KOSTAT-UNFPA Summer Seminar on Population

Workshop 1. Demography in R

Day 1: Introduction to Rstudio, R, and RMarkdown

Instructor: Tim Riffe
tim.riffe@gmail.com

Assistant: Rustam Tursun-Zade
rustam.tursunzade@gmail.com

26 July 2021

Contents

1 About the instructor

Hi, I'm Tim Riffe, a US-American demographer living and working at the University of the Basque Country (Spain), thanks to a program that brought me here called Ikerbasque. I did my PhD in demography at the Autonomous University of Barcelona in 2013, and spent some years in between working with the Human Mortality Database at the University of California, Berkeley (USA) then working as a research scientist with the Max Planck Institute for Demographic Research (Germany). I've been hooked on `R` since 2009 when I did the European Doctoral School of Demography, and have since authored several packages, mostly that do demographic things, but sometimes that do plotting things. My own research data prep, analyses, analytic plotting, and presentation-quality plotting are all done in `R`. Around three years ago I began adopting the tidyverse approach in my work and teaching.

Rustam Tursun-Zade is an aspiring PhD student, with lots of experience in cancer epidemiology and demography. He is currently affiliated with the European University at St. Petersburg. He also did the European Doctoral School of Demography (2019-2020), and he's an advanced *tidyverse* wizard, and he's able to assist also if there are character or locale issues that arise when using `R` in different language settings.

2 About this workshop

In this workshop we have several direct and indirect objectives. The **direct objectives** are to teach you what you need to comfortably work with data in `R`, including how to:

- read in new data
- reformat data
- reshape data
- merge data together
- summarize
- calculate within groups
- create your own functions
- flexible visualization using `ggplot2`

The **indirect objectives** are to teach you:

- Flexible reporting with `R markdown`
- Reproducible analyses
- Confidence with data
- a bit of `git` version control maybe.
- demographic thinking via examples, including:
  - life tables
  - standardization
  - decomposition
  - Lexis surfaces
  - others

Each day of the workshop will have a prepared handout, this being the first one. The handout sets out the lesson plan for the day, including code examples. When convenient, I will follow this verbatim, however, most code will generally be derived in a live session. When appropriate, I'll also share improvised / derived code with you via a Google Doc. The hope is that you will be able to follow along with me. Code will be presented in small units, or *chunks*, and commented live both verbally and in writing, as we progress. This will indirectly introduce you to markdown. In case on any day we do not arrive at the end of a lesson plan, the day's handout is also a reproducible tutorial that you can follow on your own. The markdown document created progressively through the day will be tidied up and posted to the course repository each day after class.

We will have frequent short breaks, very likely inspired by the loud church bells that will interrupt the session on my end every hour.

2.1 Course materials

All material is available in a public `github` repository, here: https://github.com/timriffe/KOSTAT_Workshop1

The handout for each day is available in its R markdown format and as a built pdf document, following the naming convention `01_handout.Rmd` (pretty document called `01_handout.pdf`), etc. The session materials are also given in `.Rmd` and pdf formats, with the naming convention `01_session.Rmd` (`01_session.pdf`). These will be available after class each day, or by the following morning at the latest. Exercises may be given in the handout for each day, and solutions either provided or derived the following day in a markdown document called e.g. `01_solutions.Rmd` (`01_solutions.pdf`). The `README` file will be updated daily as well as new items are added.

Our Google Doc pastebin is here: https://tinyurl.com/6ec2zfyy . This link gives edit permissions, so you could also use it to paste code that isn't working, or to pose questions. We'll try to not let this get cluttered up.

