DESE71006
Tutorial 2: Data visualisation in practice

The aim of this tutorial session is to look at how to produce more advanced visualisations with R, and also to move forward with R Shiny. For all nice resources related to R, R Shiny and visualisations in R, please go back to all the links provided in Tutorial 1.

Before to start, remember to make sure to set your working directory, either in R Studio through the files pane (bottom right quadrant), or by using the `setwd` in the R console directly.

Remember that in R, to use packages, your first need to install them (only once) with

```
install.packages("packagename")
```

And then before to use it in the code, you load it with

```
library(packagename)
```

We will need to install and use quite a number of packages in this Tutorial.

Activity 1: Visualising the population of England

Let us consider the population of England, then divided per region and per county. We want to find a good visualisation for it and implement it with R. That will also involve a bit of data scraping. For instance, the input data is available at en.wikipedia.org/wiki/Counties_in_England_by_population

Step 1. What kind of variables are involved here? Browse through data-to-viz.com to explore relevant visualisations, based on the variables involved and the setup.

Step 2. We need to first scrape the data from that wikipedia page. There are so many resources online, that you can just google "R scrape data from a wikipedia page" and you will find a page that will tell you how. Here, let us use this page: r-bloggers.com/2021/07/politely-scraping-wikipedia-tables-2.

Write the necessary code to scrape the data from that wikipedia page (and specific table, which is at node "table.wikitable" in the page), and place it in a data frame called `mydata`.

Step 3. In the above step, what we did was raw data scraping. However, there may always be some issues with that raw data (hence, the need for data curation). Here, the 3 issues are that

- the column `total_population` is of type `character` while we need it to be `numeric`
- the use of commas for separating thousands and millions is not consistent with the way R sees numbers
- there is a mistake in the table for North Yorkshire, as the region should be "Yorkshire and the Humber" only.

For the first point, there is the function `as.numeric` that can be used. For instance with

```
mydata$total_population <- as.numeric(mydata$total_population)
```

For the second point, we need to reformat the data, before conversion. This can be done with `gsub` (converting ',' to nothing). The final line (combining both points) would look like

```
mydata$total_population <- as.numeric(gsub(",", "", mydata$total_population))
```

Finally, for the small mistake, you can just correct it a posteriori with

```
mydata$region[15] <- mydata$region[10]
```

Our data is now ready and available in `mydata`.

Step 4. For populations and such a hierarchy, treemaps work perfectly. Look at this page to see how to produce treemaps with groups (here, regions) and subgroups (here, counties): r-graph-gallery.com/235-treemap-with-subgroups.html.

Here, our data frame can be used directly! You first need to prepare the dataset with

```
group <- mydata$region
subgroup <- mydata$county
value <- mydata$total_population
data <- data.frame(group,subgroup,value)
```

And then, you can use the treemap function directly.

Step 5. Look at the function documentation (with `?treemap`) to see your options to make the treemap look nicer (e.g., change text color and size, the algorithm to produce the treemap, the colour and width of border lines, etc.). Eventually, save it as a png file.

Step 6. Alternatively, we could produce an interactive circular packing chart. Go to this page to see how to do it: r-graph-gallery.com/338-interactive-circle-packing-with-circlepacker.

There seems to be some issues to install the `circlepackeR` package... Please use these command lines:

```
install.packages("devtools")
devtools::install_github("jeromefroe/circlepackeR")
```

Note that, in contrast to their example, your dataset is already ready! What you are left to do is to change the format of your data to have a data tree, and then to plot.

Eventually, you can play with the colouring scheme and save it as an html widget.

Activity 2: Visualising summary statistics for weather data

As a basis for this activity, we will continue using weather data from the Met Office. For simplicity, we use a starting point the Heathrow weather data from Tutorial 1, originally retrieved from metoffice.gov.uk/pub/data/weather/uk/climate/stationdata/heathrowdata.txt

Step 1. Get the data from the following url:
http://pierrepinson.com/wp-content/uploads/2023/10/Heathrow-weather-data-1948-2022.csv. Make sure you get it into a data frame, for instance called `WeatherData`. This is something you have already done in Tutorial 1.

Step 2. Let us focus on temperature for instance (`tmin` and `tmax`). Extract these 2 vectors and reshape them to become matrices where each line is for a month, and each column is for a year.

Step 3. In Tutorial 1, you got to calculate the average for each month (i.e., mean per row of the matrix). Look at the documentation for the `sd` function to also calculate the standard deviation

for each of them. Apply the function to the first line of your matrix for instance to calculate the standard deviation for January.

Then, look at the function `apply` to see how to readily calculate the standard deviation for all rows at once.

How does the standard deviation evolve throughout the year (for both `tmin` and `tmax`)? What does that mean?

