Slides for Day 2

Statistics with R

A fast route to Data Science

Montserrat Guillen

Dept. Econometrics, Statistics and Applied Economics & Riskcenter, UB

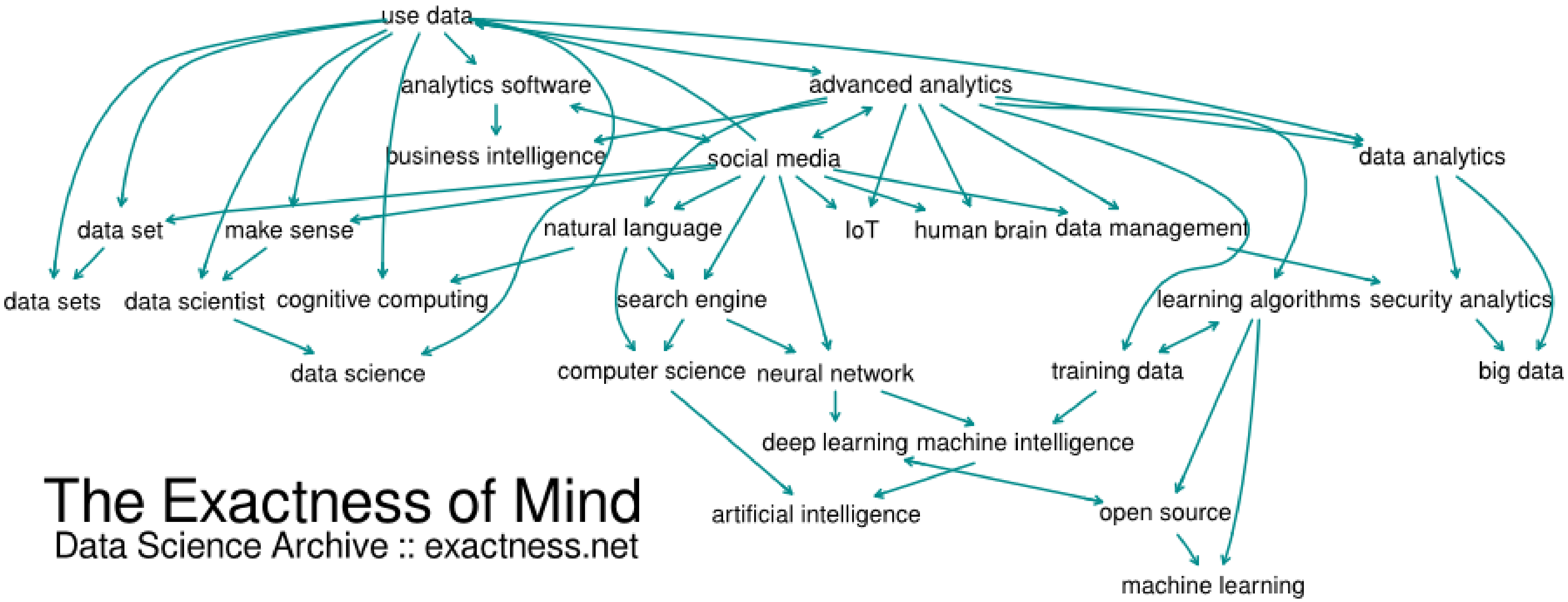
Riskcenter

Universitat de Barcelona

What did we do?

- **RStudio projects, working directory, scripts and packages**
- **Data structures:** **vectors, matrices, data.frames, lists, objects**
- **Data wrangling:** **injestion+digestion**
- **R programming:** **if-the-else, loops, functions… a detour to graphics**
- **Rmarkdown:** **producing HTML, PDF, LateX, PPT**

What do we need?



The Exactness of Mind
Data Science Archive :: exactness.net

What will we do today?

- **Build a package and deal with Spark**
- **Introduction to Shiny**
- **A case study: Bank telemarketing**
  - **Practice exploratory data analysis (EDA):** what is in my data?
  - **Predictive modeling:** Is there noise or is there something else?
  - **Other fancy models out there:** decision tree model, random forests, support vector machine, Bayesian networks, neural networks.
  - **Prediction and cross-validation**

R packages

We will learn to:

- Use other people’s packages

  For example:

  ```
  install.package(“lmer4”)
  library(lmer4)
  ```

- Use our own packages
- Create our packages

Let’s create an R package

Doc-05.pdf

- Collect functions
- Create the package directory (easy if you install things before that or use RStudio)
- Document the functions
- Build process and install
- **Make the package a GitHub repository**

  or Contribute to CRAN
- **An example with our course**

R4DSUB.zip

Our R4DSUB package... **USAGE**

- Download R4DSUB.zip and place in a folder
- Open Rstudio and go to Working directory

> install.packages(“R4DSUB”) Does not WORK

  - From Packages (right down window) Mark install from .zip not from CRAN

> help(package= R4DSUB) #to see package desc

> help(PredictiveModel1) #to see function desc

Our own new package… **CREATION 1**

- We will start with
- Create a folder
- Open Rstudio
- Go to the folder
  - Session > Set Working directory
- Create New Project
  - File > New Project (choose in current directory)
- Open New Script window
  - File > New File > R Script

Our own new package… **CREATION 2**

- First we will create two functions:

```
FunPredictiveModel1<-function(mydataset,myformula){
  mod1<-glm(myformula, mydataset, family=binomial)
  return(mod1)
}
install.packages("randomForest")
library(randomForest)
FunPredictiveModel2<-function(mydataset, myregressors, myy){
  forest<-randomForest(as.factor(myy)~myregressors,mydataset,
importance=TRUE, ntree=100)
  return(forest)
}
```

- Check that they are created at the environment

Our own new package… **CREATION 3**

- Let us create R4DSUBv2
- In the Script window write

```
>   package.skeleton(name = "R4DSUBv2", path = ".", force = FALSE)
```

- You will see….

```
Creating directories ...
Creating DESCRIPTION ...
Creating NAMESPACE ...
Creating Read-and-delete-me ...
Saving functions and data ...
Making help files ...
Done.
Further steps are described in './R4DSUBv2/Read-and-delete-me'
```

Before CREATION is completed

- Edit DESCRIPTION FILE
- Edit HELP
    - You should go to Files (right lower window of Rstudio)
    - Click on the man folder
- We do now omit this step…. Long
    - Function 1 and write something in the title
    - Function 2 and write something in the title
    - Package name ans write something in examples

Our own new package… **CREATION 4**

- Build the package
  - Main Menu Build
  - Configure Build
    - (Project build tools choose package)
    - Package directory

      Write here **R4DSUBv2**
  - In the script, run

```
install.packages("pkgbuild“)
library(devtools)
library(pkgbuild)
```

R4DSUBv2.zip

  - In the Build menu Build binary package

R shiny

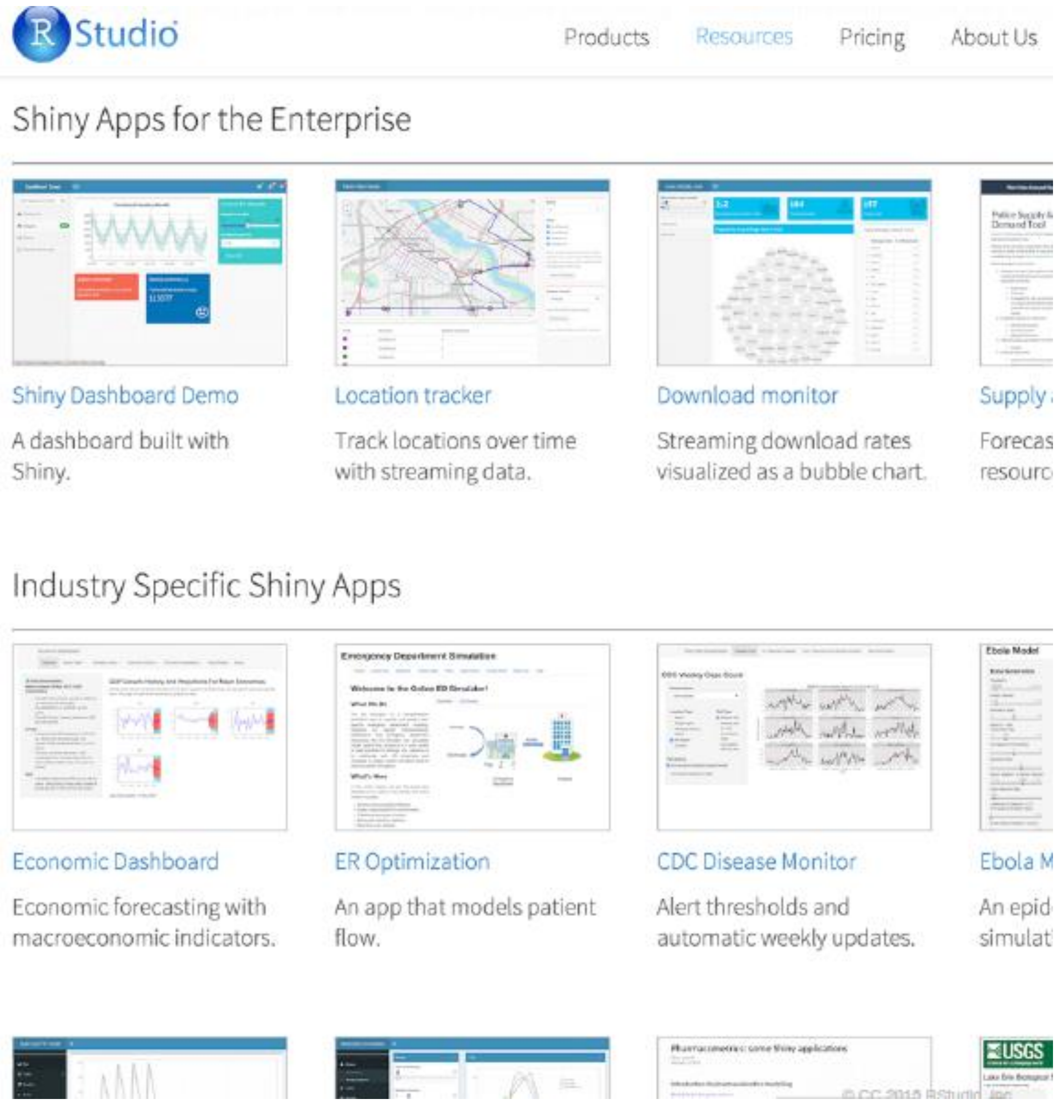
We will learn to:

- Understand Shiny as user
- Create a simple Shiny application

More code and more slides at:

bit.ly/shiny-quickstart-1
bit.ly/shiny-quickstart-2
bit.ly/shiny-quickstart-3

www.rstudio.com/products/shiny/shiny-user-showcase



Shiny Apps for the Enterprise

Shiny Dashboard Demo

A dashboard built with Shiny.

