Introduction to R Shiny
Day 2

Posit Implementation Programme

11/09/2023

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 and help buttons
- ggplotly for interactive charts
- Data downloads
- Why you should be using GitHub
- Good practice

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 affect these displays, minor data wrangling.
- Previously we created a basic single script dashboard. We're going to look at advanced dashboard building where the code is split across multiple scripts.

Global

- The Global script is generally used for loading packages, functions, and prepared data files as named objects.

- Can be used to define other things such as colour palettes and plot parameters.

- **Key point:** adding this script helps to keep your code clean and tidy. When running the app, all packages, functions and data will be read in from the Global script, while the app itself is created from the UI and Server.

Building a PHS Shiny dashboard

- I’ve given you some pre-prepared scripts. Here’s what we’re going to include in our dashboard:
  - An information tab: background information and a pop-up text modal.
  - A tab on allergic conditions data: text information, an interactive chart, drop-down menus and a data multi-selection box.
  - A tab on asthma: information, 6 separate charts based on sex and age, also containing filtering options.
  - A data tables tab: allergic conditions and asthma data can be downloaded as raw .csv files using a download button.

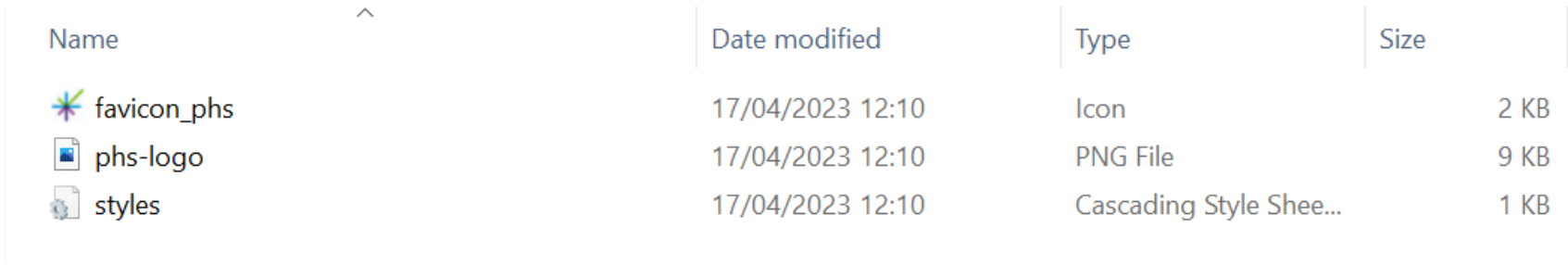
Building a PHS Shiny dashboard – global.R

- We load packages and create objects for the two datasets we will be using in this Shiny app.
- We'll be creating drop-down filters for allergic conditions and asthma diagnosis codes, as well as selected data for download. It's good to create these lists/objects in the global script as well.

The objects created here will be important later.

```
# GLOBAL SCRIPT
# SHINY TRAINING DAY 2

# LOAD PACKAGES ----

library(dplyr) #data manipulation
library(plotly) #charts
library(ggplot2)
library(shiny)
library(shinyWidgets)
library(tidyr)
library(magrittr)
library(readr)

# LOAD DATA ----

data_allergy <- readRDS("data/allergy_scotland_chart_PRA.rds")

data_asthma <- readRDS("data/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")
```

“www” Folder



Name	Date modified	Type	Size
favicon_phs	17/04/2023 12:10	Icon	2 KB
phs-logo	17/04/2023 12:10	PNG File	9 KB
styles	17/04/2023 12:10	Cascading Style Shee...	1 KB

- Locally store the elements that will be rendered in the web browser and are not the output of the scripts.
- For example, save any images, and html files in this folder

