Building data products

library(shiny)

ui <- fluidPage()

User interface:

- Inputs defined and laid out
- Outputs lay out

server <- function(input, output) {}

Server function:

- Outputs calculated
- Any other calculations needed to run the app

shinyApp(ui=ui, server=server)

Runs the application

Best practice:

- You need to have saved at least 2 .R files in your directory

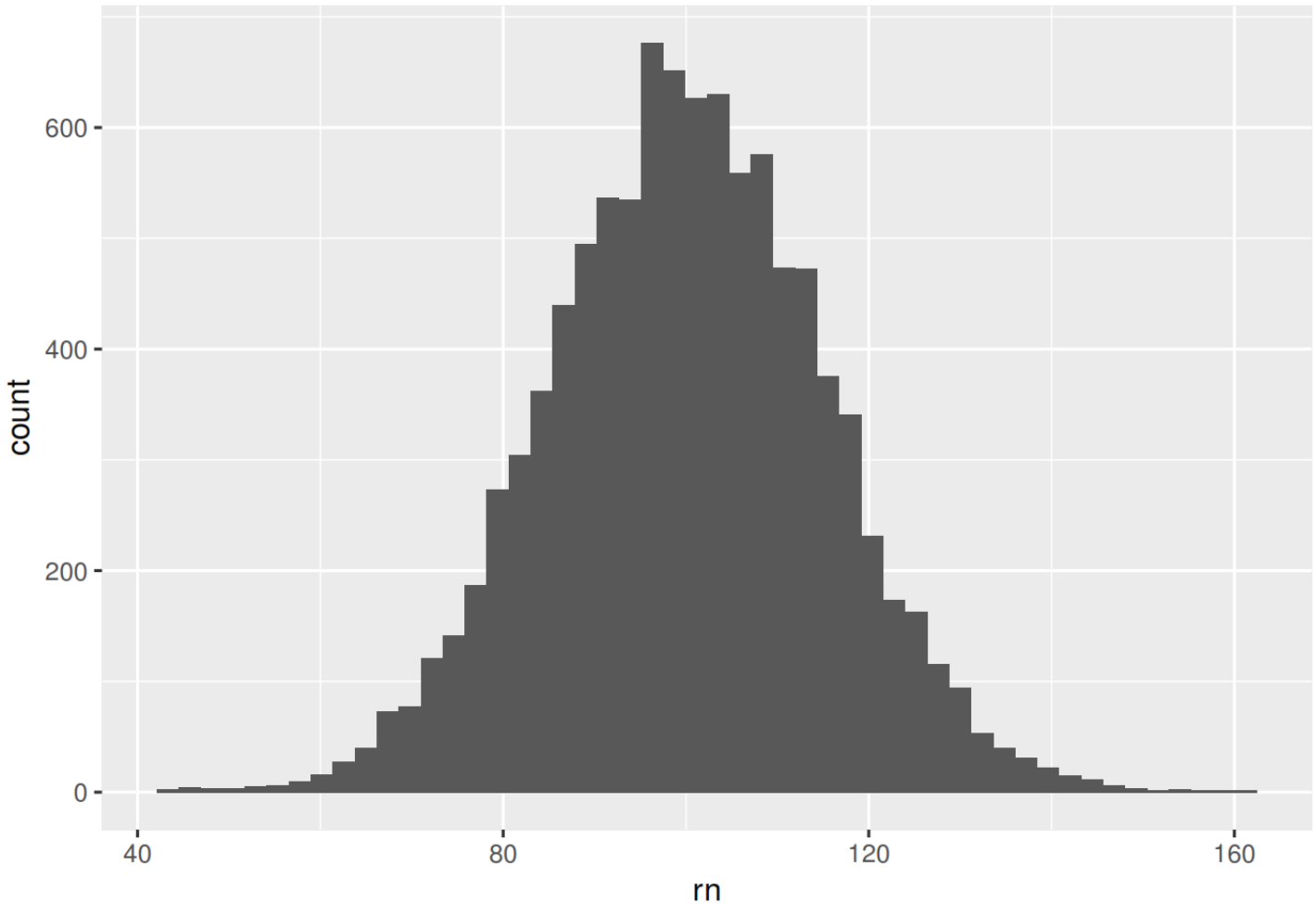
  ui.R

  server.R

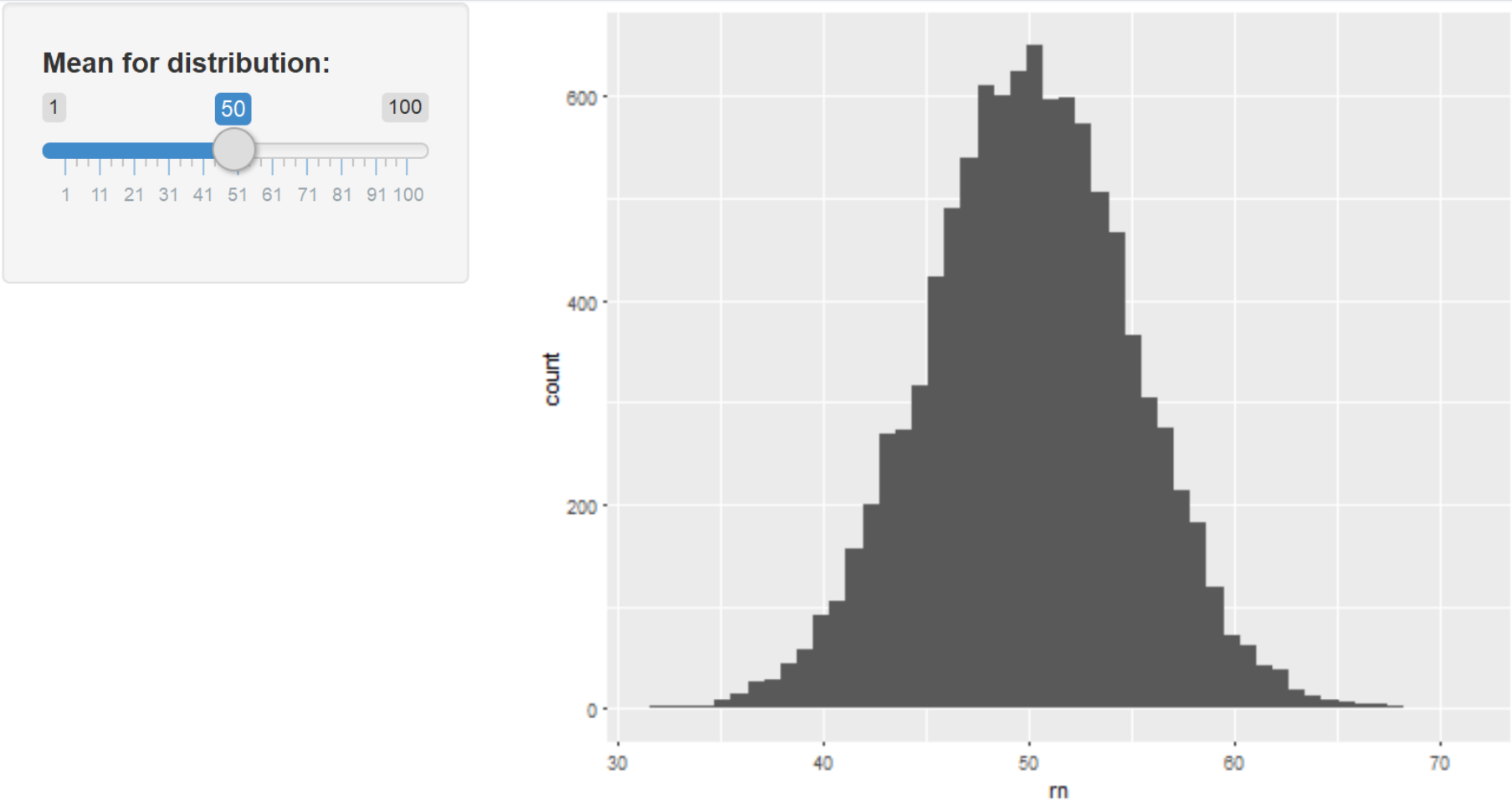
- Also you can have both server and ui in the same R file
- Optionally you can have an R file Global.R that contains functions, code that does not fit in either server or ui.
- For example: user defined functions can be defined in global.R

