October 2024

R Shiny Masterclass Series - Advanced

Session 1
Introduction, recap and responsive interfaces in R Shiny

EPI-interactive

How we run the workshop

- Zoom
- Posit Cloud: code, assignments
  - https://posit.cloud/spaces/567774/content
- Learning platform: slides, exercises, discussion forum
- Agenda

How we run the workshop

- Recording of sessions
- We encourage you to turn on video
- Please mute your mic when you are not speaking
- Ask your question in the Zoom chat – will be addressed either throughout or at the end of each section
- Use the chat or raise your hand for questions and comments (don’t forget to un-mute)
- Polls and breakout rooms

How we run the workshop

Learning platform - Course material, assignments and forum

https://learn.epi-interactive.com

Course overview

This advanced masterclass will expand on basic R Shiny functionality and teach you how to tackle more complex features in a structured way. We will dive into creating dynamic interfaces for multi devices and explore R packages that provide a richer user experience.

Introduce yourself

- Name
- Organization
- Why are you joining the Masterclass?
- What are you looking forward to the most?
- Fun fact about yourself

Agenda

Session 1 | October 28th | Introduction, recap and responsive interfaces in R Shiny

Session 2 | October 29th | Advanced reactivity and UX considerations

Session 3 | October 31st | Useful R packages to extend core Shiny functionality

Session 4 | November 4th | Managing complexity: modularizing with the module pattern

Session 5 | November 5th | Case Study, Advanced data sources and processing

Session 6 | November 7th | Automated report generation

Session 7 | November 11th | User authentication, Extended exercise

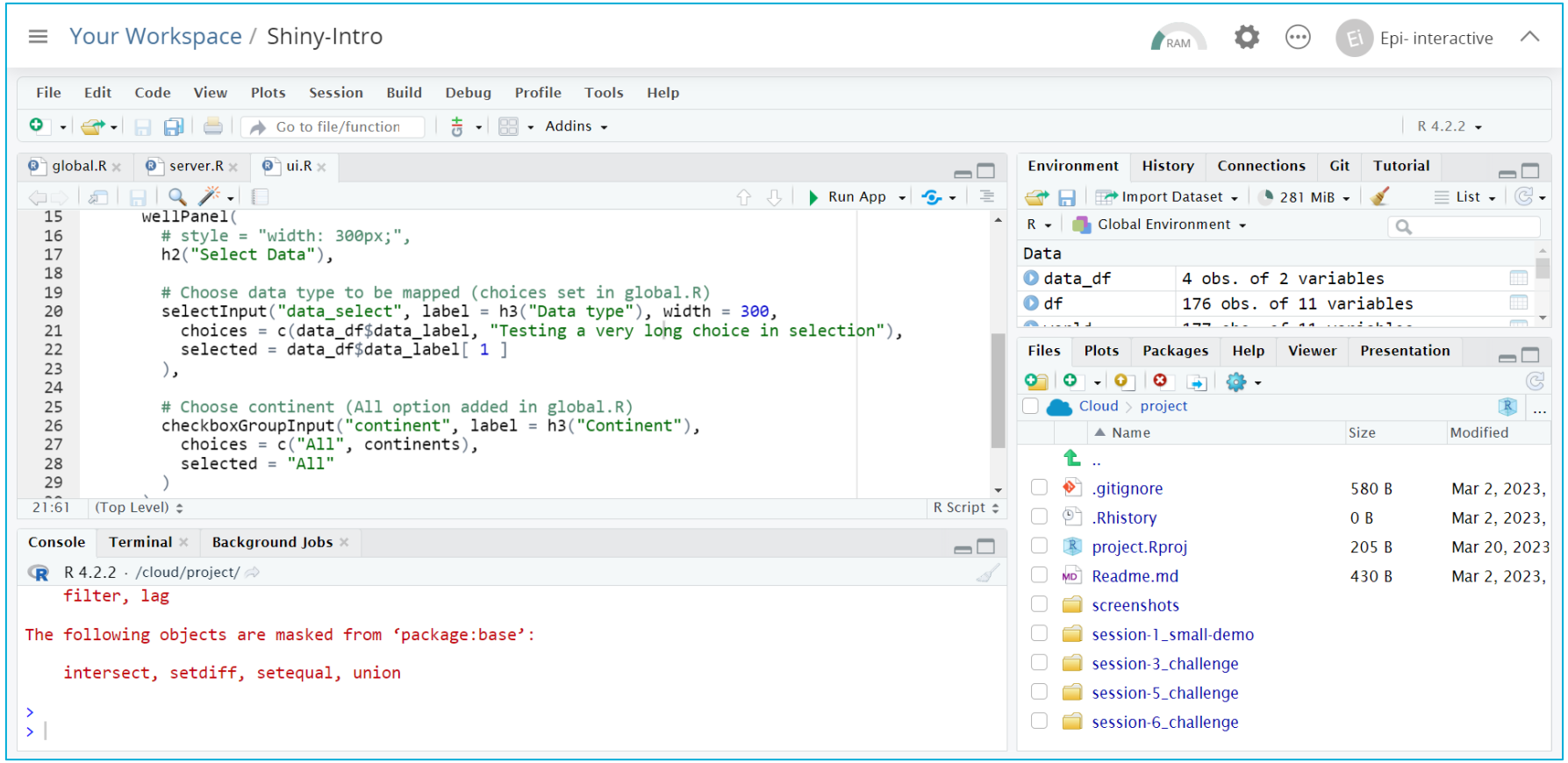
Session 8 | November 12th | AI Tools, Programming sins and how to avoid them

Today

- Recap introductory masterclass concepts
- Responsive interfaces in R Shiny
- Bootstrap grid expanded
- Media queries
- RenderUI

Introduction Masterclass: Recap

Recap: Posit Cloud



Recap: ui.R / server.R / global.R

R files which will be automatically recognised in an R Shiny app:

ui.R:

- Define layout, look / feel of project
- Defined with R Shiny components, not R code

server.R:

- Define functionality of R Shiny project
- All R code, defined inside of R function

global.R:

- Define R code for common variables / functions
- Runs first in R Shiny application process

Recap: Shiny UI components

- Static tags
  - HTML()
  - p("paragraph") becomes paragraph
  - Tags with children: tags$ul(tags$li())
  - Tag specific parameters: tags$img(src=…)
- Navigation
  - Tabsets, Navlist, Navbar
  - tabPanel
- Layouts
  - fluidPage, fixedPage,
  - titlePanel, sidebarLayout, sidebarPanel, mainPanel, wellPanel

Example: https://shiny.rstudio.com/articles/layout-guide.html

Recap: inputs/outputs

Shiny inputs:

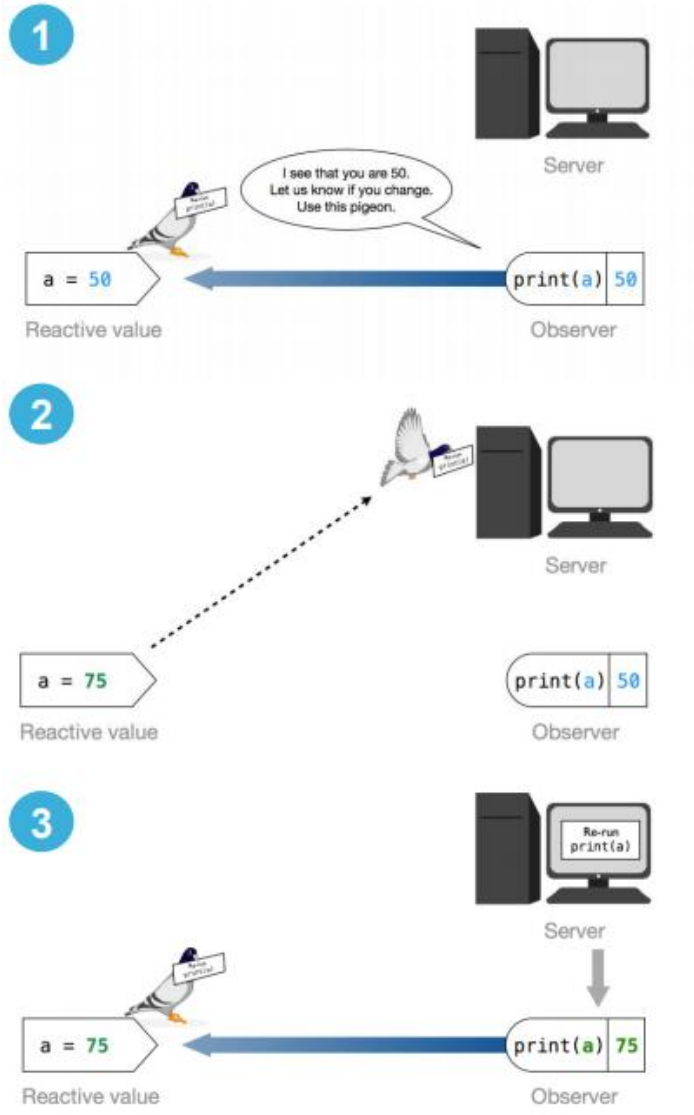
- UI components (widgets) that allow the user to interact with the app
- Widgets will send messages to the necessary components when they are changed
- Generally found in the ui.R file
- Examples: sliderInput, dateInput, selectInput, checkboxInput, textInput
- https://shiny.posit.co/r/gallery/widgets/widget-gallery/

Shiny outputs:

- Displays results based off a given input
- Examples: tableOutput, textOutput, plotOutput
- Needs to be both in ui.R and server.R
- Examples: textOutput(textID) in ui.R, output$textID <- renderText(…) in server.R

Recap: reactivity

- Core mechanic behind responsive R Shiny apps
- App updates itself when user makes changes
- 3 Components:
  - Server
  - Reactive value
  - Observer



Recap: reactivity

- Reactive values
  - E.g. input$slider, reactive() function results
- Reactive context
  - Can only access reactive values from inside a reactive context
- What counts as a reactive context?
  - reactive() functions
  - render functions *(e.g. renderPlot, renderUI…)*
- *Reactivity is lazy*

Recap: debugging

Debugging in Shiny can be challenging due to reactivity, code running behind Shiny framework.

Important to have a good strategy for debugging!

Debugging tools:

- Reset R session
- Clear environment
- Reprex
- print()
- View()
- browser()

Recap: Data sources

- Different data types:
    - Arbitrary data (files)
    - Structured rectangular data (tables, databases)
- Different data sources:
    - Local
    - Remote
- Serialised vs. deserialised data
- Examples: CSV and RDS files

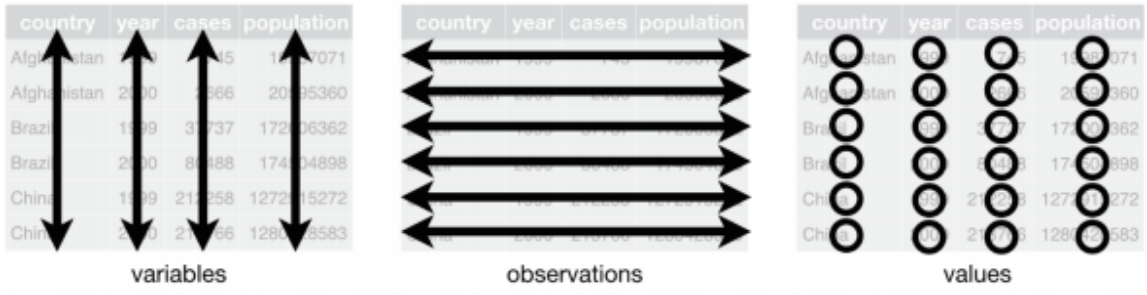
What points to consider when making decisions on your data?

Recap: data processing

- Data processing guidelines:
  - Each column should represent a single variable, each row should represent a single observation.
  - Do the largest modifications first. Order matters!

Common tidyverse functions for manipulating data:

- rename("new_name" = "old_name")
- mutate(col_name = col_name * 5)
- select(col_name, …)
- filter(col_name == …)
- group_by(col_name)
- summarise(…)

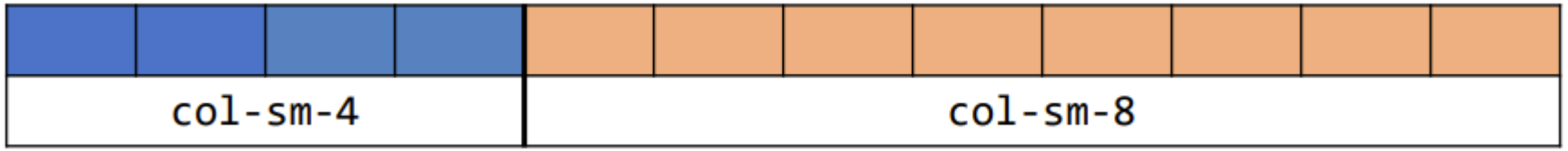


EPI-interactive

Responsive interfaces in R Shiny

Bootstrap grid

- Break up page elements into 12 equal width units
- Use these units to define the size of content on the page:



- Guaranteed vertical ordering with rows
- Columns may have variable height, don’t always need to add up to 12 units across
- Rows and columns can be freely combined and nested for precise control
- Use containers to control page width / sizing rules

Bootstrap grid

- 5/6 Different screen sizes to work with:
  - Extra small (<= 576px)
  - Small (<= 768px)
  - Medium (<= 992px)
  - Large (<= 1200px)
  - Extra large (<= 1400px)
  - Extra extra large (Bootstrap 5, > 1400px)

- Sizing rules for columns will apply to that screen size or higher
- Columns with no rules defined for a screen size default to full width

Example: Bootstrap layout in Shiny

```
fluidPage(
        fluidRow(
                column(6, “Column A”),
                column(6, “Column B”)
        ),
        fluidRow(
                column(4, “Column C”)
        )
)
```

```
<div class="container-fluid">
    <div class="row">
        <div class="col-sm-6">
                Column A
        </div>
        <div class="col-sm-6">
                Column B
        </div>
    </div>
    <div class="row">
        <div class="col-sm-4">
                Column C
        </div>
    </div>
</div>
```

Exercise : Reverse Div Soup

- Given the image on the following slide:
  - Examine the structure of the page
  - What is defined using a bootstrap grid layout?
  - What is defined outside of the bootstrap layout?
  - Annotate the image with the row / column structure
  - **Bonus** – create the HTML / Shiny UI structure for this image

- *Note: there can be more than one way to achieve the same layout!*

Our Work

Making data, analytics and research findings beautiful, useful, and accessible is our passion. Our agile and interdisciplinary team can support you in many different ways ranging from expert advice to co-development to full service solutions.

As Oceania's Full Service Certified Posit Partner and with our accredited cloud engineers, we can also help you with your data science infrastructure needs. Learn more on our Posit page.

Just for Fun

Have a look back at our past Christmas projects and other fun extras.

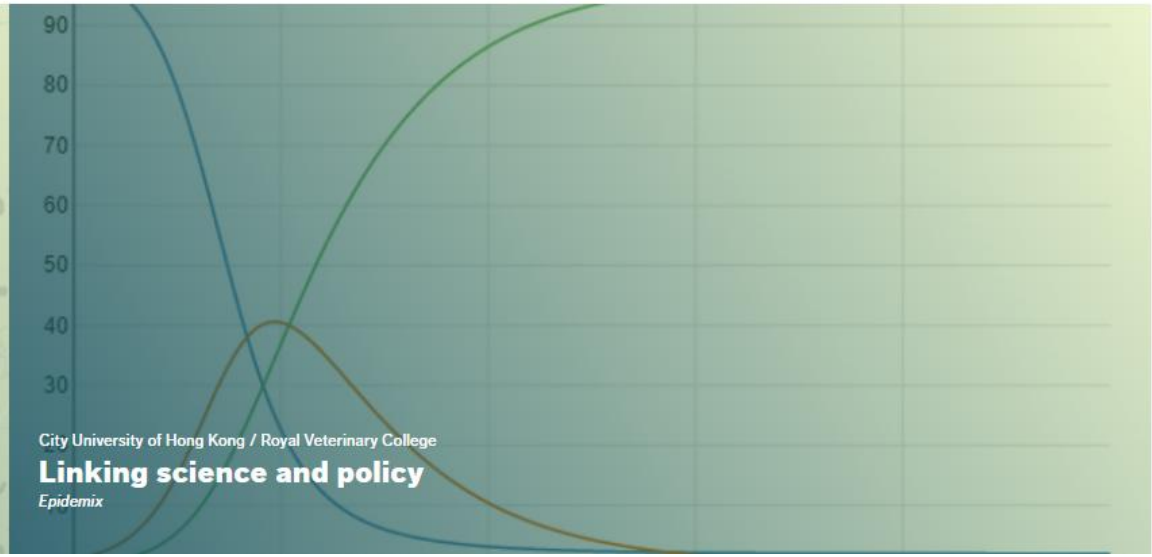
Research & Publications

We actively contribute to the scientific community through publications and presentations.

University of Minnesota / Minnesota Aquatic Invasive Species Research Center

Science-supported biosecurity decision making

AIS Explorer



City University of Hong Kong / Royal Veterinary College

Linking science and policy

Epidemix

World Health Organization

About the GLASS dashboard | Global AMC data | Global AMR data | Country, territory or area profiles

GLASS dashboard

Global antimicrobial resistance and use surveillance system (GLASS): 2022 report

The GLASS dashboard presents global antimicrobial consumption (AMC) and antimicrobial resistance (AMR) data from GLASS-AMC and GLASS-AMR surveillance activities, by means of interactive visualisations. Comprehensive country, territory, and area (CTA) profiles for AMR and AMC are also provided. Dashboards are optimised for use in Google Chrome.

All figures and underlying data are downloadable.

A pdf version of the 2022 report summarising all key findings and messages can be found in the link below.

Go to GLASS report

GLASS
Global Antimicrobial Resistance and Use Surveillance System

Global AMC data

Explore

Global AMR data

Explore

Country, territory or area profiles

Explore

EPI-interactive

ESR Science for Communities

MINISTRY OF HEALTH MANATŪ HAUORA

NZ COVID-19 Dashboard

EPI-interactive

For further epidemiological analysis please refer to ESR's COVID-19 Epidemiology Reports.

Updated: 27/05/2021 09:00 AM

▲ 1	No change	2623 Recovered
2314 Confirmed cases	356 Probable cases	26 Deceased

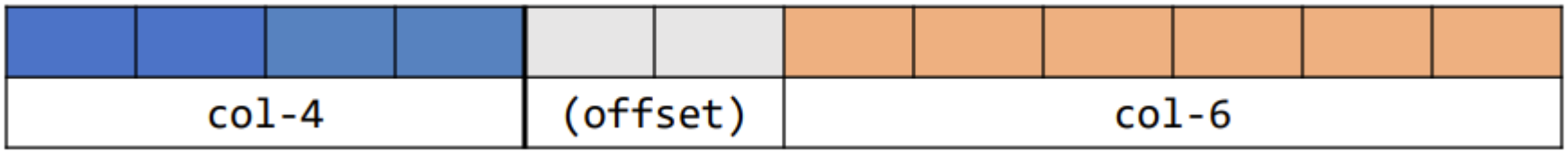
Total cases 2670

Time period: 1 Jan 2021 - current date

EPI-interactive

Bootstrap grid: offset

- Provide spacing to the left of a column by defining an offset
- Example:
  - **column(6, offset = 2)** defines a column which is **6** units wide with a **2-unit** offset.
  - fluidRow(column(4), column(6, offset = 2)) becomes:



- Columns defined with an offset still follow the 12-unit rule!

bootstrapPage

- Automatically loads required Bootstrap CSS / JS code
- Allows selection of specific Bootstrap versions
  - Shiny 1.7.0+ <= Bootstrap 5
  - Shiny 1.6.0 <= Bootstrap 4

- bootstrapPage(***theme = bslib::bs_theme(version = 5)***)

- No other HTML included
- Combine with fluidPage() or fixedPage() for easy page structure

bootstrapLib

- Automatically loads required Bootstrap CSS / JS code
- Allows selection of specific Bootstrap versions
  - Shiny 1.7.0+ <= Bootstrap 5
  - Shiny 1.6.0 <= Bootstrap 4

```
bootstrapLib(theme = bslib::bs_theme(version = 5))
```

- No other HTML included
- Using later versions of Bootstrap enables new features!

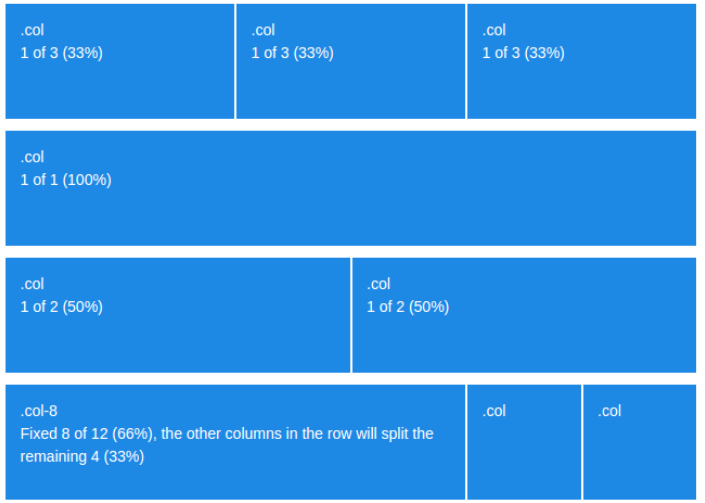
Bootstrap grid: DIY

- Instead of column(6, …), we use div(class = “col-sm-6 … ”, …)
- Need to import Bootstrap specifically for this approach:
    - Use a **bootstrapPage()** instead of a **fluidPage()**
    - Use the **bootstrapLib()** function to load the correct bootstrap version
- What do these mean? How will these look at different screen sizes?
    - **column(6)**
    - **column(6, offset = 6)**
    - **div(class = “col-sm-6 col-md-4 col-lg-3”)**
    - **div(class = “col-12 col-sm-6 col-md-4 offset-md-8 col-lg-3 offset-lg-9”)**
- Pattern to remember in Shiny: ***type-size-units***

Size is optional

Bootstrap grid: auto-sizing

- Challenges with Bootstrap Grid:
  - Uncertainty about size of content
    - Different screen sizes
    - User input
  - 12 unit structure can get messy
    - Irregular layouts
    - Not everything is divisible by 12



- Bootstrap 4 introduced auto-sizing for columns!

Bootstrap grid: auto-sizing classes

- div(class = “col”, …)
  - Columns will auto-size to equal width with other columns in the row
- div(class = “col-**XX**”, …)
  - Columns will auto-size to equal width at the specified screen size
- div(class = “col-**XX**-auto”, …)
  - Columns will auto-size based on content at the specified screen size
- div(class = “w-100”, …)
  - Bootstrap width class, can be used to force row breaks in auto-sized content

renderUI

So far we have been defining our Shiny app UI in the ui.R file…

- Static
- No access to R variables or input values
- No reactivity

Solution: renderUI

- “Modular” approach to defining the UI
- Can improve readability of complex UI layouts

renderUI

- In ui.R, we use **uiOutput("id")**
- In server.R, we use **output$id <- renderUI({ … })**
- Can integrate UI with reactivity, R code and custom data
- Can create input widgets and more uiOutputs inside renderUI

- Final output needs to be a single UI object (div, tagList etc)

renderUI: example

ui.R

```
ui <- bootstrapPage(
  div(style ="padding: 50px",
    fluidRow(
      column(12, selectInput("select", label = "Select
layout", choices = list("C1" = 1, "C2" = 2, "C3" = 3)))
    ),
    fluidRow(
      uiOutput("maincontent")
    ),
    fluidRow(
      uiOutput("calculatedcontent")
    )
  )
)
```

server.R

```
server <- function(input, output) {
  output$maincontent <- renderUI({
    column(12, h1(paste0("Layout ", input$select)))
  })

  output$calculatedcontent <- renderUI({
    size <- 12 / as.numeric(input$select)
    tagList(
      lapply(X = c(1:input$select), FUN = function(x, size) {
        column(size, wellPanel(paste0("Column ", x)))
      }, size)
    )
  })
}
```

How will this look in a Shiny app?
What does this code do?

Exercise

- Using the /stage1 folder:
- Add in the bootstrapLib element to ui.R to set the bootstrap version to 5
- In ui.R, add a new tabPanel for a table view of our filtered data.

  This tabPanel should include:
  - One row with two auto-sized columns,
    - Second column with an offset of 4 units
    - First column should show details about world_data (nrow, sum of area, sum of pop)
    - Second column should show details about filtered_data (nrow, sum area, sum pop)
  - One row with the dataTableOutput, in an auto-sized column
  - One row for the download button, inside a column with offset of 6 units
  - Each column should have its contents inside of a wellPanel

Media queries

https://css-tricks.com/a-complete-guide-to-css-media-queries/

- CSS technique for defining conditional style rules
- Can change application layout / style for different devices

- Order matters!
- Direction matters!

```
@media screen and (max-width: 768px) {
        body {
          background-color: lightblue;
         }
}
```

Media queries

- Using bootstrap grid with media queries makes it easy to define style behaviour at different screen sizes:

```
// Small devices (landscape phones, 576px and up)
@media (min-width: 576px) { ... }
// Medium devices (tablets, 768px and up)
@media (min-width: 768px) { ... }
```

```
// Large devices (laptops, 992px and up)
@media (min-width: 992px) { ... }

// Extra large devices (desktops, 1200px and up)
@media (min-width: 1200px) { ... }

// Extra extra large devices (Large desktops, 1400px and up) { … }
```

Next time

- Advanced reactivity
- UX considerations

Challenge

- If you haven't already, use the "DIY" approach to create your bootstrap columns
- Change the application so that instead of using a sidebarLayout, the overall structure is created using Bootstrap rows and columns.
- Ensure that the sidebar is 3 units wide on 'large' devices or higher
- Ensure that on 'medium' devices or lower, the sidebar is 12 units wide
- Using CSS Media queries and CSS classes, change the colour of the wellPanels and add a border to your columns at the XS, MD and XL screen sizes