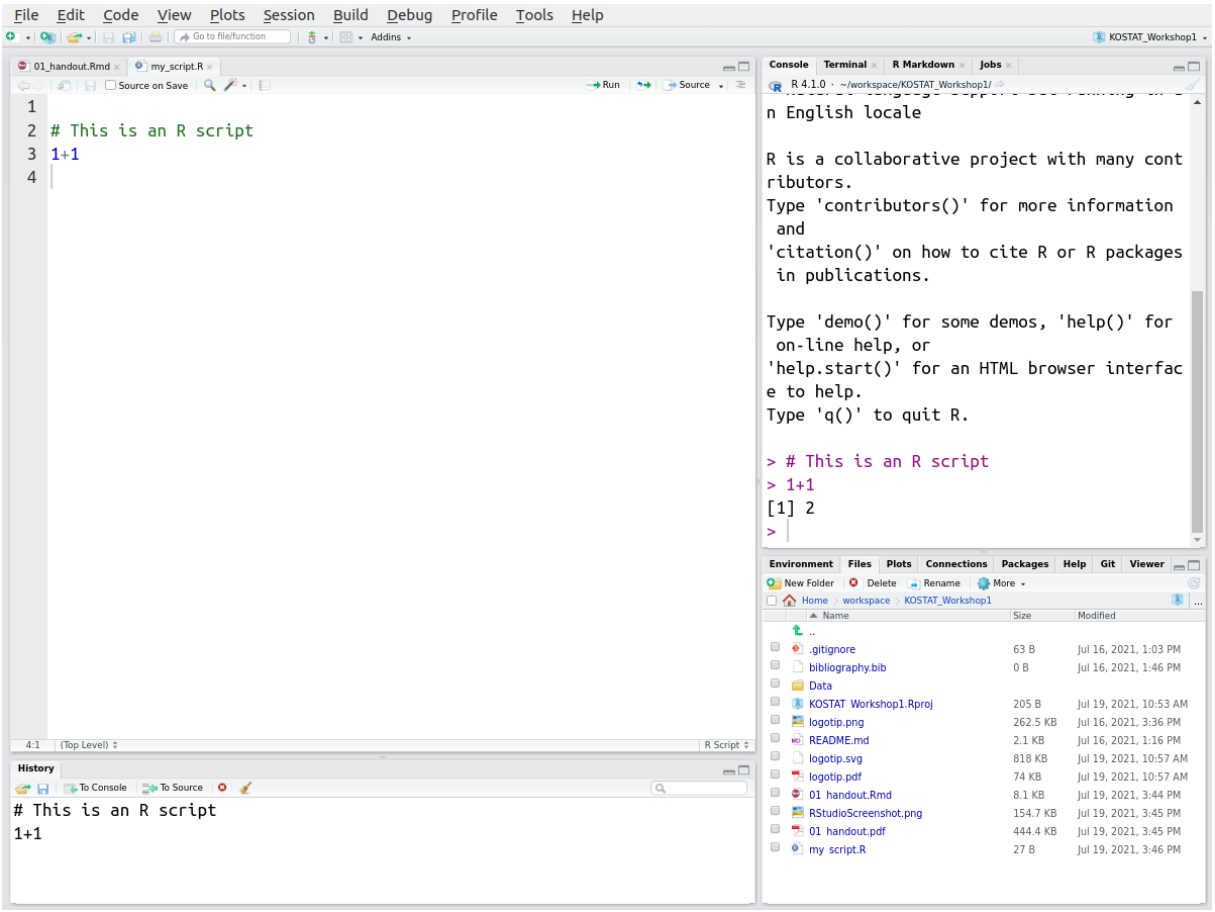
2.2 Additional help

If you do not manage to ask a question during the session (either verbally, or in the chat), you have some other options: paste the question in the Google Doc. Post an `issue` on the repository https://github.com/timriffe/KOSTAT_Workshop1/issues- in that case I encourage you to make the problem reproducible so that it's more direct for us to troubleshoot. Alternatively, you can ask us informally by email (both the instructor and assistant). Please please please ask questions or ask me to slow down or stop or "*scroll up to show the code again*" if necessary. If you have a question then very likely someone else has the same question, so please ask and don't worry about interrupting me.

3 Tour of Rstudio

I presume by now you have `RStudio` installed. If not, go here https://www.rstudio.com/products/rstudio/download/preview/ and install the *preview* version. We're using this version because some issues specific to some operating systems are resolved there.

When you open `RStudio` you see four main panels (panes):



3.1 Editor

The editor pane is where we create text documents, such as `.R` scripts and `.Rmd` R markdown documents containing both text and code. It usually makes sense for this to be the **biggest** pane. We will type in here most of the time. Typing code in the editor is inert.

3.2 Console

The `R` console is literally a live `R` session. You can *send* code here from the editor by selecting it and pressing `Ctrl + Enter`. This is true for scripts and for markdown code chunks (to be explained). I will demonstrate different ways of doing this. You can also type code directly in the console and execute it by pressing `Enter`. This pane may also contain other tabs for things like the `Terminal` (i.e. the black box), `R Markdown`, but we won't use these probably.

