Public Health Scotland

Introduction to R Shiny
Day 2

Public Health Scotland

Learning Outcomes – Day 2

- Multiscript dashboards: UI, Server and Global scripts
- Using multiple Public Health Scotland datasets within a dashboard
- Branded dashboards: Public Health Scotland colours and logos
- Modals, help buttons and data download buttons
- Plotly for interactive charts
- Deploying an app
- Why you should be using GitHub
- Good practice

Public Health Scotland

Refresh: dashboard components

- **`ui.R`** – controls what is displayed on the application page and how the components are laid out, for example navigation bars, text outputs, plot outputs, user input widgets.
- **`server.R`** – controls what happens behind the scenes, for example generation of plots and charts, and how user inputs from widgets affects these displays, minor data wrangling.
- **`global.R`** – where we load packages, functions, prepared data, define colour palettes.

Public Health Scotland

Building a PHS Shiny dashboard

- Information tab with background information and a pop-up text modal with future updates.
- Allergic conditions tab - text information, an interactive plotly chart which can be filtered using drop-down menus and a data multi-selection box.
- Asthma tab - data broken down into 6 separate charts based on sex and age with filtering options.
- Data tables tab - allergic conditions and asthma data can be downloaded as raw .csv files using a download button.

Building a PHS Shiny dashboard – global.R

```
############################################## LOAD PACKAGES
library(dplyr) #data manipulation
library(plotly) #charts
library(shiny)
library(shinyWidgets)
library(tidyr)
library(magrittr)
library(readr)

############################################
# LOAD DATA
data_allergy <- readRDS("allergy_scotland_chart_PRA.rds")
data_asthma <- readRDS("asthma_final.rds") %>%
mutate(rate = round(rate,1))

############################################
# CREATE OBJECTS USED IN OUTPUTS
condition_list <- sort(unique(data_allergy$type)) # allergies
diagnosis_list <- sort(unique(data_asthma$diagnosis)) # asthma
data_list_data_tab <- c("Allergies Data" = "data_allergy",
                        "Asthma Data" = "data_asthma")
```

We load packages and create objects for the two datasets we will be using in this Shiny app.

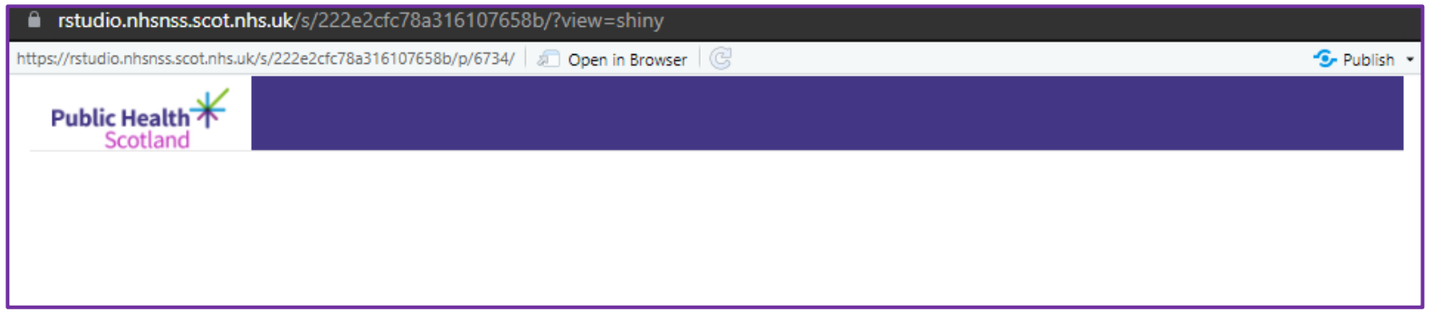
We also create an object for filtering between asthma and allergic conditions in tables and data downloads

Remember the objects we've created here – they'll all be important later.

Building a PHS Shiny dashboard – PHS Brand

This block of code, placed at the very beginning of our navbarPage() function allows us to generate a purple and white dashboard with a PHS logo which leads to our website.

```
navbarPage(title = div(tags$a(img(src="phs-logo.png", width=120,
                                  alt = "Public Health Scotland logo"), #bringing in PHS logo and look for dashboard
                              href= "https://www.publichealthscotland.scot/",
                              target = "_blank"),
                       style = "position: relative; top: -10px;"),
           header = tags$head(includeCSS("www/styles.css"), #CSS styles
                              HTML("<html lang='en'>"),
                              tags$link(rel="shortcut icon",
                                        href="favicon_phs.ico")),
```



Building a PHS Shiny dashboard – UI and Server components

We'll now begin building our empty app up into a functioning PHS dashboard containing multiple datasets displayed as charts and tables.

