November 2024

R Shiny Masterclass Series - Advanced

Session 6

Automated outputs and report generation with Quarto and LaTeX.

EPI-interactive

Agenda

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

Session 2 | November 1st | Advanced reactivity and UX considerations

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 | Advanced data sources and processing

Session 6 | November 12th | Automated report generation

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

Session 8 | November 15th | AI Tools, Programming sins and how to avoid them

Today

- Profiling
- Automated report generation with Quarto
- Image downloads*

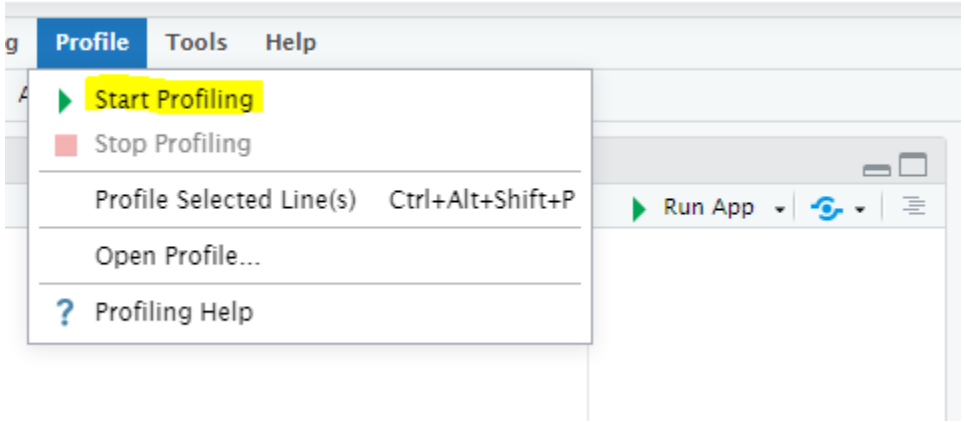
Profiling

Profiling / Profvis

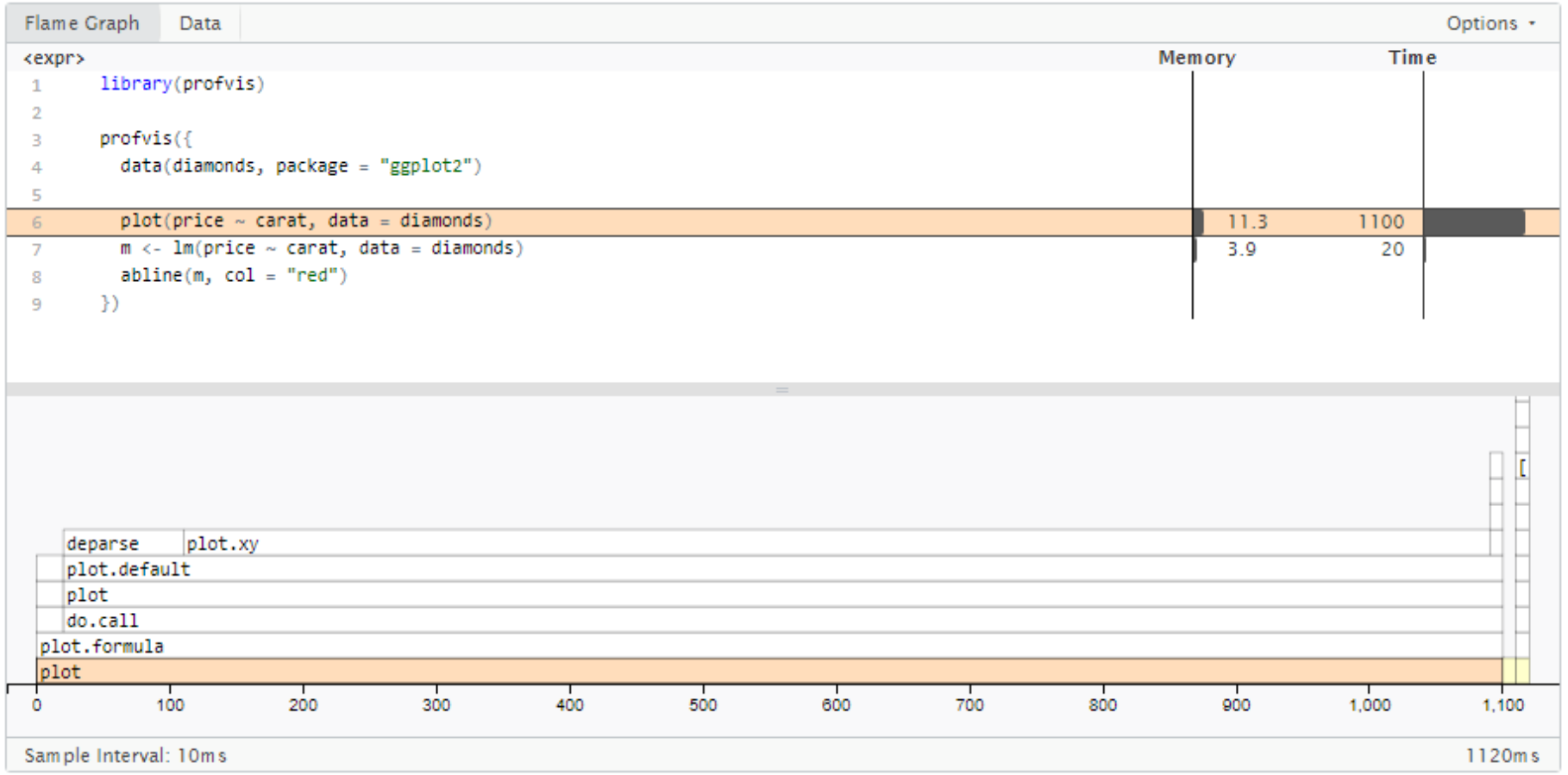
- Built into RStudio IDE
- Records app performance over time
- Allows you to find your pain points
- NOTE: Only optimize if needed

