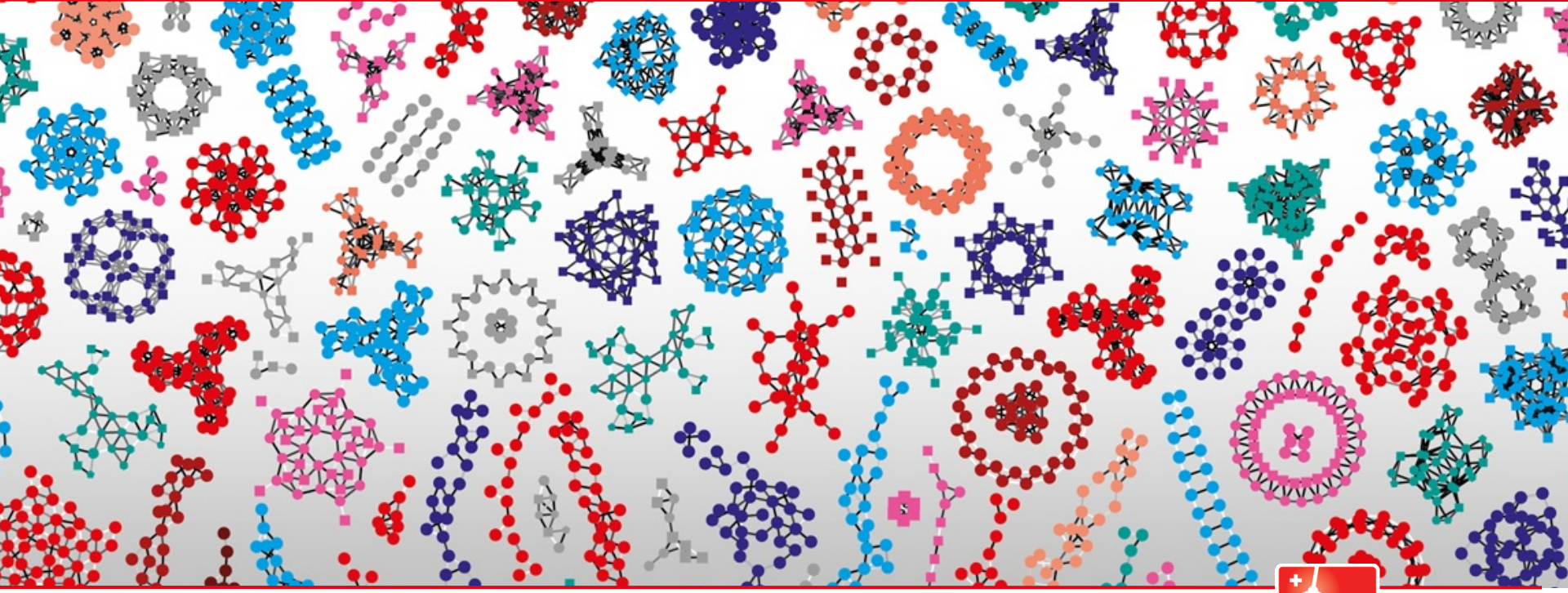


Swiss Institute of Bioinformatics

Introduction à R Shiny

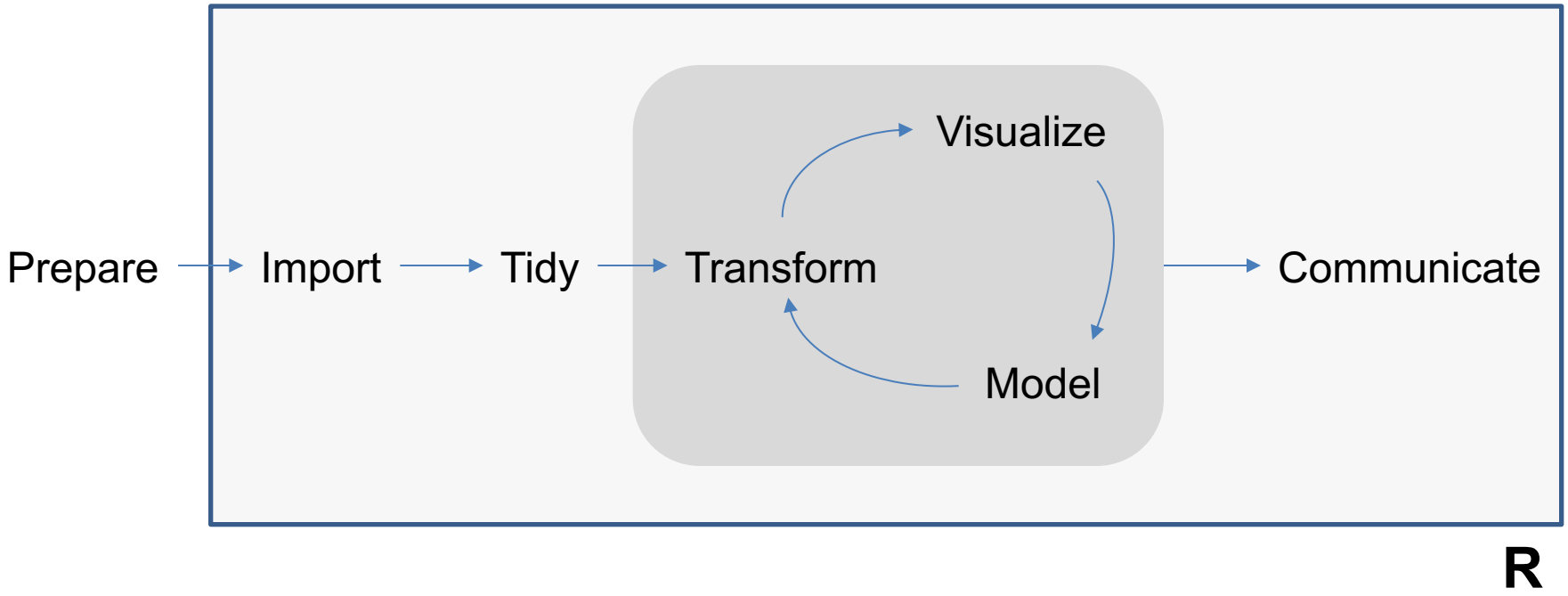
Frédéric Schütz

(avec du matériel emprunté à Linda Dib et Martial Sankar)

Matériel de cours

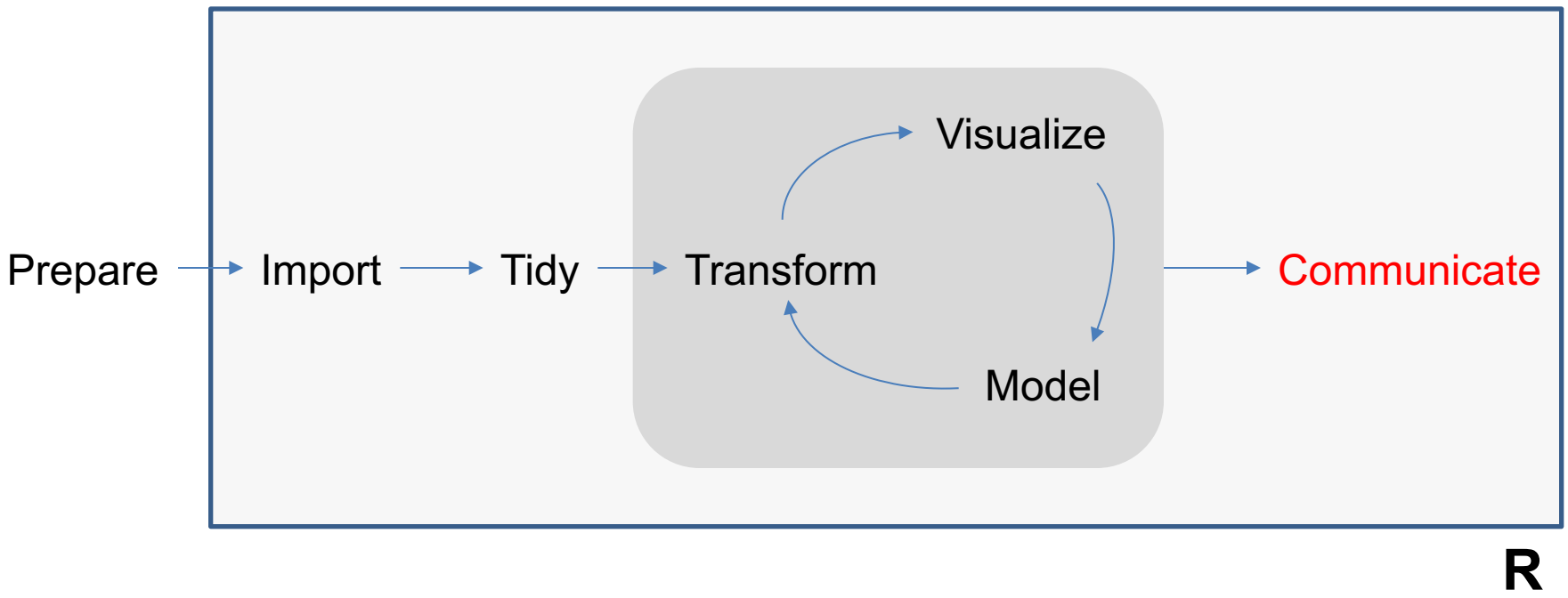
https://github.com/IFB-ElixirFr/R-Shiny_training_2019-07

Data analysis workflow



Adapted from Hadley Wickham
"R for data analysis"

Data analysis workflow



Adapted from Hadley Wickham
"R for data analysis"

knitr and R Markdown

RStudio

File Edit Code View Plots Session Build Debug Tools Help

Go to file/function Project: (None)

rmarkdown.Rmd

Knit HTML Run Chunks

```
1  ---
2  title: "RMarkdown example"
3  author: "Frédéric Schütz"
4  date: "19/11/2015"
5  output: html_document
6  ---
7
8  This is an R Markdown document. Markdown is a simple
   formatting syntax for authoring HTML, PDF, and MS Word
   documents. For more details on using R Markdown see
   <http://rmarkdown.rstudio.com>.
9
10 # Introduction
11
12 Markdown allows you to create sections and subsections,
   to emphasize text by formatting it in *italic* or **bold
   **.
13
14 ## Lists
15
16 You can also easily create lists:
17
18 * Item 1
19 * Item 2
20     + Item 2a
```

2:9 (Top Level) R Markdown

Console

~/Documents/sib/reproducible_research/rmarkdown.html

rmarkdown.html Open in Browser Publish Find

RMarkdown example

Frédéric Schütz

19/11/2015

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

Introduction

Markdown allows you to create sections and subsections, to emphasize text by formatting it in *italic* or **bold**.

Lists

You can also easily create lists:

- Item 1
- Item 2
  - Item 2a

1. Ordered list
2. Another item
   - A subitem

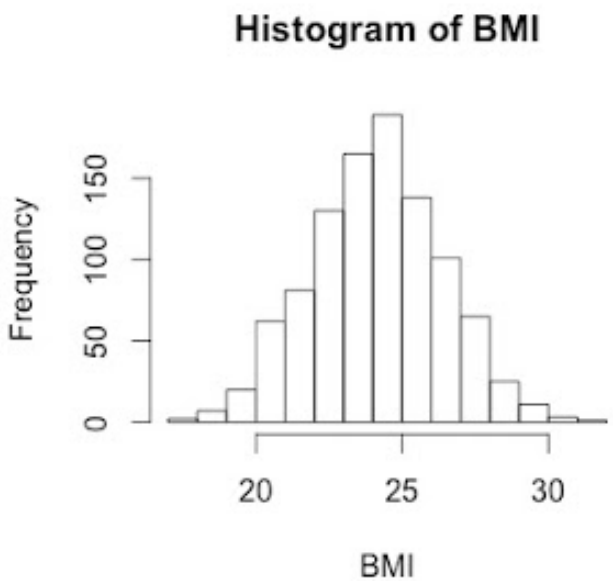
or even tables:

First Header	Second Header
Content Cell	Content Cell
Content Cell	Content Cell

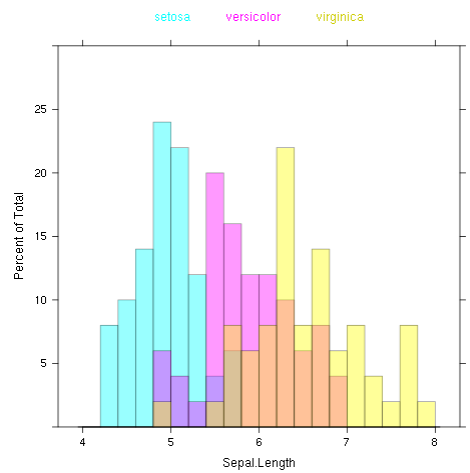
Visualising data

3 models for graphics in R

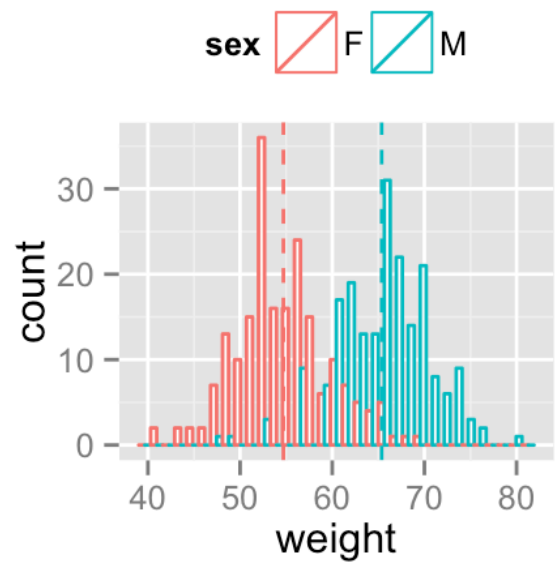
Base graphics



Lattice



ggplot



Interactive graphs

plotly Dash Consulting

SIGN IN SIGN UP REQUEST DEMO

Help | Open Source Graphing Libraries | R

Fork on GitHub

Quick Start

- Getting Started
- Cheat Sheet
- Full Reference
- User Guide
- Use Offline
- ggplot2
- Shiny Gallery
- Shiny for Python

Plotly R Open Source Graphing Library

Plotly's R graphing library makes interactive, publication-quality graphs online. Examples of how to make line plots, scatter plots, area charts, bar charts, error bars, box plots, histograms, heatmaps, subplots, multiple-axes, and 3D (WebGL based) charts.

