R Shiny简介

温灿红

中国科学技术大学管理学院

大纲

- 初识R Shiny
- 用户界面/前端
- 后台端
- 分享网页

初识R Shiny

为什么需要R Shiny

- **Shiny**是一种使用R语言创建交互式网页应用的开发框架。
- 可使得R的使用者不必太了解HTML, CSS或者JavaScript就可以快速完成非常复杂的闪亮的网页应用程序的开发，极大解放了作为R语言的生产力。
- 使得非传统程序员的R使用者不必依赖于前端、后端工程师就可以自己依照业务完成一些简单的数据可视化工作，快速验证想法的可靠性。
- Shiny并不限制你创建什么样的应用程序：它的用户界面组件可以轻松定制或扩展，它的服务器使用反应式编程，让你可以创建任何类型的后端逻辑。

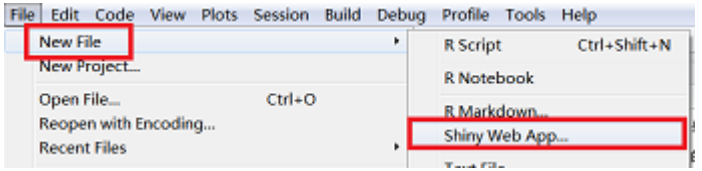
什么是R Shiny

- 简单而言，Shiny其实是一个R包，帮助你很容易产生丰富、交互的网页应用。
- https://statsoft.shinyapps.io/hello/
- 参考书：《Mastering Shiny: Build Interactive Apps, Reports, and Dashboards Powered by R》
- 网上链接：https://mastering-shiny.org/index.html

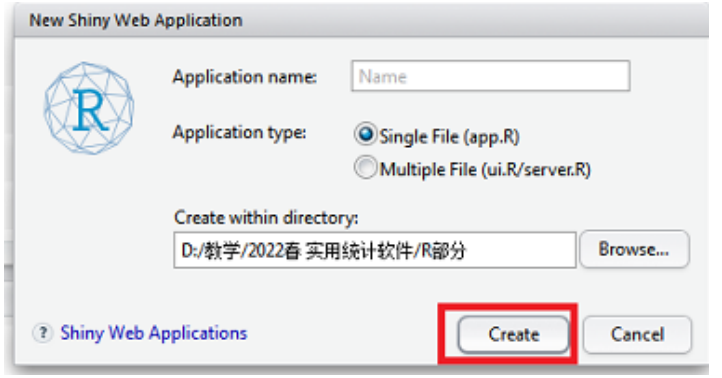
```
# install.package(shiny)
library(shiny)
```

新建Shiny的流程一

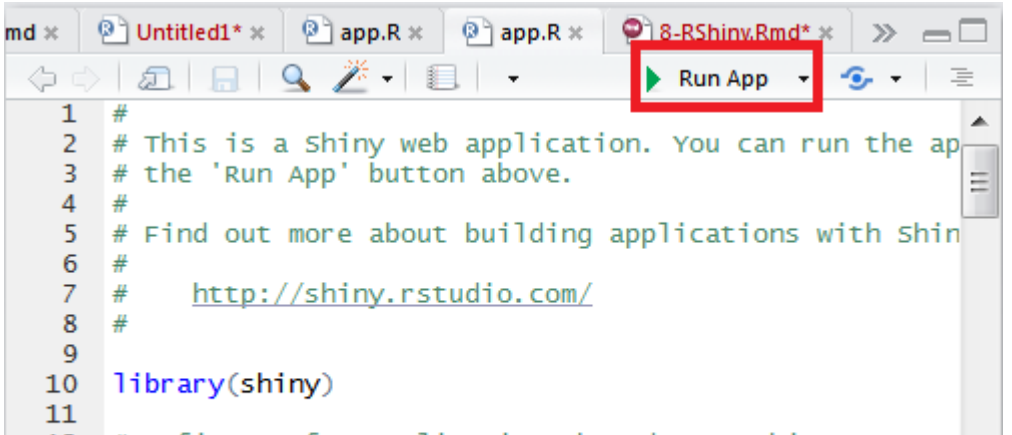
1.打开**RStudio**，点击File-> New File->"Shiny Web App..."新建一个Shiny。