Profiling: How to

- Profile > Start Profiling
  - This can be done before or during the Shiny app runtime
- Take actions in the Shiny app to make the server do some work
- Profile > Stop Profiling



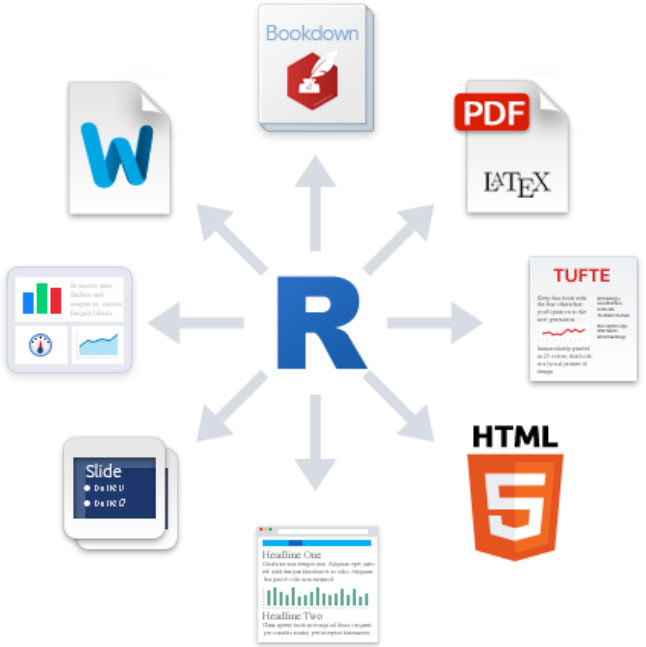
Results: flame graph



Automated reports

What are automated reports in R?