Search

Search Plotly's R & ggplot2 D

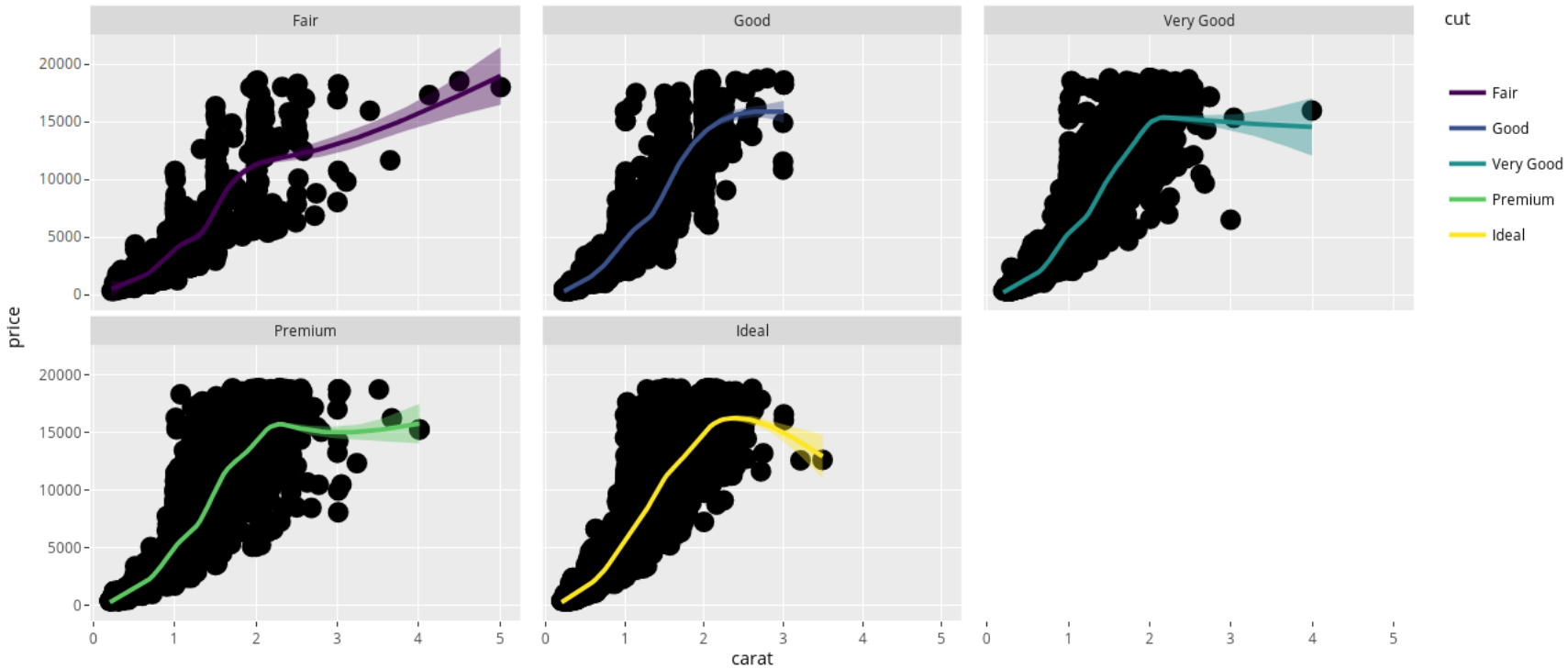
Plotly Fundamentals

You've collected the data, agonized over the right model and now you've GOT to convince management.....

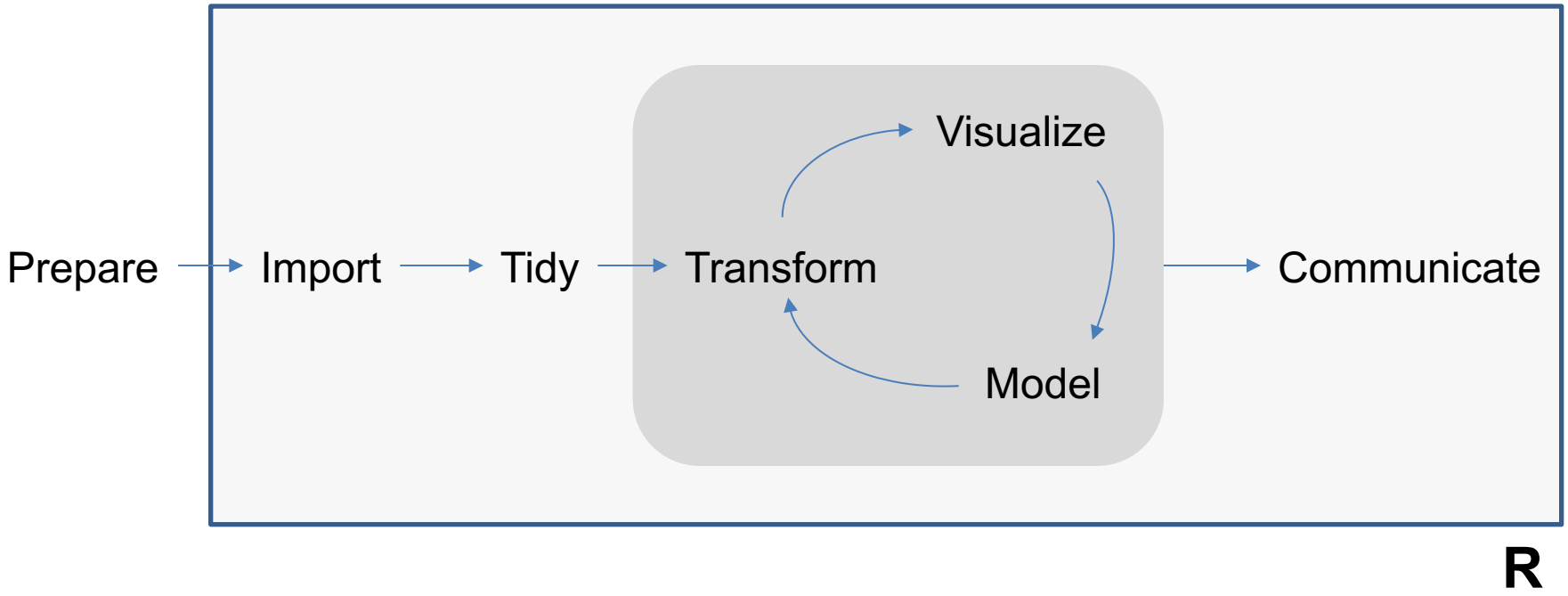
Plotly & ggplot2

```
library(plotly)
data(diamonds)

p <- ggplot(
       data = diamonds,
       aes(x = carat, y = price)) +
       geom_point(aes(text = paste("Clarity:", clarity)), size = 4) +
       geom_smooth(aes(colour = cut, fill = cut)) +
       facet_wrap(~ cut)

(gg <- ggplotly(p))
```

file:///tmp/RtmplTrhFY/viewhtml2bd15e709de2/index.html

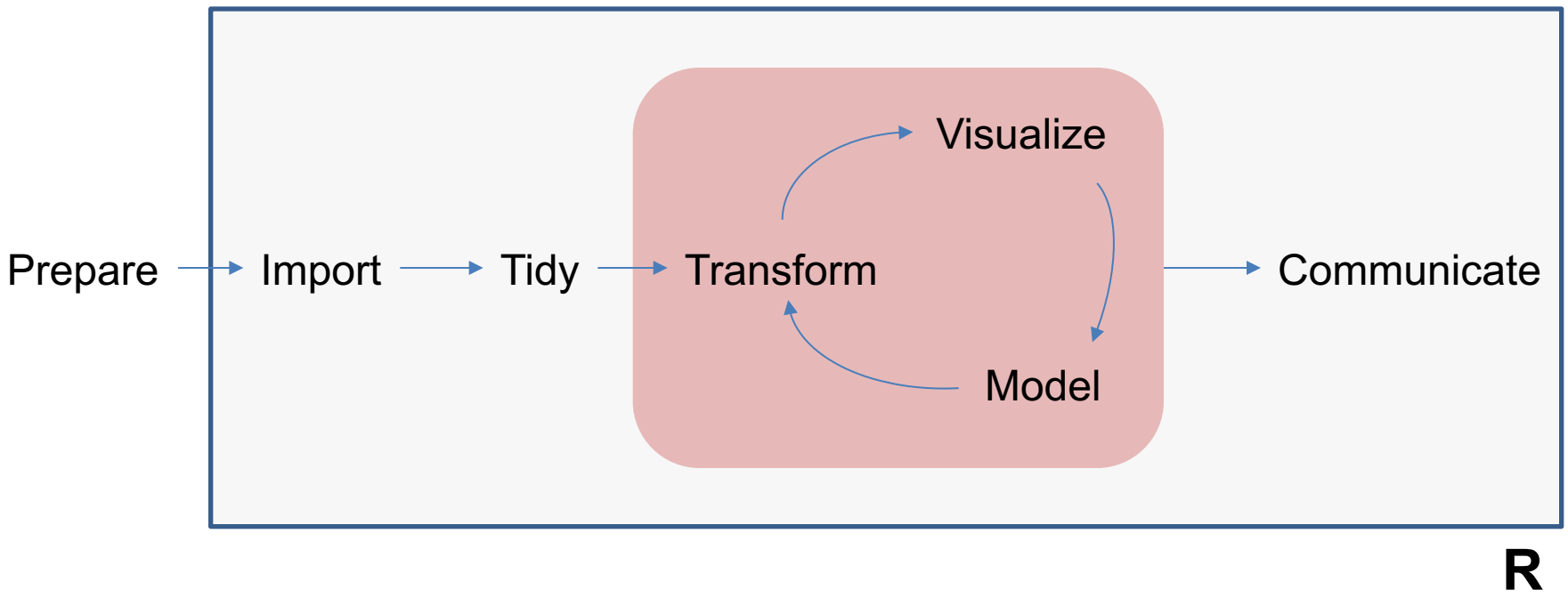


Data analysis workflow



Adapted from Hadley Wickham
"R for data analysis"

Data analysis workflow



Adapted from Hadley Wickham
"R for data analysis"

"Interactive" analysis with R (and Rstudio): accessing an R application

Example: simulation

```
n1 <- 25
n2 <- 25
d  <- 2
sd <- 2.45

p <- NULL
for (i in 1:10000) {
    data1 <- rnorm(n1, mean=0, sd=sd)
    data2 <- rnorm(n2, mean=d, sd=sd)

    p <- c(p, t.test(data1, data2)$p.value)
}

sum(p < 0.05) / length(p)
```

Example: simulation

```
n1 <- 25
n2 <- 25
d  <- 2
sd <- 2.45

p <- NULL
for (i in 1:10000) {
    data1 <- rnorm(n1, mean=0, sd=sd)
    data2 <- rnorm(n2, mean=d, sd=sd)

    p <- c(p, t.test(data1, data2)$p.value)
}

sum(p < 0.05) / length(p)
[1] 0.7997
```

Example: new simulation

```
n1 <- 35
n2 <- 15
d  <- 2
sd <- 2.45

p <- NULL
for (i in 1:10000) {
    data1 <- rnorm(n1, mean=0, sd=sd)
    data2 <- rnorm(n2, mean=d, sd=sd)

    p <- c(p, t.test(data1, data2)$p.value)
}

sum(p < 0.05) / length(p)
```

Example: new simulation

```
n1 <- 35
n2 <- 15
d  <- 2
sd <- 2.45

p <- NULL
for (i in 1:10000) {
    data1 <- rnorm(n1, mean=0, sd=sd)
    data2 <- rnorm(n2, mean=d, sd=sd)

    p <- c(p, t.test(data1, data2)$p.value)
}

sum(p < 0.05) / length(p)
[1] 0.7239
```

Example: new simulation

```
n1 <- 55
n2 <- 15
d  <- 2
sd <- 2.45

p <- NULL
for (i in 1:10000) {
    data1 <- rnorm(n1, mean=0, sd=sd)
    data2 <- rnorm(n2, mean=d, sd=sd)

    p <- c(p, t.test(data1, data2)$p.value)
}

sum(p < 0.05) / length(p)
[1] 0.769
```

