Managing Demographic Big Data Analysis in R

Jon Minton, University of Glasgow

Introduction

Welcome to a one day course on managing demographic big data in R. This course will provide some examples of automating processes involving (fairly) large amounts of data, which we'll run and discuss over the course of the day.

The data we will use come from the Human Fertility Database (HFD) and the Human Mortality Database (HMD), both of which are run by the Max Planck Institute for Demographic Research along with other partners. As the names suggest, they both comprise demographic data, which I think of as 'old big data'. The HFD is around 0.2Gb in size; and the HMD around 1.3Gb. Compared with some of the data sources currently being generated and analysed – such as those which are generated when customers use a website, or when people use hardware and apps to track their activity on a second-by-second basis - these are tiny amounts of data; they are also much less 'messy', and require less data tidying and processing, than such forms of data. However, they are large by the standards of much of the academic social sciences, freely available, and more importantly substantively interesting and important. I therefore hope some of the analyses and results presented here will interest, engage and surprise, even though they are being used pedagogically in this case, to introduce a broader pattern and approach to data process automation.

The HFD and HMD

Both databases are produced by the Max Planck Institute for Demographic Research (MPIDR) in Berlin, Germany; with the HFD produced in conjunction with the Vienna Institute of Demography (VID) in Austria, and the HMD produced in conjunction with the Department of Demography at the University of Californa, USA.

Further information about both databases, including their methods of construction, and the range of measures, countries, and years covered, are provided in the links below:

- *Human Mortality Database*
- *Human Fertility Database*

When making international comparisons between countries, or comparisons over time, it is common to use simple summary statistics. In the case of population and death data, examples of these summary statistics include measures like period life expectancy, or crude or adjusted death rates for a whole population or sub-group. For fertility data, measures like total fertility rate might be used.

Demographic Summary Statistics

These summary statistics 'stand in place' for a much larger number of variables and values, and can sometimes be less informative than they first appear. For example, a period life expectancy for a country in the most recent year combines mortality risks at many different ages, and is likely to be an underestimate of how long most people alive can expect to live because it implicitly 'assumes' that long-term improvements in age specific mortality risks suddenly freeze.

In the figure in this short blog entry which I produced for the International Longevity Centre, this is equivalent to assuming that the contour lines, indicating specific mortality risks, which have been drifting rightwards, from younger to older ages, suddenly become fixed, and should be assumed to be vertical in the 'missing' triangle at the top right of each tile. This is clearly an implausible extrapolation given the trends observed, and *perhaps one reason why longevity improvements, and therefore pension liabilities, have often been consistently underestimated in rich world nations for many decades.*

The HFD and HMD both allow you to go beyond the summary statistics, to explore the components that go to produce them. Doing this in a way that's enjoyable and productive involves finding ways to reduce the friction involved in the workflow process. This means finding ways of getting R, a computer programming language, to work for you, and in particular to do repetitive and dull work so you don't have to.

Process automation

Process automation is important when there is a need to do the same thing, or almost the same thing, many times, quickly, and without substantial risk of human error. R is a statistical programming language, and like other programming languages allows many processes to be automated by specifying general sequences of operations to be performed to data in the form of functions. Functions take a particular type of input, 'work on' (process) that input in a particular way, then produce a particular type of output. If you can specify the process clearly enough, and in a general enough way, then the same function can be re-used many times, each time processing a different input in the same way. When there are many dozens, hundreds or thousands of inputs that all need the same 'work' doing to them, then using a function to process multiple inputs can save minutes, hours, days or even weeks of your time.

The vague but useful mantra of data science is often described as 'turning data into knowledge'. The input is 'data' and the output is 'knowledge' (or 'value', or 'information', or 'insight'), and 'data science' is the magical pipeline linking input to output. However, (to paraphrase Arthur C Clarke) data science isn't really magic, it's just an advanced data processing technology, made up of many other smaller and simpler technologies. Each of these simpler technologies can also be thought of as a 'function', with a specific input, process, and output. These functions chain together, with the output from one function becoming the input to another. Often, the chain of functions isn't even linear: the output from a later function in the chain can feed back to form an input to an earlier chain.

