Barcelona Summer School of Demography

Module 1. Introduction to R

2. Tidy Pipelines

Tim Riffe
tim.riffe@ehu.eus

4 July 2023

Contents

1 Summary

In the first session we saw an intro to visualizing data that is tidy. Today we'll see an approach for turning messy data into tidy data: **data wrangling**. The exercises we'll do today are designed to expose you to some diversity in the functions needed to data wrangle. We'll see that complex processing chains (or pipelines) are composed of small and intuitive steps.

2 Data processing verbalized

We know we'll need tidyverse functions today, so let's just get this loaded:

```
library(tidyverse)
```

2.1 Introducing the data sources:

First make a `/Data/` folder in your project: look in the `Files` tab of the lower right panel, and click `New folder`. Let's download files to place there.

2.1.1 WPP 2022:

I downloaded a big spreadsheet of the most recent WPP demographic aggregates from here: https://population.un.org/wpp/Download/Standard/MostUsed/

The exact data file link is this: https://population.un.org/wpp/Download/Files/1_Indicators%20(Standard)/EXCEL_FILES/1_General/WPP2022_GEN_F01_DEMOGRAPHIC_INDICATORS_COMPACT_REV1.xlsx

Download it and stick it in your `/Data/` folder.

2.1.2 GPI 2023

I downloaded Global Peace Index data from here: https://www.visionofhumanity.org/public-release-data/

The exact data file link is this: https://www.visionofhumanity.org/wp-content/uploads/2023/06/GPI-2023-overall-scores-and-domains-2008-2023.xlsx Download it and stick it in your `/Data/` folder.

It contains data summarized by them as follows

```
The Global Peace Index (GPI) ranks 163 independent states and territories
according to their level of peacefulness. Produced by the Institute for Economics
and Peace (IEP), the GPI is the world's leading measure of global peacefulness.
```

2.1.3 Worldbank Gender Stats

I downloaded selected gender stats from here: https://databank.worldbank.org/source/gender-statistics#

That's an interactive selection tool with tons of options. I made an interactive selection that resulted in this exact file, which you can download from the module's github site: https://github.com/timriffe/BSSD2023Module1/raw/master/Data/P_Data_Extract_From_Gender_Statistics.xlsx

Download it and stick it in your `/Data/` folder.

The file contains a few haphazardly selected variables, which may or may not be interesting for us. The five variables included have different sources, including the World Bank itself, UNICEF, and assorted household surveys. That metadata can be found in the second spreadsheet tab.

Also from World Bank, I downloaded this file: https://github.com/timriffe/BSSD2023Module1/raw/master/Data/P_Popular%20Indicators.xlsx Which contains a large set of popular indicators, of which we'll use GDP per capita and CO2 emissions per capita (from Climate Watch Historical GHG Emissions).

2.1.4 ILO data

From this website https://ilostat.ilo.org/data/data-catalogue/ I downloaded this file, which you can save in `/Data/`: https://www.ilo.org/ilostat-files/WEB_bulk_download/indicator/SDG_0851_SEX_OCU_NB_A.csv.gz This contains information on the relative wages of men and women.

I also got labor force size estimates by gender from this file, so get that too! https://www.ilo.org/ilostat-files/WEB_bulk_download/indicator/EAP_TEAP_SEX_MTS_NB_A.csv.gz

2.2 The objective

We want to harmonize each of these datasets to be able to join them by country and year. For each dataset, we'll need to succeed at:

1. Read in the relevant part of the file

2. Harmonize variables as necessary, most notably country codes
3. Select variables to keep
4. Rename variables to easy standard names

For some datasets we'll want to derive secondary quantities. When we're done with each dataset, we can join them.

2.3 The tidyverse tools

To achieve these things, we will use a bunch of basic data handling functions from a data wrangling package called `dplyr`, and elsewhere from the tidyverse as well. Here's an overview of the tools we'll use today:

- `read_excel()` to read in cell ranges from an Excel spreadsheet.
- `read_csv()` to read in from csv files
- `filter()` to subset rows
- `select()` to pick out columns
- `rename()` to change column names
- `pivot_longer()` to stack a range of columns into a single column, i.e. make the data longer.
- `pivot_wider()` to do the opposite
- `mutate()` to create a new variable
- `separate()` to split a column into two or more based on a separator character
- `full_join()` for a lossless merge of all data files (once they're harmonized).

Here's a nice cheat sheet of these tools: https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf

Let's get going!

2.4 Step 1: Harmonize selected WPP variables

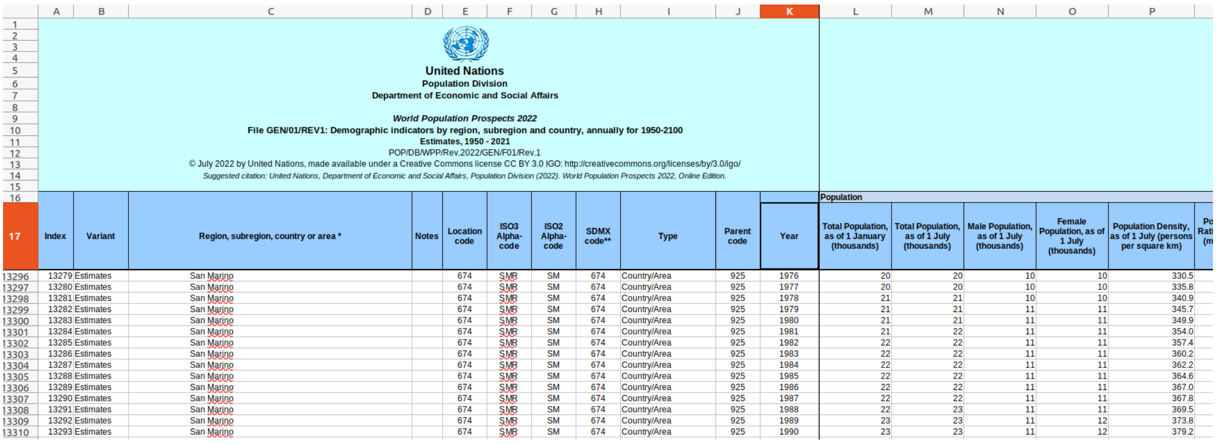
The WPP data looks like this if we look at it in a spreadsheet program:

Our objective will be to harmonize it to end up with columns for `iso3`, `year`, `tfr`, `e0f`, `e0m`, `popf`, `popm`, where e0 means life expectancy at birth.

That is, we want to select and rename some columns, and we want to filter only rows referring to countries with ISO3 codes. ISO3 codes are the easiest way to merge national datasets. Way easier than trying to match country names!

