



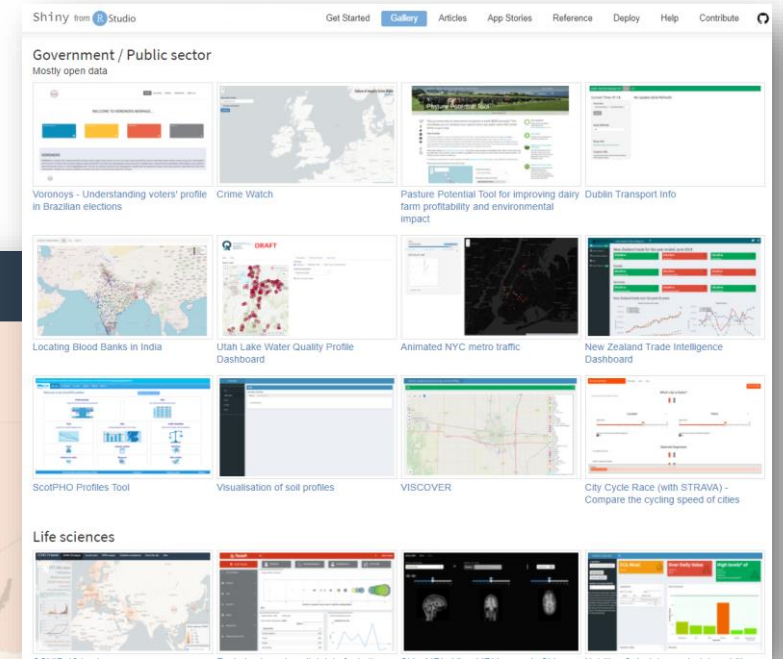
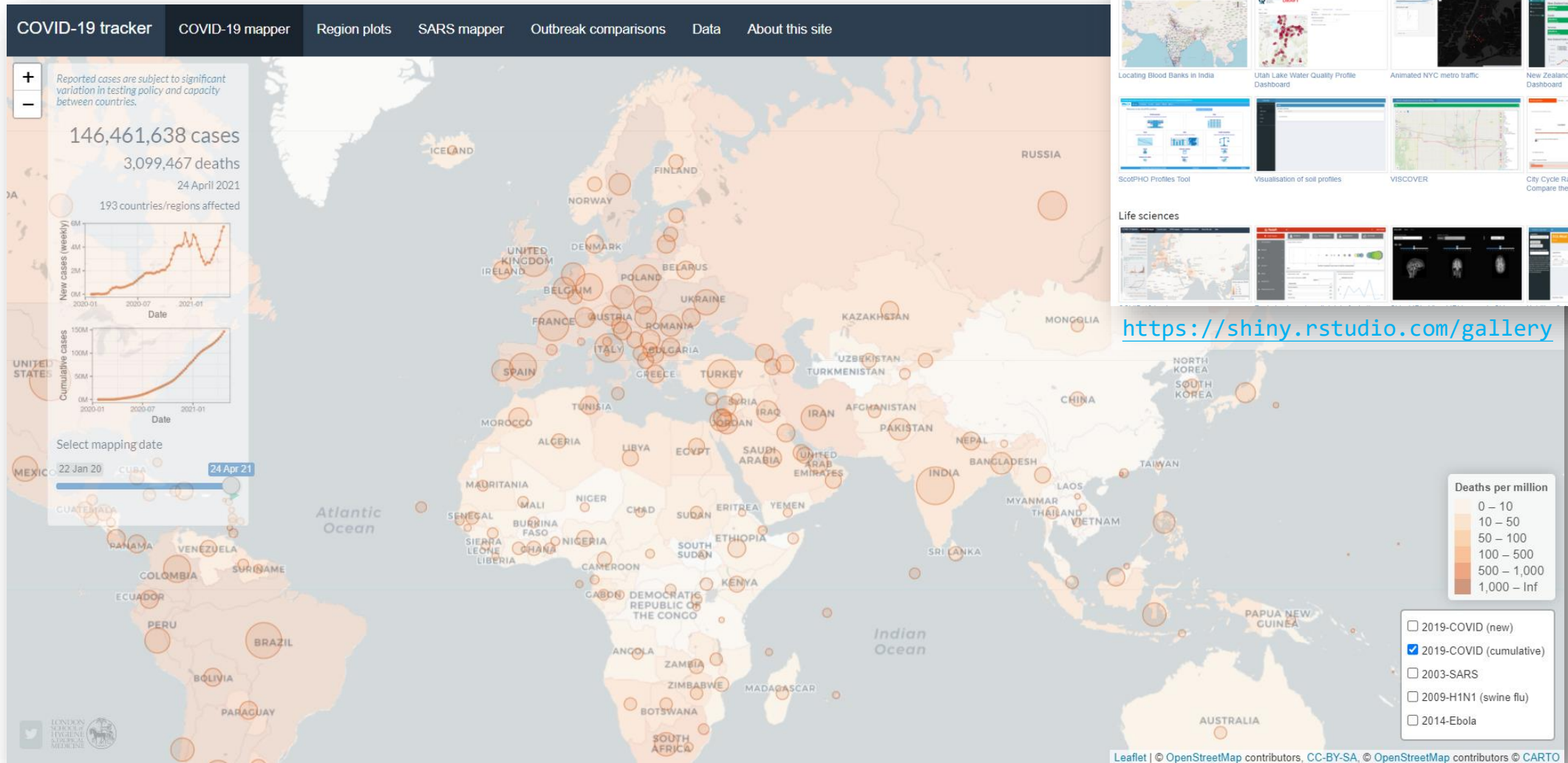
# R Shiny

## 互動式地圖網頁

109-2  
空間分析

杜承軒  
2021.06.21

# Interactive Data Visualization in R



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

COVID-19 tracker [https://vac-lshtm.shinyapps.io/ncov\\_tracker](https://vac-lshtm.shinyapps.io/ncov_tracker)