Create a histogram by changing the parameter for mean

```
rn <- rnorm(n=10000, mean=100, sd=15)
ggplot()+geom_histogram(aes(x=rn), bins = 50)
```

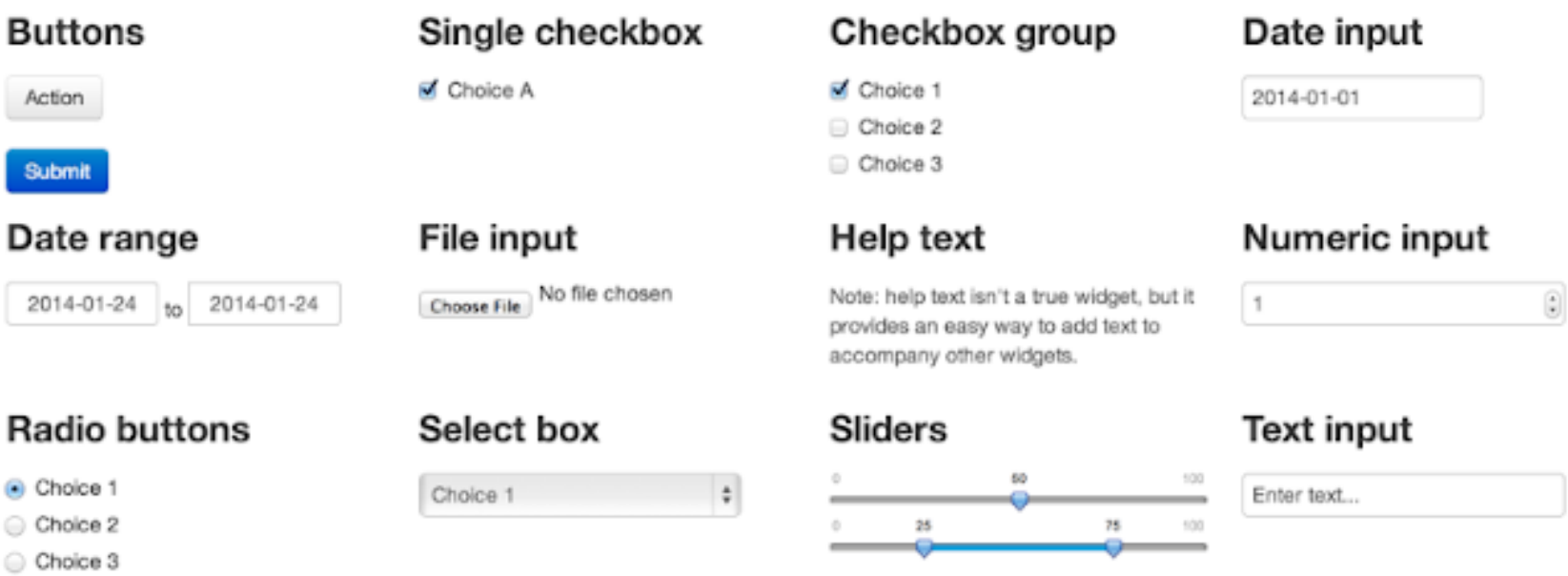


Building web applications with shiny



Selectors

The widget gallery with codes



To run the app don't forget to add shinyApp(ui = ui, server = server) to your .R file

```
ui <- fluidPage(
sidebarLayout(
  sidebarPanel(
    sliderInput(inputId = "mu",
                label = "Mean for distribution:",
                min = 1,max = 100, value = 50)),
  mainPanel(plotOutput("histogram"))
  ))
```

```
server <- function(input, output){
  output$histogram <- renderPlot({
    rn <- rnorm(n=10000, mean=input$mu, sd=5)
    ggplot()+geom_histogram(aes(x=rn), bins=50)
  })
}
```

- server takes the value for mean from ui
  - to specify input from ui use input$<name of the input>

```
ui <- fluidPage(
sidebarLayout(
  sidebarPanel(
    sliderInput(inputId = "mu",
                label = "Mean for distribution:",
                min = 1,max = 100, value = 50)),
  mainPanel(plotOutput("histogram"))
  ))
```

```
server <- function(input, output){
  output$histogram <- renderPlot({
    rn <- rnorm(n=10000, mean=input$mu, sd=5)
    ggplot()+geom_histogram(aes(x=rn), bins=50)
  })
}
```

- Server creates a plot using renderPlot({}) and saves as an object output$histogra,
- to show the plot on ui use plotOutput and the name of the output

```
ui <- fluidPage(
sidebarLayout(
  sidebarPanel(
    sliderInput(inputId = "mu",
                label = "Mean for distribution:",
                min = 1,max = 100, value = 50)),
  mainPanel(plotOutput("histogram"))
  ))
```

```
server <- function(input, output){
  output$histogram <- renderPlot({
    rn <- rnorm(n=10000, mean=input$mu, sd=5)
    ggplot()+geom_histogram(aes(x=rn), bins=50)
  })
}
```

sidebarLayout(

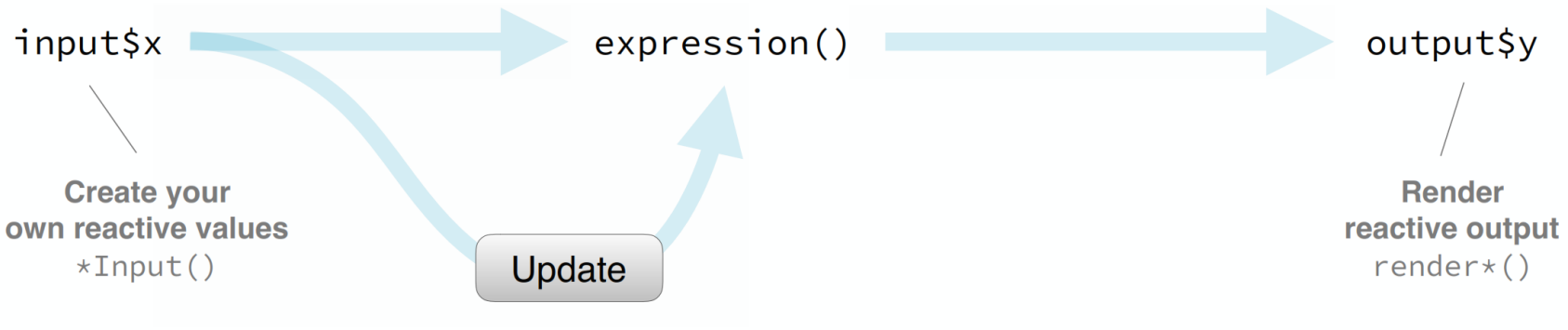
sidebarPanel()
mainPanel())

Controls the layout of the app

```
ui <- fluidPage(
sidebarLayout(
  sidebarPanel(
    sliderInput(inputId = "mu",
                label = "Mean for distribution:",
                min = 1,max = 100, value = 50)),
  mainPanel(plotOutput("histogram"))
  ))
```

```
server <- function(input, output){
  output$histogram <- renderPlot({
    rn <- rnorm(n=10000, mean=input$mu, sd=5)
    ggplot()+geom_histogram(aes(x=rn), bins=50)
  })
}
```

Reactive flow



```
    sliderInput(inputId = "mu",
                label = "Mean for distribution:",
                min = 1,max = 100, value = 50)),
```

inputId – you will use this to refer to the input in server

label – the label of the slider

min, max – the minimum and maximum values that the mean can have

value – the default value

Add another slidebar for standard deviation

ui

```
ui <- fluidPage(

  sidebarLayout(
    sidebarPanel(
      sliderInput(inputId = "mu",
                  label = "Mean for distribution:",
                  min = 1,max = 100, value = 50),
      sliderInput(inputId = "sd",
                  label = "Standard deviation for distribution:",
                  min = 5,max = 50, value = 40)),
    mainPanel(
      plotOutput("histogram"))
  )
)
```

Server

```
server <- function(input, output){
  output$histogram <- renderPlot({
    rn <- rnorm(n=10000, mean=input$mu, sd=input$sd)
    ggplot()+geom_histogram(aes(x=rn), bins=50)
  })
}
```

Add numeric input for sample size

- set the minimum to 100 and maximum to 10,000

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

Step 1: Add an R object to the UI

Shiny provides a family of functions that turn R objects into output for your user interface. Each function creates a specific type of output.

