins", …)
```

```
plotOutput("distPlot")
```

```
input$bins
```

Server

```
renderPlot({...})
```


Input changes
Old Faithful Geyser Data
Number of bins:
Histogram of x
Frequency
x
Server
Running
R code
Output changes
UI
sliderInput("bins", …)
plotOutput("distPlot")
input$bins
output$distPlot
Server
output$distPlot<-renderPlot(
{...
})

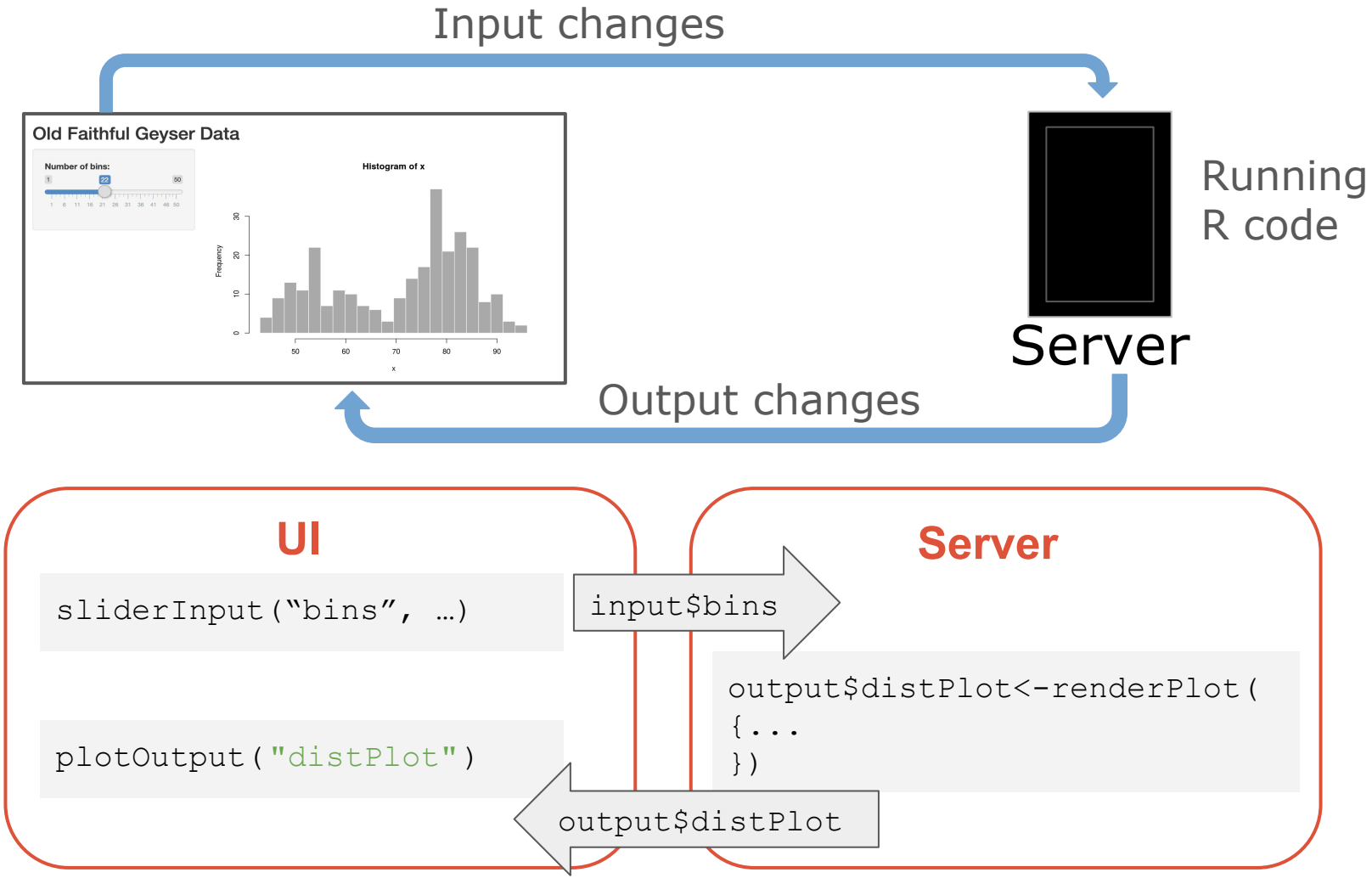
3 Rules for the Server Function

1. Save objects you want to display as **output$**
2. Build object with a **render*()**

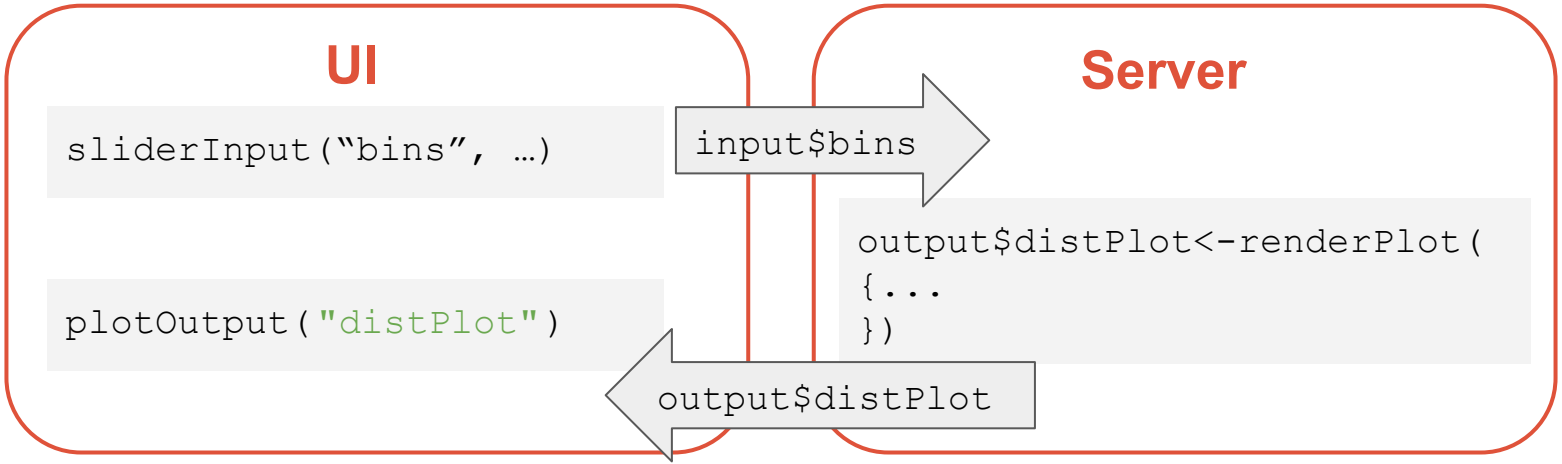
```
output$distPlot<-renderPlot({...})
```

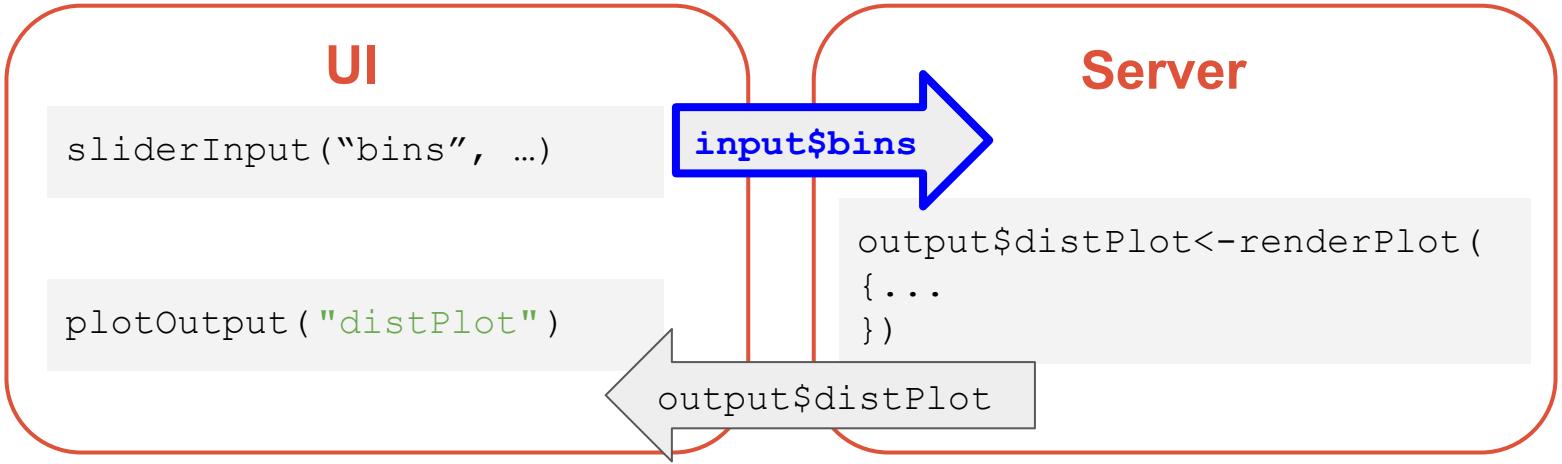
3. Access input values with **input$**

