*Florianne Verkroost is a PhD candidate at Nuffield College at the University of Oxford. With a passion for data science and a background in mathematics and econometrics. She applies her interdisciplinary knowledge to computationally address societal problems of inequality.*

In this post, We will show you how to create interactive world maps and how to show these in the form of an R Shiny app. As the Shiny app cannot be embedded into this blog, I will direct you to the [live app](https://fverkroost.shinyapps.io/interactive_worldmap_app/) and show you in [this post](https://github.com/fverkroost/RStudio-Blogs/blob/master/interactive_worldmap_shiny_embedded.Rmd) on my GitHub how to embed a Shiny app in your R Markdown files, which is a really cool and innovative way of preparing interactive documents. To show you how to adapt the interface of the app to the choices of the users, we’ll make use of two data sources such that the user can choose what data they want to explore, and that the app adapts the possible input choices to the users’ previous choices. The data sources here are about childlessness and gender inequality, which is the focus of my PhD research, in which I computationally analyse the effects of gender and parental status on socio-economic inequalities.

We’ll start by loading and cleaning the data, whereafter we will build our interactive world maps in R Shiny. Let’s first load the required packages into RStudio.

**Importing, exploring and cleaning the data**

Now, we can continue with loading our data. As we’ll make world maps, we need a way to map our data sets to geographical data containing coordinates (longitude and latitude). As different data sets have different formats for country names (e.g., “United Kingdom of Great Britain and Northern Ireland” versus “United Kingdom”), we’ll match country names to ISO3 codes to easily merge all data sets later on. Therefore, we first scrape an HTML table of country names, ISO3, ISO2 and UN codes for all countries worldwide. We use the rvest package using the XPath to indicate what part of the web page contains our table of interest. We use the pipe (%>%) from the magrittr package to feed our URL of interest into functions that read the HTML table using the XPath and convert that to a data frame in R. One can obtain the XPath by hovering over the HTML table in developer mode in the browser, and having it show the XPath.

The first element in the resulting list contains our table of interest, and as the first column is empty, we delete it. Also, as you can see from the HTML table in the link, there are some rows that show the letter of the alphabet before starting with a list of countries of which the name starts with that letter. As these rows contain the particular letter in all columns, we can delete these by deleting all rows for which all columns have equal values.

library(magrittr)

library(rvest)

url <- "<https://www.nationsonline.org/oneworld/country_code_list.htm>"

iso\_codes <- url %>%

read\_html() %>%

html\_nodes(xpath = '//\*[@id="CountryCode"]') %>%

html\_table()

iso\_codes <- iso\_codes[[1]][, -1]

iso\_codes <- iso\_codes[!apply(iso\_codes, 1, function(x){all(x == x[1])}), ]

names(iso\_codes) <- c("Country", "ISO2", "ISO3", "UN")

head(iso\_codes)

## Country ISO2 ISO3 UN

## 2 Afghanistan AF AFG 004

## 3 Aland Islands AX ALA 248

## 4 Albania AL ALB 008

## 5 Algeria DZ DZA 012

## 6 American Samoa AS ASM 016

## 7 Andorra AD AND 020

Next, we’ll collect our first data set, which is a data set on childlessness provided by the United Nations. We download the file from the link, save it locally, and then load it into RStudio using the read\_excel() function in the readxl package.

library(readxl)

url <- "<https://www.un.org/en/development/desa/population/publications/dataset/fertility/wfr2012/Data/Data_Sources/TABLE%20A.8.%20%20Percentage%20of%20childless%20women%20and%20women%20with%20parity%20three%20or%20higher.xlsx>"

destfile <- "dataset\_childlessness.xlsx"

download.file(url, destfile)

childlessness\_data <- read\_excel(destfile)

head(childlessness\_data)

## # A tibble: 6 x 17

## `United Nations… ...2 ...3 ...4 ...5 ...6 ...7 ...8 ...9 ...10

##

## 1 "TABLE A.8. PE…

## 2 Country ISO … Peri… Refe… Perc… Perc…

## 3 35-39 40-44 45-49 35-39 40-44 45-49

## 4 Afghanistan 4 Earl… .. .. .. .. .. .. ..

## 5 Afghanistan 4 Midd… .. .. .. .. .. .. ..

## 6 Afghanistan 4 Late… 2010 2.6 2.6 2.1 93.8 94.5 94

## # … with 7 more variables: ...11 , ...12 , ...13 ,

## # ...14 , ...15 , ...16 , ...17

We can see that the childlessness data are a bit messy, especially when it comes to the first couple of rows and column names. We only want to maintain the columns that have country names, periods, and childlessness estimates for different age groups, as well as the rows that refer to data for specific countries. The resulting data look much better. Note that when we convert the childlessness percentage columns to numeric type later on, the “..” values will automatically change to NA.