3.3 History

This panel just shows you the console output from code that has been executed. You can minimize this panel, as it's usually a waste of space.

3.4 everything else

This includes useful things like:

- Environment: allows you to browse and view data objects that you have in memory in `R`
- Files: a file browser, can be more convenient than having a second one open
- Plots: viewing window for plots created live
- Packages: a helpful way to install packages sometimes
- Help: a viewer for function and package help files
- Git: You might see me manage github things here

Although this is a useful panel, it does not need to be that big.

4 Step-by-step get started

4.1 Use an `Rstudio` *project* for this course

We should all work on this course within an `RStudio` *project*. This defines a nice neat space to stash files for the course, and it makes file path specification much much easier. It also makes the entire course material portable, which helps with course reproducbility in the medium term. I give three alternatives for getting started with the course project. I will demonstrate these approaches in the session.

1. In `RStudio` select `File - New Project - New Directory - New Project` - type in `KOSTAT_Workshop1` and pick out a directory location you won't forget - `Create Project`. This will create a new empty project that you can put materials in for this course. This ought to work for everyone.

2. Alternatively, if you have `git` installed, you can do:

`File` - `New Project` - `Version Control` - `git` - paste `https://github.com/timriffe/KOSTAT_Workshop1.git` into the `url` box, and select a directory location you won't forget - `Create Project`. This will create a new project that is a *clone* of the materials currently shared in this workshop's `github` repository. This is not necessary, but it might be nice if you want to get started with `git`.

3. Alternatively, if you don't have `git` installed and do not want to install it, but still want to start with all materials shared, go to https://github.com/timriffe/KOSTAT_Workshop1 in your browser, click on the green button named `Code`, and select `Download ZIP`. Unpack the zip in a directory location you won't forget and be sure to call the folder `KOSTAT_Workshop1`. Then, in `RStudio` select `File - New Project - Existing directory` - select the folder `KOSTAT_Workshop1` that you just created - `Create project`.

No matter which method you've done, you'll now be in a project view, which is convenient.

4.2 Start a new markdown document

Given we're in a new project, we can now create files in it. Let's make a new markdown file for this session. You can call yours whatever you want. To do this, do `File` - `New File` - `R Markdown` - this opens a dialogue where you can specify a document title, say `"Monday class notes"`. If you have `LaTeX` installed, you can select `PDF` as the output type, otherwise choose `html` or `Word` as you like. Click `OK` and a new unnamed `.Rmd` file will be created with a template in it to help you get started with markdown. Save the file in the standard way, calling it something like `01_notes.Rmd` or similar. I will call mine `01_session.Rmd`, but you should use a different name for yours. I will demonstrate these approaches in the session, and this will be where the entire class today will take place :-)

4.2.1 *build* the template provided

The document that has been created for you has a template structure that serves to orient you in the ways of markdown. Before looking at its parts, let's build the document. In the top icons you'll see one called `Knit` with a tiny picture of a ball of yarn... Click that and a new file will be created. If your markdown file is saved as `01_notes.Rmd` then the new file will be called `01_notes.pdf` or `01_notes.docx` or `01_notes.html` and it may or may not automatically open for you to view it. Pretty cool eh? See how the document has parts that are just normal text, others that are nicely formatted code, and others that are the results of that code having been executed (console output or perhaps a figure?). I hope you'll agree this sort of document will be an excellent note-taking environment for this course. Indeed, it's the only environment we'll use. In the document

4.2.2 Code chunks

`R` code in a markdown document can be considered *live* `R` code. It is written in chunks inside the following construct

```
```{ r }
# anything inside here is R code

```
```

To crate a blank code chunk, type `Ctrl + Alt + i` (mac: `Cmd + Option + i`. Remember this! Otherwise, you need to find the *back-tic* key on your keyboard! On my keyboard it is in the far upper left corner, but in yours it might not be.

Create a chunk and type `1+1` in it:

```
1+1
```

```
## [1] 2
```

Place you cursor on the line and press `Ctrl + Enter` (mac: `Cmd + Enter`). This executes the code in the console, but does not *create* anything. If you `Knit` the document again then you'll see the code chunk featured, along with the console output `[1] 2`, where `[1]` is the line number of the output (unimportant) and `2` is the actual result. Alternatively, in the top right corner of the chunk itself, you'll see a little green `play` arrow. Clicking that executes the chunk.