Here's how to successfully read in the dataset:

```
library(readxl)
wpp <- read_excel("Data/WPP2022_GEN_F01_DEMOGRAPHIC_INDICATORS_COMPACT_REV1.xlsx",
                  skip = 16,
                  col_types = c(rep("text",10),rep("numeric",55)),
                  na = "...")
```



United Nations
Population Division
Department of Economic and Social Affairs

World Population Prospects 2022
File GEN/01/REV1: Demographic indicators by region, subregion and country, annually for 1950-2100
Estimates, 1950 - 2021
POP/DB/WPP/Rev.2022/GEN/F01/Rev.1

Suggested citation: United Nations, Department of Economic and Social Affairs, Population Division (2022). World Population Prospects 2022, Online Edition.

	Index	Variant	Region, subregion, country or area *	Notes	Location code	ISO3 Alpha-code	ISO2 Alpha-code	SDMX code**	Type	Parent code	Year	Population: Total Population, as of 1 January (thousands)	Total Population, as of 1 July (thousands)	Male Population, as of 1 July (thousands)	Female Population, as of 1 July (thousands)	Population Density, as of 1 July (persons per square km)	Po Rati (m
13296	13279	Estimates	San Marino		674	SMR	SM	674	Country/Area	925	1976	20	20	10	10	330.5	
13297	13280	Estimates	San Marino		674	SMR	SM	674	Country/Area	925	1977	20	20	10	10	335.8	
13298	13281	Estimates	San Marino		674	SMR	SM	674	Country/Area	925	1978	21	21	10	10	340.9	
13299	13282	Estimates	San Marino		674	SMR	SM	674	Country/Area	925	1979	21	21	11	11	345.7	
13300	13283	Estimates	San Marino		674	SMR	SM	674	Country/Area	925	1980	21	21	11	11	349.9	
13301	13284	Estimates	San Marino		674	SMR	SM	674	Country/Area	925	1981	21	22	11	11	354.0	
13302	13285	Estimates	San Marino		674	SMR	SM	674	Country/Area	925	1982	22	22	11	11	357.4	
13303	13286	Estimates	San Marino		674	SMR	SM	674	Country/Area	925	1983	22	22	11	11	360.2	
13304	13287	Estimates	San Marino		674	SMR	SM	674	Country/Area	925	1984	22	22	11	11	362.2	
13305	13288	Estimates	San Marino		674	SMR	SM	674	Country/Area	925	1985	22	22	11	11	364.6	
13306	13289	Estimates	San Marino		674	SMR	SM	674	Country/Area	925	1986	22	22	11	11	367.0	
13307	13290	Estimates	San Marino		674	SMR	SM	674	Country/Area	925	1987	22	22	11	11	367.8	
13308	13291	Estimates	San Marino		674	SMR	SM	674	Country/Area	925	1988	22	23	11	11	369.5	
13309	13292	Estimates	San Marino		674	SMR	SM	674	Country/Area	925	1989	23	23	11	12	373.8	
13310	13293	Estimates	San Marino		674	SMR	SM	674	Country/Area	925	1990	23	23	11	12	379.2	

Figure 1: The WPP data, as downloaded

The main argument is the file path, while everything else is telling `read_excel()` how to do things.

- `skip = 16` tells it to skip the first 16 rows before reading. We know this from visual inspection.
- `col_types` is given an exhaustive vector of the data type we want used for each column. You can skip this argument, in which case `read_excel()` guesses the column type, but for this dataset it will guess wrong for most columns! Usually it gets types right, but not here. To create the vector I use `c()` to join two vectors, and `rep()` to repeat a value a given number of times. We end up with a vector with 65 elements, the first 10 of which are `"text"`, and the last 55 of which are `"numeric"`. I got those values from visual inspection of the spreadsheet.
- `na = "..."` Different statistical agencies have different ways of specifying missing data; this one uses `...`, which I found from visual inspection.

Now we have an object `wpp`, a tibble with 20596 rows and 65 columns. We'll throw out most columns for this exercise. Let's use the `select()` function to both select and rename columns, like so:

```
wpp = select(
        .data = wpp,
        iso3 = `ISO3 Alpha-code`,
        year = Year,
        tfr = `Total Fertility Rate (live births per woman)`,
        e0f = `Female Life Expectancy at Birth (years)`,
        e0m = `Male Life Expectancy at Birth (years)`,
        popf = `Female Population, as of 1 July (thousands)`,
        popm = `Male Population, as of 1 July (thousands)`)
```

This action follows the form `new name = old name`. Finally, we can select down to rows with valid ISO3 codes:

```
wpp = filter(
        .data = wpp,
        !is.na(iso3))
```

Only countries have ISO3 codes, whereas the UN gives estimates also for various kinds of geographic aggregates of populations, which we don't need.

Now we have arrived at the objective format:

```
head(wpp)
```

```
## # A tibble: 6 x 7
##   iso3   year   tfr   e0f   e0m  popf  popm
##   <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 BDI    1950  6.92  41.9  39.2 1174. 1080.
## 2 BDI    1951  6.91  42.2  39.3 1199. 1105.
## 3 BDI    1952  6.9   42.4  39.6 1223. 1129.
## 4 BDI    1953  6.92  42.6  39.9 1246. 1153.
## 5 BDI    1954  6.92  43.0  40.1 1270. 1177.
## 6 BDI    1955  6.93  43.3  40.4 1294. 1201.
```

Note we achieved this in three steps: `read_excel()`, `select()`, and `filter()`. These steps can be joined into a single sequence to be executed like so:

```
wpp <-
  read_excel(
    "Data/WPP2022_GEN_F01_DEMOGRAPHIC_INDICATORS_COMPACT_REV1.xlsx",
    skip = 16,
    col_types = c(rep("text", 10),
                  rep("numeric", 55)),
    na = "...") |>
 select(
   iso3 = `ISO3 Alpha-code`,
   year = Year,
   tfr = `Total Fertility Rate (live births per woman)`,
   e0f = `Female Life Expectancy at Birth (years)`,
   e0m = `Male Life Expectancy at Birth (years)`,
   popf = `Female Population, as of 1 July (thousands)`,
   popm = `Male Population, as of 1 July (thousands)`) |>
  filter(!is.na(iso3))
```

Here the funny symbol `|>` is R's native pipe operator. You can get it by typing `Ctrl Alt m` (`Cmd Option m` for mac). You might get this symbol instead: '%>%. That does the same thing, coming from the `magrittr` package that ships with `tidyverse`. They are the same for us. If you have a preference, you can change the default pipe operator to the native `|>` by clicking `Tools | Global Options | Code | Use Native Pipe Operator`.