```
output$distPlot<-renderPlot({...
        bins <- seq(min(x), max(x), length.out = input$bins + 1)
....})
```


Input changes
Old Faithful Geyser Data
Number of bins:
Histogram of x
Frequency
x
Server
Running
R code
Output changes
UI
sliderInput("bins", …)
plotOutput("distPlot")
input$bins
output$distPlot
Server
output$distPlot<-renderPlot(
{...
})

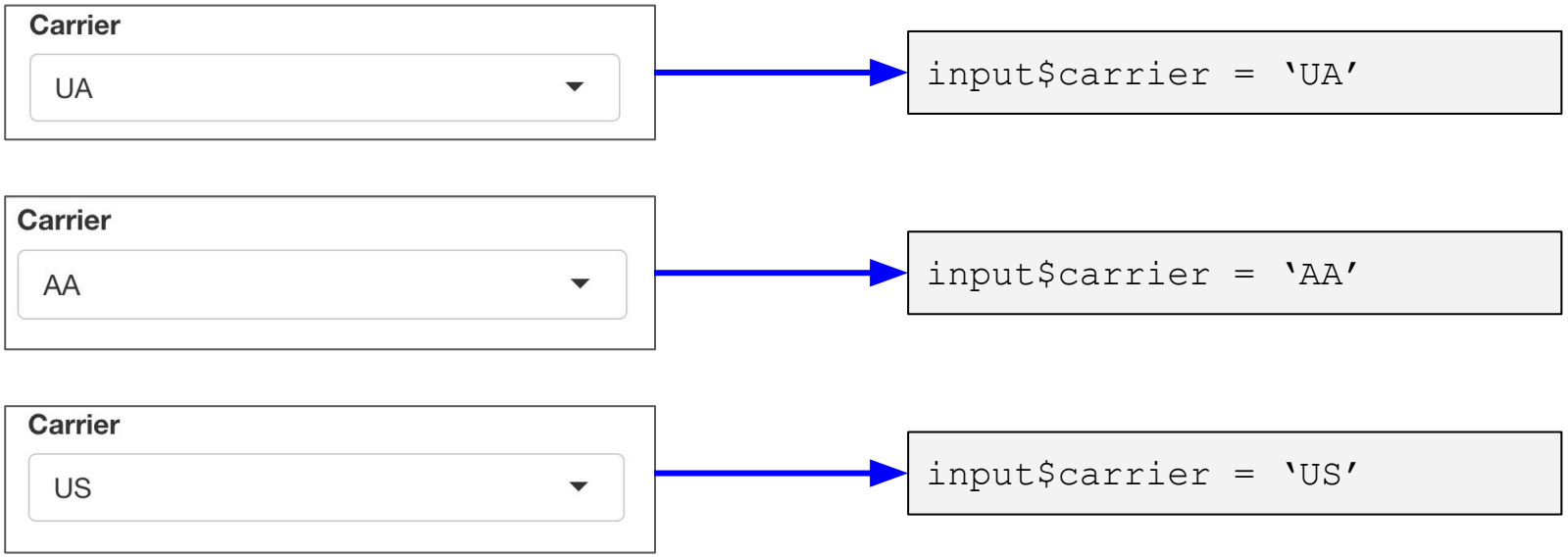
A little about Reactivity


UI
sliderInput("bins", …)
plotOutput("distPlot")
