KOSTAT-UNFPA Summer Seminar on Population

Workshop 1. Demography in R

Day 1: Basic Demographic Data and Concepts

Instructor:
Tim Riffe `tim.riffe@ehu.eus`

Assistant:
Dr. Da eun Kwan: `dekwan20@kdis.ac.kr`

24 July 2023

Contents

1 About the instructor

Hi, I'm Tim Riffe, a US-American demographer living in Spain and working at the University of the Basque Country since 2021, thanks to a program that brought me here called Ikerbasque. I defended 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 (2012-2015, USA) then working as a research scientist with the Max Planck Institute for Demographic Research (2015-2021, 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`.

2 About this workshop

In this workshop we have several direct and indirect objectives. The **direct objectives** are to teach you to think like a demographer, including main concepts such as:

- stocks versus flows
- age, period, and cohort time perspectives
- mortality and fertility measures [trends, inequalities]
- population structure [heterogeneity, aging]
- population growth [balancing equation, Leslie matrix, stability]
- projection [Lee Carter Model]

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

- Flexible reporting with `R markdown`
- Reproducible analyses
- Confidence with data
- programming concepts such as:
  - read in new data
  - reformat data
  - reshape data
  - merge data together
  - summarize
  - calculate within groups
  - create your own functions
  - flexible visualization using `ggplot2`
- demographic examples, including:
  - life tables
  - standardization
  - decomposition
  - Lexis surfaces

Each day of the workshop will have a lecture component and a practical component. The lecture component will consist in slides and marker board presentation, whereas the practical component will be based on live coding. The objective of the lecture component will be to

introduce basic demographic concepts and teach you the basic perspectives needed in order to think like a demographer. The practical component will introduce basic coding tools and give you a contemporary coding strategy capable of handling most situations.

I will provide a handout to accompany both components, this being the first one. The handout sets out the lesson plan for the day, including code examples. When convenient, I will follow these code examples verbatim, however, most code will generally be derived in a live session. When appropriate, I'll also share improvised / derived code with you spontaneously 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. I highly recommend going through the handouts and the session code after each session to solidify the material of the day.

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 an 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 Theoretical

This section will introduce broad concepts within demography. We have not yet learned a thing about R at this point. I do however include some R-code that you will only come to understand later in this workshop. The practical component of this lecture will begin to introduce our R computing environment. Please do not feel like you need to understand the R chunks displayed in this section at this time. ## What is a population?

Broadly defined, demographers refer to a population as "*the collection of persons alive at a specific point in time who meet certain criteria*" (Preston, Heuveline, and Guillot 2001).

Often we just deal with populations defined by administrative boundaries, but this is not necessarily the case.

3.1 What is demography?

According to the Max Planck Institute for Demographic Research: "*Demography is the science of populations. Demographers seek to understand population dynamics by investigating three main demographic processes: birth, migration, and aging (including death).*" [https://www.demogr.mpg.de/en/about_us_6113/what_is_demography_6674/]

According to the Oxford Dictionary demography is "*The study of statistics such as births, deaths, income, or the incidence of disease, which illustrate the changing structure of human populations.*"

One more... "*Demography is the study of the size, territorial distribution, and composition of population, changes therein, and the components of such changes.*" (Hauser and Duncan 1959)

In summary, demographers use quantitative approaches to study population dynamics and changes by investigating their

- Size
- Growth
- Composition/structure (age, socioeconomic status, etc.)
- Processes: Fertility, Mortality and Migration

And many more things because really the field is a cognate of many other sciences.

3.2 Three dimensions of changes

Generally, demographers study demographic events (birth, death and migration) on three interrelated aspects of temporal structure: age, period, and cohort.