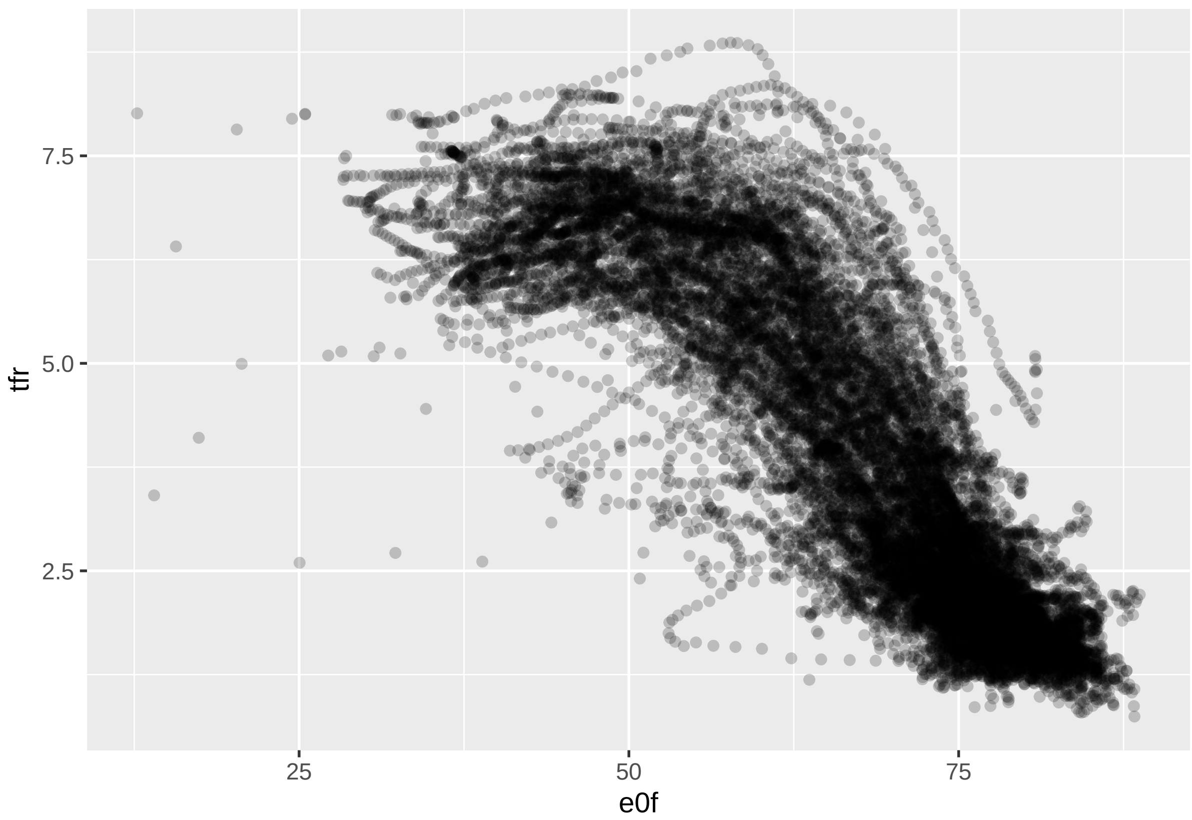
The pipe merely send the result of the left-side operation to the right-side operation, forming an execution sequence. Doing this makes you code more compact, more regular, and easier to visually inspect and *verbalize*. In this case, the verbalization is *read then select and rename columns then filter rows*. The pipe is like the *then*.

Whenever we read in data like this, we should get a sense of it. Here are some quick, plots of diagnostic value:

1. TFR by female life expectancy (you can intuit trajectories), sort of a logistic trajectory overall.

```
wpp |>
  ggplot(aes(x = e0f, y = tfr)) +
  geom_point(alpha = .2)
```

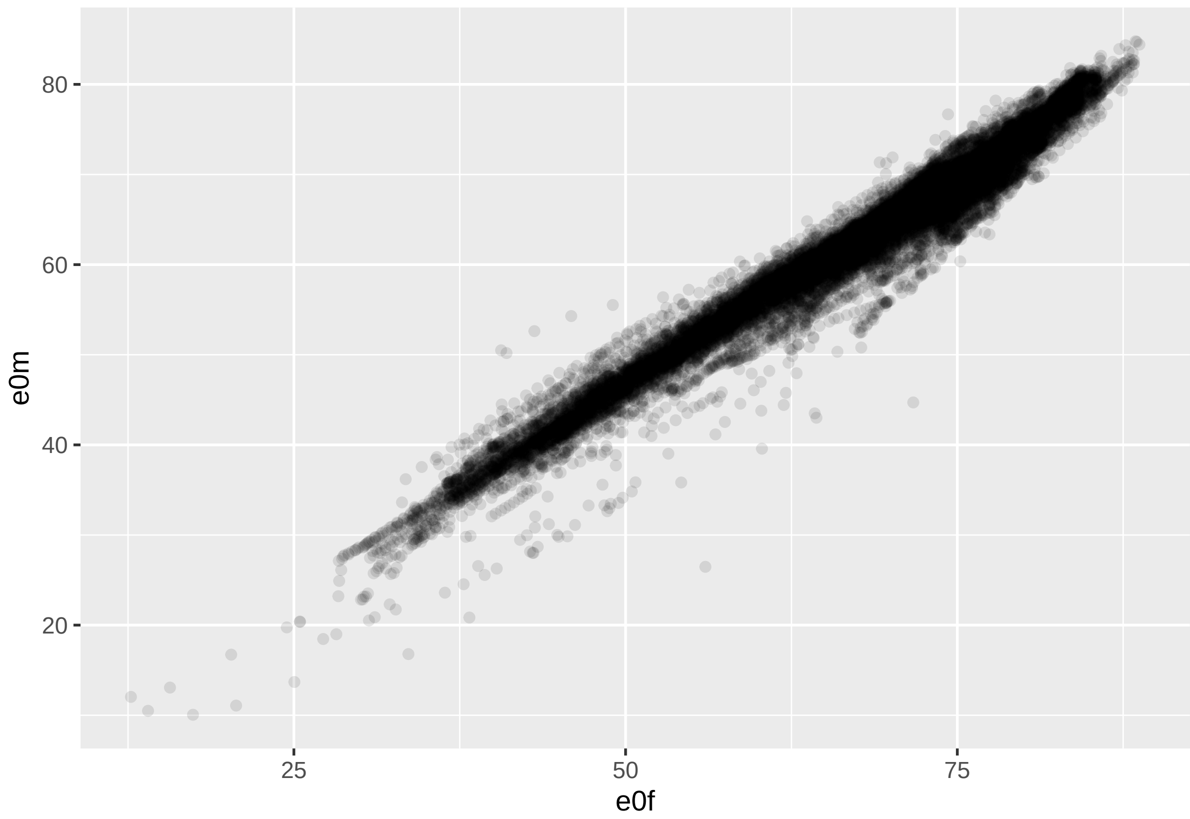
```
## Warning: Removed 72 rows containing missing values (‘geom_point()‘).
```



