Data Analytics Software Development via RShiny and RMarkdown[1]

Kayvan Nejabati Zenouz[2]

University of Greenwich

November 2, 2021

[1] Online Version: www.nejabatiz.com/SWRSM.pdf

[2] Office: QM315, Email: K.NejabatiZenouz@greenwich.ac.uk,
Student Drop-in Hours: MONDAYS 12:00-13:00 (MATHS ARCADE/TEAMS) AND TUESDAYS 14:00-15:00 (QM315/TEAMS)

Content I

Intended Learning Outcomes

By the end of the session you will...

1. Understand the uses of RShiny and RMarkdown in data science industry
2. Look at applications designed for
   - Interactive data analysis of Incurred But Not Reported insurance claims
   - Automated web browsing and web scrapping used for price optimisation
3. Create RShiny application in order to analyse/visualise data

Class Activity with www.menti.com

Please **scan** the barcode with your **phone** in order to take part in the class activity.

https://www.menti.com/r69f96tc6u

Alternatively, go to www.menti.com on your electronic devices and use the code provided.

Introduction

- The programming language R was developed around 1993 - it is object orientated and open source
- R has become a powerful tool used by **statistician** and **data scientists** used for
  - Data analysis and visualisation
  - Statistical modelling
  - HTML application development
  - Automated web-browsing, and many other tasks

How to Get R and its GUI

1. Go To www.r-project.org and to install R click on **download R** in the first paragraph
2. Select a **server** from the list, download and **install** R for your operating system
3. For GUI go To www.rstudio.com and click on **download RStudio**
4. Select a free version, your operating system, and install
5. Alternatively, you may use **RStudio Cloud**