Location tracker

Track locations over time with streaming data.

Download monitor

Streaming download rates visualized as a bubble chart.

Supply

Forecas resourc

Industry Specific Shiny Apps

Economic Dashboard

Economic forecasting with macroeconomic indicators.

ER Optimization

An app that models patient flow.

CDC Disease Monitor

Alert thresholds and automatic weekly updates.

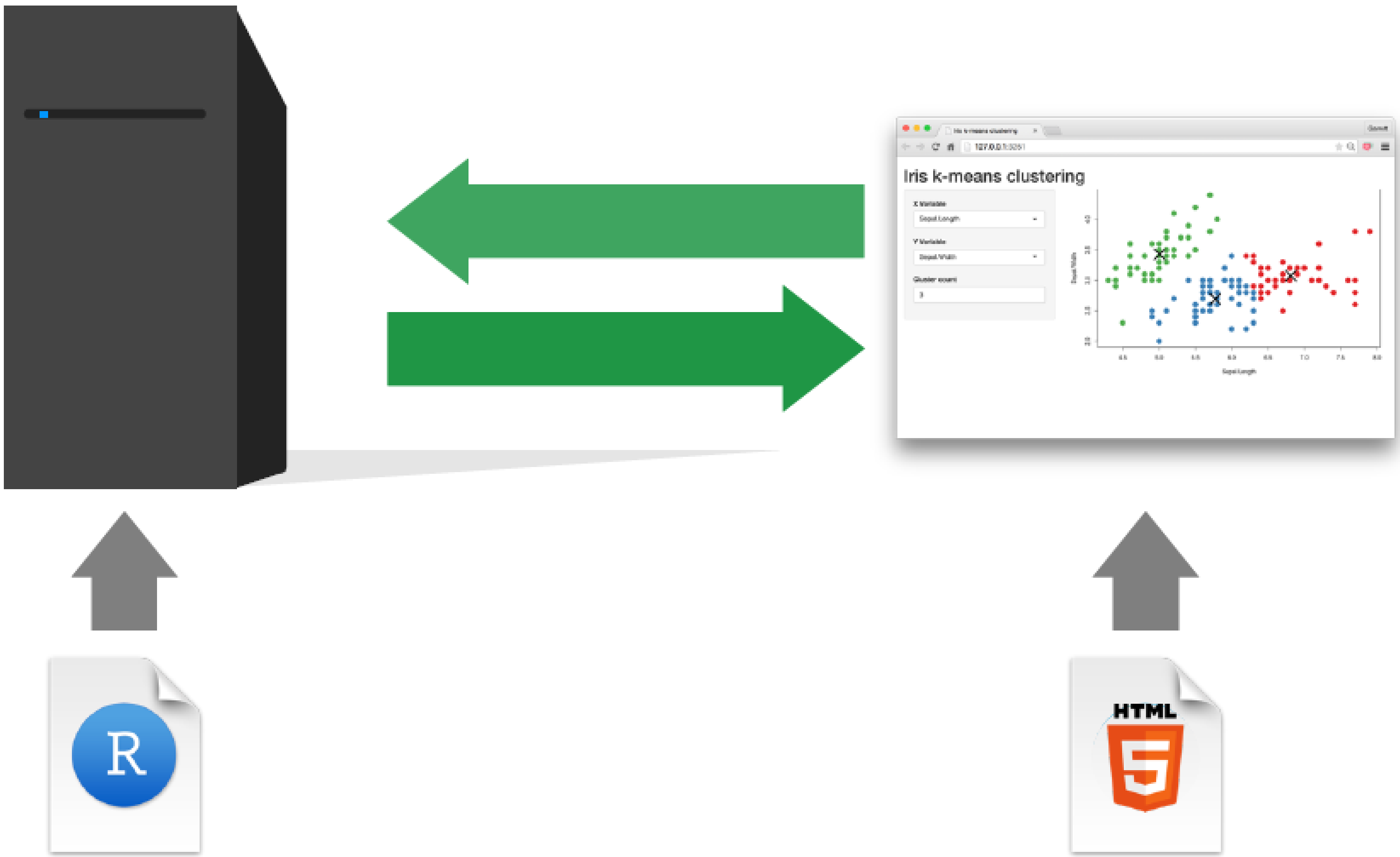
Ebola M

An epid simulat

© CC 2015 RStudio, Inc.

Use a simple Shiny App

- Open 02-hist-app.R
- Hit Run App



Iris k-means clustering

X Variable: Sepal Length

Y Variable: Sepal Width

Cluster count: 3

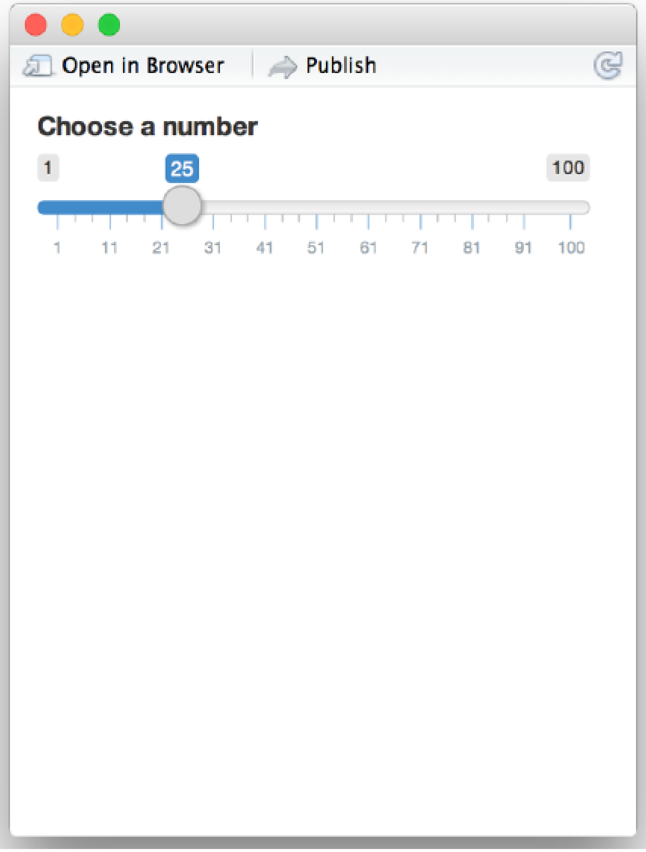
Server Instructions

User Interface (UI)