We'll be working on the UI and the server in conjunction with one another – remember whenever you want to include a chart or table output in your UI so that it's visible to the user, this output must also be created in the server, and under the same name. Therefore, it's good practice to be updating both of these scripts at the same time.

Building a PHS Shiny dashboard – Introduction tab

We've already opened our **`navbarPage()`**. This function can be used in place of **`fluidPage()`** as it also creates a resizable app based on users browser dimensions. We've included a chunk of code which sets the PHS theme and logo for our dashboard.

Now we can use **`tabPanel()`** to create out first tab.

Key point: simple text prints in UI don't require any server function.

Exercise: Building a PHS Shiny dashboard – Introduction tab

```
tabPanel(title = "Information", icon = icon("info-circle"),
         fluidRow(
            column(6,
              h2("Background Information"))
         ),
         fluidRow(
            column(12,
              h4(tags$b("Allergic Conditions")),
              p("Allergic conditions arise from an abnormal reaction of the immune system to a typically harmless
                environmental trigger. Allergic conditions have a wide variety of impacts on health, ranging from
                those that generally cause only minor symptoms, such as hay fever or conjunctivitis, to those that may be
                chronic and disabling, such as asthma, eczema or urticaria (hives)."),
              p("More text."),
```

Exercise:

- Add your tab, give it a name and an icon like we've done before.
- Add some text about allergic conditions and asthma.
- Give each block of text a heading, larger, bold, italics etc. (hints: **`h1()`**, **`tags$b()`**)
- try adjusting the column widths to see how this affects the appearance eg. **column(3,** … **column(6,** … **column(12,** … etc.

Building a PHS Shiny dashboard – Introduction tab

Now we're going to introduce a few new shiny functions to create an action button which users can click to open a pop-up information box.

These buttons and pop-up boxes are great for when you have excess information which may be important, but you don't want it to be present on the face of your dashboard at all times.

Now it's time to delve back into the server, as well as the UI.

Building a PHS Shiny dashboard – action buttons and modals.

```
tabPanel(title = "Information", icon = icon("info-circle"),
         fluidRow(
            column(6,
              h2("Background Information")),
            column(6,
              actionButton("new_next", tags$b("New content and future updates"),
                           icon = icon("calendar-alt")))
         ),
         fluidRow(
            column(12,
              h4(tags$b("Allergic Conditions")),
              p("Allergic conditions arise from an abnormal reaction of the immune system to a typically harmless
                environmental trigger. Allergic conditions have a wide variety of impacts on health, ranging from
```

`actionButton()` creates the actual button we see and click on within the tab. We require three arguments here: `inputId`, `label`, and `icon()`

Key point: Try running the app with this code added, but WITHOUT adding to the server – what happens?

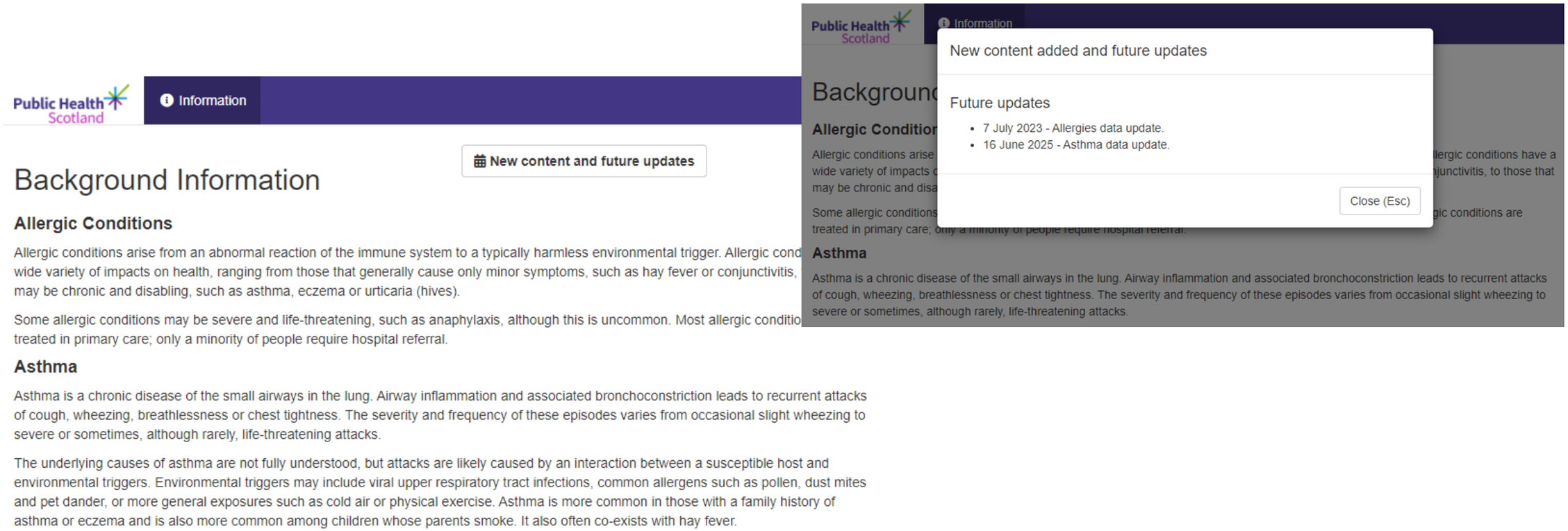
Building a PHS Shiny dashboard – action buttons and modals.