2. Male by female life expectancy, pretty linear.

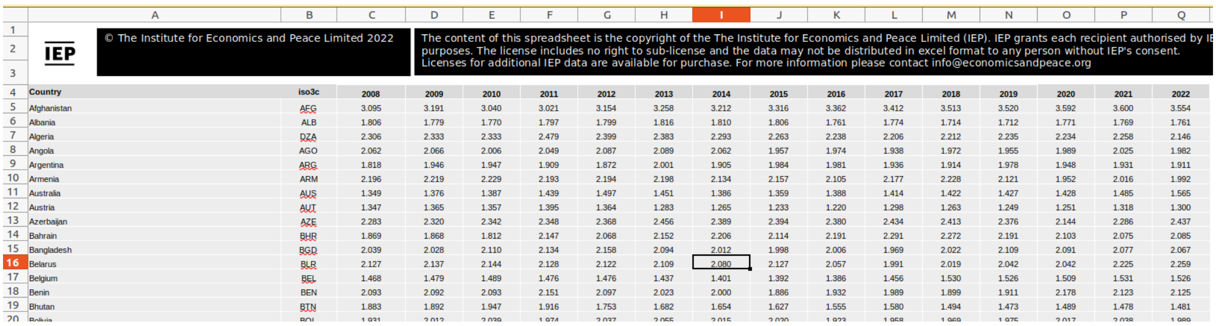
```
wpp |>
  ggplot(aes(x = e0f, y = e0m)) +
  geom_point(alpha = .1)
```

```
## Warning: Removed 72 rows containing missing values (‘geom_point()‘).
```



2.5 Step 2: Harmonize GPI data

The GPI spreadsheet is in wide format, with years spread over columns. This is a super common way for data to be delivered. We'll use `pivot_longer()` to stack the columns.



IEP

© The Institute for Economics and Peace Limited 2022

The content of this spreadsheet is the copyright of the The Institute for Economics and Peace Limited (IEP). IEP grants each recipient authorised by IE
purposes. The license includes no right to sub-license and the data may not be distributed in excel format to any person without IEP's consent.
Licenses for additional IEP data are available for purchase. For more information please contact info@economicsandpeace.org

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
4	Country	iso3c	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022
5	Afghanistan	AFG	3.095	3.191	3.040	3.021	3.154	3.258	3.212	3.316	3.362	3.412	3.513	3.520	3.592	3.600	3.554
6	Albania	ALB	1.806	1.779	1.770	1.797	1.799	1.816	1.810	1.806	1.761	1.774	1.714	1.712	1.771	1.769	1.761
7	Algeria	DZA	2.306	2.333	2.333	2.479	2.399	2.383	2.293	2.263	2.238	2.206	2.212	2.235	2.234	2.258	2.146
8	Angola	AGO	2.062	2.066	2.006	2.049	2.087	2.089	2.062	1.957	1.974	1.938	1.972	1.955	1.989	2.025	1.982
9	Argentina	ARG	1.818	1.946	1.947	1.909	1.872	2.001	1.905	1.984	1.981	1.936	1.914	1.978	1.948	1.931	1.911
10	Armenia	ARM	2.196	2.219	2.229	2.193	2.194	2.198	2.134	2.157	2.105	2.177	2.228	2.121	1.952	2.016	1.992
11	Australia	AUS	1.349	1.376	1.387	1.439	1.497	1.451	1.386	1.359	1.388	1.414	1.422	1.427	1.428	1.485	1.565
12	Austria	AUT	1.347	1.365	1.357	1.395	1.364	1.283	1.265	1.233	1.220	1.298	1.263	1.249	1.251	1.318	1.300
13	Azerbaijan	AZE	2.283	2.320	2.342	2.348	2.368	2.456	2.389	2.394	2.380	2.434	2.413	2.376	2.144	2.286	2.437
14	Bahrain	BHR	1.869	1.868	1.812	2.147	2.068	2.152	2.206	2.114	2.191	2.291	2.272	2.191	2.103	2.075	2.085
15	Bangladesh	BGD	2.039	2.028	2.110	2.134	2.158	2.094	2.012	1.998	2.006	1.969	2.022	2.109	2.091	2.077	2.067
16	Belarus	BLR	2.127	2.137	2.144	2.128	2.122	2.109	2.080	2.127	2.057	1.991	2.019	2.042	2.042	2.225	2.259
17	Belgium	BEL	1.468	1.479	1.489	1.476	1.476	1.437	1.401	1.392	1.386	1.456	1.530	1.526	1.509	1.531	1.526
18	Benin	BEN	2.093	2.092	2.093	2.151	2.097	2.023	2.000	1.886	1.932	1.989	1.899	1.911	2.178	2.123	2.125
19	Bhutan	BTN	1.883	1.892	1.947	1.916	1.753	1.682	1.654	1.627	1.555	1.580	1.494	1.473	1.489	1.478	1.481
20	Bolivia	BOL	1.931	2.012	2.039	1.974	2.037	2.055	2.015	2.020	1.923	1.958	1.969	1.975	2.017	2.038	1.989

Figure 2: The GPI data, as downloaded

The objective format will to have columns `iso3`, `year`, and `gpi`, and that's it. Here's the final pipeline:

```
gpi <-
  read_excel("Data/GPI-2022-overall-scores-and-domains-2008-2022.xlsx",
             sheet = 2, skip = 3) |>
  pivot_longer(-c(1, 2),
               names_to = "year",
```

```r
                 values_to = "gpi") |>
  select(iso3 = iso3c, year, gpi) |>
  mutate(year = as.integer(year))

head(gpi)
```

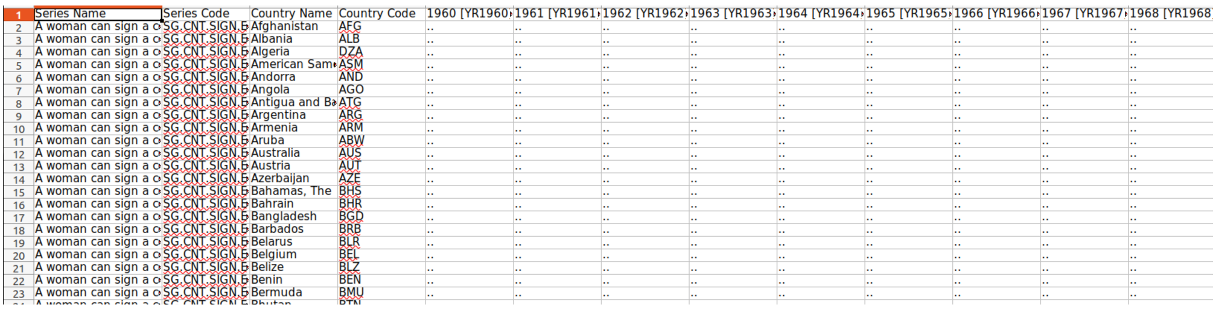
```
## # A tibble: 6 x 3
##   iso3   year   gpi
##   <chr> <int> <dbl>
## 1 AFG    2008  3.10
## 2 AFG    2009  3.19
## 3 AFG    2010  3.04
## 4 AFG    2011  3.02
## 5 AFG    2012  3.15
## 6 AFG    2013  3.26
```

Let's dissect this:

- `pivot_longer()` is the hero here. We can give it a positive specification of columns to act on, or in this case tell it which columns to exclude from the action (`-c(1,2)` means skip the first and second columns. `names_to` will be a new variable collecting the column names, and `values_to` will be the new value column. What we before a value over rows and columns is now just a single column of values, `gpi`. I use lowercase for faster typing.
- `select()` picks out the three columns we want (discarding `Country`) and renames `iso3c` to `iso3`.
- `mutate()` coerces the `year` column from character to integer values.

