Data Wrangling for EDSDers

Reshaping datasets

13-16 Nov, 2023

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

15 Nov, 2023

Note

This material was crafted on request from the students and has not been drafted as a standard handout. The notes and code below were written during the 15-11-2023 session and were not heavily edited thereafter. In the major worked example we show tricks for working with HRS data. The data were edited in advance to constitute a simplified subsample that requires the same reshaping tricks as the original data in order to calculate health transitions. If you want to see the code I used to read in the original file, look for a script called `hrs_prep.R` at the top level of the course repository. That code is not heavily annotated, and it might not be elegant, FYI. At the end of this exercise we do an ad hoc mortality comparison with HMD as well.

reshaping

`pivot_longer()` stacks some variables, usually creating one or more "name" columns and usually one value column. It will almost always result in more rows and less columns.

`pivot_wider()` unstacks data, distributing it over columns in some systematic way, for example spreading years over columns.

```
library(tidyverse)
library(gapminder)

gapminder |>
  pivot_longer(c(lifeExp, pop, gdpPercap),
               names_to = "variable",
               values_to = "value") |>
  pivot_wider(names_from = "variable", values_from = "value")
```

```
## # A tibble: 1,704 x 6
##    country     continent  year lifeExp      pop gdpPercap
```

```
##    <fct>       <fct>     <int>   <dbl>    <dbl>     <dbl>
##  1 Afghanistan Asia       1952    28.8  8425333      779.
##  2 Afghanistan Asia       1957    30.3  9240934      821.
##  3 Afghanistan Asia       1962    32.0 10267083      853.
##  4 Afghanistan Asia       1967    34.0 11537966      836.
##  5 Afghanistan Asia       1972    36.1 13079460      740.
##  6 Afghanistan Asia       1977    38.4 14880372      786.
##  7 Afghanistan Asia       1982    39.9 12881816      978.
##  8 Afghanistan Asia       1987    40.8 13867957      852.
##  9 Afghanistan Asia       1992    41.7 16317921      649.
## 10 Afghanistan Asia       1997    41.8 22227415      635.
## # i 1,694 more rows
```

What if we have some sort of hierarchical structure that's distributed over the columns? How do we use `pivot_longer()` to create more than one variable?

First is the brute force approach. This approach will never fail you. The strategy is to pivot to *super long*, manage columns (if needed), then pivot wider with just one `names_from` column. You'll see commented out another nifty trick.

```
gapminder |>
  pivot_wider(names_from = "year",
              values_from = c("lifeExp", "pop", "gdpPercap")) |>
  pivot_longer(lifeExp_1952:gdpPercap_2007,
               #names_to = c("variable", "year"),
               names_to = "variable_year",
               values_to = "value") |>  #,
               #names_sep = "_")
  # If you do the above commented-out trick, then
  # no need for this step; be aware that year results
  # as character either way.
  separate_wider_delim(variable_year,
           names = c("variable", "year"),
           delim = "_") |>
  pivot_wider(names_from = "variable", values_from = "value")
```

```
## # A tibble: 1,704 x 6
##    country     continent year  lifeExp      pop gdpPercap
##    <fct>       <fct>     <chr>   <dbl>    <dbl>     <dbl>
##  1 Afghanistan Asia      1952     28.8  8425333      779.
##  2 Afghanistan Asia      1957     30.3  9240934      821.
##  3 Afghanistan Asia      1962     32.0 10267083      853.
##  4 Afghanistan Asia      1967     34.0 11537966      836.
##  5 Afghanistan Asia      1972     36.1 13079460      740.
##  6 Afghanistan Asia      1977     38.4 14880372      786.
##  7 Afghanistan Asia      1982     39.9 12881816      978.
##  8 Afghanistan Asia      1987     40.8 13867957      852.
##  9 Afghanistan Asia      1992     41.7 16317921      649.
## 10 Afghanistan Asia      1997     41.8 22227415      635.
## # i 1,694 more rows
```

There is a way to handle the above `pivot_longer()` scenario in fewer steps, even handling all steps at once. The trick is to specify `names_to` as a vector with two parts: `".value"` stands for the various column

names, which need to be extracted from the knarly concatenated names you currently have, and then you use `names_sep` to specify the separator. Hopefully you have such as a separator!

```
gapminder |>
  pivot_wider(names_from = "year",
              values_from = c("lifeExp", "pop", "gdpPercap")) |>
  # a one-liner "pivot_longer() to multiple columns"
  pivot_longer(lifeExp_1952:gdpPercap_2007,
               names_to = c(".value","year"),
               names_sep = "_")
```

```
## # A tibble: 1,704 x 6
##    country     continent year  lifeExp      pop gdpPercap
##    <fct>       <fct>     <chr>   <dbl>    <int>     <dbl>
##  1 Afghanistan Asia      1952     28.8  8425333      779.
##  2 Afghanistan Asia      1957     30.3  9240934      821.
##  3 Afghanistan Asia      1962     32.0 10267083      853.
##  4 Afghanistan Asia      1967     34.0 11537966      836.
##  5 Afghanistan Asia      1972     36.1 13079460      740.
##  6 Afghanistan Asia      1977     38.4 14880372      786.
##  7 Afghanistan Asia      1982     39.9 12881816      978.
##  8 Afghanistan Asia      1987     40.8 13867957      852.
##  9 Afghanistan Asia      1992     41.7 16317921      649.
## 10 Afghanistan Asia      1997     41.8 22227415      635.
## # i 1,694 more rows
```