`observeEvent()` responds to “event-like” reactive inputs. We’re setting the input as “new_next” **– remember this matches what we’ve named it in the UI!**

```
function(input, output) {

observeEvent(input$new_next,
             showModal(modalDialog(
               title = "New content added and future updates",
               h4("Future updates"),
               tags$ul(
                 tags$li("7 July 2023 - Allergies data update."),
                 tags$li("16 June 2025 - Asthma data update.")),
               size = "m",
               easyClose = TRUE, fade=FALSE,footer = modalButton("Close (Esc)"))))
}
```

We introduce **`showModal()`** and **`modalDialog()`** and the arguments within these.

Building a PHS Shiny dashboard – Introduction tab



Building a PHS Shiny dashboard – Allergic conditions data

Go to the **`global.R`** script. Run and open the **`data_allergy`** object which contains our allergy data.

Our aim is to create an interactive plotly chart which responds to user inputs:

- A drop down menu to select either rate or crude number.
- A selection box which allows us to put data for up to 4 allergic conditions on the line chart at once.

We can also include some headings and text above the chart for extra information.

Exercise: Building a PHS Shiny dashboard – Allergic conditions data UI

Begin with the **`ui.R`**. We need to use **`tabPanel()`** again underneath the first, to create another new tab.

- Give your new tab a title and icon.
- Add a heading and some information text within the tab.

```
tabPanel(title = "All Allergic Conditions", icon = icon("hospital"),
         h2("Hospital admissions for different allergic conditions"),
         p("Since most people with allergic conditions are managed in primary care, secondary care (hospital) data
           relate to a smaller group of people, generally with more severe conditions."),
         p("The chart shows data from the Scottish Morbidity Record (SMR01) scheme, which records hospital inpatient
           and day case activity for Scotland."),
         p("The chart shows that rates of hospital admissions per 100,000 of the population have remained relatively
           constant for all allergies across the past 10 years in Scotland."),
```

Add any text you like.

Exercise: Building a PHS Shiny dashboard – Allergic conditions data UI

Note that in this case we've used "`conditions_list`" – defined in the `global.R` script!

```
p("The chart shows that rates of hospital admissions per 100,000 of the population have remained
  relatively constant for all allergies across the past 10 years in Scotland."),

fluidRow(
     column(3,
          selectInput("measure", label = "Select numbers or rates",
               choices = c("Number", "Rate"), selected = "Rate")
           ),
     column(3,
          selectizeInput("conditions", label = "Select one or more allergic conditions (up to four)",
               choices = condition_list, multiple = TRUE, selected = "All allergies",
               options = list(maxItems =4L))
           )
        ),
fluidRow(
     column(3,
          plotlyOutput("chart", width = "100%", height = "350px"))
           )
        )
```

Key point: note the `inputId` used! eg. "`measure`", "`conditions`", "`chart`". We need to match these in the server side!

Exercise: Building a PHS Shiny dashboard – Allergic conditions data server

Let's move to the server side for the allergic conditions tab.

Hopefully you have some prior knowledge of plotly or have read up on it in advance. It works in a similar way to ggplot2 but of course, functions and parameters differ in various ways.

The only thing we need to add to our server side is code for our plot output.