Output function	Creates
dataTableOutput	DataTable
htmlOutput	raw HTML
imageOutput	image
plotOutput	plot
tableOutput	table
textOutput	text
uiOutput	raw HTML
verbatimTextOutput	text

Add calculated median of the vector to the dashboard

```
ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(
      sliderInput(inputId = "mu",
                  label = "Mean for distribution:",
                  min = 1,max = 100, value = 50),
      sliderInput(inputId = "sd",
                  label = "Standard deviation for distribution:",
                  min = 5,max = 50, value = 40)),
    mainPanel(
      plotOutput("histogram"),
      textOutput("text_median"))
  )
)
```

use reactive to make a reactive object, thus an object that reacts to user input

```
server <- function(input, output){
  rn <- reactive({rnorm(n=10000, mean=input$mu, sd=input$sd)})

  output$histogram <- renderPlot({
    ggplot()+geom_histogram(aes(x=rn()), bins=50)
  })

  output$text_median <- renderText({
    median(rn())
  })
}
```

- If you use reactive object in later code, use rn()
- Thus this value will be recalculated only when inputs would change

```
server <- function(input, output){
  rn <- reactive({rnorm(n=10000, mean=input$mu, sd=input$sd)})

  output$histogram <- renderPlot({
    ggplot()+geom_histogram(aes(x=rn()), bins=50)
  })

  output$text_median <- renderText({
    median(rn())
  })
}
```

- use renderText to make a text
- add it to output, with the name output$text_median

```
server <- function(input, output){
  rn <- reactive({rnorm(n=10000, mean=input$mu, sd=input$sd)})

  output$histogram <- renderPlot({
    ggplot()+geom_histogram(aes(x=rn()), bins=50)
  })

  output$text_median <- renderText({
    median(rn())
  })
}
```

Add textOutput to your ui

```
ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(
      sliderInput(inputId = "mu",
                  label = "Mean for distribution:",
                  min = 1,max = 100, value = 50),
      sliderInput(inputId = "sd",
                  label = "Standard deviation for distribution:",
                  min = 5,max = 50, value = 40)),
    mainPanel(
      plotOutput("histogram"),
      textOutput("text_median"))
  )
)
```

Use past() to add other text elements

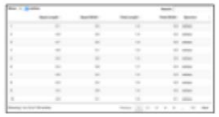
```
server <- function(input, output){
  rn <- reactive({rnorm(n=10000, mean=input$mu, sd=input$sd)})

  output$histogram <- renderPlot({
    ggplot()+geom_histogram(aes(x=rn()), bins=50)
  })

  output$text_median <- renderText({
    ################################
    paste("Median is", median(rn()))
    ################################
  })
}
```

Use html headers (h1, h2, h3, etc) to change the size of the text

```
ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(
      sliderInput(inputId = "mu",
                  label = "Mean for distribution:",
                  min = 1,max = 100, value = 50),
      sliderInput(inputId = "sd",
                  label = "Standard deviation for distribution:",
                  min = 5,max = 50, value = 40)),
    mainPanel(
      plotOutput("histogram"),
      ##########################
      h3(textOutput("text_median")))
      ##########################
  )
)
```

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

works with

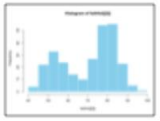


DT::**renderDataTable**(expr, options, callback, escape, env, quoted)

ataTableOutput(outputId, icon, …)

renderImage(expr, env, quoted, deleteFile)

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

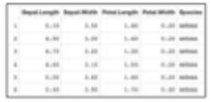


renderPlot(expr, width, height, res, …, env, quoted, func)

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

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)

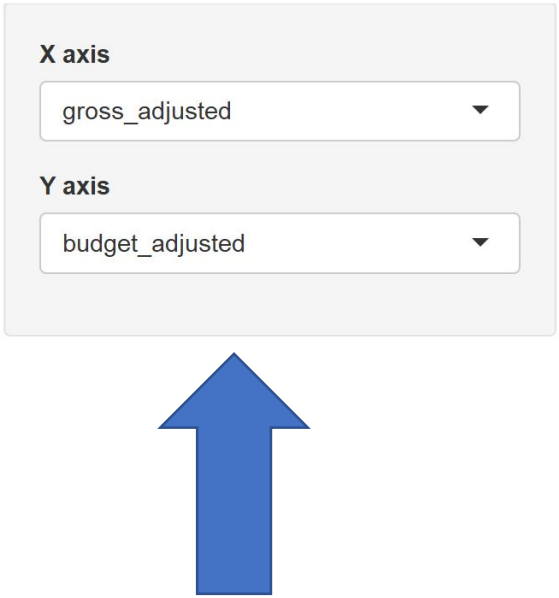
&

uiOutput(outputId, inline, container, …)

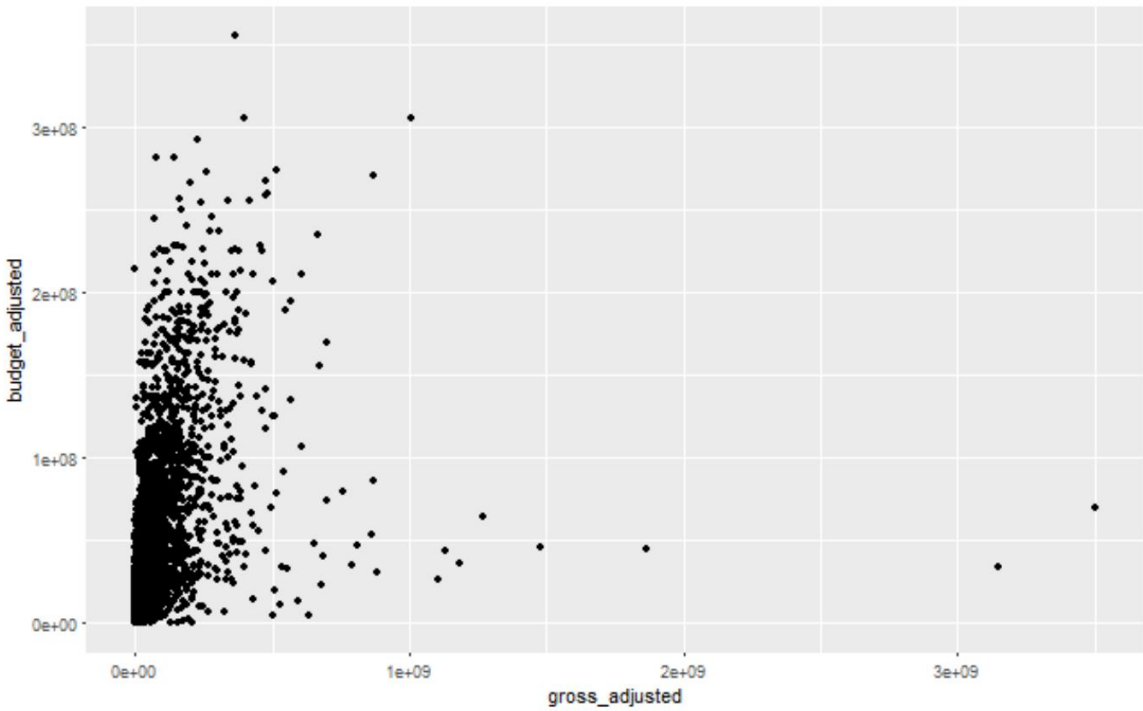
htmlOutput(outputId, inline, container, …)

The app we need

chart



selectInput



Look at the structure of selectInput

```
movies <- read.csv("movies3.csv", stringsAsFactors = F)
movies_num <- movies[,sapply(movies, is.numeric)]

ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(
      selectInput(inputId = "first", label="X axis",
                  choices = colnames(movies_num),
                  selected = "gross_adjusted"),
      selectInput(inputId = "second", label="Y axis",
                  choices = colnames(movies_num),
                  selected = "budget_adjusted")
                   ),
    mainPanel(
        plotOutput("scatterplot")
                   )
  )
    )
```

