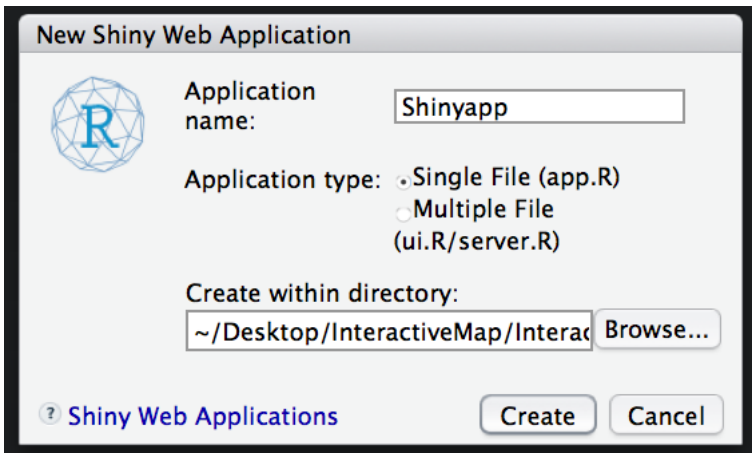
7. Developing Interactive Data Visualisation Web Applications using R and Shiny

Introduction

Shiny is an environment that allows interactive dataviz web applications to be created in R. It is best to illustrate the power of Shiny using the built in template for developing shiny apps provided in R Studio.

Open **R Studio** and select **New File** and then **Shiny Web app**. Enter a name for the app and select **Single File (app.R)** and then the directory path as shown below.



Select **Create** and code from a built-in template is shown as seen in Figure 7.2. This code creates a visualisation in the form of a histogram of the time between eruptions of *Old Faithful Geyser* as shown in Figure 7.1. This interface was generated by selecting **Run app** from the menu bar on the right of the **R Studio** home page.

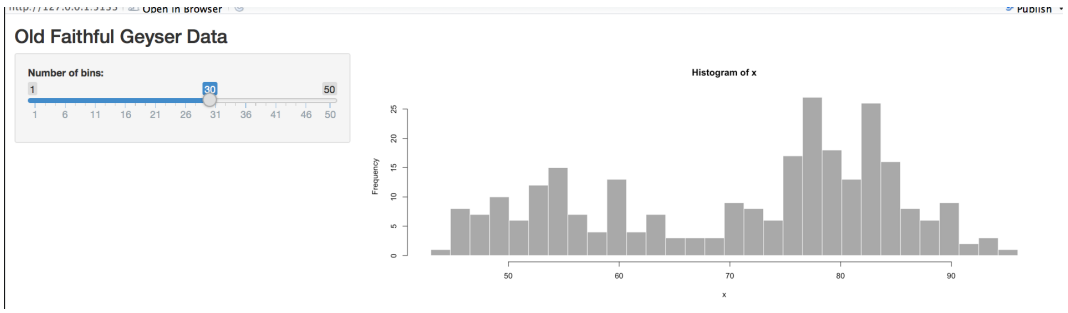


Figure 7.1: Shiny app for the time between eruptions of *Old Faithful*

```
#
# 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
   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")
      )
   )
)

# 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')
   })
}

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


```

Figure 7.2: Shiny app code to generate a histogram of the times for the eruption of *Old Faithful*

Shiny apps have two main components known as the **ui** and the **server**.

The last line of a shiny code combines these two elements as shown in line 49 in Figure 7.2.

The **ui** sets up the code for the **user interface** allowing users to input data using a variety of input devices known as **widgets**. There are a large number of widgets available which including **radio buttons**, **check boxes** and, in this example, **slider inputs**. A listing of some available widgets is shown below in Figure 7.3 which is taken from the Shiny website:
https://shiny.rstudio.com/gallery/widget-gallery.html