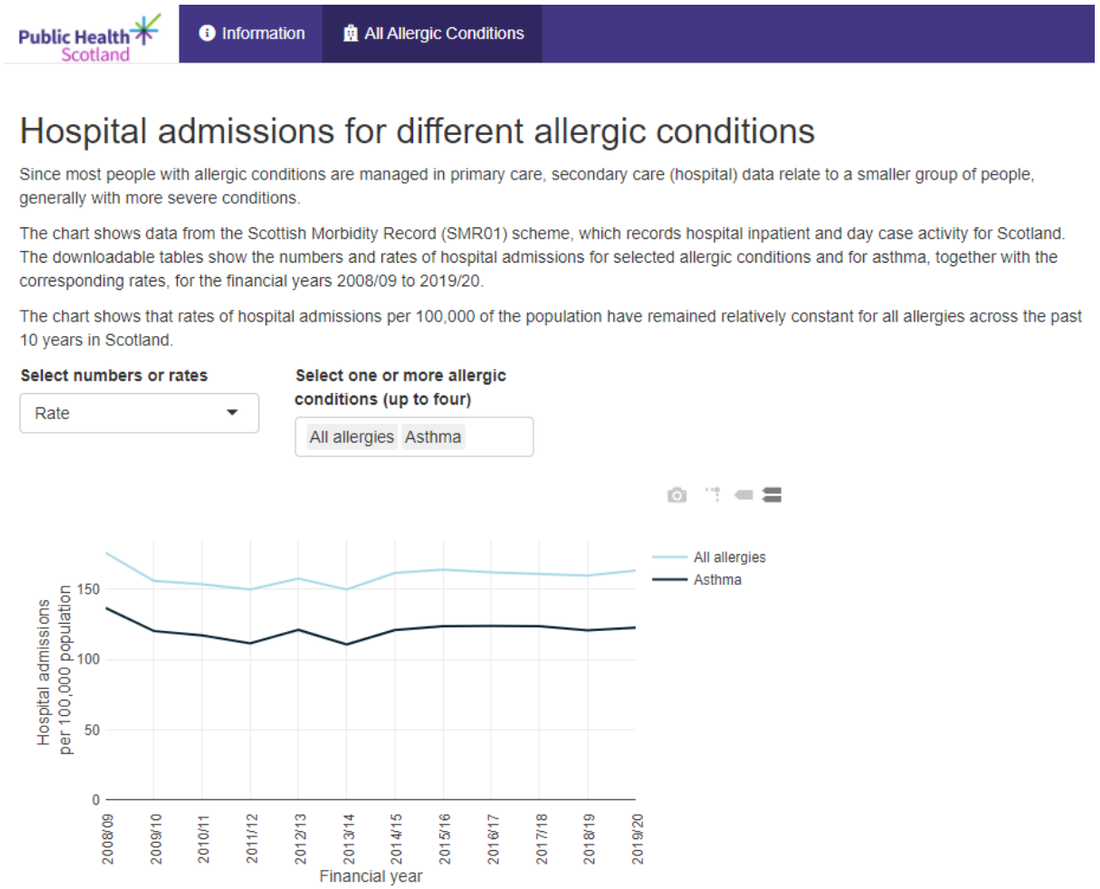
I've given you some pre-prepared plotly code (currently commented out on the **`server.R`** script, uncomment it for the following steps).

Exercise: Building a PHS Shiny dashboard – Allergic conditions data server

We create **`output$chart`** using **`renderPlotly({…})`**

```
output$chart <- renderPlotly({

#Data for condition
     data_condition <- data_allergy %>% subset(type %in% input$conditions & measure==input$measure)

     yaxistitle <- case_when(input$measure == "Number" ~ "Number of hospital admissions",
                             input$measure == "Rate" ~ "Hospital admissions <br>per 100,000 population")

     plot <- plot_ly(data=data_condition, x=~year, y = ~value, color = ~type,
                     colors = c('#abd9e9', '#74add1', '#4575b4', '#313695', '#022031'),
                     type = "scatter", mode = 'lines',
                     width = 650, height = 350) %>%

#Layout
     layout(annotations = list(),
          yaxis = list(title = yaxistitle, rangemode="tozero", fixedrange=TRUE),
          xaxis = list(title = "Financial year",  fixedrange=TRUE, tickangle = 270),
          font = list(family = 'Arial, sans-serif'),
          margin = list(pad = 4, t = 50, r = 30), hovermode = 'false') %>%
     config(displayModeBar= T, displaylogo = F)
})
```

Building a PHS Shiny dashboard – Allergic conditions tab



Now we have a new tab containing some text information, and a chart showing allergic conditions data.

The chart responds to the user selecting either rate or number, and the user may also add up to 4 data lines on the chart at once using the second selection menu.

Building a PHS Shiny dashboard – Asthma exploration tab

We're now going to run and take a look at the **`data_asthma`** object where we read in our raw asthma data. Get a feel for what the data looks like and how we might be using it.

Our aim is to create another new tab displaying the asthma data:

- A header and some text for above the charts
- A drop-down menu for selecting either rate or numerator
- A selection box allowing us to select up to 4 asthma-related diagnosis codes
- 6 different charts: hospitalisations for all males, all females, males under 10 years, females under 10 years, males over 10 years, females over 10 years.