If you have more than one chunk, then knitting the document executes them in order. What happens in earlier chunks can affect what goes on in later chunks: they are sequentially dependent (potentially).

Other features of code chunks and how to control them will be demonstrated live over the course of this week, but not explicitly covered. You can learn more about code chunks here: https://rmarkdown.rstudio.com/lesson-3.html

4.2.3 `YAML` header

Your template document will also have a header looking something like this:

```
---
title: "01_notes"
author: "Your name"
date: "7/26/2021"
output: pdf_document
---
```

These are just control parameters for the document. This is not R and it's also not R markdown! Let's not stress about it, but instead just note that you can control how the document gets created with more parameters, and this may be interesting to know. You can find out more here https://r4ds.had.co.nz/r-markdown.html. Or check out this R package that helps you write them with a more advanced interface https://cran.r-project.org/web/packages/ymlthis/vignettes/introduction-to-ymlthis.html.

4.2.4 The text in between

Note that whatever you write in between code chunks is rather flexible: it could be just normal text, no frills, or it could have markup elements like itemization, italics, etc using native markdown, or it could be native LaTeX if you have that installed and are outputting to pdf, or it could be native html if you're outputting to html. If you're outputting to Word you can even use another manually created Word document as a template for formatting preferences or to follow institutional formatting guidelines. To learn how to do that look here: https://bookdown.org/yihui/rmarkdown-cookbook/word-template.html.

I will be using such options from time to time, and will not make a very big deal out of it. For now, just know that you have quite a lot of control over this document, if you want to, but the default settings are pretty useful.

4.3 Moving forward with this project

You can of course create more markdown files, each building to a different destination object. The important thing for us is that you do so *inside* the project you've defined for this workshop. When in doubt: File - Open project - navigate to and select KOSTAT_Workshop1.Rproj - Open project. (double clicking KOTAT_Workshop1.Rproj inside your usual file browser will also open the project in RStudio. That's it, you can always get back to this workspace once you've set it up, and you can set up other project spaces for other projects of yours.

5 R basics

Now that we're situated in a document, and know how to make code chunks, let's see what you can do inside them!

5.1 Functions live in packages

R is a *functional* programming language, meaning stuff gets done using functions. We can get by with the functions provided with basic R, or else we can load more specialized functions from particular packages. To load a package, use the function install.packages(). Start a new chunk (Ctrl + Alt + i) and type install.packages("tidyverse") to install the package called tidyverse. Execute the code by typing Ctrl + Enter with your cursor on the line, or by clicking the green play arrow for the chunk.

```
install.packages("tidyverse")
```

Having run this code chunk once, let's now comment out the install.packages() line so that we don't inadvertently re-install the package each time we build this document! To comment out code, and to make notes in code in general, use the hash # to mark which lines to ignore.

```
# comment out line so it doesn't re-run each time we build!
# install.packages("tidyverse")
```

We have now used the function `install.packges()`. The name of the function is everything before the parentheses `install.packages`, and the part in the parentheses `()` is the function input parameters, or arguments. In this case, we just had to give a character string telling which package we want. By default, this function will look in the main `R` package repositories, called `CRAN`, here: https://cran.r-project.org/. Anything posted to that archive has passed a big set of standardized checks and can be considered safe. The code you find there may or may not have been subjected to methodological review. On the other hand, the code you find in packages there is absolutely free and open, meaning you have the ability to see how it works.

We will make use of various packages from there. Let's install a few more. Remember to comment these lines out after you run them!

```
# Functions for generalized demographic decomposition
install.packages("DemoDecomp")

# Functions for reading in assorted data files, e.g. read_csv()
install.packages("readr")

# read_excel() reads in data from an .xlsx file
install.packages("readxl")

# nice date handling functions
install.packages("lubridate")

# nice axis scales
install.packages("scales")

# great color palettes
install.packages("colorspace")
```

To *load* the packages, i.e. make their functions available to you in your `R` session, use `library()`:

```
library(tidyverse)
library(readr)
library(readxl)
library(lubridate)
library(scales)
library(colorspace)
library(DemoDecomp)
```

5.2 Help files

Often when trying to use a function, you'll not be sure about all of the options it might have. To get a glimpse of the documentation provided for `library()`, type `?library` in the console. This will open up the help file in the `Help` panel of `RStudio`. Most functions in `R` have help files, and often but not always these are useful. They just take some getting used to. The main sections to pay attention to are:

- **Description** for a short human language summary of what the function is supposed to do.
- **Arguments** for a description of each argument
- **Details** for any gory details the author thinks you might want to know
- **Value** tells you what the function gives back
- **Examples** show some real world examples of the function being used. If you copy and paste them, they ought to run. This is very useful information.

