KOSTAT-UNFPA Summer Seminar on Population

Workshop 1. Demography in R

Day 6: Processing and visualizing South Korean fertility microdata

Instructor: Tim Riffe
tim.riffe@ehu.eus

Assistants:
Jinyeon Jo: jyjo43043@gmail.com
Rustam Tursun-Zade: rustam.tursunzade@gmail.com

3 August 2022

Contents

1 Summary

Today we will exercise concepts previously presented to process and analyse births microdata delivered in *flat* files or *fixed-width* format. This means that data have no delimiters, but each column has a fixed character width. We will see that this data is easy and fast to read in bulk, and we will spend most of the session aggregating, merging with population data, and visualizing various kinds of results.

2 Packages we'll use today

```
#install.packages("tidyverse")
## for standardized column names
#install.packages("janitor")
## for easy date handling
#install.packages("lubridate")
## for ridge plots
#install.packages("ggridges")
## read in data fast!
#install.packages("vroom")
## because country codes are a mess!
#install.packages("countrycode")
## we need this to install a package from github!
# install.packages("remotes")
library(tidyverse)
library(janitor)
library(lubridate)
library(ggridges)
library(vroom)
library(colorspace)

# for merging with population counts
library(countrycode)
```

```
library(remotes)
# run once!
# install_github("PPgp/wpp2022")
library(wpp2022)
```

3 Fixed width microdata

KOSTAT kindly provided the birth data for the purpose of this workshop. Each observation in this data is an individual birth. These data contain 100% of registered births in Korea, and a selection of the many variables available from the original source. Typically these data are only accessible under restricted conditions after an application process. I will provide a link to a zipped folder containing the data to be used today, but these **should be deleted** when the workshop is over. If you require access to these data for research on Korean fertility, you should apply through the standard channels at KOSTAT.

Note that data of this exact kind is openly available for a few countries around the world, including but not limited to Spain and the United States

Initially, KOSTAT kindly provided the data to me already in comma-separated format, which we already know how to read into `R`. I put it (back?) in fixed-width format myself. If you're curious to see how I did that, and what I did to test today's code, you can refer to the supplementary handout `06_data_prep.pdf`. You will not be able to execute the `.Rmd` file because you do not have the original `.csv` files.

3.1 Download the data

Download the data (`Korea_births_fwf.zip`) using the link given in the `Google Doc` for the course (on the session day), and move it to the `Data` folder of the workshop project. Once there, you can unzip the file from `R` like so:

```
unzip("Data/Korea_births_fwf.zip",
      # in case this isn't the first time
      overwrite = TRUE,
      exdir = "Data")
```

A new folder will be created: `Data/Korea_births_fwf`, which contains one file each for years 2000-2020. Files follow the naming convention `kbyyyy.txt`, where `yyyy` is the given year. You can open one of these in a text editor and have a look. For some columns, in some places it seems obvious what a variable is, but in general we see that some sort of parsing is needed in order to get this data into any program.

Here is a glimpse of the first several rows of the first fixed-width file:

```
2000    2000    1    1       12000    1 36         34         35     2.36
2000    2000    1    1       12000    1 27         27         37     2.4
2000    2000    1    1       12000    1 25         25         37     2.9
2000    2000    1    1       12000    1 29         26         38     3.49
2000    2000    1    1       12000    1 31         24         39     3.16
2000    2000    1    1       12000    1 35         29         39     3.24
```

3.2 Read the data into `R`

The `readr` package has a nice function `read_fwf()` to read in this sort of data. It requires you to either know the the starting position and width of each column (in terms of character

positions), or the starting and stopping positions. If all you know are the widths of each column, then you must know these for each column in the data. When we read the data in, white space is automatically eliminated, and `R` guesses the data type for each column (which we can of course override if we want). Look at the help file (`?read_fwf`) for examples of all the different ways to read in fixed-width data.

3.2.1 Create metadata

In this case, you can read in the metadata directly from a `.csv` file in github like so:

```
# cutting url in two pieces because it's so long it won't fit on a page!
url_base <- "https://raw.githubusercontent.com/timriffe/KOSTAT_Workshop1/"
url_end  <- "master/Data/Korea_births_fwf_metadata.csv"
my_url   <- paste0(url_base, url_end)
widths <- read_csv(my_url,
                   # suppress messages
                   show_col_types = FALSE)
```

Let's have a look:

```
widths
```

```
## # A tibble: 11 x 2
##    colname                    width
##    <chr>                      <dbl>
##  1 Report year                    7
##  2 Year                           7
##  3 Report month                   4
##  4 Report date                    7
##  5 Sex                            1
##  6 Year of birth                  7
##  7 Month of birth                 2
##  8 (Father) Detailed age         10
##  9 (Mother) Detailed age         10
## 10 No. of weeks of pregnancy      6
## 11 Birth weight                   6
```

From this we see we have 11 columns, and we know the width of each. These should add up to the full width of a given line in the data. To use `read_fwf()`, we just need to convert this into some standard *begin-end* specifications, like so:

```
specs <-
  fwf_widths(widths = widths$width,
             col_names = widths$colname)
specs
```

```
## # A tibble: 11 x 3
##    begin   end col_names
##    <dbl> <dbl> <chr>
##  1     0     7 Report year
##  2     7    14 Year
##  3    14    18 Report month
##  4    18    25 Report date
##  5    25    26 Sex
##  6    26    33 Year of birth
```

```
##  7    33    35 Month of birth
##  8    35    45 (Father) Detailed age
##  9    45    55 (Mother) Detailed age
## 10    55    61 No. of weeks of pregnancy
## 11    61    67 Birth weight
```

This translation from widths should be pretty straightforward. Sometimes your metadata is already delivered just like this, with start and stop positions, ideally in a spreadsheet, but sometimes unfortunately in a `.pdf`. Even so, it's good to create this `R` object using one of the helper functions listed when you type `?fwf_widths` to ensure that it follows the standards anticipated.

Note: *begin-end* specification is the most flexible. Sometimes data of this kind is very large and you don't need all the columns. Using this specification, you could include only those columns that you need, as so save some of your machine memory :-)

3.2.2 Trial for one file

Let's see what we get with just one file. With this we can eyeball the column classes and explicitly specify these when we read them all in at once.

```
# read_fwf() comes from readr package
data_in <- read_fwf("Data/Korea_births_fwf/kb2000.txt",
                    col_positions = specs)
data_in
```

```
## # A tibble: 640,089 x 11
##    Report ~1  Year Repor~2 Repor~3   Sex Year ~4 Month~5 (Fath~6 (Moth~7 No. o~8
##        <dbl> <dbl>   <dbl>   <dbl> <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
##  1      2000  2000       1       1     1    2000       1      36      34      35
##  2      2000  2000       1       1     1    2000       1      27      27      37
##  3      2000  2000       1       1     1    2000       1      25      25      37
##  4      2000  2000       1       1     1    2000       1      29      26      38
##  5      2000  2000       1       1     1    2000       1      31      24      39
##  6      2000  2000       1       1     1    2000       1      35      29      39
##  7      2000  2000       1       1     1    2000       1      29      28      40
##  8      2000  2000       1       1     1    2000       1      34      27      40
##  9      2000  2000       1       1     1    2000       1      29      25      40
## 10      2000  2000       1       1     1    2000       1      41      35      40
## # ... with 640,079 more rows, 1 more variable: `Birth weight` <dbl>, and
## #   abbreviated variable names 1: `Report year`, 2: `Report month`,
## #   3: `Report date`, 4: `Year of birth`, 5: `Month of birth`,
## #   6: `(Father) Detailed age`, 7: `(Mother) Detailed age`,
## #   8: `No. of weeks of pregnancy`
## # i Use `print(n = ...)` to see more rows, and `colnames()` to see all variable names
```

- NOTE: in practice, file formats change every few years. You might start with some metadata for ranges of years, which then switches, or you may have unique metadata for each year. In that case, you'd need to repeat the above as much as needed, and it's a bit more laborious. But you'll of course still want to write the code to do it, which will make the task repeatable.