3.2.1 Age and period

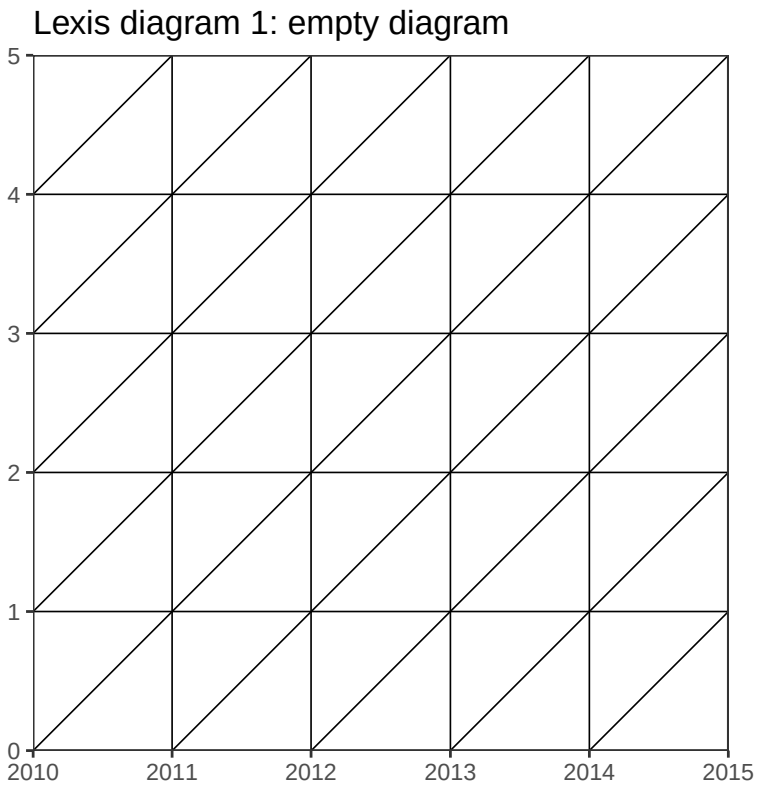
In censuses, civil registries, and most surveys, data and events will generally be recorded by year and age of occurrence. For example, in 2014, 1202 infants died before age 1 in Spain (Human Mortality Database 2018). ### Cohort

A cohort is defined as "*the aggregate of all units that experience a particular demographic event during a specific time interval*" (Preston, Heuveline, and Guillot 2001). For example, the 1950 Spanish birth cohort consist of all individuals born in 1950 in Spain.

3.2.2 Lexis diagram

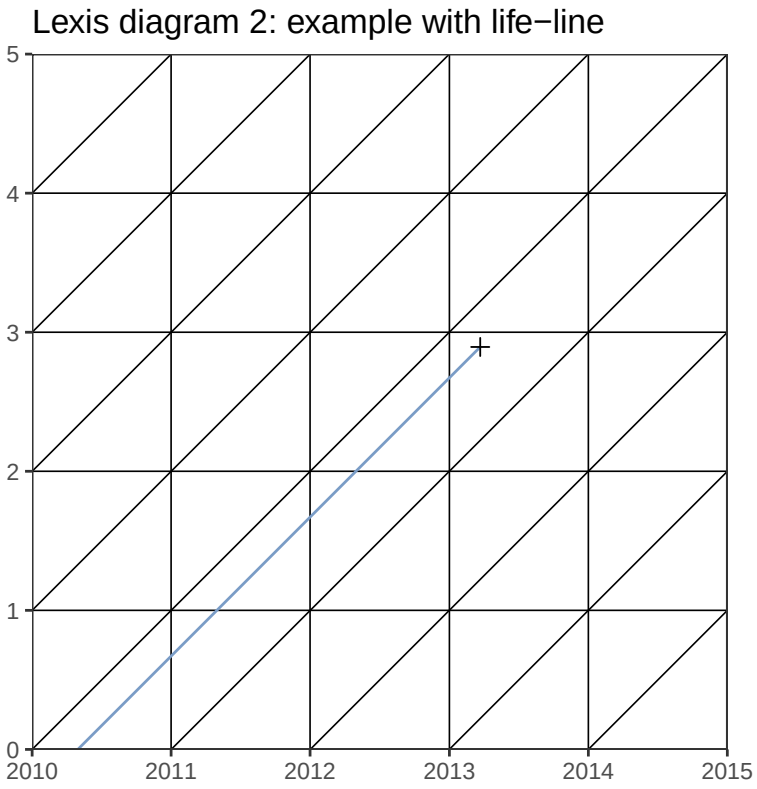
The Lexis diagram is a two-dimensional figure used to represent demographic events across year, age, and cohort. On the diagram, age is one dimension and calendar year is the other. Cohorts advance through life along a 45-degree line.

```
library("LexisPlotR")
library(ggplot2)
mylexis <- lexis_grid(year_start = 2010, year_end = 2015,
                      age_start = 0, age_end = 5)+
  labs(title = "Lexis diagram 1: empty diagram")
mylexis
```