RStudio GUI

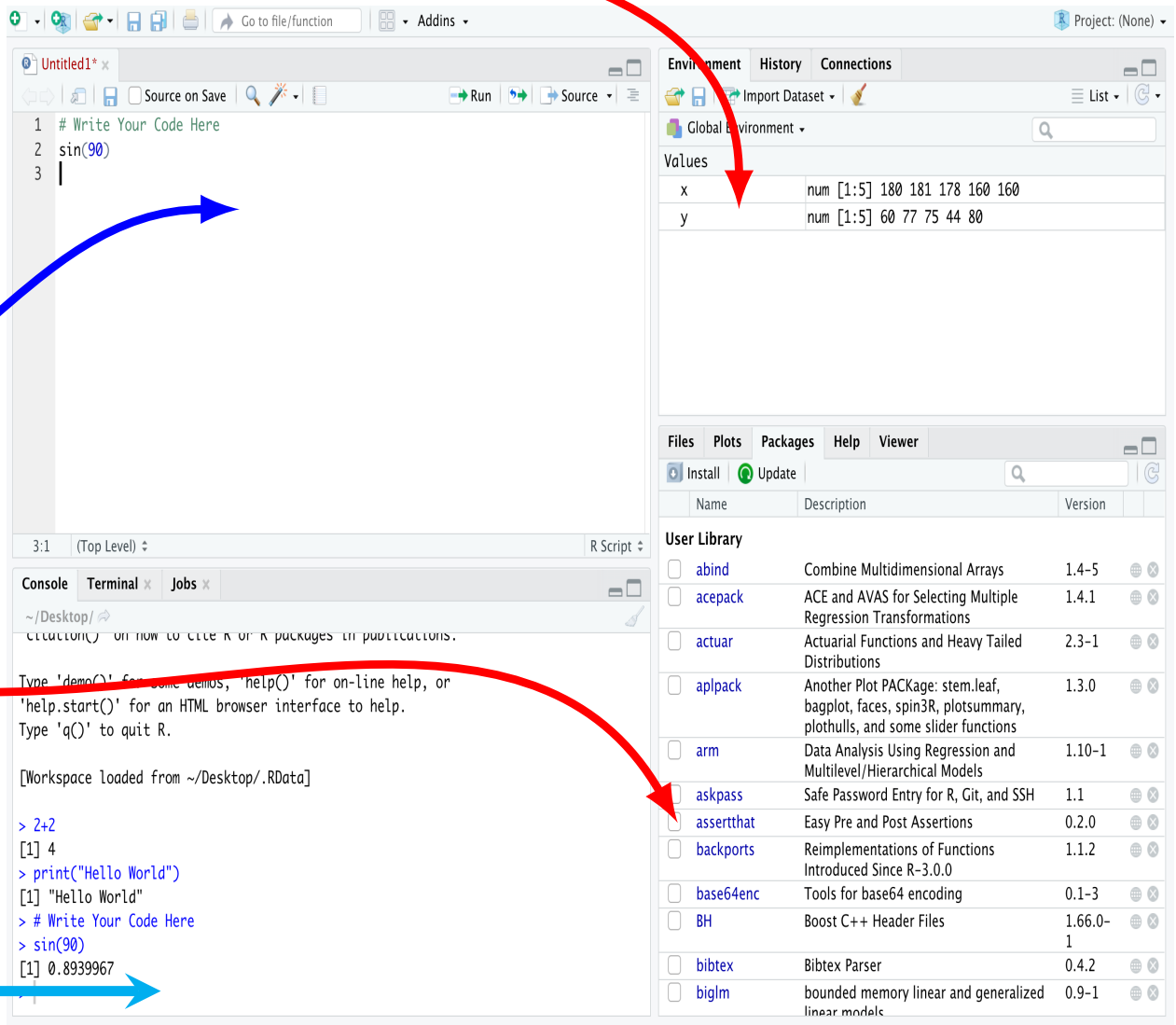
See datasets and variables

R script

'cmd/ctrl+enter' to run

Plots and packages

R Console



Shiny Package for R

- **Shiny** offers functionality to produce **interactive HTML** applications
- You can host standalone apps on a **webpage**, embed them in **RMarkdown** documents, or build **dashboards**
- You can add **CSS themes**, **htmlwidgets**, and **JavaScript** actions
- Through Shiny you combine **statistics** and **interactivity**
- The best place to learn about Shiny is the **website** https://shiny.rstudio.com/

Components of a Shiny App

A shiny application has **three components**

```
library(shiny) # Load Shiny Package
ui <- ... # definitions of user interface

server <- ... # definitions of server

shinyApp(ui = ui, server = server) # Create Shiny app
```

One way to **create** a **shiny** application is to define an *interface*, **server**, and **call shiny** to join them together.

Minimal Shiny app

The following **example** defines a Shiny to show a 1000 samples from a standard normal distribution.

```
library(shiny) # Load Shiny Package
ui <- fluidPage( # Create a HTML page ----
  titlePanel("Hello Shiny!"), # App title ----
  # Output: Define Histogram ----
  plotOutput(outputId = "distPlot")
  )
# Define server logic required fill the UI ----
server <- function(input, output) { # Server
  output$distPlot <- renderPlot({ # Create Histogram Plot
  hist(rnorm(1000,0,1), col = "#75AADB", border = "white",
  xlab = "x",
  main = "Histogram Normal Distribution")
    })
  }
# Create Shiny app ----
shinyApp(ui = ui, server = server)
```

Shiny in-Built Examples

You can see more basic Shiny examples.

```
library(shiny) # Load Shiny Package
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") # 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
```

Advanced Industry Examples I (IBNR)

Definition (IBNR)

In **insurance** industry, **Incurred But Not Reported** (IBNR) refers to the **claims** not yet known to the insurer, but for which liability is thought to exist.

- This is natural as not all insurance claims are reported immediately
- Insurers need to **estimate** the number and amount of claims that are likely to arrive in order to reserve enough capital
- A prominent method in actuarial loss reserving is **Chain-Ladder development** which is used in property and casualty and health insurance.

Industry Problem

Industry Problem 1

Given **claim data** to date, **estimate accurately** the **number** and **severity** of Incurred But Not Reported claims.