- Creating a snapshot of your information
- Different outputs
  - HTML document
  - PDF document
  - Word document
  - Interactive documents
- Can be generated different ways
  - **Quarto (.Qmd)**
  - R Markdown (.Rmd)
  - LaTeX (.Rnw)



Creating automated reports

- **Prepare** data / visualisations in R and Shiny
- **Construct** report structure in Quarto (.Qmd) or LaTeX (.Rnw)
- **Embed** data and visualisations into report structure
- **Render** the reports to desired output (PDF / HTML / Word)
- ***Link*** *to Shiny application to produce the reports on demand*

R Markdown & Quarto vs LaTeX

R Markdown & Quarto:

- Simpler syntax
  - Define content in plain text
  - Markdown formatting is simple and minimal
- Less control over final report appearance
  - Consistency not guaranteed across environments
- Quick reporting

LaTeX:

- Complex syntax
  - Define content in LaTeX code
  - Formatting requires nesting of LaTeX functions, options etc
- More control over final report appearance
  - Consistency guaranteed across environments
- Polished reporting

.Rmd (R Markdown)

- R flavoured Markdown documents
  - Formatted plain text
- Inline R code:
  - *The date today is: `r `sys.Date()`.*
- Code blocks:
  - ```` ```{r section_name echo = FALSE} ````
    ```
    dst <-  rnorm(1:100)
    hist(dst)
    ```
  - Settings

```
_italic_    **bold**

# Header 1

## Header 2

*** (h rule)

[link](url)

![alt text][image]

* List (Unordered)
    + Item
```

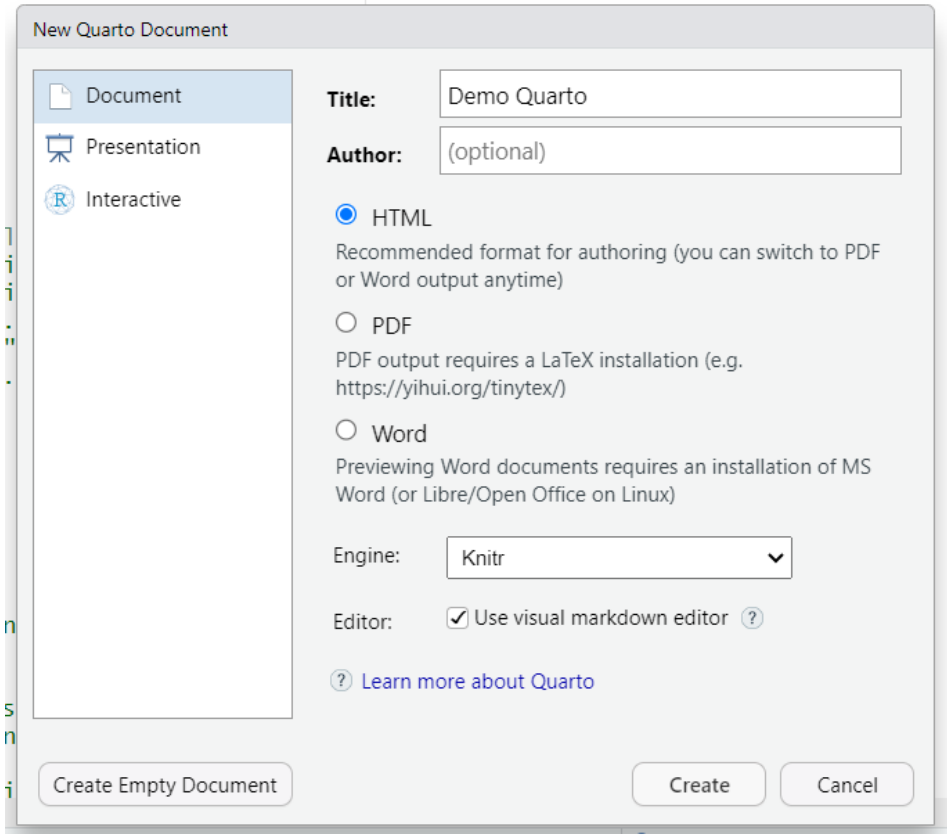
Quarto

- Released by Posit in July 2022 as a *successor* to R Markdown
- ***Backwards compatible with R Markdown***
- Integration with popular report features
  - Bookdown, blogdown, officedown
- Multiple languages supported
  - R, Python, JavaScript, Julia

https://quarto.org/docs/guide/

Quarto document types

- Different doc types available in Quarto
- Reports
  - HTML
  - PDF
  - Word
- Presentation – RevealJS
- Interactive – Shiny documents



.Qmd (Quarto) format

- Variant on original .Rmd format
  - Formatted plain text
- Inline R code:
  - *The date today is:* `` `r sys.Date()` ``
- Code blocks:
  - ```` ```{r section_name}
    #| echo: false
    dst <- rnorm(1:100)
    hist(dst)
    ``` ````
- Basics

```
*italic*    **bold**

# Header 1
## Header 2

*** (h rule)

[link](url)

![alt text][image]

* List (Unordered)
    + Item
1. List (Ordered)
    + Item
```
```

Anatomy of a Quarto document