This last `pivot_wider()` approach is often the one we need to handle panel data. Well, at least if waves-variables are spread over columns it's the most handy trick.

HRS data

Due to a student request made back in Rostock, we will now work with panel data. We do this today because (1) we use some of the same tools from the previous lessons, and (2) we will use `pivot_longer()` to do the fundamental reshaping. We will work with a simplified version of the original dataset. If you want to do a full reproduction, then you need to get the original data here: https://hrsdata.isr.umich.edu/data-products/rand-hrs-longitudinal-file-2020 (requires painless registration) and run the auxiliary script called `hrs_prep.R`. The code is commented in-line.

```
library(tidyverse)
library(lubridate)
hrs <- read_csv("Data/hrs_subset_wide.csv.gz",
                show_col_types = FALSE)
```

Note the above line reads in the data just fine no matter what, but date columns might be read in as character or as date, depending on your installation (I guess, at least it happened in class). If your dates are character then in the below code you'll see commented-out lines to coerce to date classes, which you should uncomment.

```
# This saves an intermediate data object, but we could join
# pipelines into
hrs2 <-
```

```
  hrs |>
  # collect waves into a variable,
  # keeping value variables separate
  pivot_longer(contains("_"),
               names_to = c(".value", "wave"),
               names_sep = "_") |>
  # remove invalid interviews
  # This filter action might be too aggressive: the variable
  # iwstat could indicate a death in the period, when iwmid is NA,
  # but we'll capture this info later from death date info (not
  # without risk). It would be rigorous to rethink this step
  filter(!is.na(iwmid)) |>
  # create age and recode sex,
  # possibly you need to coerce dates from character?
  mutate(#iwmid = as_date(iwmid),
         #rabdate = as_date(rabdate),
         age = decimal_date(iwmid) - decimal_date(rabdate),
         age = floor(age),
         sex = if_else(ragender == 1, "Male", "Female"),
         # where to make the new columns:
         .after = rabdate) |>
  # remove unneeded columns to reduce clutter
  select(-ragender) |>
  # health variable recode with mroe than 2 categories:
  mutate(disab_from =
           case_when(iadla == 0 & adla == 0 ~ "Healthy",
                     adla > 0 ~ "Severe",
                     iadla > 0 ~ "Mild",
                     TRUE ~ "Missing")# we end up throwing this out.
           ) |>
  select(-slfmem, -iadla, -adla)
```

It's worth pointing out `case_when()` here, generally useful for any *exhaustive* recoding situation where you have more than 2 cases to handle and less than (it's up to you) 20 outgoing categories (that is, if you feel like typing so much). For more than 8 categories I actually prefer to handle the recoding problem using an externally-maintained look-up table and an appropriate `*_join()` to the data.

Now time to create the time 1 and time 2 health observations. To get two consecutive interviews side-by-side we use `lead()` (opposite of `lag()`). That has to be grouped on individual ids because its a within-individual data operation (we don't want health status jumping between individuals).

Then we need to create a new outcome (destination state) for the `disab_to` variable that is death. Deaths aren't coded inside the health variabels themselves; we need to get that info elsewhere. We could have saved the info from the `iwstat` variable but we inadvertently threw it out with an earlier filter statement, shrug. However, we can capture deaths from the death date information (year and month). Problem: we need to actually convert them to proper dates! In the following code it's a 3-step process. HT Maria for pointing out that the `as_date()` conversion was slow mainly due to unneeded groups.

Again we interrupt the continuous pipeline by saving to `hrs3` for the sake of annotation.