input$bins
output$distPlot
Server
output$distPlot<-renderPlot(
{...
})


UI
sliderInput("bins", …)
plotOutput("distPlot")
input$bins
Server
output$distPlot<-renderPlot(
{...
})
output$distPlot

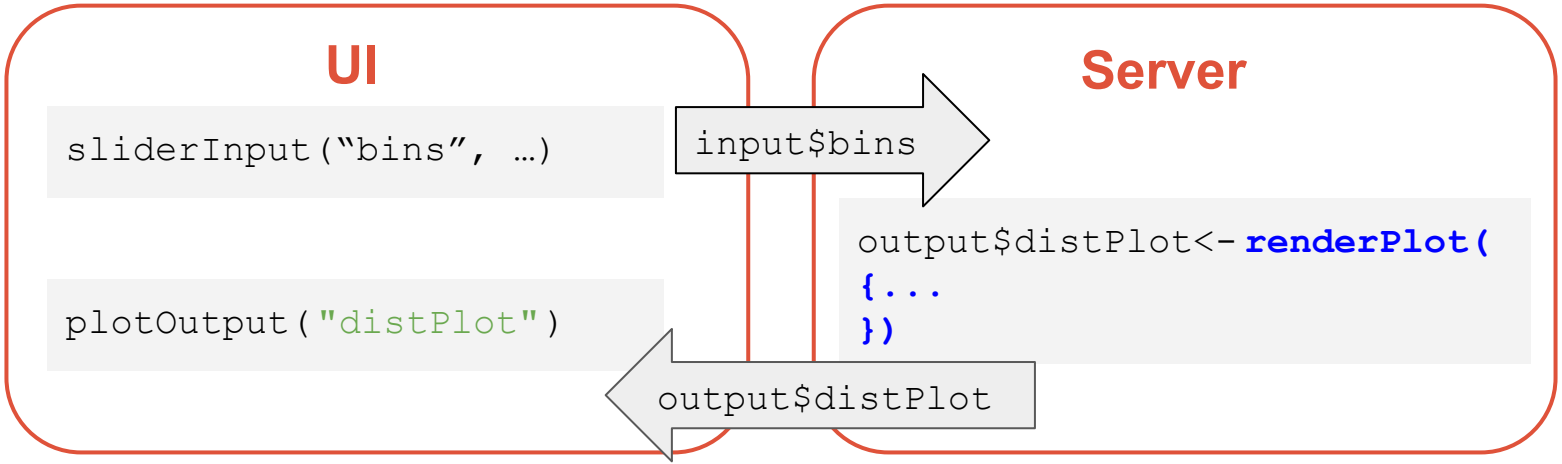
Reactive Values in Shiny

- Change whenever user changes input



Reactive Values in Shiny

- Must be used inside reactive functions

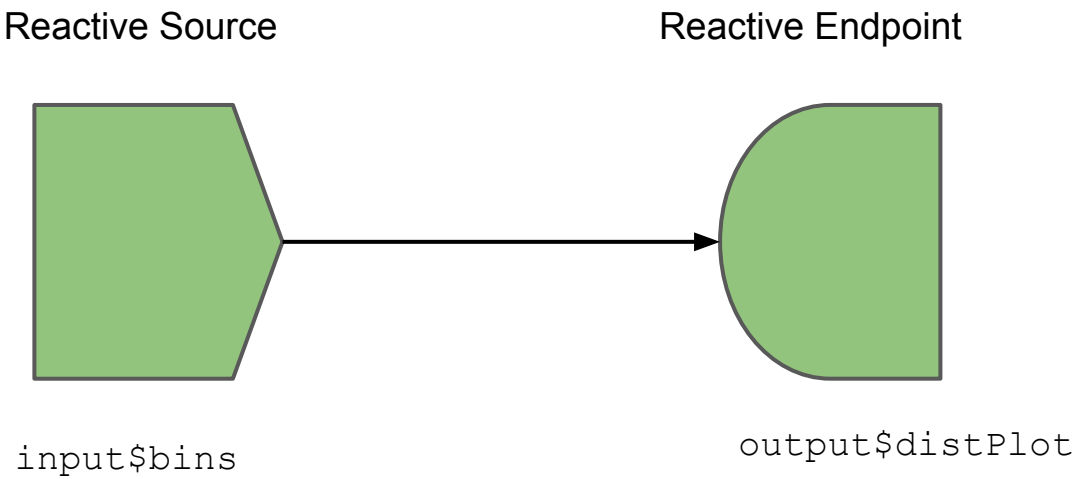


Reactive Values in Shiny

- Must be used inside reactive functions

