Data Wrangling for EDSDers

Flat files

13-16 Nov, 2023

Tim Riffe
Universidad del País Vasco & Ikerbasque (Basque Foundation for Science)

14 Nov, 2023

Introduction

Flat files refer to large fixed-width format text files that typically contain information on birth, death, and marriage and similar registries. These are commonly produced by Spain, USA, Mexico, and others. Anyone know of other countries that openly share data in this form? Here are two examples:

1. USA https://www.cdc.gov/nchs/data_access/vitalstatsonline.htm
2. Spain ine.es This link for mortality, but there are other pages for marriage, natality, and other things.

Download the USA 2022 Natality file and save it in the `Data` folder (around 200Mb when zipped). It will take up much more memory when we read it in. We'll just read in selected columns and then do fun things from there. This is really quite fast and easy to do using the `readr` package and browsing the data documentation.

Load packages

```
library(tidyverse)
```

If you're daring you can download it from `R` like so, note you need to give it more than the default 60 seconds to download, so we increase the time allowed using `options(timeout = 400)`- you may need more time depending on your connection speed. I just copy-pasted this url. If you download it manually, and prefer that, that's OK, just extract the text file and run the last bit of code here to g-zip it, then delete the text file.

```
library(R.utils)
library(here)
file_url <- "https://ftp.cdc.gov/pub/Health_Statistics/NCHS/Datasets/DVS/natality/Nat2022us.zip"

options(timeout=400)
download.file(url = file_url,
```

```
              destfile = "Data/nat2022us.zip",
              mode = "wb",
              overwrite = TRUE)

# now unzip it (it's ~4.6Gb!)
# we use here() because this function apparently wants absolute file paths
unzip(zipfile = here("Data/nat2022us.zip"),
             exdir = here("Data"),
             overwrite = TRUE,
      # TR: I needed to tell it where my unzip program was...YMMV
             unzip = Sys.which("unzip"))

# now re-save as .gz so we can read it directly. This function
# comes from R.utils package...
gzip("Data/nat2022us.txt")

# and go ahead and delete the uncompressed text file and the zip version..
unlink("Data/nat2022us.txt")
unlink("Data/nat2022us.zip")
```

Reading a fixed width file

Give `read_fwf()` a whirl. This worked on the first try after glancing at the options in the help file (type `?read_fwf` in the console).

```
pos <- fwf_positions(
         start = c(9, 13, 475, 446, 75, 147, 504),
         end = c(12, 14, 475, 446, 76, 148, 507),
         col_names = c("year", "month", "sex","apgar","mage","fage","bwt"))

NAT <- read_fwf("Data/nat2022us.txt.gz",
  col_positions = pos,
  col_types = "iiciiii")

print(object.size(NAT), units ="Mb")
```

```
## 112.2 Mb
```

The pre-step `fwf_positions()` is a helper function to tell `read_fwf()` where the columns are that we want to extract. This is the quickest and lightest option, since we don't have to read in the whole dataset, which would max out memory on machines under 4Gb of memory. I got the column positions (`start` and `end` vectors) from the pdf documentation that one can download alongside the data. If the providers were a bit more polite they would provide the column positions in a spreadsheet, but this wasn't so arduous. By reading in just columns we're interested in the data object is kept lighter and easier to work with. Data like this really shouldn't cause hangups when using these tools. There are other tricks one can use for truly large data, but we've not got into those.

Redistribute missing values

Check missing apgar: 16248 cases.

```
NAT |>
  filter(apgar == 5) |>
  nrow()
```

```
## [1] 16248
```

Check missing birth weight, 3084 cases (everything else is grams)

```
NAT |>
  filter(bwt == 9999) |>
  nrow()
```

```
## [1] 3084
```

For the exercises I have in mind it's probably fine to throw these cases. But we could also redistribute them. In this case, how? There are people who dedicate themselves to this question. If we're lazy we might do so proportional to the distribution of births matched to all the characteristics they have in common. That means we could create a big multidimensional probability distribution of time of day for the combination of knowns for each birth with unknown time of day. It's laborious, maybe also semirigorous. Even more rigorous would be to include information on other variables you might not be interested in for the redistribution. But the more things you include in the redistribution the smaller the cells get, and it ends up being a bunch of noise and 0s. In that case, you might want to smooth the distribution used to redistribute unknowns, and doing so already puts things in a more statistical setting, where it's best to take advice from an actual statistician. For problems of this kind, and large count data like these, it's common to use one or two variables for such redistributions. Dudel and Klüsener (2018) does something similar for missing ages of fathers (rescale by the distribution of known fathers age within each mother age). IMO this is better than imputing means. Before such an exercise we'd want to do a tabulation, which will make these data quite a bit lighter.