IBNR Row Data

- Data is recorded by operators registering the claim as they are notified sometime after the incident has taken place
- A sample of data recorded over 2 days looks like the following table

Claim	Notified On	Loss On	Reserve	Incurred in £
1	11/10/2013	08/10/2013	200.00	50.05
2	11/10/2013	28/09/2013	7500.00	1,091.66
3	11/10/2013	26/09/2013	5000.00	63.28
4	11/10/2013	08/10/2013	2000.00	2,280.00
5	14/10/2013	08/10/2013	1500.00	10,685.43
6	14/10/2013	07/10/2013	198.00	97.31
7	14/10/2013	07/10/2013	372.00	162.05
8	14/10/2013	05/10/2013	2000.00	653.39
9	14/10/2013	08/10/2013	7500.00	201.06
10	14/10/2013	07/10/2013	8000.00	15,192.93
⋮	⋮	⋮	⋮	⋮

IBNR Triangles

Data is **aggregated** and put into **incremental** claim developments triangles

Table: Number of Claims

Loss On YM	Month 0	Month 1	Month 2	Month 3	Month 4	Month 5
201302	260	533	173	44	14	15
201303	345	546	94	50	18	?
201304	314	288	146	39	?	?
201305	301	472	196	?	?	?
201306	445	533	?	?	?	?
201307	516	?	?	?	?	?

Table: Severity of Claims

Loss On YM	Month 0	Month 1	Month 2	Month 3	Month 4	Month 5
201302	749656	344462	114761	21215	4118	4419
201303	786987	317614	62299	46992	6347	?
201304	523443	291967	132184	30692	?	?
201305	485051	635845	135094	?	?	?
201306	878307	545531	?	?	?	?
201307	957221	?	?	?	?	?

The goal is to **estimate** spaces filled with **question marks**

Cumulative tables

Triangles are converted into **cumulative** triangles.

Table: Number of Claims

Loss On YM	Month 0	Month 1	Month 2	Month 3	Month 4	Month 5
201302	260	793	966	1010	1024	1039
201303	345	891	985	1035	1053	
201304	314	602	748	787		
201305	301	773	969			
201306	445	978				
201307	516					

Table: Severity of Claims

Loss On YM	Month 0	Month 1	Month 2	Month 3	Month 4	Month 5
201302	749656	1094117	1208878	1230093	1234211	1238630
201303	786987	1104600	1166899	1213892	1220239	
201304	523443	815410	947594	978286		
201305	485051	1120896	1255990			
201306	878307	1423837				
201307	957221					

Age-to-Age Link Ratios

Calculate the **age-age ratios**.

Table: Number of Claims

Loss On YM	Month 0-1	Month 1-2	Month 2-3	Month 3-4	Month 4-5
201302	3.05000	1.21816	1.04555	1.01386	1.01465
201303	2.58261	1.10550	1.05076	1.01739	
201304	1.91720	1.24252	1.05214		
201305	2.56811	1.25356			
201306	2.19775				
201307					

Table: Severity of Claims

Loss On YM	Month 0-1	Month 1-2	Month 2-3	Month 3-4	Month 4-5
201302	1.45949	1.10489	1.01755	1.00335	1.00358
201303	1.40358	1.05640	1.04027	1.00523	
201304	1.55778	1.16211	1.03239		
201305	2.31088	1.12052			
201306	1.62112				
201307					

Finally, **averages** of the age-to-age factors are calculated and estimates are arrived at

IBNR Shiny App

- This procedure of estimating IBNR is **implemented** in R as a package using several different Chain-Ladder methods
- The package has many features as well as visualisations
- A Shiny app was produced where claim data .csv file can be uploaded
- The app manipulates data and uses the Chain-Ladder package to do the following
  1. Visualise the development of claims
  2. Creates incremental/cumulative triangles
  3. Arrive at accurate estimates using a suitable Chain-Ladder
  4. Complete the cumulative triangle and produce uncertainties
  5. Create visualisation to validate the estimations
- This hugely **reduces** the amount of work required to create IBNR report every month!

Definition (Web Scraping)

Web scraping is a process used for **extracting data** from a **website**.

Example

Your manager asks you to compare the prices of the companies' travel insurance products with a rival company and gives you the task of extracting 1000 quotes everyday from the rival companies website.

