Pivoting datasets

Intro

Pivoting: it’s a verb from our every day life used to describe the movement of an object or a person. Well datasets can pivot too ! Today we are going to see why pivoting them is tool for efficient and high-level.

Learning Objectives

- You will understand what a wide data format is, and what a long data format is
- You will know how to pivot long data to wide data using `pivot_long()`
- You will know how to pivot wide data to long data using `pivot_wider()`
- You have the intuition why the long data format is easier for plotting and wrangling

Packages

Datasets

We will see many different ones. So we will introduce them as we go along. But here is a brief overview:

- Worldwide, number of infant deaths (under 1 years old), per country, per year
- Worldwide, number of births, per country, per year
- European-scale, number of births, per country, per year
- Worldwide, number of malaria cases, per country, per year
- Worldwide, number of HIV cases, per country, per year

So many datasets! Now let's learn about their formats:

What do "wide" and "long" mean?

"Wide" data means that each data entry is a unique observation unit. This observation is the subject of further measurements or information collection. A unique observation unit can be a person, a country, a laboratory sample etc.

"Long" data means that each data entry is a data collection event. If the same observation unit had data collected in three ways, then it will have three data entries (3 rows).

Based on this explanation, "wide" and "long" formats are the **same dataset** if **data collection only occurred once for each unique observation unit.**

Some examples

Let's see an example of "wide" vs "long" data format for the same dataset. Imagine we are handling a timeseries of patients whose blood pressure you have recorded everyday.

You can record the data in a "wide" format like this:

patient	blood_pressure_day_1	blood_pressure_day_2	blood_pressure_day_1
A	110	112	114
B	120	122	124
C	100	104	105

Fig: "wide" dataset for a timeseries of patients.

Each unique observation unit (each patient) has only one row. The events (different measuring days) occupy different columns. You could say that the **focus** is on the unique observations(the patients).

Or you could record the data in a "long" format as so :

patient	day	blood pressure
A	1	110
B	2	112
C	3	114
A	1	120
B	2	122
C	3	124
A	1	100
B	2	104
C	3	105

Fig: "long" dataset for a timeseries of patients.

Here the unique days of recording (3 different events) define the data entries (the rows) of the data set. There are multiple rows per patient (the units of observation). Here the **focus** is on the time points / the events, the days.

Of course, the different collection ways do not have to be events / time points. They can be localizations (different neighborhoods, different cities) or any other variables which can stratify (divide into subgroups) the main units of observations.

Lab sample	City	Country-side
Sample 1	50	47
Sample 2	33	190
Sample 3	89	4

Fig: "wide" dataset where the unique observation unit is a lab sample and the different collection ways are different sampling sites.

Lab sample	Sampling site	Bacterial count
Sample 1	City	50
Sample 1	Country-side	47
Sample 2	City	33
Sample 2	Country-side	190
Sample 3	City	89
Sample 3	Country-side	4

Fig: "long" dataset where the unique observation unit is a lab sample and the different collection ways are different sampling sites.

Likewise, the unit of observation does not have to a patient. It can be a country (as presented here), or any other entity.

country	year	metric
x	1960	10
x	1970	13
x	2010	15
y	1960	20
y	1970	23
y	2010	25
z	1960	30
z	1970	33
z	2010	35

country	yr1960	yr1970	yr2010
x	10	13	15
y	20	23	25
z	30	33	35

Let's load the number of Nipah cases in different regions of malaysia.

```
nipah <- outbreaks::nipah_malaysia %>% as_tibble()
nipah
```

```
## # A tibble: 5 × 5
##   date       perak negeri_sembilan selangor singapore
##   <date>     <int>           <int>    <int>     <int>
## 1 1997-01-04     0               0        0         0
## 2 1997-01-11     1               0        0         0
## 3 1997-01-18     0               0        0         0
## 4 1997-01-25     1               0        0         0
## 5 1997-02-01     1               0        0         0
```

Is this a wide or long data format?

```
# Q_nipah_type <- "_____"
#
```

What is your unique observational unit ?

```
# Q_nipah_observations <- "_____"
#
```

When should you use wide vs long data?

The truth is: it really depends on what you want to do ! The wide format is great for *displaying data* because it's easy to visually compare values of unique observations unit this way. Long data is best for a majority of data analysis tasks such as grouping, plotting.

It will be essential for you to know how to switch from one to the other easily.

Switching from the "wide" to the "long" format (or the other way around) is called **pivoting**.

Pivoting wide to long

Many datasets you'll find in online data hubs and services will come in a wide format. The reason for this is that the wide format is a common way of inputing/entering data into

tables during data collection.