Inputs

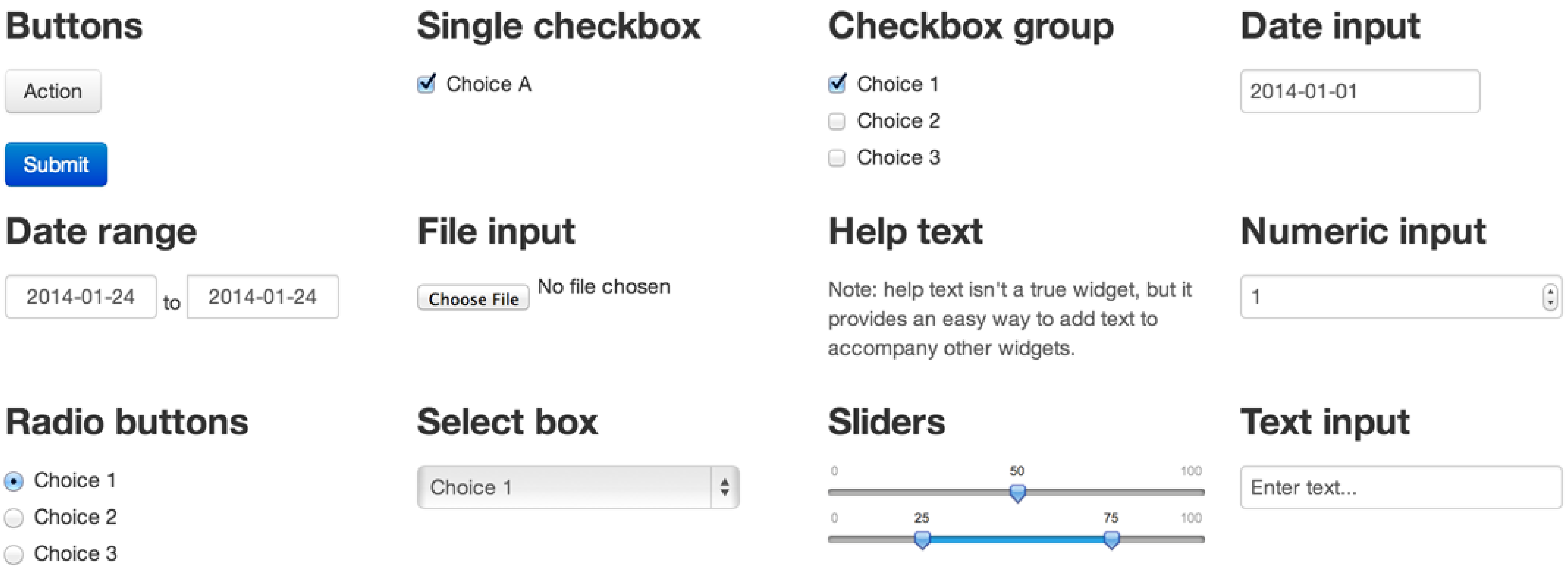
Create an input with an input function.

```
library(shiny)
ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100)
)

server <- function(input, output) {}

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



Choose a number

Inputs



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

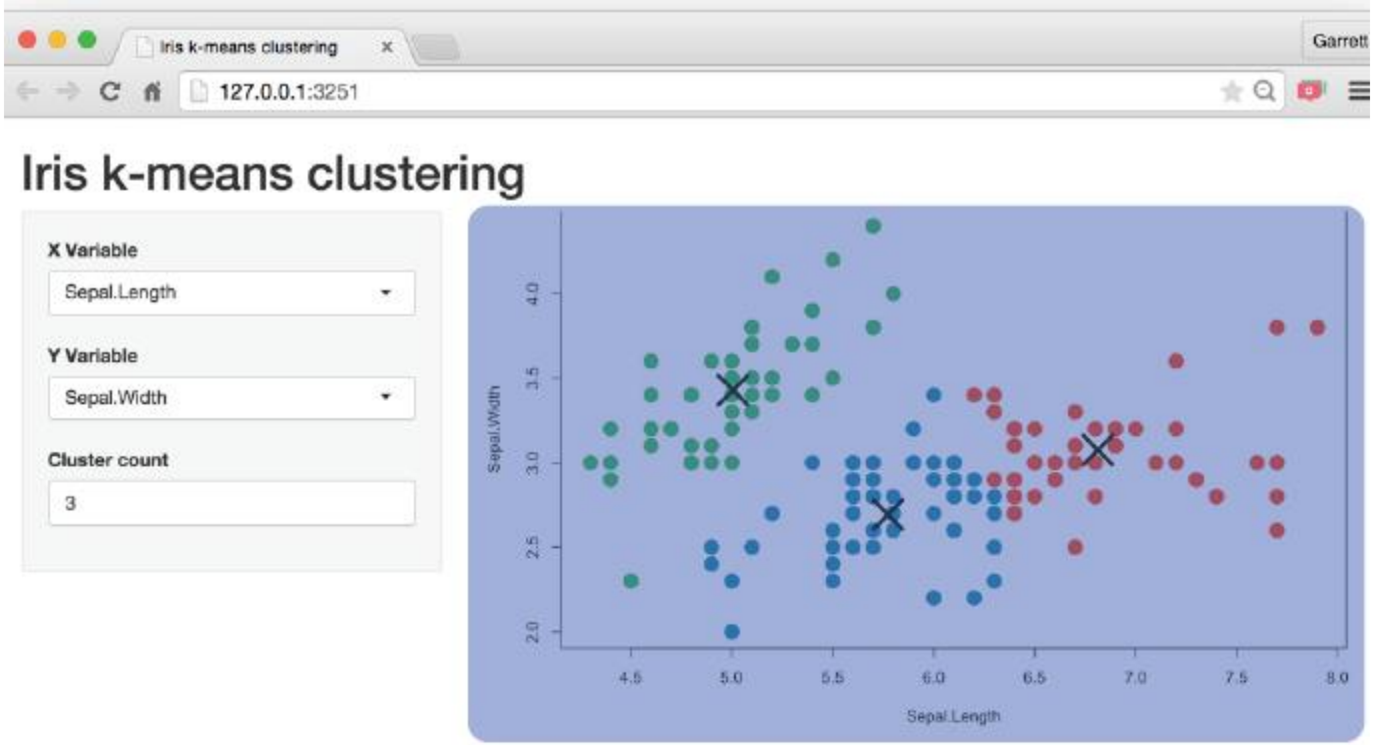
Sliders

Text input

Enter text...

Outputs

Build your app around **inputs** and **outputs**



Iris k-means clustering

X Variable

Sepal.Length

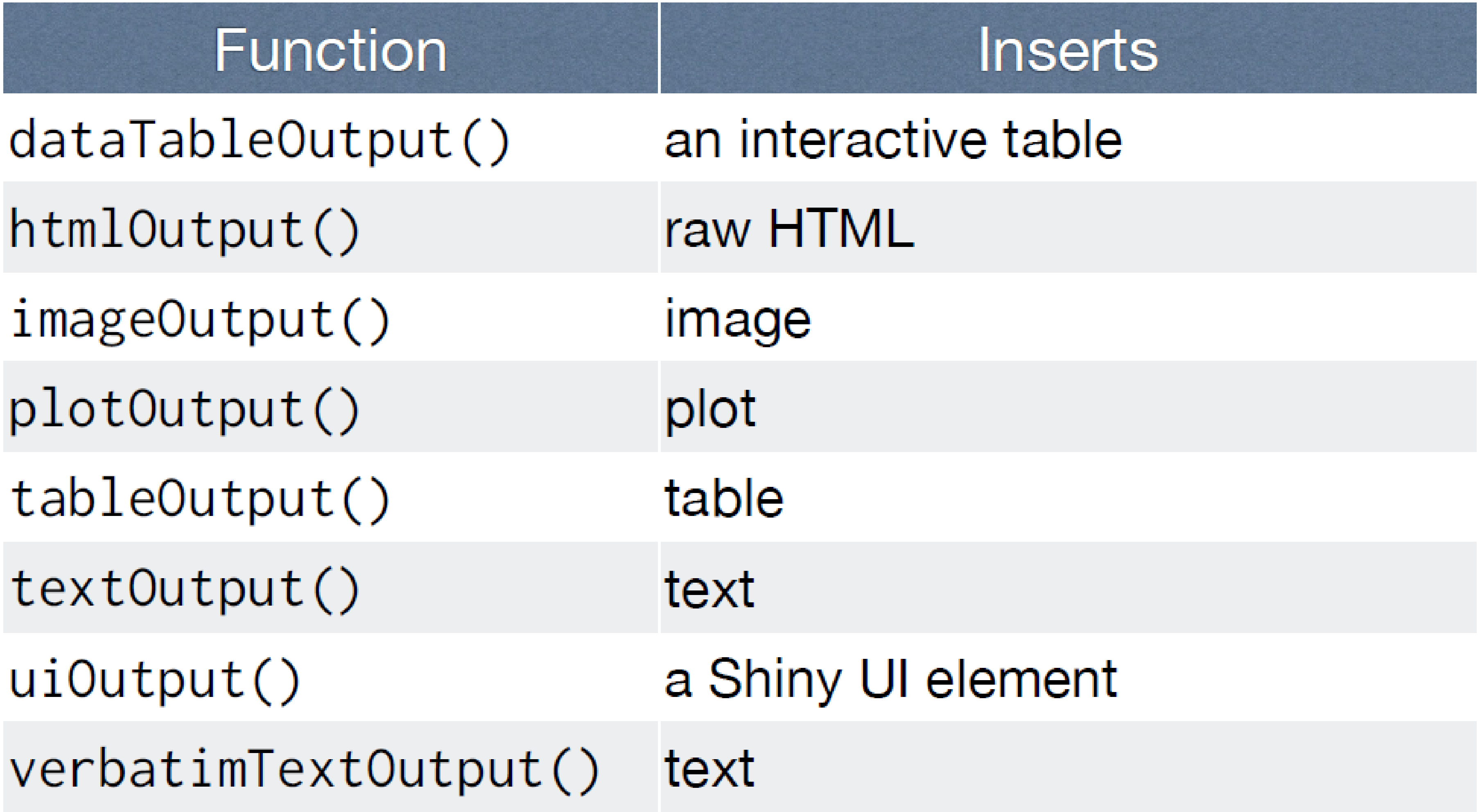
Y Variable

Sepal.Width

Cluster count

3

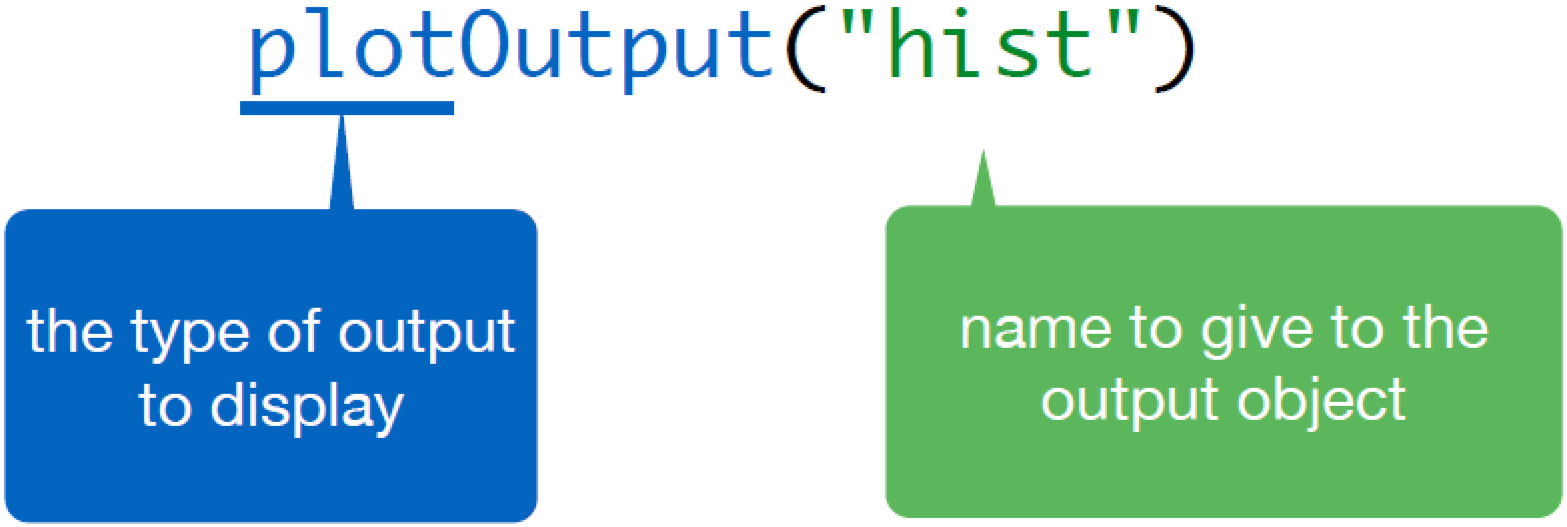
Outputs



Function	Inserts
`dataTableOutput()`	an interactive table
`htmlOutput()`	raw HTML
`imageOutput()`	image
`plotOutput()`	plot
`tableOutput()`	table
`textOutput()`	text
`uiOutput()`	a Shiny UI element
`verbatimTextOutput()`	text

*Output()

To display output, add it to fluidPage() with an *Output() function

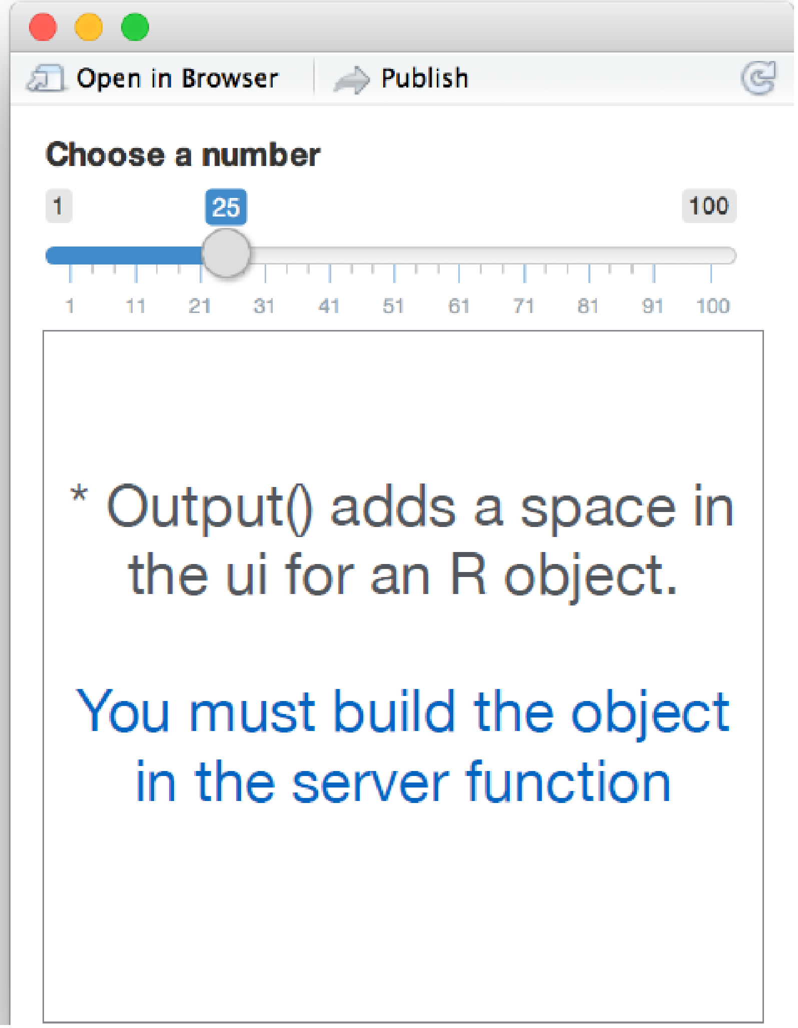