- aes_string is used when you have quoted names (“gross_adjusted” and not gross_adjusted)
- the input$ format from selectInput is a quoted text

```
server <- function(input, output){
  output$scatterplot <- renderPlot({
    ggplot(data=movies_num, aes_string(x=input$first, y=input$second))+geom_point()
  })
}
```

Adding some more functionality
alpha, the parameter for point transparency

```
ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(
      selectInput(inputId = "first", label="X axis",
                  choices = colnames(movies_num),
                  selected = "gross_adjusted"),
      selectInput(inputId = "second", label="Y axis",
                  choices = colnames(movies_num),
                  selected = "budget_adjusted"),
      ########################################
      sliderInput(inputId = "alpha",
                  label = "Transparency:",
                  min = 0,max = 1, value = 0.5)),
      ########################################
    mainPanel(
        plotOutput("scatterplot")
                 )
  )
    )
```

Adding data table

```
server <- function(input, output){
  output$scatterplot <- renderPlot({
    ggplot(data=movies_num, aes_string(x=input$first, y=input$second))+
      geom_point(alpha=input$alpha)
  })
  ##############################################
  output$summary <- renderDataTable({
    movies_num[,c(input$first, input$second)]
    })
  ##############################################
}
```

UI

```
ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(
      selectInput(inputId = "first", label="X axis",
                  choices = colnames(movies_num),
                  selected = "gross_adjusted"),
      selectInput(inputId = "second", label="Y axis",
                  choices = colnames(movies_num),
                  selected = "budget_adjusted"),
      sliderInput(inputId = "alpha",
                  label = "Transparency:",
                  min = 0,max = 1, value = 0.5)),
    mainPanel(
        plotOutput("scatterplot"),
        ###########################
        dataTableOutput("summary"))
        ###########################
  )
    )
```

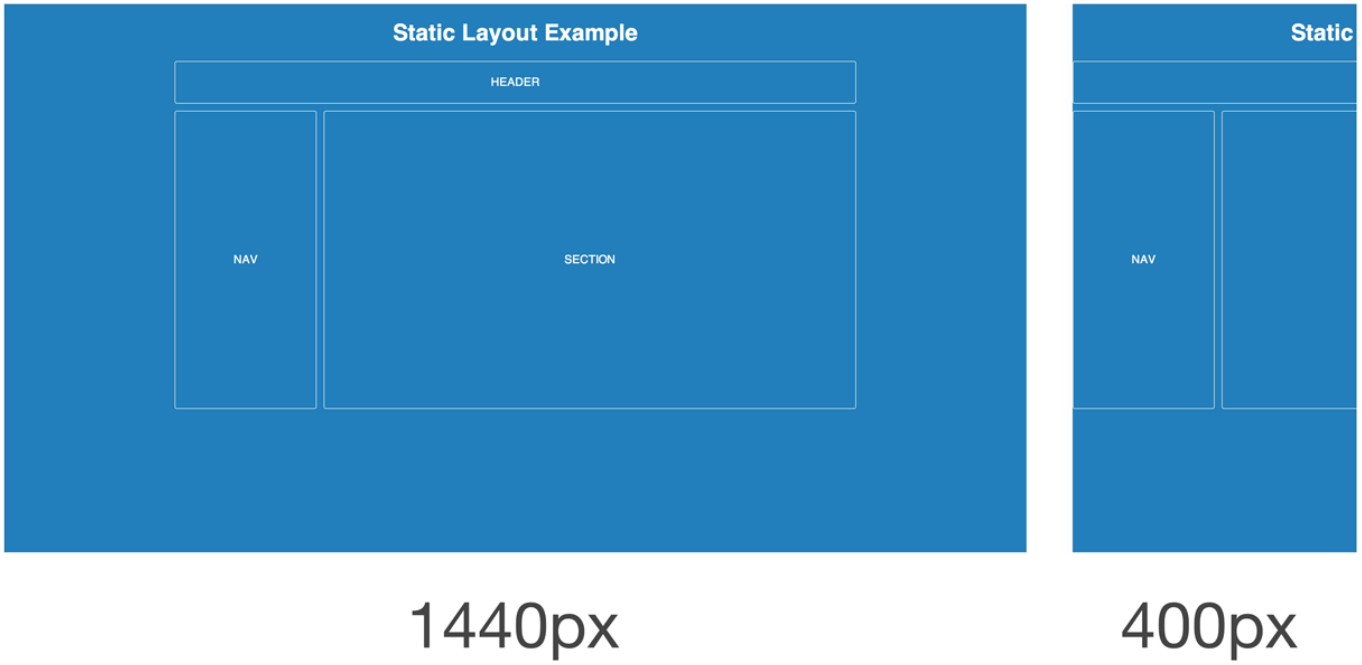
ui Layout

- A static page layout (sometimes called a “fixed” layout or “fixed width” layout) uses a preset page size and does not change based on the browser width. In other words, the page layout might have a permanent width of 960 pixels no matter what.

- Different devices will treat a static page layout in various ways, so the rendered page could be slightly unpredictable.

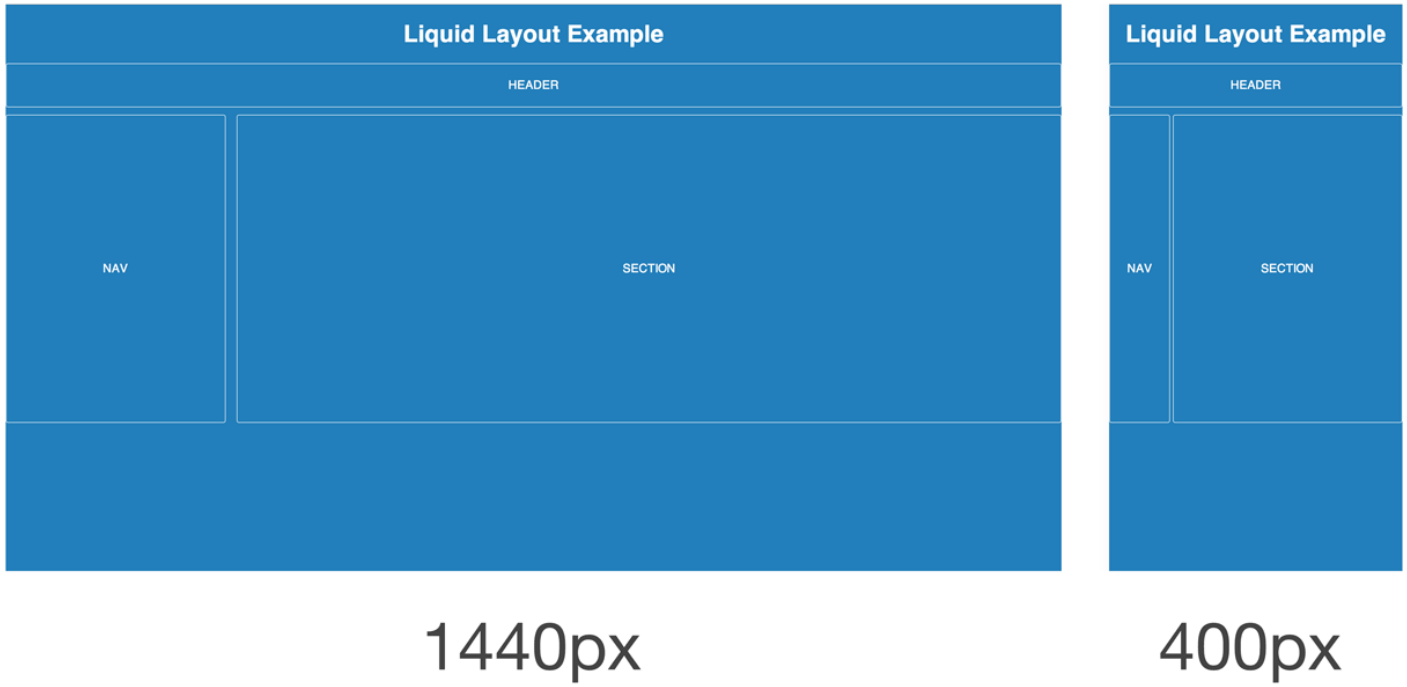
- For example, on a desktop browser, if the window is too small horizontally, then the page will be cut off and horizontal scroll bars will be displayed.