Let's have a look at Gapminder's open source data about the **Number of infant deaths (under 1 years old)**. Check out Gapminder: an awesome resource !

And here is how we went to find this dataset:

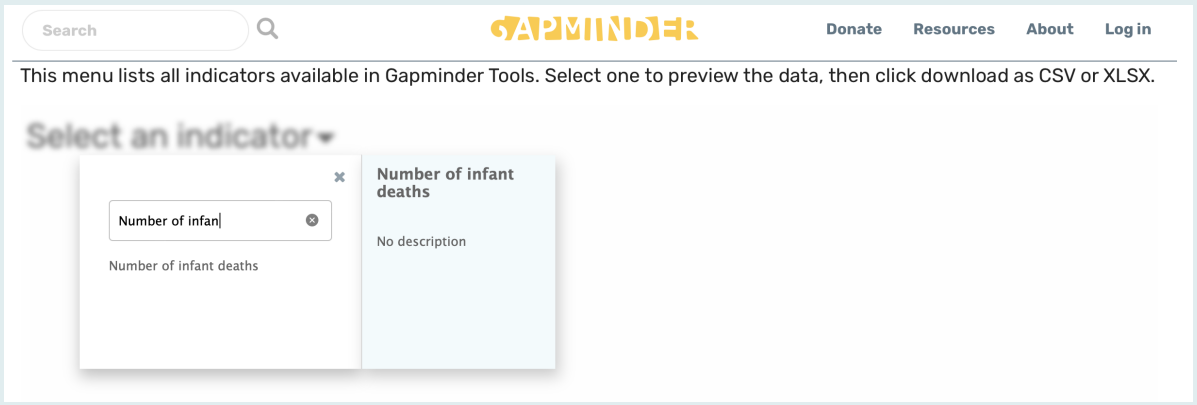


Fig: Navigating open-source datasets on Gapminder. Here we searched for "Number of infant deaths."

Let's read in the CSV we downloaded from Gapminder. We are going to select the years between 2010 and 2015 to keep it simple.

```
infant_deaths_wide <-
  read_csv(here("data/gapminder_infant_deaths.csv")) %>%
  select(country, x2010:x2015)

infant_deaths_wide
```

```
## # A tibble: 5 × 7
##   country              x2010 x2011 x2012 x2013 x2014 x2015
##   <chr>                <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 Afghanistan          74600 72000 69500 67100 64800 62700
## 2 Angola               79100 76400 73700 71200 69000 67200
## 3 Albania                420   384   354   331   313   301
## 4 United Arab Emirates   683   687   686   681   672   658
## 5 Argentina             9550  9230  8860  8480  8100  7720
```

Following our example above, we observe that each row corresponds to a unique observation unit: a country. The different events of data collection per country are different years: they are spread across different columns. Hence this dataset is in a "wide" format.

The numbers in each box of country x year dataset are the counts of infant deaths for that country and for that year. As an example, in 2010, there were 74600 infant deaths recorded for Afghanistan.

If we wanted to convert the data format to "long" format then we can use a convenient function `pivot_longer` :

```
infant_deaths_wide %>%
  pivot_longer(cols = x2010:x2015)
```

```
## # A tibble: 5 × 3
##   country     name  value
##   <chr>       <chr> <dbl>
## 1 Afghanistan x2010 74600
## 2 Afghanistan x2011 72000
## 3 Afghanistan x2012 69500
## 4 Afghanistan x2013 67100
## 5 Afghanistan x2014 64800
```

Within `pivot_longer` we define, using the `cols` argument, which columns we want to pivot.

We observe that the "long" format dataset now has each country occupying 5 rows (one per year between 2010 and 2015). The years have been pivoted into a single variable (`names`) instead of each representing their own variable (`x2010:x2015`). All the numbers in the country x years dataset were pivoted into their own variable (`values`). The infant deaths used to be in matrix format (2D: 2 dimensions), now they are in vector format (1D: 1 dimension).

This long dataset will be much more handy for most data analysis functions. Intuitively, manipulating 1D is always easier than manipulating 2D. More on this later !

However, as good data analysts, you might be cringing at the new names of your variables: `names` and `values` are generic and do not represent our data at all ! No worries, you can give custom column names:

```
infant_deaths_wide %>%
  pivot_longer(cols = x2010:x2015,
               names_to = "year",
               values_to = "deaths_count")
```

```
## # A tibble: 5 × 3
##   country     year  deaths_count
##   <chr>       <chr>        <dbl>
## 1 Afghanistan x2010        74600
## 2 Afghanistan x2011        72000
## 3 Afghanistan x2012        69500
```

```
## 4 Afghanistan x2013        67100
## 5 Afghanistan x2014        64800
```

The `cols` argument defining the variables chosen for pivot are the same. However `names_to` allows to define the variable name of the 1-D vector now regrouping your multiple pivoted columns. As you are pivoting years, it makes sense to call their variable `year`. We apply the same reasoning for the number of infant deaths pivoted from a matrix to a sole column: we name the variable embodying this column `deaths_counts`.