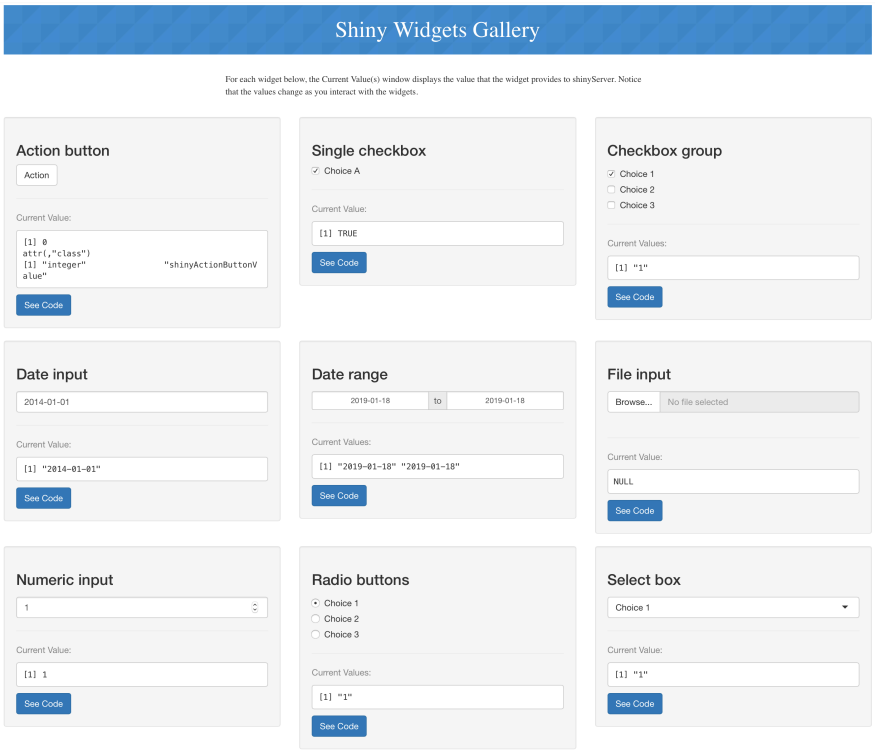


Figure 7.3: Examples of widget shiny inputs

The functions required for calling the above widgets are shown in Table 7.1

function	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

Table 7.1:Widget function calls

In this example the function call is the widget **sliderInput**.

7.2 Ui Component of Shiny App

To include a widget in your app, place the widget function in **sidebarPanel** or the **ui** section. Each widget function requires several arguments which can be seen by selecting **see code** in the widgets gallery.

The first two arguments for each widget are a **name** and a **label**.
In this example the widget name is called **bins** which should be a character string. The user will not see **bins** but its value is is used in the server code where it is pointed to as **input$bins**.

The second argument is a **label.** This label will appear with the widget in the interface. It should also be a character string, but it can be an empty string "".
In this example the label is "**Number of bins**".

The remaining arguments vary from widget to widget, depending on what the widget needs to do its job. They include things like initial values, ranges, and increments. You can find the exact arguments by reference to the above widget gallery.

Alternatively, for more information type **?sliderInput** into R studio and the information below is returned together with an explanation of each argument:

```
sliderInput(inputId, label, min, max, value, step = NULL, round = FALSE,
  format = NULL, locale = NULL, ticks = TRUE, animate = FALSE,
  width = NULL, sep = ",", pre = NULL, post = NULL, timeFormat = NULL,
  timezone = NULL, dragRange = TRUE)
```

In this example the panel on the left of the ui is created from the code shown in lines 13 to 26 in Figure 7.2. Using the **fluidpage** function the title of the panel is given on line 16. **fluidpage** scales rows and column components to fill all available browser width - again enter **?fluidpage** for more details.

The sidebar with slider input is created using the code in lines 19 to 26 while lines 29 to 30 specify the output plot as **distPlot.** The output plot will also reside in the servercode where it is referred to as **output$distPlot**.

The function mainPanel defines what output will be displayed in the main panel of the ui. There are a large number of possible output formats including **textOutput** and **TableOutput**. Further details of can be found in the shiny **cheat sheet** in Appendix 1.

7.3 Server Component of Shiny App

The **server code** is where all the heavy lifting is done and line 36 is always the first line on the server code. Line 38 uses the function **renderplot** to draw the histogram plot. Line 40 points the 272 data points of the time taken for *Old Faithful* to erupt to a variable **x** while Line 41 is the code use to create the number of bins which is assigned as the variable name **bins**. Note this is an important line as it **connects** the ui input (bins) with the output server code (length.out) using the snippet **length.out = input$bins +1**. Line 44 draws the histogram while line 49 is the code required to run the application.