# Running the First R Shiny app

```
library(shiny)
runExample("01_hello")
```



The screenshot shows a Shiny application window titled "Hello Shiny!". On the left, there is a sidebar with a slider input labeled "Number of bins:" with a value of 33. The main panel displays a histogram titled "Histogram of waiting times" showing the frequency distribution of waiting times to the next eruption in minutes. The x-axis is labeled "Waiting time to next eruption (in mins)" and ranges from 50 to 90. The y-axis is labeled "Frequency" and ranges from 0 to 25. Below the histogram, the source code for the app is displayed in a code editor window titled "app.R".

```
app.R
library(shiny)

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

  # App title ----
  titlePanel("Hello Shiny!"),

  # Sidebar layout with input and output definitions ----
  sidebarLayout(

    # Sidebar panel for inputs ----
    sidebarPanel(

      # Input: Slider for the number of bins ----
      sliderInput(inputId = "bins",
                  label = "Number of bins:",
                  min = 1,
                  max = 50,
                  value = 30)

    ),

    # Main panel for displaying outputs ----
    mainPanel(

      # Output: Histogram ----
      plotOutput(outputId = "distPlot")

    )

  )

)

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

  # Histogram of the Old Faithful Geyser Data ----
  # with requested number of bins
  # This expression that generates a histogram is wrapped in a call
  # to renderPlot to indicate that:
  #
  # 1. It is "reactive" and therefore should be automatically
  #    re-executed when inputs (input$bins) change
  # 2. Its output type is a plot
  output$distPlot <- renderPlot({

    x <- faithful$waiting
    bins <- seq(min(x), max(x), length.out = input$bins + 1)
```

## More Examples:

```
"01_hello", "02_text", "03_reactivity",
"04_mpg", "05_sliders", "06_tabsets",
"07_widgets", "08_html", "09_upload",
"10_download", "11_timer"
```

 app.R

```
library(shiny)

ui = fluidPage(

  titlePanel("Hello Shiny!"),

  sidebarLayout(
    sidebarPanel(sliderInput(.....)),
    mainPanel(plotOutput("distPlot"))
  )

)

server = function(input, output) {

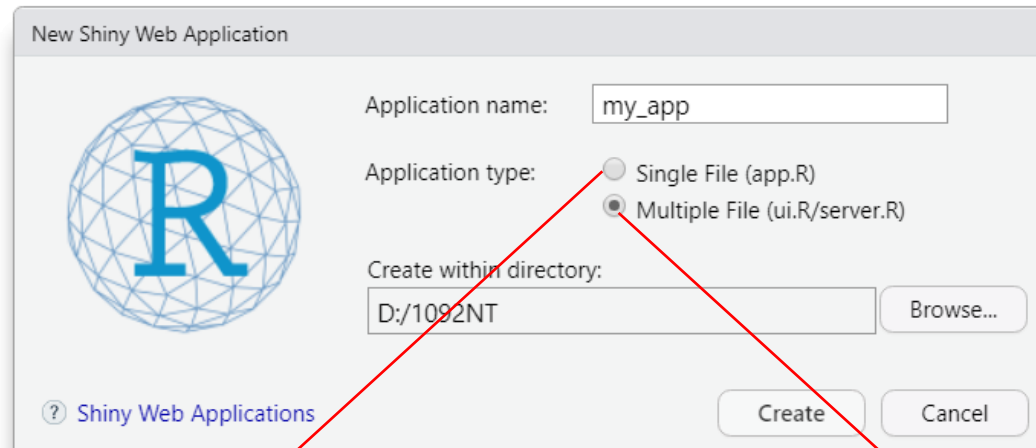
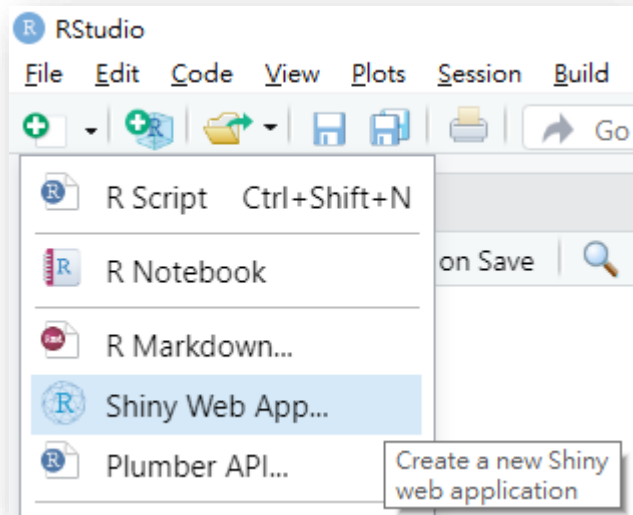
  output$distPlot = renderPlot({.....})

}

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

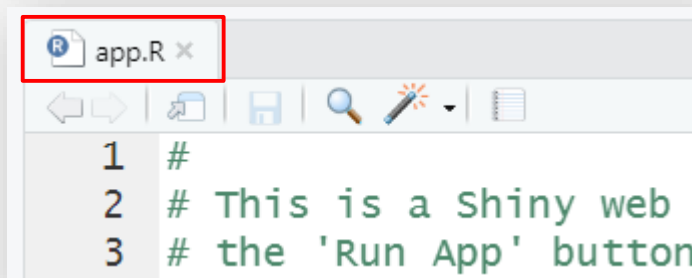
Shiny applications have two components, a **user interface object** and a **server function**, that are passed as arguments to the `shinyApp` function that creates a Shiny app object from this UI/server pair.

# ShinyApp

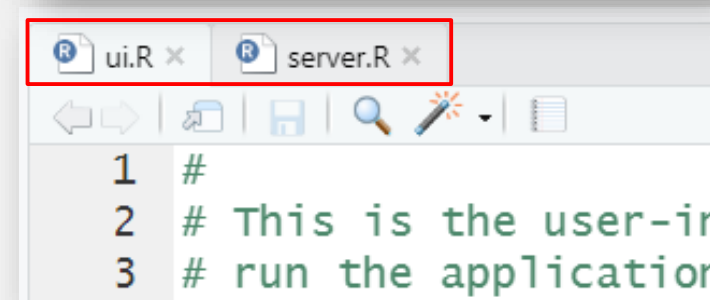


兩者效果相同

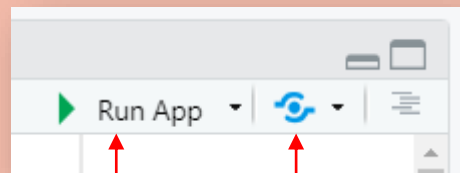
## Single File (app.R)



## Multiple File (ui.R/server.R)



## Run ShinyApp



Run App

Publish to web

# ShinyApp

R app.R

```
library(shiny)

[highlighted]

ui = fluidPage(
  titlePanel("Hello Shiny!"),
  sidebarLayout(
    sidebarPanel(sliderInput(.....)),
    mainPanel(plotOutput("distPlot"))
  )
)

server = function(input, output) {
  output$distPlot = renderPlot({.....})
}

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



(以下.R檔要放同個資料夾)

Diagram showing the decomposition of the app.R code into separate files:

- global.R**: [highlighted]
- ui.R**:

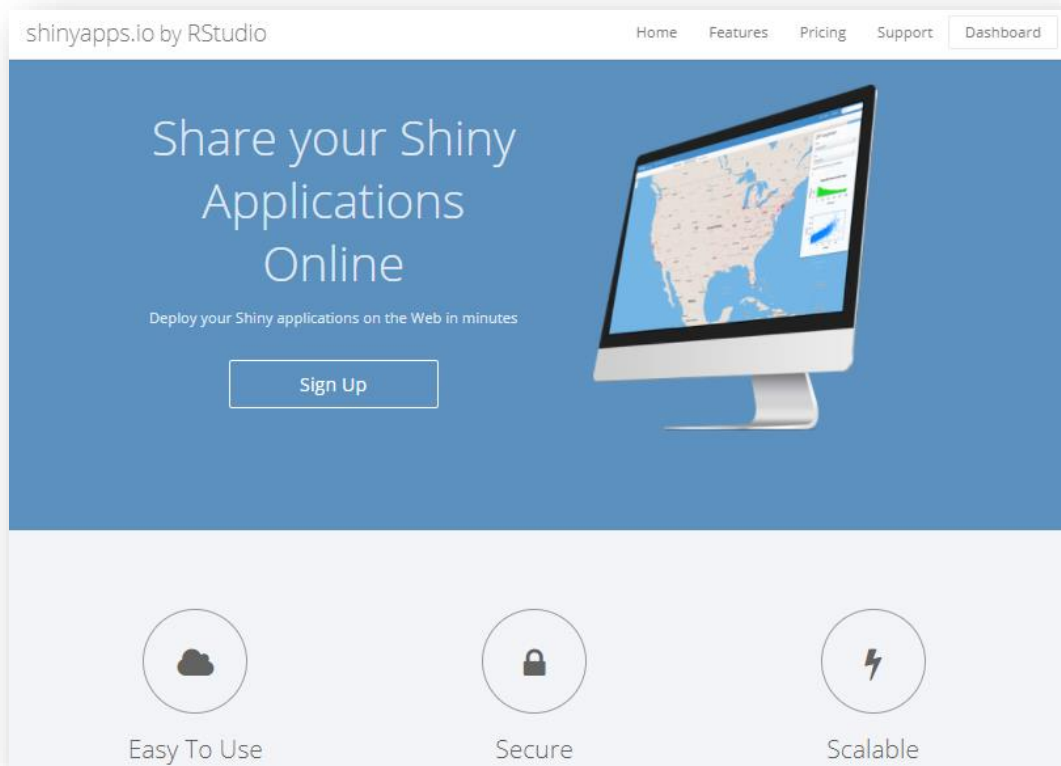
```
library(shiny)
shinyUI(fluidPage(
  titlePanel("Hello Shiny!"),
  sidebarLayout(
    sidebarPanel(sliderInput(.....)),
    mainPanel(plotOutput("distPlot"))
  )
))
```
- server.R**:

```
library(shiny)
shinyServer(function(input, output) {
  output$distPlot = renderPlot({.....})
})
```