Exercise: Building a PHS Shiny dashboard – Asthma exploration tab UI

Using your knowledge so far to create another new tab:

- Insert another **`tabPanel()`** under the first two, giving it a title and an icon.
- Write a header and some text about asthma (or other gibberish).
- Use **`selectInput()`** to create a drop-down to select rate or numerator.
- Use **`selectizeInput()`** to create a multiple-selection box where the user can choose up to four diagnoses to visualise on the chart.

Key point: REMEMBER your `inputId` for user inputs (drop-down selections) can't be the same as what you used for the allergies tab!

Exercise: Building a PHS Shiny dashboard – Asthma exploration tab UI

Note: for the "**`choices =`**" argument, we've used "**`diagnosis_list`**". This was defined in the **`global.R`** script.

```
tabPanel(title = "Asthma Breakdown", icon = icon("area-chart"),
         h2("Asthma Exploration Work"),
         p("The selection of charts below show how hospital admissions for asthma related
            diagnosis codes have changed over the past 10 years across different age groups."),
         p("The data is split by sex (all males and all females) and by the following age
            groups: under 10 years old and over 10 years old."),

         fluidRow(
              column(3,
                   selectInput("measure_asthma", label = "Select numerator or rates",
                               choices = c("Numerator", "Rate"), selected = "Rate")),
              column(3,
                   selectizeInput("diagnosis", label = "Select one or more diagnosis",
                               choices = diagnosis_list, multiple = TRUE,
                               selected = "Status asthmaticus (J46) first position"),
                   options = list(maxItems =4L))
                 )
         )
```

Exercise: Building a PHS Shiny dashboard – Asthma exploration tab UI

We also need to add our **`plotlyOutput()`** functions for chart to the UI.

```
            column(3,
                selectizeInput("diagnosis", label = "Select one or more diagnosis",
                               choices = diagnosis_list, multiple = TRUE,
                               selected = "Status asthmaticus (J46) first position"),
                options = list(maxItems =4L))
              ),

        fluidRow(
            column(6,
                plotlyOutput("male_all", width = "100%")),
            column(6,
                plotlyOutput("female_all", width = "100%")),
            column(6,
                plotlyOutput("male_under10", width = "100%")),
            column(6,
                plotlyOutput("female_under10", width = "100%")),
            column(6,
                plotlyOutput("male_over10", width = "100%")),
            column(6,
                plotlyOutput("female_over10", width = "100%"))
              )
        )
```

Building a PHS Shiny dashboard – Asthma exploration tab server

It’s time to update our server side to reflect what we’ve just added to the UI.

You’ll remember the large chunk of plotly code we just used to generate our allergies tab chart.

I’ve created a very similar chunk of code for generating the asthma charts to what we used for the allergy one. But this time, I’ve created a function from it so that it can be used for each of the 6 asthma charts.

Building a PHS Shiny dashboard – Asthma exploration tab server

```
plot_charts <- function(sex_chosen, age_grp_chosen) {

     data_plot <- data_asthma %>% subset(diagnosis %in% input$diagnosis &
                                                 sex == sex_chosen &
                                                 age_grp == age_grp_chosen)
     yaxistitle <- case_when(input$measure_asthma == "Numerator" ~ "Number of hospital admissions",
                             input$measure_asthma == "Rate" ~ "Hospital admissions <br>per 100,000 population")

     plot <- plot_ly(data=data_plot, x=~year, y = ~get(tolower(input$measure_asthma)), color = ~diagnosis,
                     colors = c('#abd9e9', '#74add1', '#4575b4', '#313695', '#022031'),
                     type = "scatter", mode = 'lines') %>%

     layout(annotations = list(),
          yaxis = list(title = yaxistitle, rangemode="tozero", fixedrange=TRUE),
          xaxis = list(title = "Financial year",  fixedrange=TRUE, tickangle = 270),
          font = list(family = 'Arial, sans-serif'),
          margin = list(pad = 4, t = 50, r = 30),
          hovermode = 'false') %>%
          config(displayModeBar= T, displaylogo = F)
}
```