On a Lexis diagram, each individual is represented as a *life-line*, advancing through time and age as a parallel line to the cohort line.

```
mylexis<-lexis_grid(year_start = 2010, year_end = 2015,
                      age_start = 0, age_end = 5)+
  labs(title = "Lexis diagram 2: example with life-line")
lexis_lifeline(lg = mylexis,
               birth = "2010-05-01",
               exit="2013-03-23",
               lineends = TRUE)
```



4 Data

We will use the Spanish females data from the HMD (Human Mortality Database 2018) and HFD (Human Fertility Database 2018) for the year 2014 (and 2015 population counts). Please read in the raw data using these lines:

```
# read in, harmonize, filter, and merge data sources
library(tidyverse)
library(readr)

# Birth counts in 2014
B2014 <- read_csv("Data/BirthsKORHFD.csv",
                  show_col_types = FALSE) |>
  mutate(sex = "Total") %>%
  select(year = Year, age = Age, sex, births = Total) |>
  mutate(age = parse_number(age)) |>
  filter(year == 2014)

# Death counts in 2014
D2014 <- read_csv("Data/DeathsKORHMD.csv",
                  show_col_types = FALSE)  |>
  select(-OpenInterval) |>
  pivot_longer(Female:Total,
               names_to = "sex",
               values_to = "deaths") |>
  select(year = Year, age = Age, sex, deaths) |>
```

```
  filter(year == 2014)

# HMD exposure estimates for 2014
E2014 <- read_csv("Data/ExposuresKORHMD.csv",
                  show_col_types = FALSE)  |>
         pivot_longer(Female:Total,
                      names_to = "sex",
                      values_to = "exposure") |>
        select(year = Year, age = Age, sex, exposure) |>
  filter(year == 2014)

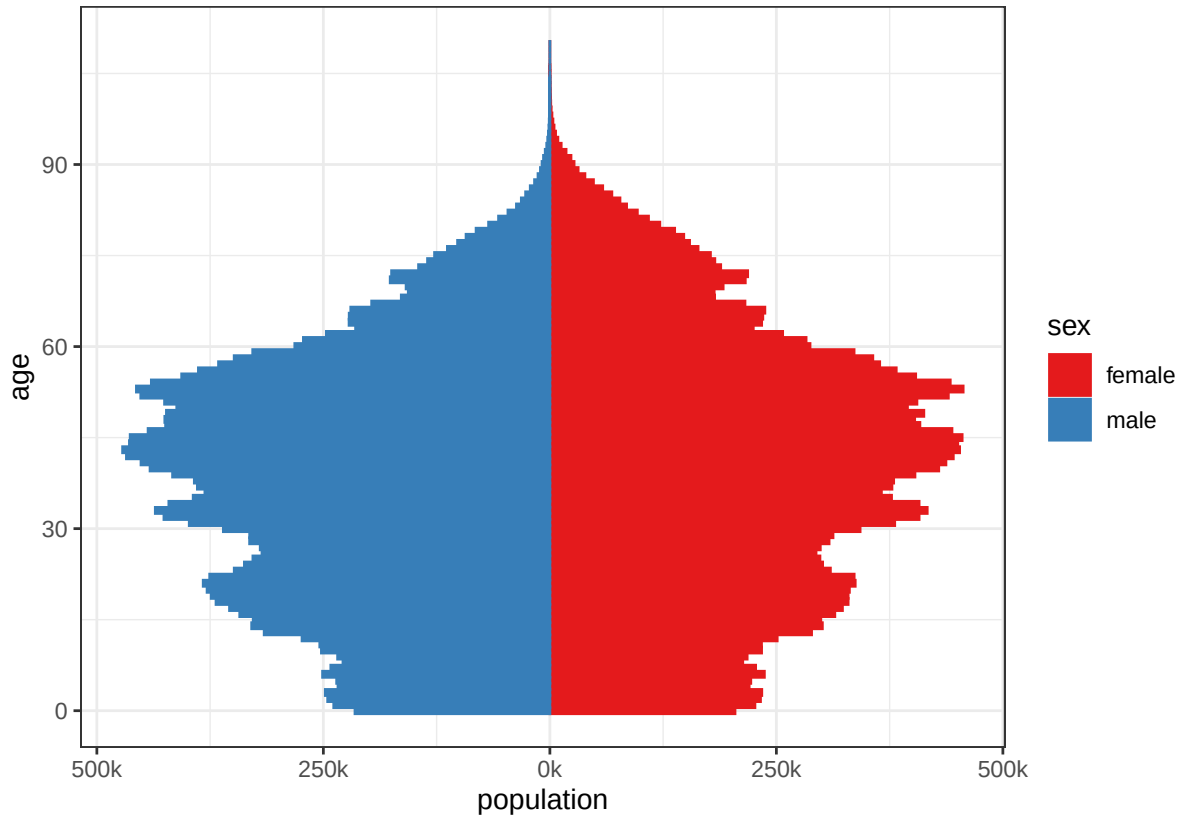
# merged 2014 data
KOR2014 <-
  E2014 |>
  left_join(D2014, by = c("year", "age", "sex")) |>
  left_join(B2014, by = c("year", "age", "sex")) |>
  arrange(sex, age) |>
  mutate(sex = tolower(sex),
         births = if_else(is.na(births),0,births))

write_csv(KOR2014, "Data/KOR2014.csv")
```