→ 後面會更詳細說明。直接動手操作更好懂！

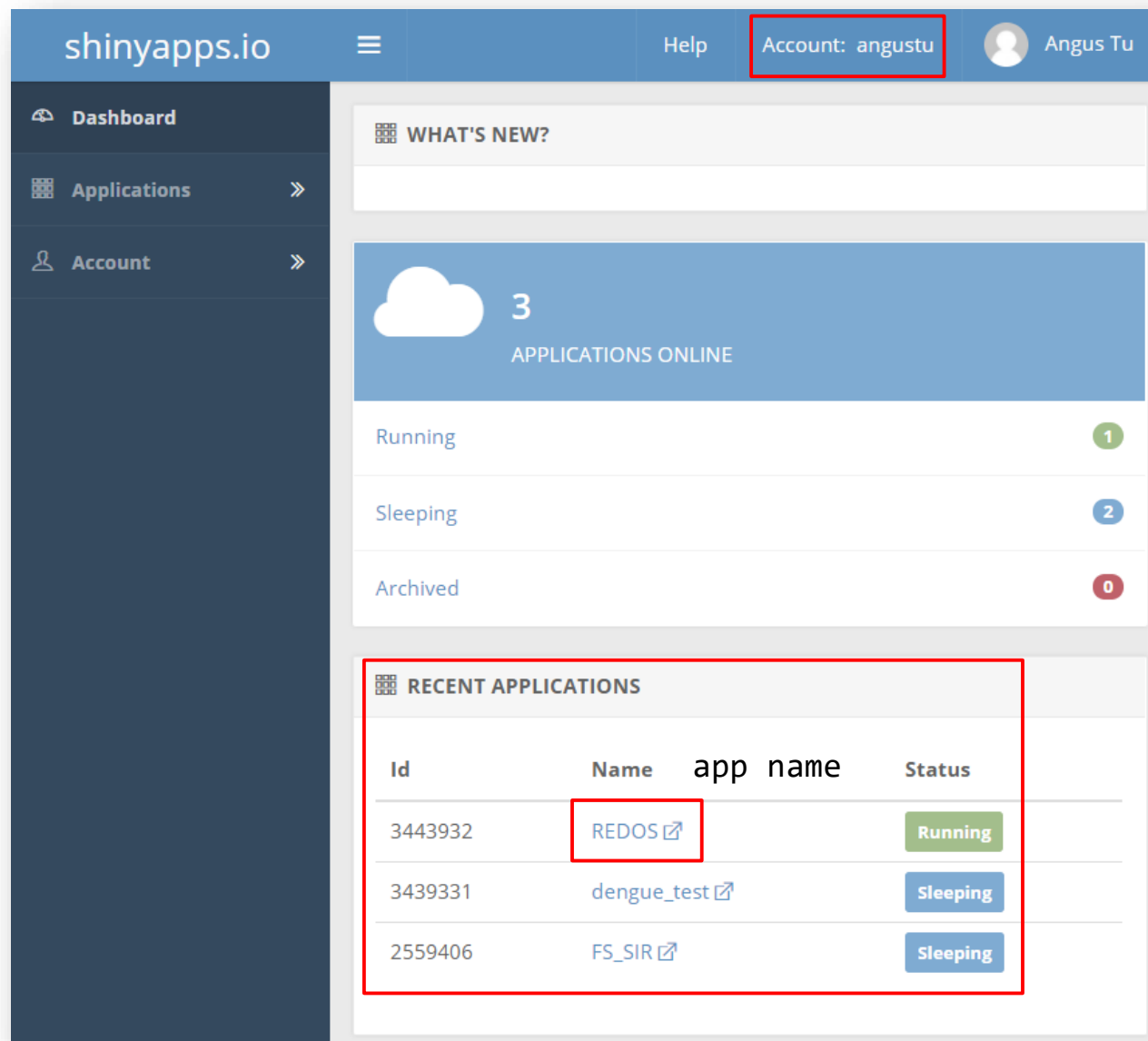
# Shiny App Server

<http://www.shinyapps.io/>



→ Sign Up

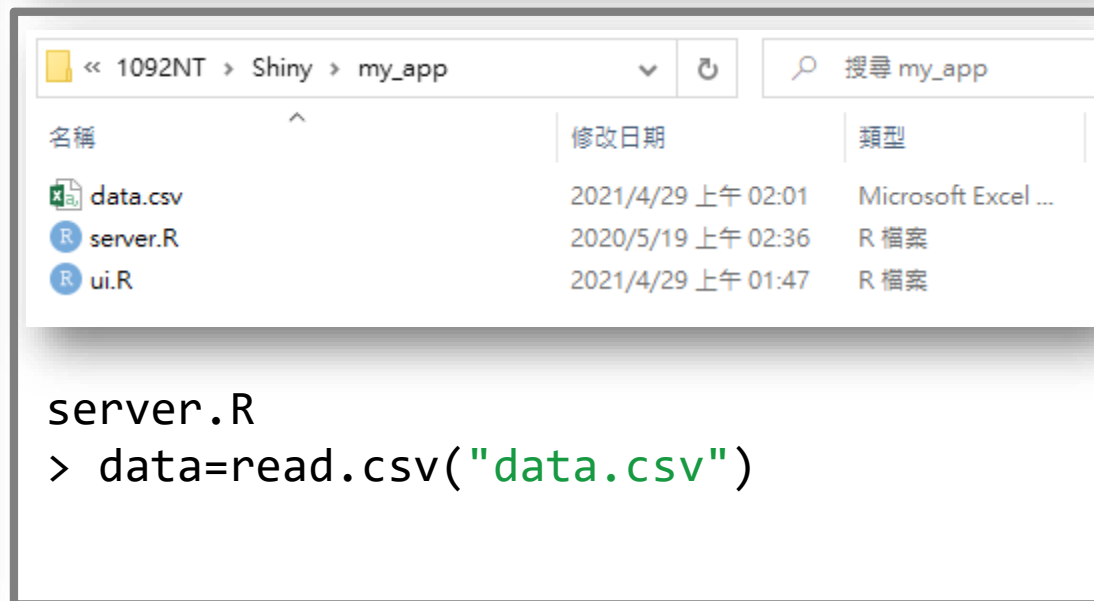
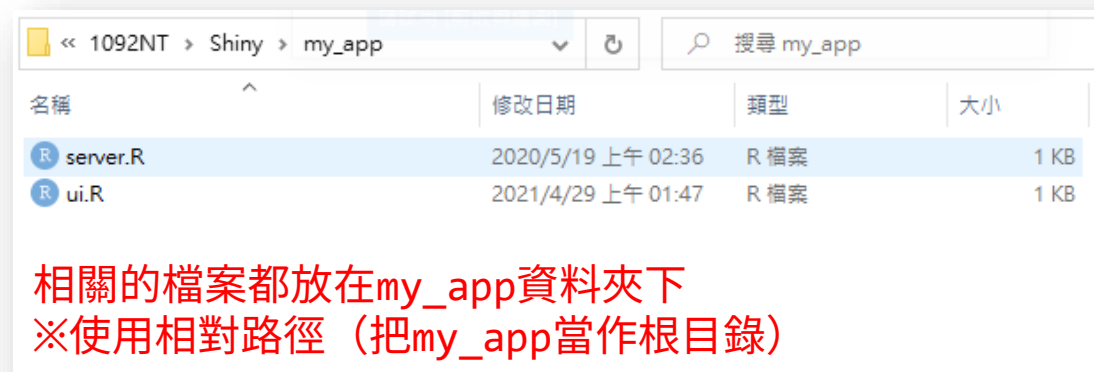
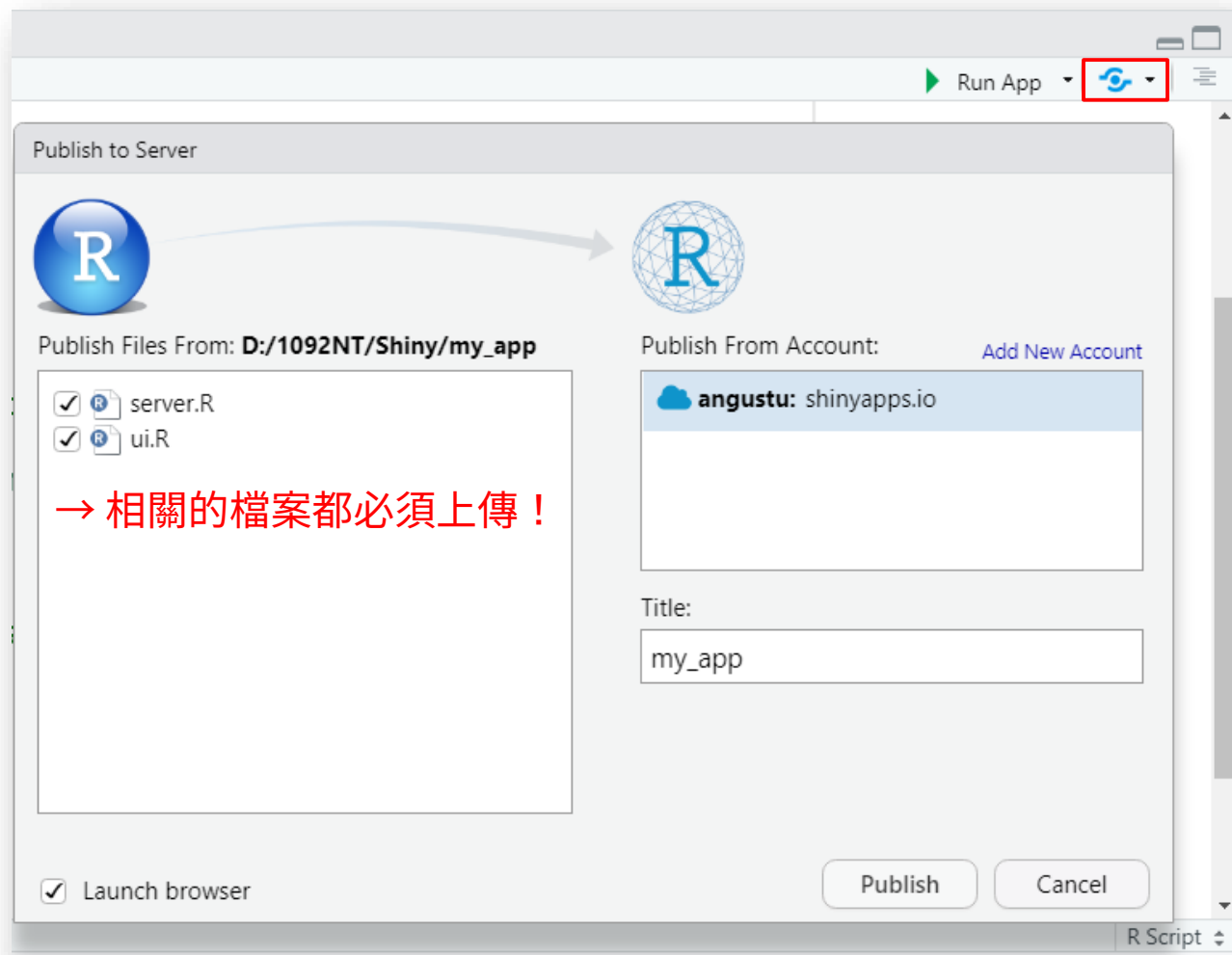
※中文問題