3.2.3 Read and merge all files using `vroom`

To read in the file, use `vroom_fwf()`, and feed our `specs` object to the `col_positions` argument. Here I explicitly state what the data type should be for each column using a shorthand: `i` stands for integer, and `d` stands for double (data with decimals). If you omit this step, then `Birth weight` will be interpreted as character. That's not really a problem, you could coerce it back to double using `mutate(Birth weight= as.double(Birth weight))`. We use `vroom` because it's blazing fast compared to other options. In our case, we have identically formatted files in a folder, and it just reads them all in at once.

```
rm(data_in) # remove earlier test file from memory
# a vector of file paths:
kb_folder <- "Data/Korea_births_fwf"
files <- file.path(kb_folder,
                   dir(kb_folder))

# read and merge, very fast :-)
KB     <- vroom_fwf(files,
                    col_positions = specs,
                    col_types = "iiiiiiiidddd")
# str(KB)
head(KB)
```

```
## # A tibble: 6 x 11
##   Report y~1  Year Repor~2 Repor~3   Sex Year ~4 Month~5 (Fath~6 (Moth~7 No. o~8
##        <int> <int>   <int>   <int> <int>   <int>   <int>   <dbl>   <dbl>   <dbl>
## 1       2000  2000       1       1     1    2000       1      36      34      35
## 2       2000  2000       1       1     1    2000       1      27      27      37
## 3       2000  2000       1       1     1    2000       1      25      25      37
## 4       2000  2000       1       1     1    2000       1      29      26      38
## 5       2000  2000       1       1     1    2000       1      31      24      39
## 6       2000  2000       1       1     1    2000       1      35      29      39
## # ... with 1 more variable: `Birth weight` <dbl>, and abbreviated variable
## #   names 1: `Report year`, 2: `Report month`, 3: `Report date`,
## #   4: `Year of birth`, 5: `Month of birth`, 6: `(Father) Detailed age`,
## #   7: `(Mother) Detailed age`, 8: `No. of weeks of pregnancy`
## # i Use `colnames()` to see all variable names
```

```
dim(KB)
```

```
## [1] 9369102      11
```

It appears there were 9369102 births registered in South Korea between the years 2000 and 2020, inclusive.

3.3 Tidy the data

3.3.1 Simplify names and create dates

First, the names are a bit awkward to type out in back-tics whenever we want to refer to a variable in code. I'd prefer to type `mother_detailed_age` rather than `(Mother) detailed age`. Even better would be `mo_age`! Also, in various places we'll want time coded as a date. For this we can use a helper function from the `lubridate` package, `ymd()`, you can give it strings like `"2000-01-01"` or `"2000 01 01"` or `"2000 1 1"`, and it'll interpret them properly and coerce to date data type. See also `dmy()`, `mdy()`, `as_date()`, and others.

```
KB <-
  KB %>%
  clean_names() %>%
  # date useful for plotting
  mutate(date = ymd(paste(year_of_birth,
                     month_of_birth,
                     "01")))
```

Check values to infer missing codes: no missing month or year of birth variables; 999 used for missing ages. `floor()` is used to *round down* age to the nearest integer, which is the way demographers most commonly handle age.

```
KB %>% pull(month_of_birth) %>% table()
```

```
## .
##      1      2      3      4      5      6      7      8      9     10     11
## 901506 788091 857452 792774 772433 722480 753519 767578 790556 788585 741032
##     12
## 693096
```

```
KB %>% pull(year_of_birth) %>% table()
```

```
## .
##   2000   2001   2002   2003   2004   2005   2006   2007   2008   2009   2010
## 640089 559934 496911 495036 476958 438707 451759 496822 465892 444849 470171
##   2011   2012   2013   2014   2015   2016   2017   2018   2019   2020
## 471265 484550 436455 435435 438420 406243 357771 326822 302676 272337
```

```
# code 999 is unstated age
KB %>% pull(father_detailed_age) %>% floor() %>% table()
```

```
## .
##     14     15     16     17     18     19     20     21     22     23     24
##      4     54    323   1304   3441   7739  11215  14348  23573  43143  71183
##     25     26     27     28     29     30     31     32     33     34     35
## 113325 184001 303389 433862 578180 734044 845787 898129 878155 816882 722995
##     36     37     38     39     40     41     42     43     44     45     46
## 610632 489385 386495 296729 224291 163880 118287  86745  62293  44291  30943
##     47     48     49     50     51     52     53     54     55     56     57
##  22030  15488  10898   7837   5621   3876   2896   2191   1565   1104    879
##     58     59     60     61     62     63     64     65     66     67     68
##    681    467    330    245    182    140    128     65     44     44     31
##     69     70     71     72     73     74     75     77    999
##     26      9     14      3      3      1      4      1  93252
```

```
KB %>% pull(mother_detailed_age) %>% floor() %>% table()
```

```
## .
##     12     13     14     15     16     17     18     19     20     21     22
##      2     65    406   1175   3278   7183  14343  33591  51136  74917 107804
##     23     24     25     26     27     28     29     30     31     32     33
## 156246 242367 331512 476019 651780 801405 890251 926320 905958 826404 705346
##     34     35     36     37     38     39     40     41     42     43     44
## 578013 458694 349743 256294 182256 124182  81876  50389  29765  16247   8350
##     45     46     47     48     49     50     51     52     53     54     55
```

```
##    4001    1933     961     580     392     263     182     109      77      52      52
##      56      57      58      59      60      61      62      63     999
##      33      16       9       6       2       2       1       2   17112
```

For some of our explorations, we can throw out unknown ages, and for others we should redistribute them. When we redistribute unknown ages, this usually follows tabulation rather than precedes it.