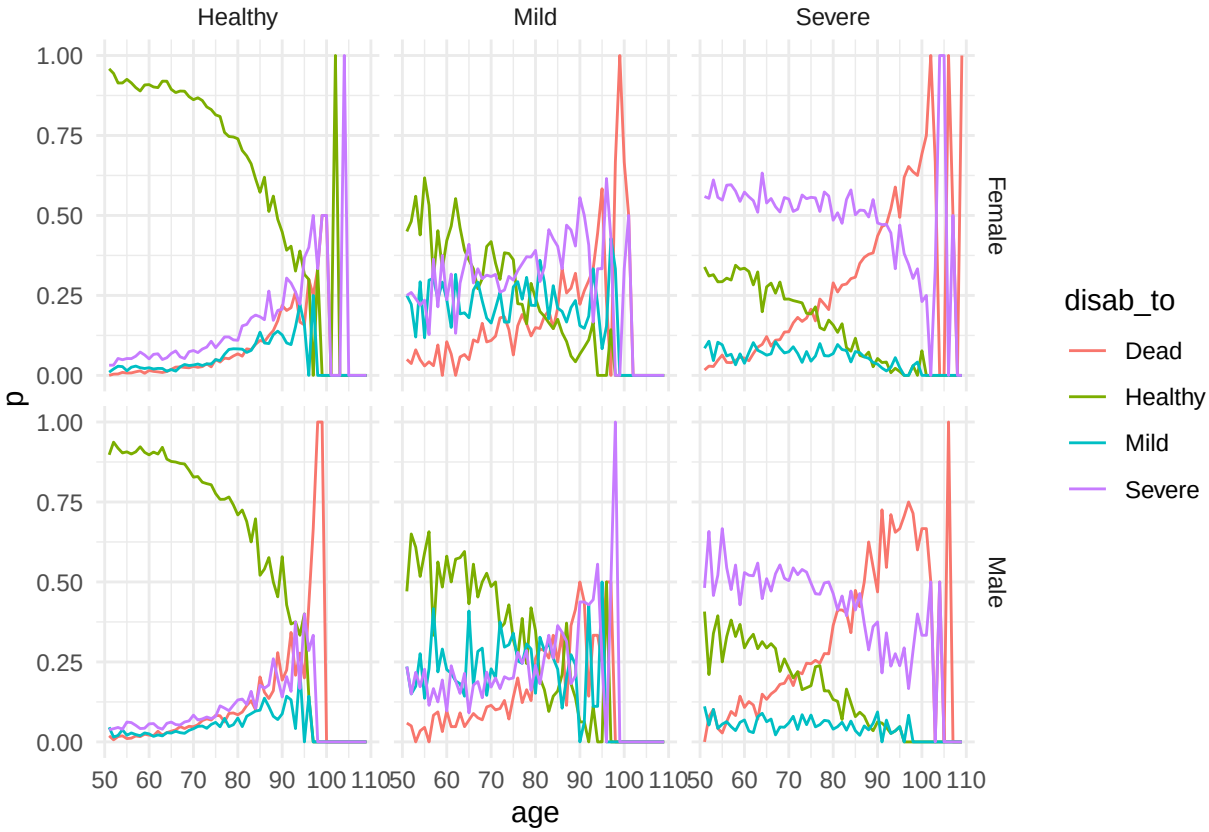
```
all_ages <- 51:109
hrs3 <-
  hrs2 |>
  group_by(hhidpn) |>
  # pull up "next interview" to be next to "this interview"
```

```
  # problem: deaths likely NAs, we need to explicitly find them
  mutate(disab_to = lead(disab_from)) |>
  ungroup() |>
  # find deaths using death date info, hackish solution.
  mutate(ddate = paste(radyear, radmonth, 15, sep = "-"),
         ddate = if_else(ddate == "NA-NA-15", NA, ddate),
         ddate = suppressWarnings(as_date(ddate)),
         disab_to = if_else(
           (!is.na(ddate)) & (ddate < (iwmid + (2 * 365))),
           "Dead",
           disab_to)) |>
  filter(age > 50) |>
  group_by(sex, age, disab_from, disab_to) |>
  summarize(# transitions = sum(wtcrnh))
    transitions = n(),
    .groups = "drop") |>
  # revisit this decision if you want to investigate forms of attrition.
  filter(disab_from != "Missing",
         disab_to != "Missing",
         # is.na(disab_to)
         !is.na(disab_to)) |>
  # plug in 0s where they are needed
  complete(sex, age = all_ages, disab_from, disab_to,
           fill = list(transitions = 0)) |>

  # now we can tabulate denominators
  group_by(sex, age, disab_from) |>
  mutate(denom = sum(transitions)) |>
  ungroup()
```

Take a look at the raw transition probabilities (recall 2-year probability scale!)

```
hrs3 |>
  mutate(p = transitions / denom,
         p = if_else(is.nan(p), 0, p)) |>
  ggplot(aes(x = age, y = p, color = disab_to)) +
  geom_line() +
  facet_grid(vars(sex), vars(disab_from)) +
  theme_minimal()
```