2.起名并指定存储位置，点击"Create"即可。

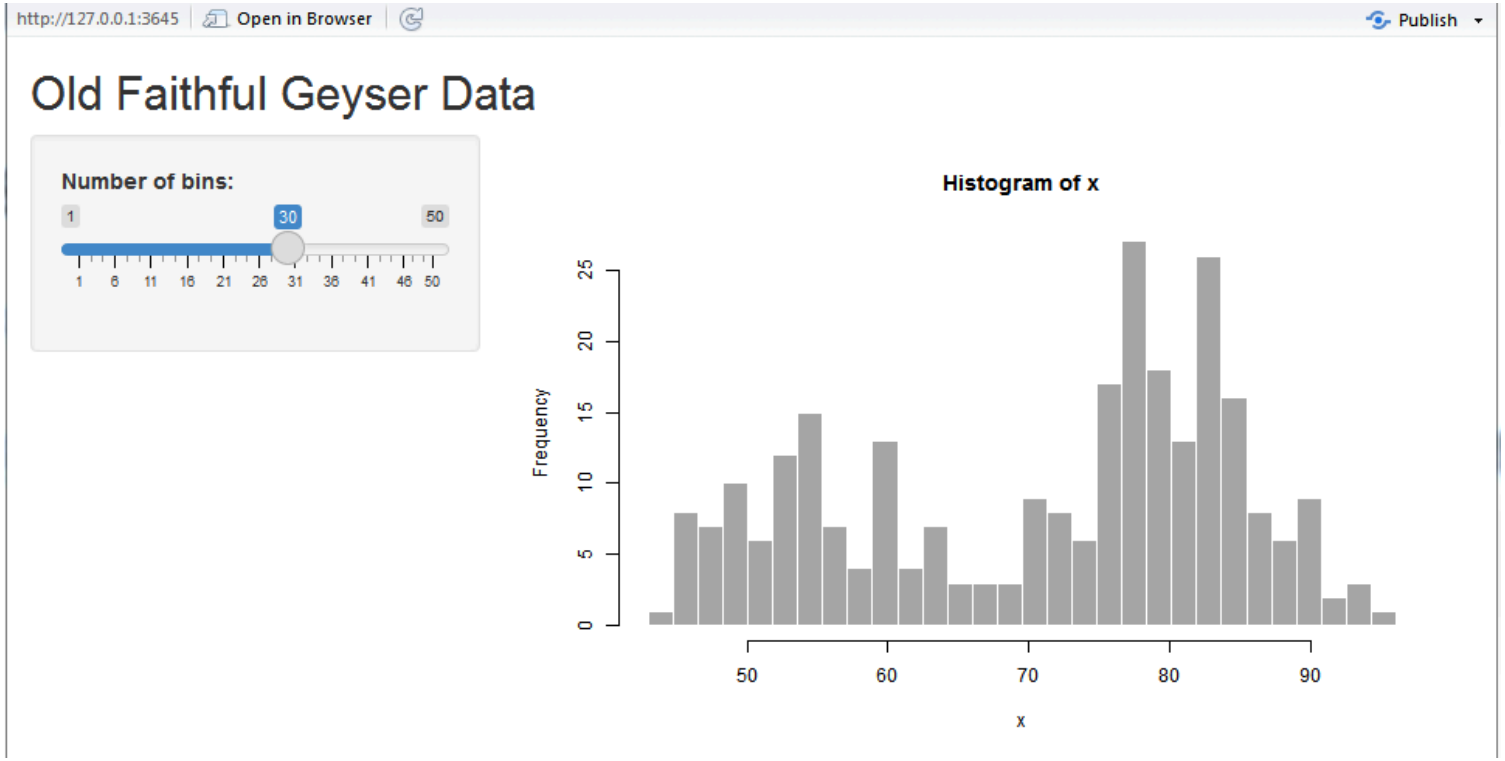


3.这时候有一个基本的样例文件"app.R"， 点击"Run App"即可编译得到网页。



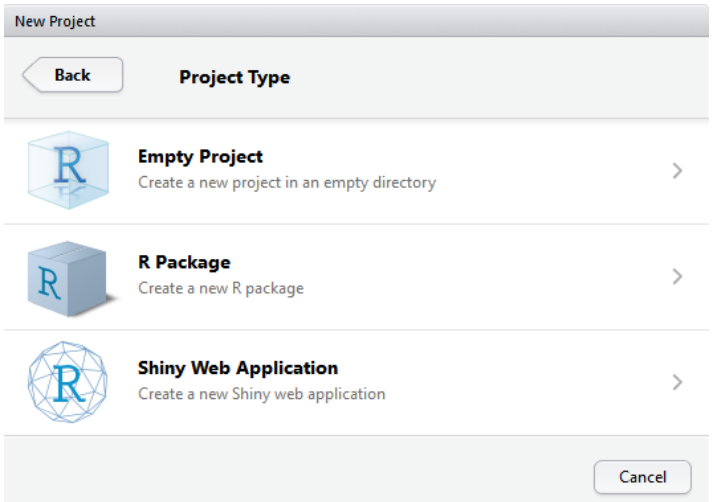
新建Shiny的流程二

- 大功告成!

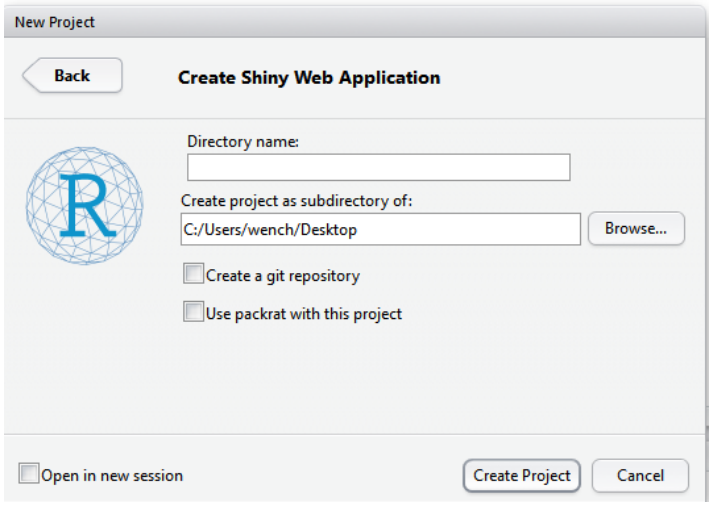


新建Shiny项目的流程

1.打开**RStudio**，点击File -> New Project...-> New Directory -> Shiny Web Application 新建一个Shiny项目。



2.起名并指定存储位置，点击"Create Project"即可。



我的第一个Shiny 网页

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

    # Application title
    titlePanel("Old Faithful Geyser Data"),

    # Sidebar with a slider input for number of bins
    sidebarLayout(
        sidebarPanel(
            sliderInput("bins",
                        "Number of bins:",
                        min = 1,
                        max = 50,
                        value = 30)
        ),

        # Show a plot of the generated distribution
        mainPanel(
           plotOutput("distPlot")
        )
    )
)
```

我的第一个Shiny 网页

```
# Define server logic required to draw a histogram
server <- function(input, output) {

    output$distPlot <- renderPlot({
        # generate bins based on input$bins from ui.R
        x    <- faithful[, 2]
        bins <- seq(min(x), max(x), length.out = input$bins + 1)

        # draw the histogram with the specified number of bins
        hist(x, breaks = bins, col = 'darkgray', border = 'white',
             xlab = 'Waiting time to next eruption (in mins)',
             main = 'Histogram of waiting times')
    })
}

# Run the application
shinyApp(ui = ui, server = server)
```

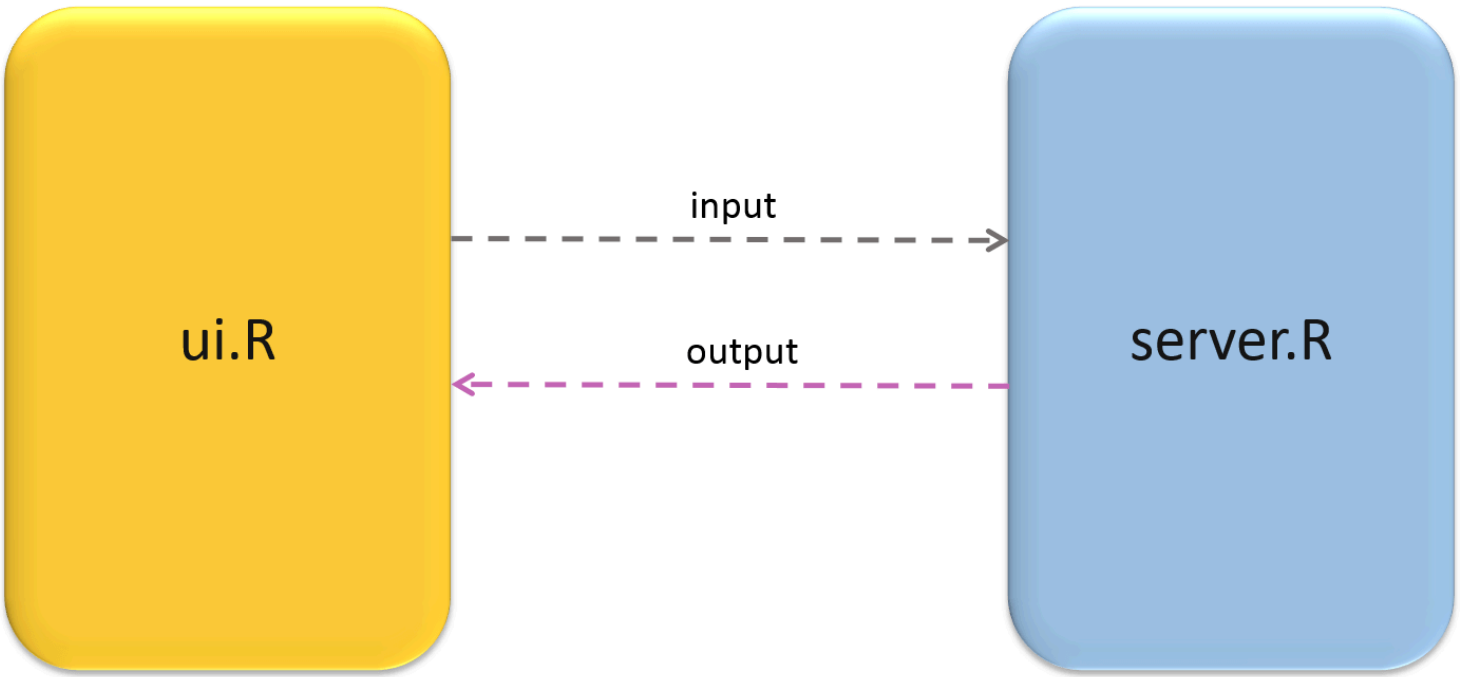
Shiny applications not supported in static R Markdown documents

Shiny网页的基本架构

- Shiny应用有两个组成部分：
  - 用户界面对象(User interface, UI)，这里UI使用`fluidPage()`函数来指定网页的布局。
  - 服务端函数(Server function)
- 这两个部分被作为参数传输到`ShinyApp`功能函数，再根据UI/Server对生成一个Shiny应用网页。

Shiny网页的基本架构

