Tuesday session notes

Summary

Today we'll get an introduction to the tidyverse. Which includes reading in data, and reshaping it, recoding it, and merging data together.

Read in the births data

Tidy data (our objective) is defined as a tabular arrangement of data, where columns are strictly variables and rows consist in single observations.

```r
# packages we'll need
# install.packages("here")
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.3     v dplyr   1.0.7
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   2.0.0     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
library(readxl)
library(here)
```

```
## here() starts at /home/tim/workspace/KOSTAT_Workshop1
```

```r
Wide <- read_excel(path = here("Data", "demo_fasec.xlsx"),
           range = "A10:H158")
glimpse(Wide)
```

```
## Rows: 148
## Columns: 8
## $ AGE        <chr> "Total", "Total", "Total", "Total", "15 years", "15 years",~
## $ `GEO/TIME` <chr> "Belgium", "Czechia", "Spain", "Croatia", "Belgium", "Czech~
## $ `2011`     <dbl> 128705, 108673, 470553, 41197, 74, 67, 414, 35, 171, 224, 9~
## $ `2012`     <dbl> 128051, 108576, 453348, 41771, 72, 53, 379, 36, 161, 225, 8~
## $ `2013`     <dbl> 125606, 106751, 424440, 39939, 48, 55, 391, 42, 183, 204, 8~
## $ `2014`     <dbl> 125014, 109860, 426076, 39566, 52, 62, 375, 27, 180, 230, 8~
## $ `2015`     <dbl> 122274, 110764, 418432, 37503, 47, 66, 390, 24, 130, 177, 8~
## $ `2016`     <dbl> 121896, 112663, 408734, 37537, 37, 47, 341, 18, 119, 223, 8~
```

`pivot_longer()` collects a range of columns and stacks them. `names_to` is where the previous columns names get collected in a new column. `values_to` is where the cell values are collected as a single column.

```r
Long <-
   pivot_longer(data = Wide,
```

```
                cols = 3:8,
                names_to = "TIME",
                values_to = "Births")
Long
```

```
## # A tibble: 888 x 4
##    AGE   `GEO/TIME` TIME  Births
##    <chr> <chr>      <chr>  <dbl>
##  1 Total Belgium    2011  128705
##  2 Total Belgium    2012  128051
##  3 Total Belgium    2013  125606
##  4 Total Belgium    2014  125014
##  5 Total Belgium    2015  122274
##  6 Total Belgium    2016  121896
##  7 Total Czechia    2011  108673
##  8 Total Czechia    2012  108576
##  9 Total Czechia    2013  106751
## 10 Total Czechia    2014  109860
## # ... with 878 more rows
```

The column range can be specified by name too, or also using various kinds of conditional selection. In the second example it chooses all columns where the data type is `double`.

```
# select using column name range
 pivot_longer(data = Wide,
                cols = `2011`:`2016`,
                names_to = "TIME",
                values_to = "Births")
```

```
## # A tibble: 888 x 4
##    AGE   `GEO/TIME` TIME  Births
##    <chr> <chr>      <chr>  <dbl>
##  1 Total Belgium    2011  128705
##  2 Total Belgium    2012  128051
##  3 Total Belgium    2013  125606
##  4 Total Belgium    2014  125014
##  5 Total Belgium    2015  122274
##  6 Total Belgium    2016  121896
##  7 Total Czechia    2011  108673
##  8 Total Czechia    2012  108576
##  9 Total Czechia    2013  106751
## 10 Total Czechia    2014  109860
## # ... with 878 more rows
```

```
 pivot_longer(data = Wide,
                cols = where(is.double),
                names_to = "TIME",
                values_to = "Births")
```

```
## # A tibble: 888 x 4
##    AGE   `GEO/TIME` TIME  Births
##    <chr> <chr>      <chr>  <dbl>
##  1 Total Belgium    2011  128705
##  2 Total Belgium    2012  128051
##  3 Total Belgium    2013  125606
##  4 Total Belgium    2014  125014
```

```
##  5 Total Belgium    2015  122274
##  6 Total Belgium    2016  121896
##  7 Total Czechia    2011  108673
##  8 Total Czechia    2012  108576
##  9 Total Czechia    2013  106751
## 10 Total Czechia    2014  109860
## # ... with 878 more rows
```