Building a PHS Shiny dashboard – PHS Brand

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

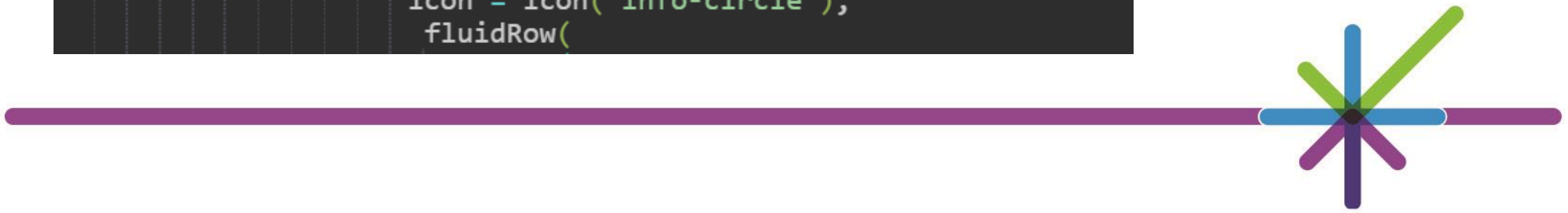
- This block of code, placed at the very beginning of our **navbarPage()** function allows us to generate a purple and white dashboard with a PHS logo which leads to our website.
- This can be copy and pasted into any future dashboards you make.

Building a PHS Shiny dashboard – Information tab

We've already opened our **navbarPage()**. This can be used in place of **fluidPage()** as it also creates a resizable app based on the users browser dimensions, although if you were to include both nothing would go wrong. Within this function, 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! We've used these functions before when looking at layouts, think back to using **fluidRow()** and **column()** for our grid-style layout.

```
# INFORMATION TAB ----
            tabPanel(title = "Information",
                     icon = icon("info-circle"),
                      fluidRow(
```



Exercise: Building a PHS Shiny dashboard – Information tab

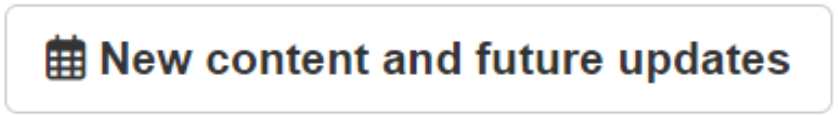
- Add your tab, give it a name and an icon. (https://fontawesome.com/v5.15/icons?d=gallery&p=2)
- Add some text about allergic conditions and asthma.
- Give each block of text a heading, try making it larger, try bolding it. (hints: **h1()**, **tags$b()**, etc…)
- Make some of the text italic.
- If you use **fluidRow()** and **column()**, try adjusting the column widths to see how this affects the appearance eg. column(3, … column(6, … column(12, … etc.

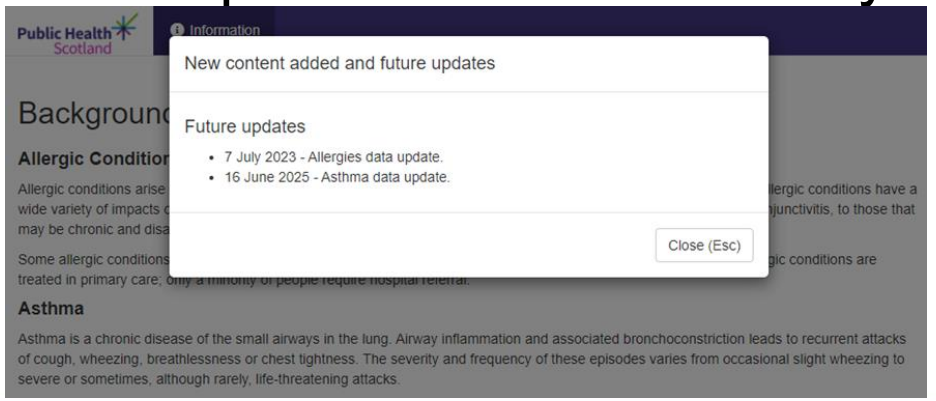
Exercise: Building a PHS Shiny dashboard – Information tab

```
tabPanel(title = "Information",
         icon = icon("info-circle"),
          fluidRow(
            column(6,
                 h2("Background Information")),
          ),
          fluidRow(
            column(12,
              h4(tags$b("Allergic Conditions")),
              p("Allergic conditions arise from an abnormal reaction of the immune system to a
                typically harmless environmental trigger. Allergic conditions have a wide variety
                of impacts on health, ranging from those that generally cause only minor symptoms,
               such as hay fever or conjunctivitis, to those that may be chronic and disabling,
               such as asthma, eczema or urticaria (hives)."),
              p("Some allergic conditions may be severe and life-threatening, such as anaphylaxis,
                 although this is uncommon. Most allergic conditions are treated in primary care;
                 only a minority of people require hospital referral."),
              h4(tags$b("Asthma")),
              p("Asthma is a chronic disease of the small airways in the lung. Airway inflammation and
                associated bronchoconstriction leads to recurrent attacks of cough, wheezing,
                breathlessness or chest tightness. The severity and frequency of these episodes varies from
```