3.3.2 Convert age to integer and simplify names even more

For the sake of simplified typing, I think we'd prefer to have `fa_age` rather than `father_detailed_age`. Let's rename some columns, and also select only those that we need (since the data are somewhat big) using `select()`. Also for ages, we won't need anything more than integer precision, and, we can replace `999` codes with `NA` (for this we use a helper function `na_if()`. We prefer to use actual `NA` values so that they're easier to remember and detect whenever doing arithmetic things with age itself.

```
KB <-
  KB %>%
  # rename and select columns
  select(date,
         month = month_of_birth,
         year = year_of_birth,
         fa_age = father_detailed_age,
         mo_age = mother_detailed_age,
         sex,
         p_weeks = no_of_weeks_of_pregnancy,
         birth_weight) %>%
  # coerce age to integer
  mutate(mo_age = floor(mo_age),
         fa_age = floor(fa_age),
         # replace 999 with NA
         mo_age = na_if(mo_age, 999),
         fa_age = na_if(fa_age, 999))
head(KB)
```

```
## # A tibble: 6 x 8
##   date       month  year fa_age mo_age   sex p_weeks birth_weight
##   <date>     <int> <int>  <dbl>  <dbl> <int>   <dbl>        <dbl>
## 1 2000-01-01     1  2000     36     34     1      35         2.36
## 2 2000-01-01     1  2000     27     27     1      37         2.4
## 3 2000-01-01     1  2000     25     25     1      37         2.9
## 4 2000-01-01     1  2000     29     26     1      38         3.49
## 5 2000-01-01     1  2000     31     24     1      39         3.16
## 6 2000-01-01     1  2000     35     29     1      39         3.24
```

4 Visual exploration of the data

We will no pose questions and answer them with graphs. In session, we may pose different questions, we will see.

4.1 Explore seasonality

We are going to explore seasonality not in the strict sense being used in many excess mortality methods: Here we're not going to *separate* seasonality from a smooth medium-term trend, we'll just eyeball it atop the trend.

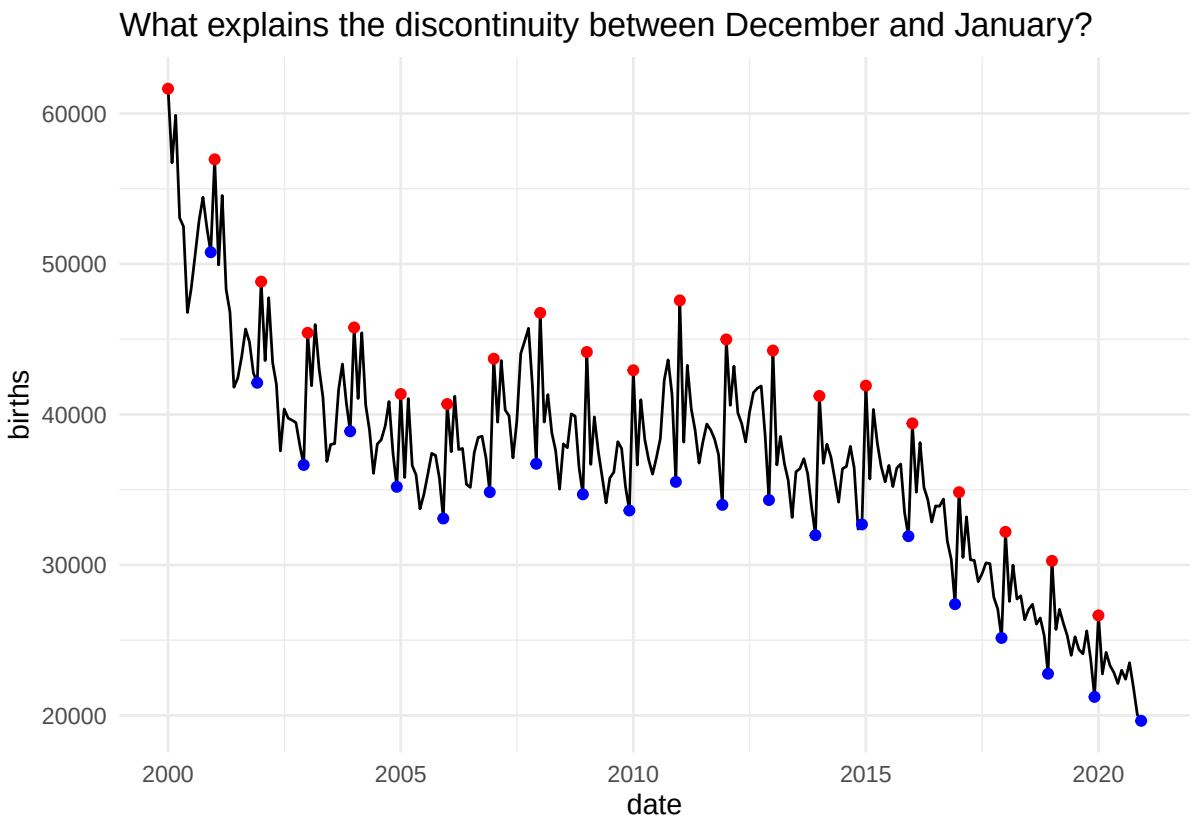
4.1.1 seasonality of births by month

Let's get a sense of how seasonal births are. This will happen in two steps: 1. Aggregate the data by year, month; use the `n()` function inside `summarize()` to count rows 2. Create a date variable 3. plot the result

```
month_totals <-
  KB %>%
  group_by(date, month) %>%
  summarize(births = n(),
            .groups = "drop")
```

In plotting, we notice that births tend to have a January spike:

```
month_totals %>%
  ggplot(aes(x = date, y = births)) +
  geom_line() +
  # more honest but too much visual vibration IMO:
  # geom_step()

  # put a red dot on each January value
  geom_point(data = month_totals %>%
               filter(month == 1),
             color = "red") +
  geom_point(data = month_totals %>%
               filter(month == 12),
             color = "blue") +
  labs(title = "What explains the discontinuity between December and January?") +
  theme_minimal()
```



I presume some of the Korean workshop participants know what is going on. I, however, do not. Is it that conceptions (natural or assisted) are equally seasonal? Although conceptions are surely also seasonal, they cannot produce a discontinuity like this because there is a *smooth*ish distribution of gestation durations that prevents it. Are gestational periods being shortened / lengthened in order to achieve this result? Is it low-key fraud to give children a better position within their cohort? Are December due-dates more likely to be aborted? I unfortunately did not request variables for cesarean deliveries, nor for whether a conception was assisted, or for birth parity. Further, we lack in these data any conceptions that did not come to term.