→ [https://<account\\_name>.shinyapps.io/<app\\_name>/](https://<account_name>.shinyapps.io/<app_name>/)

# Deploying Shiny apps to the web

# Path of Files





## Interactive Web Apps with shiny Cheat Sheet

learn more at [shiny.rstudio.com](https://shiny.rstudio.com)



### Basics

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).

#### App template

Begin writing a new app with this template. Preview the app by running the code at the R command line.

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

- ui** - nested R functions that assemble an HTML user interface for your app
- server** - a function with instructions on how to build and rebuild the R objects displayed in the UI
- shinyApp** - combines **ui** and **server** into a functioning app. Wrap with **runApp()** if calling from a sourced script or inside a function.

#### Share your app

The easiest way to share your app is to host it on [shinyapps.io](https://shinyapps.io), a cloud based service from RStudio

- Create a free or professional account at <http://shinyapps.io>
- Click the **Publish** icon in the RStudio IDE (>=0.99) or run: `rsconnect::deployApp("<path to directory>")`

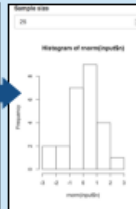
Build or purchase your own Shiny Server at [www.rstudio.com/products/shiny-server/](https://www.rstudio.com/products/shiny-server/)

### Building an App - Complete the template by adding arguments to fluidPage() and a body to the server function.

Add inputs to the UI with **\*Input()** functions  
Add outputs with **\*Output()** functions  
Tell server how to render outputs with R in the server function. To do this:

- Refer to outputs with `output$<id>`
- Refer to inputs with `input$<id>`
- Wrap code in a `render*`() function before saving to output

```
library(shiny)
ui <- fluidPage(
  numericInput(inputId = "n",
    "Sample size", value = 25),
  plotOutput(outputId = "hist")
)
server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$n))
  })
}
shinyApp(ui = ui, server = server)
```



Save your template as **app.R**. Alternatively, split your template into two files named **ui.R** and **server.R**.

```
library(shiny)
ui <- fluidPage(
  numericInput(inputId = "n",
    "Sample size", value = 25),
  plotOutput(outputId = "hist")
)
server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$n))
  })
}
shinyApp(ui = ui, server = server)
```

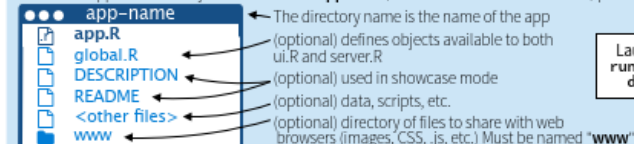
```
# ui.R
fluidPage(
  numericInput(inputId = "n",
    "Sample size", value = 25),
  plotOutput(outputId = "hist")
)
# server.R
function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$n))
  })
}
```

**ui.R** contains everything you would save to ui.

**server.R** ends with the function you would save to server.

No need to call `shinyApp()`.

Save each app as a directory that contains an **app.R** file (or a **server.R** file and a **ui.R** file) plus optional extra files.



Launch apps with `runApp("<path to directory>")`

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

	<code>DT::renderDataTable(expr, options, callback, escape, env, quoted)</code>	works with	<code>dataTableOutput(outputId, icon, ...)</code>
	<code>renderImage(expr, env, quoted, deleteFile)</code>		<code>imageOutput(outputId, width, height, click, dblclick, hover, hoverDelay, hoverDelayType, brush, clickId, hoverId, inline)</code>
	<code>renderPlot(expr, width, height, res, ..., env, quoted, func)</code>		<code>plotOutput(outputId, width, height, click, dblclick, hover, hoverDelay, hoverDelayType, brush, clickId, hoverId, inline)</code>
	<code>renderPrint(expr, env, quoted, func, width)</code>		<code>verbatimTextOutput(outputId)</code>
	<code>renderTable(expr, ..., env, quoted, func)</code>		<code>tableOutput(outputId)</code>
	<code>renderText(expr, env, quoted, func)</code>		<code>textOutput(outputId, container, inline)</code>
	<code>renderUI(expr, env, quoted, func)</code>		<code>uiOutput(outputId, inline, container, ...)</code> & <code>htmlOutput(outputId, inline, container, ...)</code>

### Inputs - collect values from the user

Access the current value of an input object with `input$<inputid>`. Input values are **reactive**.

- Action** `actionButton(inputId, label, icon, ...)`
- Link** `actionLink(inputId, label, icon, ...)`
- Choice 1** `checkboxGroupInput(inputId, label, choices, selected, inline)`
- Choice 2** `checkboxInput(inputId, label, value)`
- Choice 3** `checkboxInput(inputId, label, value)`
- Check me** `checkboxInput(inputId, label, value)`
- dateInput** `dateInput(inputId, label, value, min, max, format, startview, weekstart, language)`
- dateRangeInput** `dateRangeInput(inputId, label, start, end, min, max, format, startview, weekstart, language, separator)`

- Choose File** `fileInput(inputId, label, multiple, accept)`
- 1** `numericInput(inputId, label, value, min, max, step)`
- password** `passwordInput(inputId, label, value)`

- Choice A** `radioButtons(inputId, label, choices, selected, inline)`
- Choice B**
- Choice C**

- Choice 1** `selectInput(inputId, label, choices, selected, multiple, selectize, width, size) (also selectizeInput())`
- Choice 2**