5 Population pyramid

Population pyramid or age pyramid is the most common tool to visualize the age structure of a population by sex. Each age is represented by a horizontal bar and male age distribution is plotted on the left and female age distribution on the right. Young ages are found at the bottom and older ages at the top.

```
#Plot the pyramid with ggplot2 (loaded w tidyverse)
KOR2014 |>
  select(sex, age, population = exposure) |>
  filter(sex != "total") |>
  mutate(population = ifelse(sex == "male", -population, population)) |>
  ggplot( aes(x = age, y = population, fill = sex, color = sex)) +
  geom_bar(stat = "identity",width=1) +
  scale_y_continuous(breaks = seq(-500000, 500000, 250000),
                     labels = paste0(as.character(c(seq(500, 0, -250),
                                                    seq(250, 500, 250))), "k")) +
  coord_flip() +
  scale_color_brewer(palette = "Set1") +
  scale_fill_brewer(palette = "Set1") +
  theme_bw()
```



A population pyramid is an artifact of the demographic history of a population. Its profile is driven by past changes in fertility, mortality, and migration.

6 Demographic rates vs probability

6.1 Probability

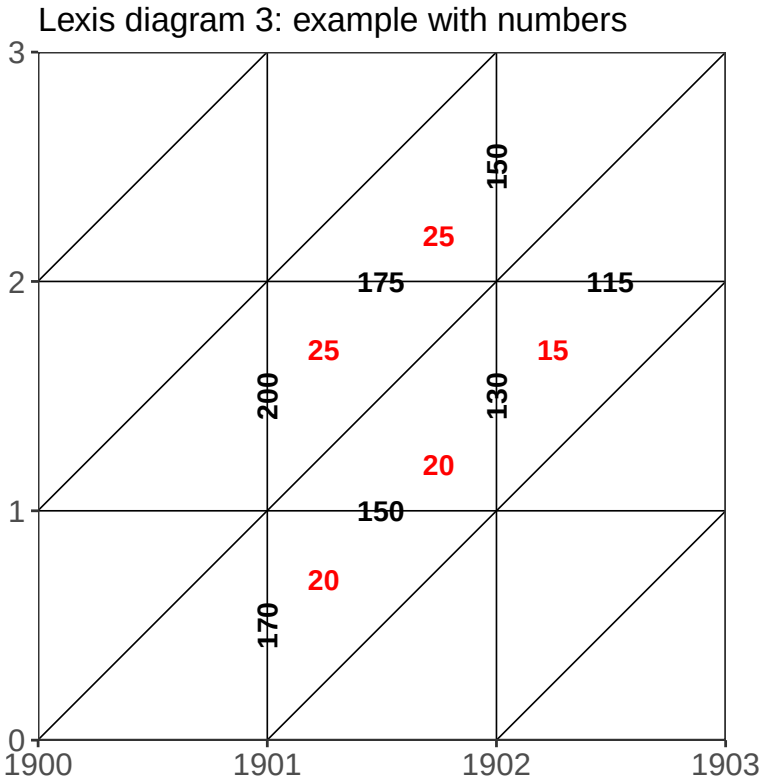
A probability can only be directly observed for a cohort. It refers to the chance that an event will occur (Preston, Heuveline, and Guillot 2001) and is calculated as:

$$Probability = \frac{Number\ of\ Occurences}{Number\ of\ preceeding\ Events\ or\ trials}$$

For example, the probability of dying at a specific age - e.g. age 5 - is the number of deaths occurring at age 5 (between the exact age 5 and exact age 6) in a cohort, divided by the number of people who reached the exact age of 5 in the same cohort. In the Lexis diagram below, the numbers in red are the events (deaths) and the numbers in black are the number of people alive at an exact age or date. Using this example, the death probability at age 1 from the birth cohort 1900 is:

```
deaths      <- 20 + 15
preAlive    <- 150
probability <- deaths / preAlive
probability
```

```
## [1] 0.2333333
```



Probabilities are can not fall outside the range $(0, 1)$ (inclusive).

6.2 Rate

Rather than the chance that an event occurs, as the probability, we here look at the rate (or speed) at which events occur. Rates can be both applied to period and cohort analysis. However, as most demographic information is periodic, period rates are more often used in practice. Other period-perspective demographic quantities, such as period life expectancy, or conditional probabilities, are then derived as needed from rates. We will discuss the implications of period-perspective measures in tomorrow's lesson.

In demography, rates are normally known as occurrence-exposure rates (events related to exposure to risk). Often, exposure rates referred to person-year lived (Preston, Heuveline, and Guillot 2001). Rates differ from probabilities in the denominator:

$$Rate = \frac{Number\ of\ Occurences}{Number\ of\ person-years\ lived}$$

With rates, the number of occurrences is scaled to the population size **per unit time**. Rates tell us the intensity of occurrence for a phenomenon.

6.2.1 Person-years

The person-years concept refers to the number year a person lived - i.e. is *exposed* to the event - in a specific interval. For example, if a person lives 1 year in an interval 0 to 1 year, then this person contributed one person-year. If another person lived 5 days in the same interval, this person contributed (about) 5/365ths of a person-year in the same time interval.

In the Lexis diagram 2, the life-line illustrates the case of a person who contributed 0.89 (326/365) person-year of exposure to the risk of dying at age 2 for the birth cohort 2010. The same person contributed 2.89 person-years of exposure in the full 2010 birth cohort.

When person-years are used, the rate is an *annualized* rate.

The person-years are however rarely observed or counted (Preston, Heuveline, and Guillot 2001). Thus, person-years (PY) are often calculated by assuming that the persons (P) who experienced an event (E) in a given interval ($t : t + n$), experienced it at the mid-point of the interval:

$$_nPY_t = n \cdot \frac{P_t + P_{t+n}}{2}$$

For example, the death rate at age 1 for the birth cohort 1900 on the Lexis diagram 3 is:

```
deaths <- 20+15
Pyear  <- (150+115)/2
rate   <- deaths/Pyear
rate
```

```
## [1] 0.2641509
```

The period death rate at age 1 in 1901 is:

```
deaths <- 25+20
Pyear  <- (200+130)/2
rate   <- deaths/Pyear
rate
```

```
## [1] 0.2727273
```

Sometimes statistical offices release midyear population estimates, derived either directly from population registers or on the basis of demographic accounting. There are also plenty of other approximations out there.