There are a large number of resources for learning shiny that are provided on R Studio site including a number of video presentations and the **Shiny cheatsheet**. Please consult these sources for more information.

Example 2 Shiny interactive web app using Leaflet

In the example on we will develop a web app which visualises road traffic accidents in Dublin using the html widget **leaflet** and **Shiny**. Figure 7.4 is a visualisation of road traffic accidents which we developed in Section 4.2 using the package leaflet. However leaflet has limited interactive capability so we will now use shiny with leaflet to allow users greater interactivity by allowing users to filter accidents by the type of accident.

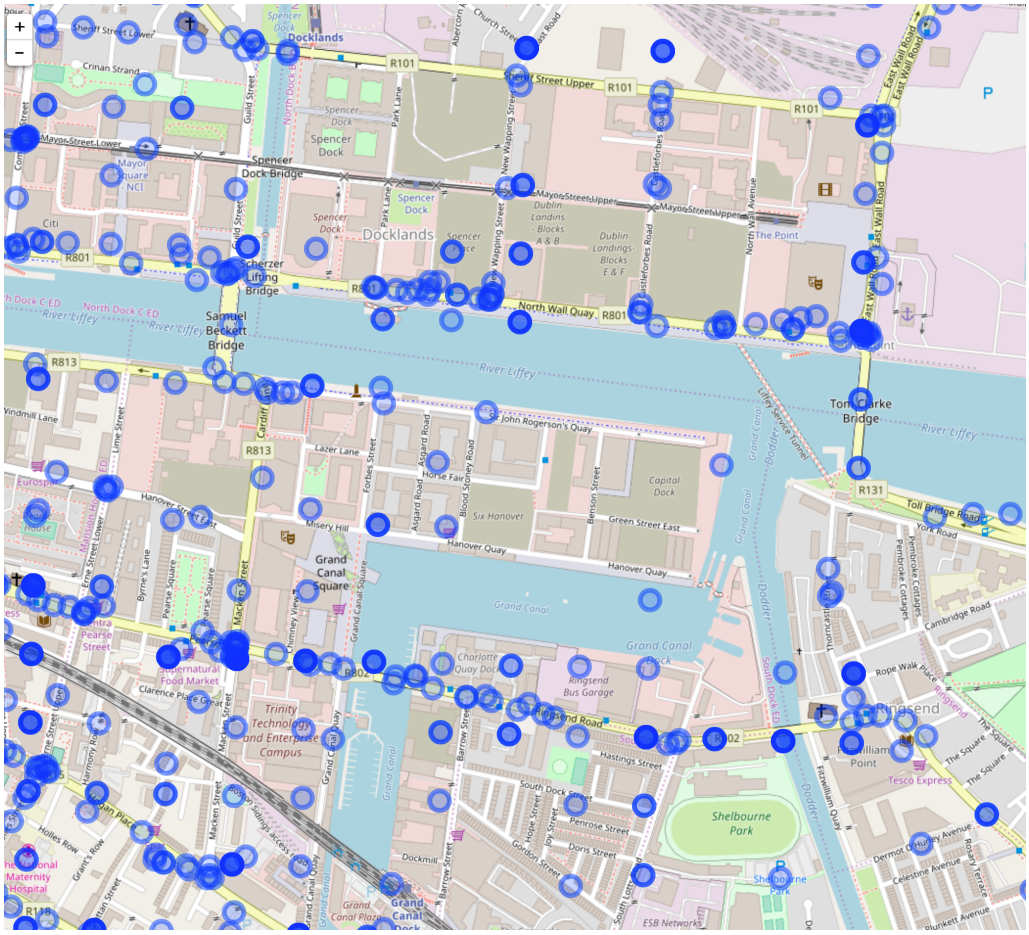


Figure 7.4: Road Traffic Accidents in Dublin City using leaflet

```
1  # This is the user-interface definition of a Shiny web application. You can
2  # run the application by clicking 'Run App' above.
3  #
4  # Find out more about building applications with Shiny here:
5  #
6  #    http://shiny.rstudio.com/
7  #
8
9  library("shiny")
10 library("leaflet")
11 library("tidyverse")
12
13 qpal <- colorFactor("Paired", DubRTA3$Type)
14
15 ui = fluidPage(
16   titlePanel("Accidents in Dublin"),
17   sidebarLayout(
18     sidebarPanel(
19       selectInput("type", "Accident Type",
20                   choices=c("Pedestrian","Other Single Vehicle","Head - On Conflict","Rear End, Straight",
21                             "Angle, Both Straight","Angle, Right Turn","Rear End, Right Turn","Other")),
22       hr(),
23       helpText("Data from RSA")
24     ),
25     leafletOutput("map")
26   )
27
28 )
29
30 server = function(input, output) {
31
32   output$map <- renderLeaflet({
33     DubRTA = DubRTA3[DubRTA3$Type == input$type, ]