cols <- which(grepl("childless", childlessness\_data[2, ]))

childlessness\_data <- childlessness\_data[-c(1:3), c(1, 3, cols:(cols + 2))]

names(childlessness\_data) <- c("Country", "Period", "35-39", "40-44", "45-49")

head(childlessness\_data)

## # A tibble: 6 x 5

## Country Period `35-39` `40-44` `45-49`

##

## 1 Afghanistan Earlier .. .. ..

## 2 Afghanistan Middle .. .. ..

## 3 Afghanistan Latest 2.6 2.6 2.1

## 4 Albania Earlier 7.2 5.5 5.2

## 5 Albania Middle .. .. ..

## 6 Albania Latest 4.8 4.3 3.3

Our second data set is about measures of gender inequality, provided by the World Bank. We read this .csv file directly into RStudio from the URL link.

gender\_index\_data <- read.csv("<https://s3.amazonaws.com/datascope-ast-datasets-nov29/datasets/743/data.csv>")

head(gender\_index\_data)

## Country.ISO3 Country.Name Indicator.Id

## 1 AGO Angola 27959

## 2 AGO Angola 27960

## 3 AGO Angola 27961

## 4 AGO Angola 27962

## 5 AGO Angola 28158

## 6 AGO Angola 28159

## Indicator

## 1 Overall Global Gender Gap Index

## 2 Global Gender Gap Political Empowerment subindex

## 3 Global Gender Gap Political Empowerment subindex

## 4 Overall Global Gender Gap Index

## 5 Global Gender Gap Economic Participation and Opportunity Subindex

## 6 Global Gender Gap Economic Participation and Opportunity Subindex

## Subindicator.Type X2006 X2007 X2008 X2009 X2010 X2011

## 1 Index 0.6038 0.6034 0.6032 0.6353 0.6712 0.6624

## 2 Rank 81.0000 92.0000 103.0000 36.0000 24.0000 24.0000

## 3 Index 0.0696 0.0696 0.0711 0.2007 0.2901 0.2898

## 4 Rank 96.0000 110.0000 114.0000 106.0000 81.0000 87.0000

## 5 Rank 69.0000 87.0000 87.0000 96.0000 76.0000 96.0000

## 6 Index 0.5872 0.5851 0.5843 0.5832 0.6296 0.5937

## X2012 X2013 X2014 X2015 X2016 X2018

## 1 NA 0.6659 0.6311 0.637 0.643 0.633

## 2 NA 34.0000 38.0000 38.000 40.000 58.000

## 3 NA 0.2614 0.2402 0.251 0.251 0.206

## 4 NA 92.0000 121.0000 126.000 117.000 125.000

## 5 NA 92.0000 111.0000 116.000 120.000 113.000

## 6 NA 0.6163 0.5878 0.590 0.565 0.602

Luckily, these data are better structured than the childlessness data. The data contains gender inequality measures per year, and for convenience we add a new column with the values for the most recent year for which data are available. In this post, we’ll only look at the rank indicators rather than indices and normalized scores. We drop the Subindicator and IndicatorID columns using the select() function from the dplyr package, as we won’t need these further.

library(dplyr)

gender\_index\_data["RecentYear"] <- apply(gender\_index\_data, 1, function(x){as.numeric(x[max(which(![is.na](http://is.na)(x)))])})

gender\_index\_data <- gender\_index\_data[gender\_index\_data$Subindicator.Type == "Rank", ] %>%

select(-Subindicator.Type, -Indicator.Id)

names(gender\_index\_data) <- c("ISO3", "Country", "Indicator", as.character(c(2006:2016, 2018)), "RecentYear")

head(gender\_index\_data)

