STQD6134: Business Analytics

Shiny Part I

Materials from: https://ourcodingclub.github.io/tutorials/shiny/

Introduction

The package is used to create web-applications, but uses the R language rather than javascript or HTML5, which are traditionally used for web applications. By using R, Shiny provides an efficient method of creating web applications designed around data presentation and analysis.

Shiny apps is useful for:

- **Interactive data visualisation for presentations and websites**
- Sharing results with collaborators
- Communicating science in an accessible way
- Bridging the gap between R users and non-R users

Some examples

Education purpose: https://shiny.rstudio.com/gallery/immune-response-modeling.html

Finance / banking: https://shiny.rstudio.com/gallery/career-pathfinder.html

Government: https://shiny.rstudio.com/gallery/nz-trade-dash.html

Download Shiny and related packages

```
install.packages("shiny")
install.packages("rsconnect")  # For publishing apps online
install.packages("agridat")  # For the dataset in today's tutorial
```

Shiny app structure

Next, select *File/ New File/ Shiny Web App…*, give the application a descriptive name (**no spaces**) and change the application type to "*Single File (app.R)*", save the app in an appropriate directory and click *Create*.

RStudio generates a template R script called `app.R`. **Delete all the code in the template so you have a blank script.**

Notice that the name you gave to your app was assigned to the directory, not the app script file. For your app to work, the file must remain named `app.R`!

It is possible to create a Shiny app with two files called `ui.R` and `server.R`, but the same can be accomplished by using one file. In the past, Shiny apps had to be created using two files, but the Shiny package has since been updated to allow the single file app structure, making things much tidier. You will see some tutorials on the internet using the old two file structure, but these can be easily translated to the one file structure. This tutorial will assume you have the one file app structure.

Now we can set up the rest of the folders for your app. Add a folder called `Data` and a folder called `www` in your app directory. `Data` will hold any data used by the app and `www` will hold any images and other web elements.

To review, a Shiny application should have this specific folder structure to work properly:

```
Test_App
├── app.R
├── Data
│   └── data.csv
└── www
    └── A.jpg
```

app.R layout

A basic app.R consists of five sections:

1) A section at the top of the script loading any packages needed for the app to run. shiny is required at the very least, but others like dplyr or ggplot2 could be added as they are needed:

```
# Packages ----
library(shiny)  # Required to run any Shiny app
library(ggplot2)  # For creating pretty plots
library(dplyr)  # For filtering and manipulating data
library(agridat)  # The package where the data comes from
```

2) A section loading any data needed by the app:

```
# Loading data ----
Barley <- as.data.frame(beaven.barley)
```

app.R layout

3) An object called ui, which contains information about the layout of the app as it appears in your web browser. fluidPage() defines a layout that will resize according to the size of the browser window. All the app code will be placed within the brackets.

```
# ui.R ----
ui <- fluidPage()
```

4) An object called server, which contains information about the computation of the app, creating plots, tables, maps etc. using information provided by the user. All the app code will be placed within the curly brackets.

```
# server.R ----
server <- function(input, output) {}
```

app.R layout

5) A command to run the app. This should be included at the very end of app.R. It tells shiny that the user interface comes from the object called ui and that the server information (data, plots, tables, etc.) comes from the object called server.

```
# Run the app ----
shinyApp(ui = ui, server = server)
```

Layout of Shiny apps

- Shiny apps are structured using panels, which are laid out in different arrangements. Panels can contain text, widgets, plots, tables, maps, images, etc.

- Here is a good set of examples on how the panel layout can be changed: https://shiny.rstudio.com/articles/layout-guide.html

- The most basic layout uses fluidRow() and column() to manually create grids of a given size. fluidRow() allows a lot of customisation, but is more fiddly. In this tutorial, we will be using sidebarLayout(), which creates a large panel and a smaller inset side panel.