Can we provide better access to these…

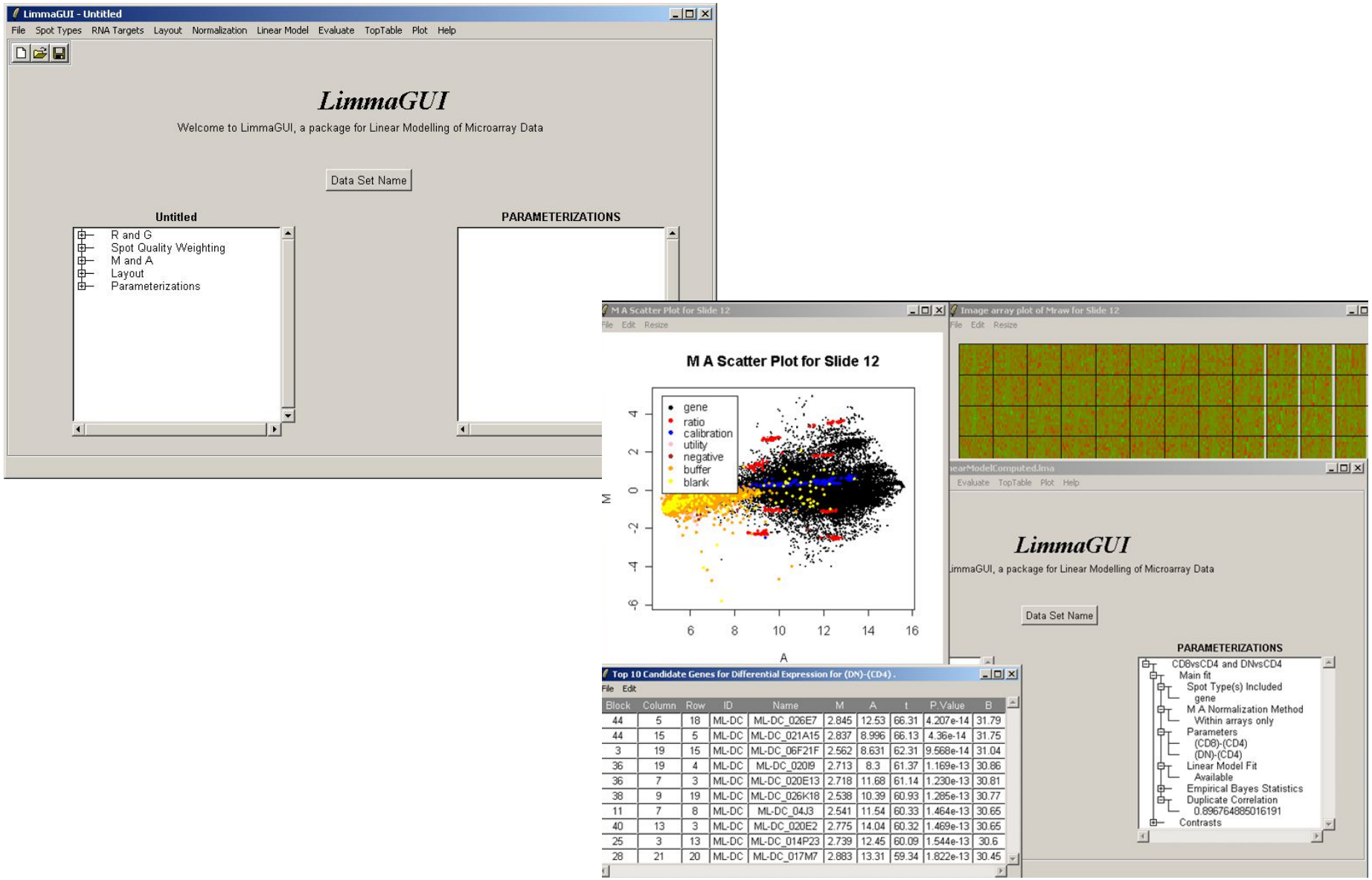
… results

…visualizations

… applications ?

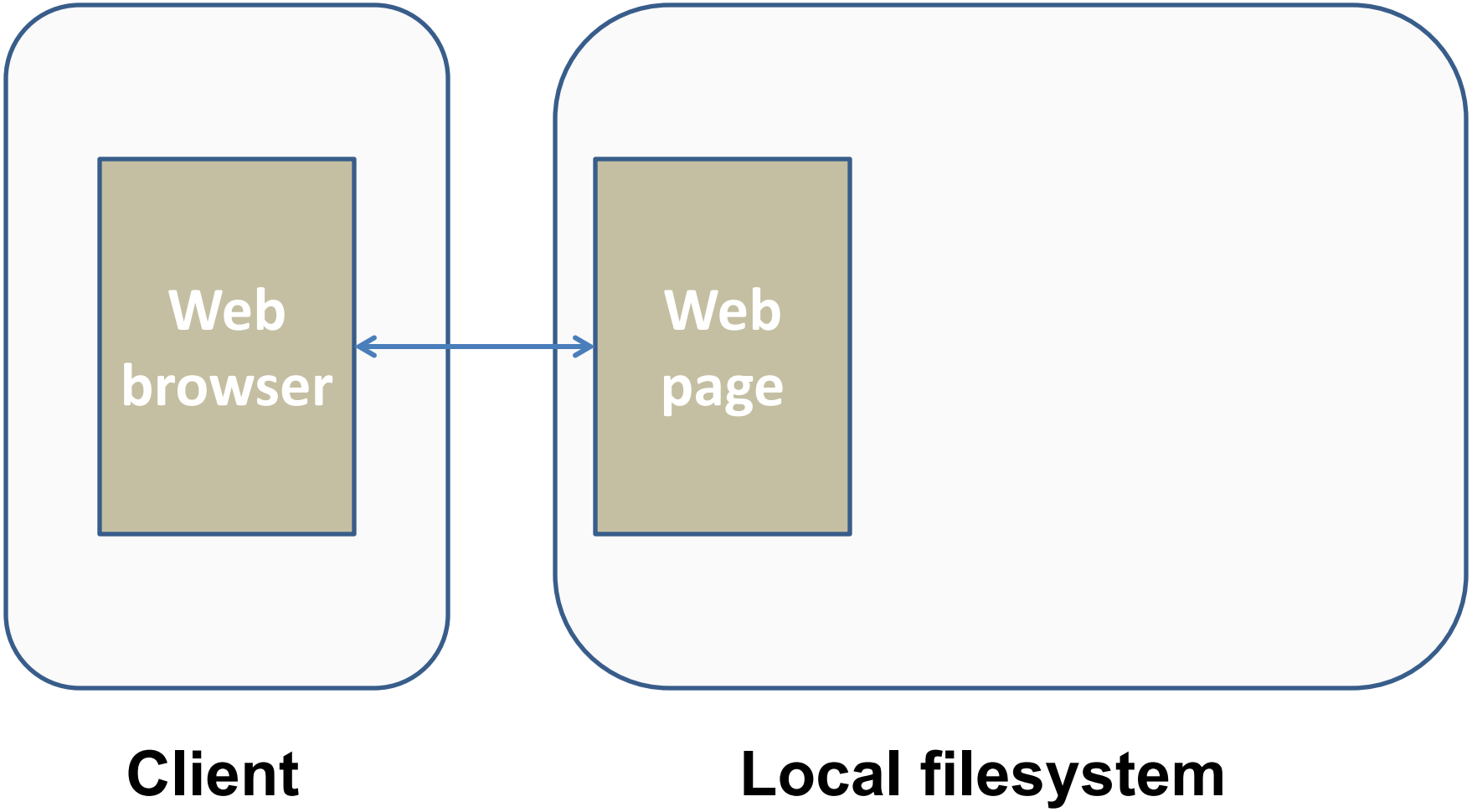
1. Through a dedicated application

Example: limmaGUI

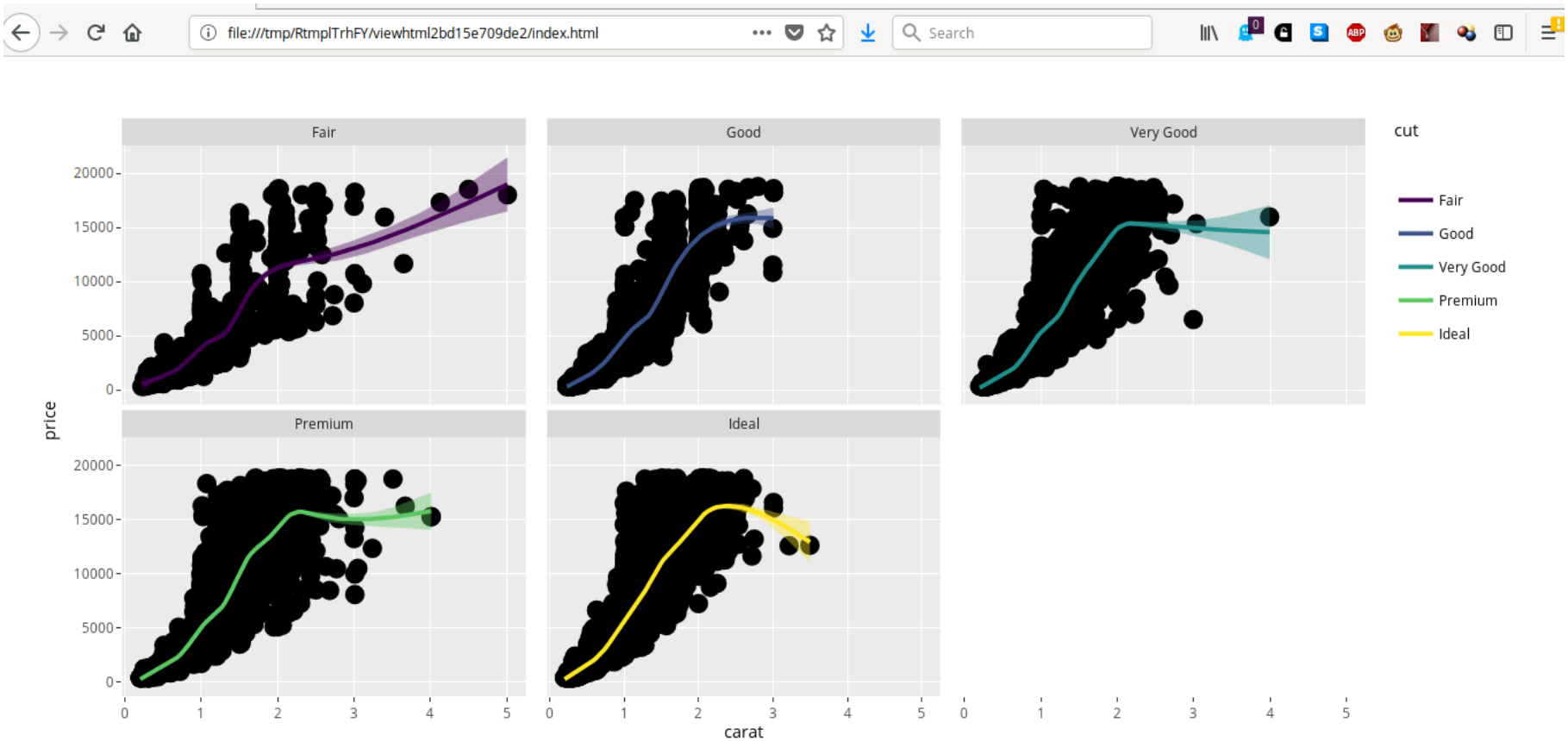


Block	Column	Row	ID	Name	M	A	t	P.Value	B
44	5	18	ML-DC	ML-DC_026E7	2.845	12.53	66.31	4.207e-14	31.79
44	15	5	ML-DC	ML-DC_021A15	2.837	8.996	66.13	4.36e-14	31.75
3	19	15	ML-DC	ML-DC_06F21F	2.562	8.631	62.31	9.568e-14	31.04
36	19	4	ML-DC	ML-DC_020I9	2.713	8.3	61.37	1.169e-13	30.86
36	7	3	ML-DC	ML-DC_020E13	2.718	11.68	61.14	1.230e-13	30.81
38	9	19	ML-DC	ML-DC_026K18	2.538	10.39	60.93	1.285e-13	30.77
11	7	8	ML-DC	ML-DC_04J3	2.541	11.54	60.33	1.464e-13	30.65
40	13	3	ML-DC	ML-DC_020E2	2.775	14.04	60.32	1.469e-13	30.65
25	3	13	ML-DC	ML-DC_014P23	2.739	12.45	60.09	1.544e-13	30.6
28	21	20	ML-DC	ML-DC_017M7	2.883	13.31	59.34	1.822e-13	30.45