You are likely to be doing this for the next 20 years!

Industry Problem 2

What exactly do you need to do?

- Everyday come up with 1000 customer details who what to travel. That is, you need a 1000 from each of the following.
  - Insurance product information: Single/Annual trip
  - Date of cover start
  - Date of cover end
  - Destination of travel
  - Number of people travelling
  - Age of passengers
- You need to enter these details into the rival companies website and press the quote button
- Record the prices that the website shows and create a spreadsheet
- Compare the prices with your company's prices and make recommendation to your manager

Solution: Be Lazy!

- Study the website carefully, get a **few quotes** to understand the procedure
- Create **fake data** in R to be used for quotes (for a simple case single individual)
- Use RSelenium package which provides driving a web browser natively as a user would - Selenium **automates** web browsers (this works with Docker)
- Package rvest allows to **scrape** (or harvest) data from html web pages and xml2 allows to work with XML and HTML in R
- Save the data from the web as data frame in R
- Create a shiny app which encapsulates this all and by pressing a button obtains a given number of quotes and saves the data as a .csv file

Fake Data Creation

```
NumberOfQ <- 1000 # Number of quotes required
SampleStartDate <- Sys.Date()+1
SampleEndDate <- SampleStartDate+100
dapsdate <- sample(seq(as.Date(SampleStartDate), as.Date(SampleEndDate),
     by="day"), NumberOfQ) # Departure Date
dapedate <- dapsdate+sample(1:50,NumberOfQ, replace = TRUE)
# Departure Date + Duration
DepartureDate <- format(dapsdate, format="%d/%m/%Y")
ReturnDate <- format(dapedate, format="%d/%m/%Y")
ERVWebPrim <-cbind.data.frame(DepartureDate, ReturnDate, stringsAsFactors
     =FALSE)
ERVWebPrim$Destination <- sample(c("Destination_7","Destination_9", "
     Destination_2", "Destination_10", "Destination_17"), NumberOfQ,
     replace = TRUE)   # Europe1,2 - 2/10, WX- 7, W - 9, A/N - 17
ERVWebPrim$TravellerType <-"TravellerType_1"
# Individual 1, 2 Couple, 3 Family, 4 Single Parent
TripType <- "TripType_1" # Single 1, "TripType_2" for Annual
# Number of Passegners 1
# Number of Children 0
ERVWebPrim$PassAge1 <- sample(18:75,NumberOfQ, replace = TRUE)
# Age of Passenger
```

Academic Research Problem

Research Problem 1: Seasonality in UK Crime Data

How do we understand the patterns in UK monthly crime numbers and how to create accurate models and predictions for these.

Steps to Follow

1. **State** the problem to be solved clearly
2. Create **fully functioning R script** that solves the problem independent of any app
3. Identify the **interactive elements** of your shiny
4. **Visualise**, in your mind, the app's appearance, buttons, and functionalities
5. Create the **user interface** ui of the app according to your vision
6. Implement your **R script** in the server part of the app taking into account input from ui

Exercise 1: Interactive Histogram

Normal Histogram app

Create a Shiny app in which user can simulate N random samples from a normal distribution $N(\mu, \sigma^2)$ and the app produces a ggplot histogram of the samples for a given number of bins B. Modify the app produce $\text{Bin}(n, p)$ histogram.

Remark 1: App Spec

- The app has numeric inputs integer N and number of bins B and continuous inputs μ and σ
- For σ and B you may use a sliders with range 0-5 for σ and 1-200 for B
- This app is slightly more complicated than https://shiny.rstudio.com/tutorial/written-tutorial/lesson1/
- You can find a list of basic widgets from https://shiny.rstudio.com/tutorial/written-tutorial/lesson3/
- But first create the R script to produce the histogram