7 Death rates (all-cause)

- Events (occurrences): deaths
- Exposure: All person-years lived in the time interval

7.1 Crude death rate

The crude death rate (CDR) is a measure of the risk of death for a whole population:

$$CDR[t, n] = \frac{Number\ of\ deaths\ in\ the\ population\ between\ times\ t\ and\ t + n}{Number\ of\ person - years\ lived\ in\ the\ population\ between\ times\ t\ and\ t + n}$$

```
# Person-years
  KOR2014 |>
  filter(sex == "total") |>
  summarize(exposure = sum(exposure),
            deaths = sum(deaths)) |>
  mutate(CDR = deaths / exposure)
```

```
## # A tibble: 1 x 3
##    exposure  deaths     CDR
##       <dbl>   <dbl>   <dbl>
## 1 50765887. 267692. 0.00527
```

7.2 Age-specific death rates

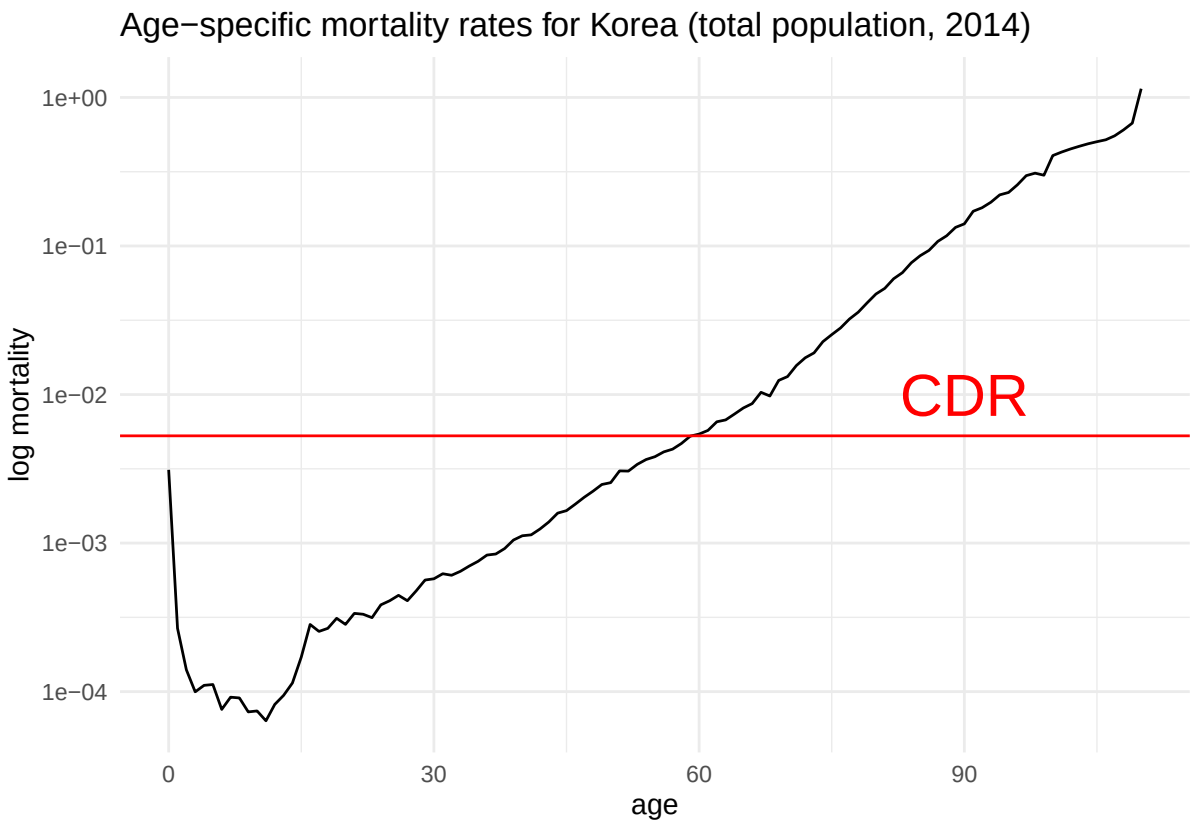
Age-specific death rates (M) measure the risk of death by age x (or between age x and $x+n$) in the population:

$$ {}_nM_x[0,T] = \frac{Number\ of\ deaths\ in\ the\ age\ range\ x\ to\ x+n\ between\ times\ T\ and\ T+t}{Number\ of\ person-years\ lived\ in\ the\ age\ range\ x\ to\ x+n\ between\ times\ T\ and\ T+t} $$