Select and rename columns to whatever standard we want. When we assign to `Long` having started with `Long`, it overwrites the old one.

```
Long <-
  select(.data = Long,
       Country = `GEO/TIME`,
       Age = AGE,
       Year = TIME,
       Births)
glimpse(Long)
```

```
## Rows: 888
## Columns: 4
## $ Country <chr> "Belgium", "Belgium", "Belgium", "Belgium", "Belgium", "Belgiu~
## $ Age     <chr> "Total", "Total", "Total", "Total", "Total", "Total", "Total",~
## $ Year    <chr> "2011", "2012", "2013", "2014", "2015", "2016", "2011", "2012"~
## $ Births  <dbl> 128705, 128051, 125606, 125014, 122274, 121896, 108673, 108576~
```

Now let's redo the above three steps making use of piping. `%>%` Ctrl + Shift + m

```
Long <-
# step 1, read it in
  read_excel(
    path = here("Data", "demo_fasec.xlsx"),
    range = "A10:H158") %>%

# step 2, stack the years
  pivot_longer(
    cols = `2011`:`2016`,
    names_to = "Year",
    values_to = "Births"
  ) %>%

# step 3 select and rename columns the way we want
  select(
    Country = `GEO/TIME`,
    Year,
    Age = AGE,
    Births)
```

Now let's recode `Age`

```
Long %>%
  pull(Age) %>%
  unique()
```

```
##  [1] "Total"    "15 years" "16 years" "17 years" "18 years" "19 years"
##  [7] "20 years" "21 years" "22 years" "23 years" "24 years" "25 years"
## [13] "26 years" "27 years" "28 years" "29 years" "30 years" "31 years"
```

```
## [19] "32 years" "33 years" "34 years" "35 years" "36 years" "37 years"
## [25] "38 years" "39 years" "40 years" "41 years" "42 years" "43 years"
## [31] "44 years" "45 years" "46 years" "47 years" "48 years" "49 years"
## [37] "Unknown"
```

First, to demonstrate the processing steps on a single subset of the data, then do it for all subsets at once!!

```
library(readr)
# example so you understand logical selection
  # Long %>%
  # mutate(my_selector = Country == "Czechia" & Year == "2011") %>%
  # filter(my_selector) %>%
  # mutate(TOT = Births[Age == "Total"])

# but this way is better!
 Long %>%

   # select subset for this example to demonstrate the logic of it
  filter(Country == "Czechia",
         Year == "2011") %>%

   # move Total births up to a column
  mutate(TOT = Births[Age == "Total"]) %>%

   # now we can throw out Total and Unknown ages
  filter(!Age %in% c("Total","Unknown")) %>%

   # redistribute births with known age of mother
   # so that they add up to the total!
  mutate(Fraction = Births / sum(Births),
         Births = TOT * Fraction,

         # pick out the integer part of age from the character strings
         Age = parse_number(Age)) %>%

   # remove temporary / instrumental columns
  select(-TOT, -Fraction)
```

```
## # A tibble: 35 x 4
##    Country Year    Age Births
##    <chr>   <chr> <dbl>  <dbl>
##  1 Czechia 2011     15   67.0
##  2 Czechia 2011     16  224.
##  3 Czechia 2011     17  511.
##  4 Czechia 2011     18  818.
##  5 Czechia 2011     19 1434.
##  6 Czechia 2011     20 1946.
##  7 Czechia 2011     21 2257.
##  8 Czechia 2011     22 2712.
##  9 Czechia 2011     23 3163.
## 10 Czechia 2011     24 3873.
## # ... with 25 more rows
```

Mini time out to understand logicals and how they can be used to select things in `R`:

```r
a <- rnorm(10)
a[a >= 0]
```

```
## [1] 0.3760477 0.1751643 0.1075481 0.1918815
```

```r
my_selector <- a >= 0
a[!my_selector]
```

```
## [1] -0.7423828 -1.0133785 -0.8500017 -1.8551080 -1.0507045 -0.2013314
```