The technology isn't even always fully automated, and nor should it be. The ghost in the machine is you, the researcher, making important analytical decisions about how to get more knowledge and insight from the data, given what you already know. As you work with the data, you learn more, perhaps leading you to choose to rebuild the machine to produce new outputs, and generate new knowledge. Being the ghost in the machine, the wizard behind the curtains, can be fun. But only if the machine works smoothly and efficiently. Computational approaches to social science allow you to spend more time being a social scientist, running the machine, by ensuring you spend less time being part of the machine. For example, if the output from one process needs to be carried by wheelbarrow to the input to another process, the guy with the wheelbarrow is part of the machine. If dozens of different input pipes need to be connected manually to the same unit, the guy constantly screwing and unscrewing pipes is part of the machine. Computational social science aims to make you and your work much less robotic, by systematically removing the need for you to be your own knowledge labourer in your own knowledge-making system. However, doing this requires learning a number of methods and patterns for working holistically and systematically with data challenges. This one day course will introduce some of these key patterns.

Key stages in the data-to-value chain

There are often a great many intermediate steps involved, but often the core stages involved in working with large amounts of data are:

1. *Inputting 'raw' data*
2. *Producing 'tidy' data*
3. *Initial exploratory analyses*
4. *Producing summary statistics and visualisations for each of the inputs*

5. *Producing final results and outputs*

The first, second, and forth stage can all be automated, and doing this tends both to save time for stages three and five, and often to allow stages three and five to be performed more quickly and effectively. In the scripts used on this course, stage one and two are largely combined, but it is important to think about them as distinct activities. The same broad tasks will be performed using both the HFD and HMD, in order to make the broader pattern and process easier to identify.

HFD and HMD as case studies

In each of the two extended case studies, one using the HFD and the other using the HMD, three distinct types of process will be demonstrated:

1. **Initial data tidying and harvesting.** This involves turning 'raw data' into 'tidy data', which in turn involves understanding the structure and syntax of the input data files, and how convert these to a target 'tidy data' form.
2. **Exploratory analyses.** This involves developing a familiarity with the data and some of the information it contains. This is an important process for a more general, iterative, and fuzzy process of insight generation, based around the concept of the Research Flywheel, as well as being important both for error checking the results of stage 1, and for prototyping some of the patterns and processes which are then formalised and automated in stage 3.
3. **Automated output generation.** This involves realising that many tasks are variations on a theme, knowing what this theme is (i.e. what is common to the group of tasks), knowing how to describe the general pattern common to the tasks as an algorithm that a computer can follow, and specifying the particular groupings or variations over which the patterns should be applied.

Both stage 1 and stage 3 will involve making use of the new `purrr` package by Hadley Wickham, based around a functional programming approach to task automation. The `purrr` package is a successor to the `plyr` package, whose split-apply-combine paradigm is described in the paper 'The Split-Apply-Combine Strategy for Data Analysis'. A very closely related functional programming strategy is known as the MapReduce programming model, and is a key element of effective big data analysis.

The general approach: do first, learn later

Given this course is only a day long, but will involve combining various methods, approaches and paradigms that will probably be unfamiliar to many of you, the aim of the course will be to get you to implement recipes I have written for solving each of the three core stages above, using firstly the HFD, and then the HMD. This course will be about learning by doing: first running code that (hopefully!) works, then trying to unpick and unpack the components and elements of the code to understand how they solve the particular challenges involved at that specific stage of the case study, and then generalising from these specific solutions to more generic strategies for working in an efficient way with large amounts of data. Though a deep understanding and mastery of the approaches and how to implement them will not be achieved by the end of the day, the course is intended to provide both a place to start and a motivation to develop your knowledge of this approach further.