- fixedPage()

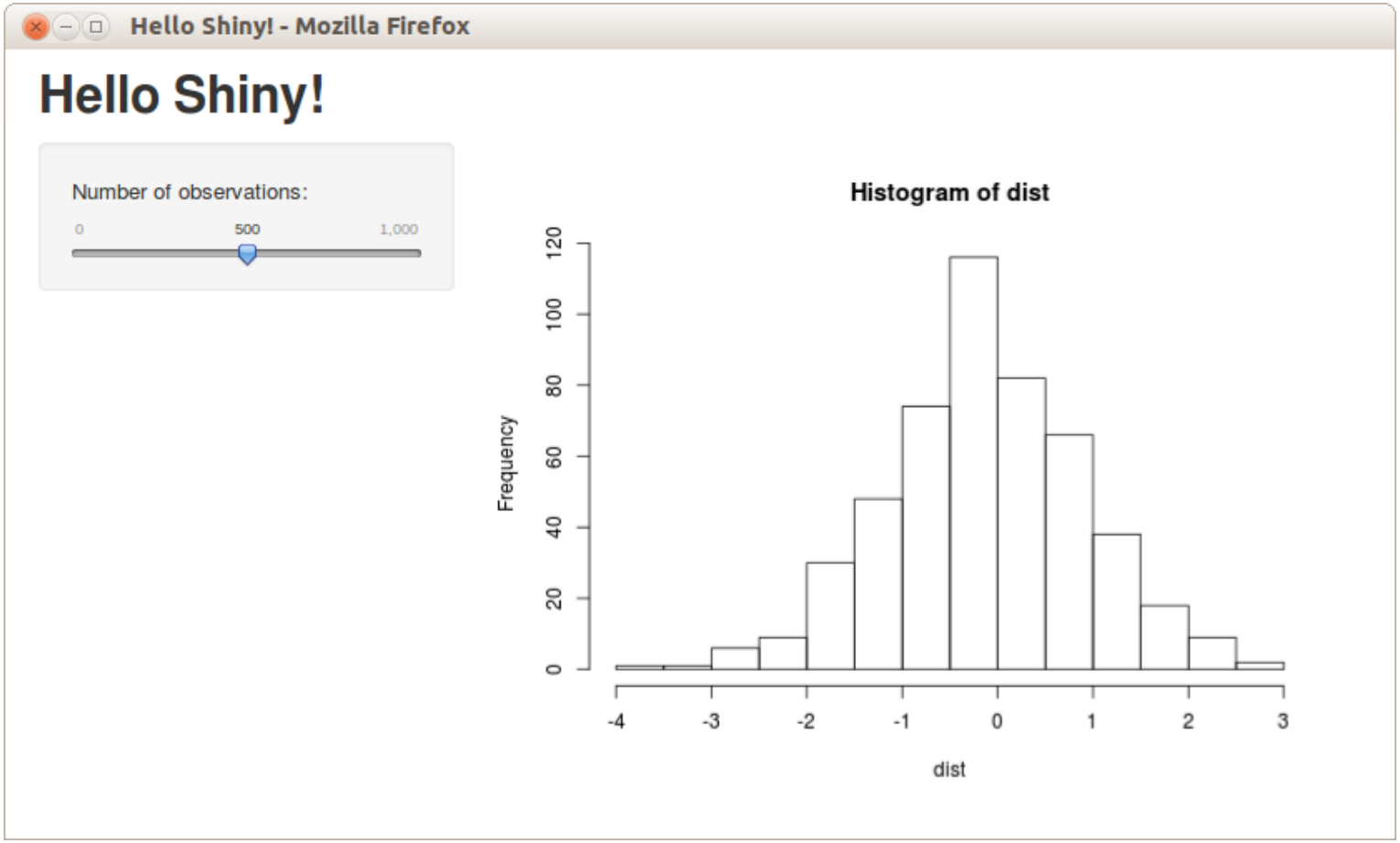


- A liquid page layout (sometimes called “fluid” or “fluid width”) uses relative units instead of fixed units.

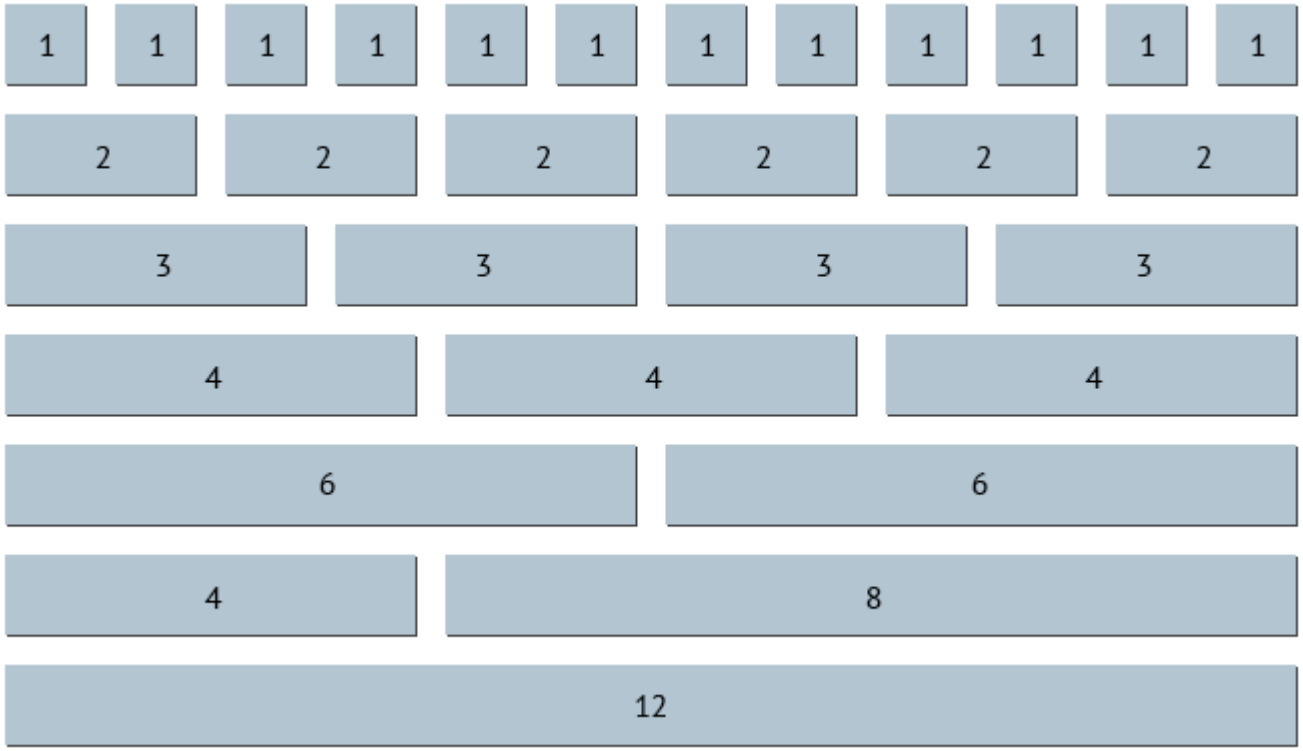
- Typically a liquid layout will use percentages instead of pixels, but any relative unit of measurement will work, such as ems.
- A liquid layout often will fill the width of the page, no matter what the width of the browser might be.
- **fluidPage()** in shiny



Sidebar layout:
This layout provides a sidebar for inputs and a large main area for output:



- Grid Layout
- The familiar sidebarLayout() described above makes use of Shiny's lower-level grid layout functions.
- Rows are created by the fluidRow() function and include columns defined by the column() function.
- Column widths are based on the Bootstrap 12-wide grid system, so should add up to 12 within a fluidRow() container.