Step 4. Now, please look at the `quantile` function to also calculate standard deviation and quantiles for each month. In terms of quantiles, get the quantiles with nominal proportions 0.05 and 0.95 (in other words, 5% and 95%).

Step 5. In terms of visualisation, it could be nice to make a plot that shows the average temperatures (done in Tutorial 1 already), but now also adding a shaded area, where the lower limit is given by the quantiles with nominal proportion 0.05, and the upper limit is given by the quantiles with nominal proportion 0.95. For that, look at the `polygon` function.

Hint 1 : To draw the polygon, you will most likely need to reverse the elements of one of your quantile vectors. This can be done with the `rev` function.

Hint 2: To have more control on colour and transparency for the shaded area, for the colour in `polygon` use `col=rgb(0,0,1,0.2)` (gives blue with some transparency) and `col=rgb(1,0,0,0.2)` (gives red with some transparency).

What is the meaning of that shaded area (representing 90% coverage intervals)?

Activity 3: Moving on with R Shiny

Go through the activities and exercises at the following links (focus on the server block, and reactivity): You will need to focus on: mastering-shiny.org/basic-reactivity.html

If you have time, I would advise you to also look at (not mandatory, but could be useful in the future):

- Layouts and themes – mastering-shiny.org/action-layout.html
- Graphics – mastering-shiny.org/action-graphics.html
- Uploads and downloads – mastering-shiny.org/action-transfer.html

Activity 4: A dashboard for the weather data statistics

Produce a dashboard requiring 2 inputs from the user:

- a file to be uploaded (file input)
- the coverage of the intervals to be plotted (using a slider)

Based on these inputs, the dashboard produces a plot like that of Acivity 2, showing the climatology for minimum and maximum temperatures, with intervals that have the coverage chosen by the user. Let's go through it step by step – we will start from a basic version and then develop it further.

Step 1. We start by producing a simple App allowing to upload a csv data file. As this is not the most interesting part (and you can find source code for that online). Let me provide it for you here (while you can download the file from blackboard learn - "app-weather-statistics-v1.R", in the Tutorial 2 folder):

```
library(shiny)

ui <- fluidPage(
  fileInput("upload", NULL, buttonLabel = "Upload...", multiple = FALSE,
                          accept = c(".csv")),
  verbatimTextOutput("code")
  )

server <- function(input, output, session) {

  data <- reactive({
    req(input$upload)
    ext <- tools::file_ext(input$upload$name)
    switch(ext,
           csv = vroom::vroom(input$upload$datapath, delim = ",",
                                                  show_col_types = FALSE),
           validate("Invalid␣file;␣Please␣upload␣a␣.csv␣file")
    )
  })

  output$code <- renderPrint({
    summary(data())
  })
}

shinyApp(ui = ui, server = server)
```

Please try it out. When you upload the Heathrow data, it automatically provides you with a summary of the data.

Step 2. We should always minimise the amount of actual coding within the definition of Shiny Apps, and hence maximise the use of functions to perform calculations. Functions are defined just after loading libraries, and right above the ui block. An example function (called `computeaverage`) calculating the average between variables `tmin` and `tmax` in a data frame would look like

```
computeaverage <- function(data) {
  data.average <- (data$tmin + data$tmax)/2
  return(data.average)
}
```

In our case, if aiming to have a function to calculate the temperature averages, this would look like

```
computeTmeans <- function(mdata) {
  tmin.mat <- mdata$tmin
  tmax.mat <- mdata$tmax
  dim(tmin.mat) <- c(12, length(tmin.mat)/12)
  dim(tmax.mat) <- c(12, length(tmax.mat)/12)
  tmin.mean <- rowMeans(tmin.mat)
  tmax.mean <- rowMeans(tmax.mat)
  yy <- data.frame(tmin = tmin.mean, tmax = tmax.mean)
  return(yy)
}
```

Add this function to your R file. Then, what lines of code do you need to have in your server block to render a summary for the `tmin` and `tmax` variables?

Step 3. Now, instead of rendering a text output with the summary of the variables, produce a plot (as in Tutorial 1) that shows the average `tmin` and `tmax` as a function of the month of the year.

Step 4. We focus here on giving the choice to the user for the coverage of the intervals to be plotted. Hence, first add a slider to the ui block, with a range between 10 and 100, and with steps of 10 (with default value of 90). In parallel, add another function that allows computing quantiles for the `tmin` and `tmax` variables of our data frame. Finally, in the server block, upgrade the plotting part to also plot the intervals.

You now can play with the app to see what the plot looks like when changing the coverage of the intervals.

Step 5. (optional) Find ways to improve the layout of the UI (e.g., with columns), to add a title, etc.