Mind Tool Number 1: Tidy Data

Data are defined as 'tidy' if they are in a rectangular format and:

1. Each variable forms a column
2. Each observation forms a row
3. Each type of observational unit forms a table

Source: 'Tidy Data'

The above definition is fairly opaque, and best understood by considering a number of examples, such as thos provided in the link above. Data which are 'tidy' according to the above definition tend to be easier for many R functions and procedures to work with, and so identifying when data are in a tidy format, and how to get the data to this format, is important for getting the data into a format where it is easy to use in many ways.

The way I approach tidy data is to think about two fundamentally different types of variable:

- **'Where' Variables:** these 'locate' and uniquely specify the characteristics of something about which some characteristics (such as height and weight) have been measured.
- **'What' Variables:** these are the measured characteristics.

The 'Where' variables that need to be stated in order to uniquely identify the observation about which the 'what' variables refer depend on the type of observational unit.

In the examples used in this course, the 'Where' variables typically include:

- Country (Represented by a standard code)
- Sex (HMD only)
- Age (in years)
- Year

And the 'What' variables include:

- Number of live births (HFD)
- Fertility exposure: Number of women 'at risk' of life birth (HFD)
- Death counts (HMD)
- Population counts (HMD)
- Mortality Exposure, i.e. average number of individuals 'at risk' of death (HMD)

(Note: It is important to consider and appreciate the difference between population counts and population exposure when working with demographic data. Especially if a demographer is reviewing your paper!)

The 'what' variables - Country, Sex, age and Year - uniquely specify the observation for which the other variables refer.

Further information and discussion about Tidy Data is provided in section 8.2 of the AQMeN Data Management Handbook. ______

Stage 1: Initial Data Harvesting and Tidying

This section will demonstrate how to harvest and 'tidy' data from many separate files in the HFD and HMD. The HFD and HMD have similar, but not identical, data structures and variables, and so require different code to perform this task. In each case the code is written with the same underlying objective and approach in mind. We will start with the HFD, as it presents a slightly simpler data management challenge.

Stage 1F: Initial Data Harvesting and Tidying using the HFD

Exercise 1

- Look at the contents of the directory `input_data/hfd`. This should contain a large number of separate datasets from the HFD, each stored as a text file with a `.txt` extension.
- Open up the files `birthRR.txt` and `exposRR.txt` in a text editor (such as in Rstudio), and think about:
- What information the two files contain
- What information you would need to tell R about the contents and format of the file in order for R (or any other programming environment/statistical package) to be able to read in the data files correctly.

Optional Exercise (After the course)

- We will be using both the `birthRR.txt` and `exposRR.txt` files to calculate age-specific fertility ratios (ASFRs). Look through the file contents to see if this information is already available. If so, load it as well and compare our calculated ASFRs with those provided by the HFD. Also look for files containing data disaggregated by Lexis triangles and consider the benefits and challenges of using these data files instead of the Lexis square based data used here.

Exercise 2

- Open the scripts `0_master_script.R` and `1_1_harvest_hfd.R`, both in the `scripts/` subdirectory.
- Run the code in `0_master_script.R` from the first line to the end of the function call to `pacman::p_load`. (approx line 42)
  - How can these packages be loaded in base R? What are the advantages and disadvantages of using the `pacman` package to load other packages?
- Look at the code in `1_1_harvest_hfd.R` and make some guesses about what each of the sections of the code do.
  - Is the line `rm(list = ls())` necessary in `1_1_harvest_hfd.R`? Why has it been included?
  - What does the `%>%` operator do? From which R package does it originate? How does it fundamentally change the way chained sequences of actions can be specified in R?
  - What are the arguments to the function `read.table`? Given what you know about the structure of the data being loaded, why have the arguments been set the way they have?
  - (Advanced) Why has `read.table` been used rather than an equivalent function within the `readr` package?
  - What is the name of the user-defined function in this code? What string manipulation function does it use, and for what purpose?
- Run the code section by section, paying attention to the new R objects that appear in the environment (visible in the top right pane).
  - print, view and glimpse the two input data objects.
- What does `inner_join` do and how does it know how to join the two data objects together?
  - From which package does the `inner_join` function come from? What other types of join operation are available? (Advanced) Compare the different join functions with `merge` in base R.
- Identify the output object. What is it called and what is its structure? How does this satisfy the tidy data paradigm?
- Complete running this script and identify the new file that has been created. Consider why this script should not be run every time you want to work with the data.