Time to do this for all the subsets at once!

```r
Births <-
# step 1, read it in
  read_excel(
    path = here("Data", "demo_fasec.xlsx"),
    range = "A10:H158") %>%

# step 2, stack the years
  pivot_longer(
    cols = `2011`:`2016`,
    names_to = "Year",
    values_to = "Births"
  ) %>%

# step 3 select and rename columns the way we want
  select(
    Country = `GEO/TIME`,
    Year,
    Age = AGE,
    Births) %>%

  # step 4 declare groups on each unique combination of Country and Year
  # that is present in these data. This creates independent groups!
  group_by(Country, Year) %>%

   # 5 move Total births up to a column
  mutate(TOT = Births[Age == "Total"]) %>%

   # 6 now we can throw out Total and Unknown ages
  filter(!Age %in% c("Total","Unknown")) %>%

   # 7 redistribute births with known age of mother
   # so that they add up to the total!
  mutate(Fraction = Births / sum(Births),
         Births = TOT * Fraction,

         # pick out the integer part of age from the character strings
         Age = parse_number(Age)) %>%

   # 8 remove temporary / instrumental columns
  select(-TOT, -Fraction) %>%

  # 9 remove the groups!
  ungroup()
```

Calculating summary measures

To calculate summary measures (including tabulations) we use `summarize()` (`summarise()`), just be sure to declare groups, if appropriate! And don't forget to remove them when done!

```
# The data are clean, let's calculate something!
MAB <-

  # the incoming data object, Births
  Births %>%

  # apply groups
  group_by(Country, Year) %>%

  # define the summary measure
  summarize(MAB = sum(Age * Births) / sum(Births) + .5,
            # remove unneeded groups
            .groups = "drop")
```

Process denominators

First, read the data in, be sure to declare the `NA` character, `":"`

```
Pop <-

  # First read in a cell range from the spreadsheet
  read_excel(path = here("Data", "demo_pjan.xlsx"),
           range = "A10:CZ510",
           # this time it's necessary to declare the NA code
           na = ":") %>%

  pivot_longer(cols = `Less than 1 year`:`Unknown`,
              names_to = "Age",
              values_to = "Population") %>%

  filter(!is.na(Population))
```

Recode age classes:

```
Pop %>%
  pull(Age) %>%
  unique()
```

```
##   [1] "Less than 1 year"     "1 year"               "2 years"
##   [4] "3 years"              "4 years"              "5 years"
##   [7] "6 years"              "7 years"              "8 years"
##  [10] "9 years"              "10 years"             "11 years"
##  [13] "12 years"             "13 years"             "14 years"
##  [16] "15 years"             "16 years"             "17 years"
##  [19] "18 years"             "19 years"             "20 years"
##  [22] "21 years"             "22 years"             "23 years"
##  [25] "24 years"             "25 years"             "26 years"
##  [28] "27 years"             "28 years"             "29 years"
##  [31] "30 years"             "31 years"             "32 years"
##  [34] "33 years"             "34 years"             "35 years"
```

```
##  [37] "36 years"            "37 years"            "38 years"
##  [40] "39 years"            "40 years"            "41 years"
##  [43] "42 years"            "43 years"            "44 years"
##  [46] "45 years"            "46 years"            "47 years"
##  [49] "48 years"            "49 years"            "50 years"
##  [52] "51 years"            "52 years"            "53 years"
##  [55] "54 years"            "55 years"            "56 years"
##  [58] "57 years"            "58 years"            "59 years"
##  [61] "60 years"            "61 years"            "62 years"
##  [64] "63 years"            "64 years"            "65 years"
##  [67] "66 years"            "67 years"            "68 years"
##  [70] "69 years"            "70 years"            "71 years"
##  [73] "72 years"            "73 years"            "74 years"
##  [76] "75 years"            "76 years"            "77 years"
##  [79] "78 years"            "79 years"            "80 years"
##  [82] "81 years"            "82 years"            "83 years"
##  [85] "84 years"            "85 years"            "86 years"
##  [88] "87 years"            "88 years"            "89 years"
##  [91] "90 years"            "91 years"            "92 years"
##  [94] "93 years"            "94 years"            "95 years"
##  [97] "96 years"            "97 years"            "98 years"
## [100] "99 years"            "Open-ended age class" "Unknown"
```