Smooth transitions, an ad hoc approach

One might want to smooth these, of course, in which case you might want to use a multinomial model with splines over age. You would choose that model, because you can't independently smooth these probabilities: We need to constrain sums over origin state to 1. A multinomial model would be the normal/pragmatic thing to do. For fun's sake, and to exercise our `dplyr` muscles, let's do something artisanal, but that still abides by the constraint, namely our friend the ALR transform, as discussed in Wikipedia's compositional data article https://en.wikipedia.org/wiki/Compositional_data and nicely implemented (along with its inverse) in the `compositions` R package.https://search.r-project.org/CRAN/refmans/compositions/html/alr.html, or `fido` package https://jsilve24.github.io/fido/. To reduce installation woes, we can implement the ALR transformation and its inverse like so (it's basically super similar to a logit):

```
#' @param x a named vector of probabilities
#' @param refernce the name of the vector element to use as ALR reference
alr_transform <- function(x, names, reference){
  # the [1] is in case reference is also a vector
  ind <- names == reference[1]
  log(x / x[ind])
}
#' @param x a named vector of probabilities
#' @param refernce the name of the vector element to use as ALR reference
alr_inv_transform <- function(x){
  exp(x) / sum(exp(x))
}
```

```
x <- c(Healthy = .9, Mild = .04, Severe = .04, Dead = .02)

(x_alr <- alr_transform(x, names(x), reference = "Healthy"))
```

```
##   Healthy      Mild    Severe      Dead
##  0.000000 -3.113515 -3.113515 -3.806662
```

```
(x_alr |> alr_inv_transform())
```

```
## Healthy    Mild  Severe    Dead
##    0.90    0.04    0.04    0.02
```

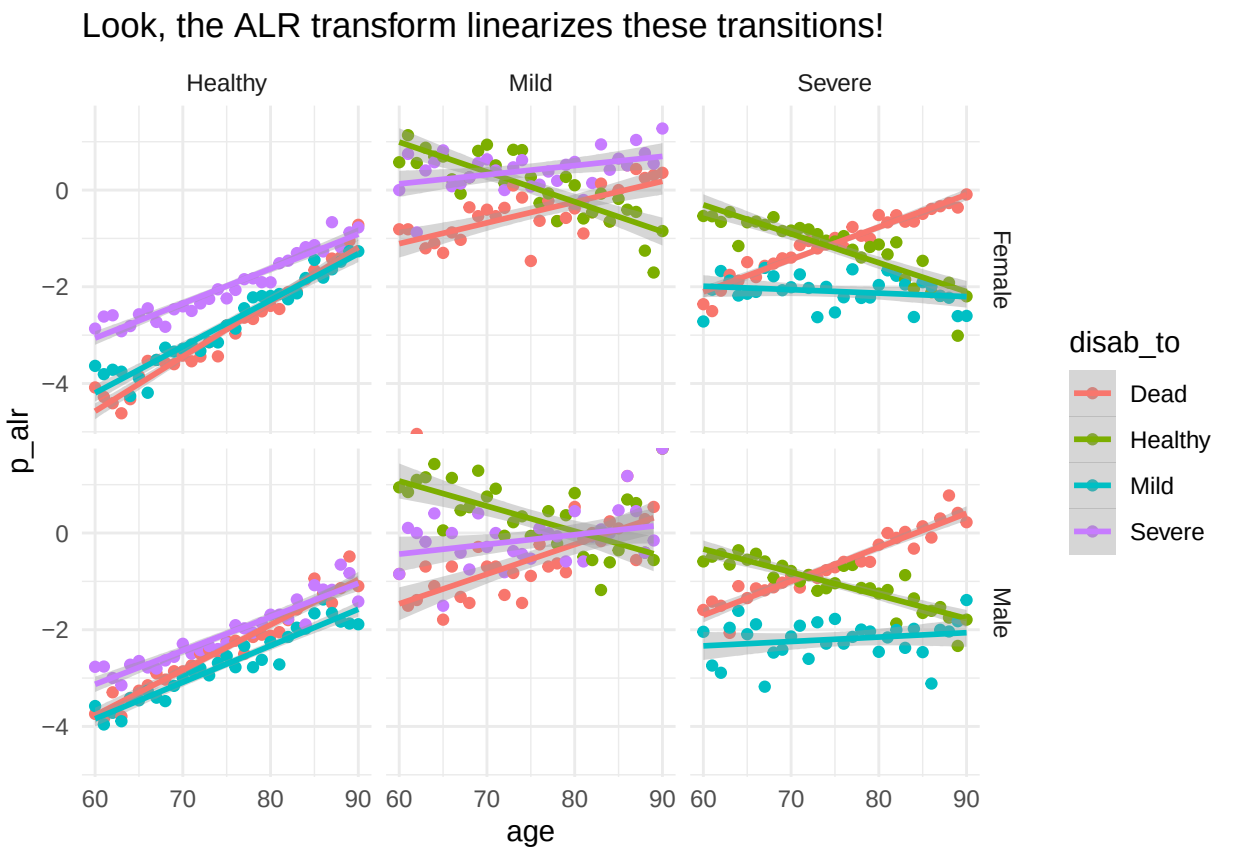
This behaves a bit different than other R implementations in that the reference element is maintained rather than dropped, and we're using names rather than index positions. The inverse does not need to know the reference, as long as it was never dropped from the vector. The plan will be to use transitions where the origin state is the same as the destination state (*self-transitions*, a.k.a. *staying put*) as the reference (denominator).