2.6 Step 3: Harmonize World Bank data

The first file of World Bank data looks like this:



1	Series Name	Series Code	Country Name	Country Code	1960 [YR1960	1961 [YR1961	1962 [YR1962	1963 [YR1963	1964 [YR1964	1965 [YR1965	1966 [YR1966	1967 [YR1967	1968 [YR1968
2	A woman can sign a c	SG.CNT.SIGN.E	Afghanistan	AFG	..	..	..	..	..	..	..	..	..
3	A woman can sign a c	SG.CNT.SIGN.E	Albania	ALB	..	..	..	..	..	..	..	..	..
4	A woman can sign a c	SG.CNT.SIGN.E	Algeria	DZA	..	..	..	..	..	..	..	..	..
5	A woman can sign a c	SG.CNT.SIGN.E	American Sam	ASM	..	..	..	..	..	..	..	..	..
6	A woman can sign a c	SG.CNT.SIGN.E	Andorra	AND	..	..	..	..	..	..	..	..	..
7	A woman can sign a c	SG.CNT.SIGN.E	Angola	AGO	..	..	..	..	..	..	..	..	..
8	A woman can sign a c	SG.CNT.SIGN.E	Antigua and Ba	ATG	..	..	..	..	..	..	..	..	..
9	A woman can sign a c	SG.CNT.SIGN.E	Argentina	ARG	..	..	..	..	..	..	..	..	..
10	A woman can sign a c	SG.CNT.SIGN.E	Armenia	ARM	..	..	..	..	..	..	..	..	..
11	A woman can sign a c	SG.CNT.SIGN.E	Aruba	ABW	..	..	..	..	..	..	..	..	..
12	A woman can sign a c	SG.CNT.SIGN.E	Australia	AUS	..	..	..	..	..	..	..	..	..
13	A woman can sign a c	SG.CNT.SIGN.E	Austria	AUT	..	..	..	..	..	..	..	..	..
14	A woman can sign a c	SG.CNT.SIGN.E	Azerbaijan	AZE	..	..	..	..	..	..	..	..	..
15	A woman can sign a c	SG.CNT.SIGN.E	Bahamas, The	BHS	..	..	..	..	..	..	..	..	..
16	A woman can sign a c	SG.CNT.SIGN.E	Bahrain	BHR	..	..	..	..	..	..	..	..	..
17	A woman can sign a c	SG.CNT.SIGN.E	Bangladesh	BGD	..	..	..	..	..	..	..	..	..
18	A woman can sign a c	SG.CNT.SIGN.E	Barbados	BRB	..	..	..	..	..	..	..	..	..
19	A woman can sign a c	SG.CNT.SIGN.E	Belarus	BLR	..	..	..	..	..	..	..	..	..
20	A woman can sign a c	SG.CNT.SIGN.E	Belgium	BEL	..	..	..	..	..	..	..	..	..
21	A woman can sign a c	SG.CNT.SIGN.E	Belize	BLZ	..	..	..	..	..	..	..	..	..
22	A woman can sign a c	SG.CNT.SIGN.E	Benin	BEN	..	..	..	..	..	..	..	..	..
23	A woman can sign a c	SG.CNT.SIGN.E	Bermuda	BMU	..	..	..	..	..	..	..	..	..

Figure 3: The World Bank data, as downloaded

Note that each row gives a time series of a single variable, and I have selected five variables! This will have some more steps. Here's the final pipeline:

```
gender <-
  read_excel("Data/P_Data_Extract_From_Gender_Statistics.xlsx",
             na="..") |>
  pivot_longer(-c(1:4),names_to = "year",values_to = "value") |>
  select(variable = `Series Code`,
         iso3 = `Country Code`,
         year, value) |>
  pivot_wider(names_from = "variable",
              values_from = "value") |>
  separate(year, into = c("year",NA),sep=" ",convert=TRUE) |>
  rename(sign_contract = SG.CNT.SIGN.EQ,
         remarry = SG.REM.RIGT.EQ,
         ind_work = SG.IND.WORK.EQ,
         births_attend = SH.STA.BRTC.ZS,
         births_completeness = SP.REG.BRTH.ZS) |>
  filter(year >= 1970)

head(gender)
```

```
## # A tibble: 6 x 7
##   iso3   year sign_contract remarry ind_work births_attend births_completeness
##   <chr> <int>         <dbl>   <dbl>    <dbl>         <dbl>               <dbl>
## 1 AFG    1970             0       0        0            NA                  NA
## 2 AFG    1971             0       0        0            NA                  NA
## 3 AFG    1972             0       0        0            NA                  NA
## 4 AFG    1973             0       0        0            NA                  NA
## 5 AFG    1974             0       0        0            NA                  NA
## 6 AFG    1975             0       0        0            NA                  NA
```

We end up with five potentially interesting variables:

1. `sign_contract`: A woman can sign a contract in the same way as a man (1=yes; 0=no)
2. `remarry`: A woman has the same rights to remarry as a man (1=yes; 0=no)
3. `ind_work`: A woman can work in an industrial job in the same way as a man (1=yes; 0=no)
4. `births_attend`: Births attended by skilled health staff (% of total)
5. `births_completeness`: Completeness of birth registration (%)

The pipeline dissection:

- `read_excel()` in this case guess column types correctly, we just need to tell it the *no data* signifier `..` using `na = ".."`.
- `pivot_longer()` stacks the years, which were intitially spread over columns. This means we here end up with a single value column containing different variables in it.

- `select()` is used for variable renaming and selecting in this case, for simplified typing.
- `pivot_wider()` unstacks the different variable series, creating a unique column for each variable (part of the tidy definition). Note that we use `names_from` and `values_from` rather than `names_to` and `values_to`!
- `rename()` gives the original codes some more memorable names
- `filter()` picks out more recent years that have more valid observations. We still have lots of NAs afterwards, but no big deal.

Note, this source has many many more gender-related variables, as do other providers, which you may find interesting. I have no theoretical motivation for the selection made here.