We'll use `parse_number()` just like before

```
Pop <-

  # incoming population data
  Pop %>%

  # recode age, accounting for all cases, assinging NA to Unknown
  mutate(Age = case_when(
    Age == "Less than 1 year" ~ 0,
    Age == "Open-ended age class" ~ 100,
    Age == "Unknown" ~ NA_real_,
    TRUE ~ parse_number(Age)
  )) %>%

  # select the columns we want to keep and rename as needed
  select(Country = `GEO/AGE`,
         Year = TIME,
         Age,
         Population)
```

```
## Warning: 783 parsing failures.
## row col expected               actual
## 101  -- a number Open-ended age class
## 102  -- a number Unknown
## 203  -- a number Open-ended age class
## 204  -- a number Unknown
## 305  -- a number Open-ended age class
## ... ... ........ ....................
## See problems(...) for more details.
```

Demonstrate `case_when()`

```
x <- 0:20
abc <- letters[1:21]
x %% 2
```

```
##  [1] 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
```

```
x %% 2 == 0
```

```
##  [1]  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE
## [13]  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE
```

```
case_when(x %% 3 == 0 ~ "Maybe tomorrow will be better",
          x %% 2 == 0 ~ "Today is a good day",
          TRUE ~ "Tomorrow for sure excellent")
```

```
##  [1] "Maybe tomorrow will be better" "Tomorrow for sure excellent"
##  [3] "Today is a good day"           "Maybe tomorrow will be better"
##  [5] "Today is a good day"           "Tomorrow for sure excellent"
##  [7] "Maybe tomorrow will be better" "Tomorrow for sure excellent"
##  [9] "Today is a good day"           "Maybe tomorrow will be better"
## [11] "Today is a good day"           "Tomorrow for sure excellent"
## [13] "Maybe tomorrow will be better" "Tomorrow for sure excellent"
## [15] "Today is a good day"           "Maybe tomorrow will be better"
## [17] "Today is a good day"           "Tomorrow for sure excellent"
## [19] "Maybe tomorrow will be better" "Tomorrow for sure excellent"
## [21] "Today is a good day"
```

Now we should redistribute population counts with unknown age, proportional to those of known age.

`ifelse(logical, TRUE, FALSE)`

```
Pop <-
  Pop %>%

  group_by(Country, Year) %>%

  mutate(UNK = Population[is.na(Age)],
         # fills created NAs with 0s, because missing Unknowns
         # just means that there were none
         UNK = ifelse(is.na(UNK), 0, UNK)) %>%

  # throw out the NA ages (unknowns)
  filter(!is.na(Age)) %>%

  # redistribution as a 1-liner,
  # Population / sum(Population) is what we called Fraction before
  mutate(Population = Population + Population / sum(Population) * UNK,
         Year = as.integer(Year)) %>%

  # remove groups no longer needed
  ungroup() %>%

  # remove unneeded column
  select(-UNK)
```

We made it this far on Tuesday Calculate exposures by taking the mean of January 1 (`P1`) and December 31 `P2` population estimates as an approximation of exposure over the year interval. The data we have consists in January 1 estimates. These can also be used as Dec 31 estimates for the preceding year. Imagining forward, we'd like to have `P1` and `P2` as two columns.