4.1.2 Exercise (3 min)

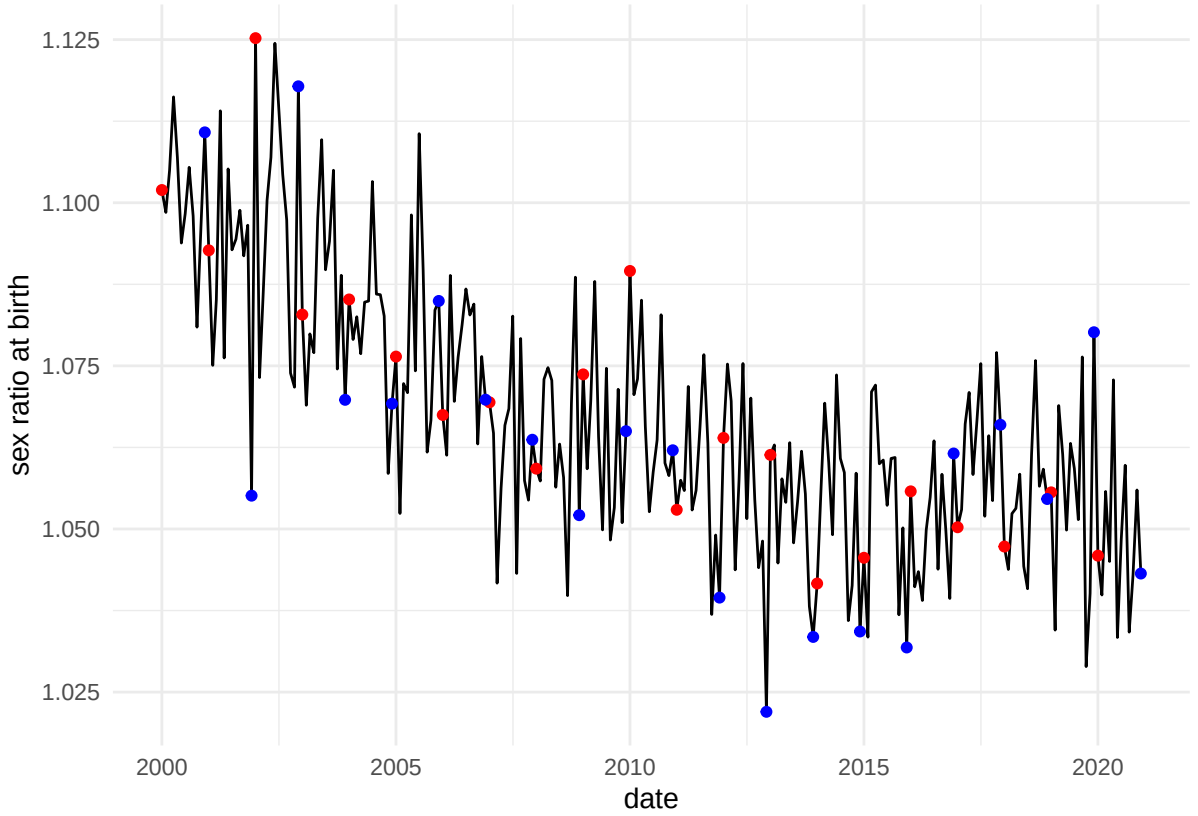
Take 3-4 minutes to calculate this time series separately for boy births and girl births, and plot the two lines, mapping color to sex. Is one series more seasonal than the other?

4.1.3 Seasonality of (secondary) sex ratio at birth

Does the sex ratio at birth have any regular seasonality? We'd expect it to be more erratic, and it does seem to be this way. Higher values means more boys. I don't see a consistent mapping to the December-January discontinuity we saw before.

```
monthly_sr <-
 KB %>%
  group_by(date, month, sex) %>%
  summarize(births = n(),
            .groups = "drop") %>%
  pivot_wider(names_from = sex,
              values_from = births) %>%
  mutate(sex_ratio = `1`/`2`)

monthly_sr %>%
```

```
  ggplot(aes(x = date, y = sex_ratio)) +
  geom_line() +
  geom_point(data = monthly_sr %>% filter(month == 1),
             color="red") +
  geom_point(data = monthly_sr %>% filter(month == 12),
             color="blue") +
  labs(y = "sex ratio at birth") +
  theme_minimal()
```



4.1.4 Seasonality of conceptions

What would a time-series of *conceptions resulting in live births* look like? This unfortunately won't be the same as an unconditional time series of conceptions, that's just the nature of live-birth data. For the case of the USA and Spanish data linked above, one could join with data on pregnancies or fetal deaths to get a closer-to-complete time series of conceptions. Caveats (that is, conditioning) aside, let's get down to pragmatic programming.

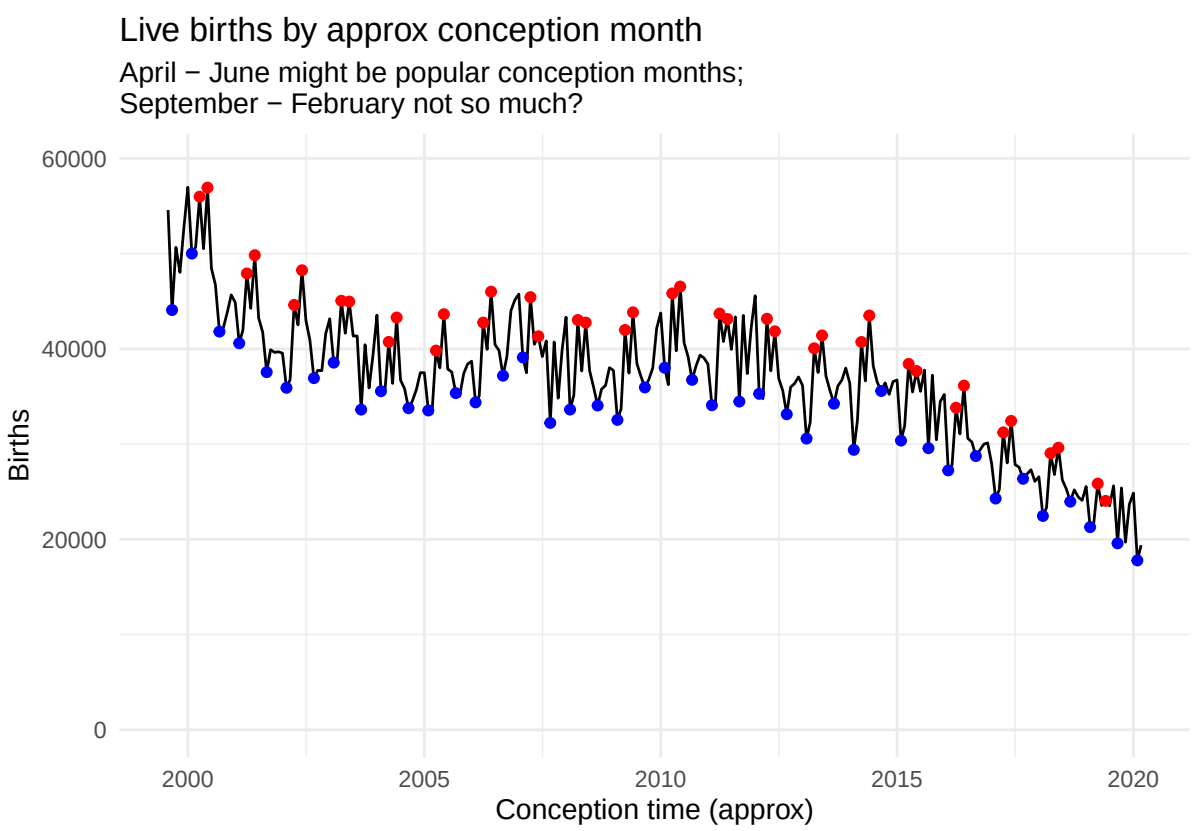
We have a variable `p_weeks`. This is in weeks, but we only have month-resolution for births, ouch. Let's figure out a way to hack this so we can make the desired time-series. My hackish approach will be to assume that a birth happens on the 15th day of the month, and to subtract whole weeks of pregnancy from this, ergo: `date_conception_approx = date - p_weeks * 7 + 15`. The resulting dates are going to be scattered, but we can group these back into months using the `month()` helper function (give it a date, it returns a month integer, see also `day()` and `year()`). Once we get the (approximate) month of conception, we can aggregate, and once again recreate a "first of the month" date column for purposes of plotting:

```
conceptions_month <-
  KB %>%
  mutate(date_conception_approx = date - p_weeks * 7 + 15,
```

```
         year = year(date_conception_approx),
         month_conception_approx = month(date_conception_approx)) %>%
  group_by(year, month_conception_approx) %>%
  summarize(conceptions = n(),
            .groups = "drop") %>%
  mutate(date_for_plotting = ymd(paste(year,
                           month_conception_approx,
                           "01")))
```

