Shiny app for beginners

Xijin Ge @SDSU

Get ready:

- Login to Posit.cloud (Or start RStudio locally, update if needed)
- install.packages("shiny")
- Login to Shinyapps.io for publishing apps
- Copy files from https://BIT.LY/SDSUshiny

- Basic concepts
- Activities
- Strategies

Everything in R is an object.

Data objects

R data types

- character
- numeric
- integer
- logical

data structures

- vectors
- list
- factor
- matrix
- data frame

If it is not a data object, it must be a **function**.

```
y <- sqrt(10^5 - 4.3)
```

A function can have many input parameters

```
plot(
   x = 1:20,
   y = rnorm(20),
   main = ”Just a plot”,
   las = 3,
   xlab = ”x value”,
   ylab = ”y value”
)
```

Define your function

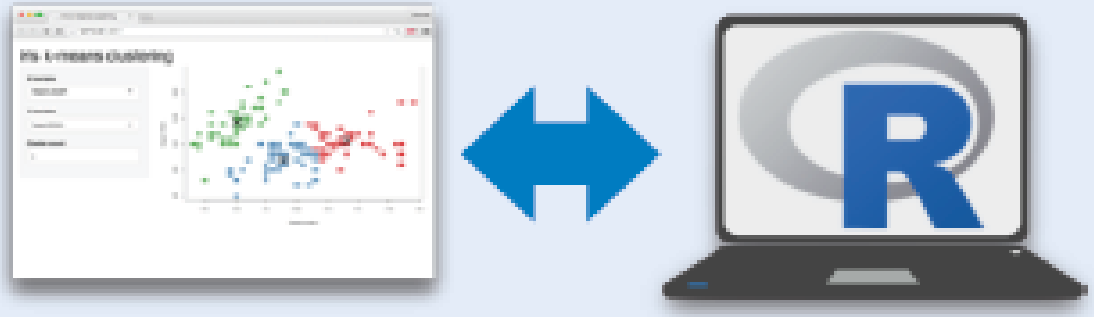
```
square <- function(x){
  x^2
}
```

Functions can take functions as input

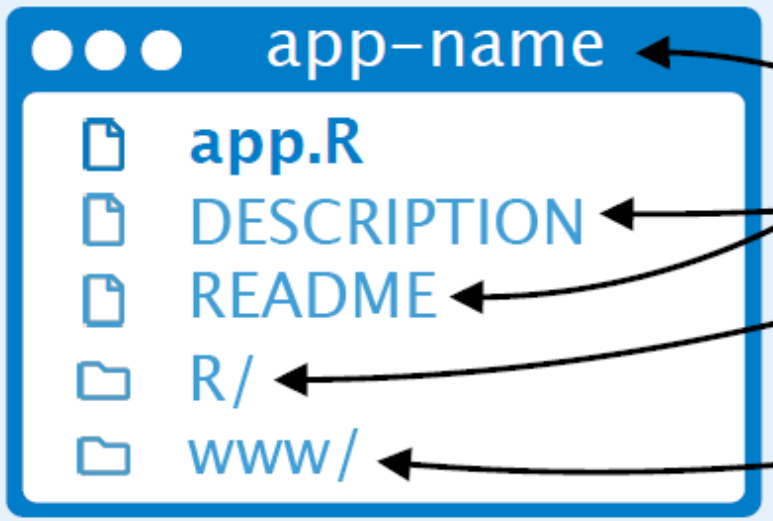
```
plots <- function(fn){
  x <- 1:50
  plot(x, fn(x))
}
plots(square)
```

Building an App

A **Shiny** app is a web page (**ui**) connected to a computer running a live R session (**server**).



Users can manipulate the UI, which will cause the server to update the UI's displays (by running R code).



The directory name is the app name

(optional) used in showcase mode

(optional) directory of supplemental .R files that are sourced automatically, must be named "**R**"

(optional) directory of files to share with web browsers (images, CSS, .js, etc.), must be named "**www**"

Launch apps stored in a directory with **runApp(**<path to directory>**)**.

```
library(shiny)
library(bslib)
ui <- page_fluid()

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

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

User interface
controls the layout and appearance of app

Server function
contains instructions needed to build app

`shinyApp()`
Creates the Shiny app object

```
library(shiny)
library(bslib)
library("movies.Rdata")
ui <- page_fluid()