2.6.1 Harmonize the second World Bank extract

The dataset `P_Popular Indicators` has many many interesting macro variables, of which we select only GDP per capita and CO2 emissions per capita. It is formatted as before, with time over columns and series in rows.

```
gdp <- read_excel("Data/P_Popular Indicators.xlsx",
                  na = "..") |>
# Emissions data are sourced from Climate Watch Historical GHG Emissions (1990-2020). 2023.
  filter(`Series Code` %in% c("NY.GDP.PCAP.CD","EN.ATM.CO2E.PC")) |>
  pivot_longer(-c(1:4),
               names_to = "year",
               values_to = "value") |>
  select(-`Series Code`) |>
  pivot_wider(names_from = `Series Name`,
              values_from = value) |>
  separate(year,
           into = c("year", NA),
           sep = " ",
           convert = TRUE) |>
  rename(iso3 = `Country Code`,
         co2pc = `CO2 emissions (metric tons per capita)`,
         gdppc = `GDP per capita (current US$)`)
```

The dissection gives some new lessons, however:

- `filter()` uses an operator `%in%` that is great for checking set membership. This operator is super useful.
- `pivot_longer()` collects the years as before
- `select()` shows us a negative selection. If we don't throw out this column before the next step then we'll do the wrong thing. Try it!
- `pivot_wider()` puts each variable in a new column, unstacking `value`

- `separate()` is used to convert `"1960 [YR1960]"` into `1960`. There are other ways one might do this. For example, selecting out the first four characters and then parsing to integer... In this case, we split the column into two columns, using the space as the separator. The new column is still called `year`, whereas the second column is discarded by giving `NA` as the name. The `convert` option tells the function to guess the intended data type, since in this case we're always text parsing, and often the extracted characters give a numeric value.
- `rename()` is used to simplify column names to something less verbose.

Exercise: can you imagine some diagnostic plots for this dataset and create them?

2.7 Step 4: Harmonize the ILO data

The ILO data look like this:

1	ref_area	indicator	source	sex	classif1	time	obs_value	obs_status	note_classif	note_indicator	note_source
2	AFG	SDG_0851_SEX_OCU_N	BA:15715	SEX_T	OCU_SKILL_TOTAL	2020	90.8	B		T30:110_I11:264	R1:3513
3	AFG	SDG_0851_SEX_OCU_N	BA:15715	SEX_T	OCU_SKILL_L3-4	2020	151.94	B		T30:110_I11:264	R1:3513
4	AFG	SDG_0851_SEX_OCU_N	BA:15715	SEX_T	OCU_SKILL_L2	2020	61.73	B		T30:110_I11:264	R1:3513
5	AFG	SDG_0851_SEX_OCU_N	BA:15715	SEX_T	OCU_SKILL_L1	2020	43.53	B		T30:110_I11:264	R1:3513
6	AFG	SDG_0851_SEX_OCU_N	BA:15715	SEX_T	OCU_SKILL_X	2020	63.66	B		T30:110_I11:264	R1:3513
7	AFG	SDG_0851_SEX_OCU_N	BA:15715	SEX_M	OCU_SKILL_TOTAL	2020	81.53	B		T30:110_I11:264	R1:3513
8	AFG	SDG_0851_SEX_OCU_N	BA:15715	SEX_M	OCU_SKILL_L3-4	2020	135.32	B		T30:110_I11:264	R1:3513
9	AFG	SDG_0851_SEX_OCU_N	BA:15715	SEX_M	OCU_SKILL_L2	2020	60.31	B		T30:110_I11:264	R1:3513
10	AFG	SDG_0851_SEX_OCU_N	BA:15715	SEX_M	OCU_SKILL_L1	2020	42.67	B		T30:110_I11:264	R1:3513
11	AFG	SDG_0851_SEX_OCU_N	BA:15715	SEX_M	OCU_SKILL_X	2020	62.85	B		T30:110_I11:264	R1:3513
12	AFG	SDG_0851_SEX_OCU_N	BA:15715	SEX_F	OCU_SKILL_TOTAL	2020	183.93	B		T30:110_I11:264	R1:3513
13	AFG	SDG_0851_SEX_OCU_N	BA:15715	SEX_F	OCU_SKILL_L3-4	2020	216.61	B		T30:110_I11:264	R1:3513
14	AFG	SDG_0851_SEX_OCU_N	BA:15715	SEX_F	OCU_SKILL_L2	2020	95.67	B		T30:110_I11:264	R1:3513
15	AFG	SDG_0851_SEX_OCU_N	BA:15715	SEX_F	OCU_SKILL_L1	2020	52.04	B		T30:110_I11:264	R1:3513
16	AFG	SDG_0851_SEX_OCU_N	BA:15715	SEX_F	OCU_SKILL_X	2020	132.04	U		T30:110_I11:264	R1:3513
17	AFG	SDG_0851_SEX_OCU_N	BA:15715	SEX_T	OCU_ISCO08_TOTAL	2020	90.8	B		T30:110_I11:264	R1:3513

Figure 4: The ILO data, as downloaded

In this case, we have indicators still stacked, but at least years are also already stacked. You can see from the start that there are data gaps for certain years / places as well. That's something we might want to remedy using interpolation of some kind, but that doesn't fit in today's lesson.

In this data, the interesting thing for us are male and female wages, and we're content to end up with a simple national aggregate wage ratio at the end of the pipeline.

```
wage <-
  read_csv("Data/SDG_0851_SEX_OCU_NB_A.csv.gz",
           show_col_types = FALSE) |>
  filter(classif1 == "OCU_SKILL_TOTAL") |>
  select(iso3 = ref_area,
         sex,
         year = time,
         value = obs_value) |>
  pivot_wider(names_from = sex,
              values_from = value) |>
  mutate(wage_ratio = SEX_F / SEX_M) |>
  select(iso3, year, wage_ratio) |>
  arrange(iso3,year)

head(wage)
```

```
## # A tibble: 6 x 3
##   iso3   year wage_ratio
##   <chr> <dbl>      <dbl>
## 1 AFG    2014      0.927
## 2 AFG    2020      2.26
## 3 AGO    2019      0.737
## 4 AGO    2021      0.873
## 5 ALB    2002      0.765
## 6 ALB    2012      0.832
```

The dissection of this pipeline shows us some new tricks as well:

- `read_csv()` is used even though the csv file is g-zipped :-). We make it less verbose by telling it not to spit the colum metadata to the console `show_col_types = FALSE`.
- `filter()` picks out just the national totals. We could also drill down to grouped occupation codes, apparently, but let's discard those for now.
- `select()` just gives us something more parsimonious and manageable
- `pivot_wider()` puts mens and women's wages side by side.
- `mutate()` is used to make us a new column for the wage ratio (women / men)
- `arrange()` sorts the result first by country, then year within country.

Exercise: read in the file `Data/EAP_TEAP_SEX_MTS_NB_A.csv.gz` and create an object `lf` containing labor for size by gender. The resulting columns should be like this:

```
  iso3   year   lfm   lff
  <chr> <dbl> <dbl> <dbl>
1 AFG    2021 6217. 1938.
2 AFG    2020 5514. 1371.
3 AFG    2017 5599. 1603.
4 AFG    2014 5731. 1874.
5 AFG    2012 5432. 1098.
```

The code should be quite similar to the other ILO dataset code. `lff` and `lfm` come from the variable `classif1 == "MTS_AGGREGATE_TOTAL"`, and then putting men and women side by side.