- slider** `sliderInput(inputId, label, min, max, value, step, round, format, locale, ticks, animate, width, sep, pre, post)`

- Apply Changes** `submitButton(text, icon) (Prevents reactions across entire app)`

- Enter text** `textInput(inputId, label, value)`



```
library(shiny)

shinyUI(fluidPage(

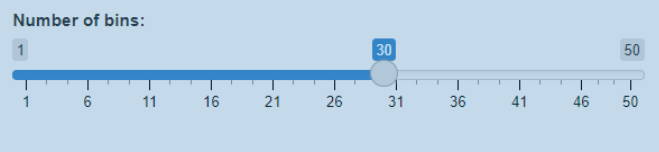
  titlePanel("Old Faithful Geyser Data"),

  sidebarLayout(
    sidebarPanel(
      sliderInput("bins", "Number of bins:", min = 1, max = 50, value = 30)
    ),

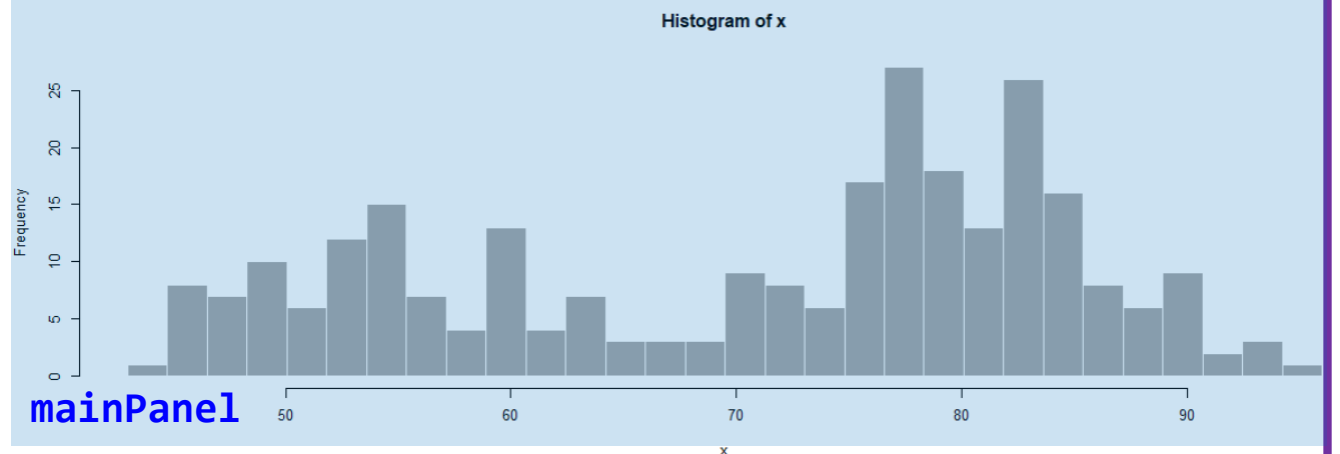
    mainPanel(
      plotOutput("distPlot")
    )
  )
))
```

titlePanel — Old Faithful Geyser Data

sidebarPanel —

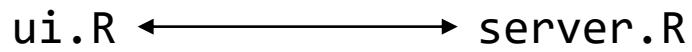


sidebarPanel



mainPanel

# ui.R ↔ server.R



Input:

```
sliderInput("bins", "Number of bins:",
min = 1, max = 50, value = 30)
```

Output:

```
plotOutput("distPlot")
```

shinyServer(function(input, output) {

```
  output$distPlot = renderPlot({
```



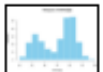



input內容要在render區才能互動

```
    x      = faithful[, 2]
    bins  = seq(min(x),max(x),length.out = input$bins + 1)
    hist(x, breaks = bins, col = 'darkgray')
```

```
  })
```

```
})
```

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

	<b>DT::renderDataTable</b> (expr, options, callback, escape, env, quoted)	works with	<b>dataTableOutput</b> (outputId, icon, ...)
	<b>renderImage</b> (expr, env, quoted, deleteFile)		<b>imageOutput</b> (outputId, width, height, click, dbclick, hover, hoverDelay, hoverDelayType, brush, clickId, hoverId, inline)
	<b>renderPlot</b> (expr, width, height, res, ..., env, quoted, func)		<b>plotOutput</b> (outputId, width, height, click, dbclick, hover, hoverDelay, hoverDelayType, brush, clickId, hoverId, inline)
	<b>renderPrint</b> (expr, env, quoted, func, width)		<b>verbatimTextOutput</b> (outputId)
	<b>renderTable</b> (expr, ..., env, quoted, func)		<b>tableOutput</b> (outputId)
<b>foo</b>	<b>renderText</b> (expr, env, quoted, func)		<b>textOutput</b> (outputId, container, inline)
	<b>renderUI</b> (expr, env, quoted, func)		<b>uiOutput</b> (outputId, inline, container, ...) & <b>htmlOutput</b> (outputId, inline, container, ...)

## renderTmap ↔ tmapOutput

## 實作練習

- 讀取 GTMA.shp 的資料

### Practice 1

- input (sidebar)
  - sliderInput: 調整分類數量
- output (main)
  - plotOutput: 各行政區病例數直方圖