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



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

Data used for this app

To generate the template, type **shinyapp** and press **Tab** in the RStudio IDE or go to **File > New Project > New Directory > Shiny Application**

```
# app.R
library(shiny)

ui <- fluidPage(
  numericInput(inputId = "n",
    "Sample size", value = 25),
  plotOutput(outputId = "hist")
)


server <- function(input, output, session) {
  output$hist <- renderPlot({
    hist(rnorm(input$n))
  })
}


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

In **ui** nest R functions to build an HTML interface

Customize the UI with **Layout Functions**

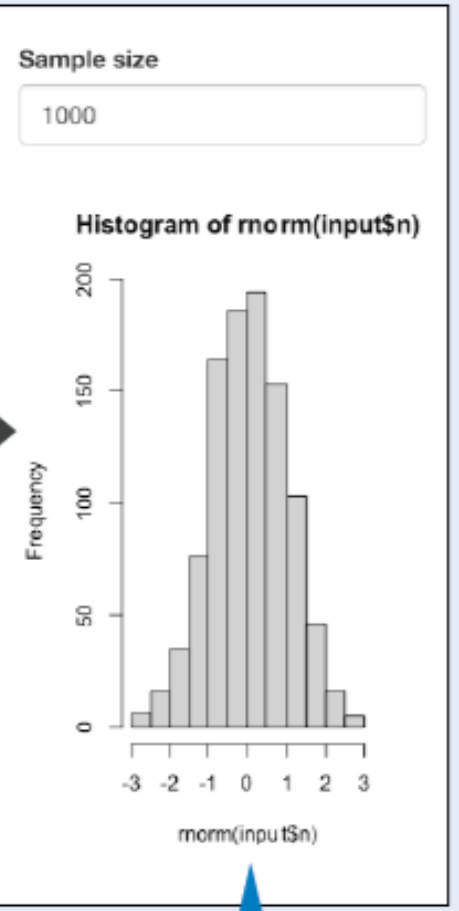
Add Inputs with ***Input()** functions

Add Outputs with ***Output()** functions

Tell the **server** how to render outputs and respond to inputs with R

Wrap code in **render*()** functions before saving to output

Refer to UI inputs with **input$<id>** and outputs with **output$<id>**



Call **shinyApp()** to combine **ui** and **server** into an interactive app!

sidebar

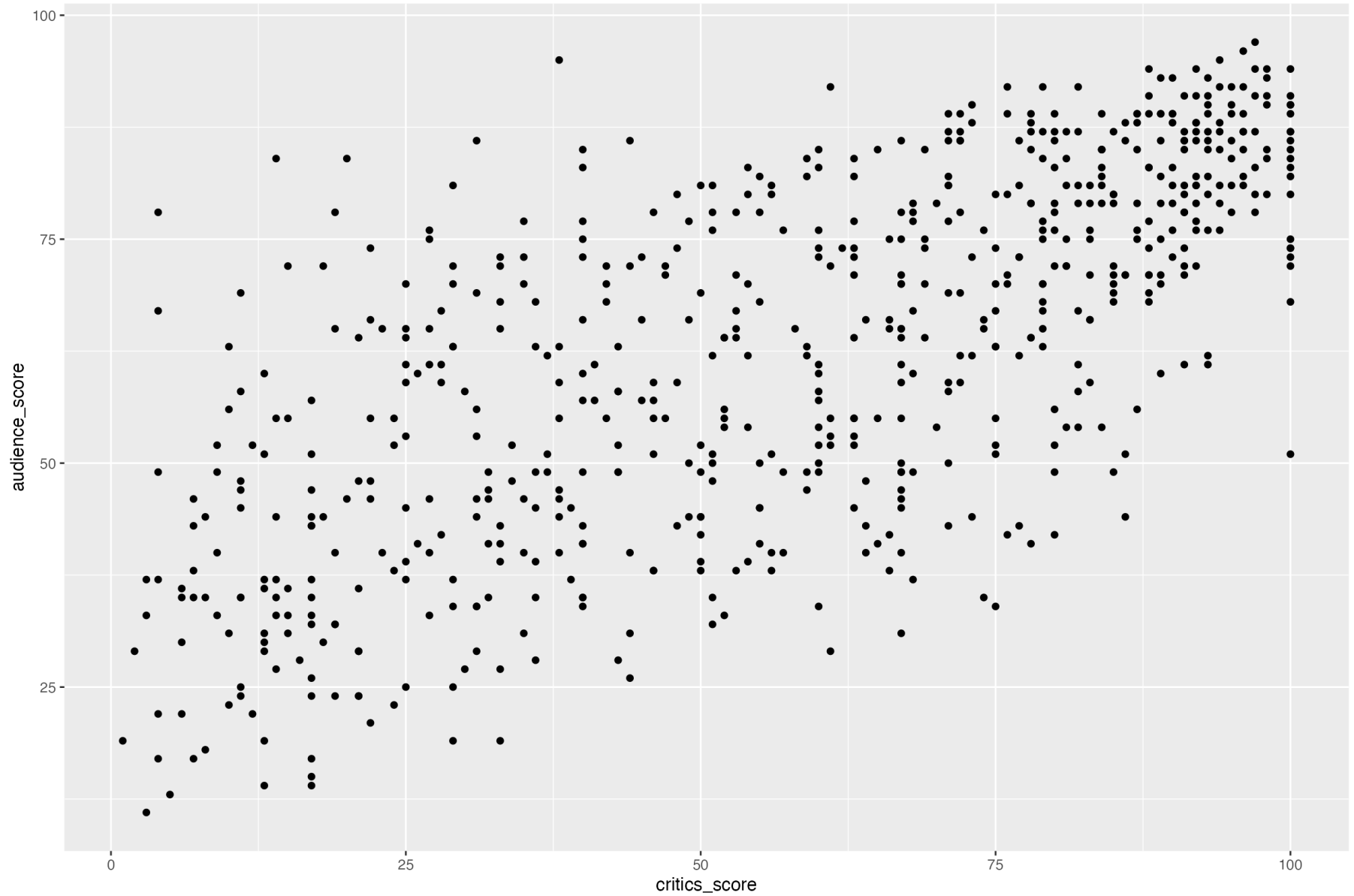
Y-axis:

audience_score

X-axis:

critics_score

main content panel