```
library(shiny)
ui <- ...
server <- ...
shinyApp(ui = ui, server = server)
```

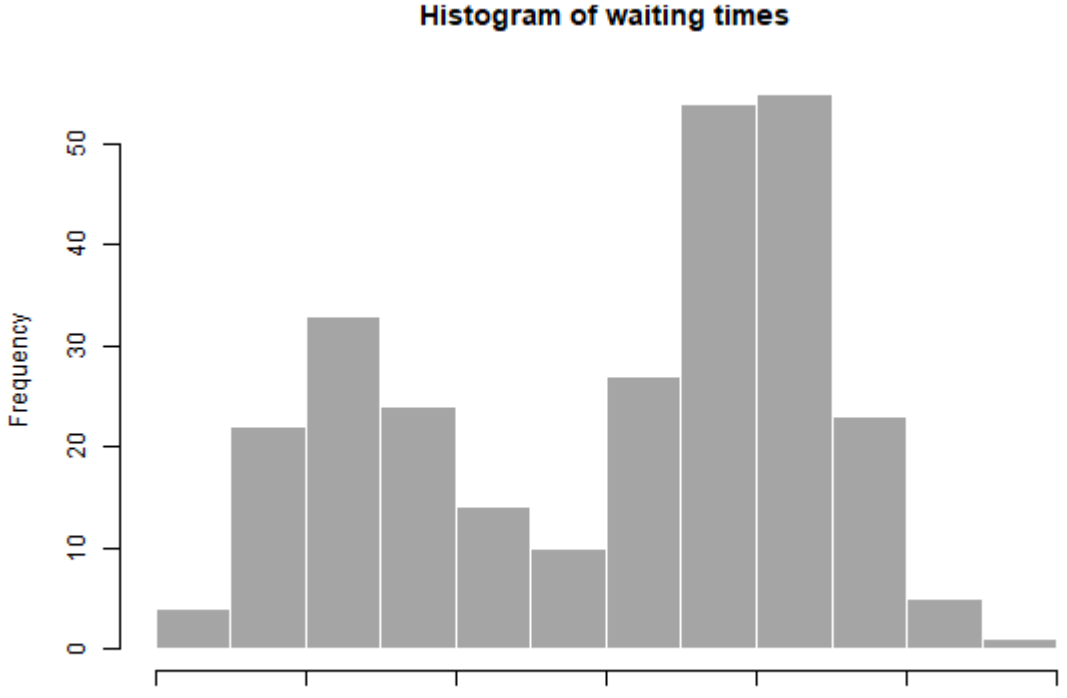


一步一步来搭建R Shiny

第一步：定义Shiny应用后台的R代码

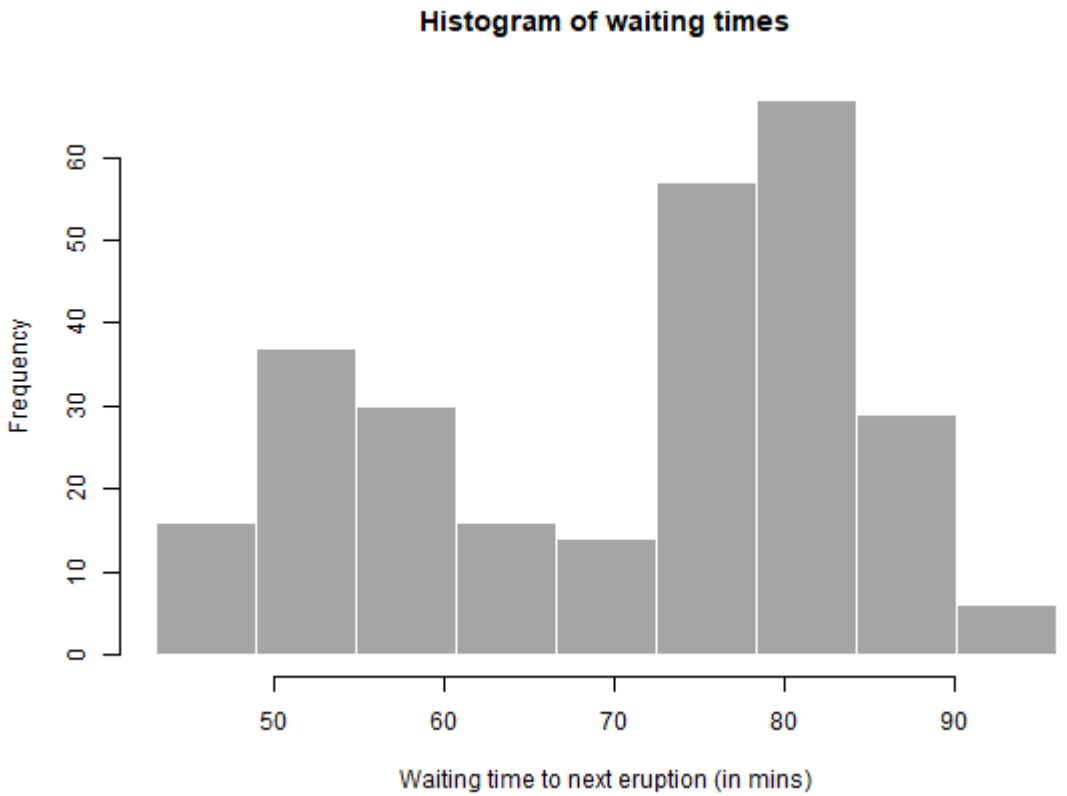
- 我们的目的是搭建一个网页应用：基于faithful数据，绘制关于火山喷发的等待时间的直方图，并允许用户通过交互指定直方图窗宽的个数。

