Shiny – R Package for Webapp development

Marta Galvani

Riccardo Lorenzon

Motivation

- R is often used for running statistical algorithms
- Visualisation and presentation of results is often done in a static form
- No fast reply with quantitative arguments to final consumer questions

Idea

- Provide the audience with a deeper understanding of analytics
- Have a webapp development tool easily accessible to the analyst
- Develop completely in R language and within Rstudio: Shiny

Agenda

- What is a Shiny App
- Server vs UI
- Reactive functions
- External content integration

- Get started with R and Shiny
- Hands-on (guided)
- Hands-on (challenge)

Shiny

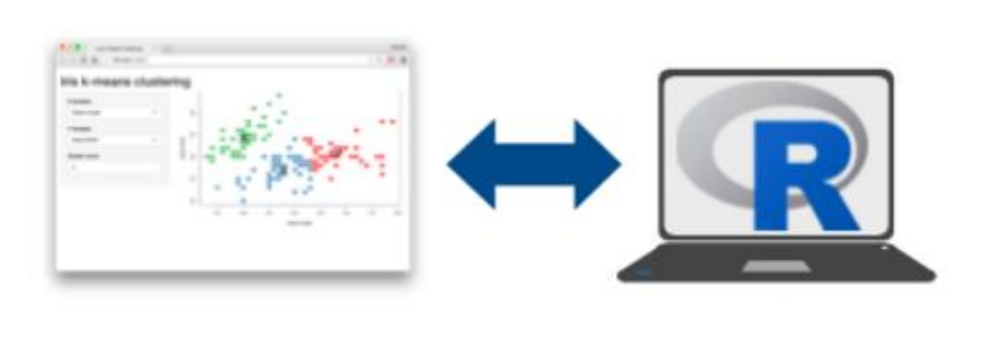
- R Shiny = R + interactivity + web made easy
  In words: Open source R package from Rstudio that creates interactive web applications around your R analysis and visualizations

- No HTML/CSS/Javascript knowledge required to implement ...

- .... but fully customizable and extensible with HTML/CSS/JavaScript

What is a Shiny app

- A Shiny app is a web page (UI) connected to a computer/server running a live R session (Server)



- Consumers can manipulate the UI, which will cause the server to update the UIs displays (by running R code)

Shiny Apps can be developed with the following template in R:

app.R:

```
> library(shiny)
> ui<-FluidPage()
> server<-function(input,output){}
> shinyApp(ui=ui,server=server)
```

What is a Shiny app

- **ui:** Nested R functions that assemble an HTML user interface for the 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 a functioning app
- Save the template as app.R

- **Alternatively**, split template into two files named ui.R and server.R:

ui.R: `> FluidPage()` **server.R:** `> function(input,output){}`

- **Remark:** No need to call shinyApp()
- Save each app as a directory that contains an app.R file (or a ui.R file and a server.R file) plus optional extra files

UI.R

```
shinyUI(bootstrapPage(
  selectInput(inputId = "n_breaks",
    label = "Number of bins in histogram
(approximate):",
    choices = c(10, 20, 35, 50),
    selected = 20),

    plotOutput(outputId = "main_plot", height =
"300px")
))
```

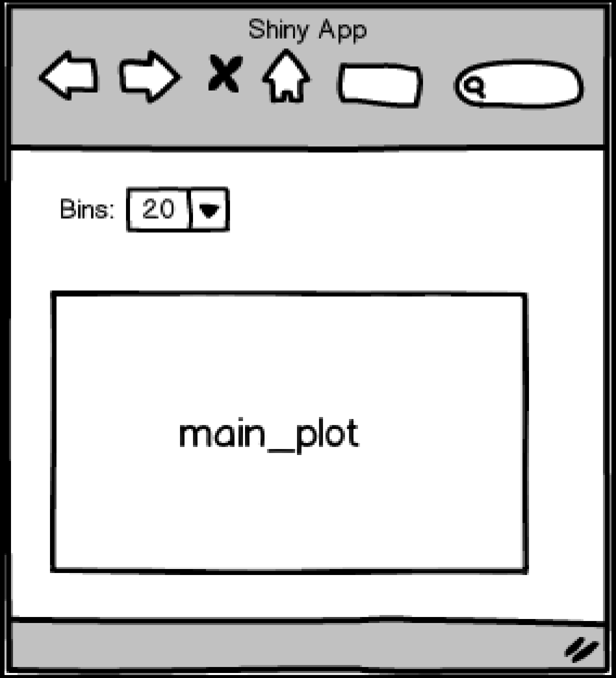
UI.R

```
shinyUI(bootstrapPage(
  selectInput(inputId = "n_breaks",
    label = "Number of bins in
     histogram (approximate):",
    choices = c(10, 20, 35, 50),
    selected = 20),

    plotOutput(outputId = "main_plot", height =
"300px")
))
```

UI.R

```
shinyUI(bootstrapPage(
  selectInput(inputId = "n_breaks",
    label = "Number of bins in
     histogram (approximate):",
    choices = c(10, 20, 35, 50),
    selected = 20),

    plotOutput(outputId = "main_plot", height =
"300px")
))
```

UI.R



```
shinyUI(bootstrapPage(
  selectInput(inputId = "n_breaks"
    label = "Number of bins in
     histogram (approximate):",
    choices = c(10, 20, 35, 50),
    selected = 20),

    plotOutput(outputId = "main_plot", height =
"300px")
))
```

server.R

```r
shinyServer(function(input, output) {
 output$main_plot <- reactivePlot(
  function(){
     hist(faithful$eruptions,
       probability = TRUE,
       breaks = as.numeric(input$n_breaks),
       xlab = "Duration (minutes)",
       main = "Geyser eruption duration")
  })
})
```

server.R

```
shinyServer(function(input, output) {
 output$main_plot <- reactivePlot(
  function(){
     hist(faithful$eruptions,
       probability = TRUE,
       breaks = as.numeric(input$n_breaks),
       xlab = "Duration (minutes)",
       main = "Geyser eruption duration")
  })
})
```

server.R

```
shinyServer(function(input, output) {
 output$main_plot <- reactivePlot(
  function(){
     hist(faithful$eruptions,
       probability = TRUE,
       breaks = as.numeric(input$n_breaks),
       xlab = "Duration (minutes)",
       main = "Geyser eruption duration")
  })
})
```

server.R

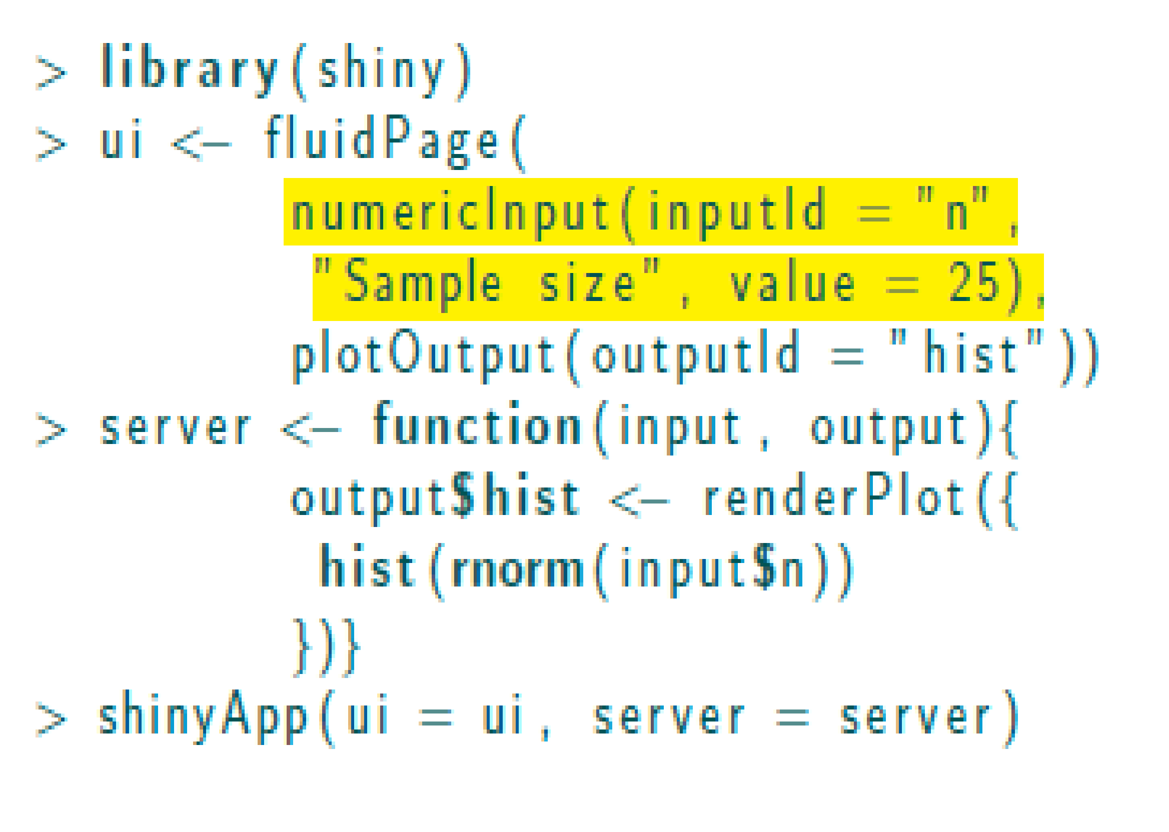
```
shinyServer(function(input, output) {
 output$main_plot <- reactivePlot(
  function(){
     hist(faithful$eruptions,
       probability = TRUE,
       breaks = as.numeric(input$n_breaks),
       xlab = "Duration (minutes)",
       main = "Geyser eruption duration")
  })
})
```