The same server

```
server <- function(input, output){
  output$scatterplot <- renderPlot({
    ggplot(data=movies_num, aes_string(x=input$first, y=input$second))+
      geom_point(alpha=input$alpha)
  })
  output$summary <- renderDataTable({
    movies_num[,c(input$first, input$second)]
  })
}
```

The ui

```
ui <- fluidPage(
  fluidRow(
    column(1, selectInput(inputId = "first", label="X axis",
                          choices = colnames(movies_num),
                          selected = "gross_adjusted")),
    column(1, selectInput(inputId = "second", label="Y axis",
                          choices = colnames(movies_num),
                          selected = "budget_adjusted")),
    column(1, sliderInput(inputId = "alpha",
                          label = "Transparency:",
                          min = 0,max = 1, value = 0.5))
          ),
  fluidRow(
    column(4, plotOutput("scatterplot")),
    column(4, dataTableOutput("summary"))
  )
  )
```

Two rows

```
ui <- fluidPage(
  fluidRow(
    column(1, selectInput(inputId = "first", label="X axis",
                          choices = colnames(movies_num),
                          selected = "gross_adjusted")),
    column(1, selectInput(inputId = "second", label="Y axis",
                          choices = colnames(movies_num),
                          selected = "budget_adjusted")),
    column(1, sliderInput(inputId = "alpha",
                          label = "Transparency:",
                          min = 0,max = 1, value = 0.5))
  ),
  fluidRow(
    column(4, plotOutput("scatterplot")),
    column(4, dataTableOutput("summary"))
  )
  )
```

The comma you don't want to miss

Columns inside the rows

```
ui <- fluidPage(
  fluidRow(
    column(1, selectInput(inputId = "first", label="X axis",
                          choices = colnames(movies_num),
                          selected = "gross_adjusted")),
    column(1, selectInput(inputId = "second", label="Y axis",
                          choices = colnames(movies_num),
                          selected = "budget_adjusted")),
    column(1, sliderInput(inputId = "alpha",
                          label = "Transparency:",
                          min = 0,max = 1, value = 0.5))
          ),
  fluidRow(
    column(4, plotOutput("scatterplot")),
    column(4, dataTableOutput("summary"))
  )
  )
```

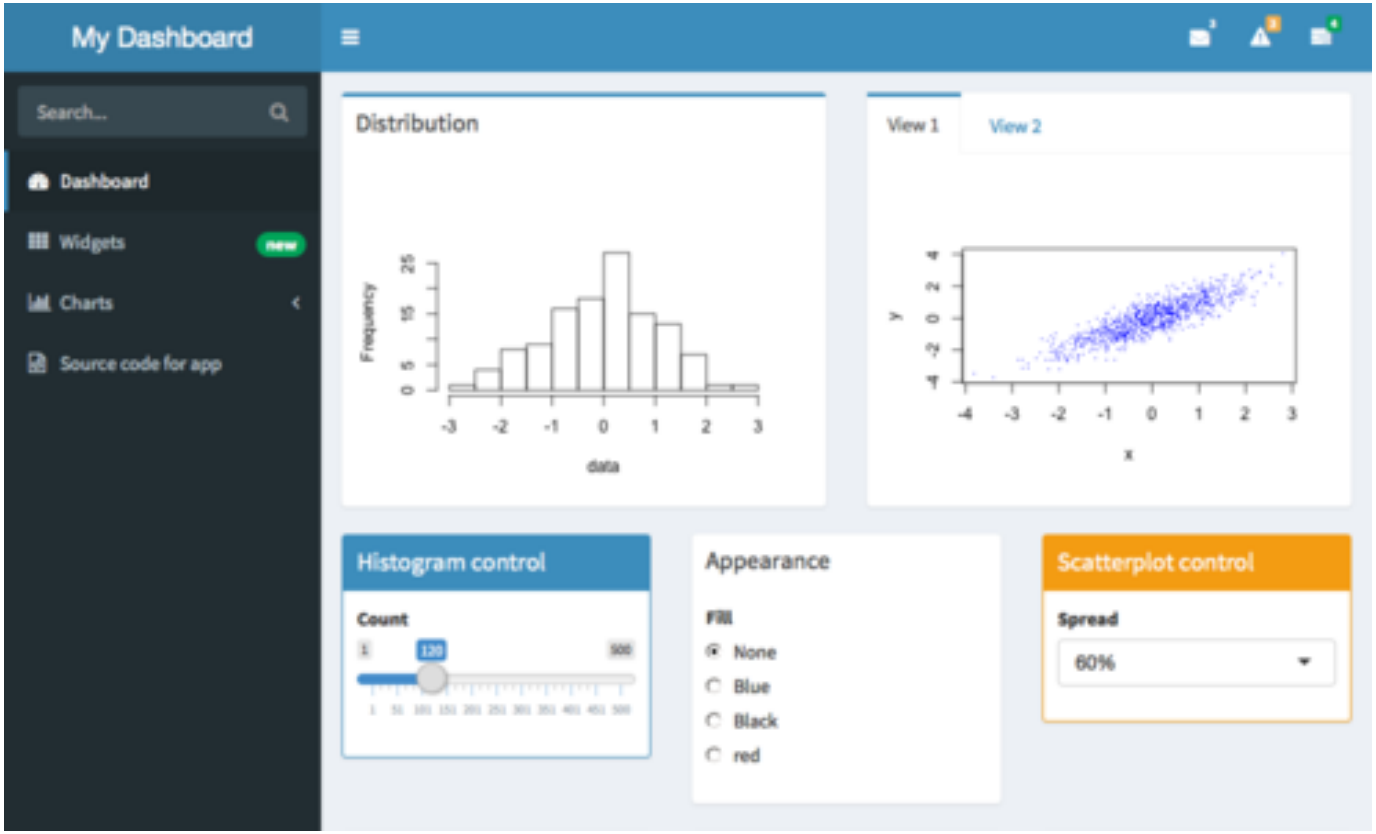
Width of the column

The sum of widths of the columns inside every row needs to be no more then 12

```
ui <- fluidPage(
  fluidRow(
    column(1, selectInput(inputId = "first", label="X axis",
                          choices = colnames(movies_num),
                          selected = "gross_adjusted")),
    column(1, selectInput(inputId = "second", label="Y axis",
                          choices = colnames(movies_num),
                          selected = "budget_adjusted")),
    column(1, sliderInput(inputId = "alpha",
                          label = "Transparency:",
                          min = 0,max = 1, value = 0.5))
          ),
  fluidRow(
    column(4, plotOutput("scatterplot")),
    column(4, dataTableOutput("summary"))
  )
  )
```

shinydashboard

Shiny dashboard makes it easy to construct dashboards
install.packages("shinydashboard")



A dashboard has three parts: a header, a sidebar, and a body. Here's the most minimal possible UI for a dashboard page.

```
library(shinydashboard)

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

Add title to the dashboard

```
ui <- dashboardPage(
  dashboardHeader(title ="This is my dashboard"),
  dashboardSidebar(),
  dashboardBody()
)

server <- function(input, output) { }

shinyApp(ui, server)
```

App with the histogram

```
ui <- dashboardPage(
  dashboardHeader(title ="This is my dashboard"),

  dashboardSidebar(sliderInput(inputId = "mu",
                               label = "Mean for distribution:",
                               min = 1,max = 100, value = 50),
                   sliderInput(inputId = "sd",
                               label = "Standard deviation for distribution:",
                               min = 5,max = 50, value = 40),
                   numericInput(inputId="sample",
                                label = "Sample size",
                                min=100, max=10000, value=1000)),

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

App user inputs are located in dashboardSidebar

```
ui <- dashboardPage(
  dashboardHeader(title ="This is my dashboard"),

  dashboardSidebar(sliderInput(inputId = "mu",
                               label = "Mean for distribution:",
                               min = 1,max = 100, value = 50),
                   sliderInput(inputId = "sd",
                               label = "Standard deviation for distribution:",
                               min = 5,max = 50, value = 40),
                   numericInput(inputId="sample",
                                label = "Sample size",
                                min=100, max=10000, value=1000)),

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

Output is in dashboardBody

```
ui <- dashboardPage(
  dashboardHeader(title ="This is my dashboard"),

  dashboardSidebar(sliderInput(inputId = "mu",
                               label = "Mean for distribution:",
                               min = 1,max = 100, value = 50),
                   sliderInput(inputId = "sd",
                               label = "Standard deviation for distribution:",
                               min = 5,max = 50, value = 40),
                   numericInput(inputId="sample",
                                label = "Sample size",
                                min=100, max=10000, value=1000)),