34     print(input$type)
35
36     DubRTA %>%
37       leaflet() %>%
38       addTiles() %>%
39       addCircleMarkers(lat = ~Latitude,lng = ~Longitude) %>%
40       setView(lng = -6.157,lat = 53.289, zoom = 14)  %>%
41       addLegend(pal = qpal,value = ~Type, title = "Accident Type")
42   })
43 }
44
45 # Run the application
46 shinyApp(ui = ui, server = server)
47
```

Figure 7.5: Code used to generate interactive shiny app of road traffic accidents in Dublin

The create an interactive map that allows users to filter on accident type we can use the code shown in Figure 7.5. As in Example 1 the ui is defined from lines 15 to 28. In this example the widget is **SelectInput** which allows users to select a specific accident type from a dropdown menu with choices listing all the possible accident types. On line 25 **plotOutput** is changed to **leafletOuput** as we are using leaflet with Shiny. Similarly the output server code on line 32 lists the function **Renderleaflet** rather than renderPlot.

The datafile with the road accidents is called **DubRTA3** which contains five variables called Latitude, Longitude, County, Type and Year. The variable **Type** is the accident type associated with each accident. Line 33 directly assigns **Type** with the input id **type** thus connecting the selection of accident type the user makes in the ui with the output map. DubRTA is then assigned as the name of the file which is passed to lines 41 to 48 where the leaflet code computes the map according to the user selected accident type.

Running this code on line 46 we obtain the visualisation of pedestrian accidents in Dublin City. The user can select any of the other accident types and the graphic will update. For example the lower image in Figure 7.6 plots accidents classified as *other*.