server.R

```r
shinyServer(function(input, output) {
 output$main_plot <- reactivePlot(
  function(){
     hist(faithful$eruptions,
       probability = TRUE,
       breaks = as.numeric(input$n_breaks),
       xlab = "Duration (minutes)",
       main = "Geyser eruption duration")
  })
})
```

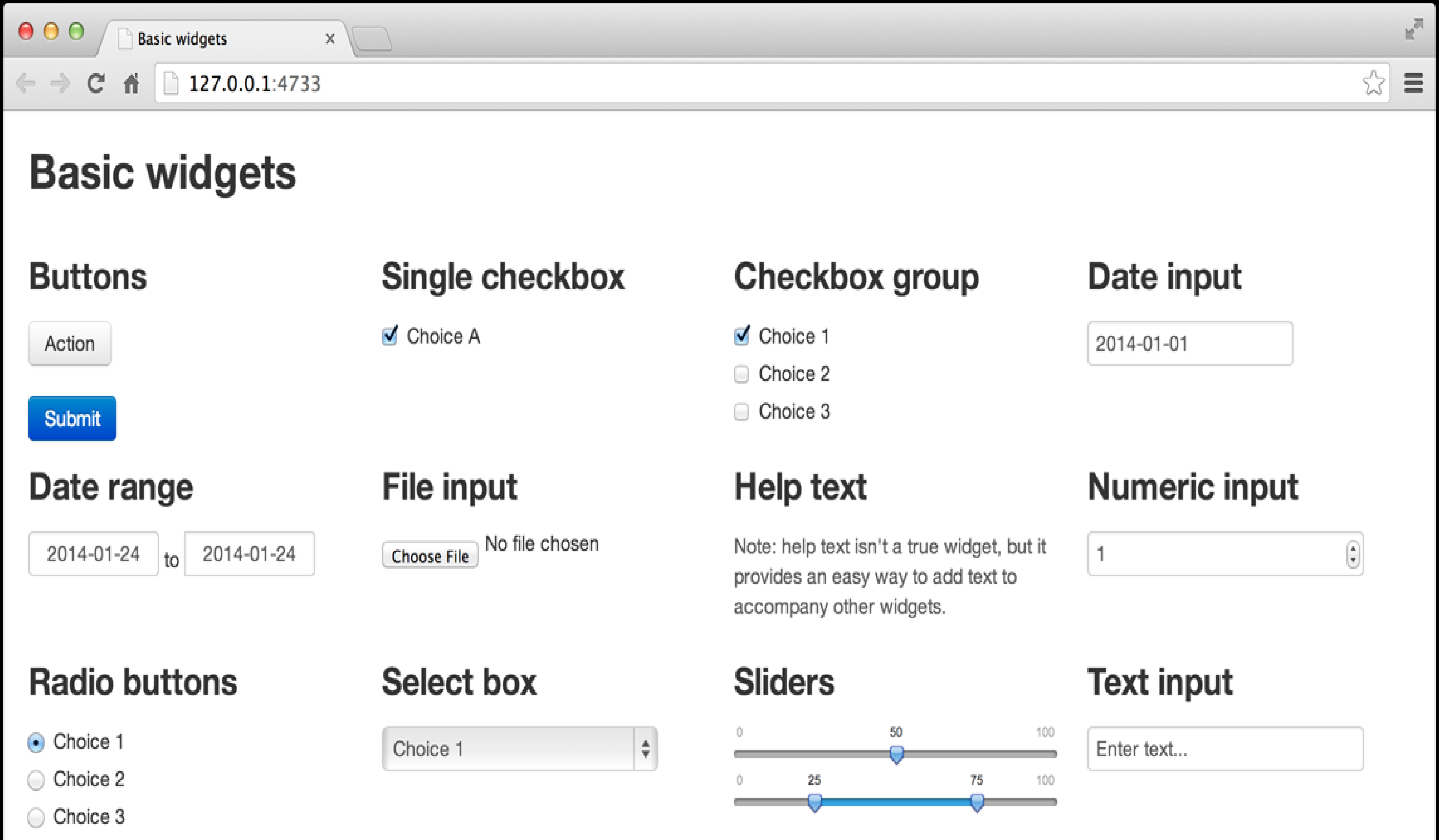
Reactive input functions

- Input values are reactive
- Access the current value of an input object with `input$<inputId>`



```
> 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)
```