And finally a note, ILO also delivers age-sex stratified data, which may be interesting to some of you :-)

2.8 Step 5 Join the data

Now, if you succeeded in creating the `lf` dataset we should have several data objects: `wpp`, `gpi`, `gender`, `gdp`, `wage`, and `lf`. There are different ways to join data. Here let's do a lossless-join, meaning match whatever we can, but throw nothing away (i.e. pad with `NA` values where needed). Then later, if needed, we can always filter down to just the interesting subsets, depending on what's interesting.

This sort of join is called a `full_join()`, we just need to tell the function what two datasets to join, and which variables to treat as the key for matching. We need to do this two pieces at a time, like so:

```
wpp |>
  full_join(gpi, by = c("iso3","year")) |>
  full_join(gender, by = c("iso3", "year"))
```

```
## # A tibble: 19,893 x 13
##    iso3   year   tfr   e0f   e0m  popf  popm   gpi sign_contract remarry
##    <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>         <dbl>   <dbl>
##  1 BDI    1950  6.92  41.9  39.2 1174. 1080.    NA            NA      NA
##  2 BDI    1951  6.91  42.2  39.3 1199. 1105.    NA            NA      NA
##  3 BDI    1952  6.9   42.4  39.6 1223. 1129.    NA            NA      NA
##  4 BDI    1953  6.92  42.6  39.9 1246. 1153.    NA            NA      NA
##  5 BDI    1954  6.92  43.0  40.1 1270. 1177.    NA            NA      NA
##  6 BDI    1955  6.93  43.3  40.4 1294. 1201.    NA            NA      NA
##  7 BDI    1956  6.93  43.5  40.6 1319. 1226.    NA            NA      NA
##  8 BDI    1957  6.93  43.7  40.9 1343. 1251.    NA            NA      NA
##  9 BDI    1958  6.95  44.0  41.0 1368. 1275.    NA            NA      NA
## 10 BDI    1959  6.98  44.2  41.3 1393. 1301.    NA            NA      NA
## # i 19,883 more rows
## # i 3 more variables: ind_work <dbl>, births_attend <dbl>,
## #   births_completeness <dbl>
```

Exercise: join all six datasets in a single pipeline, assigning the result to a new data object called `gapreminder`. But don't save the result just yet, there's still some cleanup to do!

```
write_csv(gapreminder, file = "Data/gapreminder.csv")
```

You might notice that we have a `Country Name` column, which we might as well move towards the front of the dataset:

```
head(gapreminder)
```

```
## # A tibble: 6 x 19
##   iso3   year   tfr   e0f   e0m  popf  popm   gpi sign_contract remarry ind_work
##   <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>         <dbl>   <dbl>    <dbl>
## 1 BDI    1950  6.92  41.9  39.2 1174. 1080.    NA            NA      NA       NA
## 2 BDI    1951  6.91  42.2  39.3 1199. 1105.    NA            NA      NA       NA
## 3 BDI    1952  6.9   42.4  39.6 1223. 1129.    NA            NA      NA       NA
## 4 BDI    1953  6.92  42.6  39.9 1246. 1153.    NA            NA      NA       NA
## 5 BDI    1954  6.92  43.0  40.1 1270. 1177.    NA            NA      NA       NA
## 6 BDI    1955  6.93  43.3  40.4 1294. 1201.    NA            NA      NA       NA
```

```
## # i 8 more variables: births_attend <dbl>, births_completeness <dbl>,
## #   ‘Country Name‘ <chr>, co2pc <dbl>, gdppc <dbl>, wage_ratio <dbl>,
## #   lfm <dbl>, lff <dbl>
```

```
gapreminder <-
  gapreminder |>
  relocate(`Country Name`, 2) |>
  rename(country = `Country Name`)
```

And you might then notice that some country names aren't present for particular ISO3 codes! We can fill these in, or even easier, overwrite the country column using a coding service, like the `countrycode` `R` package, which exists for this sort of thing. Install it (note in my markdown file I use `eval = FALSE`:

```
install.packages("countrycode")
```

```
library(countrycode)
gapreminder<-
  gapreminder |>
  mutate(country = countrycode(sourcevar = iso3,
                               origin = "iso3c",
                               destination = "country.name",
                               warn = FALSE)) |>
  filter(!is.na(country))
```

Here, we use the function `countrycode()` to create fresh country names from an ISO3 lookup table. Note, many of the values in `iso3` will fail to reference country names. These come from the World Bank geographic aggregates (e.g. AFE = Eastern Africa). We can discard these. Most major coding systems are covered in this helper package, and as far as I know the package is actively maintained to keep up with the times. For instance, you can see that the code `SWZ` correctly produced `Eswatini` in the `country` column (name changed from Swaziland in 2018), and other name changes are sooner or later incorporated. That may or may not make sense to do, for instance if the population of a universe of a given code changes over time. We won't deal with that problem now.

It should look like so now:

```
gapreminder |> glimpse()
```

```
## Rows: 17,208
## Columns: 19
## $ country              <chr> "Burundi", "Burundi", "Burundi", "Burundi", "Burun~
## $ year                 <dbl> 1950, 1951, 1952, 1953, 1954, 1955, 1956, 1957, 19~
## $ iso3                 <chr> "BDI", "BDI", "BDI", "BDI", "BDI", "BDI", "BDI", "~
## $ tfr                  <dbl> 6.923, 6.914, 6.900, 6.915, 6.917, 6.930, 6.931, 6~
## $ e0f                  <dbl> 41.853, 42.223, 42.382, 42.645, 42.958, 43.264, 43~
## $ e0m                  <dbl> 39.246, 39.345, 39.568, 39.859, 40.100, 40.388, 40~
## $ popf                 <dbl> 1174.340, 1198.705, 1222.726, 1246.363, 1270.051, ~
```

```
## $ popm                <dbl> 1079.773, 1104.543, 1128.810, 1152.736, 1176.707, ~
## $ gpi                 <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ sign_contract       <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ remarry             <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ ind_work            <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ births_attend       <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ births_completeness <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ co2pc               <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ gdppc               <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, 71.36022, ~
## $ wage_ratio          <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ lfm                 <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ lff                 <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
```

Let's write the resulting `gapreminder` file out to a csv:

```
gapreminder |>
  write_csv("Data/gapreminder.csv")
```

Note, I've added the resulting file to the github repository, in case you need it, but I encourage you to create it yourself.