```
output$distPlot <- renderPlot({
        x    <- faithful[, 2]
        bins <- seq(min(x), max(x), length.out = input$bins + 1)
        hist(x, breaks = bins, col = 'darkgray', border = 'white')
    })
```

A little about Reactivity



Reactive Functions in Shiny

- **render*()**

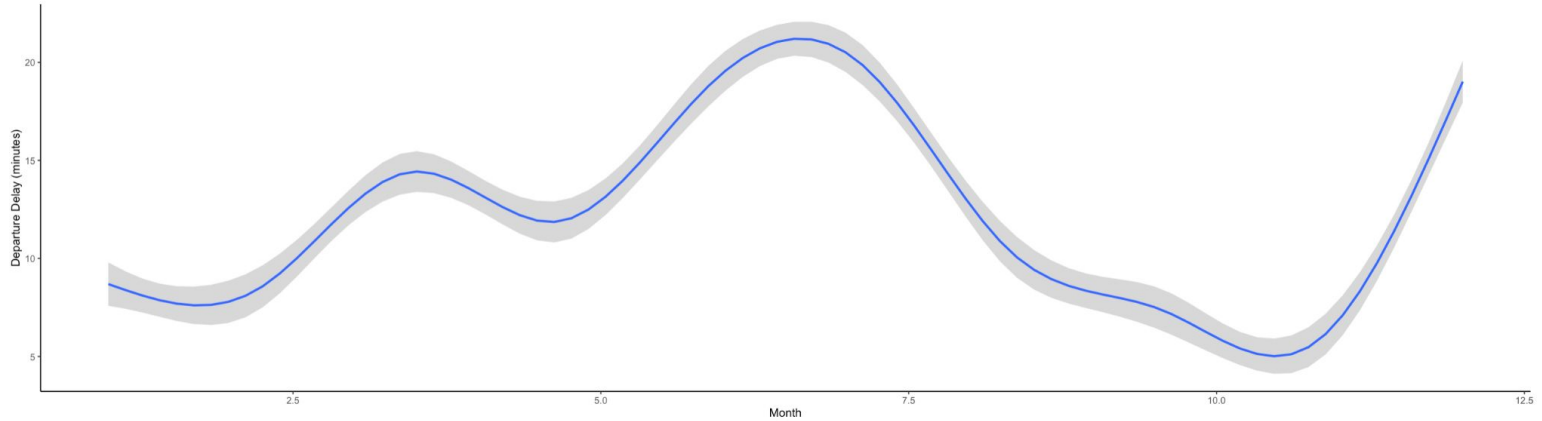
```
output$distPlot <- renderPlot({
        x    <- faithful[, 2]
        bins <- seq(min(x), max(x), length.out = input$bins + 1)
        hist(x, breaks = bins, col = 'darkgray', border = 'white')
    })
```

Reactive Functions in Shiny

- **render*()**
    - Builds something that is displayed
    - Will rerun the entire block of code every time the reactive value changes
    - Need to be saved as `output$`

Carrier

UA



Show 10 entries

Search:

	month	n
1	1	3657
2	2	3433
3	3	3913
4	4	4025
5	5	3874
6	6	3931
7	7	4046
8	8	4050
9	9	3573
10	10	3875

Showing 1 to 10 of 12 entries

Previous 1 2 Next

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



  output$plot<-renderPlot({
    flights %>%
      filter(carrier == input$carrier, origin == 'EWR') %>%
      ggplot()+ geom_smooth(aes(x=month, y= dep_delay))+
      xlab('Month')+
      ylab('Departure Delay (minutes)')+
      theme_classic()

  })

  output$table<- renderDT({
    flights %>%
      filter(carrier == input$carrier, origin == 'EWR') %>%
      count(month) %>%
      datatable()

  })
}
```