```
---
title: "Demo Quarto"
format: html
editor: visual
---

## Quarto

Quarto enables you to weave together content and executable code into a finished document. To learn more about Quarto see <https://quarto.org>.

## Running Code

When you click the **Render** button a document will be generated that includes both content and the output of embedded code. You can embed code like this:

```{r}
1 + 1
```

You can add options to executable code like this

```{r}
#| echo: false
2 * 2
```

The `echo: false` option disables the printing of code (only output is displayed).
```

Quarto code blocks

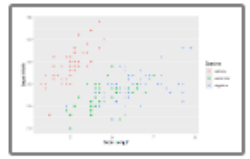
- Contained blocks of code within a Quarto document
- Evaluated when the document is rendered
- Can also be manually evaluated in the document, like a regular R script!
- Can handle a variety of languages, such as:
  - R
  - Python
  - JavaScript
  - Julia
- Can specify options to change the behaviour

Quarto code blocks

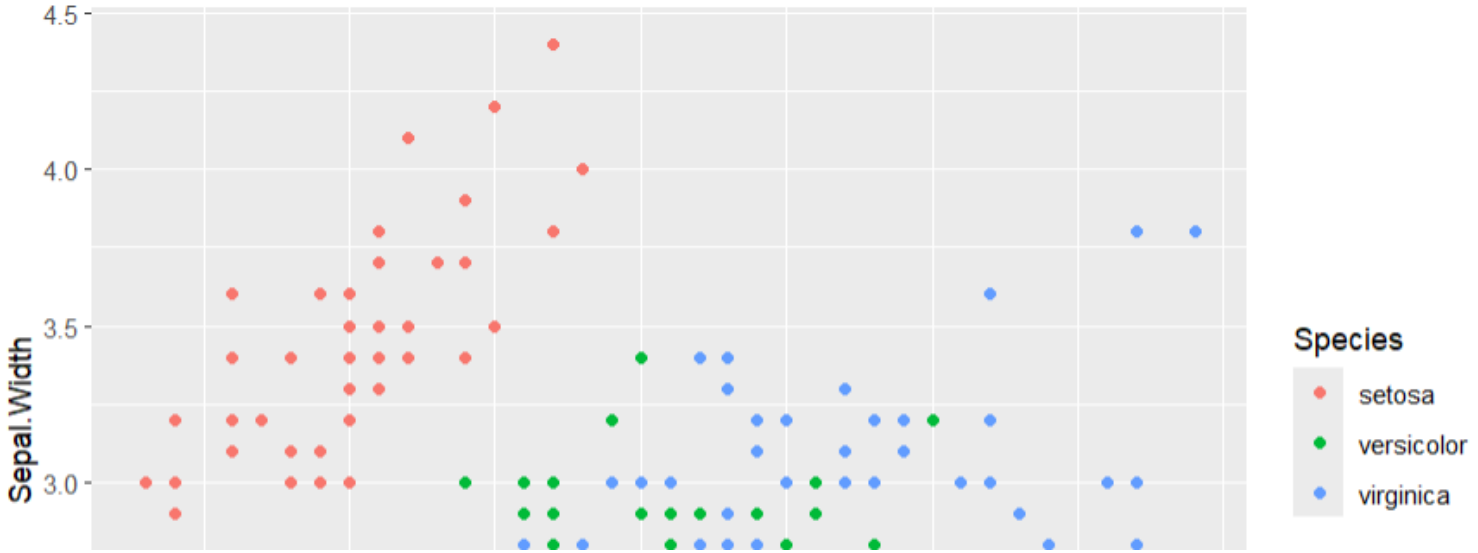
````
```{r}
#| echo: false

# create a ggplot2 chart for the iris dataset
library(ggplot2)
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, color = Species)) +
  geom_point()