```
x <- faithful[, 2]
hist(x, col = 'darkgray', border = 'white',xlab = 'Waiting time to next eruption (
```



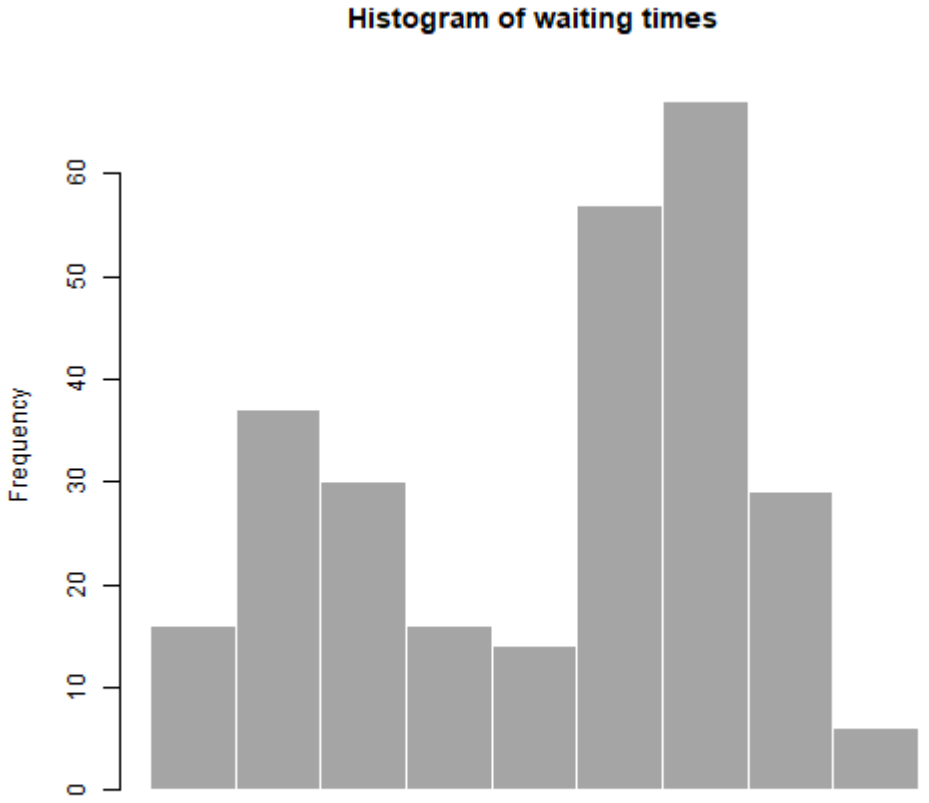
第一步：定义Shiny应用后台的R代码

```
bins <- seq(min(x), max(x), length.out = 10) # 9个柱子
hist(x, breaks = bins, col = 'darkgray', border = 'white',xlab = 'Waiting time to
```



第一步：定义Shiny应用后台的R代码

```
input <- 9
bins <- seq(min(x), max(x), length.out = input + 1) # 9个柱子
hist(x, breaks = bins, col = 'darkgray', border = 'white',xlab = 'Waiting time to
```



第二步：定义用户界面对象/前端

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

    # Application title
    titlePanel("Old Faithful Geyser Data"),

    # Sidebar with a slider input for number of bins
    sidebarLayout(
        sidebarPanel(
            sliderInput("bins",
                        "Number of bins:",
                        min = 1,
                        max = 50,
                        value = 30)
        ),

        # Show a plot of the generated distribution
        mainPanel(
           plotOutput("distPlot")
        )
    )
)
```

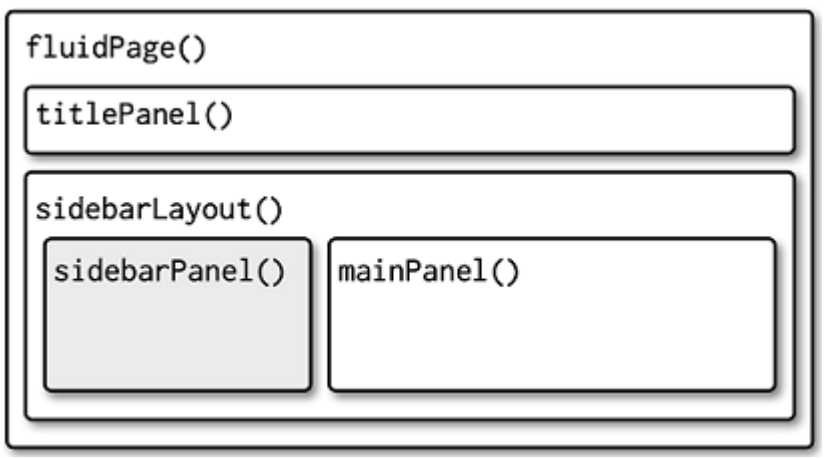
第二步：定义用户界面对象/前端

- 用户界面对象(User interface, UI)，这里UI使用`fluidPage()`函数来指定网页的布局。里面有两个主要内容：
  - 标题栏：`titlePanel()`
  - 网页的布局：通过一层层地设计，而且每一层布局都要在输出中有所对应。这里用的是`sidebarLayout()`来实现左右布局的。
    - 边栏：`sidebarPanel()`
    - 主体部分：`mainPanel()`

效果图

左边栏是用于摆放控件，右边空间用于展示图形、表格或者其他形式。

```
fluidPage(              # 创建网页
  titlePanel(),         # 标题栏
  sidebarLayout(        # 常用的边栏布局，将输入布局到左侧，输出布局到右侧
                        # 设置position="right"，则布局相反。
    sidebarPanel(),     # 边栏
    mainPanel()         # 主体部分
  )
)
```



布局之网页函数

- 王爷的布局通常是通过***Page()函数实现。除了前面所说的fluidPage()函数，还有

1. fluidPage()：创建一个流动布局的页面。
2. fixdPage()：创建一个具有固定宽度的页面，可防止网页过宽。
3. fillPage()：创建一个页面，可把一些空隙去掉，让网页内容充满整个页面。

用户界面的输入控制

- selectInput()是对输入的控制，允许用户通过输入或选择实现网页的交互。
- 所有的输入控制函数都有着相同的第一形参inputId，如果inputId为"name"，那么后台端会通过input$name来访问它。
- 常见的输入控制函数有：
  - 自由文本：textInput(),passwordInput(),textAreaInput().
  - 数值输入： numericInput(), sliderInput().
  - 日期输入： dateInput(),dateRangeInput() .
  - 单选：selectInput(), radioButtons().
  - 多选：checkboxGroupInput().
  - 上传文件： fileInput().

用户界面的输入控制

Button

Action

actionButton()

Single checkbox

Choice A

checkboxInput()

Checkbox group

Choice 1
Choice 2
Choice 3

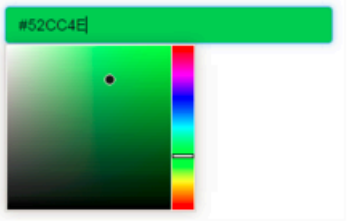
checkboxGroupInput()

Date input

2014-01-01

dateInput()

Colour input



colourpicker::colourInput()

Date range

2014-01-24 to 2014-01-24

dateRangeInput()

File input

Choose File No file chosen

fileInput()

Numeric input

1

numericInput()

Password Input

••••••••••

passwordInput()

Text area

Multiple lines
of text

textAreaInput()

Radio buttons

Choice 1
Choice 2
Choice 3

radioButtons()

Select box

Choice 1

selectInput()

Sliders

sliderInput()

Text input

Enter text...

textInput()

用户界面的输入控制例子：自由文本

```
ui <- fluidPage(
  textInput("name", "请问您的名字是？"),
  passwordInput("password", "请问您的密码是？"),
  textAreaInput("story", "请用一段话介绍一下您自己", rows = 3)
)
server <- function(input, output, session) {
}
shinyApp(ui, server)
```

Shiny applications not supported in static R Markdown documents

用户界面的输入控制例子：数值

```
ui <- fluidPage(
  numericInput("num", "Number one", value = 0, min = 0, max = 100),
  sliderInput("num2", "Number two", value = 50, min = 0, max = 100),
  sliderInput("rng", "Range", value = c(10, 20), min = 0, max = 100)
)
server <- function(input, output, session) {
}
shinyApp(ui, server)
```

Shiny applications not supported in static R Markdown documents

用户界面的输入控制例子：单选

```
ui <- fluidPage(
  dateInput("dob", "请问您的出生日期是？"),
  dateRangeInput("holiday", "请问您想什么时候完成第三次作业？")
)
server <- function(input, output, session) {
}
shinyApp(ui, server)
```