This measure controls for the effect of population age-structure, i.e. some populations have older age-structure than others resulting in higher CDR even if the ${}_nM_x$ are smaller at most ages. The age interval width n and calendar interval t are up to you and the data.

```
# Person-years by age


KOR2014 |>
  filter(sex == "total") |>
  mutate(M = deaths / exposure) |>
  ggplot(aes(x = age, y = M)) +
  geom_line() +
  scale_y_log10() +
  labs(title = "Age-specific mortality rates for Korea (total population, 2014)",
       y = "log mortality") +
  theme_minimal() +
  geom_hline(yintercept = 0.005273066, color = "red") +
  annotate("text", x = 90, y = 0.01, label = "CDR",color="red",size=8)
```

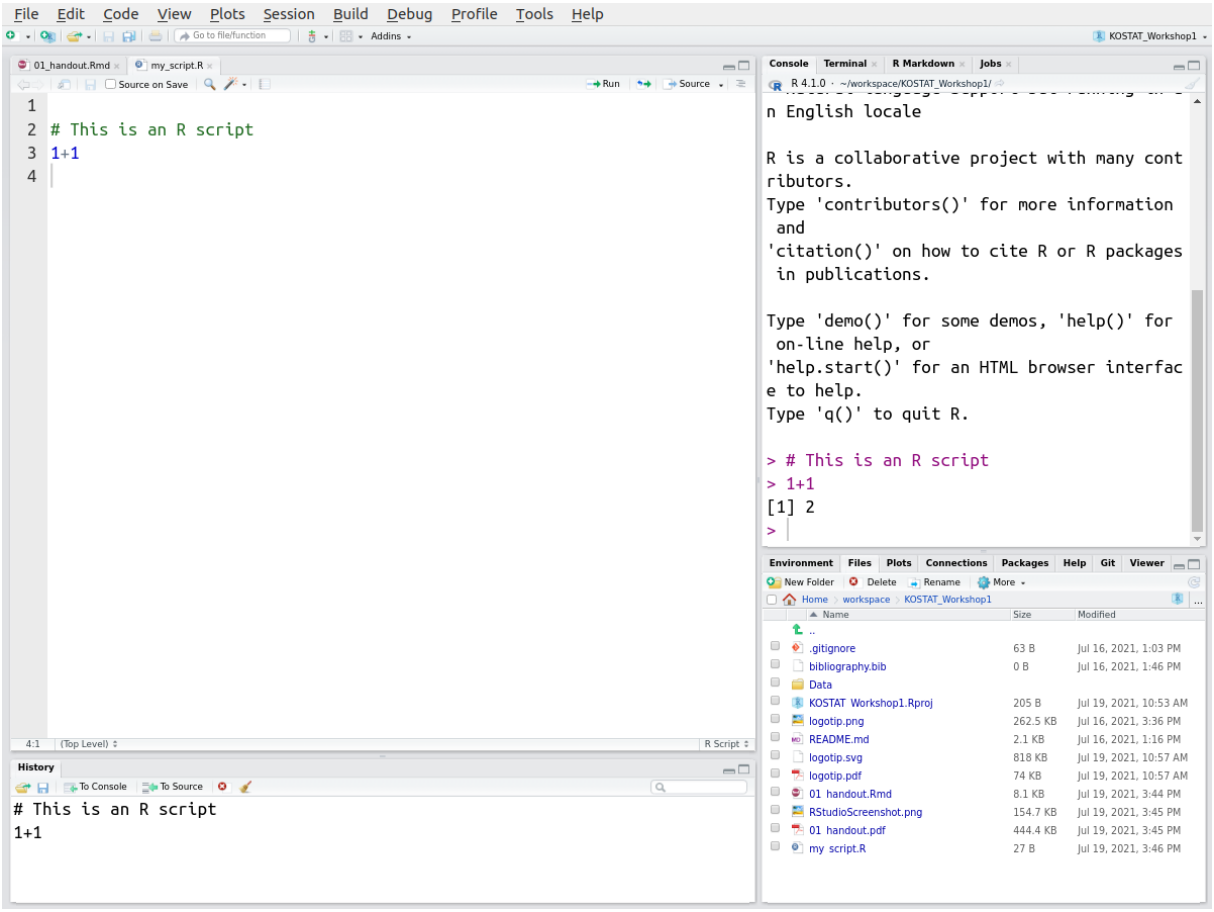


8 Practical

9 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 free desktop version.

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



9.1 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.

9.2 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.

9.3 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.

9.4 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.

10 Step-by-step get started

10.1 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 reproducibility 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* in RStudio, which is convenient for this workshop. You'll know the projcet is open properly if you see `KOSTAT_Workshop1` displayed in the upper border of RStudio.

10.2 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 :-)

10.2.1 *build* the template provided

The markdown 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.

10.2.2 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 create 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. There will be just a few such hot key combinations that you should commit to muscle memory for this workshop.

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

10.2.3 YAML header

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

```
---
title: "01_notes"
author: "Your name"
date: "7/24/2023"
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.

10.2.4 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 the document you build from markdown text, if you want to, but the default settings are just fine for note-taking in this workshop.

10.3 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.

11 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!

11.1 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. Popular packages will have been reviewed many times.

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)
```

11.2 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 reproducible 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`. When I'm *improv* coding, you may see me refer to the help files from time to time in order to make decisions on which functions to use and how to use them.

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.

11.3 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
```

```r
1 - 5
```

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

```r
2 * 5
```

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

```r
2 / 5
```

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

```r
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*.

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

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

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