Mini-lesson time: it turns out that often for this sort of data, the ALR transform of transitions turns out to be nearly linear. That's super cool and convenient! Check this out, before we actually smooth things:

```
hrs3 |>
  mutate(p = transitions / denom) |>
  group_by(sex, disab_from,age) |>
   mutate(p_alr = alr_transform(p,
                                names = disab_to,
                                reference = disab_from)) |>
  ungroup() |>
  filter(disab_to != disab_from,
         between(age, 60, 90)) |>
  ggplot(aes(x = age, y = p_alr, color = disab_to))+
  geom_point() +
  facet_grid(cols=vars(disab_from), rows= vars(sex)) +
  theme_minimal() +
  geom_smooth(method = "lm") +
  labs(title = "Look, the ALR transform linearizes these transitions!")
```

```
## `geom_smooth()` using formula = 'y ~ x'
```

```
## Warning: Removed 4 rows containing non-finite values (`stat_smooth()`).
```



Look, folks, we could smooth the ALR transformed transitions using splines or something clever, if we wanted, but, given the above image, how about we just do a linear model? In this case, I'd like to use inverse variance as weights, treating them each as binomial for the sake of being pragmatic. First, here's a little helper function we can use inside `mutate()`:

```
cheap_lm <- function(p_alr, age, w){
  newdata <- tibble(age = age)
   lm(p_alr ~ age,
      weights = w,
      na.action = "na.omit") |>
   # predict() returns a vector same length as age
   predict(newdata = newdata)
}
```

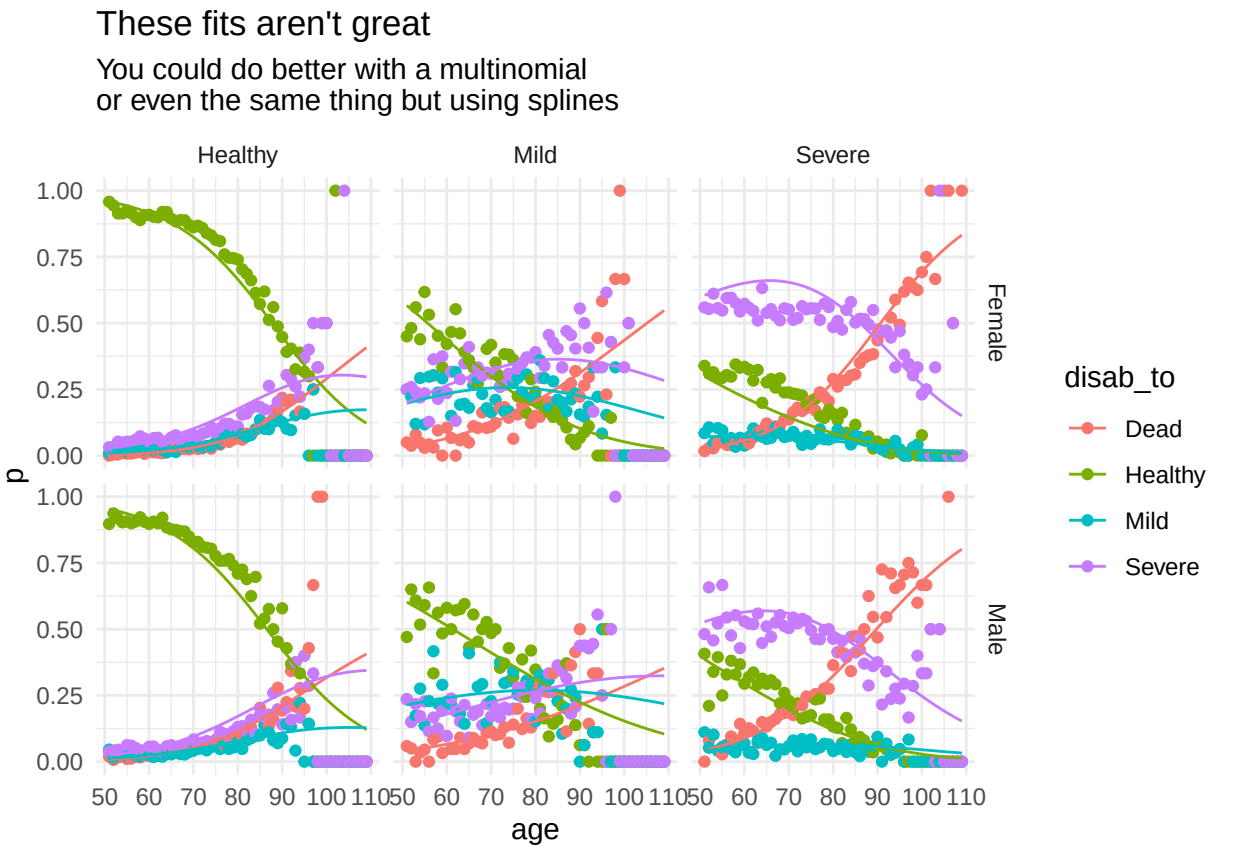
And now this pipeline uses our custom functions for ALR transforming and linear fitting, just for the sake of building something neat in a `dplyr` pipeline.

```
hrs4 <-
  hrs3 |>
  mutate(p = transitions / denom,
         p = if_else(is.nan(p), 0, p),
         # binom variance = NP(1-P)
         var_binom = denom * p * (1 - p),
         w = if_else(age > 100 | transitions == 0, 0, 1 / var_binom),
         w = if_else(is.infinite(w),0,w)) |>
  # use our custom ALR function;
```

```
  # here we need a unique vector of p for each sex, origin, and sex
  # each group will have 4 elements
  group_by(sex, disab_from, age) |>
  mutate(p_alr = alr_transform(p,
                               names = disab_to,
                               reference = disab_from),
         # we need to take care of Inf values in second step
         # lm() knows how to discard NAs but not Inf, and a 0 weight
         # also will not handle it
         p_alr = if_else(w == 0 | is.infinite(p_alr), NA, p_alr)) |>
  ungroup() |>
  # fit our simple model (over all ages and transitions, by sex)
  # these have approx 59 rows I think
  group_by(sex, disab_from, disab_to) |>
  mutate(p_alr_lm = cheap_lm(p_alr,age,w)) |>
  ungroup() |>
  # back-transform requires same original groups
  group_by(sex, disab_from,age) |>
  mutate(p_hat = alr_inv_transform(p_alr_lm)) |>
  ungroup() |>
  select(-var_binom, -w)
```