Creating Shiny apps – basic syntax

- Looking at the app and comparing to the panel layout examples in the above link, we can see that the app has a sidebarLayout with a sidebarPanel, mainPanel and titlePanel.
- titlePanel() indicates that we would like a separate panel at the top of the page in which we can put the title.
- sidebarLayout() indicates that we want our Shiny app to have the sidebar layout, one of many layouts we saw above. Within sidebarLayout we have:
- sidebarPanel() indicates that we want a sidebar panel included in our app. Sidebar panels often contain input widgets like sliders, text input boxes, radio buttons etc.
- mainPanel() indicates that we want a larger main panel. Main panels often contain the output of the app, whether it is a table, map, plot or something else.

```
# Packages ----
library(shiny)  # Required to run any Shiny app
library(ggplot2)  # For creating pretty plots
library(dplyr)  # For filtering and manipulating data
library(agridat)  # The package where the data comes from

# Loading data ----
Barley <- as.data.frame(beaven.barley)

# ui.R ----
ui <- fluidPage(
  titlePanel(""),  # Add a title panel
  sidebarLayout(  # Make the layout a sidebarLayout
    sidebarPanel(),  # Inside the sidebarLayout, add a sidebarPanel
    mainPanel()  # Inside the sidebarLayout, add a mainPanel
  )
)

# server.R ----
server <- function(input, output) {}

# Run the app ----
shinyApp(ui = ui, server = server)
```

Creating Shiny apps – basic syntax

The example app has four input widgets, a selectInput for genotype, a selectInput for histogram colour, a sliderInput for number of bins and a textInput to add some arbitrary text. Each of these widgets provides information on how to display the histogram and its accompanying table.

In the example app, all the widgets are found in the sidebarPanel so the code for these widgets should be put in the sidebarPanel command.
It uses a selectInput to choose the genotype of barley shown in the histogram and the table, another selectInput for the colour of the histogram, a sliderInput to choose the number of bins in the histogram and a textInput to display some text in the app. The histogram is located in the mainPanel along with a summary table of the data being shown, while the inputs are in the sidebarPanel.

```
ui <- fluidPage(
  titlePanel("Barley Yield"),
  sidebarLayout(
    sidebarPanel(
      selectInput(inputId = "gen",  # Give the input a name "genotype"
      label = "1. Select genotype",  # Give the input a label to be displayed in the app
      choices = c("A" = "a","B" = "b","C" = "c","D" = "d","E" = "e","F" = "f","G" = "g","H" = "h"), selected = "a"),  #
d. e.g. Display "A" and link to value "a"
    selectInput(inputId = "colour",
      label = "2. Select histogram colour",
      choices = c("blue","green","red","purple","grey"), selected = "grey"),
    sliderInput(inputId = "bin",
      label = "3. Select number of histogram bins",
      min=1, max=25, value= c(10)),
    textInput(inputId = "text",
      label = "4. Enter some text to be displayed", "")
    ),
  mainPanel()
  )
)
```

Note that `choices = c("A" = "a" ...` could be replaced with `choices = unique(Barley$gen)` to simply use the groups directly from the dataset.

More input widgets

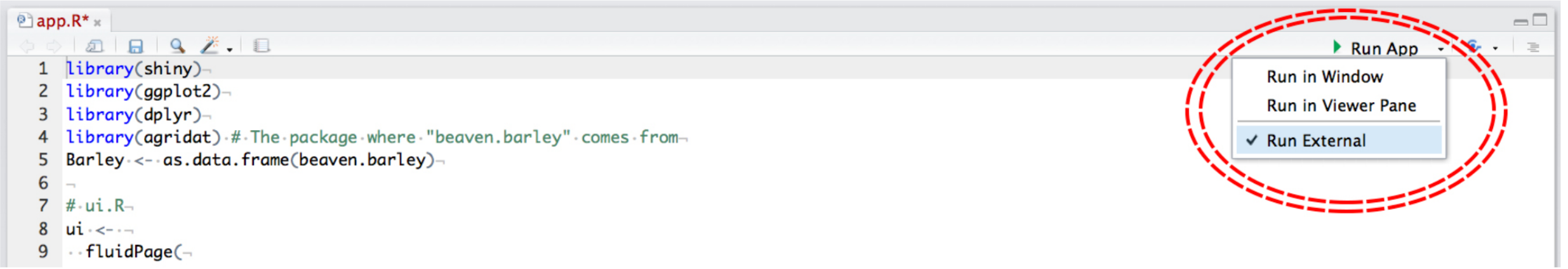
```
actionButton(inputId = "action", label = "Go!")
```

```
radioButtons(inputId = "radio", label = "Radio Buttons", choices = c("A", "B"))
```

```
selectInput(inputId = "select", label = "select", choices = c("A", "B"))
```

```
sliderInput(inputId = "slider", label = "slider", value = 5, min = 1, max = 100)
```

Running a Shiny apps



Outputs

A Shiny app without any outputs is useless. Outputs can be in the form of plots, tables, maps or text.