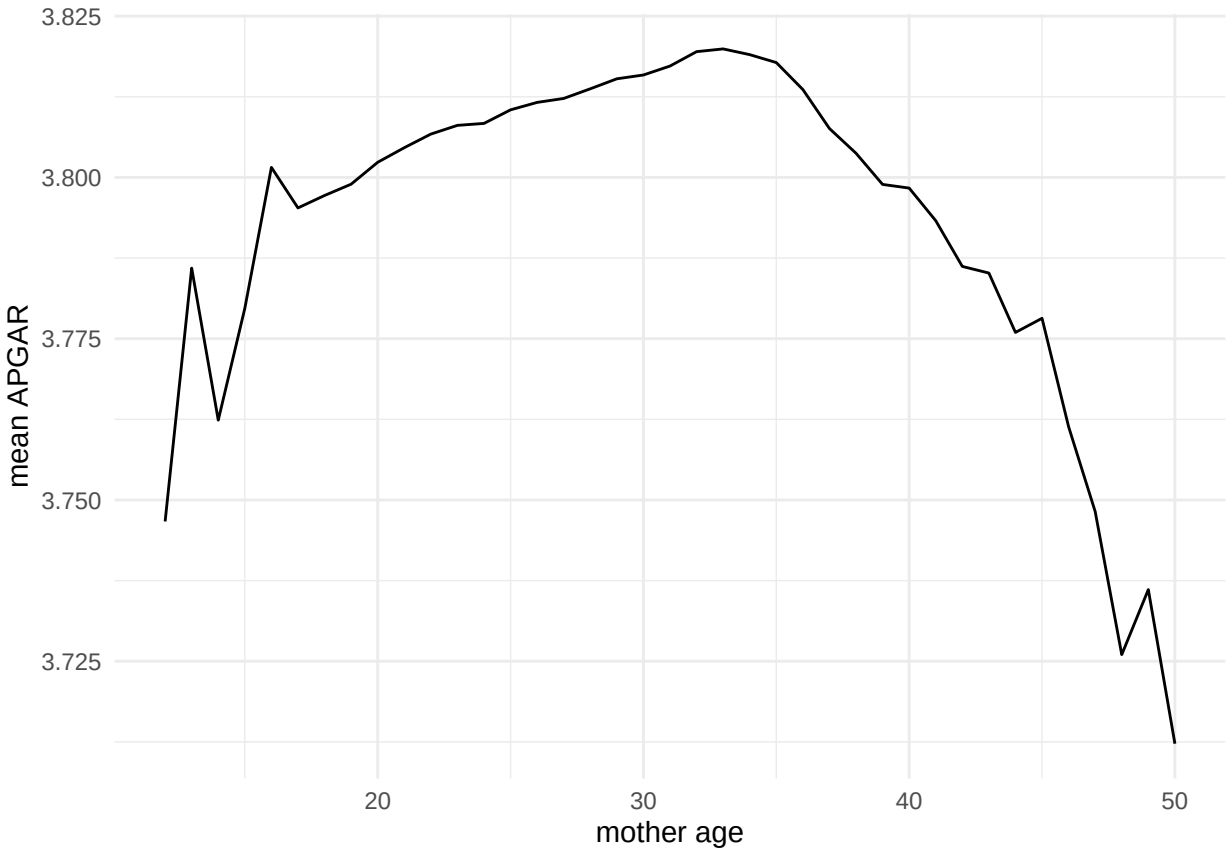
Let's redistribute missing APGAR, assuming that the distribution of births with stated APGAR depeds on mother's age only (probably there's some other covariate in the original data that explains this situation perfectly).

```
APGAR <- NAT  |>
  # tabulate by mother age, sex of baby, and apgar
  group_by(mage, sex, apgar) |>
  summarize(n = n(),
            .groups = "drop") |>
  # whatever we group by is what we redistribute within
  group_by(mage) |>
  # what is the fraction of births in each time-of-day within each mother age?
  mutate(B_total = sum(n)) |>
  filter(apgar != 5) |>
  mutate(b_frac = n / sum(n),
         births = B_total * b_frac)
```

Visualize it

Now we've redistributed births of unknown APGAR according to the distribution of known APGAR within each mother age. The *within* part of this was dealt with by the `group_by(mage)` statement, where one could add another variable or two to make it finer. Let's check whether mean apgar changes by mothers' age.

```r
APGAR |>
  # mutate(mage = mage - mage %% 5) |>
  group_by(mage) |>
  summarize(m_apgar = sum(apgar * births) / sum(births),
            .groups = "drop") |>
  ggplot(aes(x = mage, y = m_apgar)) +
  geom_line() +
  theme_minimal() +
  labs(x = "mother age", y = "mean APGAR")
```



Exercises:

1. Does the above outcome also show a concave pattern over fathers' age (`fage`)?
2. I want you to make a *ridge* plot of the birth weight distribution by mothers' age, where mother age is in 5-year bins, and we have two distributions shown per age: the boy birthweight distribution and the girl birthweight distribution. The steps:

2.1 Redistribute unknown / unstated birthweights using the same assumption as for APGAR, but in **5-year age groups** and by **sex** of birth. You can define 5-year age groups using this trick:

```r
NAT |>
  mutate(age = age - age %% 5)
```

- what do you need to `group_by()`?
- what action to you do inside summarize?
- do you maybe also want to bin birthweight?

2.2 calculate the fraction in each birthweight bin by mothers' age and sex of birth. Call it `b_frac`.

2.3 create the *ridgeplot* (you might need to install `ggridges`). You'll want to map birthweight to `x`, a mothers' age (as a factor) to y, and `b_frac` maps to `height`, `fill` maps to `sex`. Then the geom you want is called `geom_ridgeline`. You'll probably need to example in the help page or online to get this to work, that's just fine.

References

Dudel, Christian, and Sebastian Klüsener. 2018. "Estimating Men's Fertility from Vital Registration Data with Missing Values." *Population Studies*, 1–11.