Compare the resulting transitions:

```
hrs4 |>
  ggplot(aes(x = age,y=p,color = disab_to))+
  geom_point() +
  geom_line(mapping = aes(y=p_hat)) +
  facet_grid(rows = vars(sex), cols = vars(disab_from)) +
  theme_minimal() +
  labs(title = "These fits aren't great",
       subtitle = "You could do better with a multinomial\nor even the same thing but using splines")
```



Let's say you don't like these fits (I wouldn't), then here are assorted R packages that might help you out:

- `stats::glm()`
- `mgcv::gam()`
- `nnet::multinom()`
- `VGAM::multinomial()`
- `flexisurv::flexsurvreg()` (requires different data format)
- `fido` package for a Bayesian approach

There are also other tools out there, sometimes you need to hunt around to find the right one for your problem (i.e. covariate setting, censoring needs, etc.).

Compare mortality with the HMD.

Often for health outcomes (prevalence or incidence) it is hard to judge data quality and pattern plausibility. That is because different health conditions can have different levels and patterns of incidence and prevalence, and we simply do not have the same sort of rigid *laws* to describe them as we do for mortality. So have a look, and blurr the patterns with your eyes perhaps.

Here, grab just the mortality part of our estimates:

```
hrs_mort <-
  hrs3 |>
  mutate(p = transitions / denom,
         p = if_else(is.nan(p), 0, p)) |>
```

```
  filter(disab_to == "Dead") |>
  # The rest of this is to harmonize with HMD columns
  select(sex, age, disab_level = disab_from, qx = p) |>
  mutate(source = "HRS",
         .before = 1)
```

Grab HMD data from markdown, like so: this block can be run live, but the Rmd cannot be built if these functions are executed in the process of building. My strategy for including `HMDHFDplus` data grabs inside markdown is to read the data in live in a chunk set to `eval = FALSE`, and then save it locally to a `csv` or similar. Then in the following chunk we read it in using `read_csv()`, and this chunk is executed when building.

```
library(HMDHFDplus)
mlt <- readHMDweb("USA",
                  "mltper_1x10",
                  username = Sys.getenv("us"),
                  password = Sys.getenv("pw"))
flt <- readHMDweb("USA",
                  "fltper_1x10",
                  username = Sys.getenv("us"),
                  password = Sys.getenv("pw"))
write_csv(mlt, file = "Data/mlt.csv")
write_csv(flt, file = "Data/flt.csv")
```

Here take the HMD data locally, this chunk can be executed automatically

```
flt <- read_csv("Data/flt.csv",
                show_col_types = FALSE) |>
  filter(Year == 2000) |>
  mutate(sex = "Female", .before = 1)

mlt <- read_csv("Data/mlt.csv",
                show_col_types = FALSE)|>
  filter(Year == 2000)|>
  mutate(sex = "Male", .before = 1)
```