I will be using many functions that may be unfamiliar to you, and which I may only introduce briefly. for more information, you can always get to the help file using `?function_name`.

Also, `RStudio` has a very helpful autocomplete functionality that gives helpful hints when typing out function and argument names. Making use of this will make your life easier, and you'll see me exploiting this quite a lot.

5.3 Basic arithmetic

`R` has most common arithmetic and matrix operators (more than I list here)

- `+`
- `-`
- `*` standard multiplication
- `/`
- `^` standard power
- `%*%` *matrix* multiplication

```
1 + 5
```

```
## [1] 6
```

```
1 - 5
```

```
## [1] -4
```

```
2 * 5
```

```
## [1] 10
```

```
2 / 5
```

```
## [1] 0.4
```

```
2 ^ 5
```

```
## [1] 32
```

There are also many functions to help with such things (these and many many more):

- `sum()`
- `mean()`
- `prod()`
- `cumsum()`
- `cumprod()`
- `exp()` i.e. e^x
- `log()` natural log by default.

Note `c()` includes numbers in a *vector*. The first three functions return a scalar, whereas the others return a vector of results that is the same length as the input! These are said to be *vectorized*.

```
# return scalars
sum(c(2, 5, 7))
```

```
## [1] 14
```

```
mean(c(2, 5, 7))
```

```
## [1] 4.666667
```

```
prod(c(2, 5, 7))
```

```
## [1] 70
```

```
# return vectors same length
cumsum(c(2, 5, 7))
```

```
## [1]  2  7 14
```

```
cumprod(c(2, 5, 7))
```

```
## [1]  2 10 70
```

```
exp(c(2, 5, 7))
```

```
## [1]    7.389056  148.413159 1096.633158
```

```
log(c(2, 5, 7))
```

```
## [1] 0.6931472 1.6094379 1.9459101
```

You can also have fancy expressions, making intelligent use of parentheses, just like in standard mathematics, standard *order of operations* applies.

```
.07 / (1 + (1 - .5) * .7)
```

```
## [1] 0.05185185
```

5.4 Objects

You can make your life easier by using objects in `R`.

5.4.1 vectors and matrices

Take the vector `c(2,5,7)`: rather than typing this out, we can *assign* it to an object using `<-`. Here we declare an object whose name is `a`, and this object is a vector with three integers in a particular order. Once defined, this object will persist in this `R` session. Execute this chunk and have a look at the `Environment` tab in `RStudio`. You'll see a new object called `a`, and since it's small, you can even see the whole thing right there.

```
a <- c(2, 5, 7)
exp(a)
```

```
## [1]    7.389056  148.413159 1096.633158
```

```
log(a)
```

```
## [1] 0.6931472 1.6094379 1.9459101
```

```
# data storage mode
typeof(a)
```

```
## [1] "double"
```

```
# kind of R object
class(a)
```

```
## [1] "numeric"
```

```
# ask if it's a vector
is.vector(a)
```

```
## [1] TRUE
```

```
# how many elements?
length(a)
```

```
## [1] 3
```

You can create many objects in an R session. They can all be different sizes and types. Sessions can become large and cluttered in this way. From time to time, we'll have a glance at the Environment tab to check out what our session is looking like. From time to time, we'll do some house-keeping so that the session environment doesn't get cluttered. You can remove an object using rm()

```
rm(a)
```

A vector like a is the most basic kind of R object. Vectors can also be character, logical, and other kinds of things:

```
b <- c("A", "b", "C")
d <- c(TRUE, FALSE, FALSE)
```