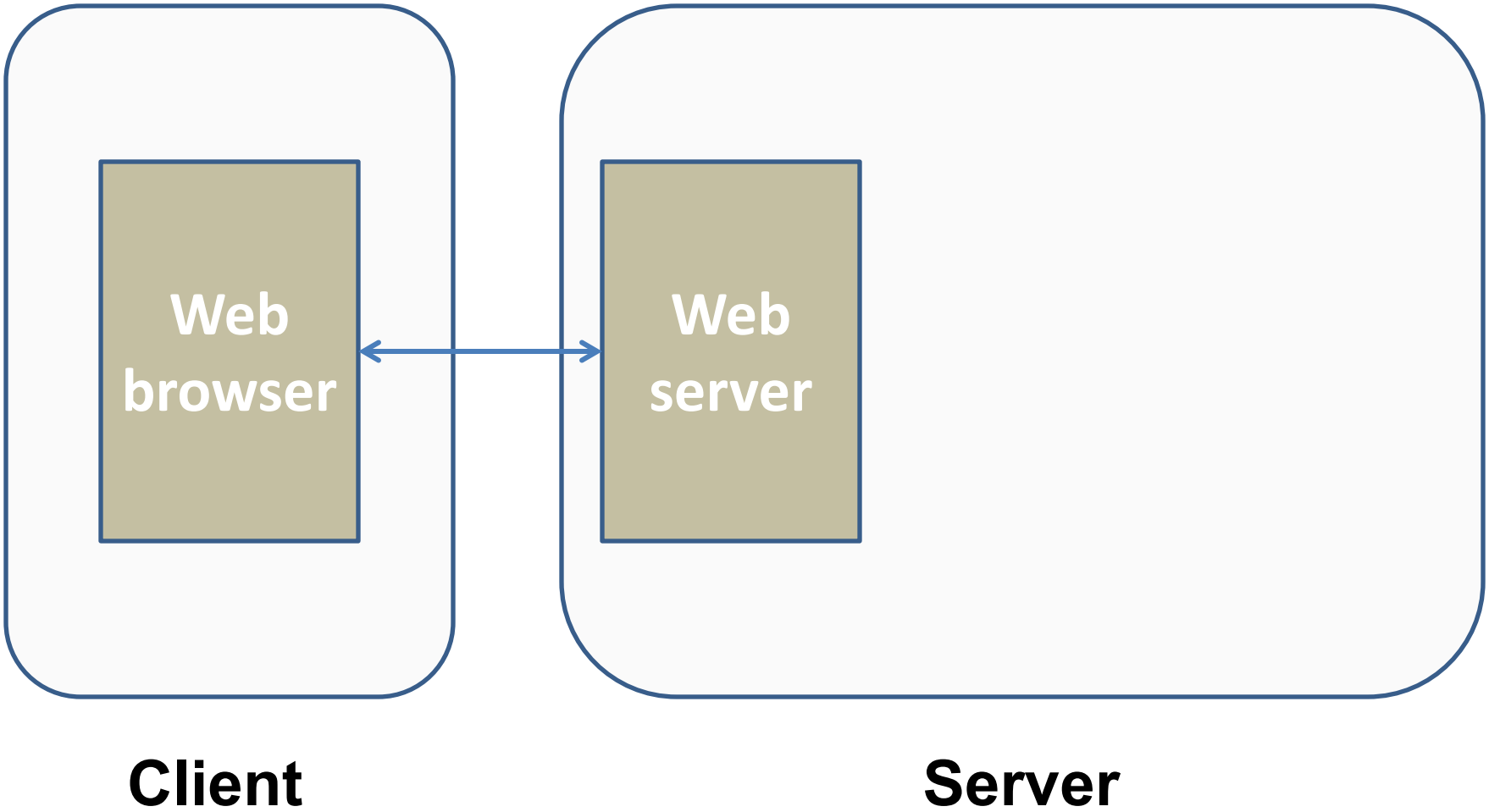
2. Through a web application

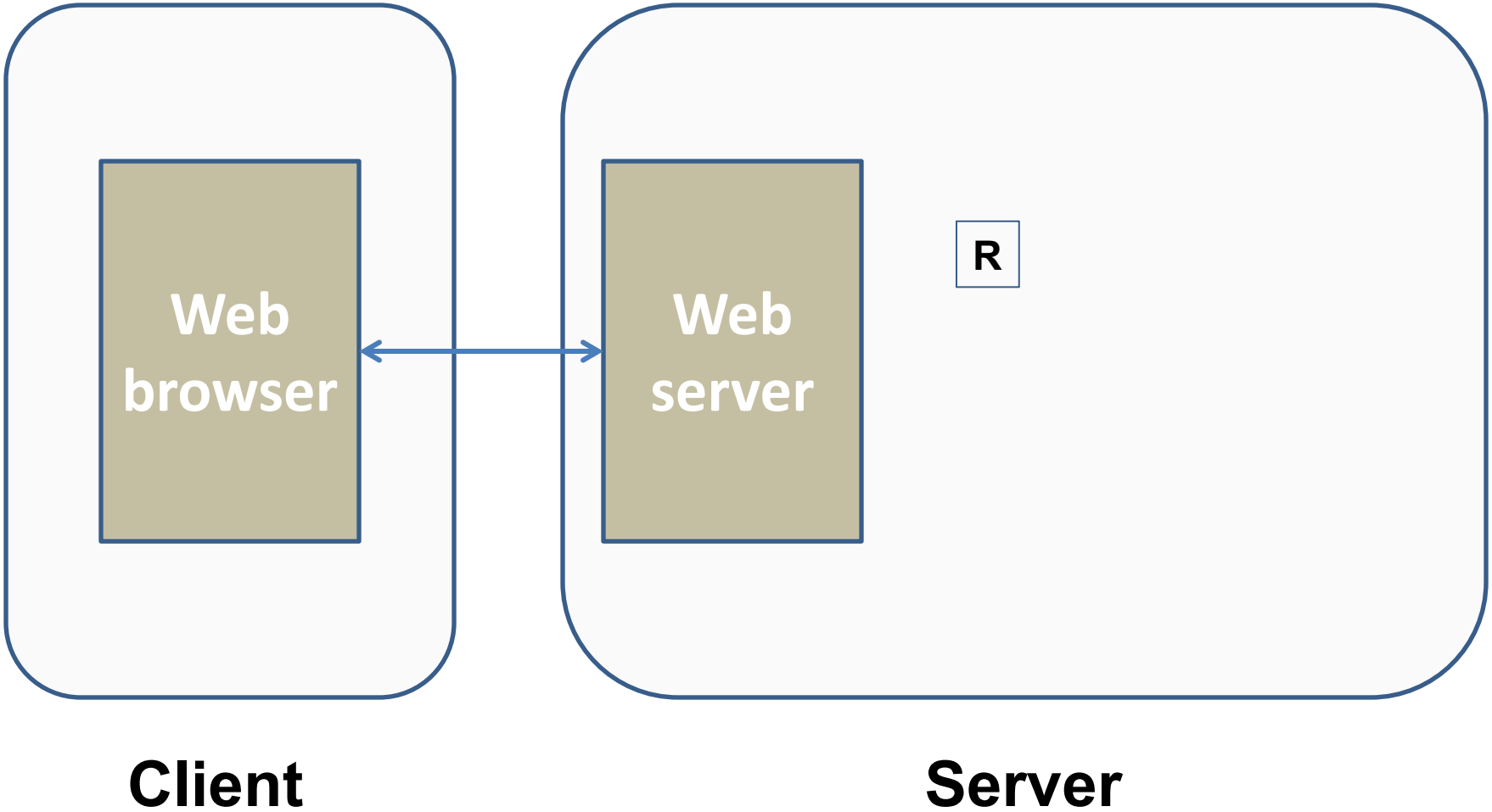

Web browser
Web page
Client
Local filesystem

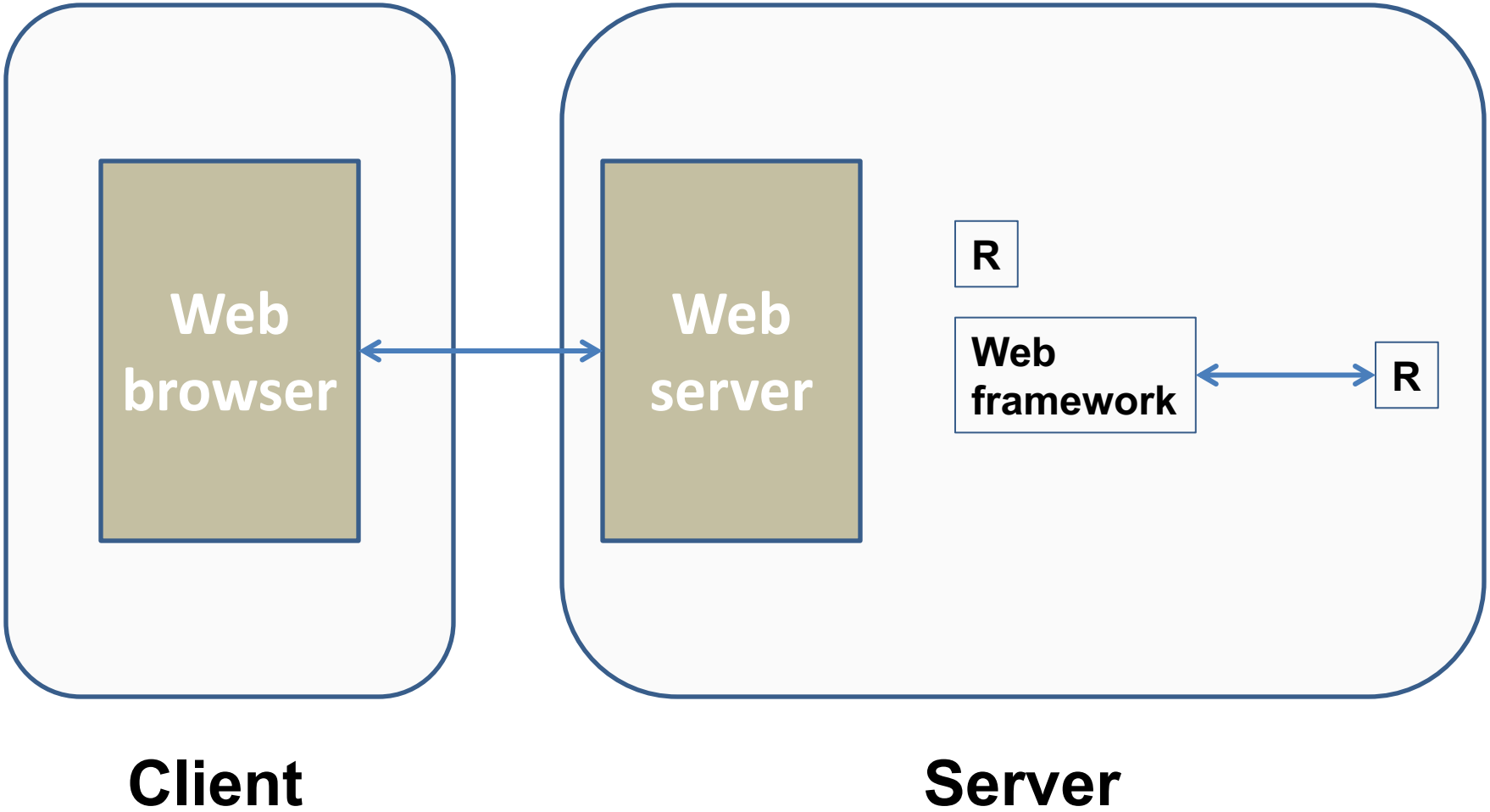
plotly



https://plot.ly/ggplot2/getting-started/


Web browser
Web server
Client
Server


Web browser
Web server
R
Client
Server


Web browser
Web server
R
Web framework
R
Client
Server

Statistical Analysis
On The Web

Rweb

To run **Rweb** just type the **R** (or Splus) code you want to execute into the text window and then click on the submit button. You will get a new html page with the text output of your code followed by the graphical output (if any) from your code.

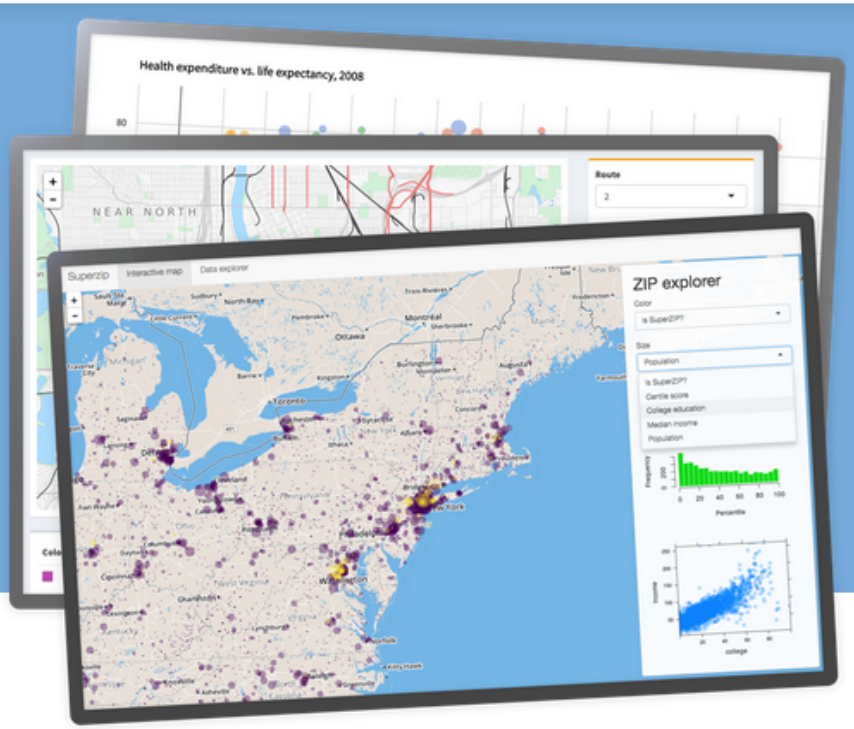
Below the submit button is a text area where you can enter the URL for a Web accessible dataset and a browse button for selecting a dataset on your computer. Either way, the dataset will be read in using **read.table** with **header=T** and stored in a dataframe called **X**. The dataframe, **X**, will then be attached so you can use the variable names. Eventually I hope to add several other options for data entry ... let me know if you have any suggestions.

If you use the back button on your browser to come back to this page you can modify your old code and then resubmit it, or you can clear the text area and type in all new code.

Shiny

from RStudio

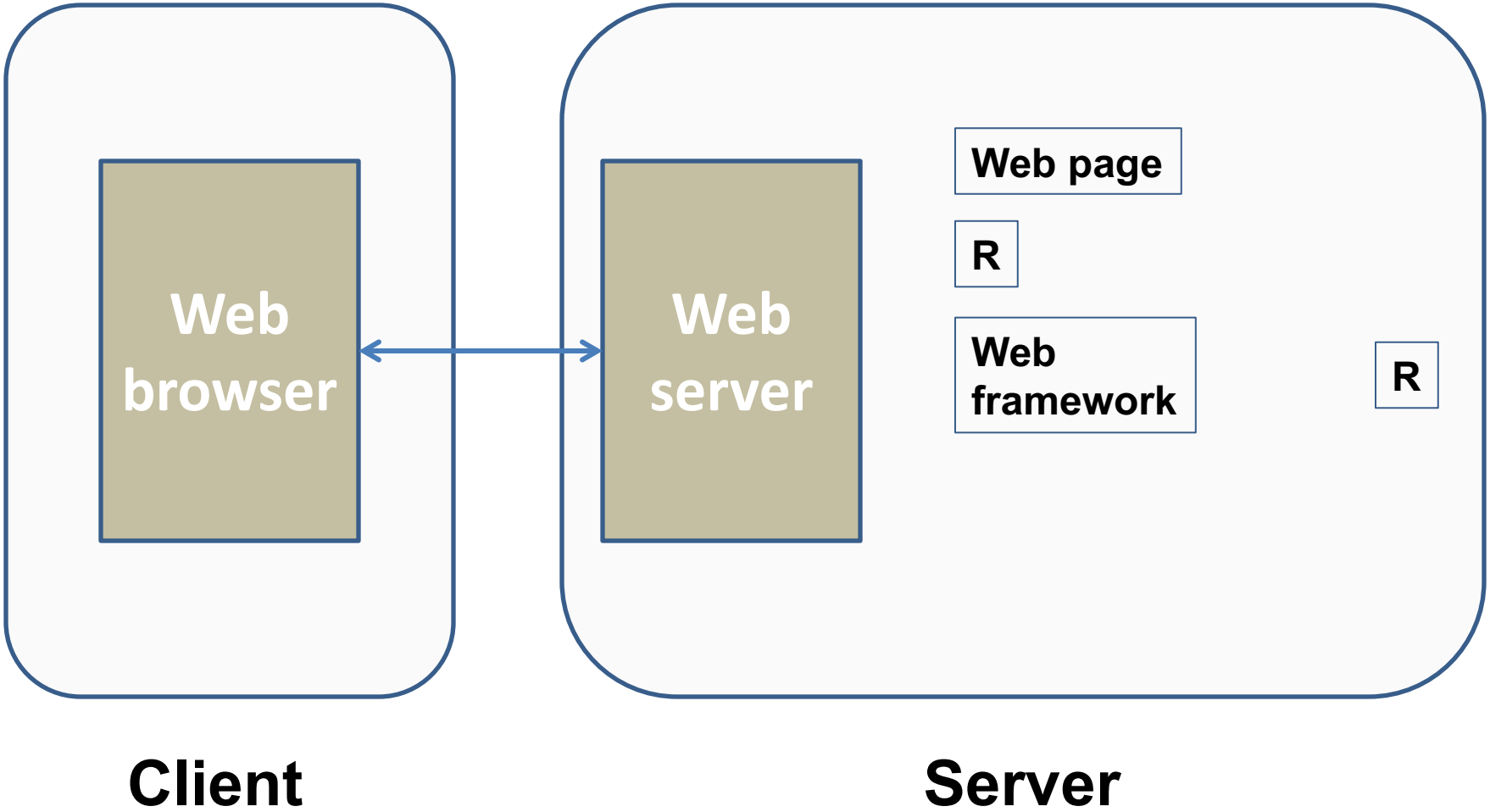
Get Started Gallery Articles Reference Deploy Help Contribute