```
````



R Console



Quarto options

Option	Description
`eval`	Evaluate the code chunk (if `false`, just echos the code into the output).
`echo`	Include the source code in output
`output`	Include the results of executing the code in the output (`true`, `false`, or `asis` to indicate that the output is raw markdown and should not have any of Quarto's standard enclosing markdown).
`warning`	Include warnings in the output.
`error`	Include errors in the output (note that this implies that errors executing code will not halt processing of the document).
`include`	Catch all for preventing any output (code or results) from being included (e.g. `include: false` suppresses all output from the code block).

Shiny elements in Quarto

Plain text / HTML / report structure

- Create directly in Quarto Markdown syntax
- From Shiny, in a code block

Tables

- DT
- Kable / KableExtra
- Print directly to report*
- From Shiny, in a code block

Charts / Maps

- Ggplot2
- Plotly
- Leaflet
- From Shiny, in a code block

DRY principles with Quarto

- We should avoid re-writing code we have already written!
- Use functions or reactives to easily replicate outputs in Quarto
- Use booleans to create print / dashboard specific variations

shared utility function:

```
generate_plot <- function(dat) {
        plotly(...)
}
```

server reference:

```
output$chart <- renderPlot({
        generate_plot(chartData())
})
```

qmd reference:

```
``` {r chart}
#| echo: false
generate_plot(dat)
```
```

quarto_render

- Function from the quarto package, used to generate reports from source .Qmd or .Rmd files
- Supports multiple output formats (HTML, PDF, Word)
- Can provide parameters to report to customize output

```
quarto::quarto_render(input = "Code/ShinyReport.Qmd",
      envir = new.env(), output_file = “ShinyReport.pdf",
     output_format = “pdf”
)
```

quarto::quarto_render

- Quarto render stores output locally at ‘output_file’

```
output$export <- downloadHandler(
filename = function() {return("Masterclass_Quarto_Report.pdf")},
content = function(file) {
   quarto::quarto_render(input = "report.Qmd",
      envir = new.env(), output_file = "report.pdf",
      output_format = "pdf"
   )
```

- Requires copy to return in download handler

```
  file.copy("report.pdf", file)
}
```

Parameterise Quarto - downloadHandler

- Passed through with execute_params

```
quarto::quarto_render(input = "report.Qmd",
      envir = new.env(), output_file = "report.pdf",
      output_format = "pdf",
      execute_params = list("course" = "Masterclass", "session" = 6)
)
```

Parameterise Quarto – QMD file

- Setting up quarto file with defaults

```
---
title: "R Shiny Advanced Masterclass"
author: "Epi-interactive"
date: "12/11/2024"
output: pdf_document
always_allow_html: true
params:
      course: NA
      session: NA
---
```

- Referencing in quarto file