Plot it, highlighting both peaks and valleys

```
conceptions_month %>%
  ggplot(aes(x = date_for_plotting, y = conceptions )) +
  geom_line() +
  geom_point(data = conceptions_month %>% filter(month_conception_approx %in% c(4,6)),
             color = "red") +
  geom_point(data = conceptions_month %>% filter(month_conception_approx %in% c(2,9)),
             color = "blue") +
  labs(title = "Live births by approx conception month",
       subtitle =
       "April - June might be popular conception months;
September - February not so much?") +
  xlim(ymd("1999-08-01"),ymd("2020-03-01")) +
  labs(x = "Conception time (approx)", y = "Births") +
  theme_minimal()
```



4.1.5 Exercise: (10 min)

Is birth-weight seasonal to any extent? What about age differences of parents? What strange questions! But see how straightforward it is to ask them of the data! Try to calculate one or the other. Try calculating mean or median weights (or age differences). There is no need to tabulate in advance of this. Then plot a time series of both.

4.2 Exploring distributions

4.2.1 Gestation weeks distributions

How does the distribution of live births by weeks of gestation change as a function of mothers' age? We use the modulo %% operator to help aggregate to 5-year age groups. Observe the behavior:

```
age <- 0:20
age
```

```
##  [1]  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

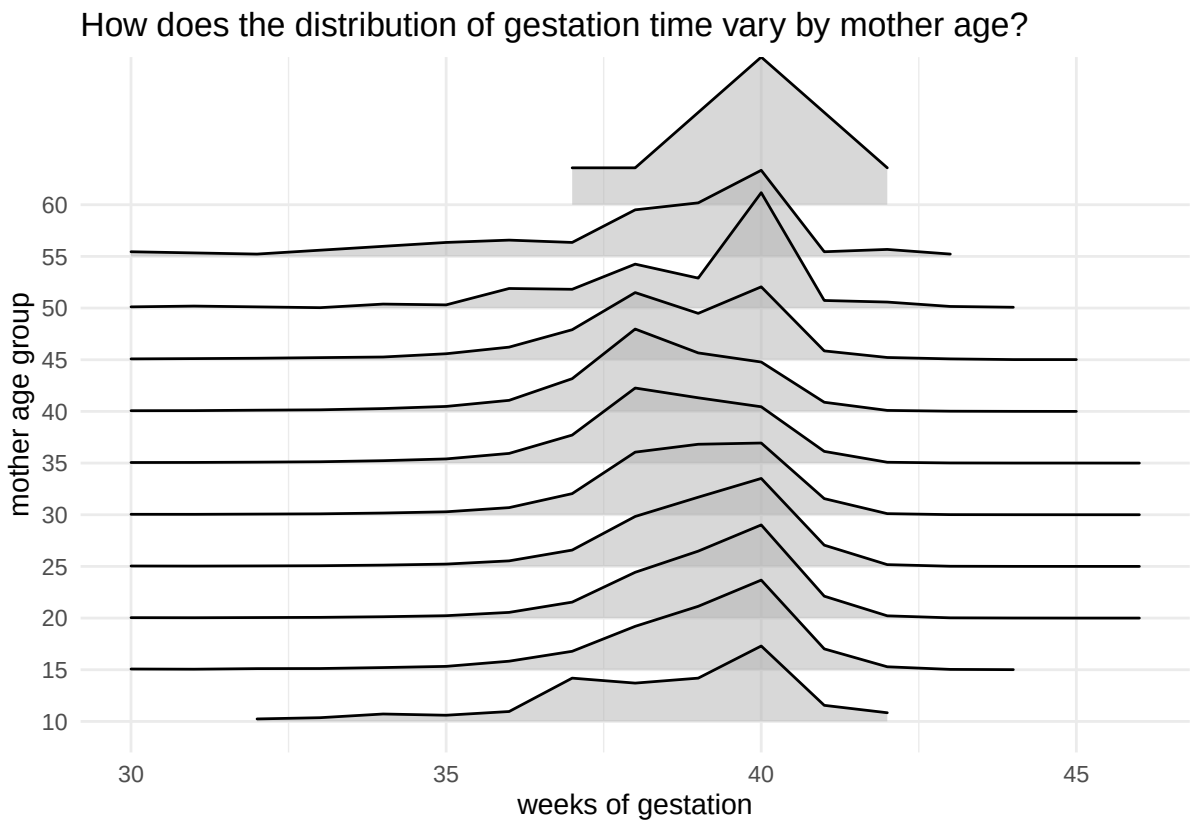
```
# this we can aggregate on, get it?
age - age %% 5
```

```
##  [1]  0  0  0  0  0  5  5  5  5  5 10 10 10 10 10 15 15 15 15 15 20
```

```
density_by_mother_age <-
  KB %>%
  filter(mo_age < 70,
         # why are some `p_weeks` equal to 0?
         p_weeks > 0) %>%
  # note the %% (modulo) trick.
  mutate(mo_age5 = mo_age - mo_age %% 5) %>%
  group_by(mo_age5, p_weeks) %>%
  summarize(n = n(), .groups = "drop") %>%
  ungroup() %>%
  group_by(mo_age5) %>%
  mutate(dens = n / sum(n) )
```

Plot it as a ridge plot! This is a new `geom` for us. `geom_ridgeline()` comes from the `ggridges` package. The main mappings are `x`, `y`, and `height`, where `x` is straightforward, `y` refers to the level of each ridge baseline, and `height` refers to `y` value of each individual distribution.