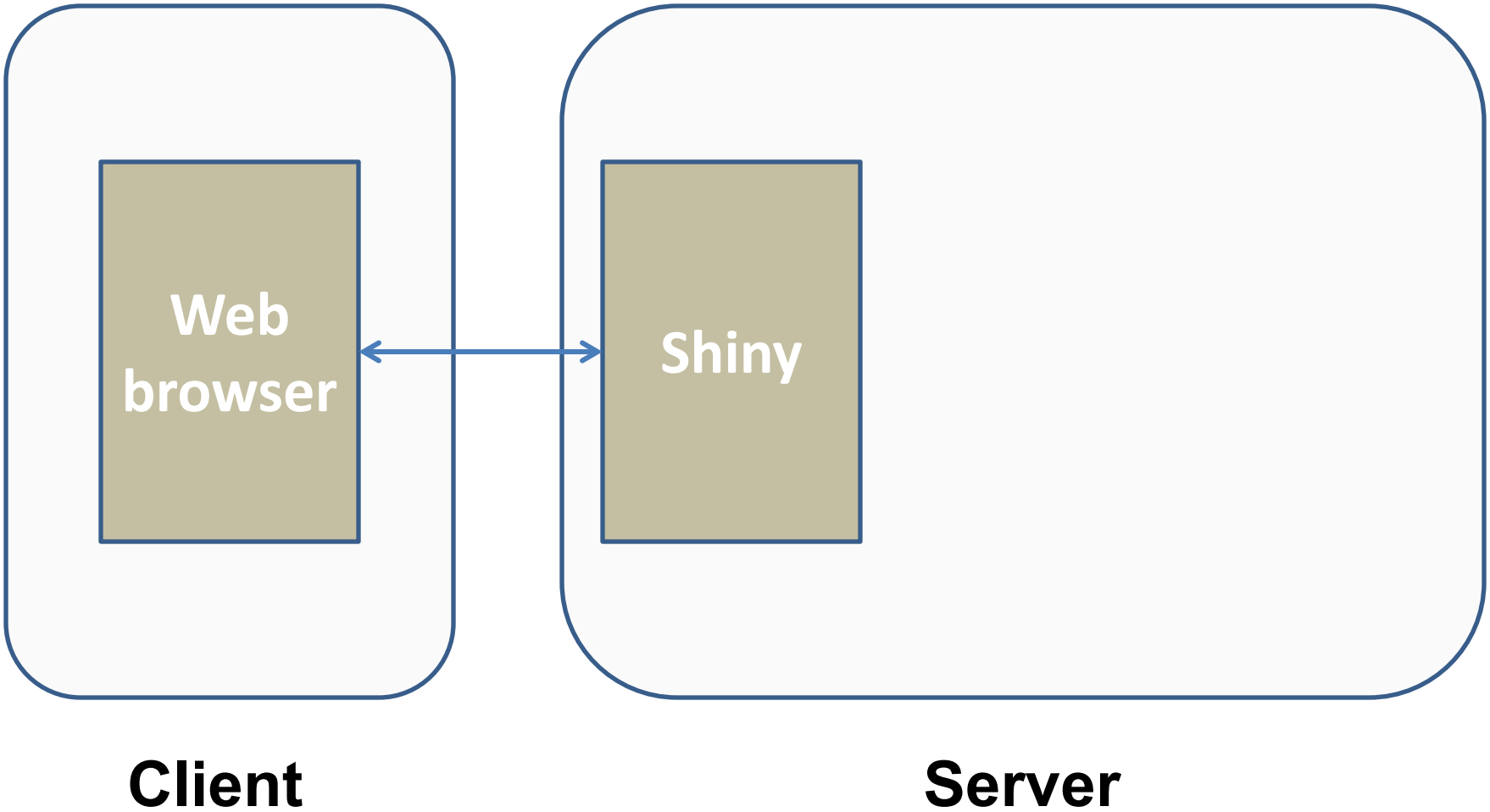


Interact. Analyze. Communicate.

Take a fresh, interactive approach to telling your data story with Shiny. Let users interact with your data and your analysis. And do it all with R.

https://shiny.rstudio.com/


Web browser
Web server
Web page
R
Web framework
R
Client
Server


Web browser
Shiny
Client
Server

" Shiny is an R package that makes it easy to build interactive web apps straight from R.

You can host standalone apps on a webpage or embed them in R Markdown documents or build dashboards.

You can also extend your Shiny apps with CSS themes, htmlwidgets, and JavaScript actions."

Documentation on Shiny

Shiny from RStudio

Get Started | Gallery | Articles | Reference | Deploy | Help | Contribute

Learn Shiny

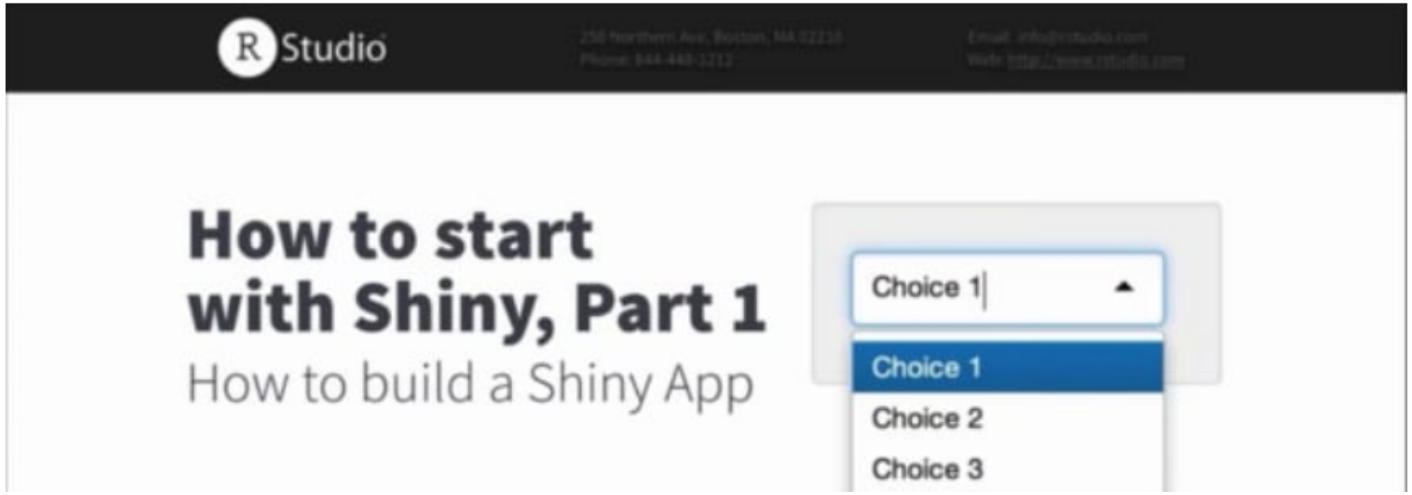
The video and written tutorials on this page are primarily designed for users who are new to Shiny and want a guided introduction.

If you use Shiny on a regular basis, you may want to skip these tutorials and visit the articles section where we cover individual Shiny topics at a more advanced level.

Video tutorials

How to Start Shiny tutorial

The How to Start Shiny video series will take you from R programmer to Shiny developer. Watch the complete tutorial, or jump to a specific chapter by clicking a link below. The entire tutorial is two hours and 25 minutes long. Download the slides and exercises here: Part 1, Part 2, and Part 3.



https://shiny.rstudio.com/tutorial/

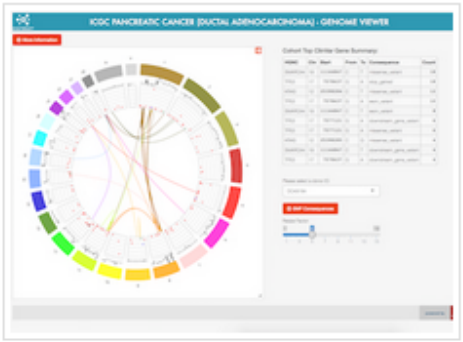
```
> install.packages("shiny")
> library(shiny)
```

Examples of Shiny apps

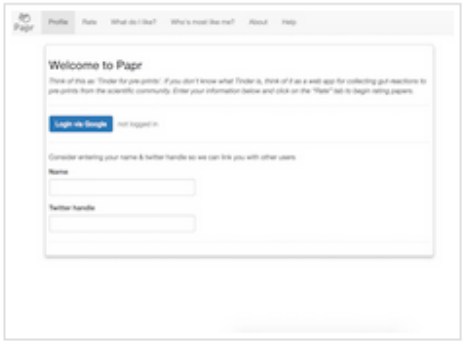
Gallery

Shiny User Showcase

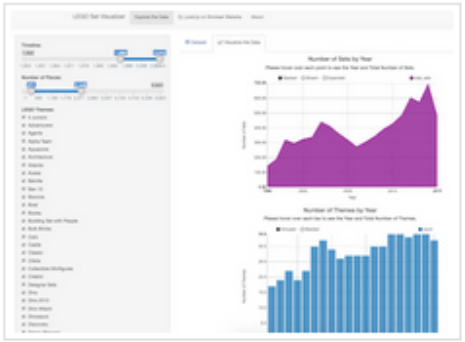
The Shiny User Showcase contains an inspiring set of sophisticated apps developed and contributed by Shiny users.



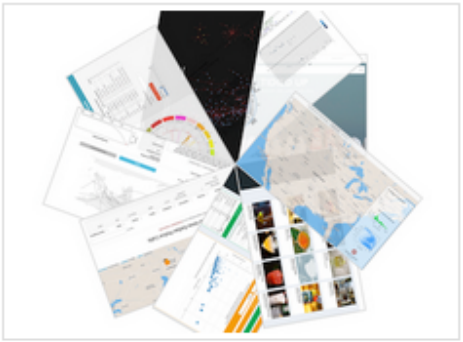
Genome browser



Papr



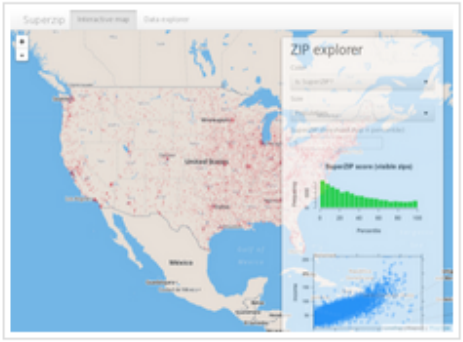
Lego Set Database Explorer



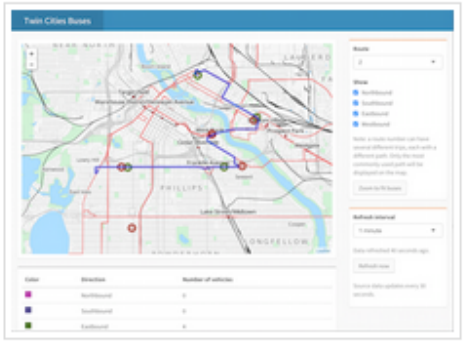
See more

Interactive visualizations

Shiny is designed for fully interactive visualization, using JavaScript libraries like d3, Leaflet, and Google Charts.