```r
# Load packages ------------------------------------------

library(shiny)
library(bslib)
library(ggplot2)

# Load data ----------------------------------------------

load("movies.RData")
```

```r
# Define UI for application that plots features of movies
ui <- page_sidebar(

  sidebar = sidebar(

    # Select variable for y-axis
    selectInput(inputId = "y",
                label = "Y-axis:",
                choices = c("imdb_rating", "imdb_num_votes", "critics_score", "audience_score", "runtime"),
                selected = "audience_score"),

    # Select variable for x-axis
    selectInput(inputId = "x",
                label = "X-axis:",
                choices = c("imdb_rating", "imdb_num_votes", "critics_score", "audience_score", "runtime"),
                selected = "critics_score"),
  ),

  # Output: Show scatterplot
  card(
    plotOutput(outputId = "scatterplot")
  )
)
```

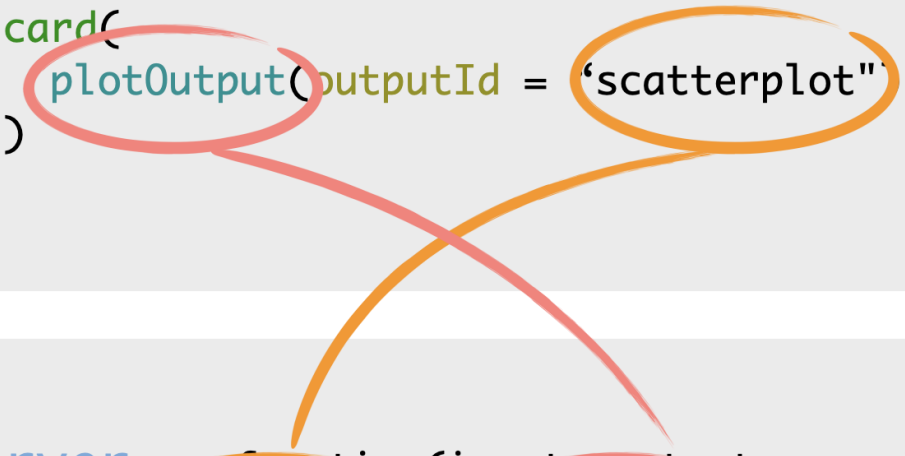
Add a card containing **output** elements to the main content area. Output elements get created in the server function.

```
# Define UI for application that plots features of movies
ui <- page_sidebar(

  . . .