Shiny applications not supported in static R Markdown documents

用户界面的输入控制例子：单选

```
animals <- c("狗", "小猫", "老鼠", "小鸟", "其它", "我不喜欢动物")
province <- c("中科大", "清华","北大")

ui <- fluidPage(
  selectInput("state", "请问您来自于哪个学校", province),
  radioButtons("animal", "请问您最喜欢的动物是什么？", animals)
)
server <- function(input, output, session) {
}
shinyApp(ui, server)
```

Shiny applications not supported in static R Markdown documents

用户界面的输入控制例子：多选

```
ui <- fluidPage(
  checkboxGroupInput("animal", "请问您最喜欢的动物是什么？", animals)
)
server <- function(input, output, session) {
}
shinyApp(ui, server)
```

Shiny applications not supported in static R Markdown documents

用户界面的输出控制

- plotOutput()是对输出的控制，Shiny中的输出函数都是以***Output来命名的。
- 所有的输出函数都有着相同的第一形参outputId，如果outputId为"name"，那么后台端会通过output$name来访问它。
- 其余形参为具体的内容，如width等。
- 常见的输出控制函数有：

函数	输出内容
plotOutput()	图形
tableOutput()	静态表格
textOutput()	带有格式的文本
verbatimTextOutput()	代码和控制台输出
dataTableOutput()	动态表格
imageOutput()	图片

用户界面的输出例子：文本，代码

```
ui <- fluidPage(
  textOutput("text"),
  verbatimTextOutput("code")
)
server <- function(input, output, session) {
  output$text <- renderText({
    "数据汇总如下："
  })
  output$code <- renderPrint({
    summary(1:10)
  })
}
shinyApp(ui, server)
```

Shiny applications not supported in static R Markdown documents

用户界面的输出例子：静态表格

```
ui <- fluidPage(
  tableOutput("static")
)
server <- function(input, output, session) {
  output$static <- renderTable(head(mtcars))
}
shinyApp(ui, server)
```

Shiny applications not supported in static R Markdown documents

用户界面的输出例子：动态表格

```
ui <- fluidPage(
  dataTableOutput("dynamic")
)
server <- function(input, output, session) {
  output$dynamic <- renderDataTable(mtcars, options = list(pageLength = 5))
}
shinyApp(ui, server)
```

Shiny applications not supported in static R Markdown documents

第三步：配置后台端（服务器）

```
# Define server logic required to draw a histogram
server <- function(input, output) {

    output$distPlot <- renderPlot({
        # generate bins based on input$bins from ui.R
        x    <- faithful[, 2]
        bins <- seq(min(x), max(x), length.out = input$bins + 1)

        # draw the histogram with the specified number of bins
        hist(x, breaks = bins, col = 'darkgray', border = 'white',
             xlab = 'Waiting time to next eruption (in mins)',
             main = 'Histogram of waiting times')
    })
}
```

第三步：配置后台端（服务器）

- 通过server()函数将前端的input输入到第一步的代码中，并将代码运行后的结果作为输出内容output和前端的输出匹配。
- 这里我们的输出内容是图形，因此使用renderPlot()进行输出，定义输出的名称为distPlot，和第二步中的plotOutput("distPlot")相对应。
- 通过调用前端的输入参数input$bins实现用户指定窗格的个数。

第三步：配置后台端（服务器）

- 注意，每一个前端的输出函数都会在后端匹配一个render***()函数，将R中某种形式的输出（如文本、表格、图形等）传递到网页上。
- 具体的匹配如下：
  - 文本： textOutput()和 renderText()
  - 代码和控制台输出：verbatimTextOutput()和 renderPrint()
  - 表格：
    - 静态表格：tableOutput()和 renderTable()
    - 动态表格： dataTableOutput()和 renderDataTable()
  - 绘图： plotOutput()和 renderPlot()

第四步：将前端和后台端连接

- 一旦我们定以后前端和后台端的内容，那么接下来通过 shinyApp() 函数将前端 ui 和后台端 server 连接起来创建网页。

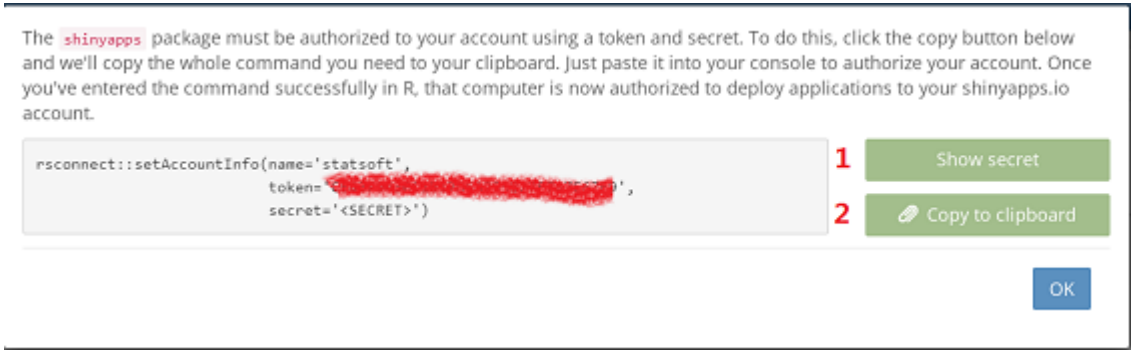
```
shinyApp(ui = ui, server = server)
```

Shiny applications not supported in static R Markdown documents

第五步：分享你的网页（非必须）

- 通过shinyapps.io发布和分享你制作的网页是最为简单的方式。

1. 官网注册
2. 在RStudio中下载并安装rsconnect
3. 如果是第一次使用的，需要按照官网提示设置好令牌信息，如下图所示:



然后把上述令牌信息粘贴到RStudio控制台运行

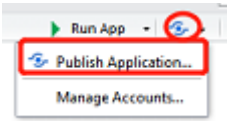
4. 发布网页

```
library(rsconnect)
rsconnect::deployApp('mc_int')
```

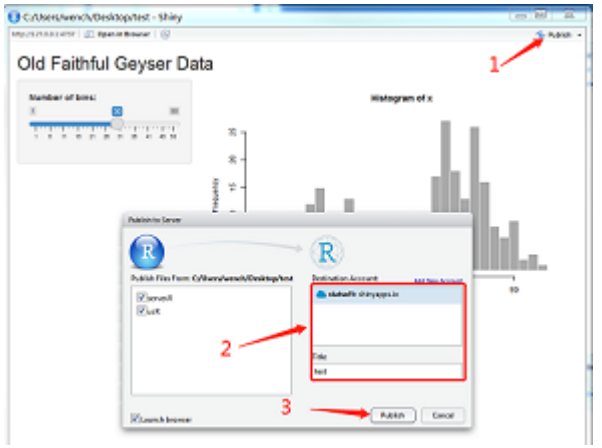
第五步：分享你的网页（非必须）

- 如果你在最开始是以新建Shiny 项目的形式开发的，那么你可以通过鼠标点击的方式进行发布。具体步骤如下：

1.点击R文件或者网页文件的"Publish"