Reactive input functions



Basic widgets

Buttons

Single checkbox

☑ Choice A

Checkbox group

☑ Choice 1
☐ Choice 2
☐ Choice 3

Date input

2014-01-01

Date range

2014-01-24 to 2014-01-24

File input

No file chosen

Help text

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

Numeric input

1

Radio buttons

● Choice 1
○ Choice 2
○ Choice 3

Select box

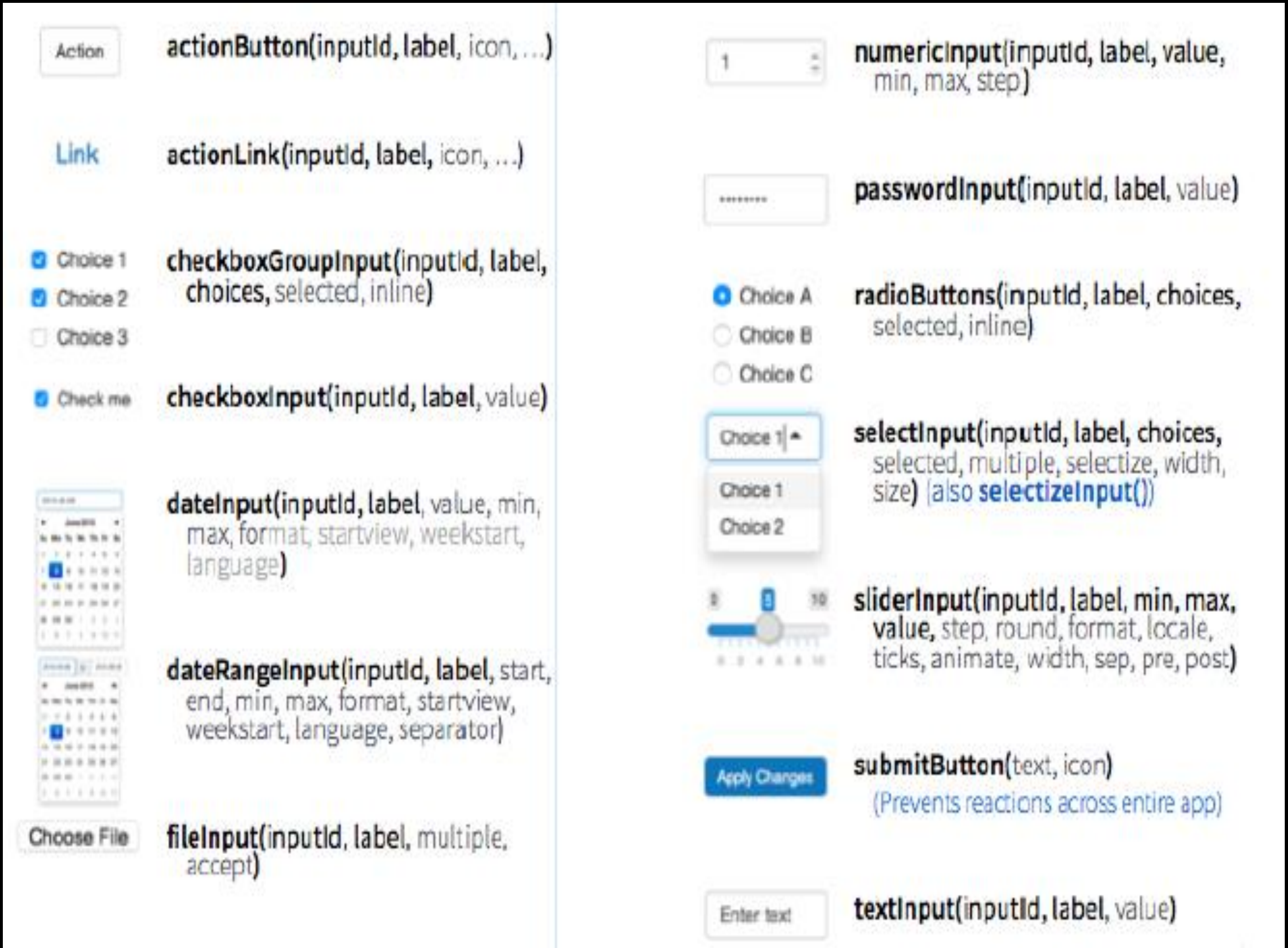
Choice 1

Sliders

Text input

Enter text...

Reactive input functions



actionButton(inputId, label, icon, …)

actionLink(inputId, label, icon, …)

checkboxGroupInput(inputId, label, choices, selected, inline)

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)

fileInput(inputId, label, multiple, accept)

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

passwordInput(inputId, label, value)

radioButtons(inputId, label, choices, selected, inline)

selectInput(inputId, label, choices, selected, multiple, selectize, width, size) (also selectizeInput())

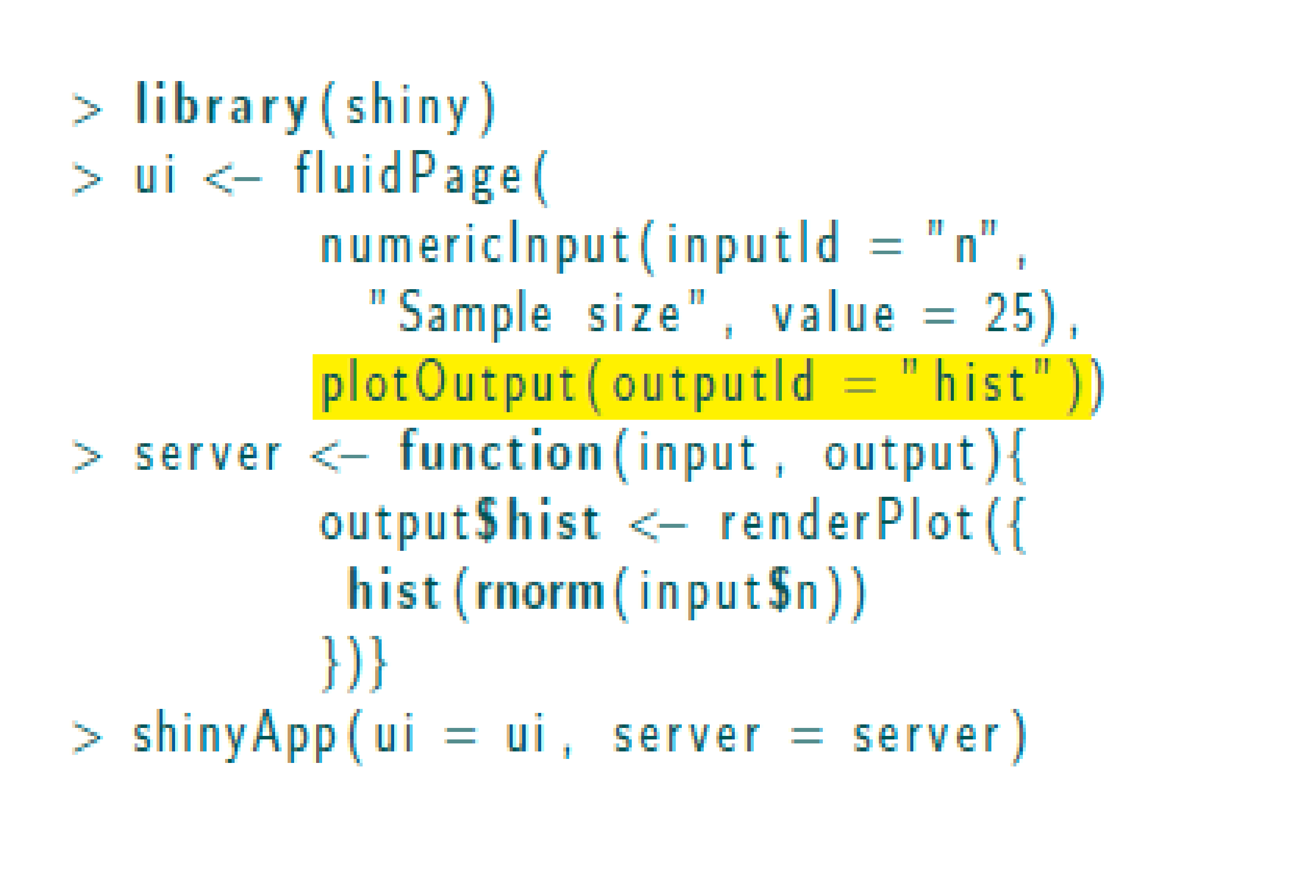
sliderInput(inputId, label, min, max, value, step, round, format, locale, ticks, animate, width, sep, pre, post)

submitButton(text, icon)
(Prevents reactions across entire app)

textInput(inputId, label, value)