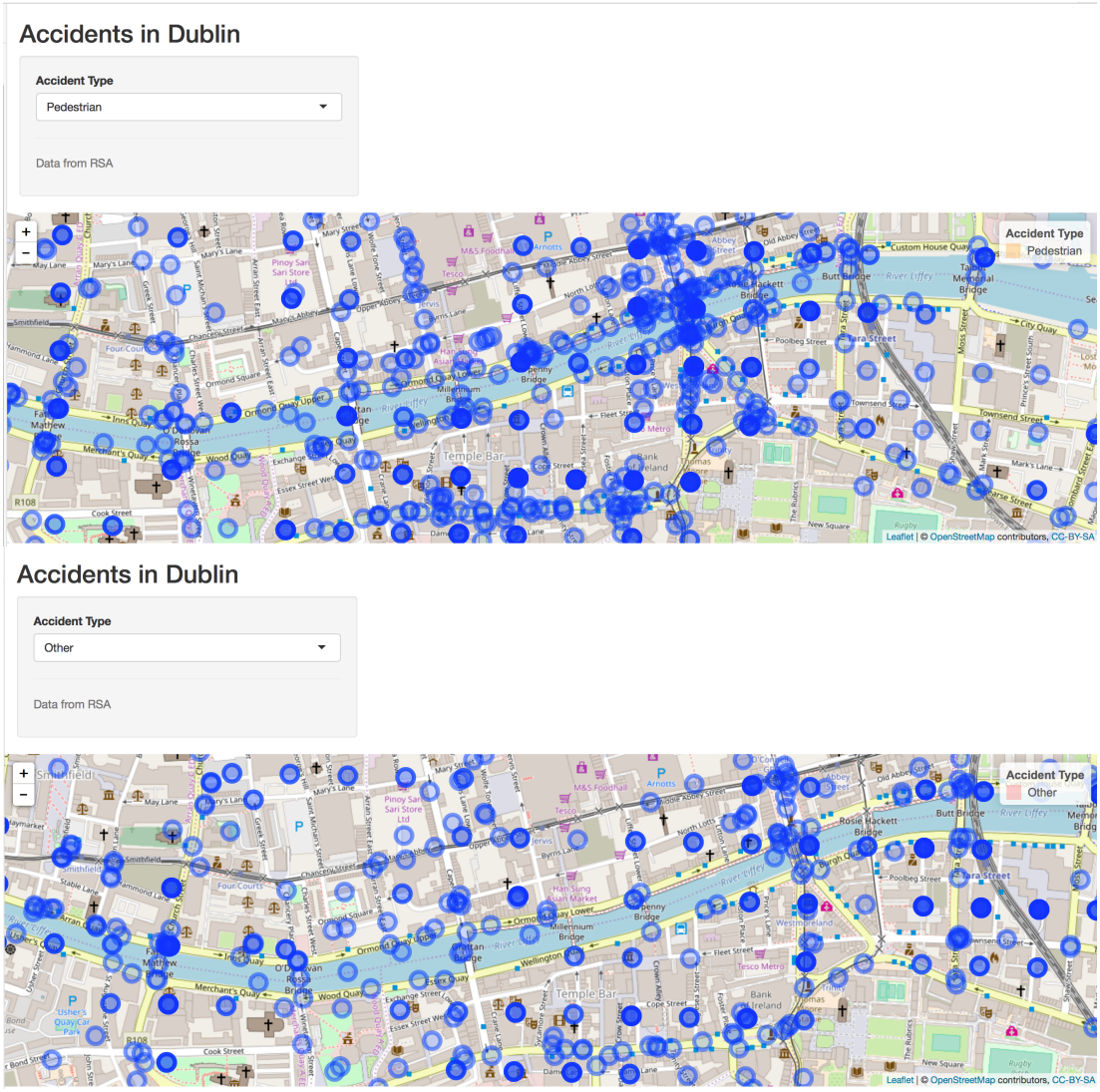


Figure 7.6 Visualisation of Pedestrian (top) and other (bottom) road traffic accidents in Dublin City Centre.

It is also possible to host the developed apps on a webpage using **RPubs**, or place them in **R Markdown** documents. The web apps can also also be hosted on **shiny apps** server for a fee while the apps can be embedded in **shiny dashboards**.

Exercises

1. The data in the Excel worksheet **ecoli4D** in *ExerciseData(2018)* records location, gender and diagnosis of three diseases Cryptosporidiosis, Verotoxigenic Escherichia coli infection and Giardiasis during 2016 and 2017 in the Greater Dublin Area. The location data has been jittered by adding noise to the longitude and latitude of each location in the form of a standard normal probability distribution. Please note that the data set is for illustrative purposes only as it excludes some locations for which the longitude and latitude were not determined.

Create a shiny app that visualises the geographical distribution of ecoli in the Greater Dublin Area. The app should contain filters that allows the user to select values of the variables diagnosis. Can you experiment with the code and see if you can also create an additional filter for gender that will allow users to filter disease by gender?

2. The data in the Excel worksheet **tourism** in *ExerciseData(2018)* contains the variables:

Variable Name	Description
Holiday trips	Number of holiday trips
Business trips	Number of business trips
VFR trips	Number of trips involving **V**isits to **F**riends and **R**elatives
Town	Name of Town Visited
Longitude	Column Value (x-axis)
Latitude	Row Value (y-axis)
Country of Residence	Britain (value 2), North American (value 3) and Mainland Europe (value 4)
Brand*	Wild Atlantic Way, Ireland's Ancient East and Dublin - A Breath of Fresh Air

The research department of Failte Ireland are interested in determining if the geographic distribution of visitors to Ireland varies according to country of residence. They would also like to obtain further insight on how visitors to the tourism brands (Wild Atlantic Way, Ireland's Ancient East and Dublin - A Breath of Fresh Air) vary by segment (business, holiday and VFR) and by country of residence (Britain, North American and Mainland Europe). This exercise will feed into Failte Irelands strategic plans for growing the business in 2019 onwards through identification of potential opportunities for growth.

a) Create a shiny app that visualises the geographical distribution of visitors to Ireland. The app should contain filters that allow the user to select values of the variables **Country of Residence** and **Brand**.

b) Using the results of a) write a short note that summarises the distribution of tourists by **Country of Residence** and **Brand**.