Exercise: shiny apps

Prof. Dr. Tillmann Schwörer

Summer Term 2020

1 Fork and clone repo

Fork the repository . . . to your GitHub account, then clone to your computer. The repo contains 5 enumerated shiny apps in increasing complexity. Once you have installed the required packages (`shiny`, etc.), you can run each of those apps from the RStudio IDE via **Run App**. While you solve the exercise, you can always compare, how particular problems are solved in these shiny apps.

2 Create a shiny app

Before you start working, create a feature branch. Then create a shiny app via File > New File > Shiny Web App in the RStudio IDE. Select **Multiple File App**, select the **directory** in which you want to store your shiny app (the exercise folder), and select an **application name**. Note: If you want to get feedback on your app (via a pull request to the upstream repo), then please use a characteristic app name that others won't use (e.g. your-github-name-app1). You now have a runnable default shiny app that uses the built-in `faithful` data set. Use `help(faitfhul)` to get information about the data.

3 Fokus on communication between server and UI

Before we care about creating beautiful apps, we should first understand shiny's core functionality. So, get rid of all design functions (sidebarLayout, sidebarPanel, . . .) that are not strictly required for the app to work. Make sure that you delete both the opening and closing parentheses. Do the cleaning iteratively, and try wether the app is still working. After the cleaning the user interface should contain little more than input and output functions. Also, delete all the comments (marked with `#`). Your final app should fit into as few lines of code as possible.

Reflect on how server and UI communicate. If a user moves the slider what is the sequence of actions that is happening under the hood?

4 Shiny app with ggplot histogram

The app uses the base R plot function `hist`. Rewrite your app such that it uses ggplot2 syntax. In a first step your app should be entirely static. In a second step, you can turn the number of bins into an input. In a third step, you can let the user choose the variable for which the histogram is drawn. Remember that most of these changes, will require adjustments in both the user interfce (add another input widget) and the server (replace the fixed input by input$. . .).

Note that you must load the required libraries (`tidyverse`, etc.) within your app. For the moment it may be fine, to do this in the `server.R` script. But if your app becomes more complex, a better place would be a separate file `global.R`, because verything that is defined in global.R will be available both to the server and the UI.

5 From static analysis to interactive shiny app

Turn the following code into a shiny app. You can either create a new app, or add it to your existing one. Again, start with a static app. Then, step by step, turn the filtervalue (2.5), the x-axis, the y-axis, and the color mapping into a variabe. Verify after each step whether the app is working.

```
faithful %>%
  filter(eruptions > 2.5) %>%
  ggplot(aes(x = eruptions, y = waiting, color = eruptions)) +
  geom_point() +
  geom_smooth()
```

6 Add another Output

We would like to see not only the plot, but also the filtered data itself. Therefore, add a `renderTable()` block to your server and a `tableOutput` to your user interface, and try to get the app running.

Abetter alternative for printing data frames in a shiny app comes with the package `DT`. Install the package, load it (either in both server and ui, or in a separate global.R file), and replace `tableOutput` by `DTOutput` and `renderTable` by `renderDT`.

7 Pull Request

To get feedback on your exercise, push to GitHub and open a pull request to the upstream repo. Or ask via Slack. Not only if your app is finished, but also if you are stuck!