R Script to Create ggplot Histogram

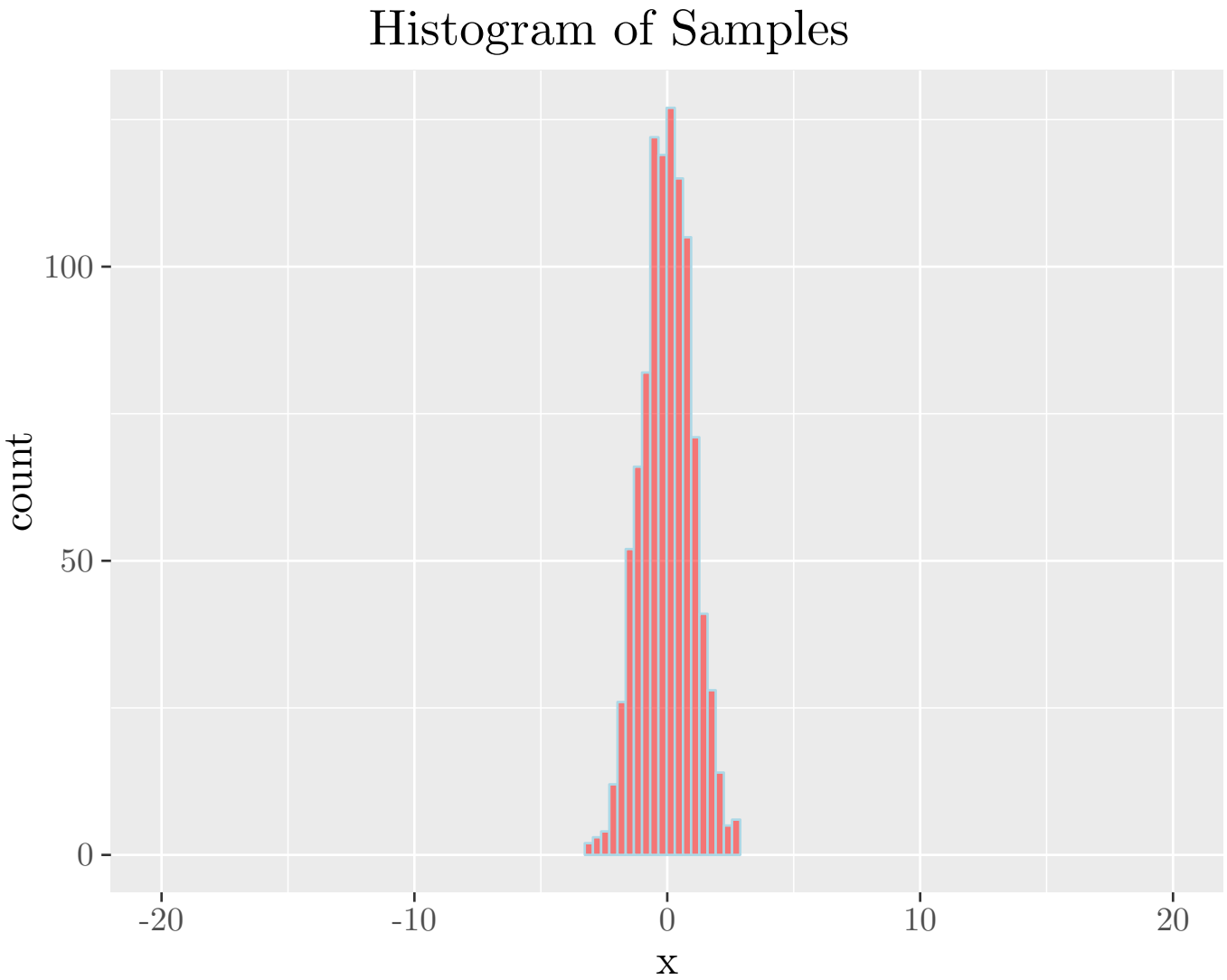
```
library(ggplot2) # ggplot Library
library(plotly) # plotly for further interactivity

N <- 1000 # To be changed via the inputId: noOfSamples
mu <- 0 # To be changed via the inputId: mu
sigma <- 1 # To be changed via the inputId: sigma
noOfBins <- 20 # To be changed via the inputId: noOfBins

x <- rnorm(N, mean=mu, sd=sigma) # Create samples
Nsamples <- as.data.frame(x) # Make a dataframe for ggplot
binBreak <- seq(min(Nsamples$x), max(Nsamples$x), length.out = noOfBins)
     # Create bins for histogram

p <- ggplot(data=Nsamples, aes(x=x)) + # Specify data and x axis
geom_histogram(# Specify histogram for ggplot
breaks=binBreak, # Specify bins
alpha=.5, color="light blue", fill="red") +
# Design colours for histogram
xlim(-20,20)+ # Set x limits
labs(title="Histogram of Samples") # Add title
ggplotly(p) # Create plotly interactive object
```