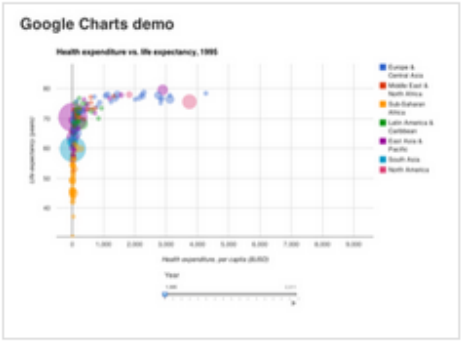


SuperZip example



Bus dashboard

Movie explorer



Google Charts

https://shiny.rstudio.com/gallery/

https://marionilab.cruk.cam.ac.uk/iSEE_allen/

Shiny includes 11 examples of apps

```
runExample("01_hello")       # a histogram
runExample("02_text")        # tables and data frames
runExample("03_reactivity") # a reactive expression
runExample("04_mpg")         # global variables
runExample("05_sliders")     # slider bars
runExample("06_tabsets")     # tabbed panels
runExample("07_widgets")     # help text and submit buttons
runExample("08_html")        # Shiny app built from HTML
runExample("09_upload")      # file upload wizard
runExample("10_download")    # file download wizard
runExample("11_timer")       # an automated timer
```

Creating my first Shiny app

A simple shiny app consists of two components:

- `ui` (*user interface*)
  creates the layout of the application and the user controls

- `server`
  creates the content based on the user's selections

One single file `app.R` (recommended)

```
ui <- fluidPage(

)

server <- function(input, output) {

}

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

Two files in the same folder (not recommended)

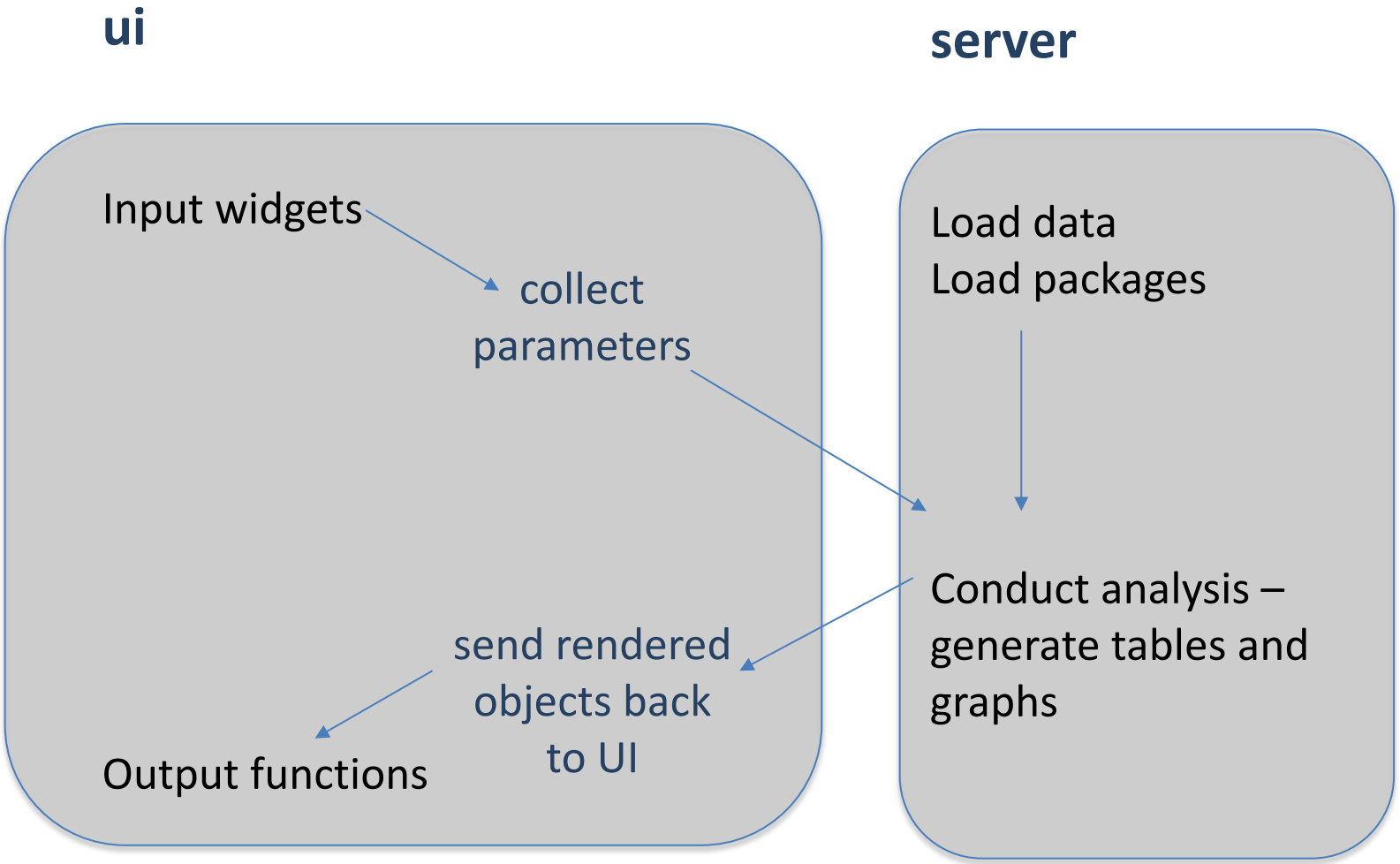
ui.R

```
shinyUI(fluidPage(

))
```

server.R

```
shinyServer(function(input, output) {

})
```

Anatomy of a shiny app: data flow



- Copy/paste the content of `app.R` or `source()` its content:

```
app <- source("app.R")
app
```

- Use the `runApp()` function:

```
runApp("test")          # Directory
runApp("test/app.R")    # File
```

- Use RStudio

- Run an app from github:
  ```
  runGitHub( repo="shiny", username="lecy")
  ```

- Run the app in "showcase mode"
  (run app and show source code)
  ```
  runApp("test", display.mode =
  "showcase")
  ```

Running the Shiny app

```
> library(shiny)
> app <- source("test1/app.R")
> app
$value

Listening on http://127.0.0.1:3625
^C
> runApp("test1/app.R")

Listening on http://127.0.0.1:3625
Warning in body(fun) : argument is not a function
^C

>
```

app

```
> li
> ap
> ap
$val

List
^C
> ru

List
Warn
^C

>
```

Mozilla Firefox

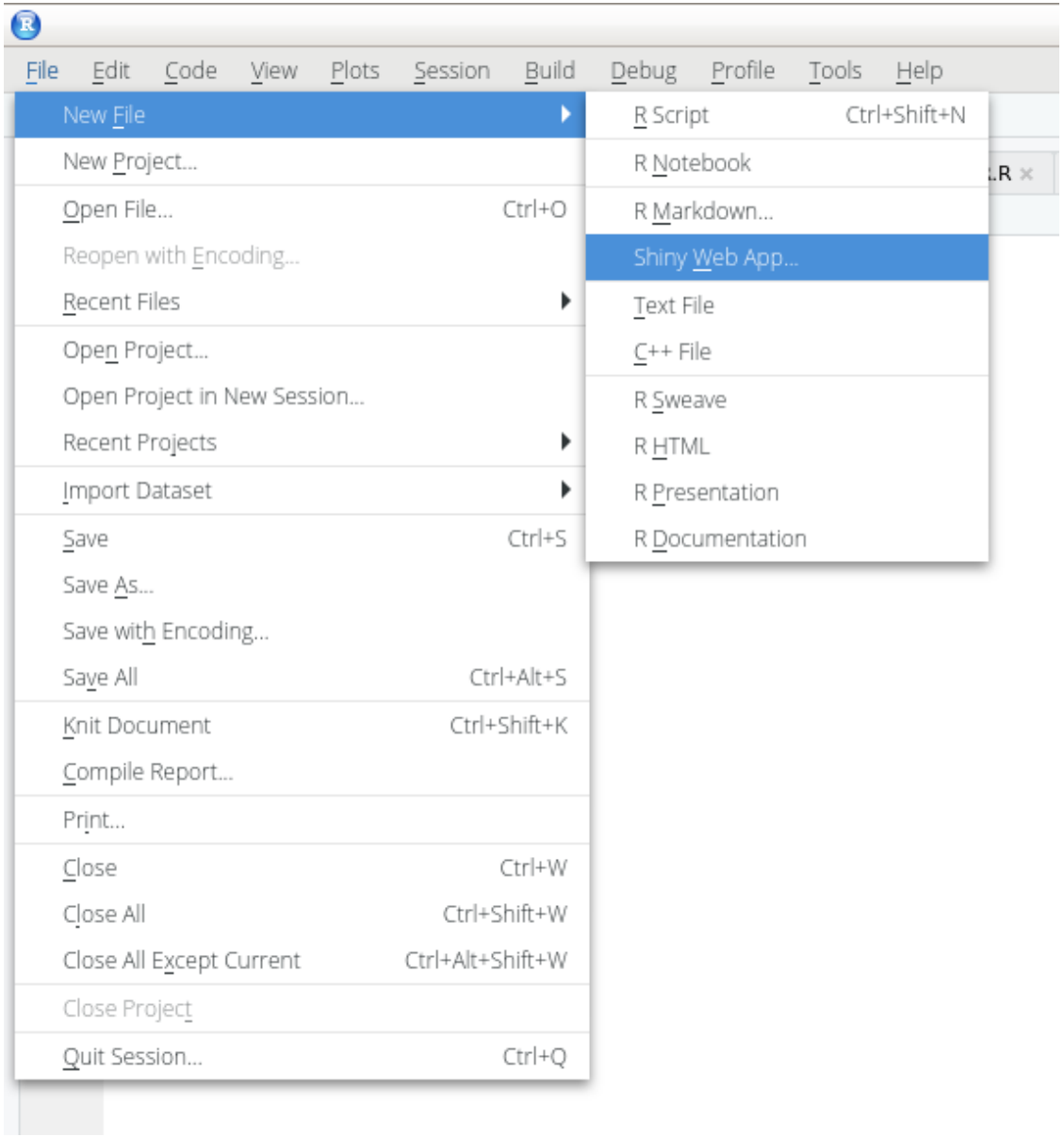
File Edit View History Bookmarks Tools Help

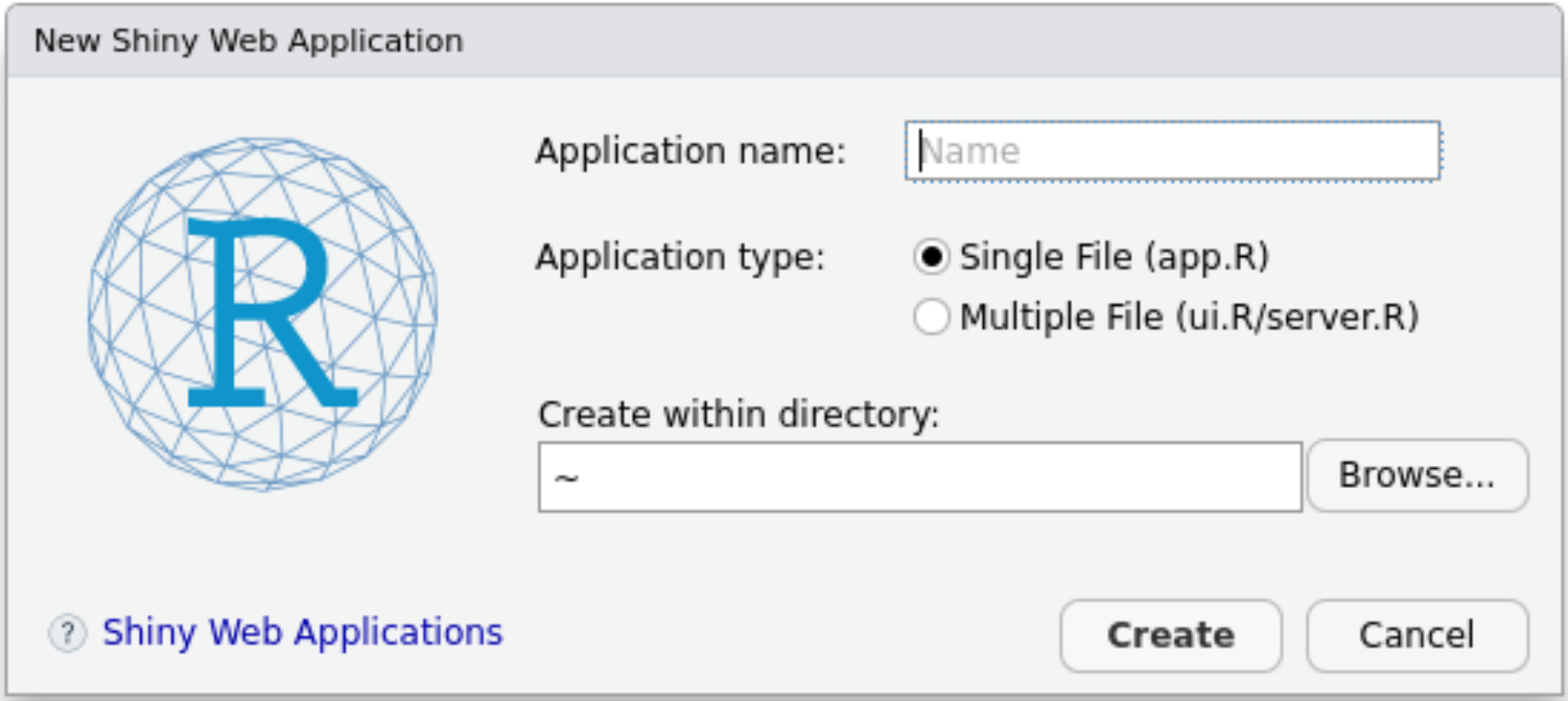
127.0.0.1:3625/

127.0.0.1:3625 110% Search

Using RStudio to create apps

Shiny in RStudio




New Shiny Web Application
Application name:
Name
Application type:
Single File (app.R)
Multiple File (ui.R/server.R)
Create within directory:
~
Browse...
Shiny Web Applications
Create
Cancel

Windows [Running] - Oracle VM VirtualBox : 1

RStudio

File Edit Code View Plots Session Build Debug Profile Tools Help

Go to file/function | Addins | Project: (None)

dataAKT | qPCR.R | test.Rmd | knitr-demo.Rmd | Untitled7* | Untitled9* | Untitled10* | Untitled | app.

Run App

```
#
# This is a Shiny web application. You can run the application by clicking
# the 'Run App' button above.
#
# Find out more about building applications with Shiny here:
#
#    http://shiny.rstudio.com/
#

library(shiny)