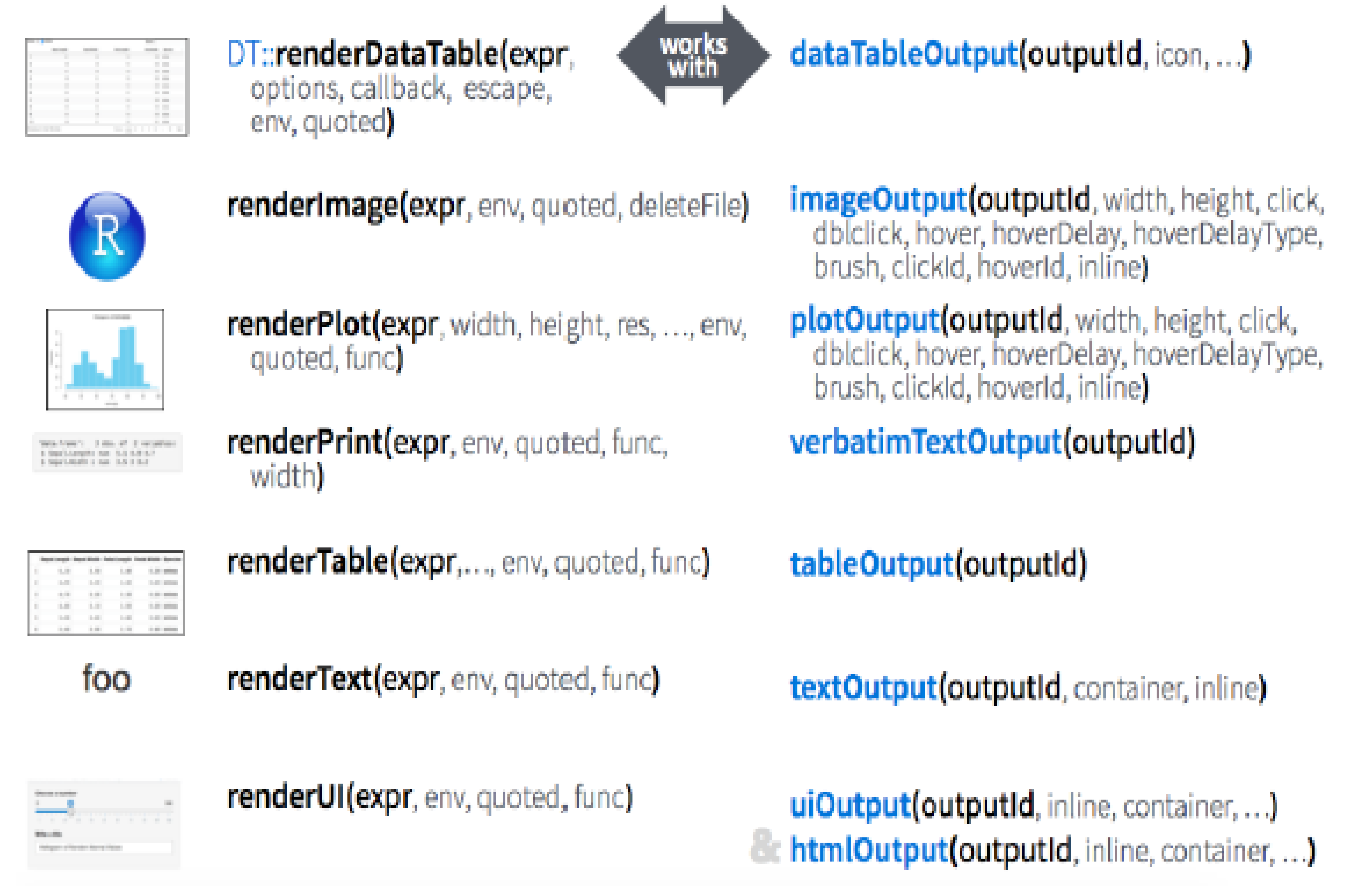
Reactive output functions

- Used to add R output to the UI framework
- Access the developed output of an output object with `output$<outputId>`



```
> 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)
```

Reactive output functions



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)

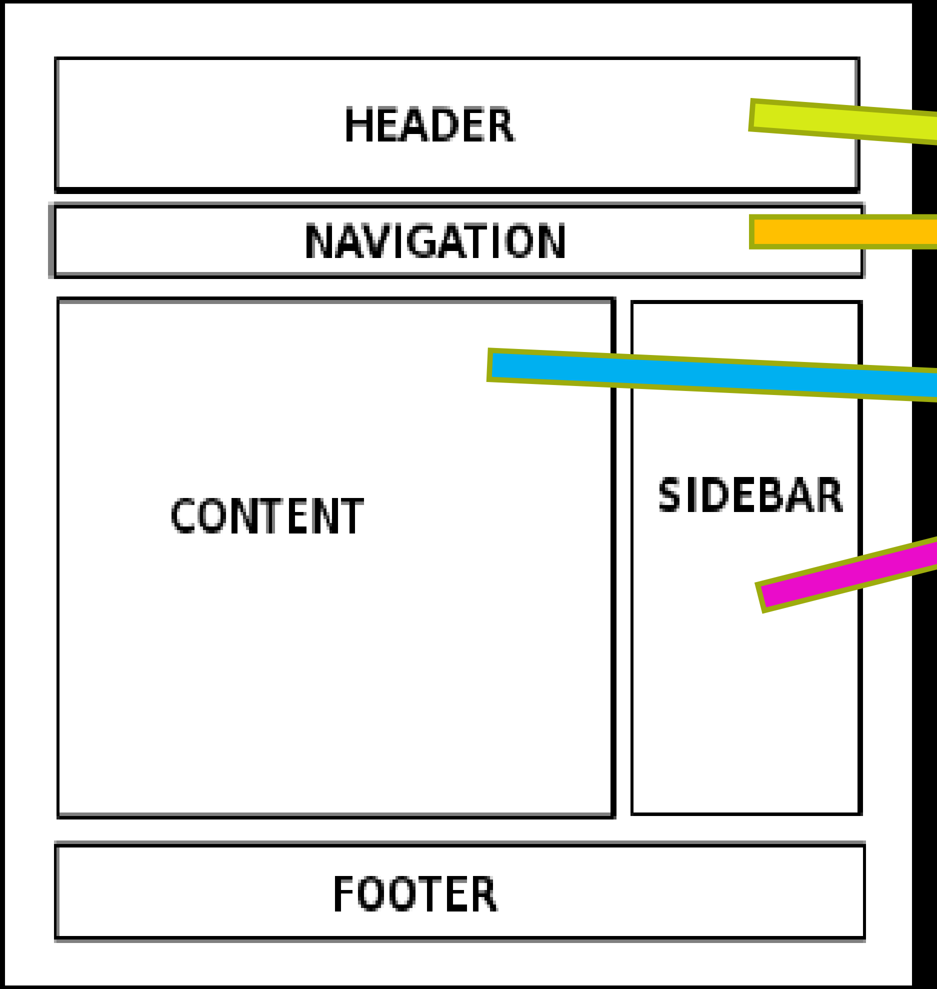
foo

renderText(expr, env, quoted, func) — textOutput(outputId, container, inline)

renderUI(expr, env, quoted, func) — uiOutput(outputId, inline, container, ...) & htmlOutput(outputId, inline, container, ...)

Interface development

- The interface is developed within the UI framework



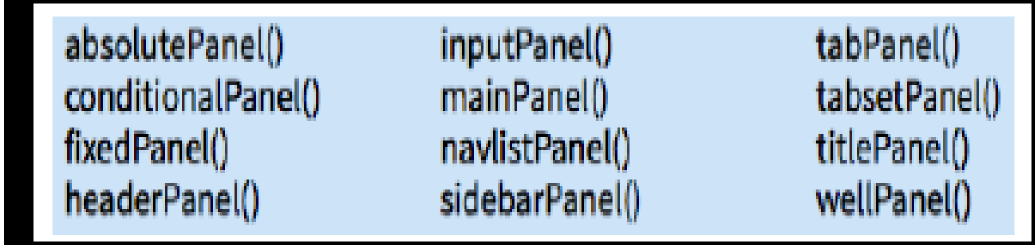
HeaderPanel()

TabPanel()

MainPanel()

sidebarPanel()

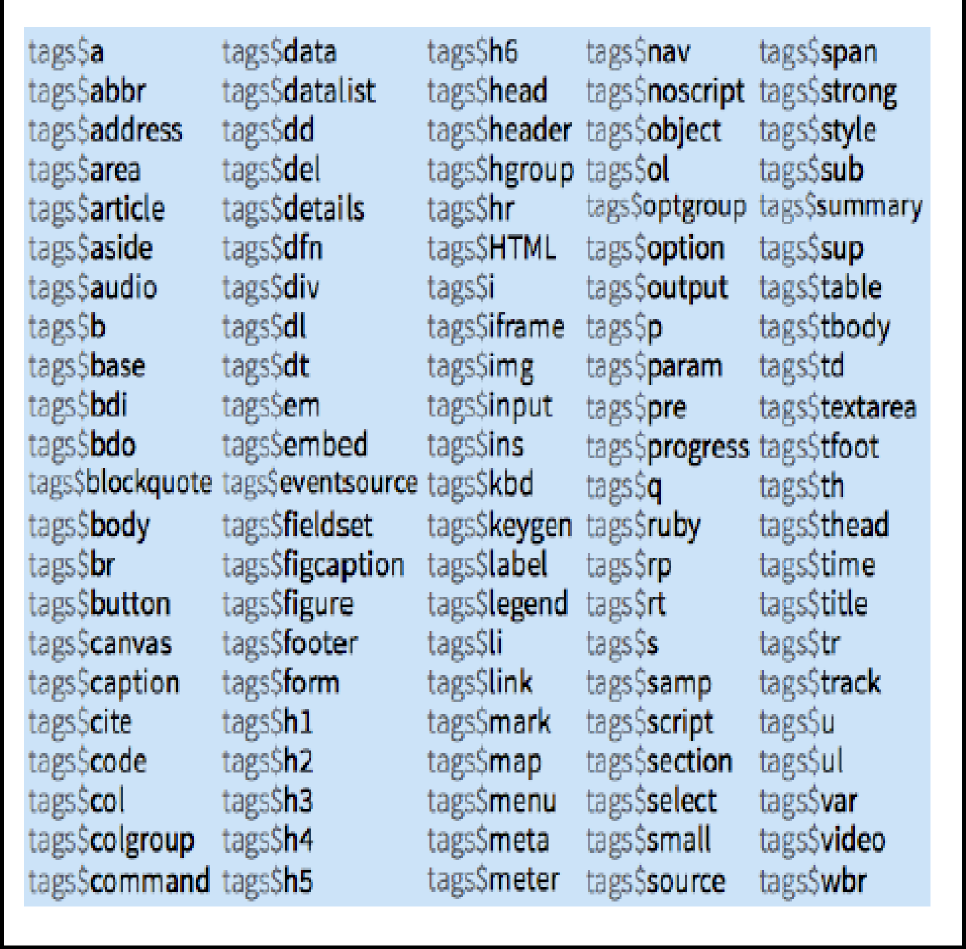
Main different panel type are available:



absolutePanel()
conditionalPanel()
fixedPanel()
headerPanel()
inputPanel()
mainPanel()
navlistPanel()
sidebarPanel()
tabPanel()
tabsetPanel()
titlePanel()
wellPanel()