Mind Tool Number 2: Piped Code and the `%>%` operator

The artist Rene Magritte will forever be famous for a picture of a pipe, accompanied by the text 'Ceci n'est pas un pipe'. In late 2014 an R package called `magrittr` was published with one role: to bring 'pipe operators' into the R programming language. Pipe operators exist in a number of other programming languages and environments, in particular Unix, and allow commands to be linked together in intuitive ways. The pipes that pipe operators are based on, however, are not the elegant but harmful tobacco consumption devices favoured by Belgian surrealists and French philosophers, but the boring but indispensible fluid transfer conduits used by plumbers.

The main pipe operator used is the `%>%` operator. The packages `tidyr`, `dplyr`, `purrr`, and many others all make use of this operator to allow commands to be chained together, from left to right, and from top to bottom. Most of the functions in `tidyr` and `dplyr` are also named after verbs, like `select`, `group_by`, `nest`, and `filter`. The combination of piping and verb-named functions has meant that, very quickly, R code has become much more human friendly. Previously, complex operations either involved lots of temporary objects, or writing code that comprised functions within functions (within functions), meaning they had to be read and interpreted 'inside out'. Piping means that R code can now be written, and so read, from left to right, and from top to bottom, just like a written language.

For more information about piping using the `%>%` operator, please see the vignette on the `magrittr` package, and section five of the AQMeN Data Management handbook, amongst other sources.

Discussion for Stage 1F

Only a limited amount of additional work needed to be done to the two data files harvested in the above example to get them into a tidy data format to work with. The dividends of the fairly formal, automated approach introduced here are much larger in cases where more steps are required to get the data into an easy-to-use 'tidy data' format.

- *Advanced and optional exercise:* Imagine you were interested in calculating not just the distribution of births at different ages, over time and in different years, but the distribution of first births, second births, third births and so on. Which data file or files could be used to investigate this, and what additional steps (if any) would be required to convert this to a tidy data format?

Stage 1M: Initial Data Harvesting and Tidying using the HMD

The HMD is a somewhat larger dataset, and has a more complex file structure, than the HFD. The benefits of an automated data tidying and harvesting approach come into their own when working with datasets of this kind of file size and complexity.

Exercise 3

- Explore the contents of `input_data/hmd`. How does this compare with the structure and contents of `input_data/hfd`? (Pay special attention to where information pertaining to the same variable for different countries are held in both datasets, and why these differences mean the processes for harvesting and tidying data from the HMD are more complex than for the HFD.)
  - Within a single country subdirectory, identify the relative location of the data files `Population.txt`, `Exposure_1x1.txt`, and `Deaths_1x1.txt`. Look at the contents of these files, and consider what information you need to tell R about the file contents in order to load the data correctly.
  - With the Tidy Data paradigm and target form in mind, consider why it is necessary to include a variable indicating the country within an output dataset. Where can R find the requisite contents for this variable?

Mind Tool Number 3: nesting and map functions

One of the newest, trickiest, but also most useful approaches for automating tasks is to use code patterns that include the following:

```
DATAFRAME %>%
  group_by(GROUPING_VARIABLE) %>%
  nest() %>%
  mutate(OBJ_IN_DATAFRAME = map(.x = THING_TO_DO_SOMETHING_DO, .f = WHAT_TO_DO_TO_THAT_THING))
```

Understanding how this works involves understanding how R manages and defines dataframe objects, essentially as lists of objects, where each object can be of a different type but is of the same length. Typical object types include numbers and characters, but can also include other lists. As dataframes are in effect lists, dataframes can therefore potentially include other dataframes. Though this is an initially confusing and non-standard way of working with dataframes, this feature is exploited in `purrr` to allow the same process to be applied to different objects, but in a 'tidy' way that does not involve, for example, the creation of many separate objects in the R environment.