Result



Shiny App

- Need numeric input for N and μ. The **numeric input** has general format

```
numericInput(inputId, label, value, min = NA, max = NA,
     step = NA, width = NULL)
```

- Need sliders for σ and B. The **slider input** has general format

```
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)
```

- We would like the control panel in the sidebar
- Produce histogram in the main panel

The User Interface ui

```
library(shiny); library(ggplot2) # ggplot Library
library(plotly) # plotly for further interactivity
# Define UI for app that draws a histogram ----
ui <- fluidPage(
titlePanel("Exercise 1: Basic Normal Histogram"), # App title
sidebarLayout(# Sidebar layout with input and output definitions
sidebarPanel(# Sidebar panel for inputs ----
# Input: Slider for the number of bins ----
numericInput(inputId = "noOfSamples", label="Number of Samples:",
    value=1000, min = 1, step = 1),
numericInput(inputId = "mu", label="Value for mu:", value=0),
sliderInput(inputId = "sigma", label = "Value for sigma:", value =
     1, min = 0, max = 5, step=0.1,
 animate = animationOptions(100)),
sliderInput(inputId = "noOfBins", label = "Number of bins:", min =
     1, max = 200, value = 20, animate = animationOptions(100))),
mainPanel(# Main panel for displaying outputs ----
plotlyOutput(outputId = "distPlot") # Output plotly: Histogram
    )
))
```

The server

```
# Define server logic required to draw a histogram ----
server <- function(input, output) {
# 1. It is "reactive" and therefore should be automatically
#    re-executed when inputs (input$...) change
# 2. Its output type is a plotly
output$distPlot <- renderPlotly({
N <- input$noOfSamples # Inputs from ui
mu <- input$mu
sigma <- input$sigma
noOfBins <- input$noOfBins
x <- rnorm(N, mean=mu, sd=sigma) # Create samples
Nsamples <- as.data.frame(x) # Make a dataframe for ggplot
binBreak <- seq(min(Nsamples$x), max(Nsamples$x), length.out = noOfBins)
     # Create bins for histogram
p <- ggplot(data=Nsamples, aes(x=x)) + # Specify data and x
geom_histogram(# Specify histogram for ggplot
breaks=binBreak, # Specify bins
alpha=.5, color="light blue", fill="red") + # Design colours
xlim(-20,20)+ # Set x limits
labs(title="Histogram of Samples") # Add title
ggplotly(p) # Create plotly interactive object
})}
shinyApp(ui = ui, server = server) # Create Shiny
```

Exercise 2: Leaflet Map to View UK Crime Data

Leaflet Map app

Create a Shiny app in which user can upload data from https://data.police.uk/data/ to visualise crimes in the UK

- Go to the website above and choose *date range* September 2019 to September 2019
- Check the *City of London Police* option. Click on *Generate file* and in the next page click on *Download now* and save in a suitable location
- Unzip the file and in a folder find a .csv file which contains around 900 crimes occurred in London during September

Remark 2: App Spec

- The app need a file input and a start button
- Once the start button has been pressed a map is created with crimes the location of crime shown on it
- Different colours need to represent different crimes