```
density_by_mother_age %>%
  ggplot(aes(x = p_weeks, y = factor(mo_age5), height = dens)) +
  geom_ridgeline(scale = 5, alpha = .5) +
  xlim(30,46) +
  labs(y = "mother age group",
       x = "weeks of gestation",
       title = "How does the distribution of gestation time vary by mother age?") +
  theme_minimal()
```



4.2.2 Exercise (5 min)

Repeat this ridge plot, except use fathers' age in 5-year age groups. Does the distribution seem to change its shape more by fathers' age or mothers' age?

4.2.3 Exercise (7 min)

Again, throwing out unknown mother and father ages, calculate a new variable called `age_diff` (father age minus mother age). Use `case_when()` to categorize age differences into 3 bins: 1. `age_diff > 2` 2. `between(age_diff,2,-2)` 3. `age_diff < -2` Recreate the first ridgeplot (using `mother_age5`), and map `fill` color to your new 3-category `age_diff`. You'll want to set `alpha` so that the densities aren't fully overlapped.

4.2.4 Age distributions

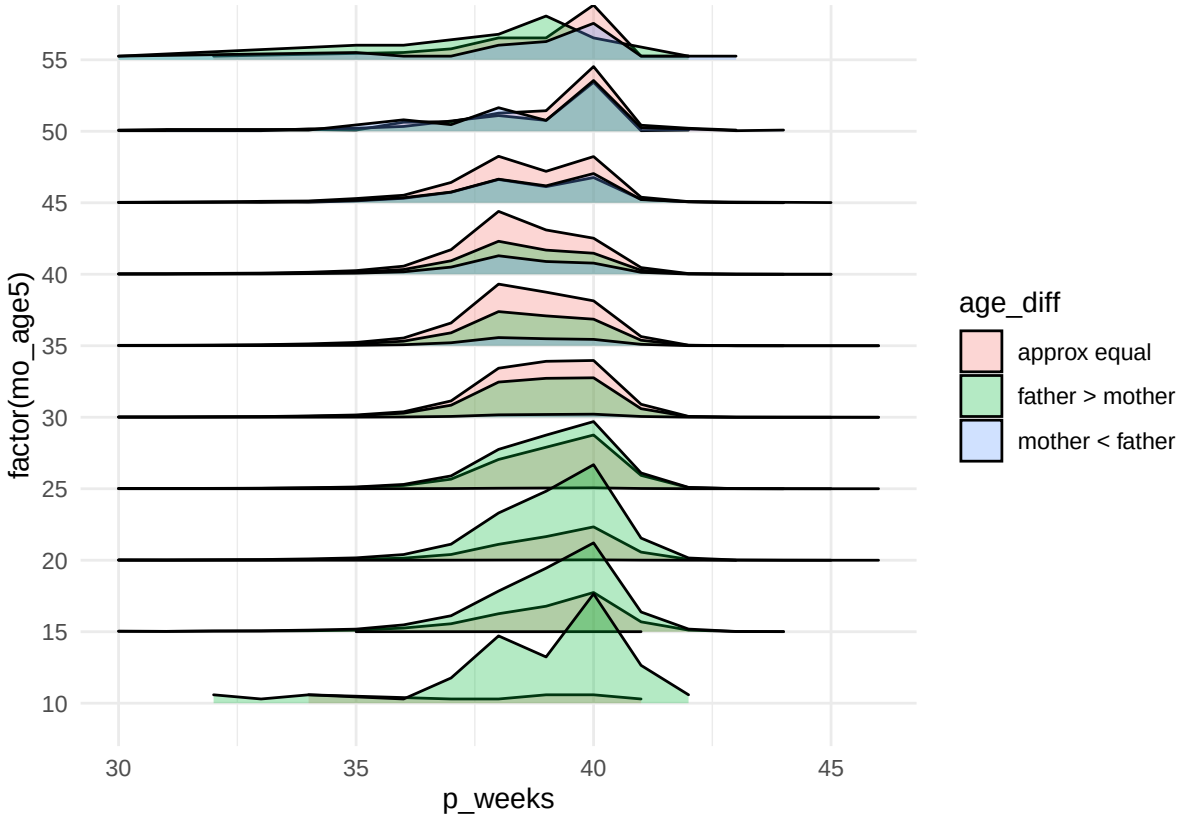
Now we'll modify a bit the previous exercise (sorry for the giveaway) in order to reveal that these virtually identical distributions are nonetheless on quite different scales, and that these relative scales shift over age. In this case, one could try to scale within mother age only as a different density:

```
density_by_mother_age_diff2 <-
  KB %>%
  filter(mo_age < 60,
         fa_age < 70,
         fa_age >= 15,
         # why are some `no_of_weeks_of_pregnancy` equal to 0?
         p_weeks > 0) %>%
  mutate(mo_age5 = mo_age - mo_age %% 5,
         age_diff = fa_age - mo_age,
```

```
        age_diff = case_when(
          age_diff > 2 ~ "father > mother",
          age_diff < -2 ~ "mother < father",
          TRUE ~ "approx equal"
        )) %>%
  group_by(age_diff, mo_age5, p_weeks) %>%
  summarize(n = n(), .groups = "drop") %>%
  ungroup() %>%
  group_by(mo_age5) %>%
  mutate(dens = n / sum(n) )
```

Plot it. This makes clearer that for younger mothers, it's more common to have older fathers, but as mother ages increase, it becomes more prevalent to have younger fathers.

```
density_by_mother_age_diff2 %>%
  ggplot(aes(x = p_weeks,
             y = factor(mo_age5),
             fill = age_diff,
             height = dens)) +
  ggridges::geom_ridgeline(scale = 5,
                           alpha = .3) +
  xlim(30,46) +
  theme_minimal()
```



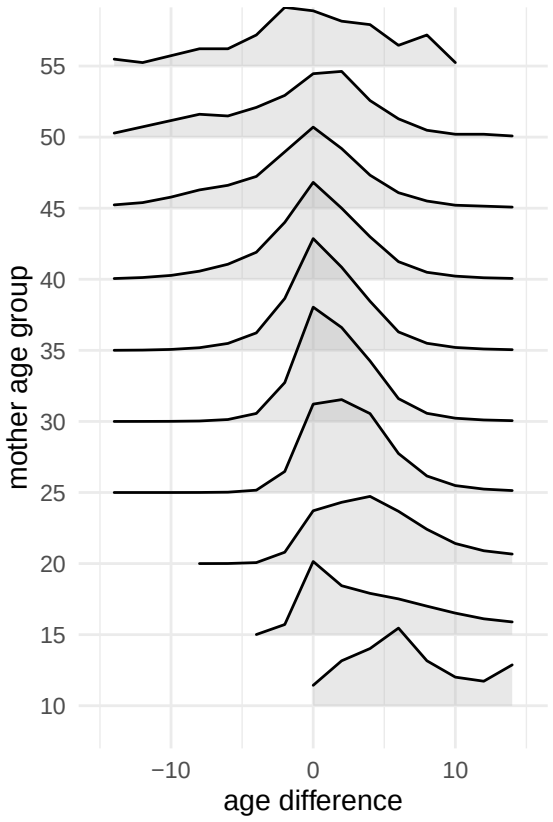
But then, since the distribution shapes don't differ except in the top category, we really should simplify the display to show the *shifting age differences* balance.