## ISO3 Country

## 2 AGO Angola

## 4 AGO Angola

## 5 AGO Angola

## 7 AGO Angola

## 9 AGO Angola

## 11 AGO Angola

## Indicator

## 2 Global Gender Gap Political Empowerment subindex

## 4 Overall Global Gender Gap Index

## 5 Global Gender Gap Economic Participation and Opportunity Subindex

## 7 Global Gender Gap Educational Attainment Subindex

## 9 Global Gender Gap Health and Survival Subindex

## 11 Wage equality between women and men for similar work (survey data, normalized on a 0-to-1 scale)

## 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2018 RecentYear

## 2 81 92 103 36 24 24 NA 34 38 38 40 58 58

## 4 96 110 114 106 81 87 NA 92 121 126 117 125 125

## 5 69 87 87 96 76 96 NA 92 111 116 120 113 113

## 7 107 119 122 127 125 126 NA 127 138 141 138 143 143

## 9 1 1 1 1 1 1 NA 1 61 1 1 1 1

## 11 NA NA NA NA NA NA NA NA NA NA 135 94 94

Next, we load in our world data with geographical coordinates directly from the ggplot2 package. These data contain geographical coordinates of all countries worldwide, which we’ll later need to plot the worldmaps.

library(maps)

library(ggplot2)

world\_data <- ggplot2::map\_data('world')

world\_data <- fortify(world\_data)

head(world\_data)

## long lat group order region subregion

## 1 -69.90 12.45 1 1 Aruba

## 2 -69.90 12.42 1 2 Aruba

## 3 -69.94 12.44 1 3 Aruba

## 4 -70.00 12.50 1 4 Aruba

## 5 -70.07 12.55 1 5 Aruba

## 6 -70.05 12.60 1 6 Aruba

To map our data, we need to merge the childlessness, gender gap index, and world map data. As I said before, these all have different notations for country names, which is why we’ll use the ISO3 codes. However, even between the ISO code data and the other data sets, there is discrepancy in country names. Unfortunately, to solve this, we need to manually change some country names in our data to match those in the ISO code data set. The code for doing so is long and tedious, so I won’t show that here, but for your reference you can find it [here](https://github.com/fverkroost/RStudio-Blogs/blob/master/interactive_worldmap_shiny_app.R).

Now that the name changes for countries have been made, we can add the ISO3 codes to our childlessness and world map data. The gender gap index data already contain these codes, so there’s no need for us to add these there.

childlessness\_data['ISO3'] <- iso\_codes$ISO3[match(childlessness\_data$Country, iso\_codes$Country)]

world\_data["ISO3"] <- iso\_codes$ISO3[match(world\_data$region, iso\_codes$Country)]

Next, we melt the childlessness and gender gap index data into long format such that they will have similar shape and column names for merging. The melt() function is included in package reshape2. The goal here is to create variables that have different unique values for the different data, such that I can show you how to adapt the R Shiny app input to the users’ choices. For example, we’ll create a *DataType* column that has value *Childlessness* for the rows of the childlessness data and value *Gender Gap Index* for all rows of the gender gap index data. We’ll also create a column *Period* that contains earlier, middle and later periods for the childlessness data, and different years for the gender gap index data. As such, when the user chooses to explore the childlessness data, the input for the period will only contain the choices relevant to the childlessness data (i.e., earlier, middle, and later periods and no years). When the user chooses to explore the gender gap index data, they will only see different years as choices for the input of the period, and not earlier, middle, and later periods. The same goes for the *Indicator* column. This may sound slightly vague at this point, but we’ll see this in practice later on when building the R Shiny app.

library(reshape2)

childlessness\_melt <- melt(childlessness\_data, id = c("Country", "ISO3", "Period"),

[variable.name](http://variable.name) = "Indicator", [value.name](http://value.name) = "Value")

childlessness\_melt$Value <- as.numeric(childlessness\_melt$Value)

gender\_index\_melt <- melt(gender\_index\_data, id = c("ISO3", "Country", "Indicator"),

[variable.name](http://variable.name) = "Period", [value.name](http://value.name) = "Value")

After melting the data and ensuring they’re in the same format, we merge them together using the rbind() function, which we can do here because the data have the same column names.

childlessness\_melt["DataType"] <- rep("Childlessness", nrow(childlessness\_melt))

gender\_index\_melt["DataType"] <- rep("Gender Gap Index", nrow(gender\_index\_melt))

df <- rbind(childlessness\_melt, gender\_index\_melt)