R Script to Create Leaflet Map

```
library(leaflet) # For producing maps
library(colortools) # For producing colours
crimeData<- read.csv(file.choose(), header = TRUE, stringsAsFactors =
    FALSE) # Choose your file
crimeData<- crimeData[!is.na(crimeData$Longitude) | !is.na(crimeData$
    Latitude), ] # Remove invalid locations
crimeData$Crime.type<-as.factor(crimeData$Crime.type)
# Treat Crime type as a factor
pal <- colorFactor(wheel("tomato",
num = length(unique(crimeData$Crime.type))), # Create colours
domain = unique(crimeData$Crime.type))
m <- leaflet(crimeData) %>% # Use Leaflet to create a map
setView(lng = mean(crimeData$Longitude), lat = mean(crimeData$Latitude),
    zoom = 13)%>%
addTiles()  %>%  # Add default OpenStreetMap map tiles
addCircleMarkers(lng=~Longitude, lat=~Latitude,
# Add circles for crime locations
popup=~Crime.type, # Add popup for crime type
label =~ Crime.type, # Add label for crime type
radius=7, # Circle properties
color = ~pal(Crime.type), # Add colour for crime type
stroke = FALSE, fillOpacity = 1)
m  # Print the map
```

- We need file input. The **file input** has general format

```
fileInput(inputId, label, multiple = FALSE, accept =
    NULL, width = NULL)
```

- Also need action button. The **action button** has general format

```
actionButton("button", "An action button")
```

- Have the control panel in the side bar
- Produce map in the main panel

The User Interface ui

```
library(shiny);options(shiny.maxRequestSize = 900*1024^2)
library(leaflet)
library(colortools)
ui <- fluidPage(
titlePanel("Exercise 2: App to Visualise UK Crime Data"),
# App title ----
sidebarLayout(# Sidebar layout
sidebarPanel(# Sidebar panel for inputs ----
fileInput('file1', 'Upload a .csv Raw Claim Data File',
    accept = c(".csv")), # File input panel
actionButton("startButton","Start")), # Action button
mainPanel(# Main panel for displaying outputs ----
leafletOutput(outputId = "distPlot",  width = "100%",
    height = "700px") # Leaflet Output
    )
  )
)
```

The server I

```
server <- function(input, output) {
dataInput1 <- eventReactive(input$startButton,{
# Event reacts to button press
inFile1 <- input$file1 # Accept the input files
if(is.null(inFile1))
return(NULL)
file.rename(inFile1$datapath,
paste(inFile1$datapath, ".csv", sep=""))
crimeData<- read.csv(paste(inFile1$datapath,
".csv", sep=""), header = TRUE, stringsAsFactors = FALSE)
crimeData<- crimeData[!is.na(crimeData$Longitude) | !is.na(
    crimeData$Latitude), ]
crimeData$Crime.type<-as.factor(crimeData$Crime.type)
return(crimeData)
})
output$distPlot <- renderLeaflet({
crimeData<-dataInput1()
```

The server II

```
# Data is carried from reactive event
pal <- colorFactor(wheel("tomato", num = length(unique(
    crimeData$Crime.type))),
domain = unique(crimeData$Crime.type))
m <- leaflet(crimeData) %>%
setView(lng = mean(crimeData$Longitude),
lat = mean(crimeData$Latitude), zoom = 13)%>%
addTiles()  %>%  # Add default OpenStreetMap map
addCircleMarkers(lng=~Longitude, lat=~Latitude,
popup=~Crime.type, label =~ Crime.type,
radius=7, color = ~pal(Crime.type),
stroke = FALSE, fillOpacity = 1)
m  # Print the map
  })
}
shinyApp(ui, server)
```

Summary: What we did today...

Remark 3: RMarkdown

You can embed your Shiny apps in RMarkdown documents, or create them within RMarkdown HTML with runtime Shiny!

- R Programming — Object-Oriented, Open Sources, Uses
- Basics of R — Downloading and GUI
- Shiny Application — ui, server, shinyApp(), examples
- Industry Examples — IBNR and Web-Scraper
- Exercises — Plotly Histogram and Leaflet
- See You Soon and Good Luck!

See You Next Time

Please Do Not Forget To

- Ask any **questions** now or through my contact details.
- Come and see me during Student Drop-in Hours: MONDAYS 12:00-13:00 (MATHS ARCADE/TEAMS) AND TUESDAYS 14:00-15:00 (QM315/TEAMS).
- Alternatively, email to make an appointment.

Thank You!