Building a PHS Shiny dashboard – Asthma exploration tab server

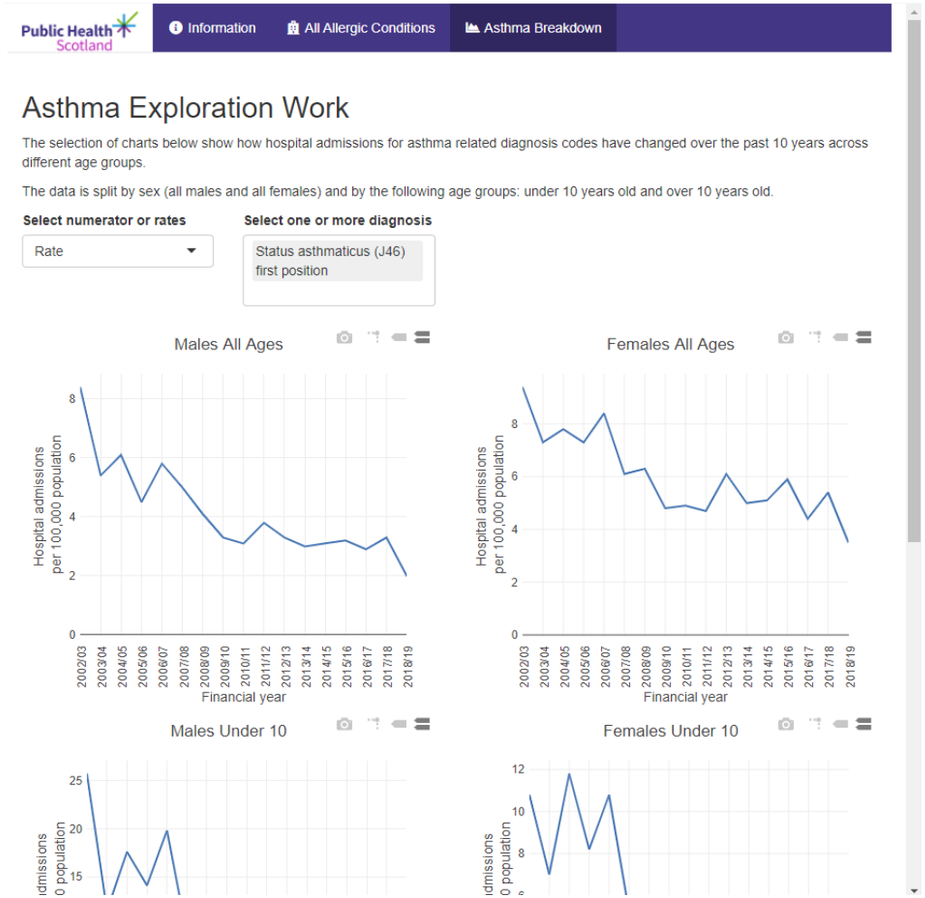
Now that we've created a function to generate the charts, we need to run it 6 times for our 6 different data splits.

```
output$male_all <- renderPlotly({ plot_charts(sex_chosen = "Male", age_grp_chosen = "All") %>%
                                    layout(title = "Males All Ages")})
output$female_all <- renderPlotly({ plot_charts(sex_chosen = "Female", age_grp_chosen = "All") %>%
                                    layout(title = "Females All Ages")})
output$male_under10 <- renderPlotly({ plot_charts(sex_chosen = "Male", age_grp_chosen = "Under 10") %>%
                                    layout(title = "Males Under 10")})
output$female_under10 <- renderPlotly({ plot_charts(sex_chosen = "Female", age_grp_chosen = "Under 10") %>%
                                    layout(title = "Females Under 10")})
output$male_over10 <- renderPlotly({ plot_charts(sex_chosen = "Male", age_grp_chosen = "Over 10") %>%
                                    layout(title = "Males Over 10")})
output$female_over10 <- renderPlotly({ plot_charts(sex_chosen = "Female", age_grp_chosen = "Over 10") %>%
                                    layout(title = "Females Over 10")})
```

Our output Ids (eg. **`output$male_all`**) match those we used in the UI code.

We use the **`renderPlotly({})`** function as before, but within each we use the **`plot_charts()`** function we've created.

Building a PHS Shiny dashboard – Asthma exploration tab



Our third tab is finished!

We've added a new tab with 6 different charts on asthma data which are reactive to user input menus.

Building a PHS Shiny dashboard – Data downloads

This will be the final tab of our dashboard.

In this section, we will display the data in table format using the **`DT`** package – we went over this in the first session.

We will include a drop-down menu which allows the user to display either the allergic conditions data or the asthma data in the table.

We will also include a download button which when clicked, will download the tabulated data in a **`.csv`** file.

Exercise: Building a PHS Shiny dashboard – Data downloads UI

Add another **`tabPanel()`** in the **`ui.R`** below the current ones.

Give your tab a name and an icon. Add a header and some text introducing the data downloads tab.

Add a **`selectInput()`** drop-down menu that allows the user to select either allergic conditions data or asthma data.

REMEMBER: an object has been created in the `global.R` script which can be used for filtering between allergic conditions and asthma data.

Exercise: Building a PHS Shiny dashboard – Data downloads UI

We're introducing the **`downloadButton()`** function. This also requires an **`inputId`** which will be matched in the server side, and we've labelled it **"`Download data`"**.