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

 data <- flights %>%
      filter(carrier == input$carrier, origin == 'EWR')

  output$plot<-renderPlot({
    data %>%
      ggplot()+ geom_smooth(aes(x=month, y= dep_delay))+
      xlab('Month')+
      ylab('Departure Delay (minutes)')+
      theme_classic()

  })

  output$table<- renderDT({
    data %>%
      count(month) %>%
      datatable()

  })
}
```

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

 data <- flights %>%
      filter(carrier == input$carrier, origin == 'EWR')

    data %>%
      count(month) %>%
      datatable()

  })
}
```

Do you need to wrap inside reactive() or observer()?
62: <Anonymous>
Error : Can't access reactive value 'carrier' outside of reactive consumer.
ℹ Do you need to wrap inside reactive() or observer()?

Reactive Functions in Shiny

- **reactive()**

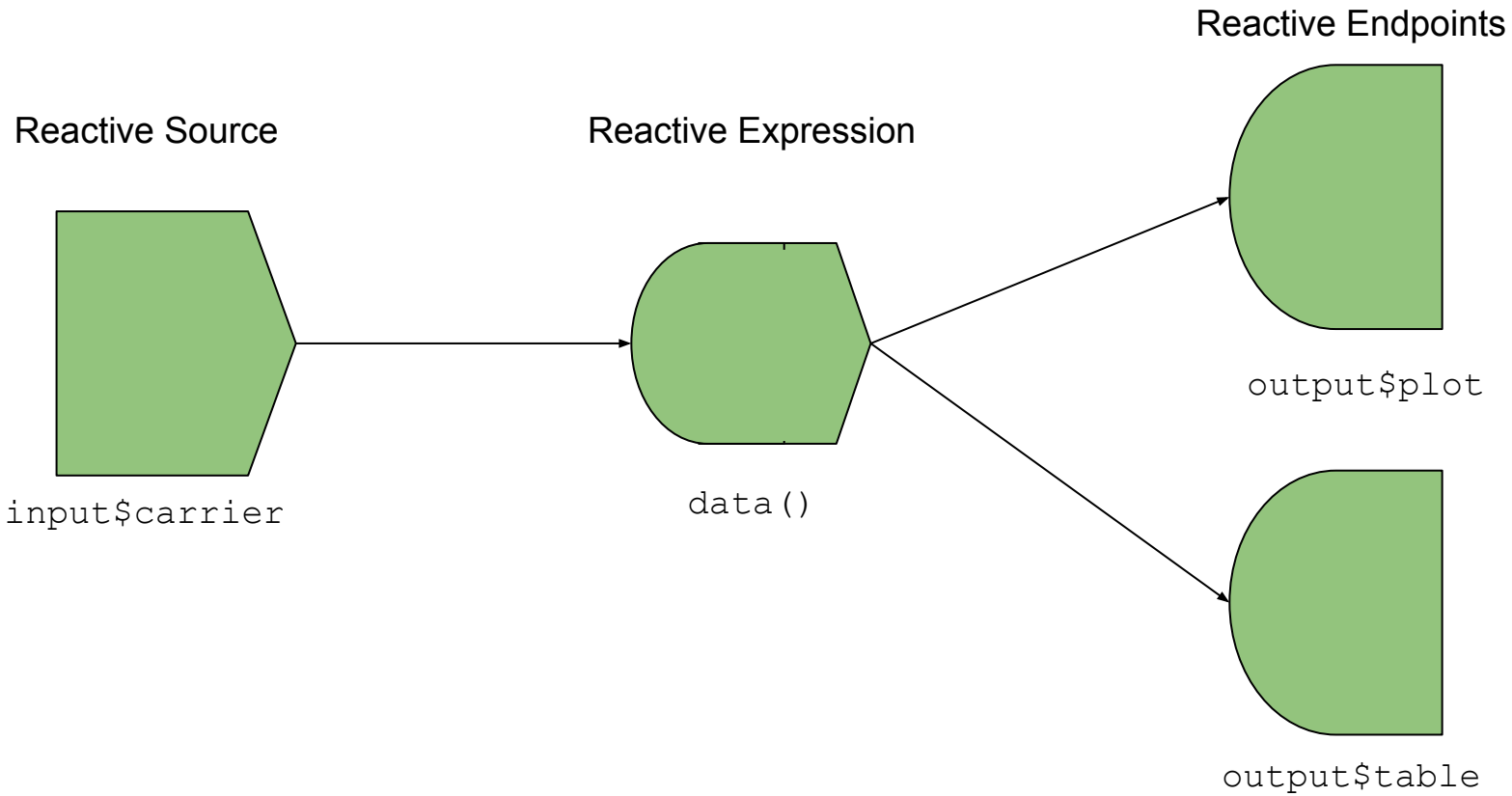
```
data <- reactive({ flights %>%
      filter(carrier == input$carrier, origin == 'EWR')  })
```

Reactive Functions in Shiny

- **reactive()**
    - Builds a reactive object called a reactive expression
    - Returns a value
    - Responds to any reactive value in the code
    - Creates an object you can use downstream in the code
        - In our example call it with: `data()`

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

 data <- reactive({ flights %>%
      filter(carrier == input$carrier, origin == 'EWR')  })

  output$plot<-renderPlot({
    data() %>%
      ggplot()+ geom_smooth(aes(x=month, y= dep_delay))+
      xlab('Month')+
      ylab('Departure Delay (minutes)')+
      theme_classic()

  })

  output$table<- renderDT({
    data() %>%
      count(month) %>%
      datatable()