```
plotOutput("hist")
```

the type of output to display

name to give to the output object

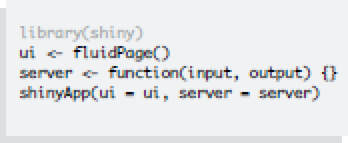
```
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist")
)

server <- function(input, output) {}

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



* Output() adds a space in the ui for an R object.

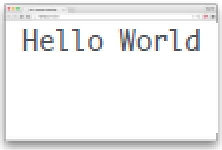
You must build the object in the server function

Recap



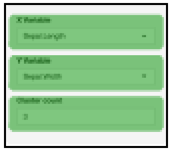
```
library(shiny)
ui <- fluidPage()
server <- function(input, output) {}
shinyApp(ui = ui, server = server)
```

Begin each app with the template



Hello World

Add elements as arguments to **fluidPage()**



X Variable

Sepal.Length

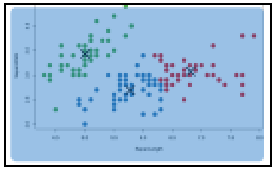
Y Variable

Sepal.Width

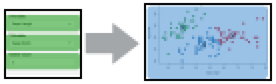
Cluster count

3

Create reactive inputs with an ***Input()** function



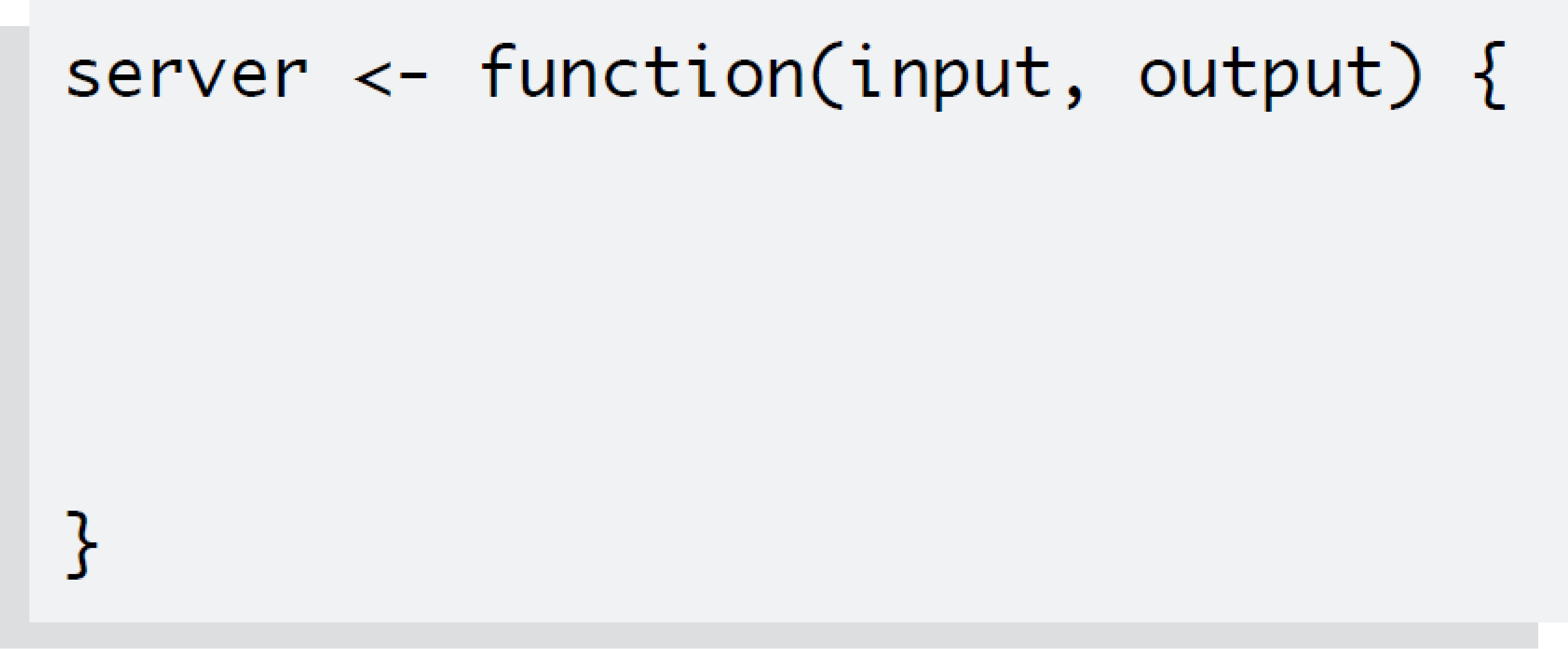
Display reactive results with an ***Output()** function



Assemble outputs from inputs in the server function

How to assemble inputs into outputs

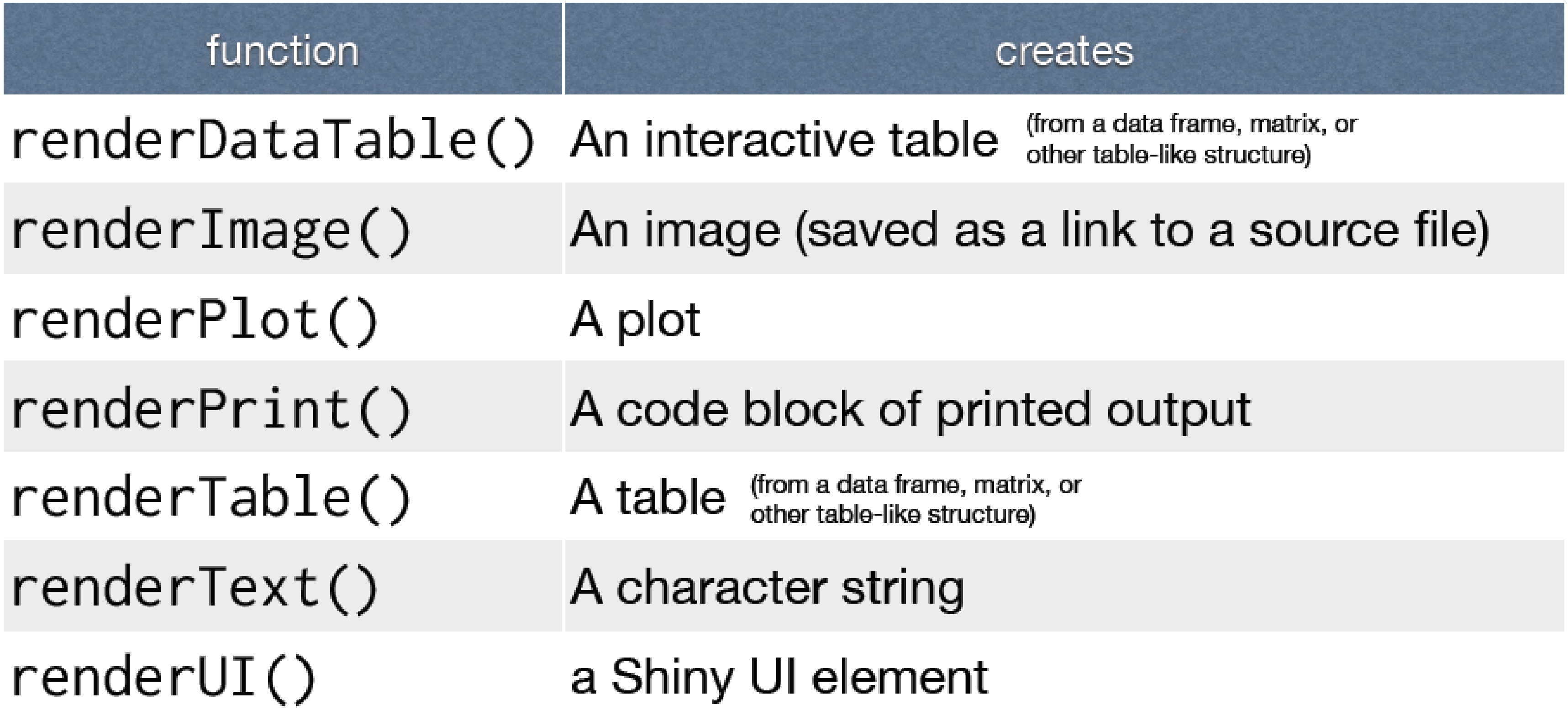
- Use 3 rules to write the server function