Assemble UI framework

- An app's UI is actually an HTML document
- Static HTML elements can be added with tags, a list of functions that parallel common HTML tags, e.g. `tags$a()`



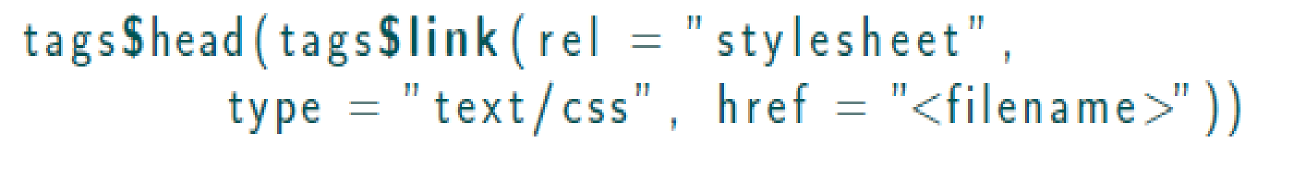
- tags$a
- tags$abbr
- tags$address
- tags$area
- tags$article
- tags$aside
- tags$audio
- tags$b
- tags$base
- tags$bdi
- tags$bdo
- tags$blockquote
- tags$body
- tags$br
- tags$button
- tags$canvas
- tags$caption
- tags$cite
- tags$code
- tags$col
- tags$colgroup
- tags$command
- tags$data
- tags$datalist
- tags$dd
- tags$del
- tags$details
- tags$dfn
- tags$div
- tags$dl
- tags$dt
- tags$em
- tags$embed
- tags$eventsource
- tags$fieldset
- tags$figcaption
- tags$figure
- tags$footer
- tags$form
- tags$h1
- tags$h2
- tags$h3
- tags$h4
- tags$h5
- tags$h6
- tags$head
- tags$header
- tags$hgroup
- tags$hr
- tags$HTML
- tags$i
- tags$iframe
- tags$img
- tags$input
- tags$ins
- tags$kbd
- tags$keygen
- tags$label
- tags$legend
- tags$li
- tags$link
- tags$mark
- tags$map
- tags$menu
- tags$meta
- tags$meter
- tags$nav
- tags$noscript
- tags$object
- tags$ol
- tags$optgroup
- tags$option
- tags$output
- tags$p
- tags$param
- tags$pre
- tags$progress
- tags$q
- tags$ruby
- tags$rp
- tags$rt
- tags$s
- tags$samp
- tags$script
- tags$section
- tags$select
- tags$small
- tags$source
- tags$span
- tags$strong
- tags$style
- tags$sub
- tags$summary
- tags$sup
- tags$table
- tags$tbody
- tags$td
- tags$textarea
- tags$tfoot
- tags$th
- tags$thead
- tags$time
- tags$title
- tags$tr
- tags$track
- tags$u
- tags$ul
- tags$var
- tags$video
- tags$wbr

Assemble UI framework

- Several files can be included as well:

- CSS:

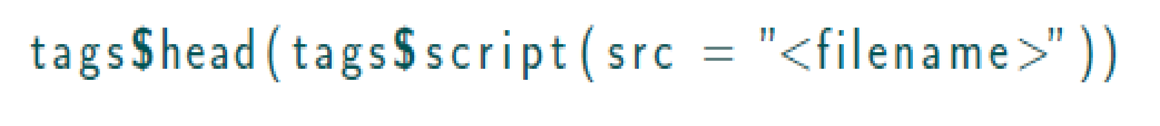
  1. Place the file in the www subdirectory
  2. Link to it with:



```
tags$head(tags$link(rel = "stylesheet",
          type = "text/css", href = "<filename>"))
```

- Javascript:

  1. Place the file in the www subdirectory
  2. Link to it with:



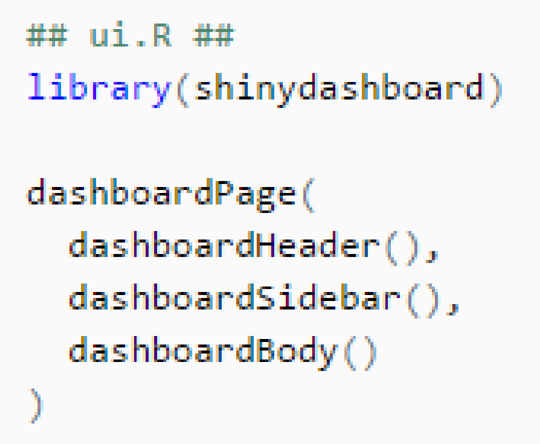
```
tags$head(tags$script(src = "<filename>"))
```

- Image:

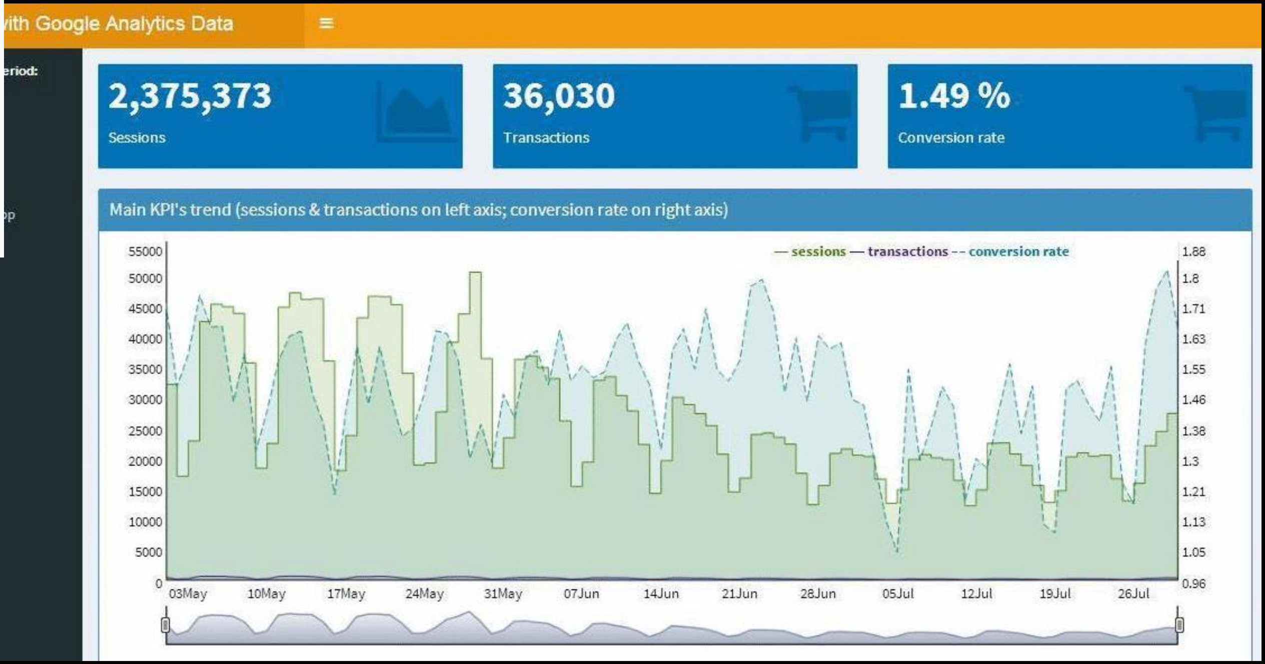
  1. Place the file in the www subdirectory
  2. Link to it with: `img(src = "<filename>")`