The key to doing this is the `map` function in the above. The `.x` argument tells the function where to look for the vector of objects to operate on, and the `.f` object indicates what function to apply to each object in the vector of objects specified in `.x`. By using the `mutate` function, the result of the `map` operation is passed to a new vector, `OBJ_IN_DATAFRAME`, which is added alongside other existing variables in `DATAFRAME`. Depending on the output produced by the function and the other objects in the dataframe, the `nest` and `unnest` function allow for multiple variables stored in separate vectors to be 'packed' into a single dataframe vector, or conversely for the contents of a dataframe vector to be 'unpacked' back into separate vectors. This complex behaviours take some getting used to, but can be an efficient way of performing many types of task which, fundamentally, involve performing the same operation on many distinct objects.

The `map` function will be used in the example below.

Exercise 4

- Open and start to look at the script `1_2_harvest_hmd.R`. (It's more complex than the previous example but don't be afraid!)
- The 'engine' in this script is a user-defined function, which itself contains two user-defined functions. What is the outer user-defined function called, and what are the two user-defined functions it contains called? How do you think the function works?
- Run the code from line 1 through to 7. Why is it important to save the output to list.files as a separate object?
- Run the code from lines 9 through 46. This is the 'engine' function. What new objects and activities have been performed at this stage?
- Run line 46 up to *but excluding** the pipe operator `%>%`. What might be the purpose of creating a dataframe that only contains one variable at this stage?
- Given what you know about the `mutate` and `map` functions, what do you expect to be the result of running lines 46 and 47 together, but again excluding the last `%>%` operator? i.e. of running:

```
data_frame(code = country_codes) %>%
  mutate(df = map(.x = code, .f = grab_and_reshape))
```

- How many columns do you expect the dataframe produced by the above to have? What will be their names and classes?

- Given your reading of the contents of the 'engine function', what do you expect the contents of the dataframe it returns to contain? What variable names?

- How do you expect the `unnest` function to change the number and type of variables in the dataframe produced after running the following?

  ```
  data_frame(code = country_codes) %>%
    mutate(df = map(.x = code, .f = grab_and_reshape)) %>%
    unnest()
  ```

- Run the complete code block, including the assignment operator to the object `all_data`.
  - Consider why you would not want to run the above code block step-by-step, without saving the outputs produced by assigning them to new objects in the environment. How might you go about testing the code without long delays without between runs? (Hint: how might changing `country_codes <- list.files(source_dir)` to something like `country_codes <- list.files(source_dir)[1:3]` help?)
- Run the last part of the script file which saves the `all_data` object. Identify the location of this file. Consider why the data have been saved as a text file rather than in another format.

Discussion for Stage 1F

Within this section a similar pattern has been used to harvest data from a wide range of separate files, to organise them in a tidy data structure, and then to save them as a text file which can then be used again without having to re-run the script file.

Now we have prepared tidy datasets from both the HFD and HMD, we can explore the data much more easily. In the exercises below we will start with some bespoke analyses to compare trends over time for different countries.

Stage 2F: Exploratory analyses of the HFD

The script `2_1_hfd_exploratory_data_analysis.R`, in the `scripts/` subdirectory, contains a series of exploratory analyses of the HFD dataset. Although in our case this dataset has already been loaded, as we have just created it, in a typical R session it will not be, and so the script begins by loading the dataset and assigning it to the object `dta_hfd`. After this object has been created, however, note that no further objects are stored in the R environment.

Exercise 5

- Open and start to look at the script `2_1_hfd_exploratory_data_analysis.R`.
- Load the dataset (2.1.1)
- Run the complete subsequent chunk of code, (2.1.2; lines 10-14).
  - Note the output produced. What, substantively, does this tell you?
- Now run this code chunk one line element at a time.
  - As before, start with the contents of the first line, up to but excluding the pipe operator `%>%`. Note the output displayed from doing this in the console.
  - Now re-run to include the pipe operator and the next instruction on the next line, again exluding the last pipe operator.
  - For this initial code chunk the code should be run as follows:

    ```
    dta_hfd
    ```

    And then

    ```
    dta_hfd %>%
      group_by(year)
    ```

    And then

    ```
    dta_hfd %>%
      group_by(year) %>%
      summarise(total_births = sum(total))
    ```

    And then

    ```
    dta_hfd %>%
      group_by(year) %>%
    ```

```
  summarise(total_births = sum(total)) %>%
  ggplot(., aes(x = year, y = total_births))
```

And then finally:

```
dta_hfd %>%
  group_by(year) %>%
  summarise(total_births = sum(total)) %>%
  ggplot(., aes(x = year, y = total_births)) +
  geom_line()
```

  - At each step in the above, predict what you think the output will be, and if the output produced is not what you were expecting, think about why this may be the case.
- Repeat the above pattern for each of the subsequent code chunks in the script, i.e.
  - Run the complete code chunk and reflect on the substantive implications of the results
  - *Predict* what you think the first executable element will do
  - *Compare* your expectations against the actual behaviour by running this element
  - *Repeat the process, with ever more elements* until you have run the complete code chunk
- Throughout, make notes of both the behaviour of the code, and code behaviours that are not as you were expecting.

Discussion for Stage 2F

The above exercise was provided without specific step-by-step instructions at a line-by-line level, but instead forms an introduction to a general process for both building and testing code chunks for exploratory data analysis using piping. Functions within the `dplyr` and to a lesser extent `tidyr` packages are made use of throughout, which as discussed previously tend to be verbs (`filter`, `select`, and so on).

Additionally, the graphics package `ggplot2` is made use of to explore and display the outputs produced through the analyses. Within `ggplot2`, the `+` operator is functionally similar to the `%>%` operator, in chaining different instructions together. `ggplot2` is a highly developed and widely used package for both exploratory and publication-quality data graphics, based around the Grammar of Graphics data visualisation approach advocated by the statistician Leland Wilkinson. Many guides and introductions to `ggplot2` exist and are easy to find. For those with some familiarity with `ggplot2` the RStudio Cheat Sheet on Data Visualisation with ggplot2 is an excellent resource for any office wall.

Stage 2M: Exploratory analyses of HMD

The basic pattern for running, writing and checking code has been established in the previous section. Some analogous code is presented in the script `2_2_hmd_exploratory_data_analysis.R`. This will look at both death rates and change in population structure in each of the many countries with records in the HMD.

Exercise 6

- Open and start to look at the script `2_2_hmd_exploratory_data_analysis.R`.
- As before, load the requisite dataset
- And as before, work through the code chunks, and for each code chunk
  - Run the code chunk in its entirity
  - Reflect on the substantive findings from running the chunk: What (if anything) do you now know about mortality and population structure that you did not know before?
  - Run the code chunk component by component:
    - First **predict** the output
    - Then **compare** the predicted with the actual output
    - Then **reflect** on why the prediction and the actual output are different, and what you need to learn more about to understand the code's behaviour.

Stage 2F: Discussion

In the exercise above we have reinforced the general workflow pattern for producing and testing code chunks that was introduced in Exercise 5. We have also gained substantive knowledge about international mortality and population trends. By harvesting and tidying the data from dozens of files within the HMD at the previous stage (1.2 - data harvesting), we have been able to compare data between countries very easily at this stage. Spending more effort and time to produce a functional programming solution to the data harvesting solution has therefore more than paid for itself in terms of much quicker and more pain-free analysis at this data exploration stage.

Stage 3: Data Vis and file outputing using HFD; and automated modelling using the HMD

In stage 1 functional programming approaches to process automation were an invaluable tool for harvesting data from multiple separate files. At stage 2, the benefits of bringing together data from these many separate files became apparent, as it allowed data to be compared between countries much more quickly than if the data for each file had to be loaded separately. The time taken to manually source and load the data from each of the files would quickly have become prohibitive, and become an impediment to the kind of friction-free curiosity based data exploration which stage 2 demonstrated for both the HFD and HMD databases.

The kind of exploratory data analysis showcased at stage 2 can be thought of as prototyping. At the the prototyping stage we aren't necessarily interested in the most methodologically robust or aesthetically pleasing models and graphics, but in quickly getting a sense of the data and the various patterns that they contain.

