Shiny Dashboard

**Team:**

D10

**Reviewers:**

Lloyd & David

**Positives**

Overall: Really good effort in delivering a shiny dashboard that investigated the effects of drug abuse and smoking on life-expectancy across Scotland. Your team succeeded in making a highly interactive application that incorporated knowledge outwith the taught CodeClan subjects. The incorporation of CSS in your code enables interactive design elements that are coherent and well executed. There was a good attempt to create modular code and facilitate working in a large group. However the extent of the modularity in the final product was excessive and actually hindered the readability, maintainability, and adaptability of your app . Nonetheless, it was a solid effort by all team members in creating a working app which fulfilled the project brief.

You delivered an excellent, informative and engaging presentation. Your presentation was well organised and demonstrated your finished app as well as good insights into the data.

Global.R

It was good to see you include a global.R file but the contents seem to be in helper.R, which isn’t standard practice.

Data Wrangling:

Best practice all round. Good use of code sections. Well written code that was easy to understand. By developing a deep understanding of the datasets you were able to cut down on the amount of wrangling required in the server.

Server.R:

You made good use of inline comments to improve readability in every part of your server code. Your use of breakpoints was also consistent and helped to break up sections of code.

Having the select inputs update depending on the dataset resulted in an easy to use interface.

This was an inventive way of splitting up your server code into chunks, allowing each of you to focus on a particular section before combining into one app. Including a blame at the top of each file was useful.

The functions you wrote were well-defined and included very little hard-coding.

It’s obvious from the amount of wrangling or lack of in the server files that you understood your data and got it into a consistent form before loading it into your app.

Use of if else flow to improve app speed was good. Your app works without issue which is impressive given the scope and complexity. You demonstrated advanced knowledge of shiny and data visualisation which resulted in an excellent final product.

UI.R:

The UI was laid out in a way that was easy to interact with and the resulting plots displayed the information well. The map looked excellent on every screen size I tested.

Having a landing page was nice - giving the user a space where they can see clearly all the sections of your app.

Splitting up the ui file into separate files broke up your app into more manageable chunks, and including a blame in your ui.R was a good idea for working collaboratively. UI code was very good, using a mix of html and css as well. The next step would be to combine your separate ui files into one final ui.R and maybe create a dedicated CSS file.

Code was written in accordance with best practices, was DRY, easy to read, and (aside from the hyper-modularity) very well structured.

**Potential Improvements**

Presentation:

Was very good, met all expectations. The only criticism I have is that it was maybe on the long side. That being said, the length felt appropriate and gave everyone an opportunity to speak, which you did very well.

App:

Although the design of the charts and graphs conveyed the data very clearly, the dashboard lacked a clear reasoning of why the analysis was conducted. Have a little introduction and state your research question to set the context. Furthermore, an absence in the interpretation and findings of the presented data required the audience looking at the dashboard to make their own conclusions. A little paragraph describing the data and a couple sentences stating your conclusions from your data analysis would have complemented the dashboard. Keeping in mind, together as a team, you have spent a lot of time looking at this data and you would understand it better than anyone looking at the dashboard. Try including a summary conclusion to help solidify and consolidate your main conclusions from your analysis. It will help link the three different data analysis you have conducted and make the presented data much more cohesive. Tell me a story with your data.

Global.R:

The modular nature of your code in your application had the unintended effect of making the code difficult to understand. global.R is where I’d expect to see what you currently have in the file helper.R. global.R is used for loading libraries, defining functions, and reading data that are necessary for your app. If you found this code was becoming extensive - perhaps because you’ve defined a lot of functions - then it would be a call to place the functions in a separate R script file and source them into your global.R where your app can then ‘see’ them and use them. Unnesting all your code into a single script or a few scripts will make it much easier review and minimise redundant code; libraries such as here, tidyverse and janitor are called multiple times across several .R scripts (datazonelookup.R, life\_expectancy.R, helper.R..etc). You run the risk of adding unused code into your application.