Building a PHS Shiny dashboard – Information tab

- It’s a great idea to add an information box letting users know when updates are due.
- We’re going to do this by introducing an action button which users can click to open a pop-up information box.
- These 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.





Building a PHS Shiny dashboard – action buttons and modals UI

```
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')))
```

- Use **column()** to set where the action button will appear (you may need to play with column widths in real situations). **Reminder: maximum column width is 12.**
- **actionButton()** creates the button we see and click on within the tab. We require three arguments:
  - "**new_next**" is the name we're giving for the **inputId** ,
  - "**New content and future updates**" is the **label** we're putting on the button. The user will see this.
  - 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 Server

```
observeEvent(input$new_next,
             showModal(modalDialog( # creats a modal: a pop-up box that contains text information
               title = "New content added and future updates",
               h4("Future updates"),
               tags$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)")))) # creates the esc button for closing the popup box
```

- We introduce another reactive element called **observeEvent()**.
  - This responds to “event-like” reactive inputs, such as the user clicking an action button.
- Set the input as “**new_next**”
  - **Remember** this matches what we’ve named it in the UI!
- **showModal()** tells the modal to appear when the action button is clicked.
  - We can set the size, enable easyClose and add the ability to close using the escape button.
- **modalDialog()** allows us to add text to the modal, in this case, a heading and bullet points highlighting key dates

Building a PHS Shiny dashboard – Allergic conditions

- I've given you the data file and prepared **global.R** script.
- Run and open the **data_allergy** object which contains our allergy data.
  - Have a look at the data – column names, variable names etc. to get a feel for where some of the labels and inputs may be coming from.
- Our aim is to create an interactive 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 UI

- Use **tabPanel()** again underneath the first to create another new tab within our navigation bar, this will appear next to our pre-existing Information tab.

- Give your new tab a title and icon.

- Add a heading and some 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."),
```

Exercise: Building a PHS Shiny dashboard – Allergic conditions UI

- Use **fluidRow()** to put two user input menus above the chart.
  - **Hint:** If you have time, play around with column widths.
- **selectizeInput()** works in much the same way as **selectInput()** but allows us to select multiple inputs for the chart – in this case, a maximum of four.
  - **Note:** "conditions_list" for our choices – this was defined in the global.R script!
- We're creating interactive charts and will be using **plotlyOutput()** instead of **plotOutput().**

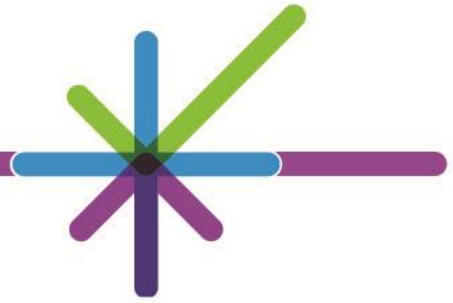
```
fluidRow(
  column(3,
         selectInput(inputId = "measure",
                     label = "Select numbers or rates",
                     choices = c("Number", "Rate"), selected = "Rate")
  ),
  column(3,
         selectizeInput(inputId = "conditions",
                        label = "Select one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

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

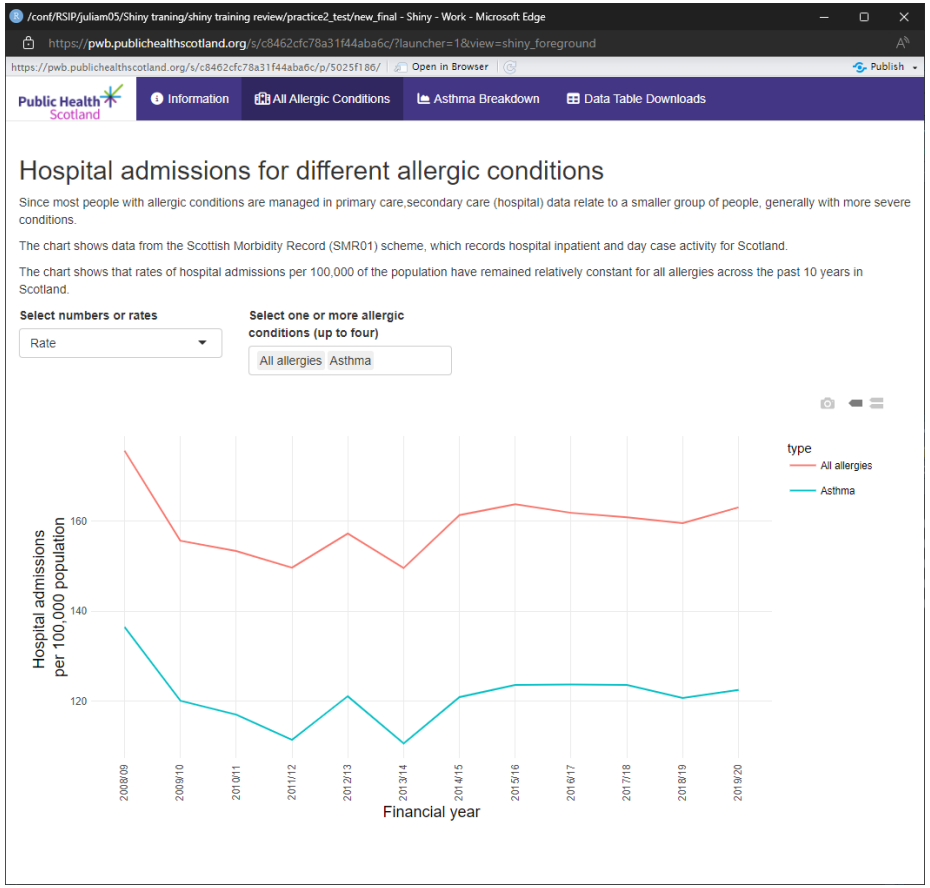
Exercise: Building a PHS Shiny dashboard – Allergic conditions Server

- Create **output$chart** using **renderPlotly({…})**
  - **Remember:** the inputId was “chart” in the UI.
- Subset data.
  - The “type” column of our data should reflect the condition the user inputs: “**type %in% input$conditions**” and the measure should be what the user inputs as measure: “**measure==input$measure**”.
- Make ggplot into plotly object
  - Define the y-axis title to change based on whatever the user inputs.

```
output$chart <- renderPlotly({

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

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

  plot <-
    ggplot(data_condition, aes(x = year, y = value, colour = type)) +
    geom_line(aes(group=1)) +
    theme_minimal()

  ggplotly(height = 500,
           width = 1000) %>%
    layout(annotations = list(), #It needs this because of a buggy behaviour
           yaxis = list(title = yaxistitle, rangemode="tozero", fixedrange=TRUE),
           xaxis = list(title = "Financial year",  fixedrange=TRUE, tickangle = 270),
           font = list(family = 'Arial, sans-serif'), # font
           margin = list(pad = 4, t = 50, r = 30), # margin-paddings
           hovermode = 'false') %>% # to get hover compare mode as default
    config(displayModeBar= T, displaylogo = F) # taking out plotly logo and collaborate button
}
)
```



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

- Run and 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 UI

- Use your knowledge so far 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.
  - Use **selectInput()** to create a drop-down allowing the user 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 that your inputId for user inputs (drop-down selections) can't be the same as what you used for the allergies tab!

If you feel confident, try adding the plotlyOutput() lines of code to the UI for each of the 6 charts.

Exercise: Building a PHS Shiny dashboard – Asthma exploration UI

- With multiple, we have more **plotlyOutput()** functions in use than our allergies tab.
- **Reminder:** your inputId for each user selection or plotlyOutput CANNOT be the same as what we used for the allergies tab – otherwise your charts will try to respond to user inputs on the allergies tab instead!

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

         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%"))
```

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

Building a PHS Shiny dashboard – Asthma exploration Server

- You'll remember the large chunk of code we just used to generate our allergies tab chart.
- Hopefully, your thoughts are **"I don't want to copy and paste this 6 times for 6 asthma charts."** Thanks to functions, we don't have to.
- Using a function with similar code to the allergies tab, we can recreate one chart six times.

Building a PHS Shiny dashboard – Asthma exploration Server

- Create a function called plot_charts with two inputs; **sex_chosen** (from sex in the data) and **age_grp_chosen** (from age_grp in the data)

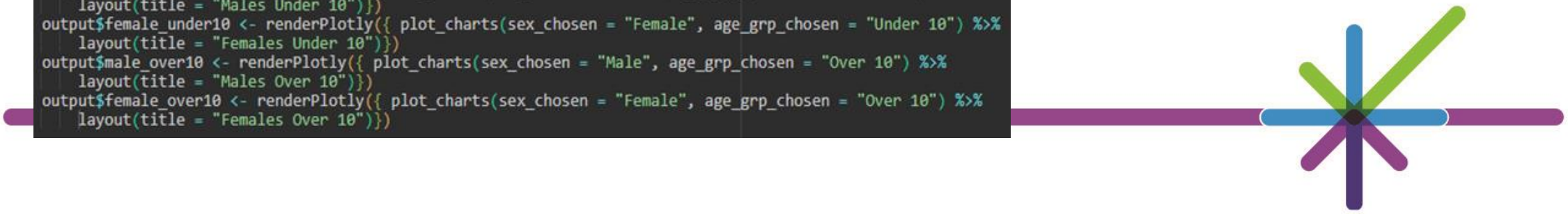
- Create data_plot to use in charts with filter and function inputs
- Create dynamic axis titles
- Create plot in ggplot
  - inputId “diagnosis” and “measure_asthma” in the UI, and so must match to this in the server.
- Create ggplotly object

```
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)

  #y axis title
  yaxistitle <- case_when(input$measure_asthma == "Numerator" ~ "Number of hospital admissions",
                          input$measure_asthma == "Rate" ~ "Hospital admissions <br>per 100,000 population")

  plot <-
    ggplot(data_plot, aes(x = year, y = get(tolower(input$measure_asthma)), colour = diagnosis)) +
    geom_line(aes(group=1)) +
    theme_minimal()

  ggplotly() %>%
    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 Server

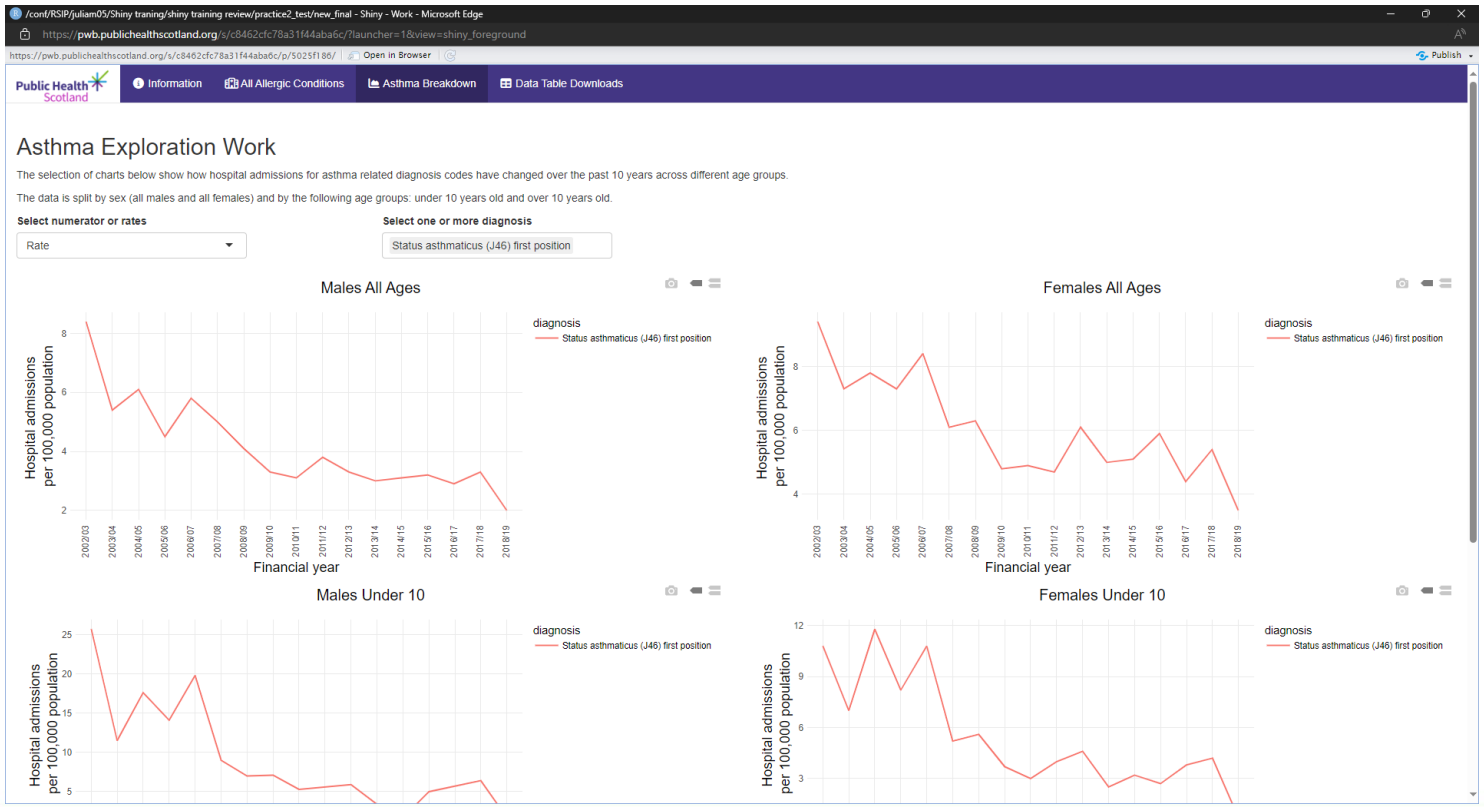
- Create six charts using the function you just wrote; male_all, female_all, male_under10, female_under10, male_over10 and female_over10.
  - **NOTE:** Our output Ids (eg. **output$male_all**) should match those we used in the UI code.
- Use the **renderPlotly({})** function as before, but within each use the **plot_charts()** function you created.
- Define **sex_chosen** and **age_grp_chosen** for each split as required arguments for the **plot_charts()** function.
- Pipe and use **layout()** to add a title to each chart at this point.

```
# Here, the plot_charts function created above is put into use
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")})
```



Building a PHS Shiny dashboard – Asthma exploration tab

- Final Asthma Tab with reactive plots



Building a PHS Shiny dashboard – Data downloads

- Final tab of the dashboard!
- Display the data in table format using the **DT** package that we went through in the first session.
- Include drop-down menu which allows the user to display either the allergic conditions data or the asthma data in the table.
- 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()** below the current ones (you should be used to doing this by now!)
- 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.

- **Then we'll introduce a data downloads button.**

Exercise: Building a PHS Shiny dashboard – Data downloads UI

- Add the **downloadButton()** function.
  - The function requires an **outputId** which will be matched in the server side, and a label e.g. **“Download data”**.
- Add a **dataTableOutput()** with 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(inputId = "data_select",
                                 label = "Select the data you want to explore.",
                                 choices = data_list_data_tab)),
           column(6, downloadButton(outputId = 'download_table_csv',
                                    label = 'Download data')),

           DT::dataTableOutput("table_filtered")
           ) # end fluidRow
         ) # end tabPanel
```

Exercise: Building a PHS Shiny dashboard – Data downloads Server

- Create reactive dataset that responds to input from the drop-down menu.
- Use the **switch()** function to switch between datasets shown in the table.
- Use an **if-else statement** to select columns based on drop-down input
  - year, type, measure and value for “data_allergy”
  - diagnosis, year, sex, age_grp, numerator and rate for “data_asthma”
  - **NOTE:** inputId for drop-down from UI must match!

```
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

- Render the data table using **DT::renderDataTable({…})** which we used in the first session. Use the Id **“table_filtered”** to match the UI.
- **Optional**: Format the column names.
- Use DT::datatable for further formatting, you can look up the package info and see what else is available for these settings.

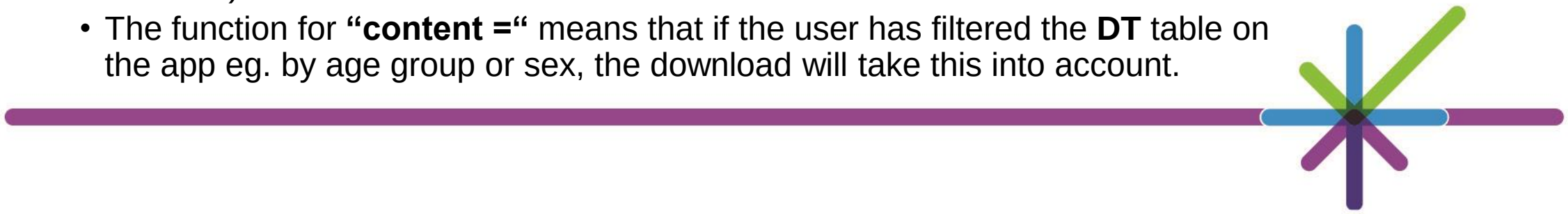
```
output$table_filtered <- DT::renderDataTable({

  # Remove the underscore from column names in the table
  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

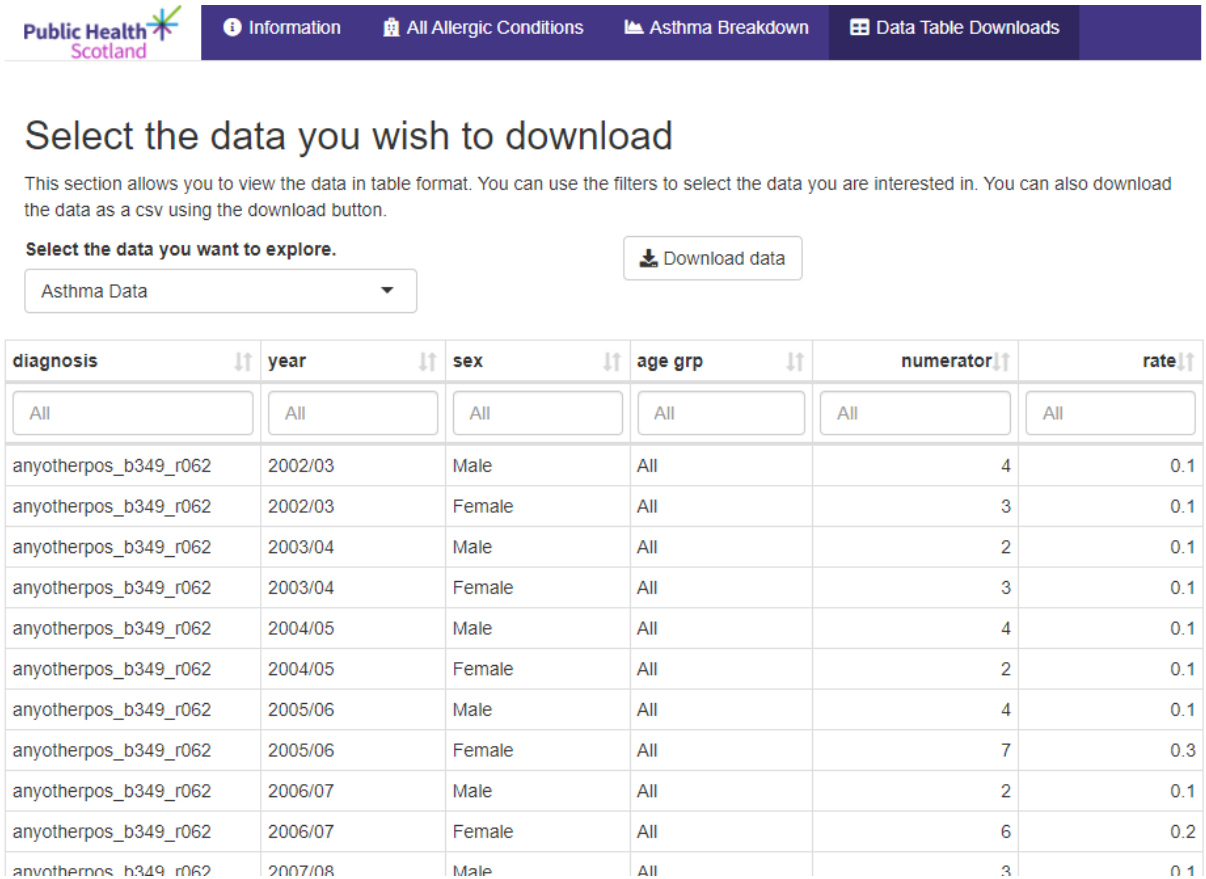
- Make the **download button** functional by inserting the following code.

```
output$download_table_csv <- downloadHandler(
  filename = function() {
    paste(input$data_select, ".csv", sep = "")
  },
  content = function(file) {
    # This downloads only the data the user has selected using the table filters
    write_csv(data_table()[input[["table_filtered_rows_all"]], ], file)
  }
)
```

- Use the Id "**download_table_csv"** as defined in the UI.
- The **downloadHandler()** function defines the file name and the contents of the file.
  - The function for **"filename ="** tells Shiny to name the file based on the user input (the data selected) and to add **.csv** at the end.
  - The function for **"content ="** means that if the user has filtered the **DT** table on the app eg. by age group or sex, the download will take this into account.



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!)**

Shiny dashboards and GitHub

- Shiny dashboards can end up being thousands upon thousands of lines of code, with live ones sometimes updated monthly, weekly, or even daily. Often with large dashboards, multiple members of PHS staff can be working on them at once.
- At this point, version control becomes **very important**.
- GitHub allows for code tracking, sharing and collaboration.
- Multiple team members can be working on the same Shiny dashboard and GitHub allows them to do this on different code branches so the overall master code is not affected, and they do not overwrite each others work.
- Team members can also access, check and review each others changes.

If you haven't checked out our teams **GitHub Training Course**, please do so.

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.
- If you are creating massive apps, you can create supporting scripts and source them into the app, for example you may create 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 and don’t load full packages if they aren’t necessary eg. the whole tidyverse.
- 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.
- Keep data wrangling to a minimum within the app – This can improve the efficiency of the app when a user is viewing it. It’s better to have smaller but multiple datasets.

End of day 2 – any questions?