Shiny Dashboard

- Additional R package built on the top of Shiny to easily build fancy Dashboards. The structure of a Dashboard is:



```
## ui.R ##
library(shinydashboard)

dashboardPage(
  dashboardHeader(),
  dashboardSidebar(),
  dashboardBody()
)
```



2,375,373
Sessions

36,030
Transactions

1.49 %
Conversion rate

Main KPI's trend (sessions & transactions on left axis; conversion rate on right axis)

Shiny Examples

Get started

- Install R and Rstudio:

  https://cran.rstudio.com/

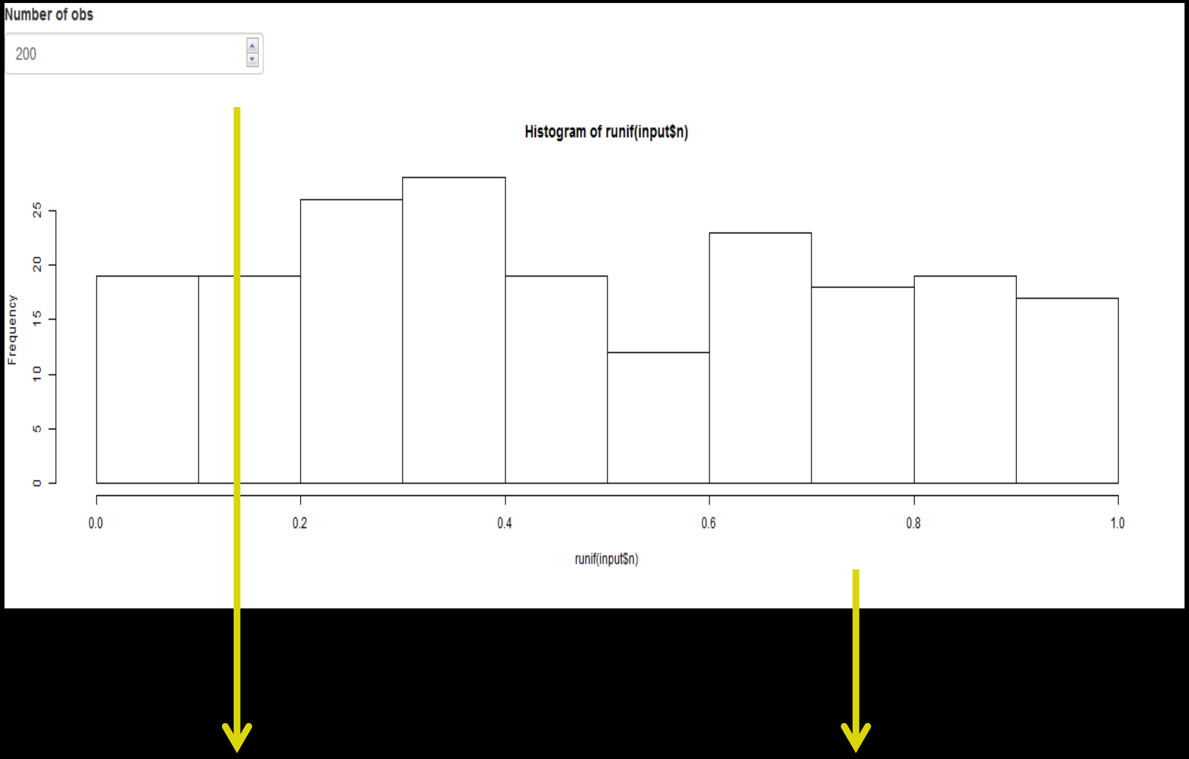
  https://www.rstudio.com/products/rstudio/download/

- Open Rstudio
- Install packages «shiny» and «shinydashboard»
- Useful cheatsheet: http://shiny.rstudio.com/articles/cheatsheet.html

Example 1: Reactive plot

```
library(shiny)
# Global variables can go here
n <- 200

# Define the UI
ui <- bootstrapPage(
  numericInput('n', 'Number of obs', n),
  plotOutput('plot')
)

# Define the server code
server <- function(input, output) {
  output$plot <- renderPlot({
    hist(runif(input$n))
  })
}

# Return a Shiny app object
shinyApp(ui = ui, server = server)
```

Create a histogram of a uniform distribution of N observations. N is chosen by the user.

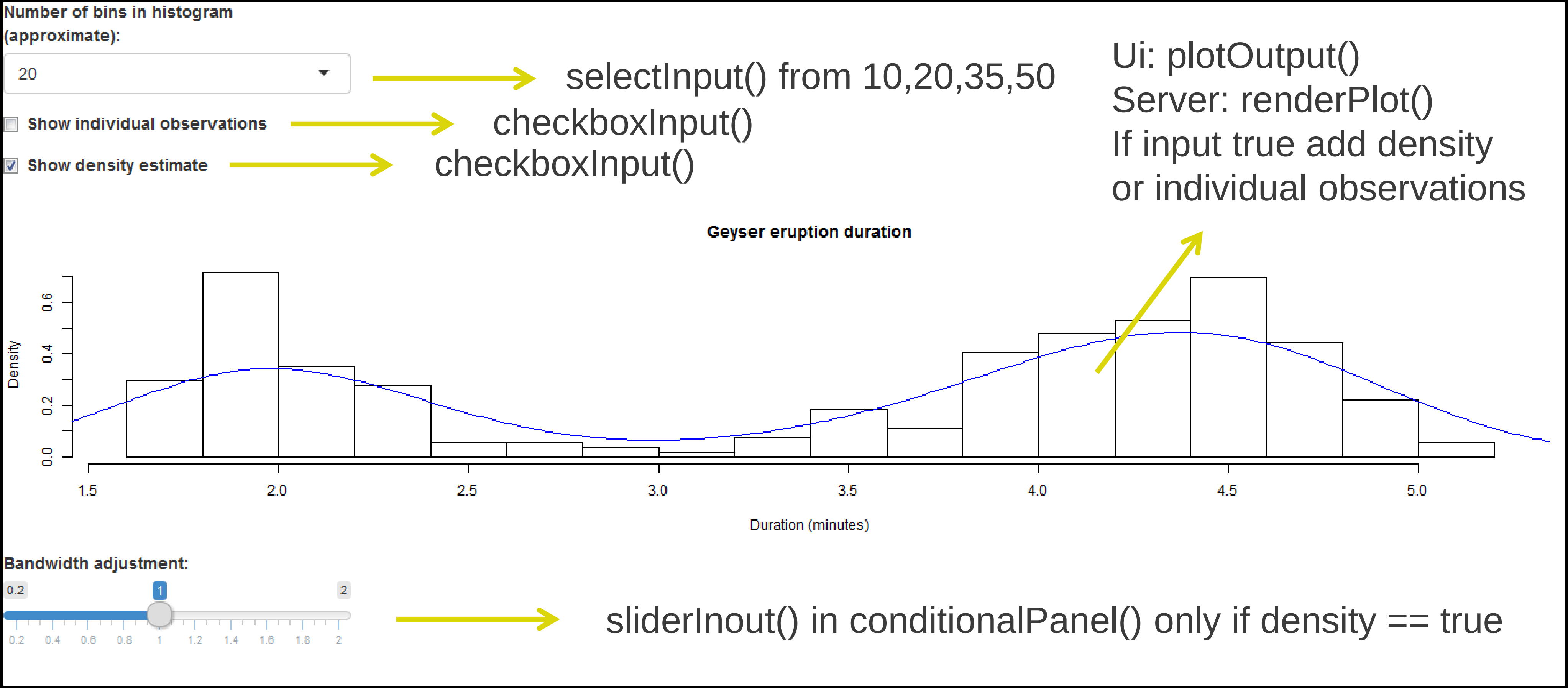

Number of obs

200

numericInput()

UI: plotOutput()
Server: renderPlot()

Example 2: Reactive plot



Number of bins in histogram (approximate):

20

□ Show individual observations

☑ Show density estimate

selectInput() from 10,20,35,50

checkboxInput()

checkboxInput()

Ui: plotOutput()
Server: renderPlot()
If input true add density
or individual observations