  })
}
```

A little about Reactivity



For a lot more about Reactivity

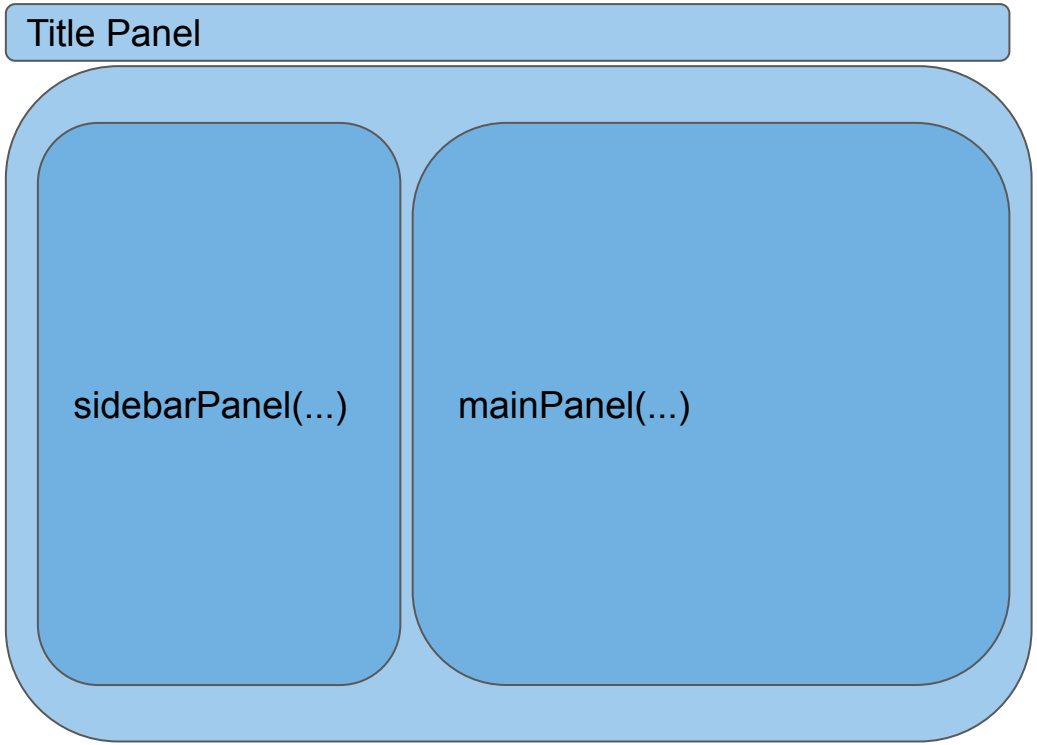
- `isolate()` - returns a result as a non-reactive value
- `observeEvent()` – triggers code to run based on input, you specify which reactive value (e.g. action button)
- `observer()` - triggers code to run (code will runs when any reactive value in code is updated)
- `eventReactive()` - Creates a reactive expression that responds to a specific reactive value(s), used to delay reactions
- `reactiveValues()` - for when you'd like a reactive list

(https://community.rstudio.com/t/reactivevalues-vs-reactive-and-eventreactive-a-general-question/27449/3)

To infinity and beyond

Upgrading Shiny Aesthetics

Shiny Layouts




```
ui <- fluidPage(
    titlePanel("Old Faithful
Geyser Data"),
    sidebarLayout(