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

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

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

```r
# 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 more involved expressions, making intelligent use of parentheses, just like in standard mathematics, standard *order of operations* applies. When coding, it improves legibility to use parentheses to group operations rather than relying on a reader to sort of order of operations.

```
(5 * .07) / (1 + (5 - 2.1) * .07)
```

```
## [1] 0.2909393
```

11.4 Objects

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

11.4.1 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"
```

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

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

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

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

```r
# 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.

Note, `a` can be used to create `b`, but once created, `b` no longer depends on `a`.

```r
b <- a * 2
# overwrite 'a'
a <- a * 3
a
```

```
## [1]  6 15 21
```

```r
b
```

```
## [1]  4 10 14
```

Likewise, if we remove `a` altogether, `b` is unaffected. From time to time, we'll do some housekeeping so that the session environment doesn't get cluttered. You can remove an object using `rm()`

```r
rm(a)

# b persists:
b
```

```
## [1]  4 10 14
```

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

```r
b <- c("A", "b", "C")
# notice how I skip 'c'? it's because it's a function already!
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

```r
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
```

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

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

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

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

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

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

```r
ncol(A)
```

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

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

```r
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 able to spend more time on data analysis.

11.4.2 `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)
DF <- data.frame(x, y)
DF
```

```
##   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`s are 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`, `tibble`s. You can treat these as synonyms.

11.4.3 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).

12 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 in future sessions. 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/Account/Auth. If you're already a user then no worries. We will use this at least once.

Hauser, Philip M., and Otis Dudley Duncan. 1959. *The Study of Population, an Inventory and Appraisal*. Chicago: University Press (London, Cambridge University Press.

Human Fertility Database. 2018. "Max Planck Institute for Demographic Research (Germany) and Vienna Institute of Demography (Austria)."

Human Mortality Database. 2018. "University of California, Berkeley (USA) and Max Planck Institute for Demographic Research (Germany)."

Preston, S, Patrick Heuveline, and Michael Guillot. 2001. "Demography: Measuring and Modeling Population Processes." *Malden, MA: Blackwell Publishers.*