The final line of code is one you've seen before in our last session. It displays the table on the app in the **`DT`** package format and we've used the Id **"`table_filtered`"**.

```
tabPanel(title = "Data Table Downloads", icon = icon("table"),
          h2("Select the data you wish to download"),
          p("This section allows you to view the data in table format.
            You can use the filters to select the data you are interested in.
            You can also download the data as a csv using the download button."),

         fluidRow(
           column(6, selectInput("data_select", "Select the data you want to explore.",
                         choices = data_list_data_tab)),
           column(6, downloadButton('download_table_csv', 'Download data')),

               DT::dataTableOutput("table_filtered")
                )
           )
) #END OF FLUIDPAGE (USER INTERFACE)
```

Exercise: Building a PHS Shiny dashboard – Data downloads server

First, we create a reactive dataset, as the data displayed depends on user input from the drop-down menu.

The **`switch()`** function allows the table to switch between allergic conditions and asthma data.

```
# CREATE THE DATA TABLE

data_table <- reactive({
     # Change dataset depending on what user selected
     table_data <- switch(input$data_select,
                          "data_allergy" = data_allergy, "data_asthma" = data_asthma)

          if (input$data_select == "data_allergy") {
               table_data %<>%
                  select(year, type, measure, value)
          } else if (input$data_select == "data_asthma") {
               table_data %<>%
                  select(diagnosis, year, sex, age_grp, numerator, rate)
          }
})
```

Exercise: Building a PHS Shiny dashboard – Data downloads server

The next chunk of code actually renders the data table using **`DT::renderDataTable({…})`.**

We use the Id **“`table_filtered`”** to match what we’ve used in the UI.

```
# RENDER THE DATA TABLE

output$table_filtered <- DT::renderDataTable({

     table_colnames  <-  gsub("_", " ", colnames(data_table()))

     DT::datatable(data_table(), style = 'bootstrap',
                   class = 'table-bordered table-condensed',
                   rownames = FALSE,
                   options = list(pageLength = 20,
                                  dom = 'tip',
                                  autoWidth = TRUE),
                   filter = "top",
                   colnames = table_colnames)
})
```

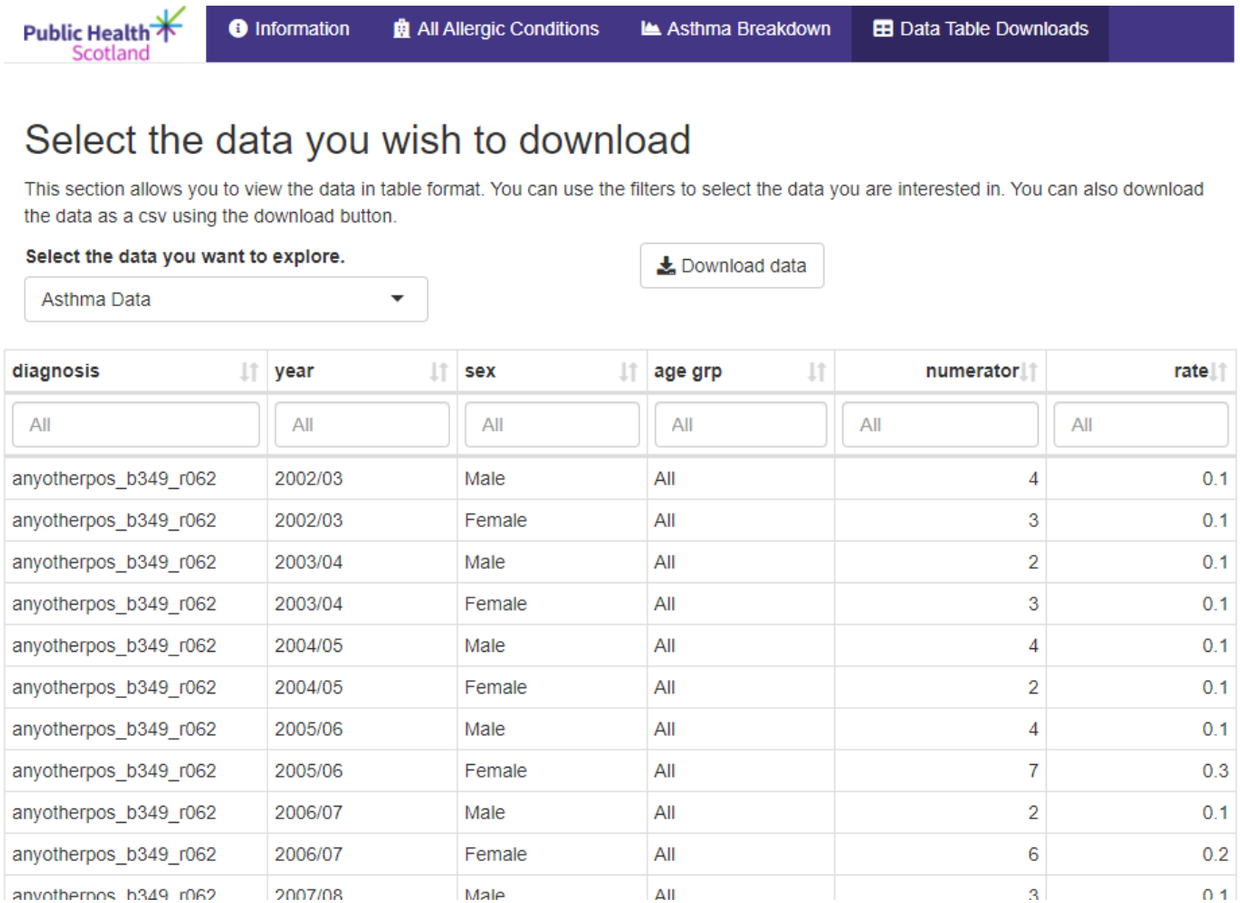
Exercise: Building a PHS Shiny dashboard – Data downloads server