As per our example app, we're going to be using ggplot() to create a histogram. For more information on creating plots in ggplot2, see our tutorials on basic data visualisation and customising ggplot graphs.

Outputs are created by placing code in the curly brackets ({}) in the server object:

```
server <- function(input, output) {
  output$plot <- renderPlot(ggplot(Barley, aes(x = yield)) +
  geom_histogram(bins = 7,  # Add a histogram to the plot
    fill = "grey",  # Make the fill colour grey
    data = Barley,  # Use data from `Barley`
    colour = "black")  # Outline the bins in black
  )
}
```

- The histogram is great, but not particularly interactive. We need to link our input widgets to our output object.

- We want to select individual genotypes to display in our histogram, which the user can select using the selectInput that we called genotype earlier. Use some base R wizardry, [] $ and ==, to select the data we want. Update server with the new reactive output arguments so it looks like the code on the right.

```
server <- function(input, output) {
output$myhist <- renderPlot(ggplot(Barley, aes(x = yield)) +
        geom_histogram(bins = input$bin, fill = input$col, group=input$gen,
                data=Barley[Barley$gen == input$gen,],
                colour = "black"))

output$mytext <- renderText(input$text)

output$mytable <- renderTable(Barley %>%
filter(gen == input$gen) %>%
summarise("Mean" = mean(yield),
        "Median" = median(yield),
        "STDEV" = sd(yield),
        "Min" = min(yield),
        "Max" = max(yield)))
}
```

Displaying outputs

To make the outputs appear on your app in the `mainPanel`, **they need to be added to the `ui` object inside `mainPanel()` like so**:

```
ui <-
  fluidPage(
    titlePanel("Barley Yield"),
    sidebarLayout(
      position = "right",
      sidebarPanel(h3("Inputs for histogram"),
                   selectInput("gen", "1. Select genotype", choices = c("A" = "a","B" = "b","C" = "c","D"
ted = "a"),
                   br(),
                   selectInput("col", "2. Select histogram colour", choices = c("blue","green","red","pur
                   br(),
                   sliderInput("bin", "3. Select number of histogram bins", min=1, max=25, value= c(10)),
                   br(),
                   textInput("text", "4. Enter some text to be displayed", "")),
                   mainPanel(
                             plotOutput("myhist"),
                             tableOutput("mytable"),
                             textOutput("mytext")
                             )
      )
    )
```

Additional elements

HTML	Shiny	Function
`<div>`	`tags$div()`	Defines a block with consistent formatting
` `	`tags$br()`	Inserts a break
`<hr>`	`tags$hr()`	Inserts a horizontal line
`<p>`	`tags$p()`	Creates a paragraph of text
`<a>`	`tags$a(href = "LINK", "displayed text")`	Creates a clickable link

A list of all HTML tags can be found using:

```
shiny::tags
```

```
tags$div(style="color:red",
  tags$p("Visit us at:"),
  tags$a(href = "https://ourcodingclub.github.io", "Coding Club")
  )
```

This creates a block of text that is coloured red (`style="color:red"`), within that block t `"http://ourcodingglub.github.io", "Coding Club"`)).

Add the code above to your Shiny app in `mainPanel()` and see what happens!