The trick will be to first take our incoming `P1` estimate, convert it into `P2`, then join it back to the original `P2`. This last step is done with `inner_join()`, which takes two datasets and filters each down only to matching *join* (`by`) variables, then combines them into a single dataset.

```
glimpse(Pop)
```

```
## Rows: 39,031
## Columns: 4
## $ Country    <chr> "Belgium", "Belgium", "Belgium", "Belgium", "Belgium", "Bel~
## $ Year       <int> 2012, 2012, 2012, 2012, 2012, 2012, 2012, 2012, 2012, 2012,~
## $ Age        <dbl> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 1~
## $ Population <dbl> 62808, 64238, 63685, 63823, 62920, 62733, 61007, 60190, 588~
```

```
Pop <-
# incoming population data
  Pop %>%

  # move Year back by one so that it becomes
  # Dec 31
  mutate(Year = Year - 1) %>%

  # rename to P2 so we don't get confused
  rename(P2 = Population) %>%

  # join back to P1, STRICTLY, only overlapping cases
  inner_join(Pop, by = c("Country", "Year","Age")) %>%

  # call the 'new' Population one P1
  rename(P1 = Population) %>%

  # Calculate Exposure approximation
  mutate(Exposure = (P1 + P2) / 2)
```

We can (and maybe *should*) take care to re-combine these discrete steps as much as possible into a fluid pipeline. This will necessarily need to be in two parts, due to the above self-join. This pipeline is considerably longer than the final one we did for births, but you can imagine reading through it, or *stepping* through it, and should be able to follow its logical flow, even *verbalize* it. This takes some mental work, but we now should understand the purpose of each operation.

```
Pop <-

  # read in the data from Excel
  read_excel(path = here("Data", "demo_pjan.xlsx"),
             range = "A10:CZ510",
             na = ":") %>%

  # next we reshape to long format, stacking ages
  pivot_longer(cols = `Less than 1 year`:`Unknown`,
               names_to = "Age",
               values_to = "Population") %>%

  # remove NAs in populations
  filter(!is.na(Population)) %>%
```

```r
  # next we recode Age
  mutate(
    Age = case_when(
    Age == "Less than 1 year" ~ 0,
    Age == "Open-ended age class" ~ 100,
    Age == "Unknown" ~ NA_real_,
    TRUE ~ parse_number(Age)
  )) %>%

  # do some column renaming
  select(
    Country = `GEO/AGE`,
    Year = TIME,
    Age,
    Population) %>%

  # declare independent groups for unknown
  # age redistribution
  group_by(Country, Year) %>%

  # Move Unknown age up to a column, repeating for each subset
  mutate(UNK = Population[is.na(Age)],
         UNK = ifelse(is.na(UNK), 0, UNK)) %>%

  # remove Unknown age row
  filter(!is.na(Age)) %>%

  # ready to redistribute
  mutate(Population = Population + Population / sum(Population) * UNK,
         Year = as.integer(Year)) %>%

  # remove groups
  ungroup() %>%

  # remove instrumental column
  select(-UNK)
```

```
## Warning: 783 parsing failures.
## row col expected               actual
## 101  -- a number Open-ended age class
## 102  -- a number Unknown
## 203  -- a number Open-ended age class
## 204  -- a number Unknown
## 305  -- a number Open-ended age class
## ... ... ........ ....................
## See problems(...) for more details.
```

```r
Pop <-
  # incoming Jan 1 population
  Pop %>%

  # convert to Dec 31 population
  mutate(Year = Year - 1) %>%
```

```r
  # label accordingly
  rename(P2 = Population) %>%

  # join back to jan 1 population
  inner_join(Pop, by = c("Country", "Year","Age")) %>%

  # rename to P1 so we don't get confused
  rename(P1 = Population) %>%

  # exposure calculation trivial once we manage
  # to get P1 and P2 next to each other
  mutate(Exposure = (P1 + P2) / 2)
```

A similar join technique is used to add exposure data to the births data from earlier in this lesson. We'll call the final joined object `ASFR`.

```r
ASFR <-
  Births %>%

  # Take care of year integer conversion so
  # that we can successfully join!
  mutate(Year = as.integer(Year)) %>%

  # Join only those combination that are present in both
  # objects
  inner_join(Pop, by = c("Country","Year","Age")) %>%

  # calculate age specific fertility rates
  mutate(ASFR = Births / Exposure) %>%

  # now sort for easy visual insepction
  arrange(Country, Year, Age)
```