# Define UI for application that draws a histogram
ui <- fluidPage(

   # Application title
```

43:1 server(input, output) R Script

Console Terminal

~/Documents/sib/20190709-shiny/app1/

```
>
>
>
>
>
```

Environ | Files

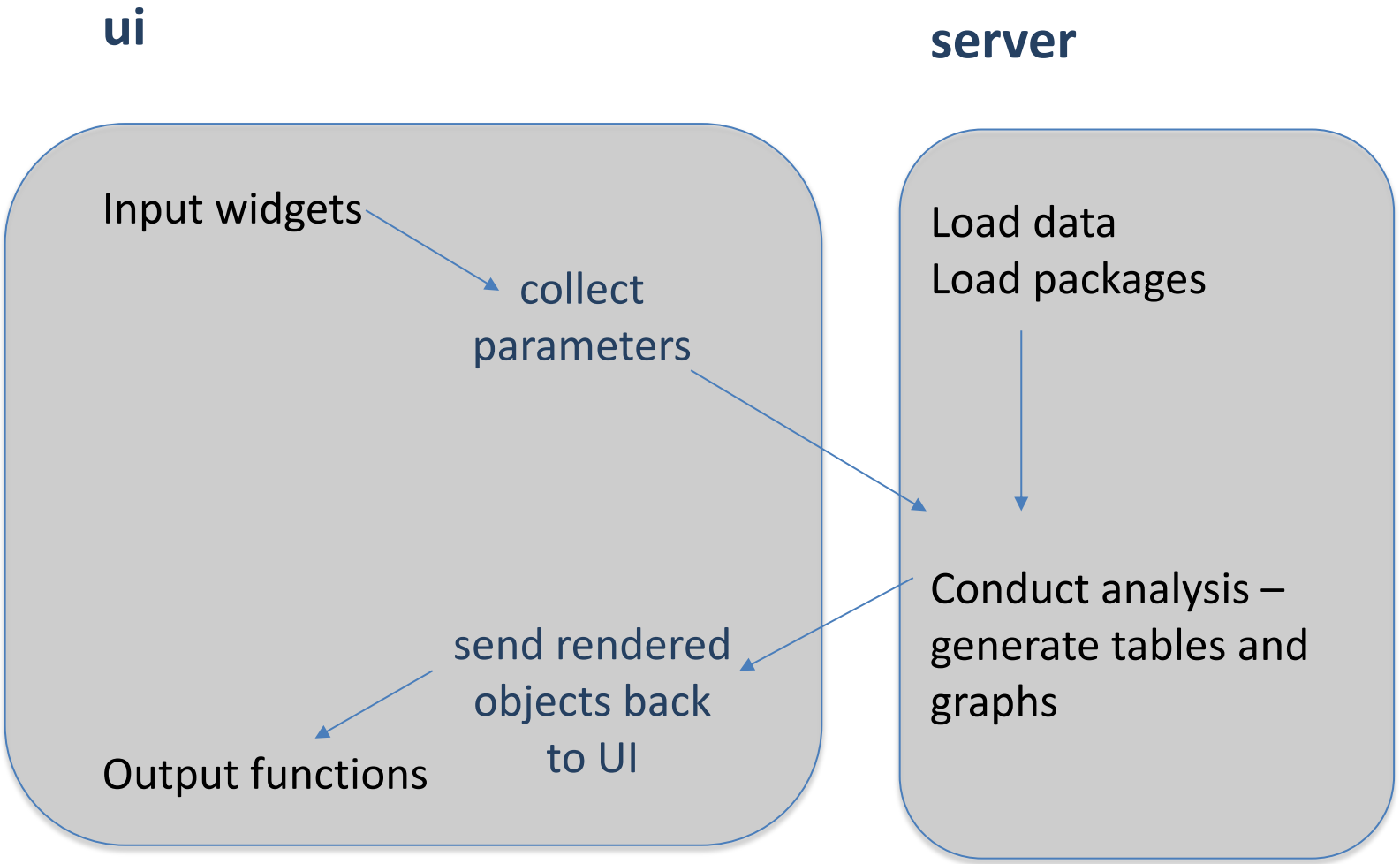
runA
{shi

Ru
Sh
Ap

Des

Runs
a
Shin
appl
This
func
norm
does
not
retur
inter
R

Anatomy of a shiny app: data flow



How do the ui and the server communicate= ?

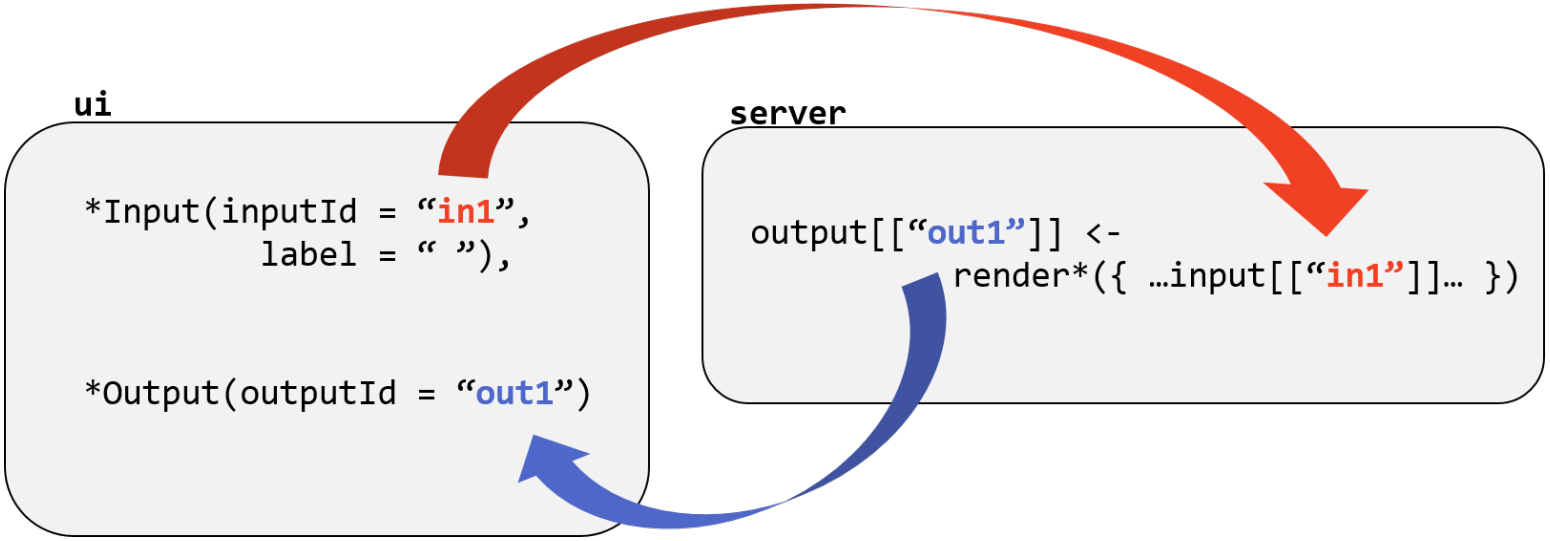
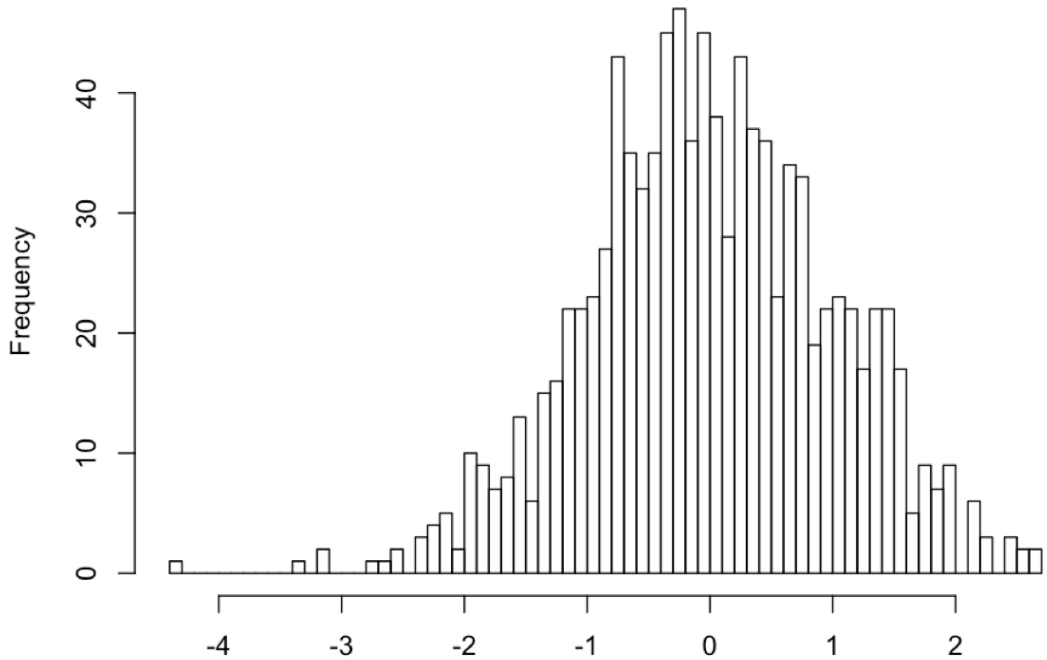


Image Credit: National Socio-Environmental Synthesis Center
https://sesync-ci.github.io/basic-Shiny-lesson/

My first (non trivial) Shiny app: generating random number and plotting and histogram

Number of observations

1000



```
ui <- fluidPage(

)

server <- function(input, output) {

}
```

```
ui <- fluidPage(
```

Most shiny apps use fluidPage():
Creates a page with rows (for element alignment), which contain columns

Fluid pages adapt the size of components according to available space (browser width)

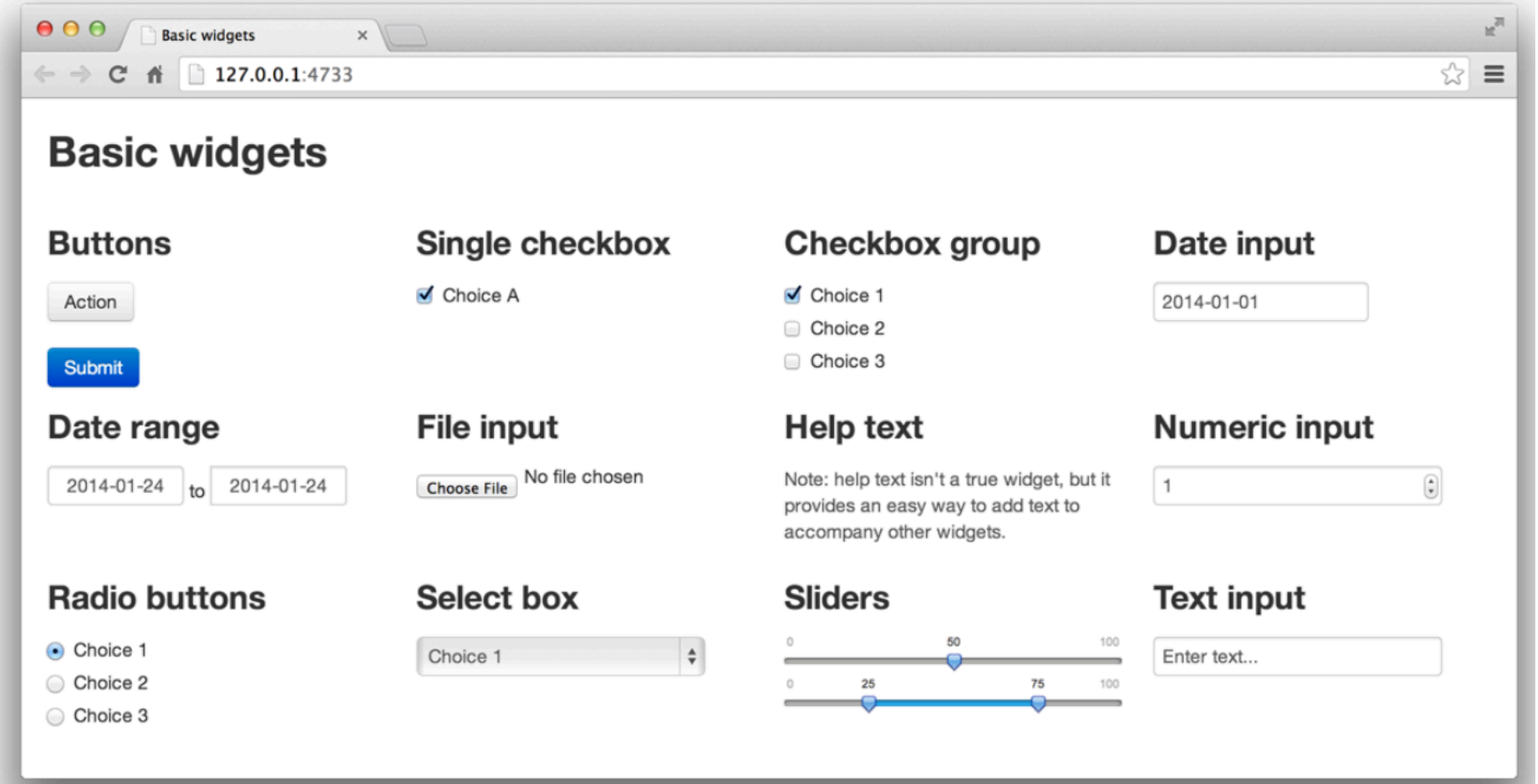
```
)

server <- function(input, output) {

}
```

```
ui <- fluidPage(
  titlePanel("Workshop - Example 1 - Basic Histogram"),
  sidebarLayout(
    sidebarPanel(
       numericInput(inputId="n",
                    label="Number of observations",
                    value=1000) ),
       mainPanel(plotOutput("plot"))
  )
)

server <- function(input, output) {
   data <- reactive({
                     x <- rnorm(input$n)
                     x })