Bandwidth adjustment:

sliderInout() in conditionalPanel() only if density == true

Example 2: Reactive plot

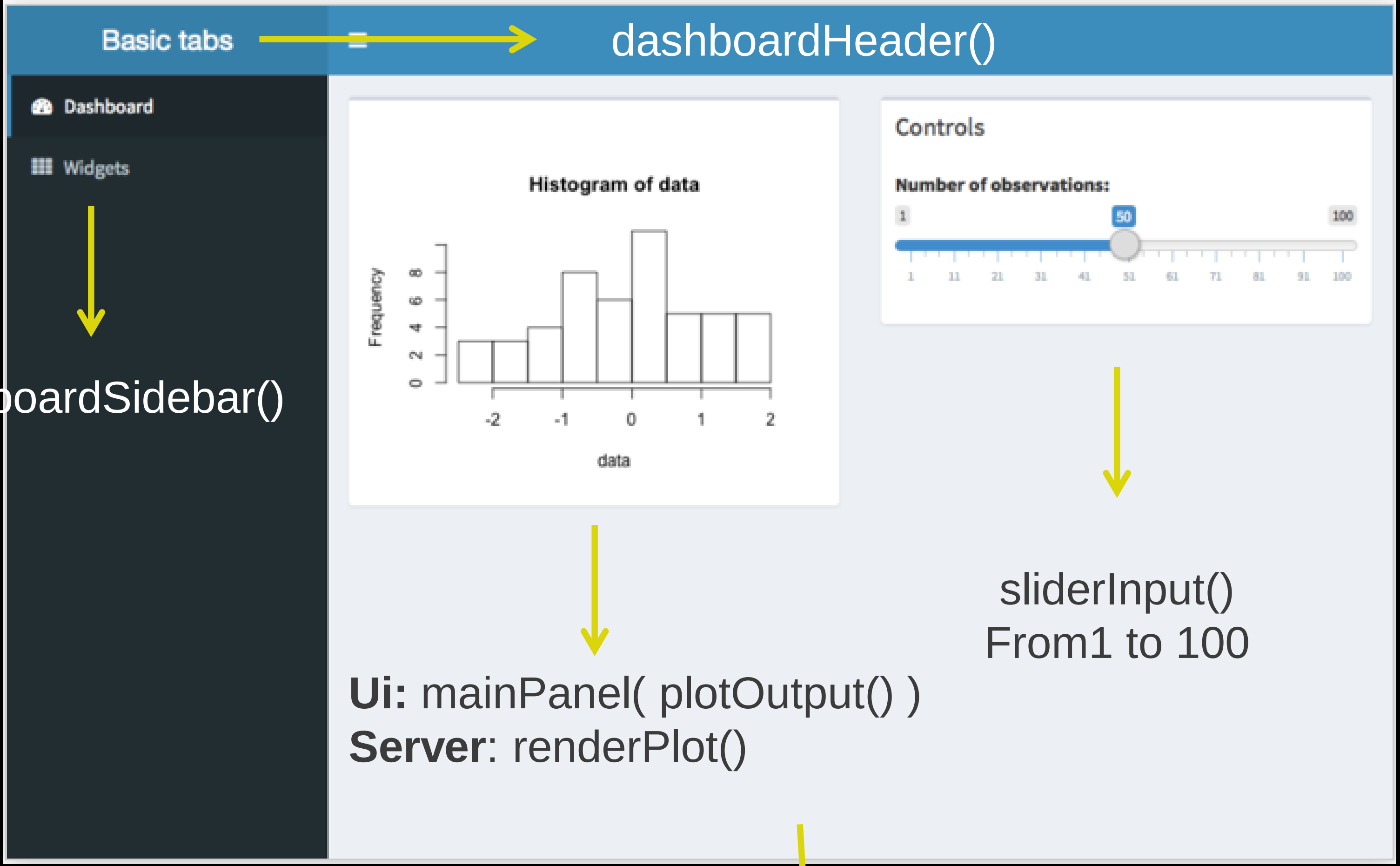
UI

```
ui <- bootstrapPage(

  selectInput(inputId = "n_breaks",
              label = "Number of bins in histogram (approximate):",
              choices = c(10, 20, 35, 50),
              selected = 20),

  checkboxInput(inputId = "individual_obs",
                label = strong("Show individual observations"),
                value = FALSE),

  checkboxInput(inputId = "density",
                label = strong("Show density estimate"),
                value = FALSE),

  plotOutput(outputId = "main_plot", height = "300px"),

  # Display this only if the density is shown
  conditionalPanel(condition = "input.density == true",
                   sliderInput(inputId = "bw_adjust",
                               label = "Bandwidth adjustment:",
                               min = 0.2, max = 2, value = 1, step = 0.2)
  )
)
```

SERVER

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

  output$main_plot <- renderPlot({

    hist(faithful$eruptions,
         probability = TRUE,
         breaks = as.numeric(input$n_breaks),
         xlab = "Duration (minutes)",
         main = "Geyser eruption duration")

    if (input$individual_obs) {
      rug(faithful$eruptions)
    }

    if (input$density) {
      dens <- density(faithful$eruptions,
                      adjust = input$bw_adjust)
      lines(dens, col = "blue")
    }

  })
}
```

Example 3: Shiny Dashboard



dashboardHeader()

dashboardSidebar()

Ui: mainPanel(plotOutput())
Server: renderPlot()

sliderInput()
From1 to 100

dashboardBody(
TabItems(
TabItem()
TabItem()))

Example 3: Shiny Dashboard

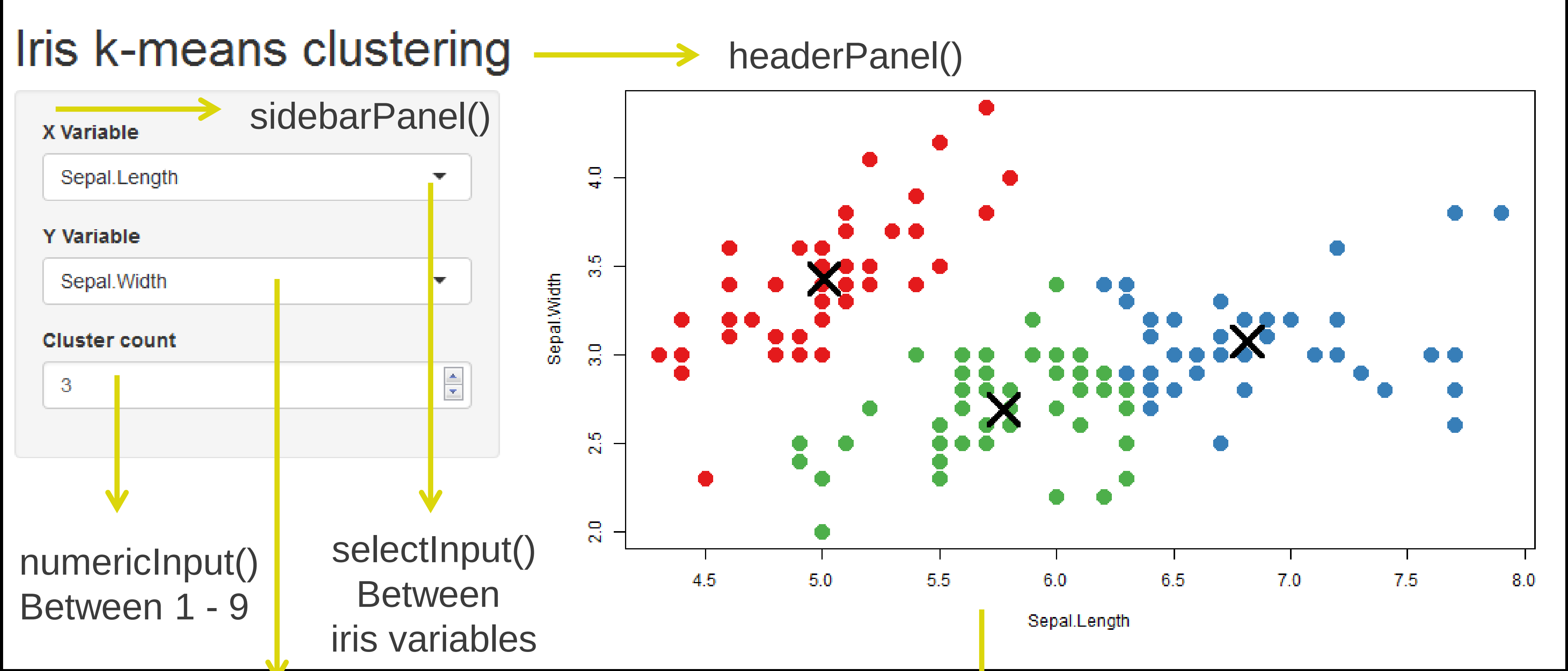
UI