A matrix is just a vector with two dimensions. Here we create a matrix using the matrix() function, by giving a vector of 6 elements and specifying we want two columns ncol = 2. The matrix is filled column-wise, unless you tell it otherwise

```
a <- c(2,5,7,3,6,8)
A <- matrix(a, ncol = 2)
A
```

```
##      [,1] [,2]
## [1,]    2    3
## [2,]    5    6
## [3,]    7    8
```

```
B <- matrix(a, ncol = 2, byrow = TRUE)
B
```

```
##      [,1] [,2]
## [1,]    2    5
## [2,]    7    3
## [3,]    6    8
```

```
# dimensions in rows, columns
dim(A)
```

```
## [1] 3 2
```

```
# or more specifically:
nrow(A)
```

```
## [1] 3
```

```
ncol(A)
```

```
## [1] 2
```

Much of statistics in R is implemented using matrix algebra. Here t() is the matrix transpose.

```
A %*% t(B)
```

```
##      [,1] [,2] [,3]
```

```
## [1,]   19   23   36
## [2,]   40   53   78
## [3,]   54   73  106
```

Traditionally, we would spend much more time in a course like this covering that, but we will skip many other details to be abke to spend more time on data analysis.

5.4.2 `data.frame`

Matrices may eventually become a fundamental part of the way you use `R`, but in this course we will emphasize data analysis using `data.frame`s, which is `R`-speak for rectangular tables with flexible data types. We can create one manually using `data.frame()`. Columns are comma-separated.

```
DF <- data.frame(
           x = c("a","a","a","b","b","b","b"),
           y = c(2, 5, 7, 8 ,1, 9, 0)
           )
DF
```

```
##   x y
## 1 a 2
## 2 a 5
## 3 a 7
## 4 b 8
## 5 b 1
## 6 b 9
## 7 b 0
```

If you had those vectors sitting around, you could just create the `data.frame` in a more economical way:

```
x <- rep(c("a","b"), times = c(3,4))
y <- c(2, 5, 7, 8 ,1, 9, 0)
data.frame(x, y)
```

```
##   x y
## 1 a 2
## 2 a 5
## 3 a 7
## 4 b 8
## 5 b 1
## 6 b 9
## 7 b 0
```

The main rules for `data.frame` is that the columns are all the same length, and that each cell has just one value. The columns can be different data types. This kind of object will form the foundation of the `tidy` data approach, which we will follow using a cousin of the `data.frame`, `tibbles`. You can treat these as synonyms.

5.4.3 Functions

Functions are little programs we write that takes a defined input and converts it to an output. They are also objects with names. Functions from packages are objects that aren't necessarily shown in your environment viewer, whereas functions *you* write are visible there. You can see the code that a function applies by typing a function name into the console (without the parentheses).

The `DemoDecomp` package has a function called `horiuchi()`. Type `horiuchi` in the console to see the function body:

```
horiuchi
```

```
## function (func, pars1, pars2, N, ...)
## {
##     y1 <- func(pars1, ...)
##     y2 <- func(pars2, ...)
##     d <- pars2 - pars1
##     n <- length(pars1)
##     delta <- d/N
##     x <- pars1 + d * matrix(rep(0.5:(N - 0.5)/N, n), byrow = TRUE,
##         ncol = N)
##     cc <- matrix(0, nrow = n, ncol = N)
##     zeros <- rep(0, n)
##     for (j in 1:N) {
##         DD <- diag(delta/2)
##         for (i in 1:n) {
##             cc[i, j] <- func((x[, j] + DD[, i]), ...) - func((x[,
##                 j] - DD[, i]), ...)
##         }
##     }
##     return(rowSums(cc))
## }
## <bytecode: 0x561e0d3cd238>
## <environment: namespace:DemoDecomp>
```

Briefly, this is an implementation of a popular demographic decomposition method. For now, just convince yourself that rather than needing to type out all this code each time you want to use this decomposition method (hypothetically), it's far easier and faster to use the `horiuchi()` function and not need to remember it or re-derive it.

Wednesday we will spend an entire session learning to write our own functions.

6 Teaser

This set of basic introductions suffices for this course. Know that the world of basic (or `Base`) `R` is extensive, and that the kind of code we'll be writing (`tidyverse` code) has been built on top of it with the aim of simplifying our interface with data. This feat has removed much of the need to *program* in order to do complex tasks in `R`.