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

using shinyapps.io

Building web applications with shiny

- Allows to deploy shiny apps on the web
- integrated with R studio
- Is free, but limited to 5 active applications and 25 Active hours
- You can update your plan on shinyapps.io
- Or build something by yourself using shinyproxy

Register with shinyapps.io

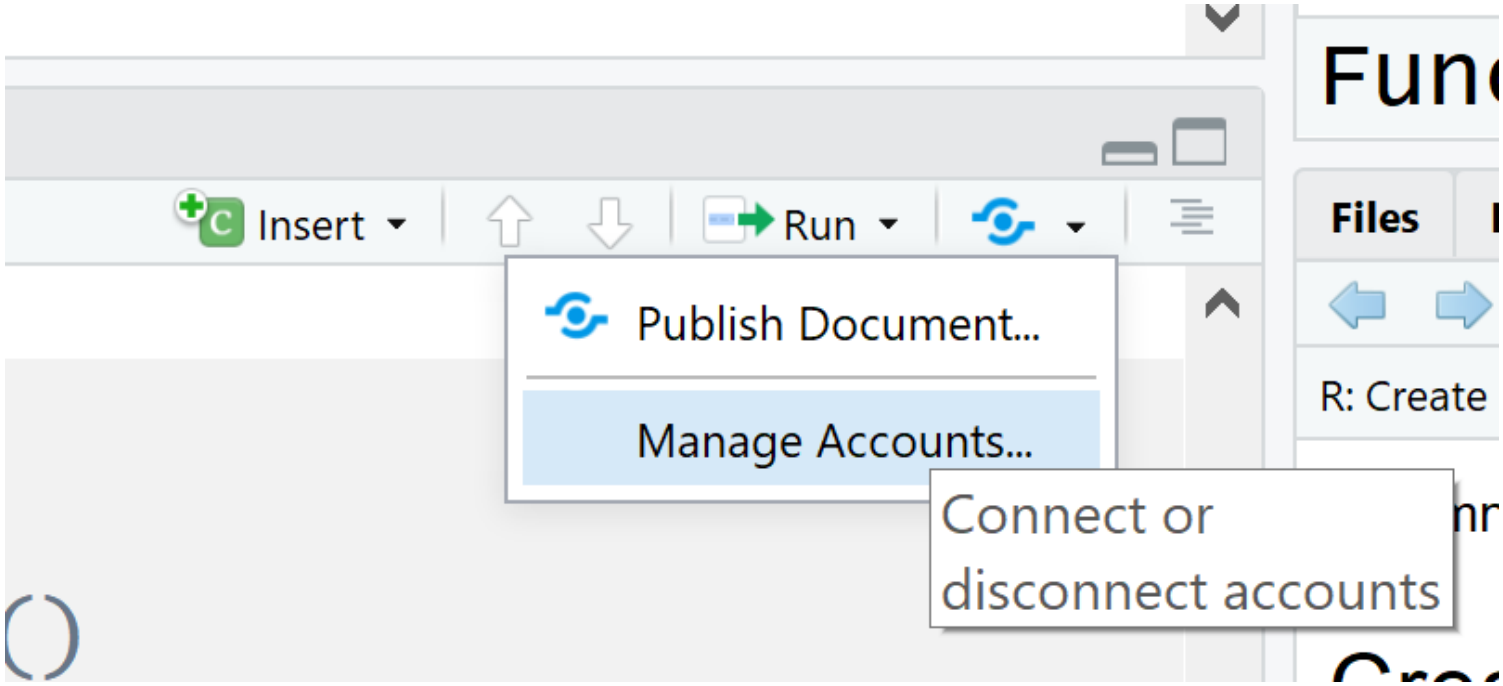
Install the following packages

```
install.packages('rsconnect')
```