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



}
```

How to assemble inputs into outputs

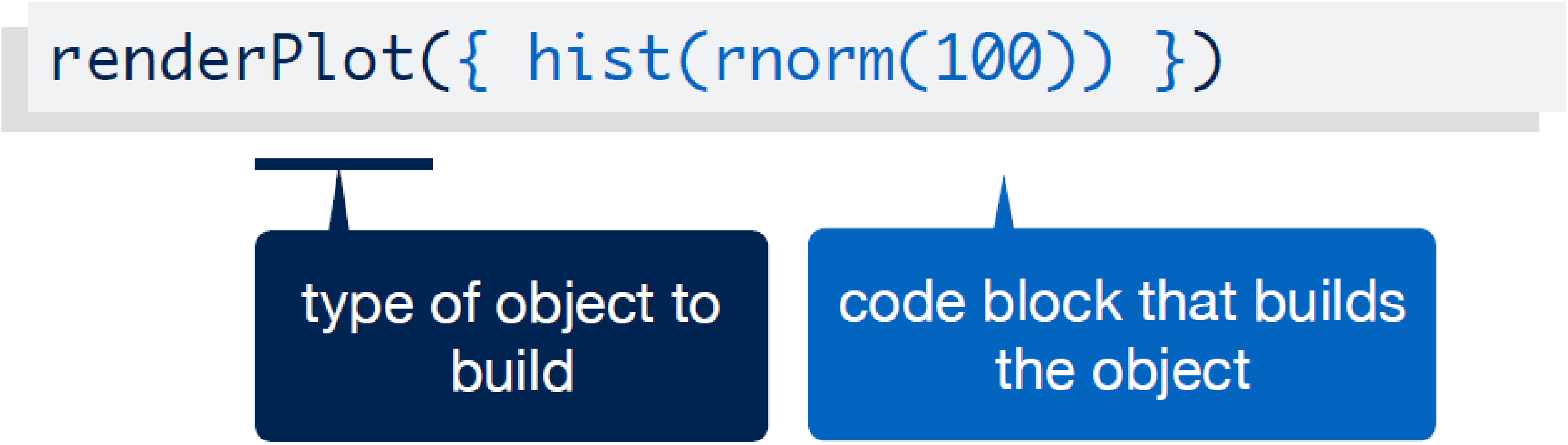
Use the **render*()** function that creates the type of output you wish to make.



function	creates
`renderDataTable()`	An interactive table (from a data frame, matrix, or other table-like structure)
`renderImage()`	An image (saved as a link to a source file)
`renderPlot()`	A plot
`renderPrint()`	A code block of printed output
`renderTable()`	A table (from a data frame, matrix, or other table-like structure)
`renderText()`	A character string
`renderUI()`	a Shiny UI element

render*()

Builds reactive output to display in UI

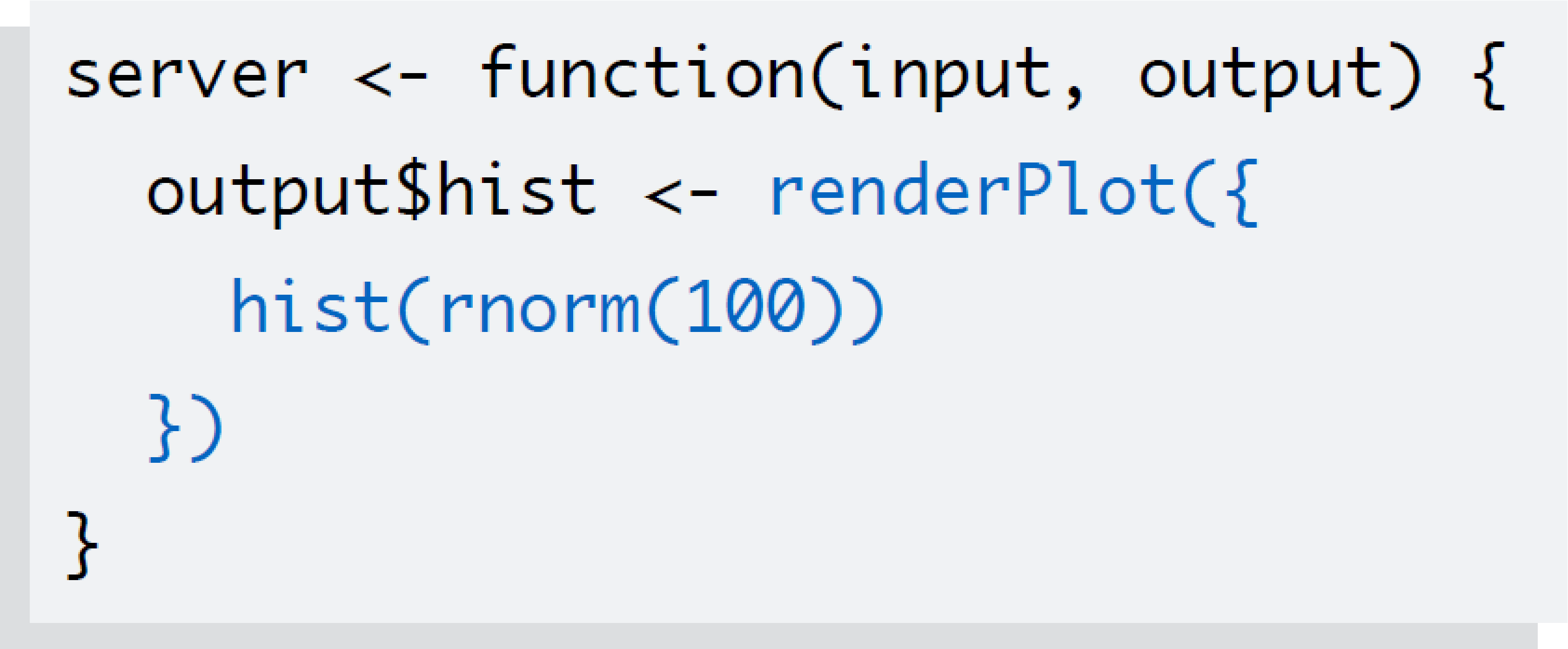


```
renderPlot({ hist(rnorm(100)) })
```

type of object to build

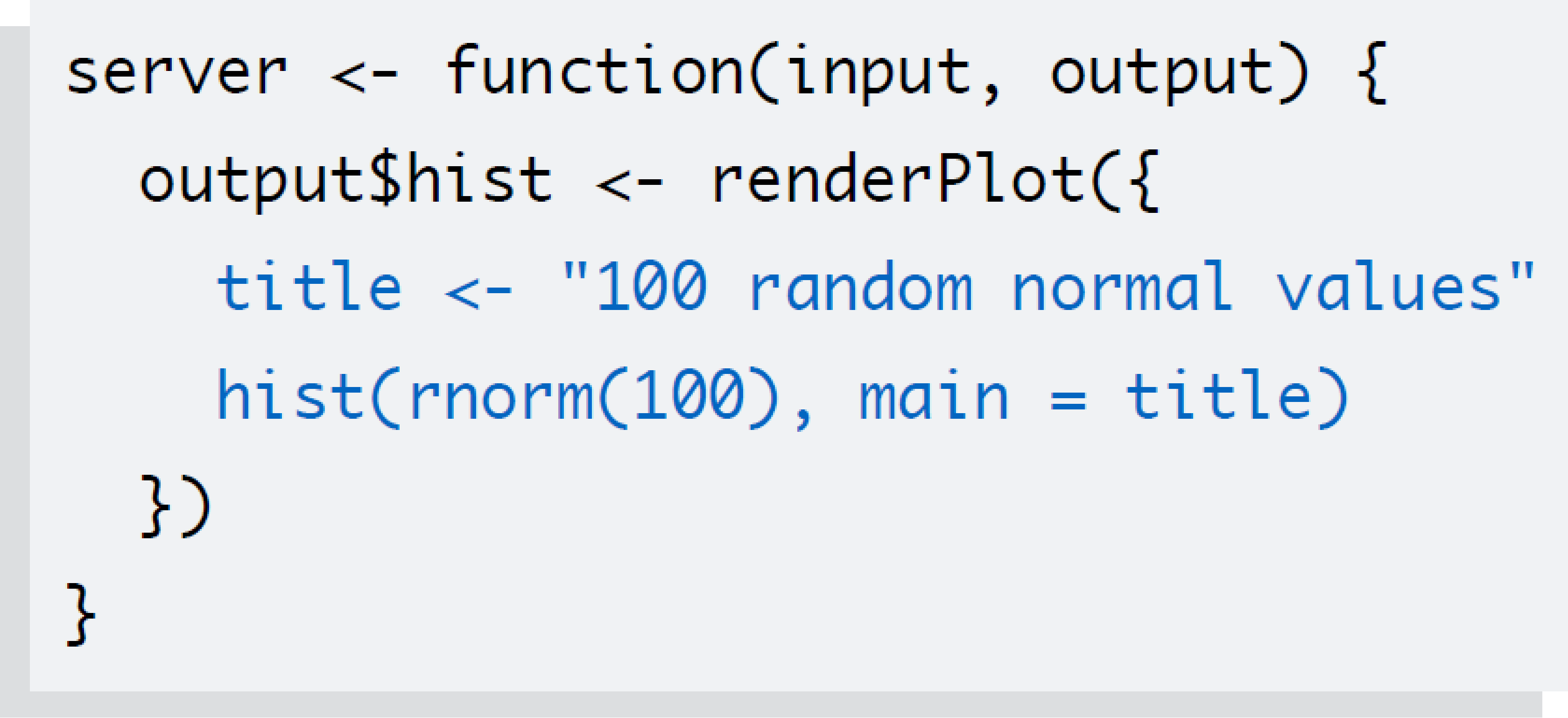
code block that builds the object

- Build objects to display with **render*()**



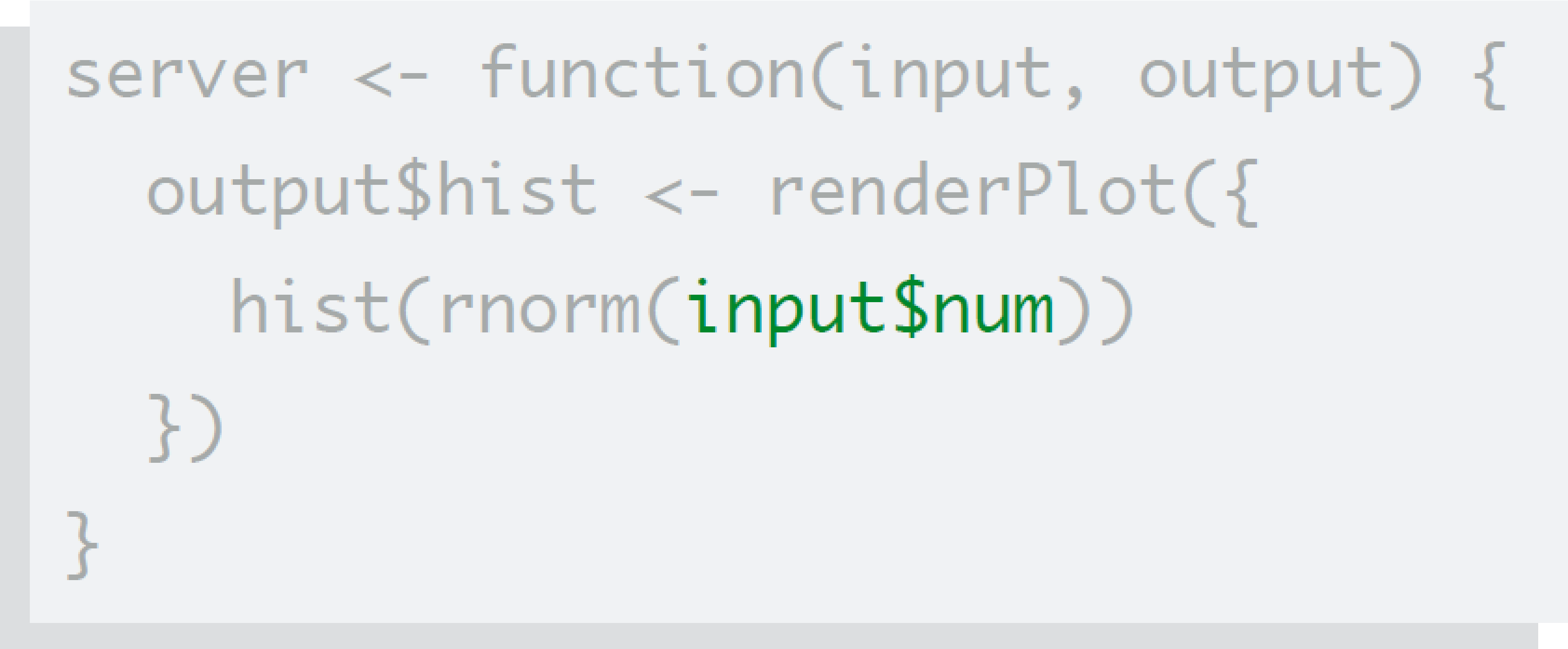
```
server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(100))
  })
}
```

- Build objects to display with **render*()**