After exploring the data and prototyping various analyses, we should have a better idea of some of the most important patterns and findings that the data can reveal. We can move from the prototype to the final designs, whether they be more robust statistical models and outputs, or more polished graphics. As we want to be able to provide more resources to other researchers, and to demonstrate we were not simply 'cherry picking' in our choice of data and analysis, it can be helpful to automate the production of supplementary material. In the case of the HFD and HMD we might, for instance, want to write a paper which focuses on one or two countries, but also to make available comparable analyses and results for the other countries as well.

As in the first stage, process automation using functional programming languages can be very helpful at this stage too. In the two final series of exercises that follow, the HFD data are used to illustrate the process of automating the production of graphics and other file outputs, whereas the HMD data is used to illustrate how functional programming can be used to run and compare many regression models at once.

Stage 3.1: Data Vis and file outputing using HFD

Exercise 7

- The basic pattern for working through the code is exactly as before. For the HFD-based example, focusing on the production of graphics and other file outputs, open and start to explore the script `3_1_hfd_data_vis_and_output.R`, again in the `scripts/` directory.
- As before, load the requisite dataset
- And as before, work through the code chunks, and for each code chunk
  - Run the code chunk in its entirity
  - Reflect on the substantive findings from running the chunk: What (if anything) do you now know about mortality and population structure that you did not know before?
  - Run the code chunk component by component:
    - First **predict** the output
    - Then **compare** the predicted with the actual output

        * Then **reflect** on why the prediction and the actual output are different, and what you need to learn more about to understand the code's behaviour.

Note: The script above will produce a series of additional files in the directories `figures/` and `tidied_data/`. If you have time, please explore the contents of these graphics and text files, and start to think about some of the substantive implications of the outputs produced. Just because we have automated the production of data outputs does not mean we should attempt to automate our intuitive thinking and understanding of what these outputs appear to show.

Stage 3.2: Model automation and testing using HMD data

Exercise 8

- The basic pattern for working through the code is exactly as before. For the HMD-based example, focusing on how we can use functional programming from `purrr` along with linear regression functions from `stats` (in Base R) and `broom` for tidying model outputs, open and start to explore the script `3_2_hmd_stat_analyses.R`, again in the `scripts/` directory.
- As before, load the requisite dataset
- And as before, work through the code chunks, and for each code chunk
    - Run the code chunk in its entirity
    - Reflect on the substantive findings from running the chunk: What (if anything) do you now know about mortality and population structure that you did not know before?
    - Run the code chunk component by component:
        * First **predict** the output
        * Then **compare** the predicted with the actual output
        * Then **reflect** on why the prediction and the actual output are different, and what you need to learn more about to understand the code's behaviour.

Stage 3 discussion

The types of process automation demonstrated using both the HFD and HMD datasets showed how processes which otherwise would involve a large amount of time, code and effort, can be greatly reduced in terms of both time and effort. Although the `nest` and `map` functions, and the functional programming paradigm they operationalise, are likely unfamiliar approaches, and to start with may involve some head scratching and mental gymnastics, they are neat and effective ways to handle a large number of processes and operations that involve performing 'variations on a theme'.

Conclusion: where to go next

This course has provided a whistlestop tour through a complete data analysis workflow, employing and combining range of linked techniques and ideas in order to get from raw data, to substantive knowledge and insight in an efficient way. Although you may not understand **how** some of the code examples work at this stage, through repeating the same steps through each of the exercises you hopefully now have enough familiarity with the general patterns and habits that I recommend for developing code within a broader framework and philosophy of data analysis. This broader habit and workflow, together with an enthusiasm to learn and practice more, is the main take-home message from today's workshop.

Repeated use of 'tidyverse' packages, including `tidyr`, `dplyr`, `broom` and `purrr`, along with more formal training in functional programming, will make you more attuned to identifying broader patterns for writing and developing code as part of efficient data analysis workflows. An excellent introduction to functional programming, hosted by not just one but two Wickham siblings, is provided by DataCamp in the course named Writing Functions in R. Even if you have used R for many years I would recommend this course without reservation.