````
### Course: `r params$course`


```{r}
#| echo: false
session <- params$session
[…]
```
````

Automating reports with Shiny

- Can integrate the report generation process to our Shiny apps
- Download button in UI
- downloadHandler in Server:
  - Quarto or R Markdown: render(…)
  - LaTeX: knit2pdf(…)

- Create reports on demand, directly from Shiny application data
- Use shiny inputs, reactive values and functions to populate report content
- Conditional report structure / formatting

Exercise

Using /stage1:

- Create a module for our quarto download, which contains a download button and a downloadHandler. Call this module directly in ui.R and server.R.
- This downloadHandler should use quarto_render to create a PDF report, using *report.Qmd* as a template
- Set up the report.Qmd file and the call to quarto_render to include parameters for ***filters, world_data*** and ***temp_data***
- To the report, add the selected filters using inline R code with markdown formatting.
- To the report, add the subregion chart, temperature chart & table, using the functions in util.R. Use quarto code blocks to do this, and use markdown formatting to give each section a title and subtitle.

Exercise example

R Shiny Advanced Masterclass

Epi-interactive

2024-03-21

Session 6

This exercise will allow you to practise the use of Quarto, integrating Quarto with R / R Shiny code, and exporting a PDF from an Quarto Markdown document through a Shiny application.

In separate paragraphs below, please insert the Quarto Markdown text described in the exercise. Feel free to include more additions to this content:

Today's date: 2024-03-21

Selected filters

- **Region:** All
- **Sub region:** All
- **Countries:** United States, Uzbekistan, Indonesia

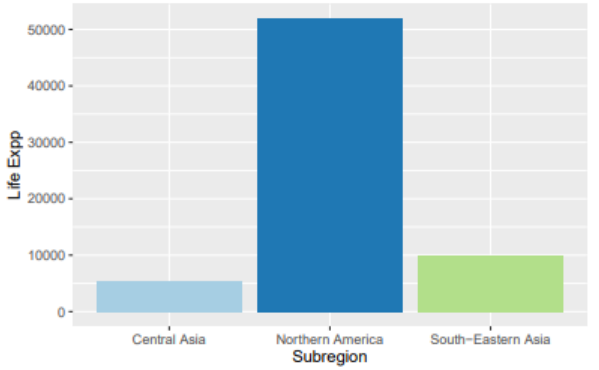
Unfiltered data

- Number of rows: 176
- Total Area: 135026868.14 sq. km
- Total Population: 7150238285.00

Filtered data

- Number of rows: 3
- Total Area: 11791405.26 sq. km
- Total Population: 604511340.00

Subregion chart



Country table

Region	Subregion	Country	Life Expectancy	GDP Per Capita	Life Exp Status
Americas	Northern America	United States	78.84146	51921.985	TRUE
Asia	Central Asia	Uzbekistan	71.03900	5370.866	TRUE
Asia	South-Eastern Asia	Indonesia	68.85600	10003.089	TRUE

Summary: Quarto

Things to watch out for:

- Interactive visualisations (Plotly, leaflet) not supported in .docx or .pdf by default
  - Instead, make an image from these, then use the image in the report
- R Code blocks:
  - Give code blocks unique identifiers
  - Make sure to handle settings (#| echo: false)

Image downloads

Recall: downloadHandler

Getting data back out of the application

To *ui.R*:

```
downloadLink(outputId = “download”, label = “Download”)
```

To *server.R*:

```
output$download <- downloadHandler(
        filename <- function() {
                ...
        },
        content = function(file) {
                write.csv(data, file)
        })
```

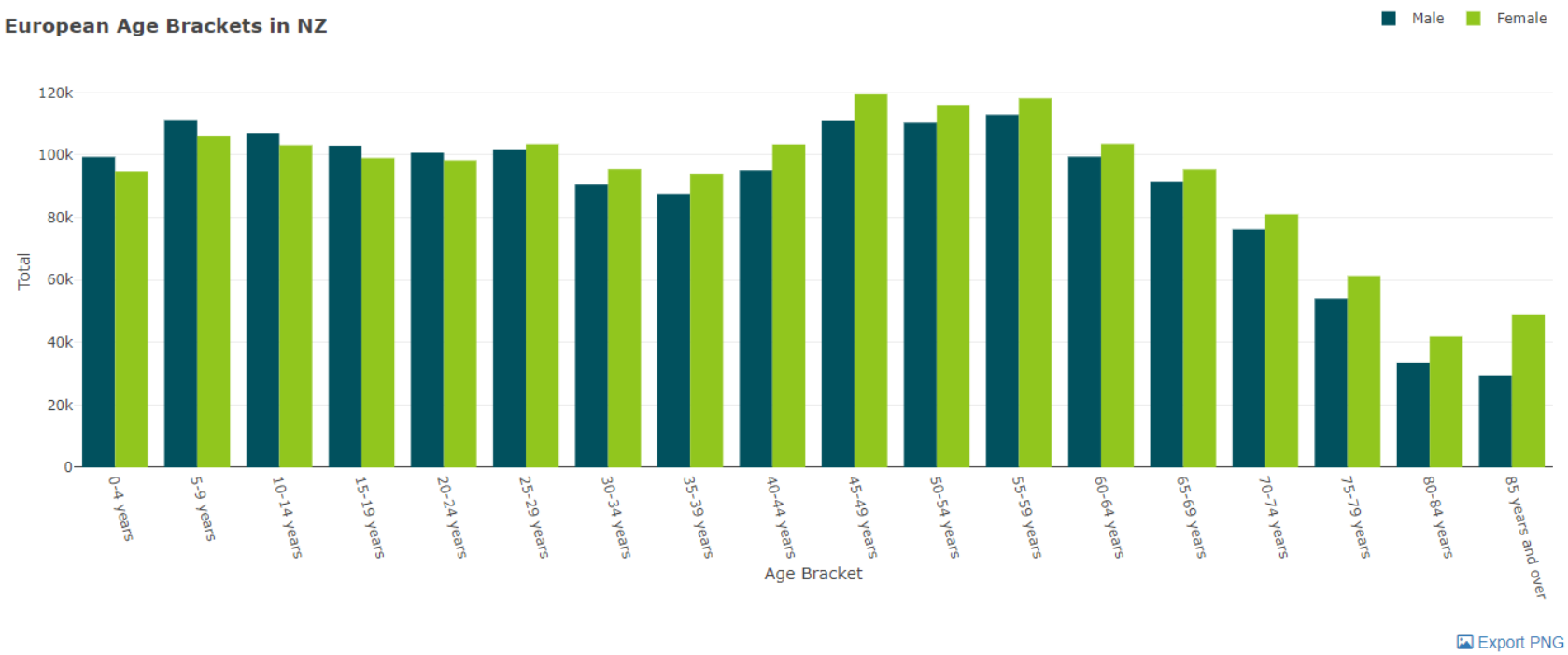
The easy way – with shinyscreenshot

- Developed and maintained by Dean Attali
- Screenshot whole page of a Shiny app, or specific parts of a Shiny app in its *current* state ***(as is)***
- https://github.com/daattali/shinyscreenshot

actionButton("go", "Take screenshot")

observeEvent(input$go, { screenshot(…) })

Image download



https://rshiny2.epi-interactive.com/apps/image_downloader/ (https://epi-interactive.github.io/)

EPI-interactive

Image download

To *ui.R*:

```
downloadLink(outputId = “export_image_download”, label = “Download”)
```

To *server.R*:

```
output$export_image_download <- downloadHandler(
        filename <- function() {
                g_getImageFileName("image_download")
        },
        content = function(file) {
                heading <- "Populations of NZ - Age/Sex distribution“
                g_getImageDownloader(getChart(), file, heading)
        },
        contentType = “image/png”
)
```

https://rshiny2.epi-interactive.com/apps/image_downloader/ (https://epi-interactive.github.io/)

Image download - getImageDownloader

```
g_getImageDownloader <- function(data, file, title ="", width=800, height=450){
   header <- […]
   footer <- […]

   orca(data, "temp.png", width = width, height = height)

   plot <- image_read("temp.png")
   allComponents <- c(header,  plot, footer)

   allComponents <- image_append(allComponents, stack=T)
   image_write(allComponents, file, format = "png")
}
```

Generating surrounding image

Generate file locally

Read locally generated file

Append components and return file

https://rshiny2.epi-interactive.com/apps/image_downloader/ (https://epi-interactive.github.io/)

EPI-interactive

Next time

- User authentication discussion
- Extended exercise

Challenge:

- Create a function in util.R to create the table in table-module.R, that takes in some filtered_data and returns a DT table. Add this to the report.
- Create a manual legend for the temperature chart
- Re-create the data summaries from the table module in the report.
- Modify your quarto module to have a selectInput to choose the file type – one of ‘PDF’, ‘HTML’ or ‘Word’.
- Change the file type being rendered depending on this user selection.