```
server <- function(input, output) {
  output$hist <- renderPlot({
    title <- "100 random normal values"
    hist(rnorm(100), main = title)
  })
}
```

- Access **input** values with input$



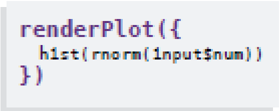
```
server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}
```

Recap: Server

Use the server function to assemble inputs into outputs. Follow 3 rules:

`output$hist <-`

1. Save the output that you build to **output$**

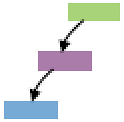


```
renderPlot({
  hist(rnorm(input$num))
})
```

2. Build the output with a **render*()** function

`input$num`

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

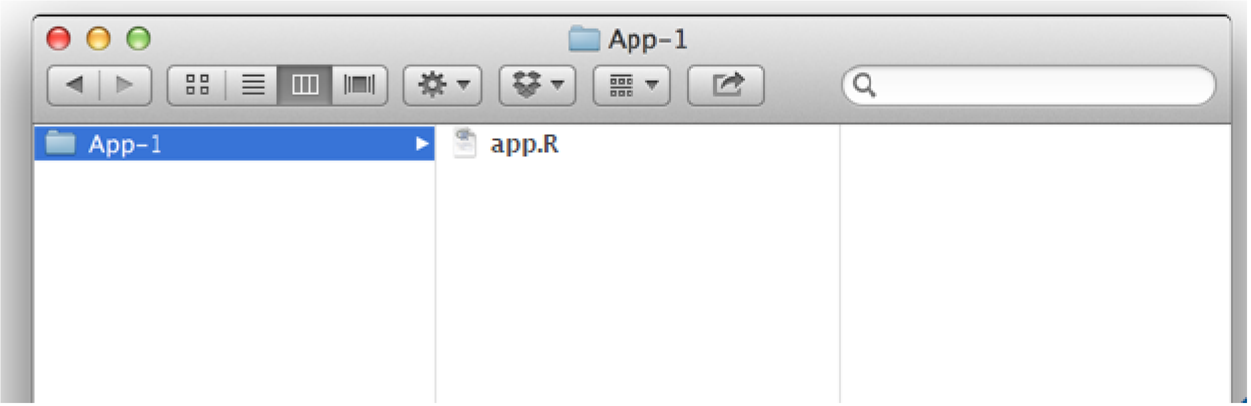


Create reactivity by using **Inputs** to build **rendered Outputs**

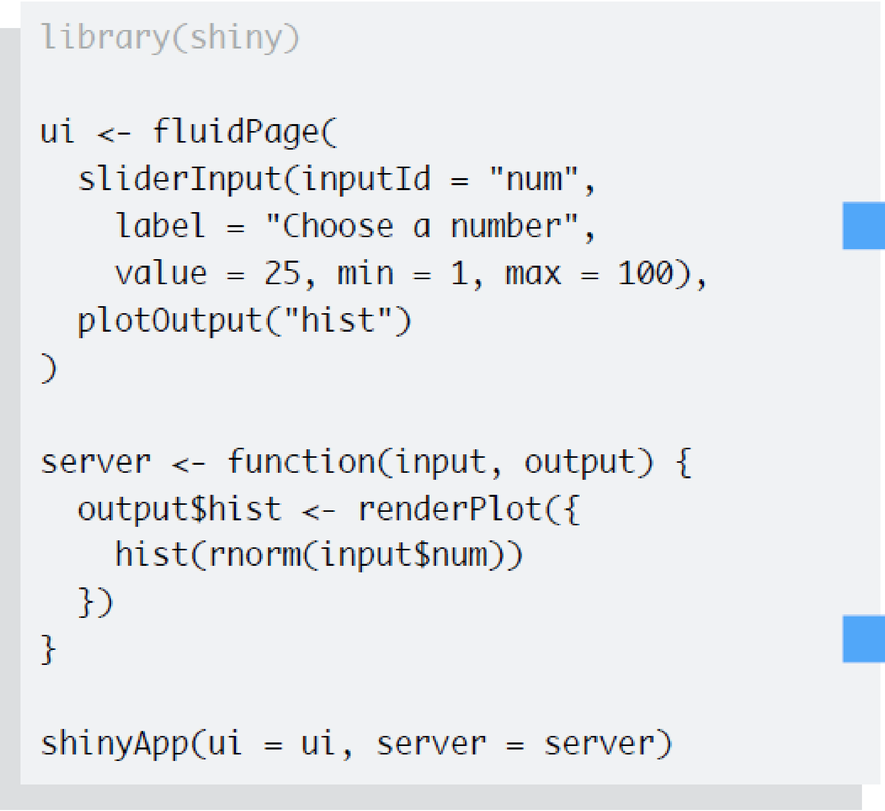
How to save your app

One directory with all the files the app needs:

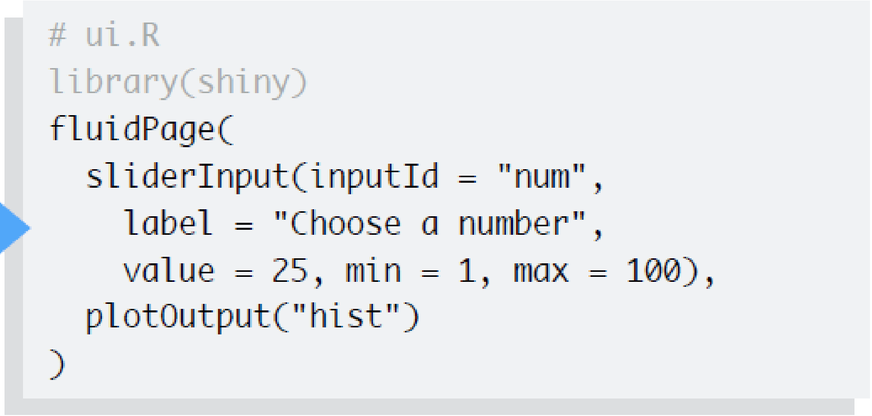
- app.R (your script which ends with a call to shinyApp())
- datasets, images, css, helper scripts, etc.



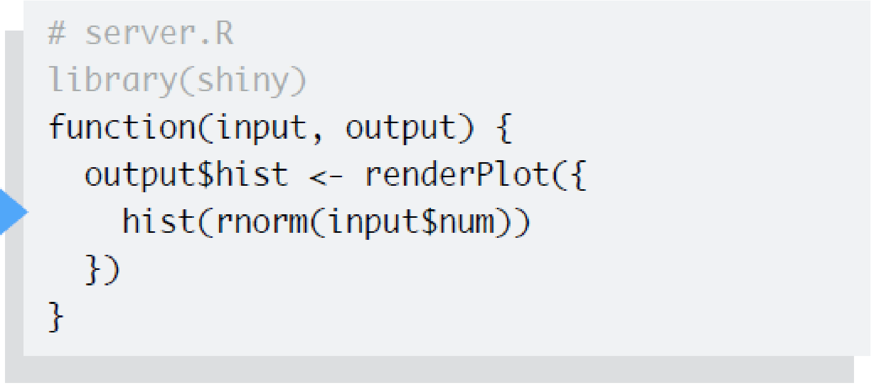
Two file apps