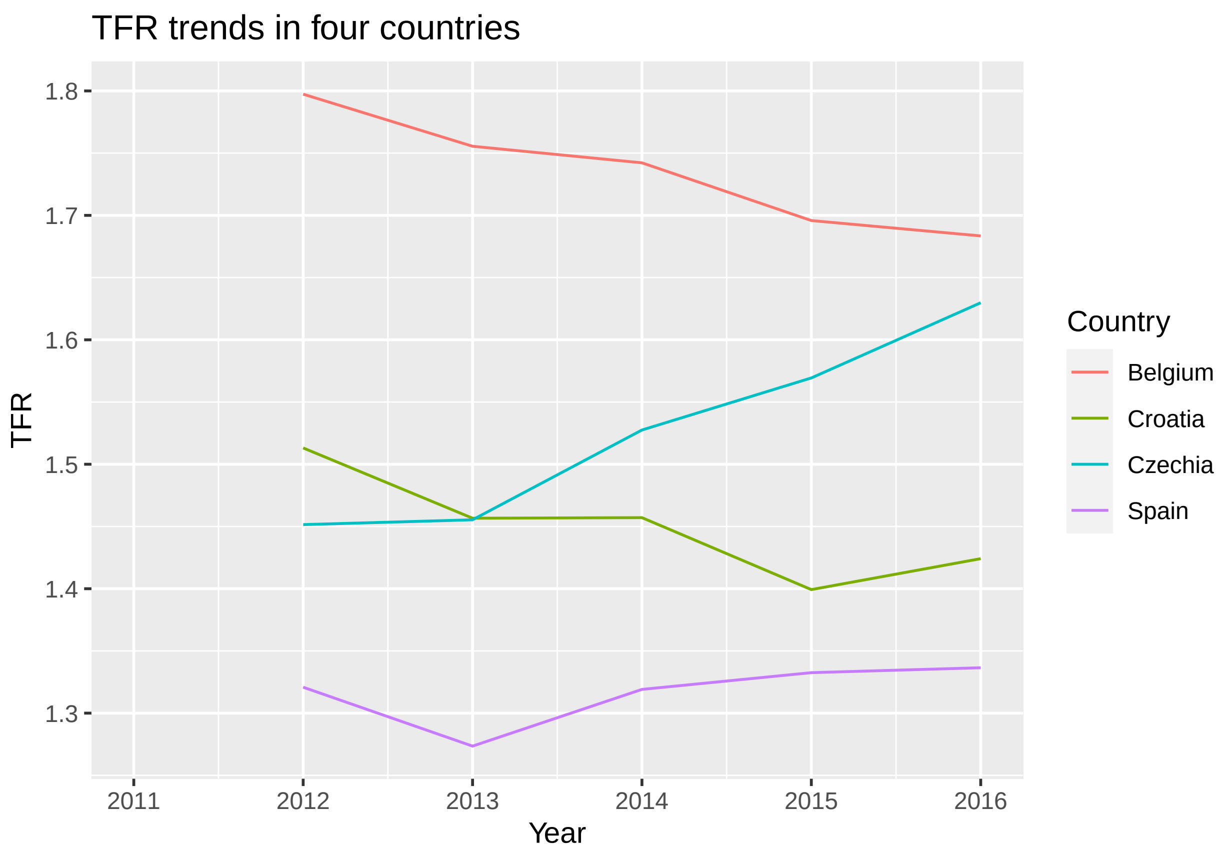
calculate summary measures:

```r
MAB <-
  MAB %>%
  mutate(Year = as.integer(Year))

Fert <-

  # incoming data (has rates and everything else)
  ASFR %>%

  # declare groups / subsets
  group_by(Country, Year) %>%

  # calculate TFR for subsets,
  # and also MAB that we can compare
  # with the birth-weighted on
  summarize(TFR = sum(ASFR),
            MAB2 = sum(Age * ASFR) / TFR + .5,
            .groups = "drop") %>%
```

```r
  # keep all rows
  full_join(MAB, by = c("Country","Year"))
```