```
age_diff_density_by_mother_age <-
  KB %>%
```

```
  filter(mo_age < 60,
         fa_age < 75) %>%
  mutate(mo_age5 = mo_age - mo_age %% 5,
         age_diff = fa_age - mo_age,
         # 2-year diff bins, why not?
         age_diff = age_diff - age_diff %% 2) %>%
  group_by(age_diff, mo_age5) %>%
  summarize(n = n(), .groups = "drop") %>%
  group_by(mo_age5) %>%
  mutate(dens = n / sum(n) )
```

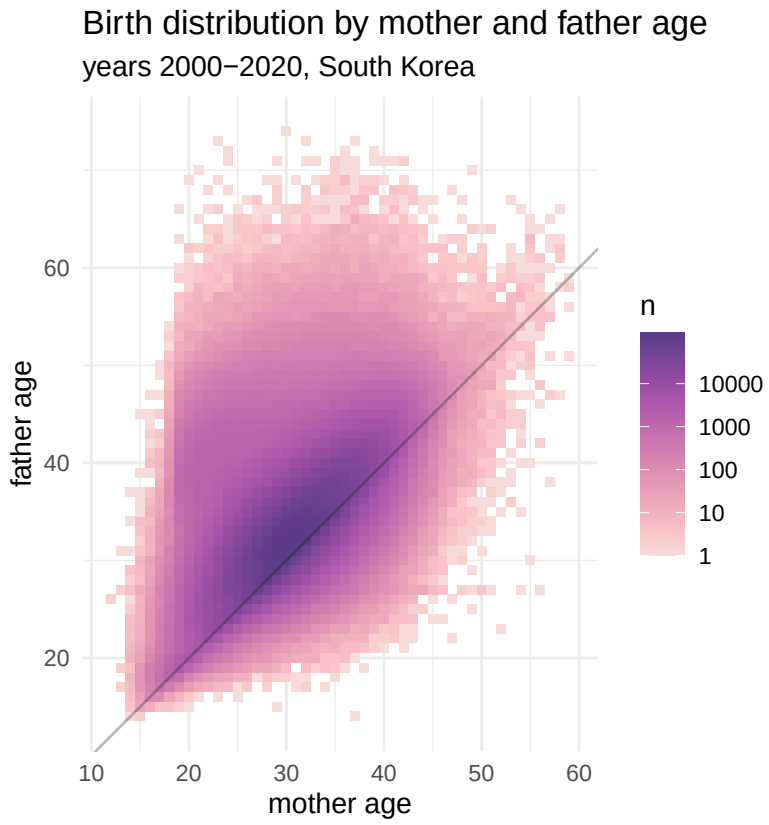
This density gets the shifting age differences over better. Remember each level is scaled to sum to 1, so this isn't a direct translation of a 2d age plot, which would give the full 2d density.

```
age_diff_density_by_mother_age %>%
  ggplot(aes(x = age_diff,
             y = factor(mo_age5),
             height = dens)) +
  ggridges::geom_ridgeline(scale = 5,
                           alpha = .3) +
  coord_fixed(ratio = 5)+
  xlim(-15, 15) +
  labs(y = "mother age group",
       x = "age difference") +
  theme_minimal()
```



Here's that same data displayed as a Lexis-like heatmap (mother age by father age)

```r
my_breaks = c(0,1,10,100,1000,10000)
 KB %>%
  filter(mo_age < 60,
         fa_age < 75) %>%
  group_by(mo_age,
           fa_age) %>%
  summarize(n = n(), .groups = "drop") %>%
  ggplot(aes(x = mo_age,
             y = fa_age,
             fill = n)) +
  geom_tile() +
   # to keep age units equal!
  coord_equal() +
   # picked out a color ramp
  scale_fill_continuous_sequential("PurpOr",
                                   trans = "log",
                                   breaks = my_breaks)+
   # line of equality
  geom_abline(slope = 1,
              intercept = 0,
              color = "#22222250") +
  labs(x = "mother age",
       y = "father age",
       title = "Birth distribution by mother and father age",
       subtitle = "years 2000-2020, South Korea") +
  theme_minimal()
```



This picture gives the overall count density, but really, we'd like to see this on a rate scale

since cohorts of presumably varying size have passed through the data. Rates give a better reflection of a conditional intensity: conditional on there being people exposed. This is easy to do for mother's age-specific exposure or fathers' age-specific exposure separately, but to consider exposures jointly is a conundrum of formal demography (the *two-sex problem*).

4.2.5 Exercise (5 min)

Aggregate years into wide periods (5-years, say), creating a new variable called `period` with values such as `"2000-2004"`, and so on (remember `case_when()`). Then recreate the above surface-plot, but *faceted* on `period`. Do you note any changes in this bivariate distribution?

4.2.6 Exercise: (5 min)

1. Make a surface plot of `p_weeks` by `birth_weight` (try `geom_density2d_filled()`) Hint: tabulate the data first, and maybe try rounding `birth_weight` to intervals of just 0.1?
2. Can you visually assess somehow if the relationship is stable over mothers' age?

5 Prepare merge with population

We will use population denominators from the recent `wpp2022` `R` package. This package is on github because it's big, and CRAN doesn't like big data packages. Earlier versions of WPP data packages weren't as big because they were in 5-year age groups and periods.

When we calculate exposures, we should take into account this notice from the github `README`:

> Caution: All annual population datasets are considered to depict population to December 31 of each year.

So, 2000 means 31 December, 2000, and we should adjust our exposure approximation accordingly. Actually, I'll just redefine `year` as `year + 1`. I'll just account for this by subtracting 1 from `year` and calculating as we did before.

We also need to scale up population counts by a factor of 1000. The rest of the code abides by how we calculated exposures elsewhere in this workshop (mid-year arithmetic approximation).

```
data(pop1dt)
str(popAge1dt)
```

```
## Classes 'data.table' and 'data.frame':   2101305 obs. of  7 variables:
##  $ country_code: int  900 900 900 900 900 900 900 900 900 900 ...
##  $ name        : chr  "World" "World" "World" "World" ...
##  $ year        : int  1949 1949 1949 1949 1949 1949 1949 1949 1949 1949 ...
##  $ age         : int  0 1 2 3 4 5 6 7 8 9 ...
##  $ popM        : num  41312 35761 33515 31076 28787 ...
##  $ popF        : num  39439 34274 32065 29781 27647 ...
##  $ pop         : num  80752 70035 65580 60857 56434 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
# what is South Korea's code in the UN system?
countrycode("South Korea", origin = "country.name.en", destination = "un")
```

```
## [1] 410
```

```
# check once again:
countrycode(410, origin = "un", destination = "iso3c")
```

```
## [1] "KOR"
```

```r
# see how the countrycode package is cool?