```r
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist")
)

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

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



```r
# ui.R
library(shiny)
fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist")
)
```

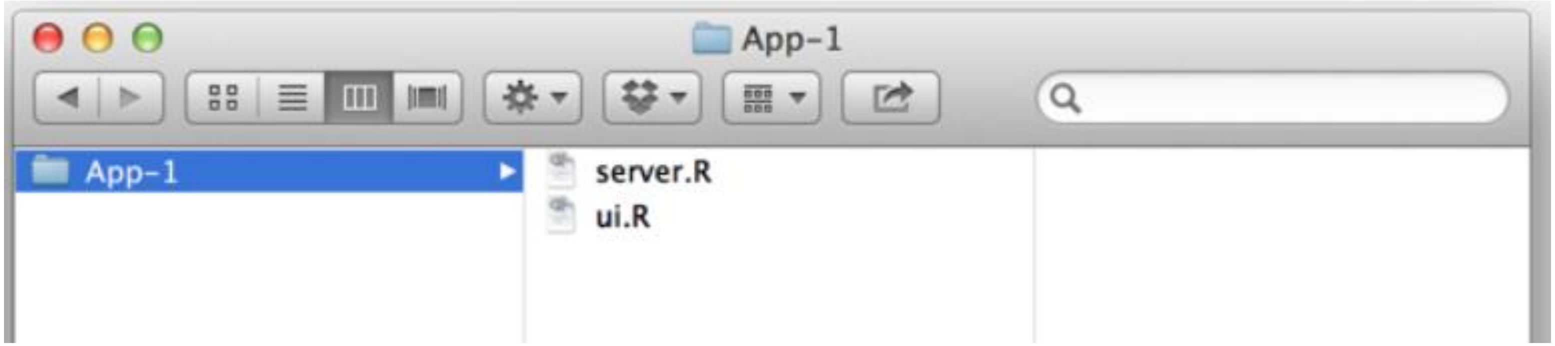


```r
# server.R
library(shiny)
function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}
```

Two file apps

One directory with two files:

- server.R
- ui.R



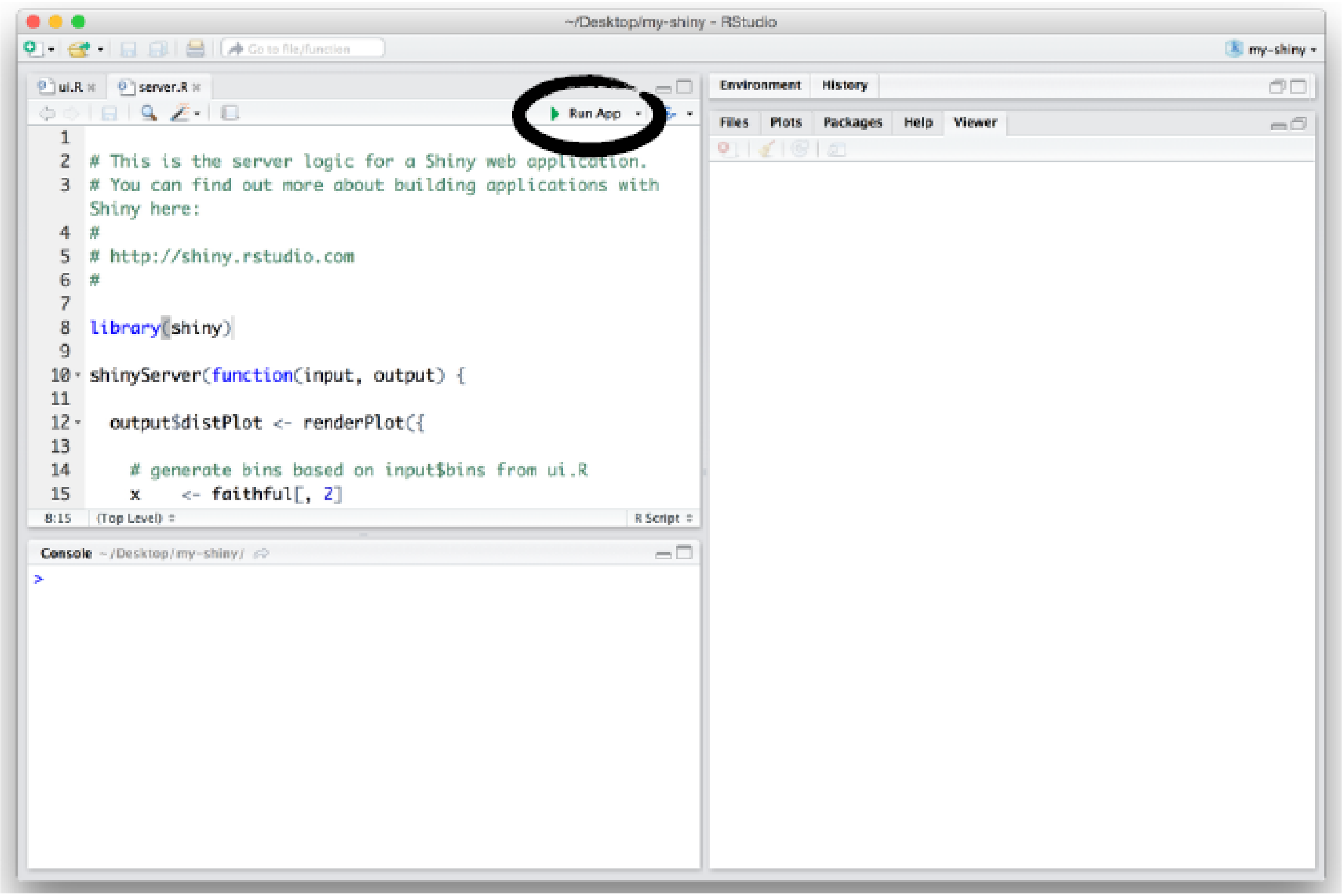
App-1

App-1

server.R

ui.R

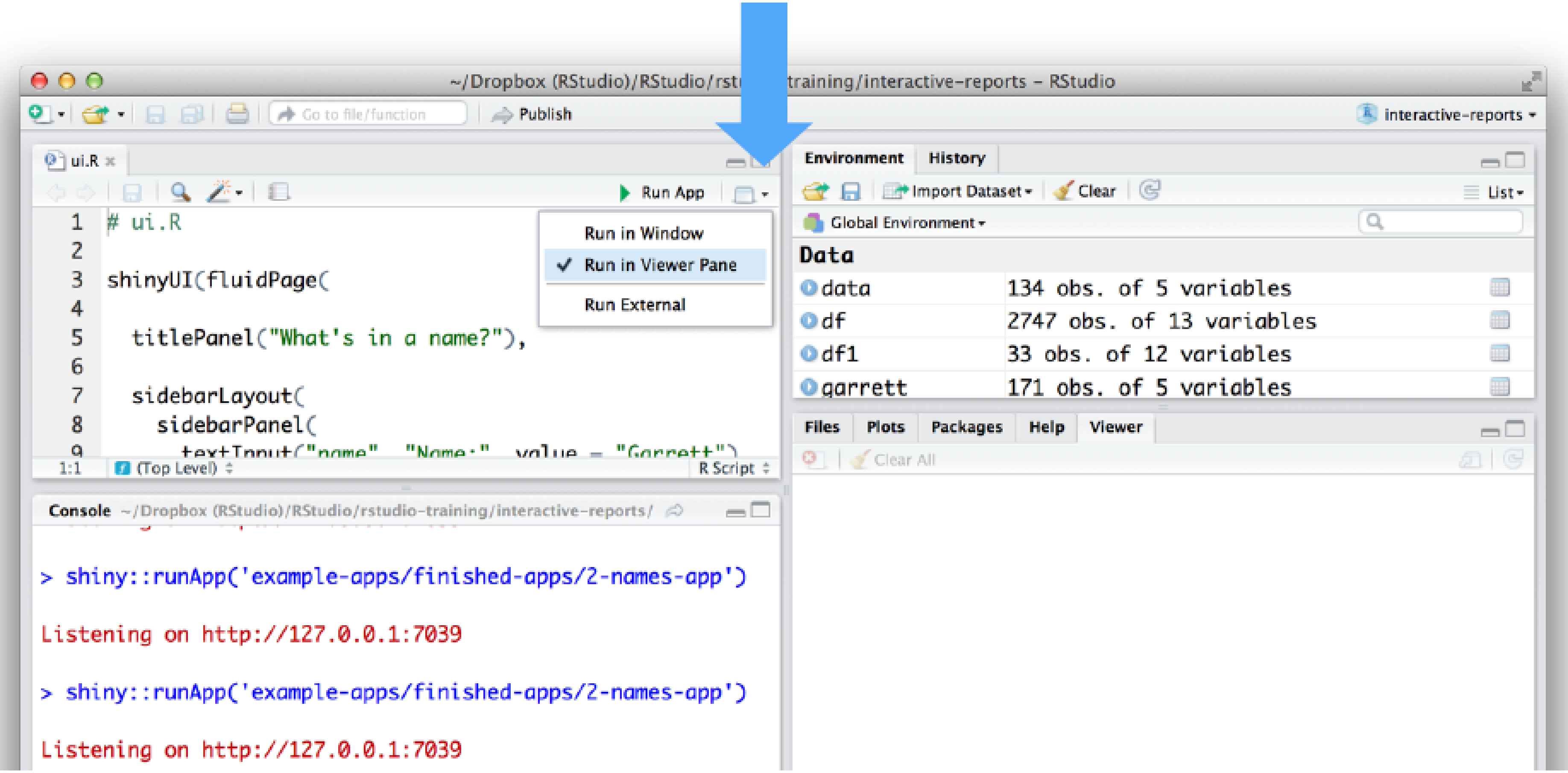
Launch an app



```

# This is the server logic for a Shiny web application.
# You can find out more about building applications with
Shiny here:
#
# http://shiny.rstudio.com
#

library(shiny)

shinyServer(function(input, output) {

  output$distPlot <- renderPlot({

    # generate bins based on input$bins from ui.R
    x    <- faithful[, 2]
```

```
>
```