# 實作參考



app1

```
#ui.R

library(shiny)
shinyUI(fluidPage(
  titlePanel("Spatial Analysis App 1"),

  sidebarLayout(
    sidebarPanel(
      sliderInput("bins", "Number of bins:", 3, 7, 5)
    ),

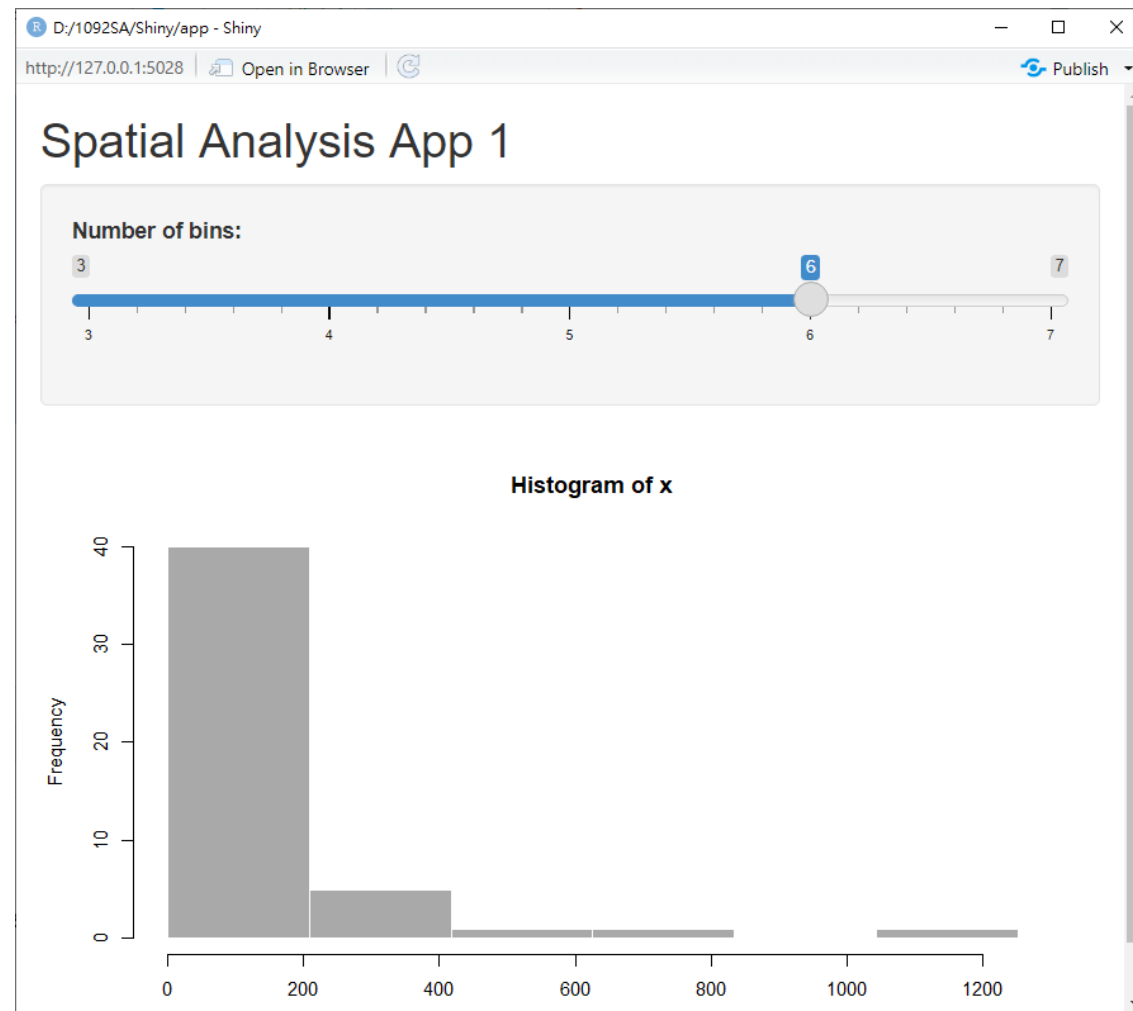
    mainPanel(
      plotOutput("histPlot")
    )
  )
))
=====
#server.R

TW=st_read("GTMA.shp")

shinyServer(function(input, output) {

  output$histPlot=renderPlot({
    x = TW$Cases
    bins = seq(min(x), max(x), length.out = input$bins + 1)
    hist(x, breaks = bins, xlab="Cases")
  })

})
```



# Input

```
sidebarLayout(  
  sidebarPanel(  
    sliderInput("bins", "Number of bins:", min = 1, max = 50, value = 30),  
  
    numericInput("num", "Numder Input", 50, 1, 100, 1),  
  
    radioButton("color", "radioButtons", c("A"="black", "B"="red", "C"="blue"),  
      "red"),  
  
    selectInput("select", "select", c("A"="black", "B"="red", "C"="blue"), "blue"),  
  
    checkboxGroupInput("check", "check", c("A", "B", "C", "D"), c("A", "D")),  
  
    dateInput("date", "date", "2021-04-30", "2021-01-01", "2021-12-31")  
  ),  
  mainPanel(.....)  
)
```

Number of bins:

1 30 50

1 6 11 16 21 26 31 36 41 46 50

Numder Input

50

radioButtons

A

B

C

select

C

check

A

B

C

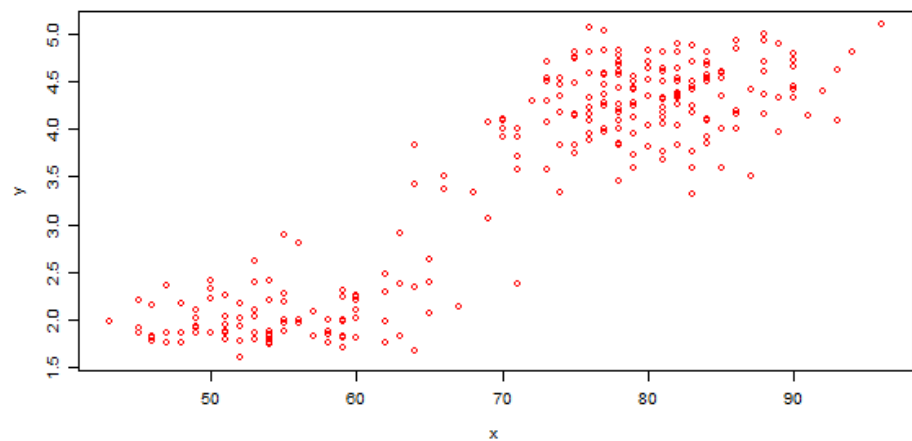
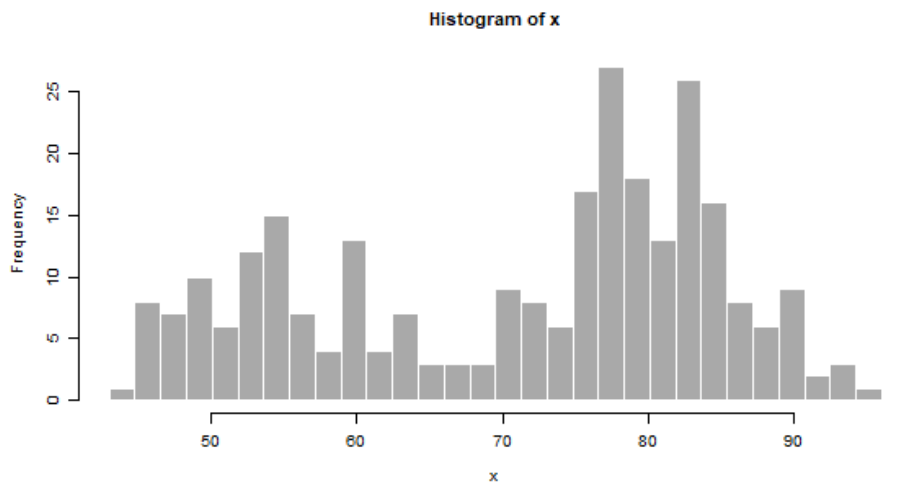
D

date

2021-04-30

# Output

```
mainPanel(  
  plotOutput("distPlot"),  
  plotOutput("dotPlot")  
)
```



```
mainPanel(  
  tabsetPanel(  
    tabPanel("Histogram", plotOutput("distPlot")),  
    tabPanel("Scatter Plot", plotOutput("dotPlot")),  
    tabPanel("DataTable", DT::dataTableOutput("dataTable"))  
  )  
)
```