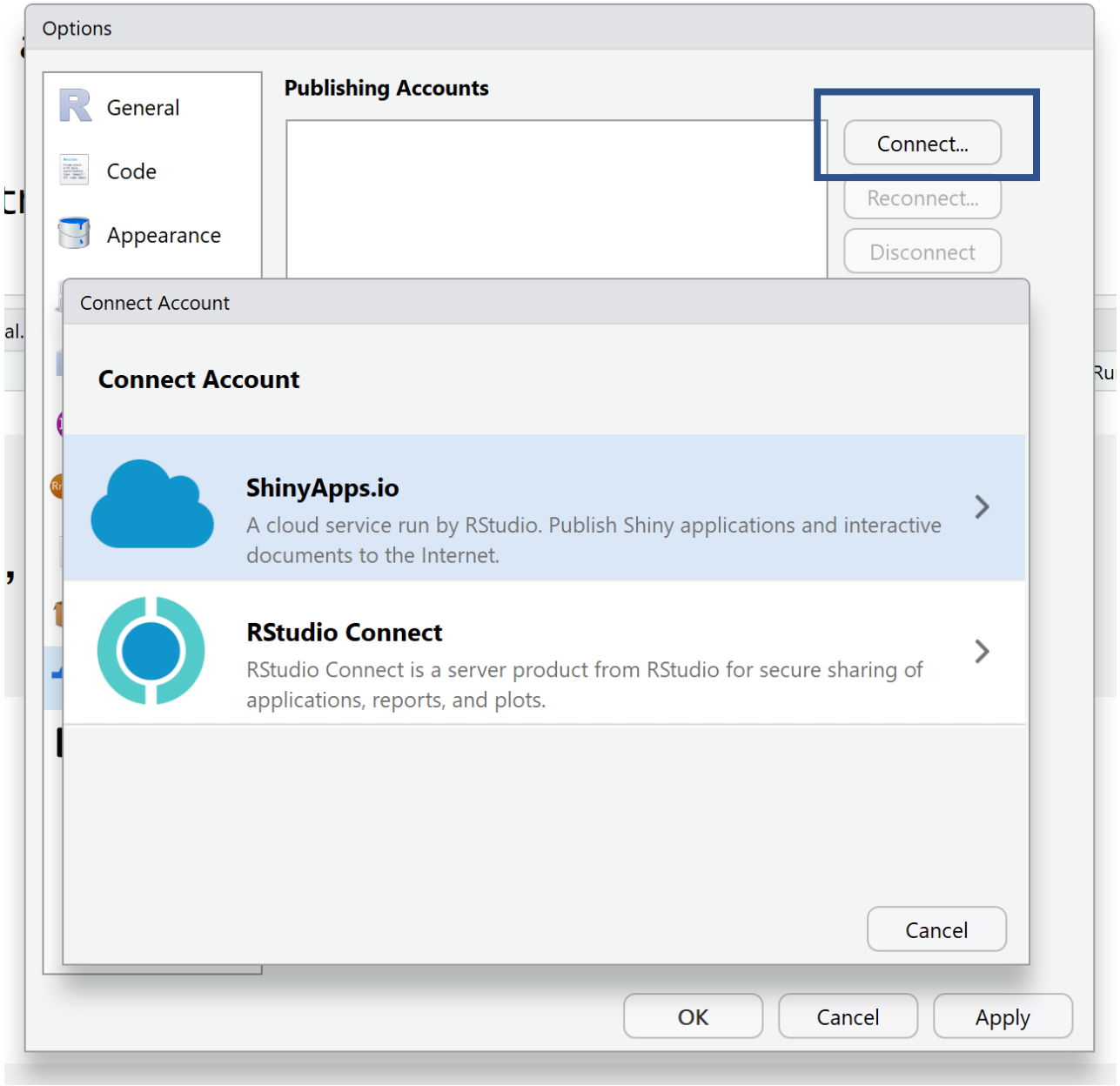
Login into your page on shinyapps.io

Copy token and secret

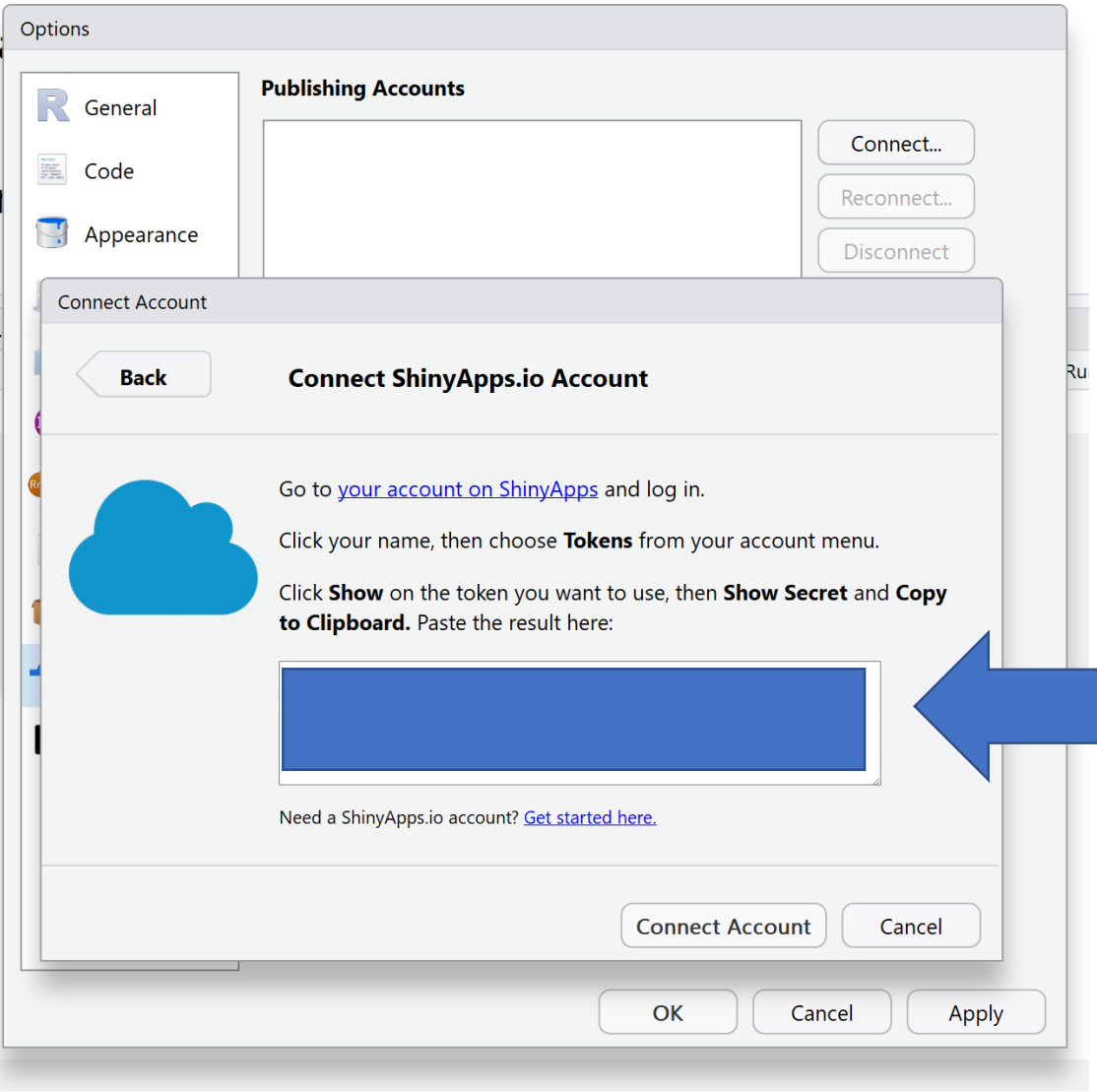
Set up your account



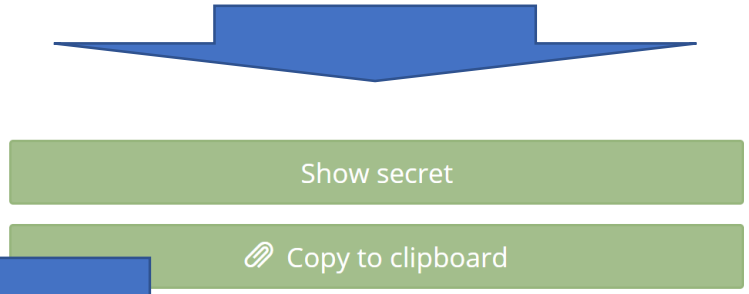
Building web applications with shiny



Building web applications with shiny



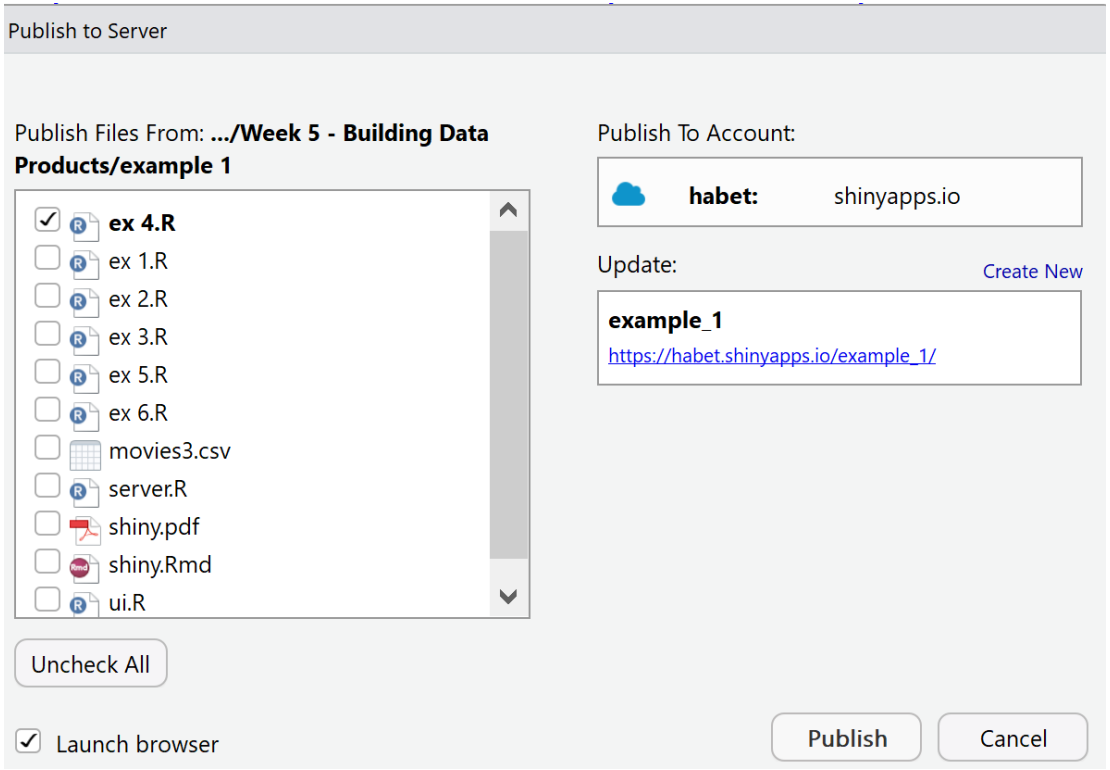
In your shinyapps.io account dashboard



Ready to publish

chose the files to upload

- can be a single file with ui and server
- separate files named ui and server



If planning to use a library include it in the R script

- no need to install, just call it

ui.R | ex 1.R | ex 2.R | ex 3.R | ex 4.R | ex 5.R*

```
1 library(ggplot2)
2
3 ui <- fluidPage(
4   sidebarLayout(
5     sidebarPanel(
6       sliderInput(inputId =
7                   label = "M
8                   min = 1,ma
9       sliderInput(inputId =
```

- if planning to use an external file, read it
- Don't forget to upload the data file into server too

```
movies <- read.csv("movies3.csv", stringsAsFactors = F)
movies_num <- movies[,sapply(movies, is.numeric)]
library(ggplot2)


ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(
      selectInput(inputId = "first", label="X axis",
                  choices = colnames(movies_num),
```

Other important/fun stuff

- plotly allows to make interactive graphs
- ggViz – interactive graphs with similar logic as ggplot
- shiny apps gallery