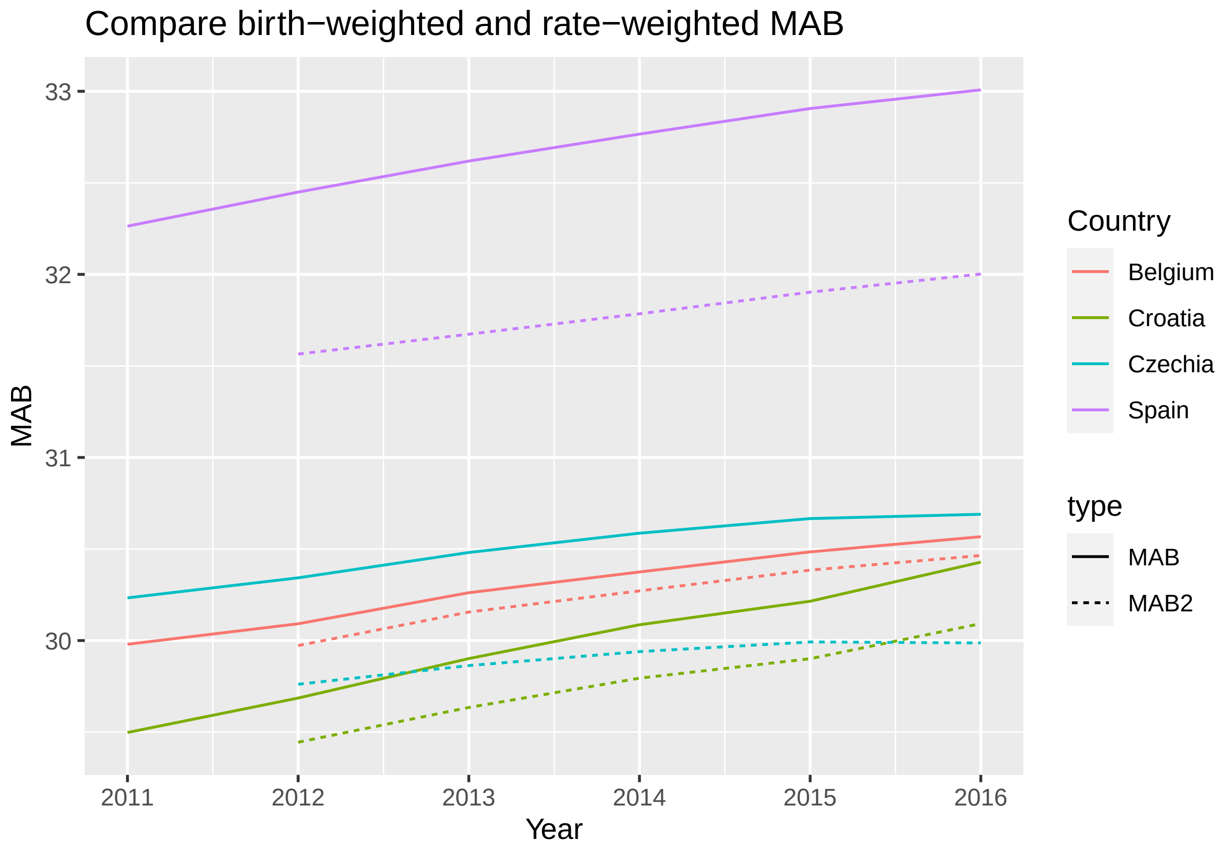
Visualize the results, as a teaser for Thursday:

```r
Fert %>%
  ggplot(aes(x = Year, y = TFR, group = Country, color = Country)) +
  geom_line() +
  labs(title = "TFR trends in four countries")
```



Likewise, we can compare MAB between the two definitions:

```r
Fert %>%
  select(-TFR) %>%
  pivot_longer(MAB:MAB2,
               names_to = "type",
               values_to = "MAB") %>%
  filter(!is.na(MAB)) %>%
  ggplot(aes(x = Year,
             y = MAB,
             group = interaction(Country, type),
             linetype = type,
             color = Country)) +
  geom_line() +
  labs(title = "Compare birth-weighted and rate-weighted MAB")
```



Compare birth−weighted and rate−weighted MAB
MAB
33
32
31
30
2011
2012
2013
2014
2015
2016
Year
Country
Belgium
Croatia
Czechia
Spain
type
MAB
MAB2