Finally, we insert some code that allows the download button to be functional.

We use the Id "**download_table_csv**" as we defined this in the UI. The **downloadHandler()** function defines the file name and the contents.

```
# ENABLE THE DATA TO BE DOWNLOADED

output$download_table_csv <- downloadHandler(
     filename = function() {
          paste(input$data_select, ".csv", sep = "")
     },
     content = function(file) {
          Write_csv(data_table()[input[["table_filtered_rows_all"]], ], file)
     })
} # END OF SERVER
```

Building a PHS Shiny dashboard – Data downloads



diagnosis	year	sex	age grp	numerator	rate
All	All	All	All	All	All
anyotherpos_b349_r062	2002/03	Male	All	4	0.1
anyotherpos_b349_r062	2002/03	Female	All	3	0.1
anyotherpos_b349_r062	2003/04	Male	All	2	0.1
anyotherpos_b349_r062	2003/04	Female	All	3	0.1
anyotherpos_b349_r062	2004/05	Male	All	4	0.1
anyotherpos_b349_r062	2004/05	Female	All	2	0.1
anyotherpos_b349_r062	2005/06	Male	All	4	0.1
anyotherpos_b349_r062	2005/06	Female	All	7	0.3
anyotherpos_b349_r062	2006/07	Male	All	2	0.1
anyotherpos_b349_r062	2006/07	Female	All	6	0.2
anyotherpos_b349_r062	2007/08	Male	All	3	0.1

Now we have a data tables tab complete with the ability to select, filter and download data as a `.csv`!

(and a complete dashboard! Well done!)

Public Health Scotland

Deploying an App

We've built an entire Shiny dashboard today, but these are only available for us to view locally.

When tasked with creating a public facing dashboard you will, at some point, need to deploy it to the Shiny server.

We're going to go over this briefly, just so that you are aware of the process.

Deploying an App

You should first run the app from the **`global.R`** script and check that everything is functioning normally.

On a separate script, we use the **`rsconnect`** package to deploy the app.

```
# DEPLOYING AN APP
library(rsconnect)

# connecting to the PHS shiny.io account
# enter the token and secret password provided by one of the account managers
rsconnect::setAccountInfo(name='scotland',
                          token='CD[redacted]3F3',
                          secret='+Rx[redacted]sUP')

# this chunk of code deploys the app located in your work area, under a name you designate
rsconnect::deployApp('/folder_1/folder_2/folder_3/folder_4/folder_5/shiny_app',
                     appName="our-shiny-training-app")
```

The first chunk of code sets the account information for the PHS shiny.io account.
The second chunk of code locates the app in your working directory and deploys it to the link designated, eg. https://scotland.shinyapps.io/**our-shiny-training-app**/

Shiny dashboards and GitHub

Shiny dashboards -> thousands of lines of code.
Shiny dashboards -> updated monthly, weekly or even daily.

At this point, version control becomes very important.

GitHub -> code tracking, sharing and collaboration.
GitHub -> Work on different code branches so the overall master code is not affected, work is not overwritten.
GitHub -> revert to previous code when you may have saved, pushed or even published mistakes.

Public Health Scotland

Good Practice

- Split large apps into 3 scripts: **`global.R`**, **`server.R`** and **`ui.R`** as opposed to using **`app.R`** where everything is kept on one.
- Create supporting scripts and source them into the server, eg. a **`functions.R`** script that contains all of your pre-written complex functions for creating plots and tables from datasets.
- Reduce the number of packages used, keep to the essentials.
- Keep as much as possible in the UI, having a nice simple server.
- Prepare your data in advance, know what you want to show on the app and how, then it's simple when it comes to setting plot and table parameters, drop down menus etc.

Public Health Scotland

End of day 2 – any questions?