Exporting finished apps

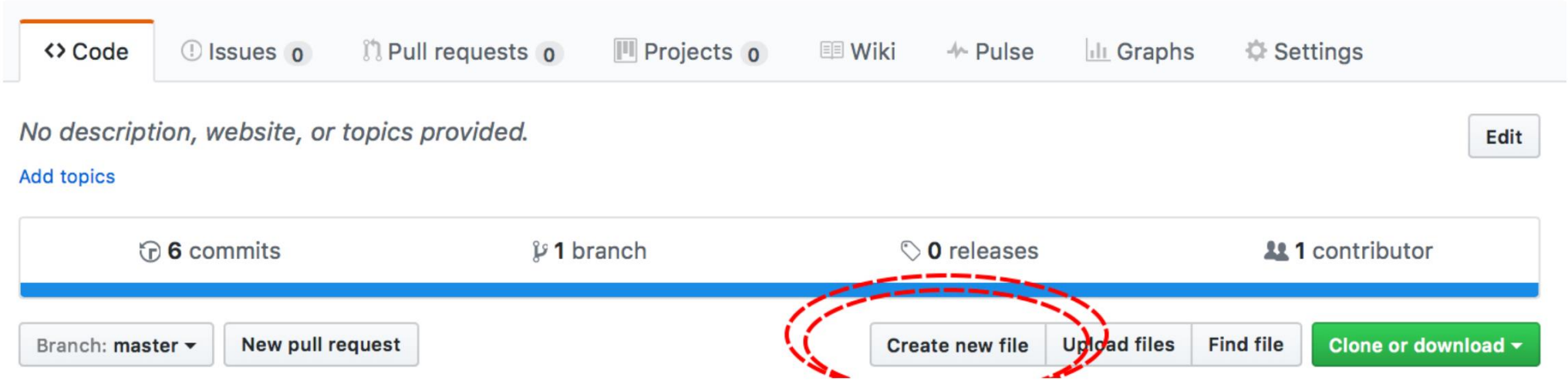
As a Github repository

It is easy to send a Shiny app to somebody else who also has RStudio. The easiest way is to send `app.R` alongside any data and other resources in a zip file to be unzipped by the recipient and run through R.

If you want to quickly share the app over the internet we recommend using Github to host the file.

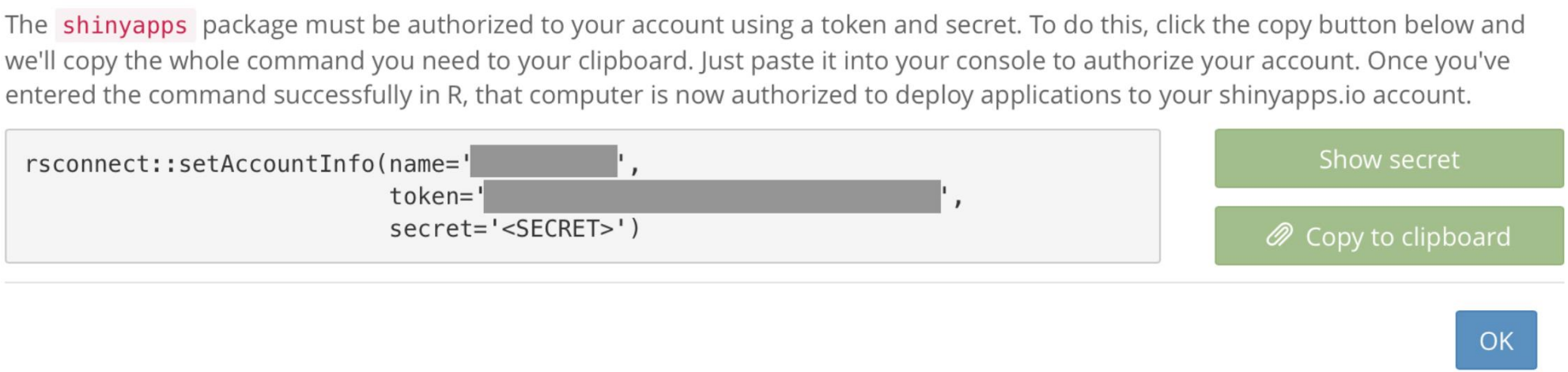
Go to Github, sign in with your account details, create a repository and upload everything from your app folder, including any `Data` and `www` folders.

Remember to add a file called `README.md` using `Create new file` in your new app repository, where you can write a quick explanation of the content of your app. `.md` files can use markdown syntax to create headers, sections, links etc.. See our tutorial on markdown and reproducible research for more markdown tips:



As a shinyapps.io app

You can also host Shiny apps on www.shinyapps.io, a webhosting platform run by RStudio that is especially built for Shiny apps. Go to their website and sign up using whatever method you choose, then go to www.shinyapps.io/admin/#/tokens, click *Show secret* and copy the `rsconnect` account info:



Then open up an R session and run the copied material to link `shinyapps.io` with R Studio.

To upload your app, open your `app.R` and click the publish button. Select a name for your app (**no spaces**) and click *Publish*.

The app can then be used by anyone with the URL for that app, which can be found by going to `shinyapps.io` and opening the app info from the dashboard: