November 2024

R Shiny Masterclass Series - Advanced

Session 3

Useful R packages and User Experience (UX)

EPI-interactive

Agenda

Session 1 | October 30th | Introduction, recap and responsive interfaces in R Shiny

Session 2 | November 1st | Advanced reactivity

Session 3 | November 5th | Useful R packages to extend core Shiny functionality

Session 4 | November 6th | Managing complexity: modularizing with the module pattern

Session 5 | November 8th | Case Study, Advanced data sources and processing

Session 6 | November 12th | Automated report generation

Session 7 | November 13th | User authentication, Extended exercise

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

Today

- Reactivity extra credit
- Techniques to improve overall UX
- Better tables with DT
- Adding JavaScript with shinyjs

Reactivity extra credit

Reactivity extra credit

Goal:

Adding additional functionality to our animal voting example from session 2, using the expanded reactivity features from Session 2.

The code is in ***reactivity-extra*** folder in Session-3 as our starting point.

Eager evaluation

Feature	observe	observeEvent
Expects a return value	✕	✕
Able to watch many values	✓	✓
Contains an isolated scope	✕	✓
Reactive context	✓	✓
Can be reassigned	✕	✕
Create dependents*	✕	✕
Can execute code**	✓	✓

**Dependents are reactive values that can trigger changes with reactive contexts*
***Rather than just containing values*

Lazy evaluation

Feature	reactive()	eventReactive	reactiveValues / reactiveVal
Expects a return value	✓	✓	✗
Able to watch many values	✓	✓	✗
Contains an isolated scope	✗	✓	✗
Reactive context	✓	✓	✗
Can be reassigned in place	✗	✗	✓
Create dependents*	✓	✓	✓
Can execute code**	✓	✓	✗

**Dependents are reactive values that can trigger changes with reactive contexts*
***Rather than just containing values*

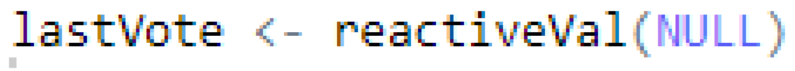
Implementing an undo button

Requirements:

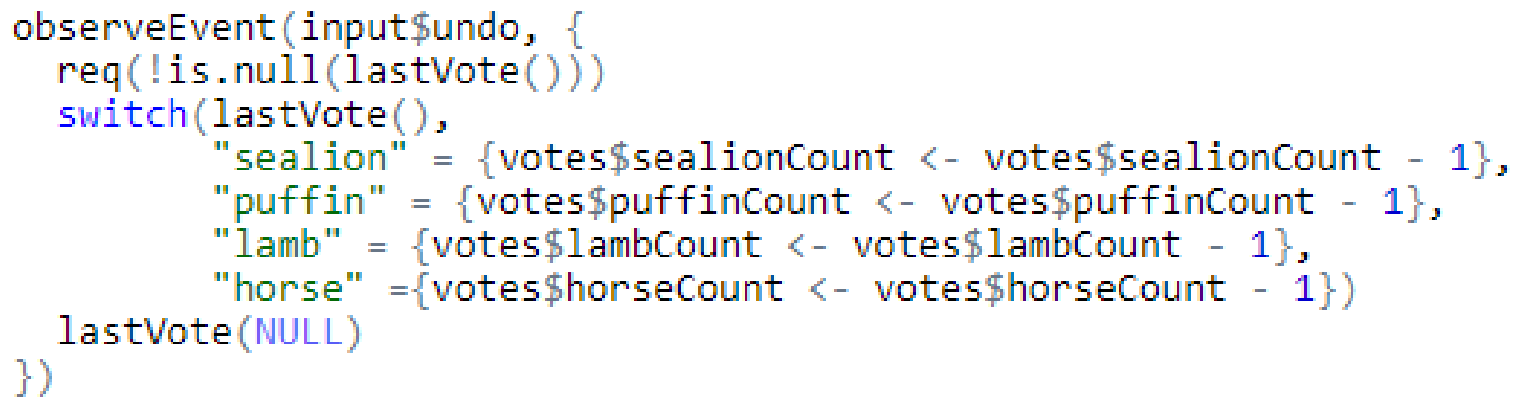
- Once the user has voted, allow them to undo the last vote.
- The user should not be able to undo a single vote more than once.

- What do we use to keep track of the last vote?
- How often do we need to keep track of it?
- Once the user hits undo, how do we ensure the vote can only cleared once?
- How do we prevent the user from undoing a vote before anything has been voted for?

Implementing an undo button



```r
lastVote <- reactiveVal(NULL)
```



```r
observeEvent(input$undo, {
  req(!is.null(lastVote()))
  switch(lastVote(),
         "sealion" = {votes$sealionCount <- votes$sealionCount - 1},
         "puffin" = {votes$puffinCount <- votes$puffinCount - 1},
         "lamb" = {votes$lambCount <- votes$lambCount - 1},
         "horse" ={votes$horseCount <- votes$horseCount - 1})
  lastVote(NULL)
})
```

Reveal final votes on a timer

Requirements:

- Instead of revealing the final votes on the press of a button, reveal the votes when time is up.
- A new timer will start counting down only after the reset button is pressed.

```
countdown <- reactiveVal(10)
observe({
          invalidateLater(1000, session) #Trigger every second
          isolate(countdown(countdown() - 1))
})
```

- How do we decide when the timer should stop? How do we put this in code.
- Once it has stopped, how to display the final results?
- How do we start the timer again on reset?

Reveal final votes on a timer

```r
totalVotes <- eventReactive(list(countdownStopped(), input$reset),{
  results()
}, ignoreInit = T)
```

```r
output$timer <- renderText(paste("Time remaining:",countdown()))
countdownStopped <- reactiveVal(F)
countdown <- reactiveVal(10)

observe({
  if(!countdownStopped()){
    if(isolate(countdown()) != 0 ){
      invalidateLater(1000, session)
      isolate(countdown(countdown() - 1))
    }else{
      countdownStopped(T)
    }
  }
})
```

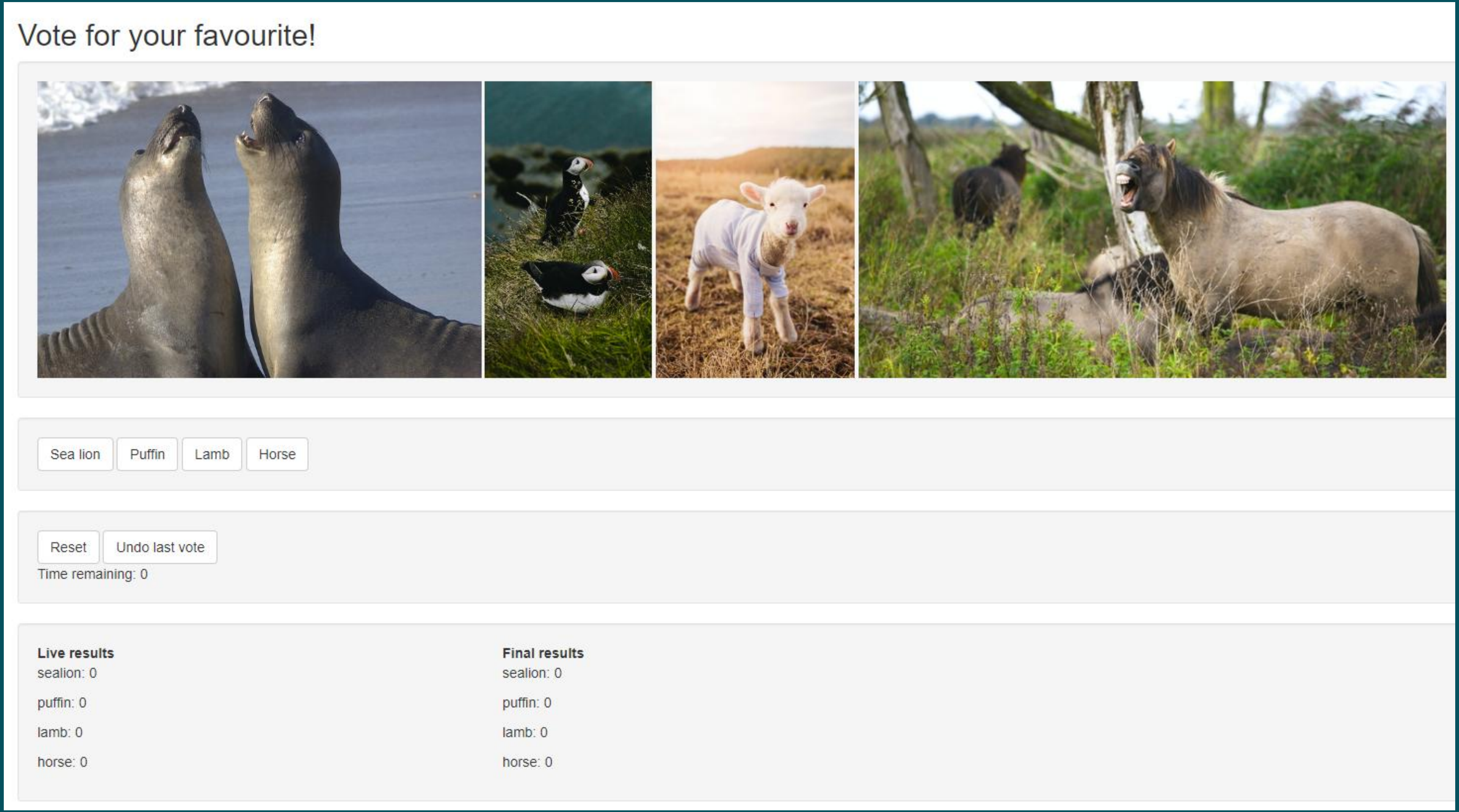
```r
observeEvent(input$reset, {
  votes$sealionCount <- 0
  votes$puffinCount <- 0
  votes$lambCount<- 0
  votes$horseCount<- 0
  countdownStopped(F)
  countdown(10)
}, priority = 1)
```

Extra credit – undo button and timer

Using our answers to the questions from the previous slides, lets:

- Implement an undo button, which reverts the last vote (*bonus: how could we enable* ***undo*** *for multiple votes? Have a go!*)
- Remove the ‘show results’ button and implement a **timer** which will show the *finalDisplay* uiOutput when the timer reaches 0
- Ensure that both the ‘***undo***’ and ‘***timer***’ features are handled appropriately by the ‘***reset***’ button.

Extra credit - example



Vote for your favourite!

Time remaining: 0

Live results
sealion: 0
puffin: 0
lamb: 0
horse: 0

Final results
sealion: 0
puffin: 0
lamb: 0
horse: 0

Error Handling

Error handling: req

- Pauses execution to check if a certain condition is “Truthy”
  - If that condition is Truthy, execution resumes without incident
  - If the condition is not Truthy, stops processing with a “silent” exception (I.e. not logged by Shiny & does not show in UI)
- Can check multiple conditions in a single req()
- Check state of reactive values, prevent reactives triggering before dependents are ready

Example:

- Ignore return value: `req(!is.null(input$someInput), input$someInput > 0)`
- Use return value: `if(req(input$value > 0, !is.null(input$someInput)) { print(“Not null!”) }`

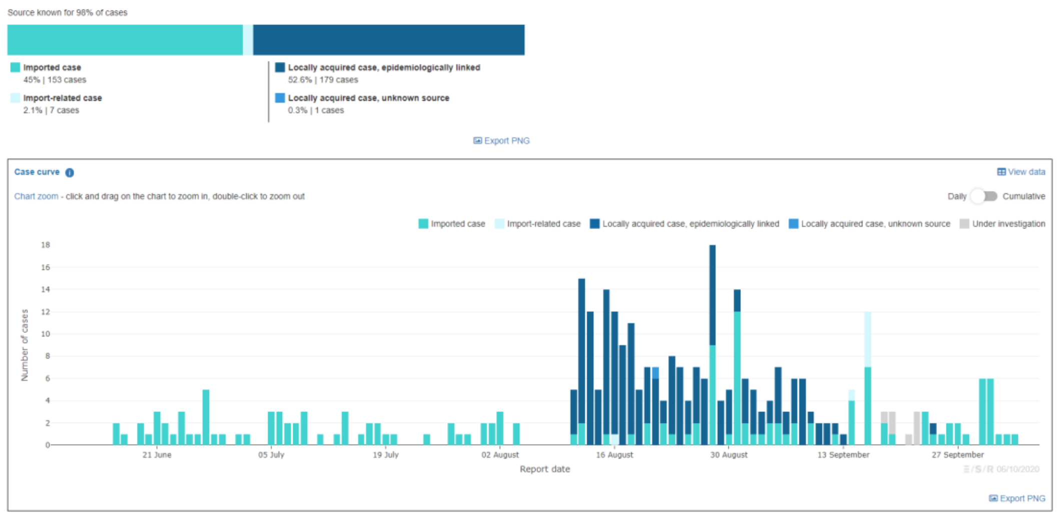
Error handling: validate & need

- Error handling specifically for rendered output elements in R Shiny
- Check that provided conditions are “Truthy” before rendering element
    - If conditions are all Truthy, render the output
    - If conditions are not all Truthy, display a (customisable) Shiny error message
- Useful mechanism for checking that data is available before rendering
- Prevents empty charts / tables being produced.

Error handling: validate

Example – NZ Covid Dashboard:

```
validate(need(nrow(outputData()) > 0,
message="No cases for selected filters"))
```


Source known for 98% of cases

🟩 **Imported case**
45% | 153 cases

⬜ **Import-related case**
2.1% | 7 cases

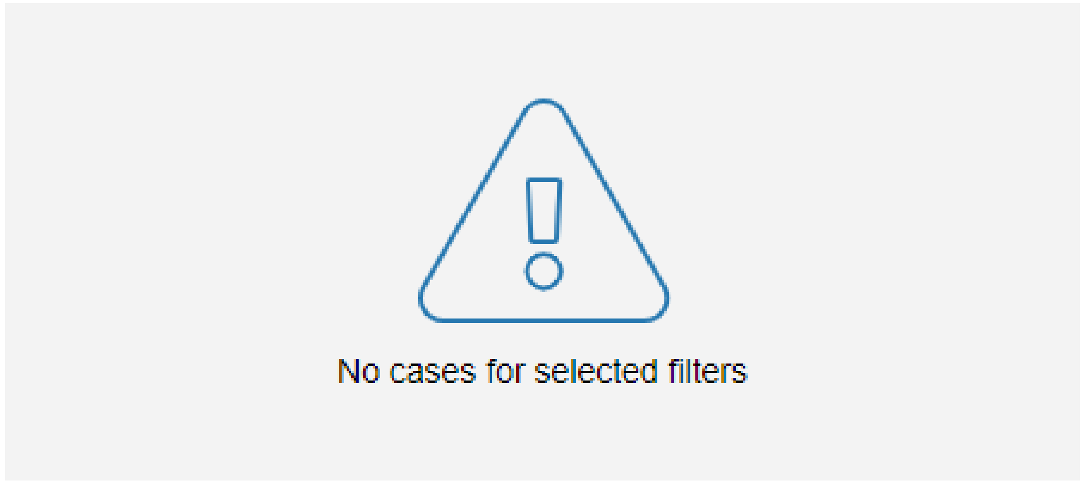
🟦 **Locally acquired case, epidemiologically linked**
52.6% | 179 cases

🟦 **Locally acquired case, unknown source**
0.3% | 1 cases

Export PNG

Case curve

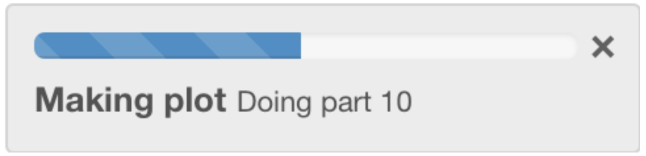
Chart zoom - click and drag on the chart to zoom in, double-click to zoom out


No cases for selected filters

Other UX features

Progress bars: R Shiny version

- Built-in Shiny progress indicator
- Gives some visual indication of progress
- Call inside of desired UI output object
- Need to know ahead of time how many steps / iterations there are



```
withProgress(message = 'Making plot', value = 0, {
n <- 10
for (i in 1:n) {
dat <- rbind(dat, data.frame(x = rnorm(1), y =
rnorm(1)))
incProgress(1/n, detail = paste("Doing part", i))
Sys.sleep(0.1)
}
}
```

```
progress <- Progress$new(session, min=1, max=15)
on.exit(progress$close())
progress$set(message = 'Calculation in progress',
detail = 'This may take a while...')
for (i in 1:15) {
    progress$set(value = i) Sys.sleep(0.5)
}
```

EPI-interactive

Progress bars: nProgress

- JavaScript alternative to R Shiny progress bars
- Provides slim progress bar across the top edge of Shiny application
- Primarily cosmetic (I.e. not representative of actual progress), can be modified to show actual progress.
- To use, need to import the nprogress css / js files, then define when the progress bar starts / stops in a JS script:

```
// Start NProgress when starting calculations
$(document).on('shiny:busy', function(event) {
    NProgress.start({
        showSpinner: false,
    });
});
```

```
// End NProgress when shiny goes idle
$(document).on('shiny:idle', function(event) {
    NProgress.done();
});
```

Spinners

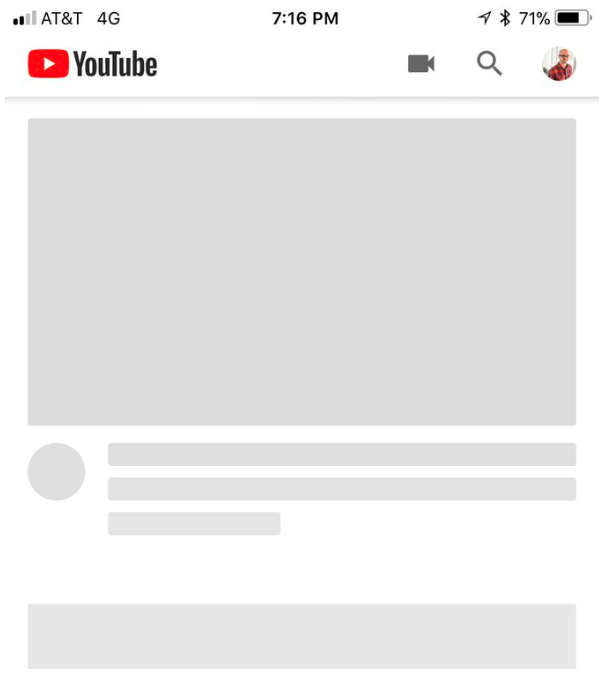
- Like nProgress, implemented with JavaScript
  - shiny:busy and shiny:idle
- Can be made in CSS using animation
- Primarily cosmetic

- nProgress comes with an optional spinner at the top, implementing your own means you can place it anywhere

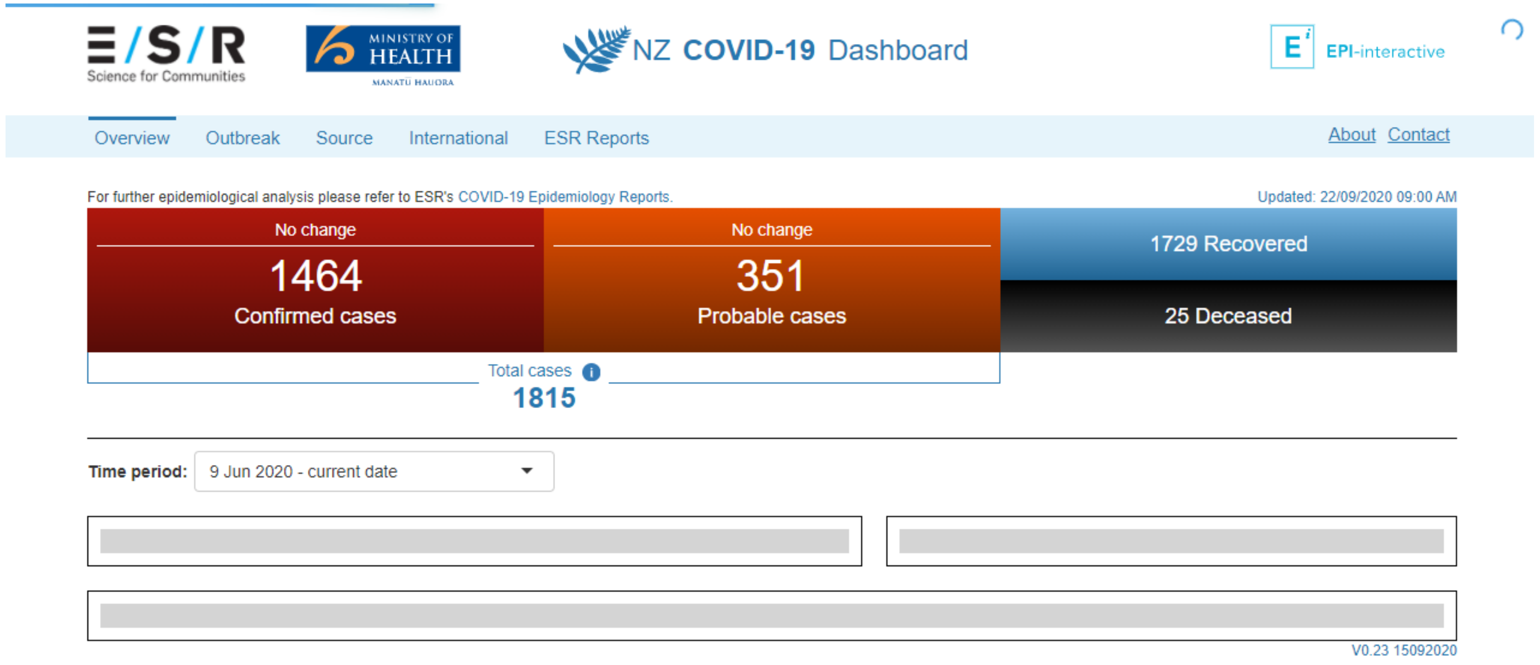
Skeletons

- Placeholder UI elements while content is loading
- Gives loose indication of layout and content ahead of time
- Improves the “perceived performance” of an app
- Widely used UX technique
- Important to design for the loading times, app won’t always run perfectly.

Skeletons: example



https://twitter.com/merhl/status/930834669670731776



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

Updated: 22/09/2020 09:00 AM

No change
1464
Confirmed cases

No change
351
Probable cases

1729 Recovered

25 Deceased

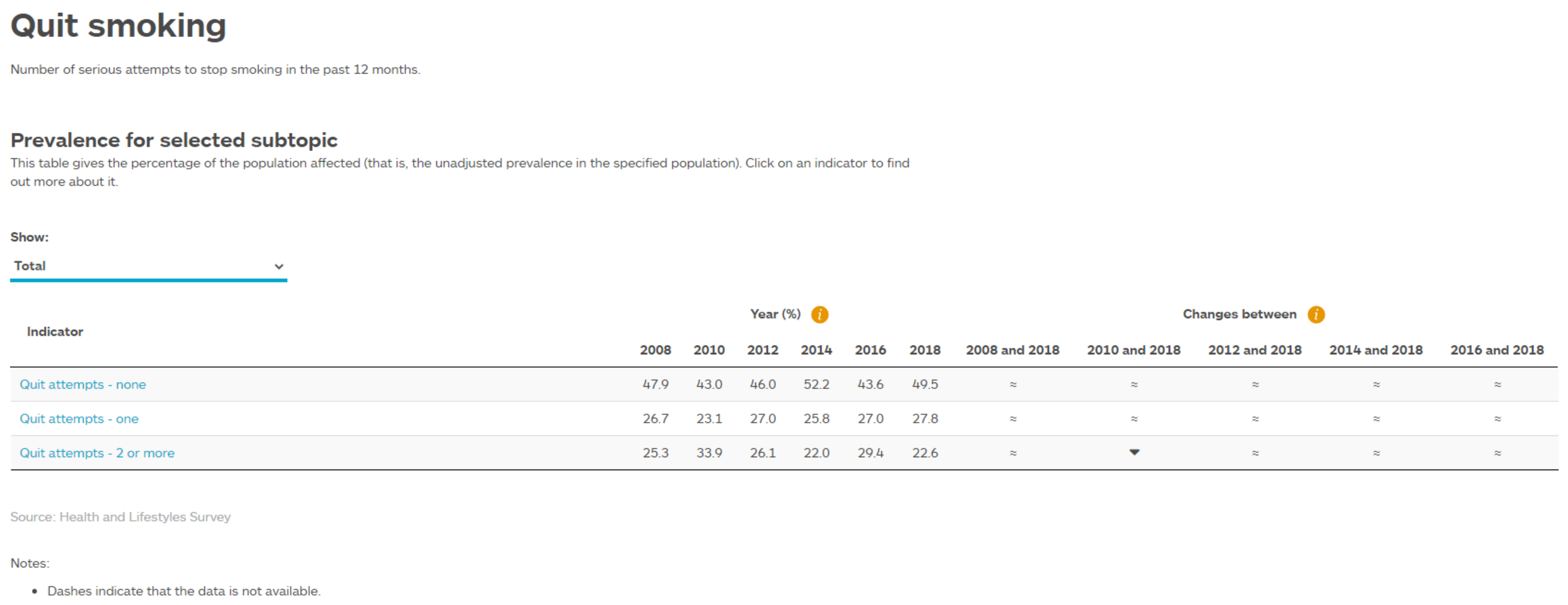
Total cases
1815

Time period: 9 Jun 2020 - current date

V0.23 15092020

Better tables with DT

Example: Kupe



Quit smoking

Number of serious attempts to stop smoking in the past 12 months.

Prevalence for selected subtopic

This table gives the percentage of the population affected (that is, the unadjusted prevalence in the specified population). Click on an indicator to find out more about it.

Show:

Total

Indicator	Year (%)						Changes between				
	2008	2010	2012	2014	2016	2018	2008 and 2018	2010 and 2018	2012 and 2018	2014 and 2018	2016 and 2018
Quit attempts - none	47.9	43.0	46.0	52.2	43.6	49.5	≈	≈	≈	≈	≈
Quit attempts - one	26.7	23.1	27.0	25.8	27.0	27.8	≈	≈	≈	≈	≈
Quit attempts - 2 or more	25.3	33.9	26.1	22.0	29.4	22.6	≈	▼	≈	≈	≈

Source: Health and Lifestyles Survey

Notes:

- Dashes indicate that the data is not available.

EPI-interactive

DT::renderDataTable

- Recommended by RStudio blog
- Built off both server-side and client-side DataTables (JS library)
  - Shiny version can only use the server side
- More options including editable tables, customisation
- https://rstudio.github.io/DT/

DT: Creating a table

server.R

```
output$myDataTable <-  DT::renderDataTable({
      datatable(data,
      options = list(),
      …
      )
})
```

ui.R

```
DT::dataTableOutput(“myDataTable”)
```

EPI-interactive

Containers

- Allows you to provide a different table container to hold the table cells
  - For example: as a header or a footer
- Add it to the datatable function using the container parameter
- Need to use htmltools::withTags() to create the new table

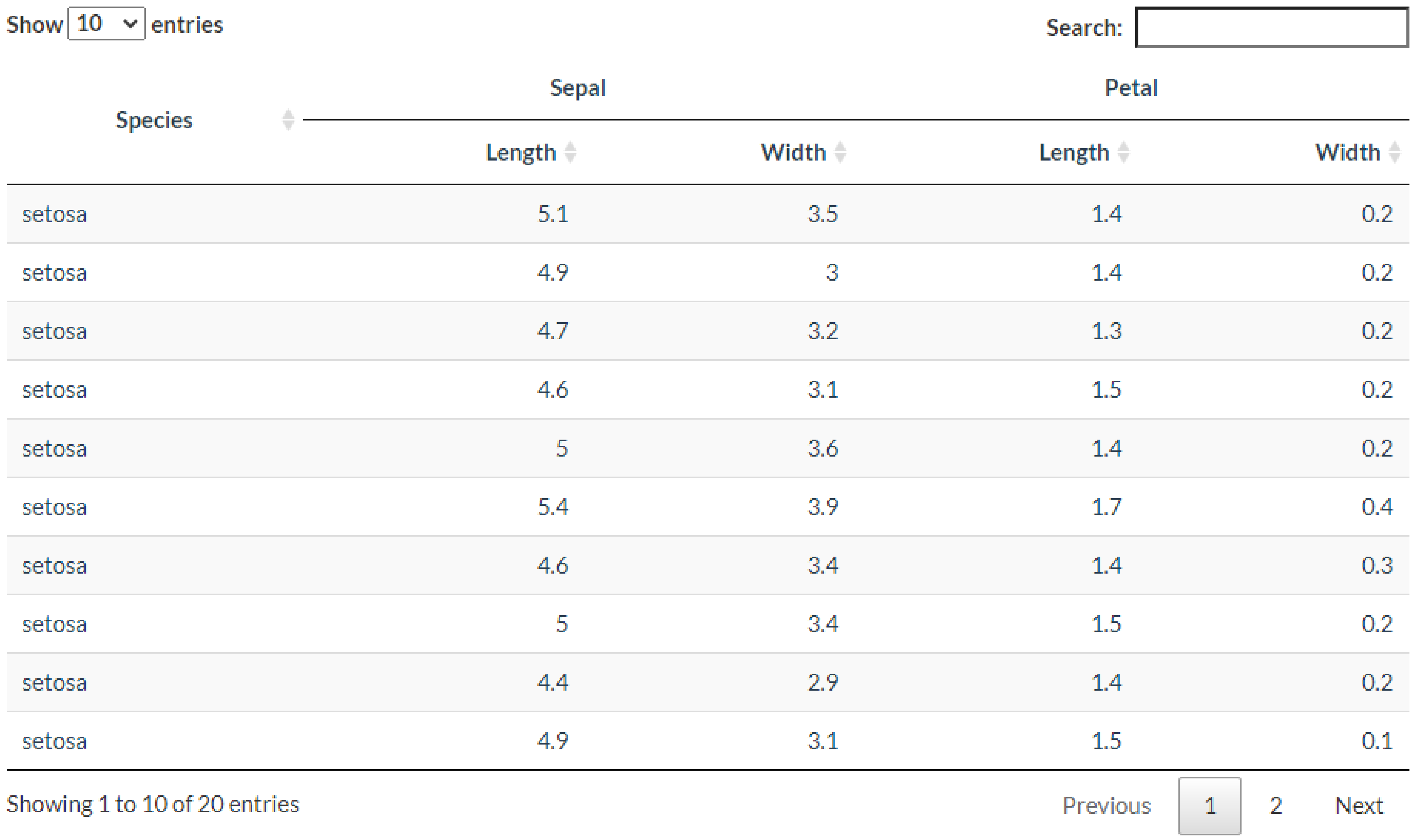
Containers: Example

```r
container = htmltools::withTags(table(
  class = 'display',
  thead(
    tr(
      th(rowspan = 2, 'Species'),
      th(colspan = 2, 'Sepal'),
      th(colspan = 2, 'Petal')
    ),
    tr(
      lapply(rep(c('Length', 'Width'), 2), th)
    )
  )
))

datatable(iris[1:20, c(5, 1:4)], container =
container, rownames = FALSE)
```

```html
<table class="display">
  <thead>
    <tr>
      <th rowspan="2">Species</th>
      <th colspan="2">Sepal</th>
      <th colspan="2">Petal</th>
    </tr>
    <tr>
      <th>Length</th>
      <th>Width</th>
      <th>Length</th>
      <th>Width</th>
    </tr>
  </thead>
</table>
```

Result


Show 10 entries

Search:

Species	Sepal		Petal	
	Length	Width	Length	Width
setosa	5.1	3.5	1.4	0.2
setosa	4.9	3	1.4	0.2
setosa	4.7	3.2	1.3	0.2
setosa	4.6	3.1	1.5	0.2
setosa	5	3.6	1.4	0.2
setosa	5.4	3.9	1.7	0.4
setosa	4.6	3.4	1.4	0.3
setosa	5	3.4	1.5	0.2
setosa	4.4	2.9	1.4	0.2
setosa	4.9	3.1	1.5	0.1

Showing 1 to 10 of 20 entries

Using JavaScript with shinyjs

shinyjs

- https://github.com/daattali/shinyjs
- Extends shiny using JavaScript (JS)
- Common functions
  - show() / hide() / toggle()
  - enable() / disable() / toggleState()
  - runjs()

Hide vs Disable

Hide:

- Show users what they need, when they need it
- Clarity for the user
- Simplicity

Disable:

- Visibility for the user
- Keeps layout consistent
- We can tell the user what they need to use disabled feature

The choice is yours!

Hide & Disable: Syntax

On event:

ui.R:

```
useShinyjs()
```

server.R:

```
observeEvent(input$button, {
       toggleState("radio") #disable
       toggle("checkbox") #hide
})
```

Conditionally:

ui.R:

```
useShinyjs()
```

server.R:

```
observeEvent(input$button, {
       if(…){
         disable("radio")
         hide("checkbox")
       }else{
        enable("radio")
        show("checkbox")
       }
})
```

EPI-interactive

Combining JS and Shiny

External file

[name].js:

```
const scrollToTop = function()
{
  window.scrollTo(0,0);
}
```

server.R:

```
runjs(“scrollToTop()”)
```

ui.R:

```
tags$script(src=“[name].js")
useShinyjs()
```

Inline

server.R:

```
runjs(
  “window.scrollTo(0,0);”
)
```

ui.R:

```
useShinyjs()
```

Exercise

Our starting point today is a modified version of our world_data app template, with some problems!

Using ***/stage1***:

- Use ShinyJS to enable the ‘Go’ button when some country(ies) are selected or disable the button otherwise.
- Use req() to prevent the filtered_data eventReactive from calculating if input$country is not ready (I.e. no countries selected)
- Use validate() & need() to show a warning if there no rows in the filtered data
  - This should show up in the chart, table and filtered_summary parts of the app
- Modify the output$table so that:
  - The table sorts by ‘GDP Per Capita’ in ascending order by default
  - Rows with life expectancy >= 65 are styled with a background colour of orange
  - *Hint: use the hidden column ‘life_exp_status’ along with the formatStyle and styleEqual functions!*

Next time

- Managing complexity: modularising code with the module pattern

Challenge (using /result):

- Create a reactive that will store the current valid countries (based on the filters)
- Create a checkbox input before the slider to (De)Select all valid countries.
    - Use *updateSelectInput* to update the selected countries
    - If the subregion or region is changed, reset the checkbox value to False
- Use *hide*/*show* to hide the subregion filter if the region filter is 'All'
- If the GDP is higher than the worldwide average ($11,330) make the whole row for that country green in the table
- Share your work on the Session 3 forum!