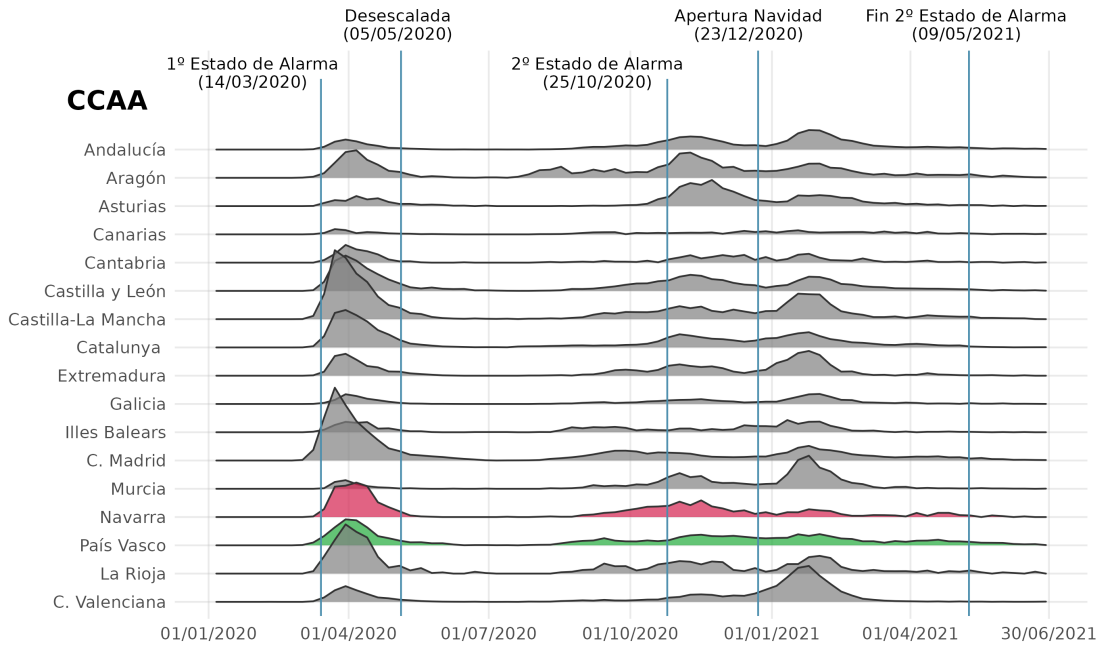
To motivate you, imagine the following multistage task:

6.1 Data (imagine)

You are given a data set containing a daily time series of COVID-19 cases, hospital admissions, urgent care admissions, and deaths by age, sex, and Spanish provinces. Population denominators are delivered in a different data set using different age groups for Spanish autonomous communities (these are aggregates of provinces). A third data source tells you which province is in which autonomous community. A fourth data set gives code equivalencies for provinces and autonomous communities, because *of course* the different data sources use different location coding systems.

6.2 Objective (imagine)

Calculate *weekly* time series of **age-sex-standardized rates** of cases, hospitalizations, urgent care admissions, and deaths for each autonomous community and visualize these in *ridgeplots* that look something like this (except with labels in your language of course):



By the end of this week, you will be able to handle all of the data loading, data joining, tabulation and calculation and `ggplot2` operations needed to complete a task of this complexity. Further the code to do so will be *parsimonious* and it will be *legible*, and you will be empowered to undertake data analysis tasks of approximately

7 Excercises

1. Look at some of the files offered by the World Population Prospects, here: https://population.un.org/wpp/Download/Standard/Population/. There is no need to register to use these data, and we will be using some of it tomorrow, **Tuesday**. Download a file or two and have a look at what's inside.
2. Register to be a user of the Human Mortality Database (HMD), here: https://www.mortality.org/mp/auth.pl. If you're already a user then no worries. We will use this on **Wednesday**.
3. Look at this gallery of `ggplot2` extensions: https://exts.ggplot2.tidyverse.org/gallery/ Know that these will be things you'll be able to extend *toward* doing if you follow along over the course of this week. **Thursday** We will cover the basics to get started with `ggplot2`
4. Vote for a data set we should work with on **Friday**, here: https://forms.gle/oqtpN7zcFj9sQUqD9
5. Look at this free book by Kieran Healey and bookmark it for future reference: https://socviz.co/. This is a great way to self-learn. That book would be a great way to supplement this course in the following weeks.