For a more general and comprehensive introduction to efficient workflows in R, I would recommend the AQMEN Social Data Science handbook, which can be downloaded along with course material by clicking on the button marked `Clone or download` here. **(N.B. I have recently been commissioned to write a methods book based on this material, so your feedback and suggestions would be very welcome.)**

Garrett Grolemund and Hadley Wickham's upcoming book, R for Data Science, is currently being written within RStudio using bookdown. Draft versions of this book are frequently updated and made available online.

RStudio has become the standard environment through which R is run, and with good reason. Amongst other free services provided by RStudio are a series of webinars covering a wide range of packages and approaches.

I hope you have enjoyed this course and it will mark the start of a journey through social data science that is long, productive, and most of all fun!

Bon Voyage!

Appendices

Data Science Principles (from the AQMeN Data Management Handbook, pp 8-9)

Unlike statisticians, data scientists are generalists, concerned with the complete data-to-knowledge value chain:

1. *The initial generation of quantitative data records;*
2. *Cleaning, standardising and tidying the data records;*
3. *Statistical analysis;*
4. *Evidence-based decision making.*

By contrast, statisticians tend to be, or at least to start off as, specialists focused on stage (3) of the above, adept with understanding and applying statistical theories and concepts to particular types of data, prepared as datasets which have been constructed in particular ways. The datasets that most statistics courses provide to students are 'tidy' (a term I'll define more clearly later on), and typically what is taught in such courses is how to analyse the data in this format. However, routine and administrative data seldom emerges in a tidy data format, ready to be loaded up and analysed in a statistical package. Instead, the data needs to be prepared and processed in a large number of ways. For example:

- *characters may need to be removed from fields;*
- *rows and columns may need to be combined;*
- *tables may need to be joined;*
- *derived variables may need to be generated;*
- *typos need to be identified and fixed;*
- *information ('metadata') about the types of variables (logical, categorical, ordinal, or cardinal) may need to be passed to formally specified in particular software*
- *et cetera, et cetera, et cetera*

Although researchers using quantitative data are generally motivated to use such data by stage (4), the production of knowledge and making good evidence-based decisions, a great deal of the time spent doing quantitative can be spent at stage (2). Often, stage (2) does not just take up 'much' of the time, but most of the time. When the 'base metal' is routinely collected administrative data, the production of tidy data often takes much longer than the statistical analysis.

The purpose of this course is to provide a series of tools, both conceptual and practical, which make stage (2), the management and tidying of administrative data, much quicker and easier to do. The reason for going into *more depth about the concepts and practice of data management is, paradoxically, because data management is not interesting. The more of your time you spend on data management issues, the less time you have to analyse the data, and to make informed decisions about the data. Conversely, if you have a series of tools and concepts at hand for managing data efficiently, you can pass through this stage more quickly, and spend more of your time at stages (3) and (4).*

Research Friction and the Research Flywheel (From the AQMeN Data Management Handbook, p 83)

The main reason for learning these tools, patterns and techniques is to greatly cut down the friction involved in exploring and understanding social phenomena through data analysis, and the ultimate reason for this is a belief that the relationship between evidence and theory should be bidirectional: looking at data makes it easier to develop simple (lower 'middle range') theories, and developing and testing these theories means looking again at the data, leading to the refinement or replacement of existing theories and the development of new theories, which in turn require further analysis of data, to new theories, to new data analysis, and so on and so on. Although a bunch (to use the in this case useful Americanism) of lower middle range theories may have enough in common to warrant being gathered together to form a more general middle middle range theory, and conversely a middle middle range theory may need to be separated (or 'operationalised') to form a series of empirically testable lower middle range theories, the process of generating knowledge and insight from data is often in practice about ensuring that the research flywheel keeps rotating smoothly from data to theory and back again. Effective data management practices and tools exist to allow for smoother running of this flywheel, and through this for better generation of knowledge and insight about social, economic, and health processes.