    # Select variable for y-axis
    selectInput(inputId = "y",
                label = "Y-axis:",
                choices = c("imdb_rating", "imdb_num_votes", "critics_score", "audience_score", "runtime"),
                selected = "audience_score"),

    # Select variable for x-axis
    selectInput(inputId = "x",
                label = "X-axis:",
                choices = c("imdb_rating", "imdb_num_votes", "critics_score", "audience_score", "runtime"),
                selected = "critics_score"),

  . . .
)
```

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

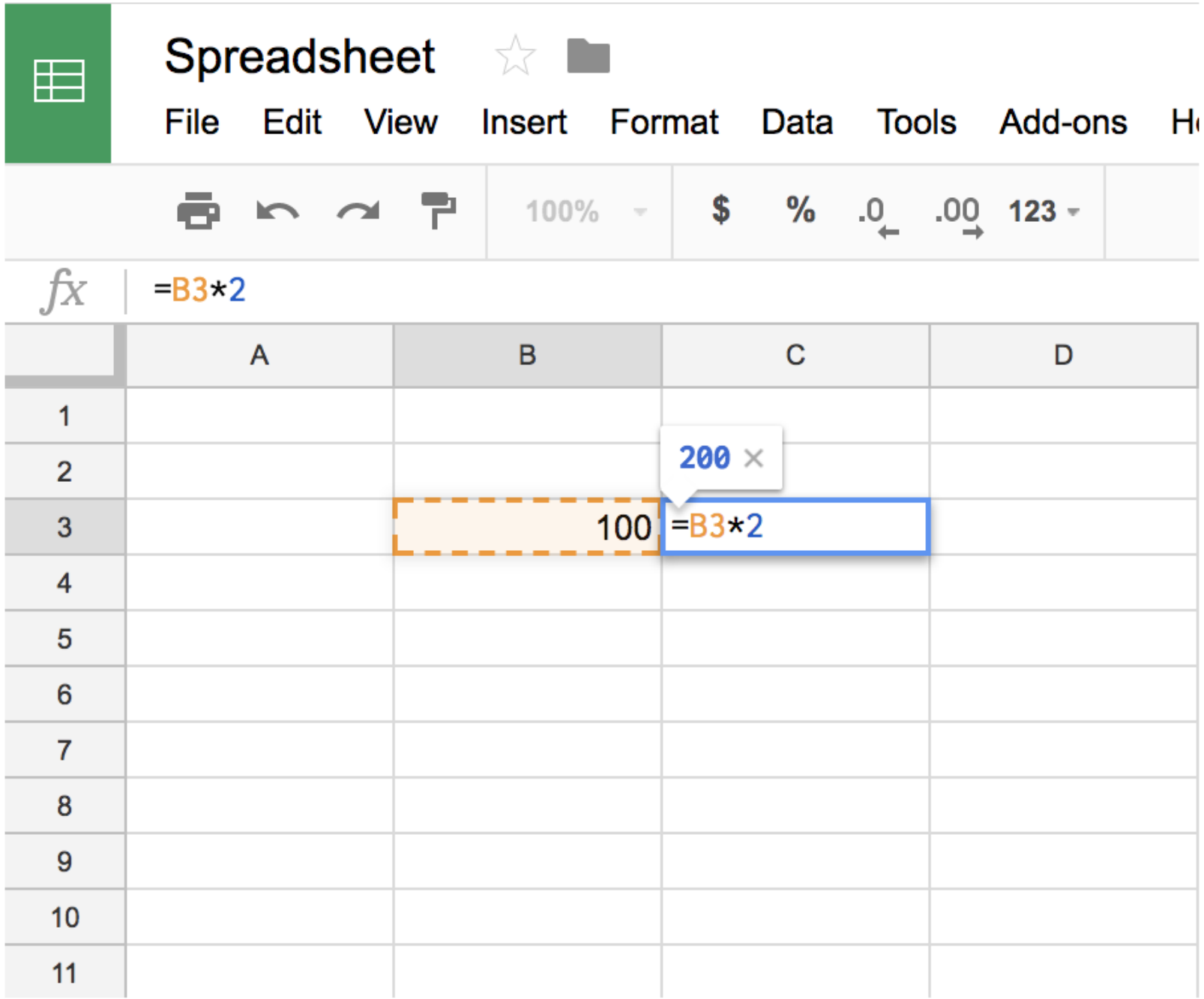
```
# Define UI for application that plots features of movies
ui <- page_sidebar(

  . . .