**Creating an interactive world map**

Next, it’s time to define the function that we’ll use for building our world maps. The inputs to this function are the merged data frame, the world data containing geographical coordinates, and the data type, period and indicator the user will select in the R Shiny app. We first define our own theme, my\_theme(), for setting the aesthetics of the plot. Next, we select only the data that the user has selected to view, resulting in *plotdf*. We keep only the rows for which the ISO3 code has been specified (some countries, e.g., Channel Islands in the childlessness data, are not contained in the ISO code data). We then add the data the user wants to see to the geographical world data. Finally, we plot the world map. The most important part of this plot is that contained in the geom\_polygon\_interactive() function from the ggiraph package. This function draws the world map in white with grey lines, fills it up according to the value of the data selected (either childlessness or gender gap rank) in a red-to-blue color scheme set using the brewer.pal() function from the RColorBrewer package, and interactively shows in the tooltip the ISO3 code and value when hovering over the plot.

worldMaps <- function(df, world\_data, data\_type, period, indicator){

# Function for setting the aesthetics of the plot

my\_theme <- function () {

theme\_bw() + theme(axis.text = element\_text(size = 14),

axis.title = element\_text(size = 14),

strip.text = element\_text(size = 14),

panel.grid.major = element\_blank(),

panel.grid.minor = element\_blank(),

panel.background = element\_blank(),

legend.position = "bottom",

panel.border = element\_blank(),

strip.background = element\_rect(fill = 'white', colour = 'white'))

}

# Select only the data that the user has selected to view

plotdf <- df[df$Indicator == indicator & df$DataType == data\_type & df$Period == period,]

plotdf <- plotdf[![is.na](http://is.na)(plotdf$ISO3), ]

# Add the data the user wants to see to the geographical world data

world\_data['DataType'] <- rep(data\_type, nrow(world\_data))

world\_data['Period'] <- rep(period, nrow(world\_data))

world\_data['Indicator'] <- rep(indicator, nrow(world\_data))

world\_data['Value'] <- plotdf$Value[match(world\_data$ISO3, plotdf$ISO3)]

# Create caption with the data source to show underneath the map

capt <- paste0("Source: ", ifelse(data\_type == "Childlessness", "United Nations" , "World Bank"))

# Specify the plot for the world map

library(RColorBrewer)

library(ggiraph)

g <- ggplot() +

geom\_polygon\_interactive(data = world\_data, color = 'gray70', size = 0.1,

aes(x = long, y = lat, fill = Value, group = group,

tooltip = sprintf("%s  
%s", ISO3, Value))) +

scale\_fill\_gradientn(colours = brewer.pal(5, "RdBu"), na.value = 'white') +

scale\_y\_continuous(limits = c(-60, 90), breaks = c()) +

scale\_x\_continuous(breaks = c()) +

labs(fill = data\_type, color = data\_type, title = NULL, x = NULL, y = NULL, caption = capt) +

my\_theme()

return(g)

}

**Building an R Shiny app**

Now that we have our data and world mapping function ready and specified, we can start building our R Shiny app. (If you’re not familiar with R Shiny, I recommend that you to have a look at the [Getting Started guide](https://shiny.rstudio.com/tutorial/) first.) We can build our app by specifying the UI and server components. In the UI, we include a fixed user input selection where the user can choose whether they want to see the childlessness or gender gap index data. We further include dynamic inputs for the period and indicators the user wants to see. As mentioned before, these are dynamic because the choices shown will depend on the selections made by the user on previous inputs. We then use the ggiraph package to output our interactive world map. We use the sidebarLayout() function to show the input selections on the left side and the world map on the right side, rather than the two stacked vertically.

Everything that depends on the inputs by the user needs to be specified in the server function, which in this case is not only the world map creation, but also the second and third input choices, since these depend on the previous inputs made by the user. For example, when we run the app later, we’ll see that when the user selects the childlessness data for the first input for data type, the third indicator input will only show age groups, and the text above the selector will also show “age group”, whereas when the user selects the gender gap index data, the third indicator will show different measures and the text above the selector will show “indicator” rather than “age group”.

Finally, we can run our app by either clicking “Run App” in the top of our RStudio IDE, or by running

shinyApp(ui = ui, server = server)

Below is a screen shot of the app. You can check out the live app [here](https://fverkroost.shinyapps.io/interactive_worldmap_app/). In [this](https://github.com/fverkroost/RStudio-Blogs/blob/master/interactive_worldmap_shiny_embedded.Rmd) post on my GitHub, you can also see how to embed a Shiny app in your R Markdown files, which is a really cool and innovative way of preparing interactive documents. Finally, the source code used to build the live app can also be found on my GitHub [here](https://github.com/fverkroost/RStudio-Blogs/blob/master/interactive_worldmap_shiny_app.R).

Now try selecting different inputs and see how the input choices change when doing so. Also, don’t forget to try hovering over the world map to see different data values for different countries interactively!