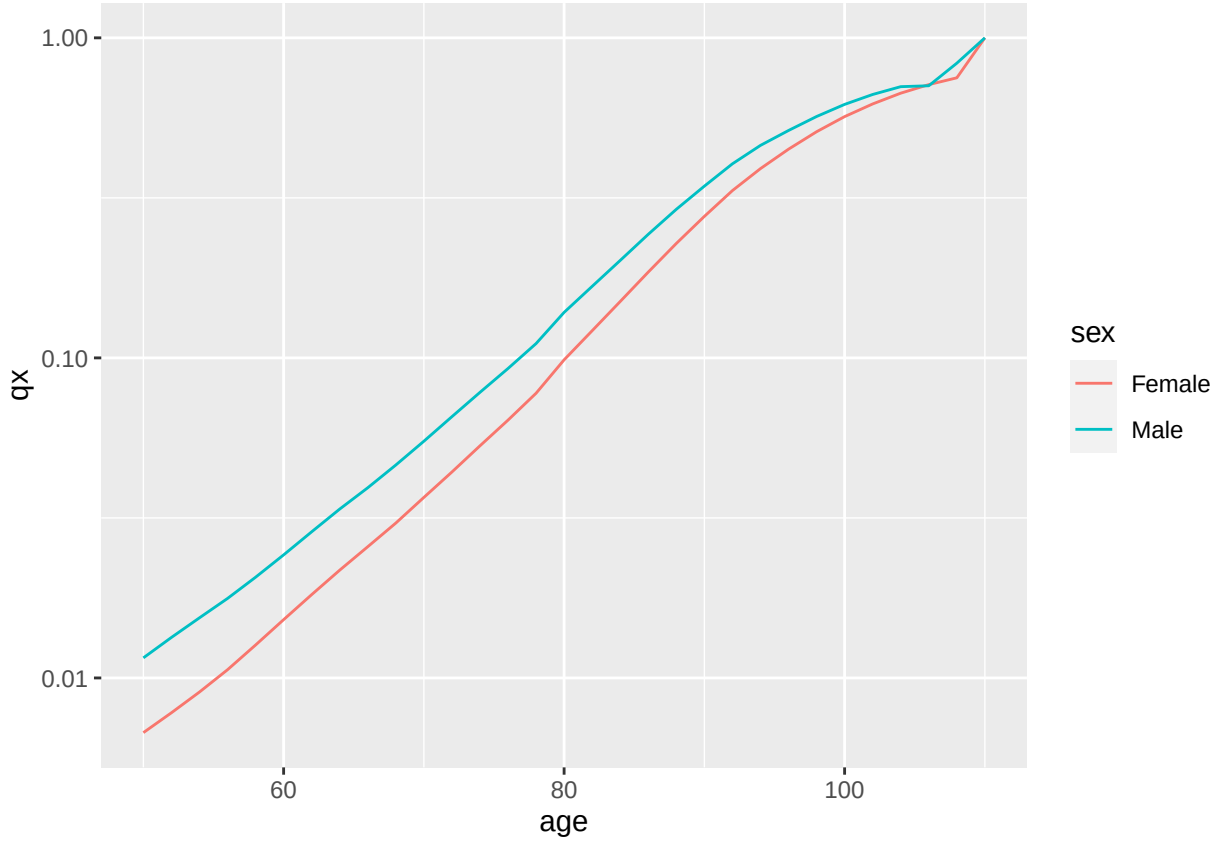
Now we need to *abridge* to two-year age groups, because our HRS death probabilities are in roughly two-year age groups. Even when referring to the same age, probabilities over different reference periods (single year vs two-year) will be on different scales and not directly comparable. That's why we need to do this transformation.

```
hmd_comparison <-
  bind_rows(flt, mlt) |>
  # first identify which 2-year age group the
  mutate(age = Age - Age %% 2) |>
  group_by(sex, age) |>
  # sum dx in 2-year ages; take "lower" lx value
  summarize(dx = sum(dx),
            lx = lx[1], # first lx is the lower one.
            .groups = "drop") |>
  # recalculate qx
  mutate(qx = dx / lx) |>
```

```
  select(sex, age, qx) |>
  # add metadata for joining to HRS
  mutate(source = "HMD",
         disab_level = "general") |>
  filter(age >= 50)
```

What do the HMD data look like? In class I noted a strange discontinuity for males around ages 105-106 or so, and I simply must know what data processing artifact on the HMD-side of things accounts for this discontinuity. Namely the single-year mx patterns should ALL be smooth, so how could a 10-year pattern *not* be smooth? This is literally the first time I've ever worked with the decade lifetables from HMD, isn't that funny?

```
hmd_comparison |>
  filter(age >= 50) |>
  ggplot(aes(x = age, y = qx, color = sex)) +
  geom_line() +
  scale_y_log10()
```