If you have created the dataset in a live session, then you already have it handy to work with. But if you have it saved, then you can just read it directly to `R` and skip the prior steps. That just looks like this:

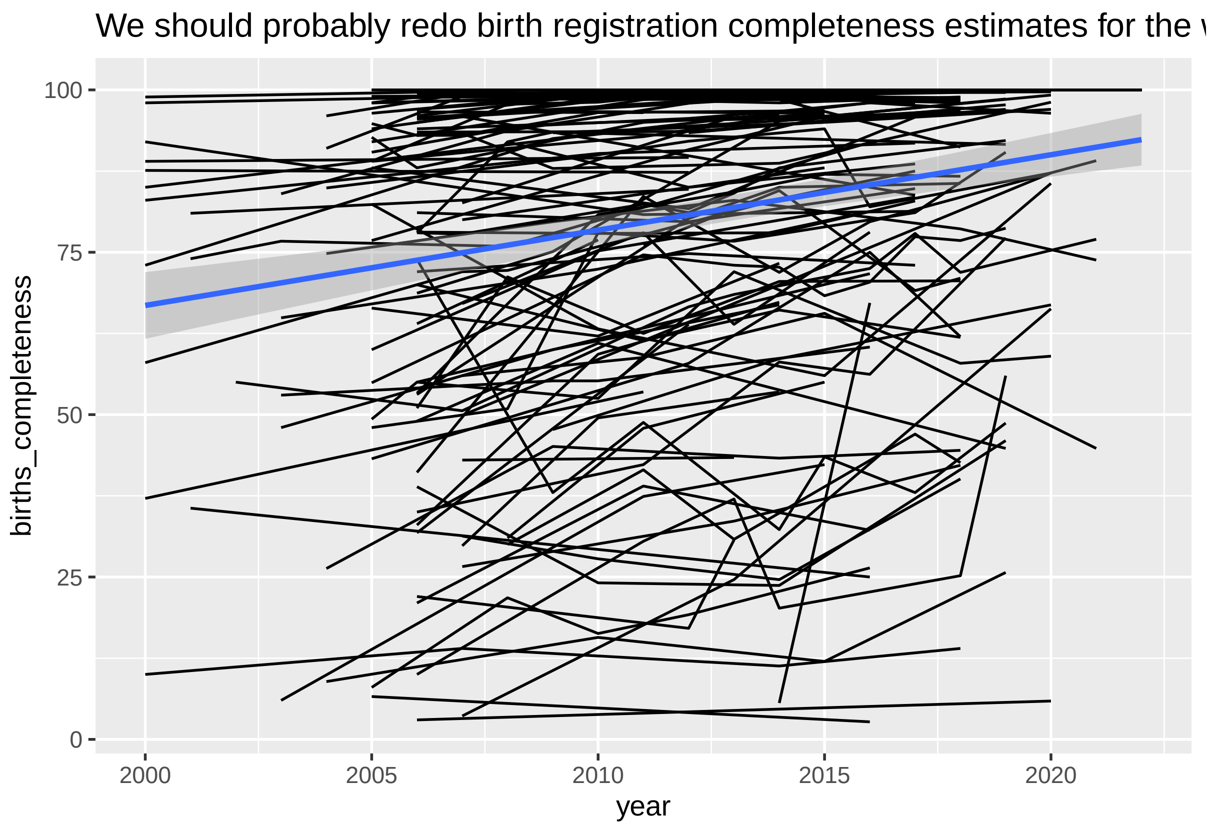
```
gapreminder <-
  read_csv("Data/gapreminder.csv")
```

2.9 Explre the data

Now the data are fairly tidy, depending on your ends. You can already start making some informative plots.

2.9.1 How has birth registration completeness changed over time?

```
gapreminder |>
  filter(!is.na(births_completeness)) |>
  ggplot(aes(x = year, y = births_completeness)) +
  geom_line(mapping = aes(group = country)) +
  geom_smooth(method = "lm") +
  labs(title = "We should probably redo birth registration completeness estimates for the wo
```



These estimates mostly come from indirect methods, and it looks like this dataset isn't a comprehensive time series, although it does cover 180 countries.

3 Exercises

3.0.1 1. How do C02 emissions per capita map to life expectancy or TFR?

(level 1 tricky)

```
gapreminder |>
  ggplot() # ...
```

For life expectancy, is the relationship similar to the original Preston curve? Which one is a stronger apparent relationship?

3.0.2 2. Best practice life expectancy

(level 2 tricky)

Make a scatterplot of life expectancy, with males and females in different panels, drawn as a background point cloud (light gray and semitransparent or similar). For each panel and each year, highlight the highest observed life expectancy using red points. Add a fitted line to the series of record high life expectancies by sex. Add another fitted line to the overall point cloud of life expectancy. Add a narrative title.

Here's how to get the highest life expectancy per year and sex, in order to get you started:

```
gapreminder |>
  filter(year < 2022) |>
  select(year, e0f, e0m) |>
  pivot_longer(-1, names_to = "sex", values_to = "e0") |>
  mutate(sex = substr(sex,3,3)) |>
  group_by(year, sex) |>
  filter(e0 == max(e0, na.rm = TRUE)) |>
  arrange(sex, year)
```

```
## # A tibble: 144 x 3
## # Groups:   year, sex [144]
##     year sex      e0
##    <dbl> <chr> <dbl>
##  1  1950 f      73.6
##  2  1951 f      74.3
##  3  1952 f      74.7
##  4  1953 f      75.1
##  5  1954 f      75.5
##  6  1955 f      75.9
##  7  1956 f      75.5
##  8  1957 f      75.5
##  9  1958 f      75.6
## 10  1959 f      75.8
## # i 134 more rows
```

In order to make the point cloud, you'll want to do something similar to the above, but without the `group_by()` and `filter()` steps. I suggest creating the two clean datasets (sssigning them to objects) and then composing the plot.

3.0.3 3. Does the Preston curve look different for men and women?

(level 3 tricky)

Here we have a conundrum: we have life expectancy by sex, but we don't know how to split gdp per capita. We do know the size of the labor force by gender though; and we also know relative wages (conditional on being employed). So maybe if you (very wrongly) assume that the entire GDP is made up of wages (it definitely is not!), then we could have a guess at it. Doing a good job of this (i.e. also splitting capital returns on gender somehow?) would be a difficult task that I think lots of people would like to see. You might try by using NTA data: https://www.ntaccounts.org/web/nta/show/. Let's do a bad job of it for now using brave assumptions and the data we have.

We know the labor force size, and the relative wages conditional on work, so also assuming that wages are have been standardized to the same working hours (also bad assumption), we can calculate the relative share of total wages that belongs to men and women, and then split GDP proportional to that.

Specifically, we have W_f the female to male wage ratio, which we can convert to proportions p_f and p_m like so:

$$p_f = \frac{W_f}{1 + W_f}$$

Where p_m is just the complement of p_f. The interpretation is that for equivalent working time, men get p_m proportion of wages and women get p_f. Then if each worker works the same amount, we can multiply the labor force size LF_f and LF_m to get something proportional to total wages earned for men and women TW_m and TW_f.

$$TW_f = p_f \cdot LF_f$$

Then split total GDP proportional to TW_f and TW_m, convert back to per capita GDP by gender.