        sidebarPanel(
            ...
        ),

        mainPanel(
         ...
        )
    )
)
```

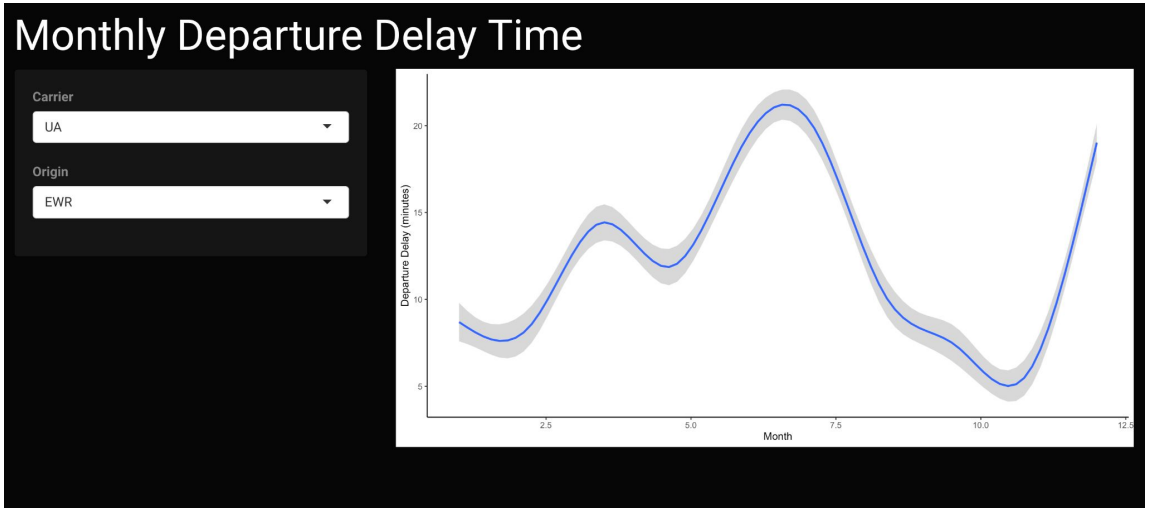
Shiny Layouts

- https://shiny.rstudio.com/articles/layout-guide.html
- https://shiny.rstudio.com/gallery/

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

Changing the aesthetic with shinythemes

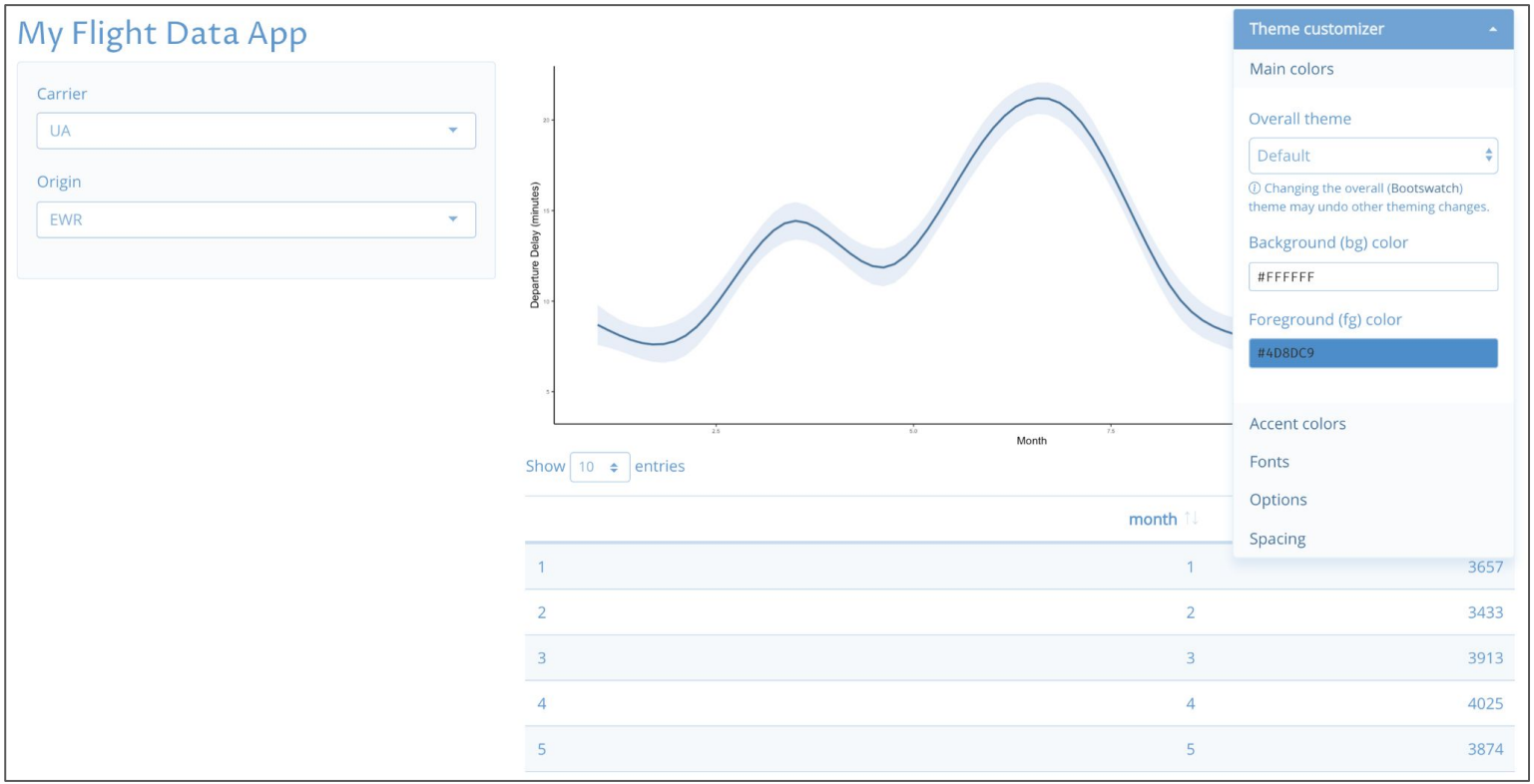
```
ui <- fluidPage(theme=
shinytheme("cyborg"),
```



Check out the gallery of shiny themes here.

Changing the aesthetic with bslib and thematic

Changing the aesthetic with bslib and thematic



Sharing your shiny app

Deploy to the cloud

Shinyapps.io

Host your Shiny apps on the web in minutes with Shinyapps.io. It is easy to use, secure, and scalable. No hardware, installation, or annual purchase contract required. Free and paid options available.

Learn more | FAQ

Deploy on-premises (open source)

Shiny Server

Deploy your Shiny apps and interactive documents on-premises with open source Shiny Server, which offers features such as multiple apps on a single server and deployment of apps behind firewalls.

Learn more

Deploy on-premises (commercial)

RStudio Connect

RStudio Connect is our flagship publishing platform for the work your teams create in R. With RStudio Connect, you can share Shiny applications, R Markdown reports, dashboards, plots, and more in one convenient place with push-button publishing from the RStudio IDE. Features include scheduled execution of reports and flexible security policies to bring the power of data science to your entire enterprise.

Learn more | FAQ

Keep Learning about Shiny!

- Shiny Gallery
- Shiny tutorials
- Shiny Cheatsheet
- Mastering Shiny
- NEW!: https://rstudio-education.github.io/shiny-course/