In standard practice we would expect to see a structure similar to:

App folder

* R (a folder called ‘R’ where you store R scripts containing helper functions - functions called by your app multiple times)
  + plotting\_functions.R (e.g.)
  + filtering\_functions.R (e.g.)
* global.R (where you load all of the necessary libraries, source files in R/, and read in your data)
* server.R
* ui.R

Data Wrangling:

smoking.R - Line 63-65: try and remove commented code when deploying. Other than that, everything else was very good.

Server.R:

Didn’t always follow best coding-practices: e.g. not indenting, adding unnecessary {s, not formatting code and removing white space. On the very few occasions where best practice wasn’t followed, code became more difficult to read.

In many sections old/obsolete/broken code was left commented out. These should be removed/moved to a development branch to be corrected.

So much of the code is functions nested within functions and often the deepest layer would include a similar control flow expression: if dataset 1 return dataset 1, else if dataset 2 return dataset 2 else return dataset 3. An alternative strategy could be to this process once.

Some server code doesn’t follow DRY principles - lots of the plot objects are generated in a very similar way (often there’s only one variable changed).

Some if else control flow was repetitive, I wonder if there is a way to perform the which\_dataset test initially and then have everything flow from there, rather than perform the test in multiple functions.

It’s best practice to include return statements for long functions. In other languages, like Python, functions won’t work without a return statement, and it makes it clear what the function’s output is.

The file server\_rank.R seems redundant. It’s not called anywhere and there are functions with the same names in other files.

Nested functions can make your code difficult to understand and trace back errors. If I examine the object ‘filtered\_rank\_data’ in the server.R file, I see that this was created with the function ‘select\_rank\_data’, which is located in a different file: server\_helper.R, but this function uses another function: select\_life\_rank\_data which is located in a different file altogether.

While I think this was a very creative solution (effectively to sidestep merge conflicts), the final app being split up into so many files makes it very difficult to read, debug, maintain, and adapt. If the client returned next year and wanted you to extend the functionality of your app to include other datasets for example, this would be difficult to implement. Because of the nested nature of the functions, it can be hard to trace back an issue to its initial cause, and issues/new errors are common given R is open source and updated on a regular basis. All this being said, you produced an excellent app, which fulfilled the brief, and managed to do so working collaboratively.

A good way to think about modularity and what should be a function is to look at your outputs. I.e. you create 2 different types of plot outputs - one for rank and one for trend, so I’d expect 2 functions - make\_trend\_plot(), make\_rank\_plot(), but instead there are 6 functions as each category smoking, drugs, and life expectancy has its own function for each type of plot, even though the x axis is always a time variable, the y axis the user’s chosen category (for trend).

UI.R :

The app was nice and easy to interact with. The following potential improvements are mostly quality of life and to improve the maintainability of the project.

The data table output for the Rank tab could be shortened so that it wasn’t leaving the boundary. Some tabs wouldn’t format well depending on the screen size - but for the default screen size they were completely fine. It was good to see a legend for the spatial plot, an extra step would be to alter the transparency to allow the user to view any portion of the map covered by the legend. The legend itself isn’t entirely clear - at the moment it shows the names of polygons; what would be more useful perhaps would be to show which bin of values that colour was associated with.

While splitting up the ui file, did break up your code into more manageable chunks and probably mitigated several merge conflicts, this could be done without the need for several files - you could have put them in one file, which not only is standard practice, it’s easier to read, debug and maintain.

For css this large, it might be worthwhile putting it into a dedicated .css file and sourcing it in: https://shiny.rstudio.com/articles/css.html

Instead of commenting large chunks of obsolete code, correct them, remove them, or move them to a development branch.

Large sections of quoted HTML code can be difficult to read and debug. An alternative would be to use the tags() function and select the appropriate HTML tag. Then style etc. can be passed as arguments.