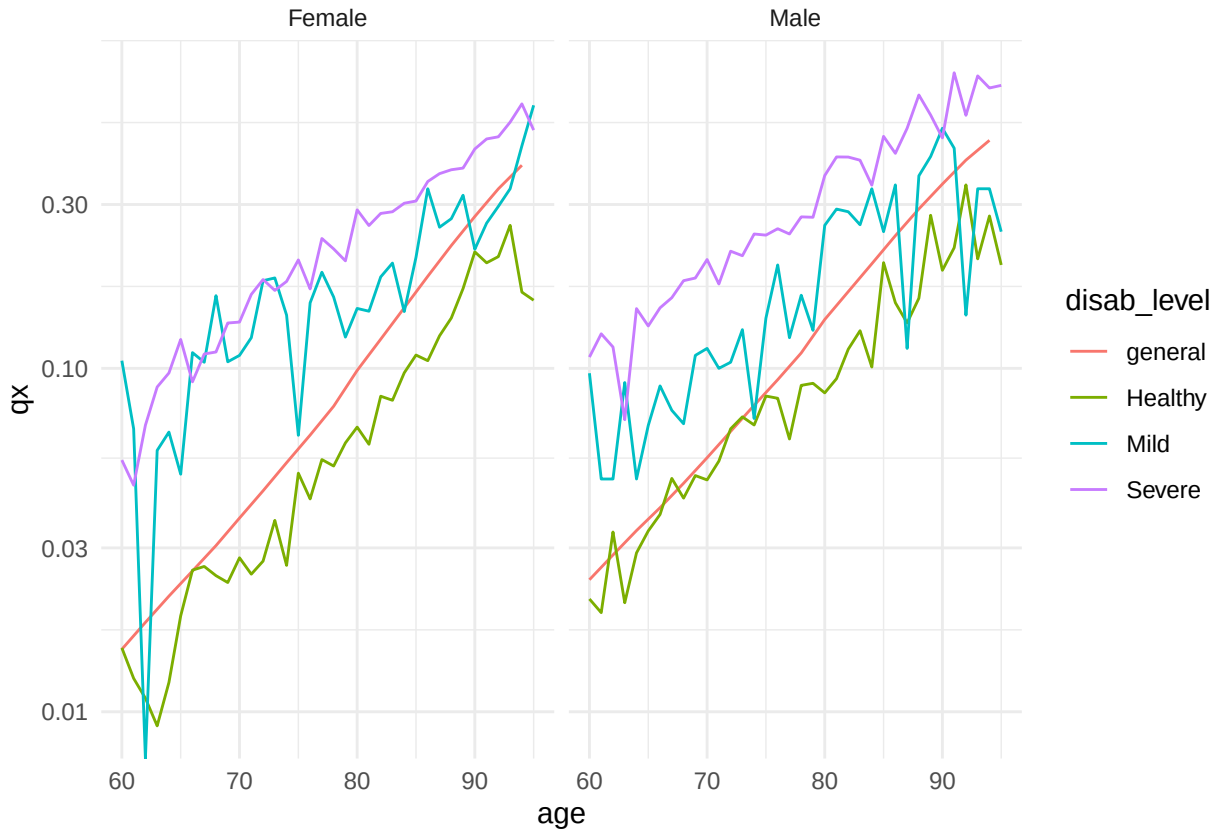
We standardized column names and variable definitions, so we're good to merge with a simple row-bind. We do it this way, because I'll want separate lines, and these need to be mappable (`disab_level`).

```
hrs_mort_check <-
  hrs_mort |>
  bind_rows(hmd_comparison)
```

Now make the comparison plot; we filter to ages 60-95 to not be distracted by the random noise outside these ages.

```
hrs_mort_check |>
  filter(between(age, 60, 95)) |>
  ggplot(aes(x = age, y = qx, color = disab_level))+
  geom_line() +
  facet_wrap(~sex) +
  scale_y_log10() +
  theme_minimal()
```

```
## Warning: Transformation introduced infinite values in continuous y-axis
```



Not bad!, we see the general-population mortality falling between the health-specific mortality, and we see the general population move from health to unhealthy mortality gradually over age, just as we'd image the prevalence-weighted average to do. This is an illustration of one of Vaupel's Ruses https://www.jstor.org/stable/pdf/2683925.pdf, but found out in nature.

This side-by-side doesn't quite give what we'd want to do something like a mortality adjustment. To check the overall mortality level in HRS we wouldn't have broken mortality down by states at all.

Regarding the HRS data

If you were in class and I shared the file `hrs_subset_wide.csv.gz` with you, you can save data items `hrs3` if you want, or `hrs_comparison`, but I'll ask you to to delete the file `hrs_subset_wide.csv.gz` from your

computer, as it should not be shared, and anyway, if you ever wanted to do something serious with this data you'd want to take advantage of more variables in it.