```
ui <- dashboardPage(
  dashboardHeader(title = "Basic dashboard"),
  dashboardSidebar(
    sidebarMenu(
      menuItem("Dashboard", tabName = "dashboard",
                                     icon = icon("dashboard")),
      menuItem("Widgets", tabName = "widgets", icon = icon("th"))
    )
  ),
  dashboardBody(
    tabItems(
      # First tab content
      tabItem(tabName = "dashboard",
              fluidRow(
                box(plotOutput("plot1", height = 250)),

                box(
                  title = "Controls",
                  sliderInput("slider", "Number of observations:", 1, 100, 50)
                )
     )),

      # Second tab content
      tabItem(tabName = "widgets",
              h2("Widgets tab content")
      ))))
```

SERVER

```
server <- function(input, output) {
  set.seed(122)
  histdata <- rnorm(500)

  output$plot1 <- renderPlot({
    data <- histdata[seq_len(input$slider)]
    hist(data)
  })
}
```

Example 4: Reactive K-Means



Iris k-means clustering

X Variable

Sepal.Length

Y Variable

Sepal.Width

Cluster count

3

headerPanel()

sidebarPanel()

numericInput()
Between 1 - 9

selectInput()
Between
iris variables

selectInput()
Between
iris variables

Ui: mainPanel(plotOutput())
Server: select data with reactive function
Kmeans apply with reactive function
renderPlot()

Example 4: Reactive K-Means

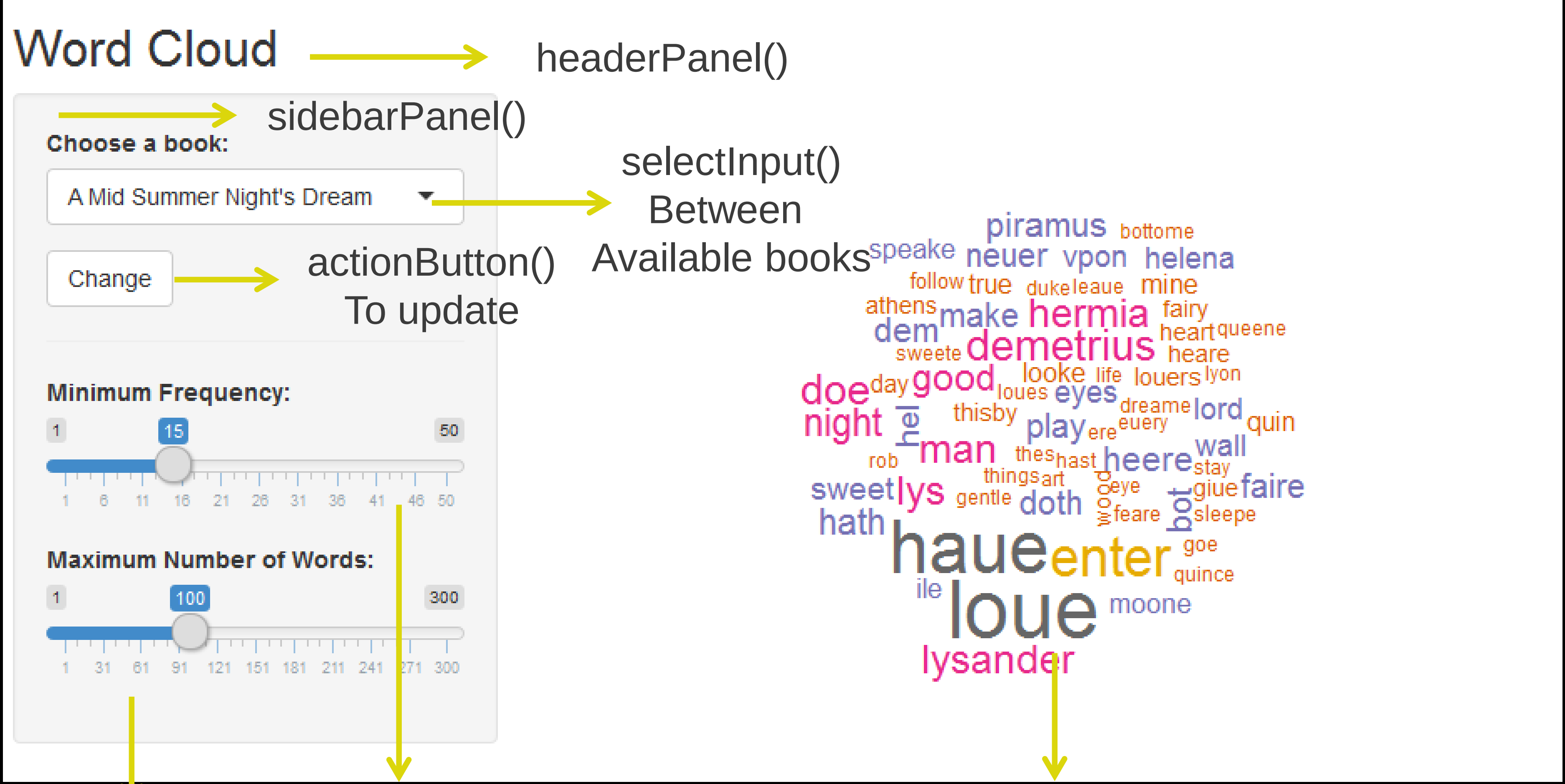
UI

```
ui <- pageWithSidebar(

 headerPanel('Iris k-means clustering'),

  sidebarPanel(

    selectInput('xcol', 'X Variable', names(iris)),

    selectInput('ycol', 'Y Variable', names(iris),
                selected=names(iris)[[2]]),

    numericInput('clusters', 'Cluster count', 3,
                 min = 1, max = 9)

  ),

mainPanel(
    plotOutput('plot1')
  )
)
```

SERVER

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

  # Combine the selected variables into a new data frame
  selectedData <- reactive({
    iris[, c(input$xcol, input$ycol)]
  })

  clusters <- reactive({
    kmeans(selectedData(), input$clusters)
  })

  output$plot1 <- renderPlot({
    palette(c("#E41A1C", "#377EB8", "#4DAF4A", "#984EA3",
              "#FF7F00", "#FFFF33", "#A65628", "#F781BF", "#999999"))

    par(mar = c(5.1, 4.1, 0, 1))
    plot(selectedData(),
         col = clusters()$cluster,
         pch = 20, cex = 3)
    points(clusters()$centers, pch = 4, cex = 4, lwd = 4)
  })
}
```

Example 5: Text Mining



Word Cloud

Choose a book:

A Mid Summer Night's Dream ▾

Change

Minimum Frequency: 1 15 50

Maximum Number of Words: 1 100 300

headerPanel()

sidebarPanel()

selectInput()
Between
Available books

actionButton()
To update

sliderInput()
From 1 to 300

sliderInput()
From 1 to 50

Ui: mainPanel(plotOutput())
Server: create Term Matrix
renderPlot()

Example 5: Text Mining

UI

```
ui <- fluidPage(
  # Application title
  titlePanel("Word Cloud"),

  sidebarLayout(
    # Sidebar with a slider and selection inputs
    sidebarPanel(
      selectInput("selection", "Choose a book:",
                  choices = books),
      actionButton("update", "Change"),
      hr(),
      sliderInput("freq",
                  "Minimum Frequency:",
                  min = 1,  max = 50, value = 15),
      sliderInput("max",
                  "Maximum Number of Words:",
                  min = 1,  max = 300,  value = 100)
    ),

    # Show Word Cloud
    mainPanel(
      plotOutput("plot")
    )
  )
)
```

SERVER

```
server <- function(input, output, session) {
  # Define a reactive expression for the document term matrix
  terms <- reactive({
    # Change when the "update" button is pressed...
    input$update
    # ...but not for anything else
    isolate({
      withProgress({
        setProgress(message = "Processing corpus...")
        getTermMatrix(input$selection)
      })
    })
  })

  # Make the wordcloud drawing predictable during a session
  wordcloud_rep <- repeatable(wordcloud)

  output$plot <- renderPlot({
    v <- terms()
    wordcloud_rep(names(v), v, scale=c(4,0.5),
                  min.freq = input$freq, max.words=input$max,
                  colors=brewer.pal(8, "Dark2"))
  })
}
```

Thanks

m.galvani@reply.it

r.lorenzon@reply.it