2.填写账户信息和相关信息，选择ShinyApps.io, 点击发布即可。



案例学习：蒲丰投针

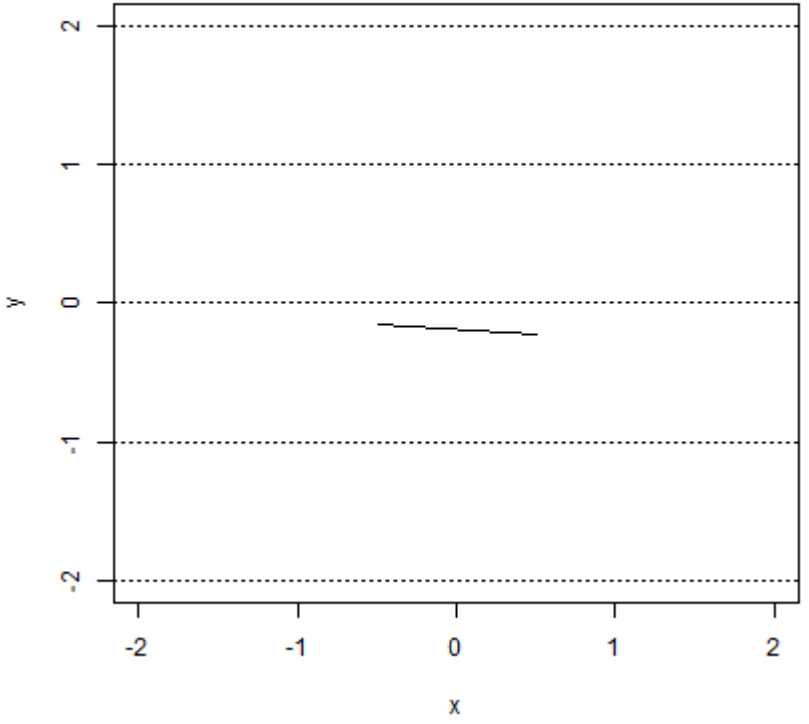
案例学习：蒲丰投针

- 定义cast_needle()函数投针，并计算是否与平行线（平行线之间的宽度为1）相交。输入为展示平面的宽度，默认针的长度为1。

```
cast_needle <- function(plane_width = 20){
  available_range <- plane_width/2 - 1 # where 1 is the length of the needle (unit
  x_start <- runif(2, -available_range, available_range)
  angle <- runif(1, 0, 2*pi)
  x_end <- c(cos(angle), sin(angle)) + x_start # where the angles are multiplied b
  cross <- floor(x_start[2]) != floor(x_end[2])
  out <- list(start = x_start, end = x_end, cross = cross)
  out
}
```

案例学习：蒲丰投针

```
needle <- cast_needle(plane_width = 4)
plot(NA, xlim = c(-2, 2), ylim = c(-2, 2), xlab = "x", ylab = "y")
abline(h = -2:2, lty = 3)
lines(c(needle$start[1], needle$end[1]), c(needle$start[2], needle$end[2]))
```



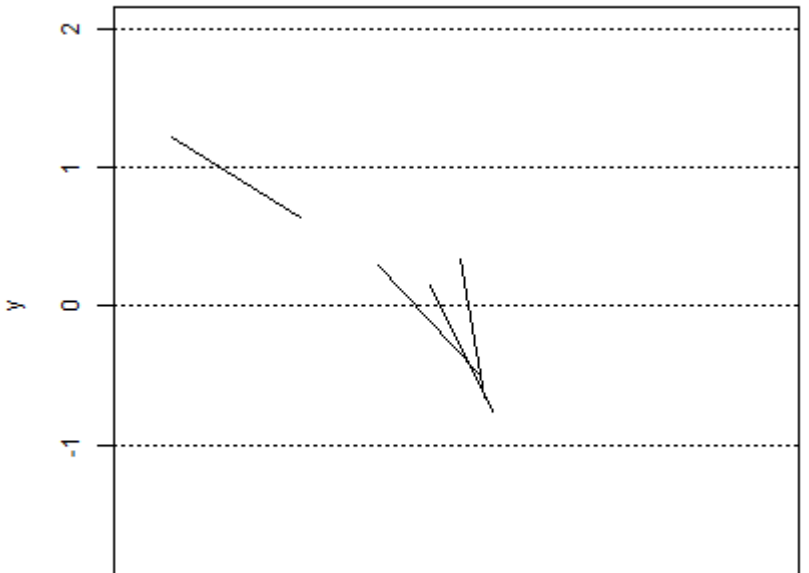
案例学习：蒲丰投针

- 接下来定义蒲丰投针试验函数，输入为针的根数B，展示平面的宽度和种子数。

```
buffon_experiment <- function(B = 2084, plane_width = 10, seed = NULL){

  if (!is.null(seed)){
    set.seed(seed)
  }

  X_start <- X_end <- matrix(NA, B, 2)
  cross <- rep(NA, B)

  for (i in 1:B){
    inter <- cast_needle(plane_width = plane_width)
    X_start[i, ] <- inter$start
    X_end[i, ] <- inter$end
    cross[i] <- inter$cross
  }

  out <- list(start = X_start, end = X_end, cross = cross, plane = plane_width)
  class(out) <- "buffon_experiment"
  out
}
```

案例学习：蒲丰投针

```
experiment <- buffon_experiment(B = 4, plane_width = 4)

plot(NA, xlim = c(-2, 2), ylim = c(-2, 2), xlab = "x", ylab = "y")
abline(h = -2:2, lty = 3)
for (i in 1:4){
  lines(c(experiment$start[i,1], experiment$end[i,1]),
        c(experiment$start[i,2], experiment$end[i,2]))
}
```



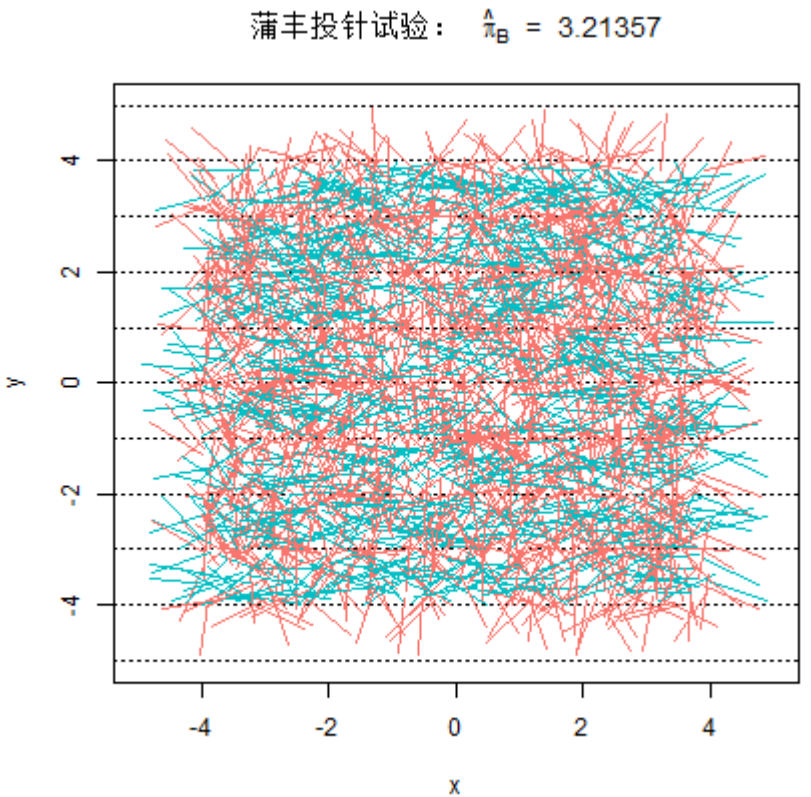
案例学习：蒲丰投针

- 根据buffon_experiment()函数的输出定义绘图函数。

```
plot.buffon_experiment <- function(obj){
  cross <- obj$cross
  X_start <- obj$start
  X_end <- obj$end
  B <- length(cross)
  cols <- rev(hcl(h = seq(15, 375, length = 3), l = 65, c = 100)[1:2])

  title_part1 <- '蒲丰投针试验：  '
  title_part2 <- ' = '
  pi_hat <- round(2/mean(obj$cross), 6)

  title <- bquote(.(title_part1) ~ hat(pi)[B] ~ .(title_part2) ~ .(pi_hat))

  plot(NA, xlab = "x", ylab = "y", xlim = c(-obj$plane/2, obj$plane/2),
       ylim = c(-obj$plan/2, obj$plan/2),
       main = title)
  abline(h = (-obj$plan):obj$plan, lty = 3)

  for (i in 1:B){
    lines(c(X_start[i,1], X_end[i,1]), c(X_start[i,2], X_end[i,2]),
          col = cols[cross[i] + 1])
  }
}
```