   output$plot <- renderPlot({
                      hist(data(), 50, main="",  xlab="x")
                    })
}
```

Standard shiny widgets (inputs)

Function Name	**Widget**
actionButton	Action Button
checkboxGroupInput	A group of check boxes
checkboxInput	A single check box
dateInput	A calendar to aid date selection
dateRangeInput	A pair of calendars for selecting a date range
fileInput	A file upload control wizard
helpText	Help text that can be added to an input form
numericInput	A field to enter numbers
radioButtons	A set of radio buttons
selectInput	A box with choices to select from
sliderInput	A slider bar
submitButton	A submit button
textInput	A field to enter text

Standard shiny widgets (inputs)



Standard shiny widgets (inputs)

Function Name	Widget
actionButton	Action Button
checkboxGroupInput	A group of check boxes
checkboxInput	A single check box
dateInput	A calendar to aid date selection
dateRangeInput	A pair of calendars for selecting a date range
fileInput	A file upload control wizard
helpText	Help text that can be added to an input form
numericInput	A field to enter numbers
radioButtons	A set of radio buttons
selectInput	A box with choices to select from
sliderInput	A slider bar
submitButton	A submit button
textInput	A field to enter text

Note each function will store different input values:

textInput = a single character element

selectInput = character elements from a list

sliderInput = two numbers in a range

checkboxInput = T / F

Widget components

Each widget function requires several arguments. The first two arguments for each widget are

- **Name for the widget. The user will not see this name, but you can use it to access the widget's value. The name should be a character string.**
- **Label. This label will appear with the widget in your app. It should be a character string, but it can be an empty string "".**

How you will access the data on server.R:

Input$name

actionButton("submit", label = "Submit Your Form")

Creates an entry at Input$submit

The remaining arguments vary from widget to widget, depending on what the widget needs to do its job.

numericInput("numInput", "A numeric input:", value = 7, min = 1, max = 30)

- `textInput(“nam”,”Name”)`
- `selectInput(“gen”,”Gender”, choices=c(“Male”,”Female”), selected=“Female”)`
- `sliderInput(“age”,”Ages”, min=0, max=100, value=20, step = 1)`
- `radioButtons(“grad", “Grade”, choices =c(“A”,”B”,”C”,”D”,”F”,”W”), selected = “A”)`

More examples can be found at http://shiny.rstudio.com/gallery/widget-gallery.html

Output functions:

The output functions take R code and "render" it as HTML objects that can be used in web browsers in order to display your dashboard. They just translate from R to HTML.

Output Function	Creates
htmlOutput	raw HTML
imageOutput	image
plotOutput	plot
tableOutput	table
textOutput	text
uiOutput	raw HTML
verbatimTextOutput	text

How do the ui and the server communicate= ?

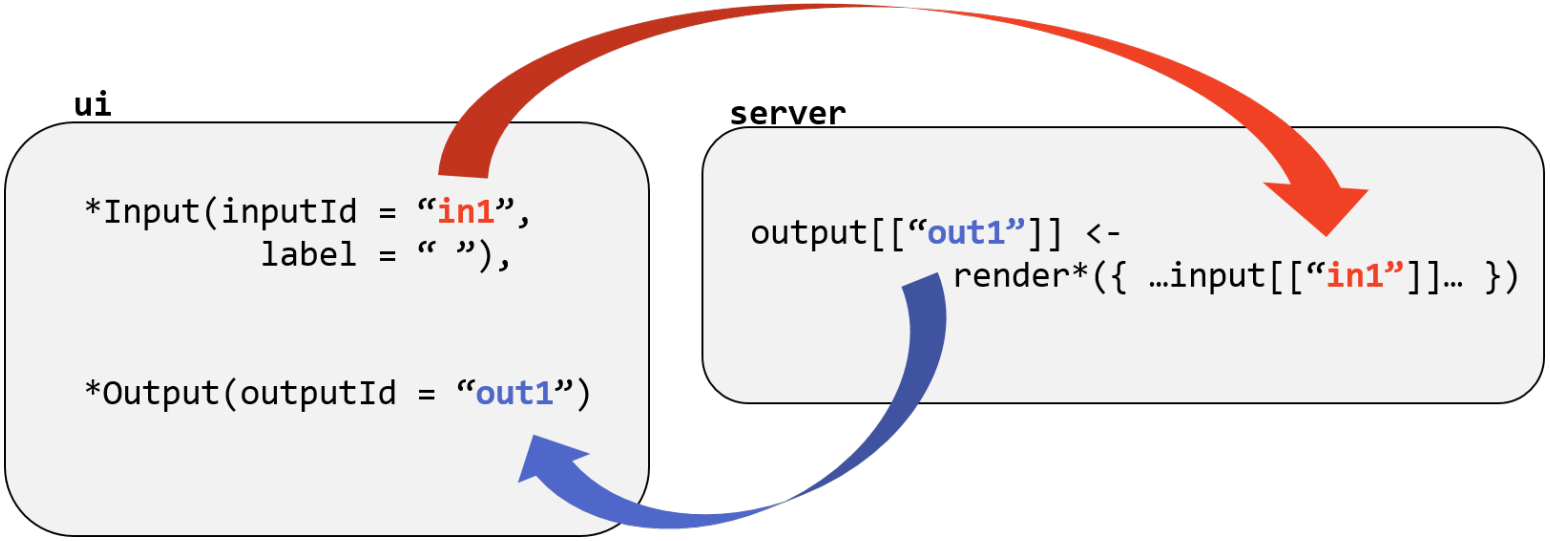
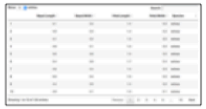
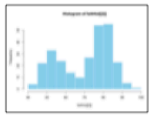


Image Credit: National Socio-Environmental Synthesis Center
https://sesync-ci.github.io/basic-Shiny-lesson/

```
ui <- fluidPage(
  titlePanel("Workshop - Example 1 – Basic  Histogram"),
  sidebarLayout(
    sidebarPanel(
       numericInput(inputId="n",
                    label="Number of observations",
                    value=1000) ),
       mainPanel(plotOutput("plot"))
  )
)

server <- function(input, output) {
   data <- reactive({
                     x <- rnorm(input$n)
                     x })

   output$plot <- renderPlot({
                      hist(data(), 50, main="",  xlab="x")
                    })
}
```

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

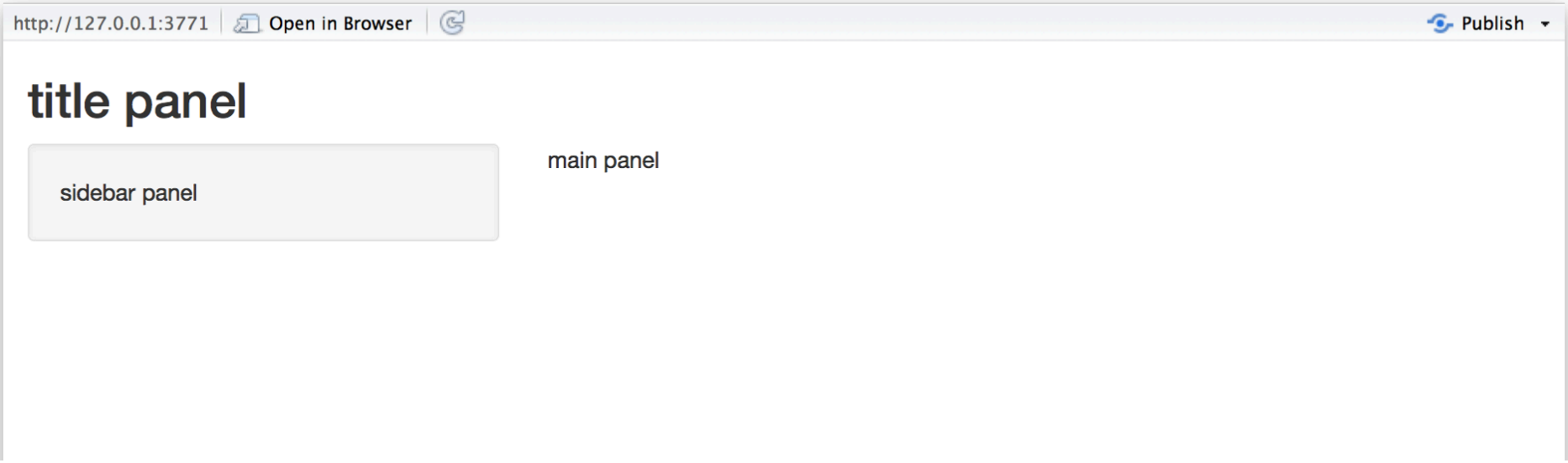
Server		User Interface
	DT::**renderDataTable**(expr, options, callback, escape, env, quoted)	**dataTableOutput**(outputId, icon, …)
	renderImage(expr, env, quoted, deleteFile)	**imageOutput**(outputId, width, height, click, dblclick, hover, hoverDelay, hoverDelayType, brush, clickId, hoverId, inline)
	**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, …)

works with

More on User Interface

User Interface

```
ui <- fluidPage(
  titlePanel("title panel"),

  sidebarLayout(
    sidebarPanel("sidebar panel"),
    mainPanel("main panel")
  )
)
```



Customize the ui with HTML/CSS

shiny functions that build your user interface are creating HTML code under the hood

```
> fluidPage()
<div class="container-fluid"></div>
```

Number of bins in histogram (approximate):

50

R Code

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

HTML

```
<div class="form-group shiny-input-container">
  <label class="control-label" for="n_breaks">Number of bins in histogram (approximate):</label>
  <div>
    <select id="n_breaks"><option value="10">10</option>
<option value="20" selected>20</option>
<option value="35">35</option>
<option value="50">50</option></select>
    <script type="application/json" data-for="n_breaks" data-nonempty="">{}</script>
  </div>
</div>
>
```

Customize the ui with HTML/CSS

Tweaking or adding HTML/CSS content can customize the look of your app

- Change the relative size of panels
- Add additional elements (e.g. headers, paragraphs, links to other content)
- Change colors, fonts, and spacing

Customize the ui with HTML/CSS

- Adding elements:
  - HTML elements are building blocks used to create webpages
  - Elements are surrounded by “tags”
    - I will become a paragraph.
    - <h1>I will become a large header.</h1>
  - shiny has 110 functions, called tags functions, that create HTML elements
    - tags$p(‘I will become a paragraph.’)
    - tags$h1(‘I will become a large header.’)
  - The most common tags functions have helper functions where you can drop the ‘tags$’ part
    - p(‘I will become a paragraph.’)
    - h1(‘I will become a large header.’)

Customize the ui with HTML/CSS

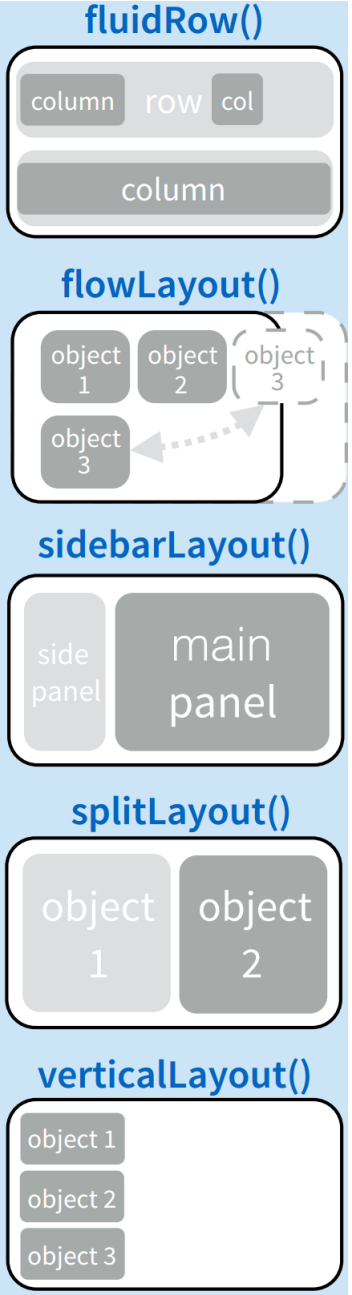
```r
library(shiny)
fluidPage(
  titlePanel("Hello Shiny!"),
  sidebarLayout(
    sidebarPanel(HTML('<p>
         <label>A numeric input:</label><br />
         <input type="number" name="n" value="7" min="1" max="30" />
         </p>')),
    mainPanel(
      p(strong("bold font "), em("italic font")),
      p(code("code block")),
      a(href="http://www.google.com", "link to Google"))
  )
)
```

User interface Layouts

- titlePanel and sidebarLayout create a basic layout for your Shiny app, but you can also create more advanced layouts.

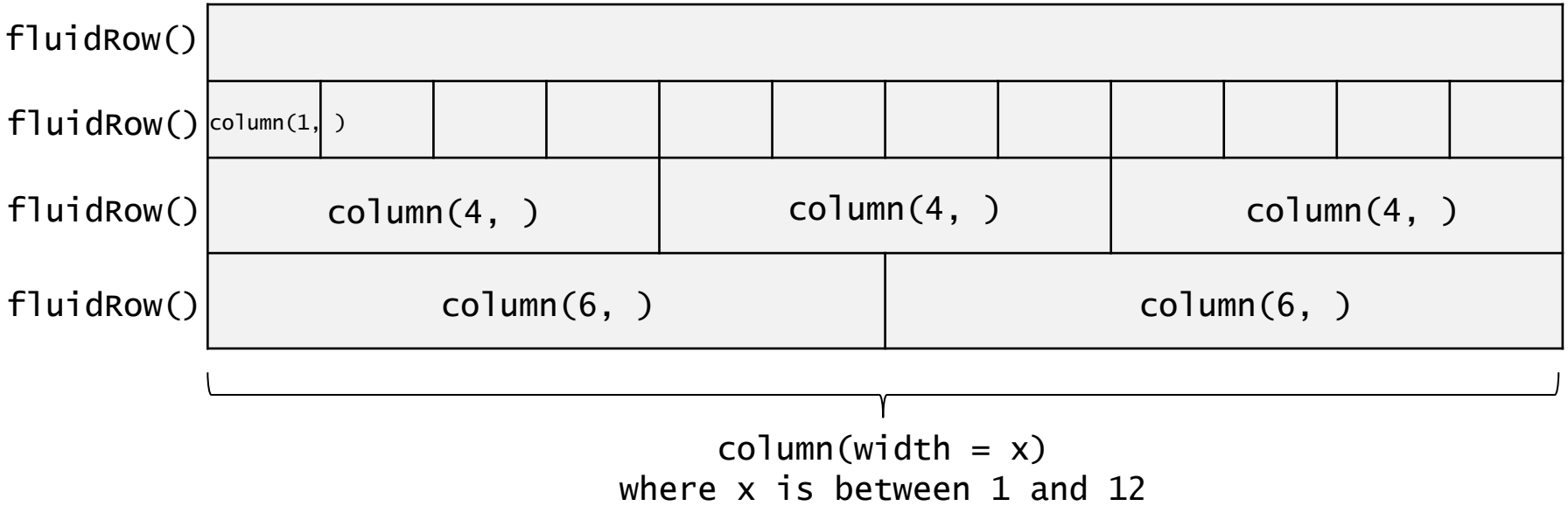
- You can use navbarPage or shinydashboard to give your app a multi-page user interface that includes a navigation bar or side.

- Or you can use fluidRow and column to build your layout up from a grid system.



fluidRow: Grid layout

- Tweaking lay-outs:
  - Utilize the fluid “shiny grid system”
  - Rows sub-divided into 12 columns
  - Based on popular HTML framework called “Bootstrap”



A vous de jouer:

Exercice 1