E <-
  popAge1dt %>%
  filter(country_code == 410,
         year %in% 1999:2020) %>%
  # scale up pop
  mutate(popM = popM * 1000,
         popF = popF * 1000,
         # bring back to Jan 1
         year = year + 1) %>%
  select(-pop) %>%
  rename(m = popM,
         f = popF) %>%
  pivot_longer(c(m,f),
               names_to = "sex",
               values_to = "pop") %>%
  arrange(sex, age, year) %>%
  group_by(sex, age) %>%
  mutate(expos = (pop + lead(pop)) / 2) %>%
  ungroup() %>%
  filter(!is.na(expos))
```

5.0.1 Considerations for joining:

In the births data, father age and mother age necessarily cross-categorized, but we have no obvious way of cross-categorizing population counts. There might be nice tricks we could do to approximate *joint* exposure, but at first I think we best concentrate on either getting female or male fertility rates. In this case, we should tabulate births by year and mother age, and redistribute unknown ages.

I'll prepare a mothers' file to join, just with births by age

```r
KB$mo_age %>% unique() %>% sort()
```

```
##  [1] 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
## [26] 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61
## [51] 62 63
```

```r
all_mo_ages <- 12:63

B_mo <-
  KB %>%

  # tabulate
  group_by(year, mo_age) %>%
  summarize(births = n(), .groups = "drop") %>%

  # redistribute unknown ages
  group_by(year) %>%
  mutate(unk = births[is.na(mo_age)]) %>%
  filter(!is.na(mo_age)) %>%
  mutate(births = births + births / sum(births) * unk) %>%
```

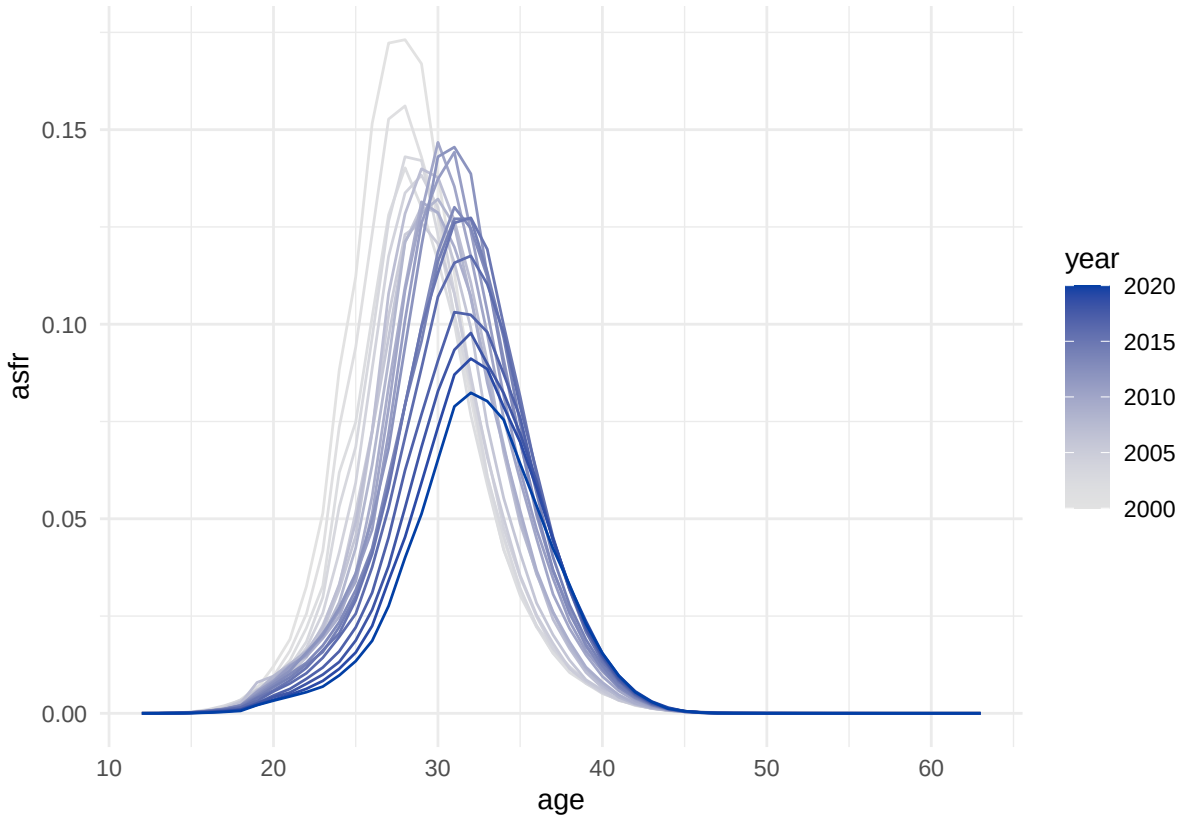
```
  ungroup() %>%
  select(-unk) %>%

  # get same age range every year, filling with 0s where no births
  # were observed.
  complete(mo_age = all_mo_ages, year, fill = list(births = 0)) %>%

  # change age header for joining!
  rename(age = mo_age)
```

Join births and exposures:

```
mo_fert <-
  E %>%
  filter(sex == "f") %>%
  right_join(B_mo, by = c("year","age")) %>%
  arrange(year, age) %>%
  mutate(asfr = births / expos)
```

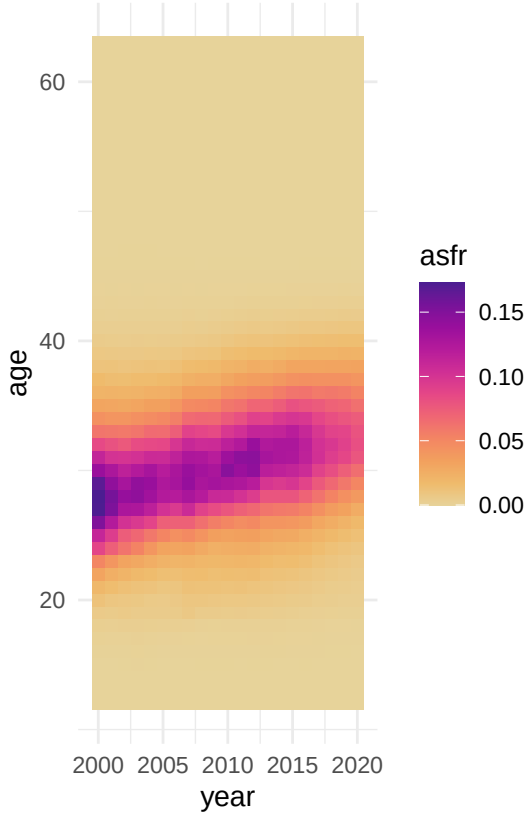
Take a look at asfr:

```
library(colorspace)
mo_fert %>%
  ggplot(aes(x = age, y = asfr, color = year, group = year)) +
  geom_line() +
  theme_minimal() +
  scale_color_continuous_sequential()
```



Or a Lexis surface:

```
mo_fert %>%
  ggplot(aes(x = year, y = age, fill = asfr)) +
  geom_tile() +
  scale_fill_continuous_sequential("ag_sunset")+
  theme_minimal() +
  coord_equal()
```



6 Exercises:

1. Calculate TFR, MAB, and the Bongaarts-Feeney adjustment of TFR for Korean mothers.
2. Calculate the same thing for *fathers*. That means process births in the same way as we did for mothers, but note the age range of fathers should extend to higher ages.
3. Merge the results for mothers and fathers and make a plot with 4 lines: TFR and adjusted TFR for mothers, and the same two again for fathers. You could map `color` to gender and `linetype` to the TFR variant perhaps? Do we notice anything interesting in the male vs female trends and levels?