案例学习：蒲丰投针

```
experiment <- buffon_experiment(B = 2084)
plot(experiment)
```



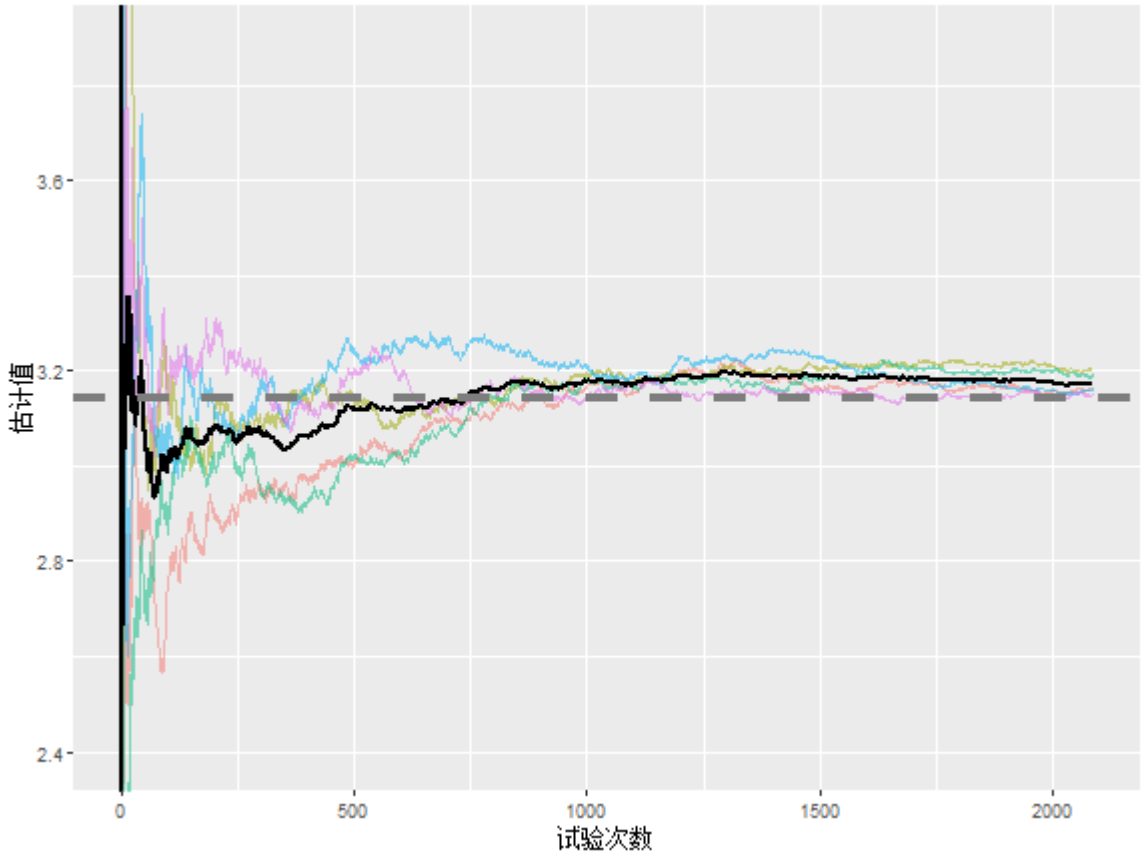
案例学习：蒲丰投针

- 最后定义converge()函数进行绘图工作，输入为针的根数B、展示平面的宽度、种子数和试验的次数

```
converge <- function(B = 2084, plane_width = 10, seed = 123, M = 10){

  if (B < 10){
    warning("针的根数太少，强制改成10根！")
    B <- 10
  }

  pi_hat <- matrix(NA, B, M)
  trials <- 1:B

  set.seed(seed)
  for (i in 1:M){
    cross <- buffon_experiment(B = B, plane_width = plane_width)$cross
    pi_hat[,i] <- 2*trials/cumsum(cross)
  }
  pi_hat_long <- reshape2::melt(pi_hat)
  pi_hat_long$Var2 <- factor(pi_hat_long$Var2)
  library(ggplot2)
  ggplot(pi_hat_long, aes(x = Var1, y = value)) +
    geom_line(aes(color = Var2), show.legend = FALSE, alpha = 0.5) +
    stat_summary(fun.y=mean, geom="line",linewidth=1, color = "black") +
    geom_hline(aes(yintercept = pi), color = "gray50", linewidth=1.5, linetype = "
    coord_cartesian(ylim = pi + c(-3/4, 3/4)) +
```

案例学习：蒲丰投针

```
converge(B = 2084, M = 5, seed = 10)
```



案例学习：蒲丰投针

- 定义一个空的网页

```
ui <- fluidPage(

  titlePanel(h4("蒲丰投针试验")),

  sidebarLayout(
    sidebarPanel(
      # 在这里添加输入控制
    ),

    mainPanel(
      tabsetPanel(
        # 添加输出的内容
      )
    )
  )
)

server <- function(input, output) {
  output$exp <- renderPlot({
    # 第一张图
  }, height = 620)

  output$conv <- renderPlot({
    # 第二张图
```

案例学习：蒲丰投针

- 定义用户界面

```
ui <- fluidPage(

  titlePanel(h4("蒲丰投针试验")),

  sidebarLayout(
    sidebarPanel(
      numericInput("plane", "展示平面的宽度", 10, 6, 100),
      numericInput("B", "针的根数", 100, 20, 10^6),
      numericInput("M", "试验的次数", 1, 1, 100),
      numericInput("seed", "随机种子", 123, 1, 1000000),
      actionButton("cast", "让我们来投针吧!", icon = icon("thumbs-up"))
    ),

    mainPanel(
      tabsetPanel(
        tabPanel("试验结果展示", plotOutput("exp")),
        tabPanel("收敛结果展示", plotOutput("conv"))
      )
    )
  )
)
```