  # Output: Show scatterplot
  card(
    plotOutput(outputId = "scatterplot")
  )

)
```

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

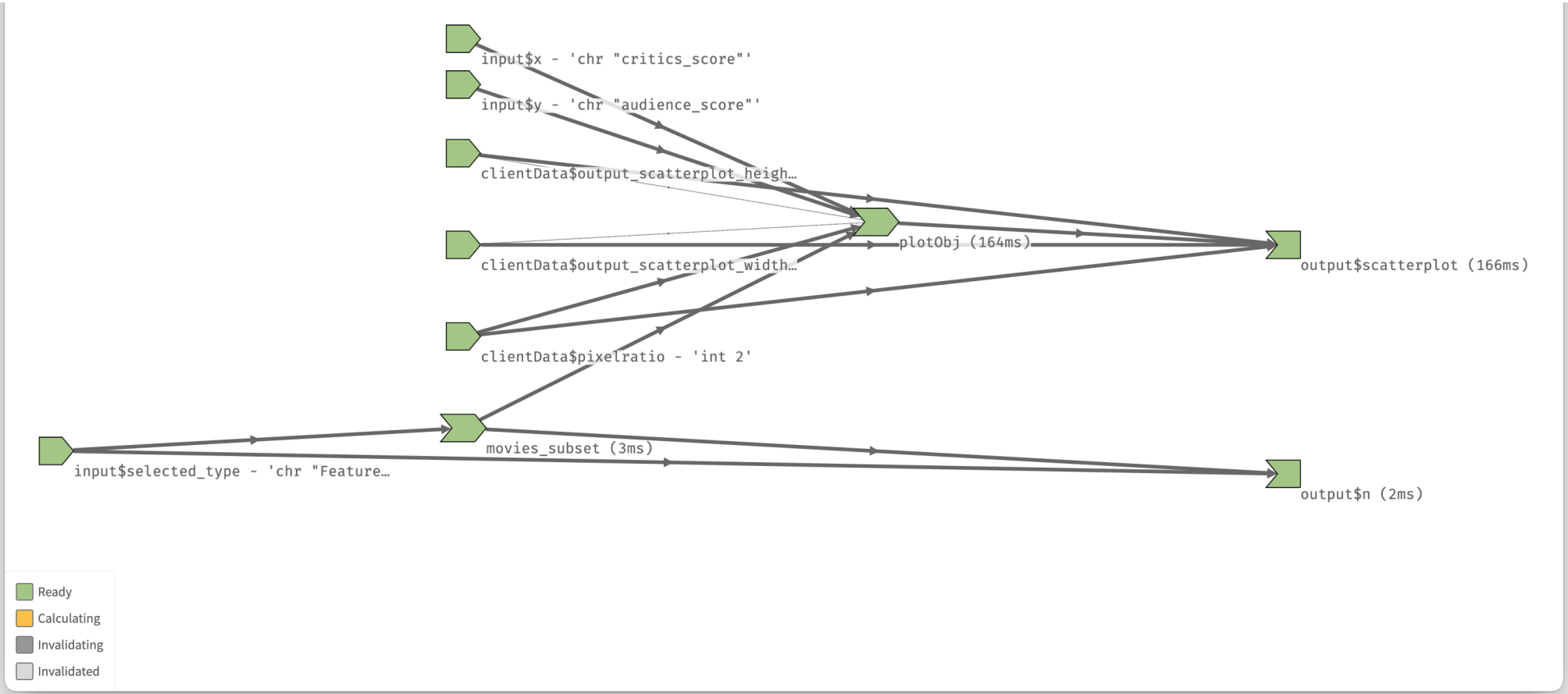



input$x
expression()
output$y
Create your
own reactive values
*Input()
Update
Render
reactive output
render*()

Reactivity


Spreadsheet
File Edit View Insert Format Data Tools Add-ons H
100% $ % .0 .00 123
fx =B3*2
A B C D
1
2
3 100 =B3*2
4
5
6
7
8
9
10
11
200

Reactivity logic can be complex: Reactlog



Practical considerations

- Start from an example. Modify to suit your needs.
- Add UI elements first. Test then change the server logic.
- Use the browser() function to debug.
- Generate plots in another window. Transpl

Shiny apps are for others to use

- Curse of knowledge
- Assume others are stupid and lazy

To learn more

- Shiny lessons: https://shiny.posit.co/
- Mastering Shiny book: https://mastering-shiny.org/