Display options



```r
# ui.R

shinyUI(fluidPage(

  titlePanel("What's in a name?"),

  sidebarLayout(
    sidebarPanel(
      textInput("name", "Name:", value = "Garrett")
```

Data

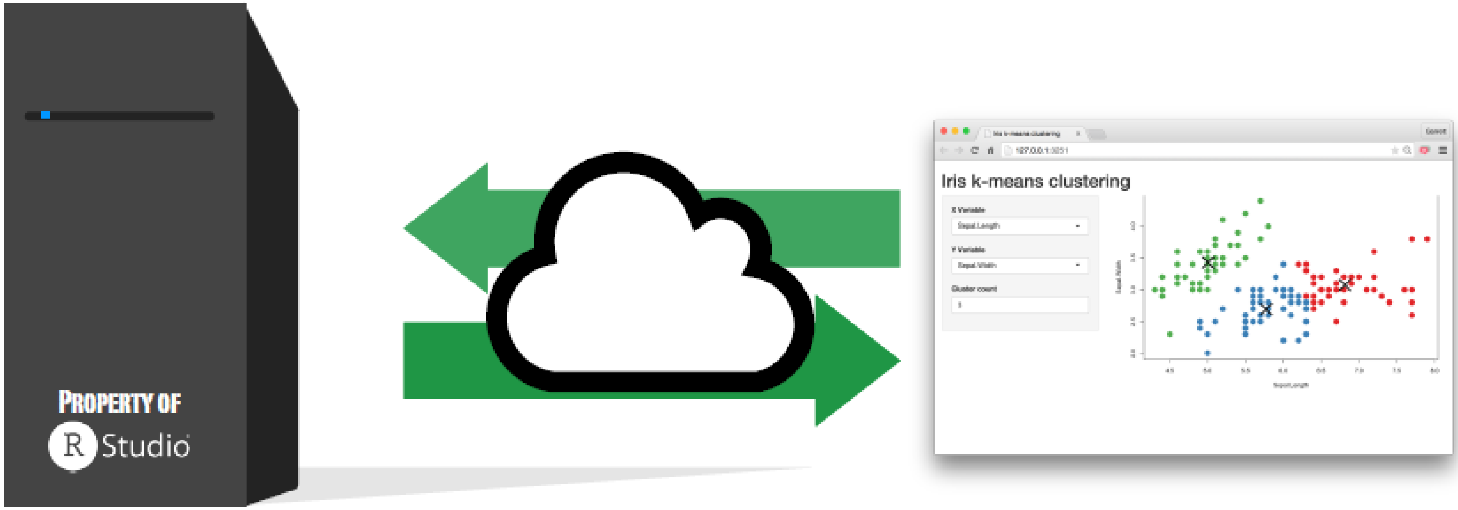
data	134 obs. of 5 variables
df	2747 obs. of 13 variables
df1	33 obs. of 12 variables
garrett	171 obs. of 5 variables

```
> shiny::runApp('example-apps/finished-apps/2-names-app')

Listening on http://127.0.0.1:7039

> shiny::runApp('example-apps/finished-apps/2-names-app')

Listening on http://127.0.0.1:7039
```

Shinyapps.io

A server maintained by Rstudio

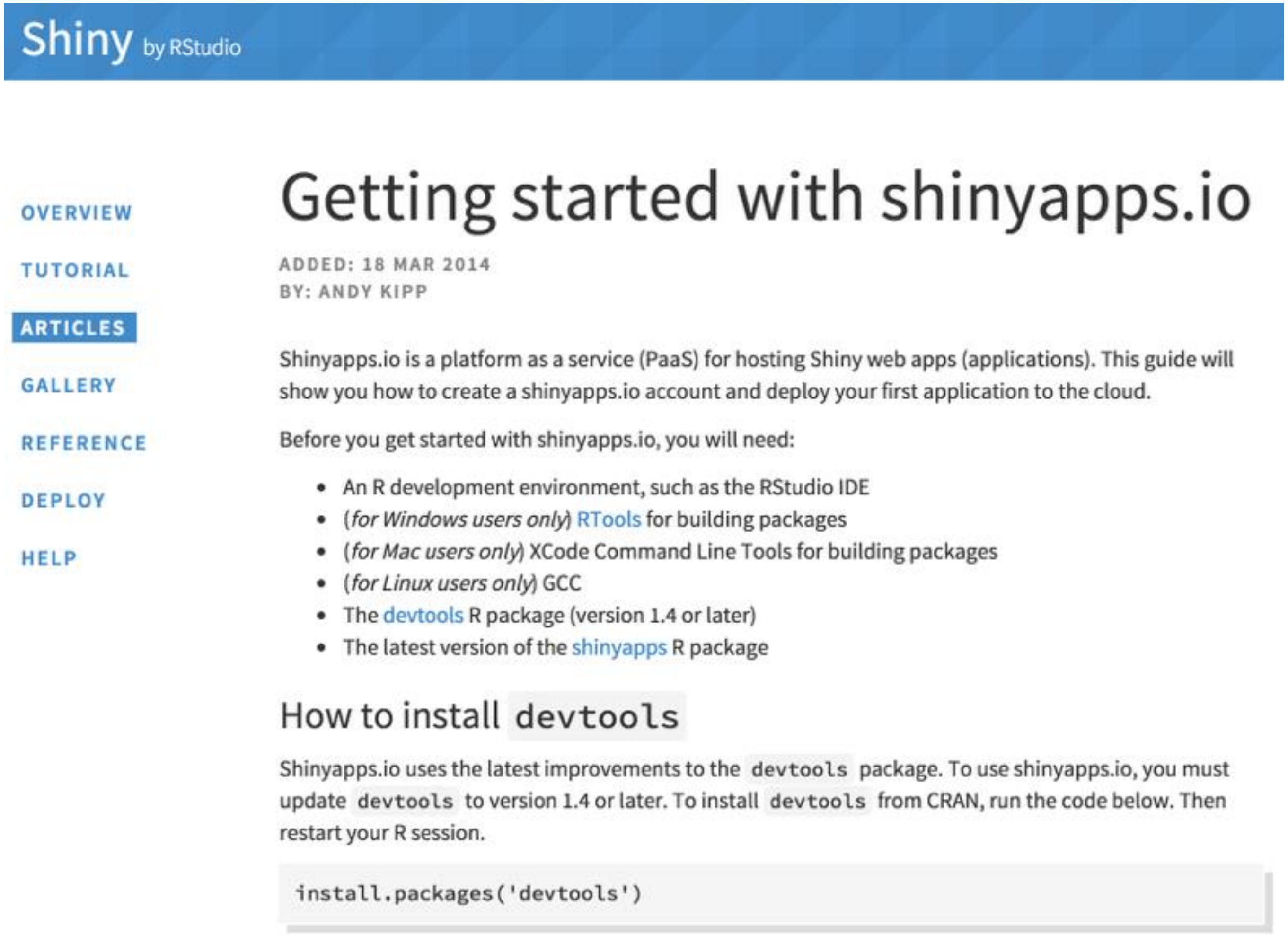
- free
- easy to use
- secure
- scalable



X Variable
Sepal.Length
Y Variable
Sepal.Width
Cluster count
3

Getting started guide

shiny.rstudio.com/articles/shinyapps.html



Getting started with shinyapps.io

ADDED: 18 MAR 2014
BY: ANDY KIPP

Shinyapps.io is a platform as a service (PaaS) for hosting Shiny web apps (applications). This guide will show you how to create a shinyapps.io account and deploy your first application to the cloud.

Before you get started with shinyapps.io, you will need:

- An R development environment, such as the RStudio IDE
- (*for Windows users only*) RTools for building packages
- (*for Mac users only*) XCode Command Line Tools for building packages
- (*for Linux users only*) GCC
- The devtools R package (version 1.4 or later)
- The latest version of the shinyapps R package

How to install `devtools`

Shinyapps.io uses the latest improvements to the `devtools` package. To use shinyapps.io, you must update `devtools` to version 1.4 or later. To install `devtools` from CRAN, run the code below. Then restart your R session.

```
install.packages('devtools')
```

R and Spark

Sparklyr is an R interface for Apache Spark, you can:

- Connect to Spark from R. The sparklyr package provides a complete dplyr backend.
- Filter and aggregate Spark datasets then bring them into R for analysis and visualization.
- Use Spark's distributed machine learning library from R.
- Create extensions that call the full Spark API and provide interfaces to Spark packages.

Once you have connected to Spark, then copying and interacting is super-fast and easy

Doc-06.pdf

Python and/or R?

- Both can be used: There were a number of Python module choices to access R. They are: rpy2, pyRserve and PypeR.
- From R, Python can also be used:

rPython - an R package which allows the user to call Python from R

References
Statistics with R

- *http://rstudio.com/cheatsheets*

R YOU SURE?

- **If I want to upgrade my data analysis skills, which programming language should I learn?**

- **Introduction to R for Python Programmers**
  http://ramnathv.github.io/pycon2014-r/
- **The Art of R Programming** Norman Matloff, WPublisher
- **R in action**, Robert I. Kabacoff, Manning Publications
- **Introductory Statistics with R**, Peter Dalgaard, Springer
- **Data Analysis and Graphics using R** , John Maindonald & W. John Braun, Cambridge University Press
- **The R Book**, Michael J. Crawley, Ed. John Wiley & Sons
- **R for dummies**, Joris Meys, Andrie de Vries Ed. John Wiley & Sons
- **Beginning R: The Statistical Programming Language**, Mark Gardener, Wrox

Now …

Working with real data, a case study

Rmardown files for the Case Study

Prog-07.Rmd	EDA bank data
Prog-08.Rmd	Logistic regression: bank data
Prog-09.Rmd	Further models: bank data
Prog-10.Rmd	Prediction and crossvalidation: bank data

Or open Prog-XX.html directly to see the output

Statistics with R

Enjoy R and Data Analysis!

Some favorite quotes:

http://www.ub.edu/riskcenter/guillen
mguillen@ub.edu
@mguillen_estany

“There are no routine statistical questions, only questionable statistical routines.”
Sir David Cox

"An approximate answer to the right problem is worth a good deal more than an exact answer to an approximate problem."
John Tukey

“All models are wrong, but some are useful. “
George E. P. Box