The UI displays three tabs: "Histogram", "Scatter Plot", and "DataTable". The "DataTable" tab is active, showing a table with 10 rows and 3 columns. The columns are labeled "x" and "y". The table has a search bar and a "Show 10 entries" dropdown. The data is as follows:

	x	y
1	79	3.6
2	54	1.8
3	74	3.333
4	62	2.283
5	85	4.533
6	55	2.883
7	88	4.7
8	85	3.6
9	51	1.95
10	85	4.35

Showing 1 to 10 of 272 entries

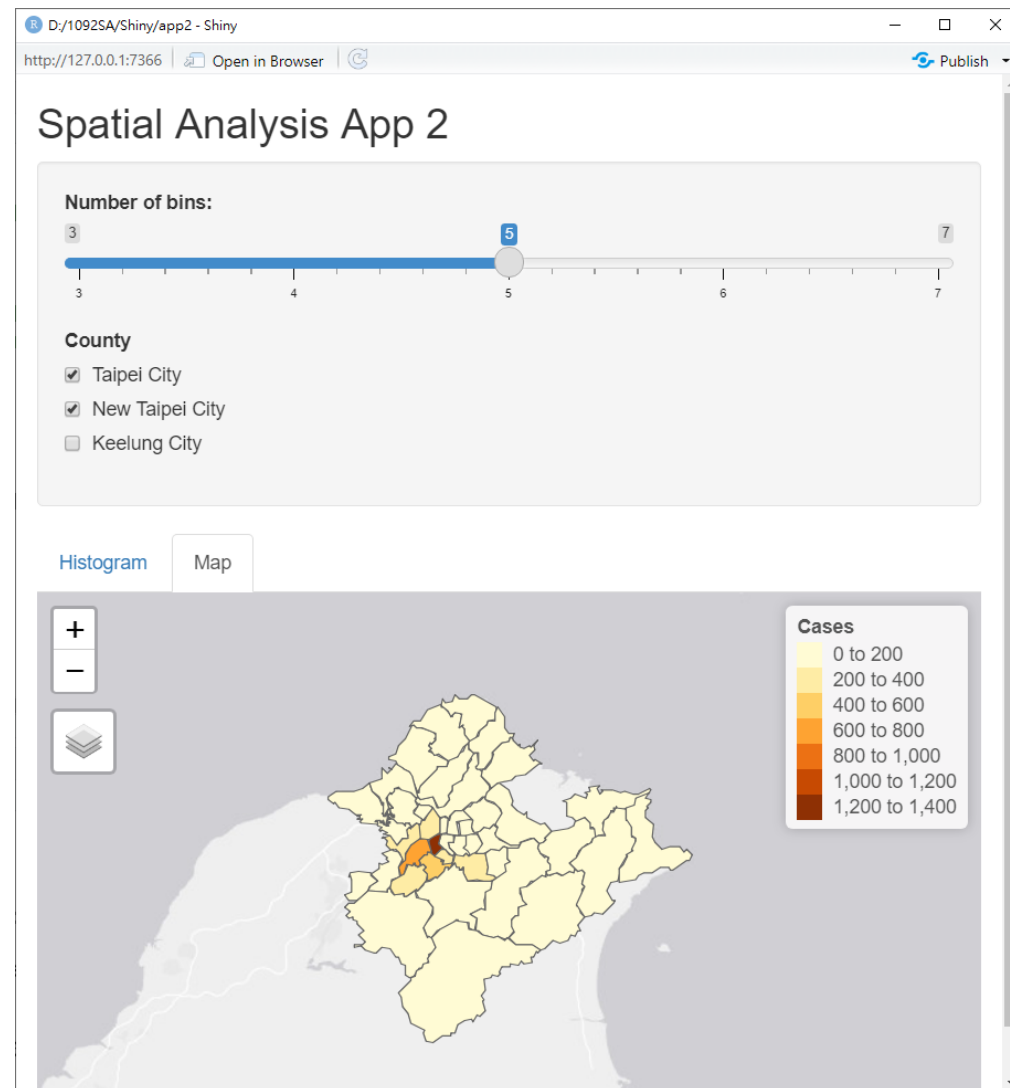
Navigation: Previous 1 2 3 4 5 ... 28 Next

# 實作練習

- 讀取 GTMA.shp 的資料

## Practice 2

- input (sidebar)
  - sliderInput: 調整分類數量
  - checkboxGroupInput: 選擇縣市
- output (main)
  - 兩個tabs
    - plotOutput: 各行政區病例數直方圖
    - tmapOutput: 面量圖







```
#ui.R

library(shiny)

shinyUI(fluidPage(
  titlePanel("Spatial Analysis App 1"),
  sidebarLayout(
    sidebarPanel(
      sliderInput("bins", "Number of bins:", min = 3, max = 7, value = 5),
      checkboxGroupInput("county", "County",
        c("Taipei City" = "63000", "New Taipei City" = "65000",
          "Keelung City" = "10017"), c(63000, 65000, 10017))
    ),
    mainPanel(
      tabsetPanel(
        tabPanel("Histogram", plotOutput("histPlot")),
        tabPanel("Map", tmapOutput("map"))
      )
    )
  )
))
```

```
#server.R

library(shiny); library(sf); library(tmap)

TW = st_read("GTMA.shp")

shinyServer(function(input, output) {

  output$histPlot = renderPlot({
    TWx = TW[TW$COUNTY_ID == input$county, ]
    x = TWx$Cases
    bins = seq(min(x), max(x), length.out = input$bins + 1)
    hist(x, breaks = bins, xlab = "Cases")
  })

  output$map = renderTmap({
    TWx = TW[TW$COUNTY_ID == input$county, ]
    qtm(TWx, 'Cases')
  })
})
```



## 解析

- 資料輸入
  - 行政區
  - 疫情資料：url
- input
  - dateRangeInput: 查詢時間
  - radioButtons: 行政區標籤
- output
  - 篩選date
  - 呈現標籤 → `if(input$label=="ALL") .....`
  - leafletOutput
    - <https://rstudio.github.io/leaflet/>
    - <https://chenhsuantu.github.io/1082SA/leaflet.html>