案例学习：蒲丰投针

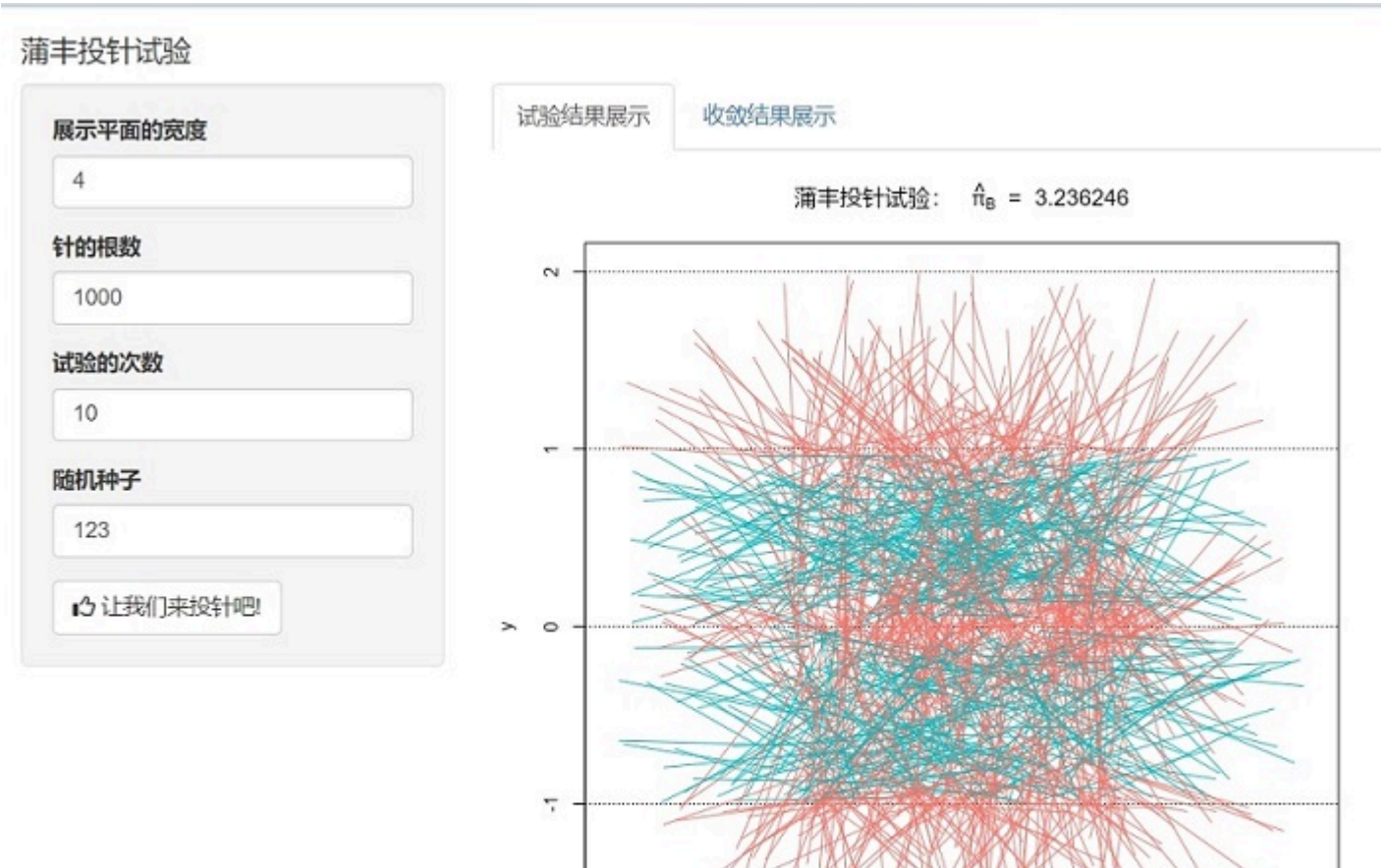
- 定义后台端函数

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

  observeEvent(input$cast,{
    updateNumericInput(session, "seed",
                       value = round(runif(1, 1, 10^4)))
  })

  cast = eventReactive(input$cast, {
    buffon_experiment(B = input$B, plane_width = input$plane,
                      seed = input$seed)
  })

  conv = eventReactive(input$cast, {
    converge(B = input$B, plane_width = input$plane,
             seed = input$seed, M = input$M)
  })

  output$exp <- renderPlot({
    plot(cast())
  }, height = 620)

  output$conv <- renderPlot({
    conv()
  }, height = 620)
```

案例学习：蒲丰投针

- 大功告成!

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



案例学习：蒲丰投针

展示平面的宽度

4

针的根数

1000

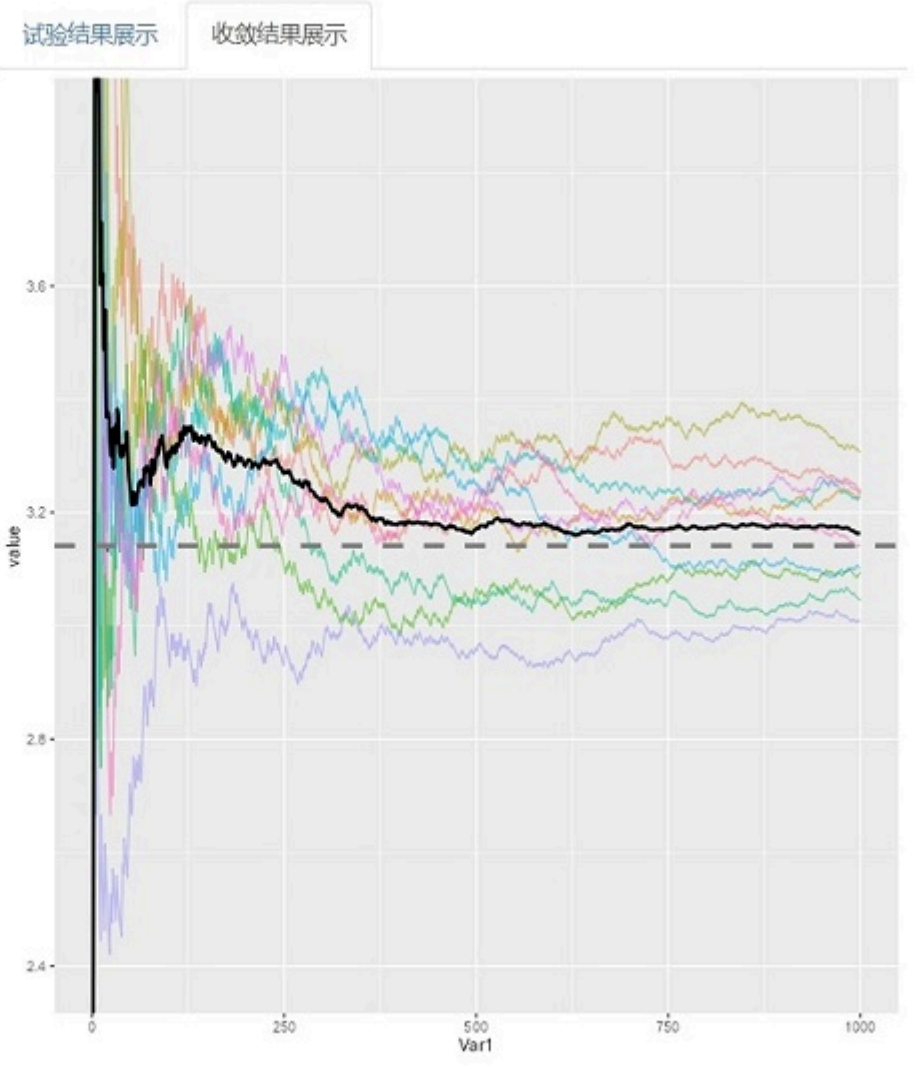
试验的次数

10

随机种子

123

让我们来投针吧!



谢 谢