One could argument that the "long" format is more informative than the "wide" format. Why? Because of these column names. In the "wide" format, unless the CSV is named `count_infant_deaths` (or something alike) or unless someone tells you "these are the counts of infant deaths per country and per year", you have no idea what those numbers represent.

You may be a bit frustrated by the weird `x` in front of your years. Here is how you would remove it using `parse_number()` function:

```
infant_deaths_wide %>%
  pivot_longer(cols = x2010:x2015,
               names_to = "year",
               values_to = "deaths_count") %>%
  mutate(year = parse_number(year))
```

```
## # A tibble: 5 × 3
##   country      year deaths_count
##   <chr>       <dbl>        <dbl>
## 1 Afghanistan  2010        74600
## 2 Afghanistan  2011        72000
## 3 Afghanistan  2012        69500
## 4 Afghanistan  2013        67100
## 5 Afghanistan  2014        64800
```

We will see more on this function in the lessons dedicated to strings. But as a brief explanation: it extracts numbers from strings.

For later, let's store this "long" format dataset of the count of infant deaths per country and per year:

```
infant_deaths_long <-
  infant_deaths_wide %>%
  pivot_longer(cols = x2010:x2015,
               names_to = "year",
               values_to = "deaths_count")
```

For this practice question, we will use the `euro_births_wide` dataset from Eurostat. It contains data on births in 50 European countries: the crude birth rate per thousand people, collected on an annual basis.

```
euro_births_wide <-
  read_csv(here("data/euro_births_wide.csv"))
euro_births_wide
```

```
## # A tibble: 5 × 8
##   country   x2015  x2016  x2017  x2018  x2019  x2020  x2021
##   <chr>     <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
## 1 Belgium  122274 121896 119690 118319 117695 114350 118349
## 2 Bulgaria  65950  64984  63955  62197  61538  59086  58678
## 3 Czechia  110764 112663 114405 114036 112231 110200 111793
## 4 Denmark   58205  61614  61397  61476  61167  60937  63473
## 5 Germany  737575 792141 784901 787523 778090 773144 795492
```

What is the unique observational unit? Your answer should be a string giving the name of the variable.

```
# Q_euro_births_unique_observational_unit <- "_____"
#
```

What variable defines the data collection in multiple ways (i.e. the potential stratification variable) ? Your answer should be a string giving the name of the variable.

```
# Q_euro_births_multiple_ways <- "_____"
#
```

The data is in a "wide" format. Convert it to a "long" format.

```
# Q_euro_births_long_format <-
#   euro_births_wide %>%
#   pivot_longer(___________)
```

Pivoting long to wide

Wide data often comes from external sources as we have seen above. Long data, on the other hand, is likely to be created by *you* in the course of your data wrangling. Let's see an example:

We will use yet another dataset: patient records from an Ebola outbreak in Sierra Leone in 2014.

```
ebola <- outbreaks::ebola_sierraleone_2014 %>% as_tibble()
ebola
```

```
## # A tibble: 5 × 8
##       id   age sex   status    date_of_onset date_of_sample district
##    <int> <dbl> <fct> <fct>     <date>        <date>         <fct>
## 1      1    20 F     confirmed 2014-05-18    2014-05-23     Kailahun
## 2      2    42 F     confirmed 2014-05-20    2014-05-25     Kailahun
## 3      3    45 F     confirmed 2014-05-20    2014-05-25     Kailahun
## 4      4    15 F     confirmed 2014-05-21    2014-05-26     Kailahun
## 5      5    19 F     confirmed 2014-05-21    2014-05-26     Kailahun
## # … with 1 more variable: chiefdom <fct>
```

Here each row corresponds to a unique unit of observation: one ebola patients. The data is in a "wide" format.

You may want to count how many cases there were in each year across districts. For this, you would apply the `summary()` functions on your datasets:

```
ebola_summary <-
  ebola %>%
  summary()

ebola_summary
```

```
##        id              age          sex             status
##  Min.   :    1   Min.   : 0.0   F   :4719   confirmed:8358
##  1st Qu.: 2976   1st Qu.:16.0   M   :5109   suspected:3545
##  Median : 5952   Median :28.0   NA's:2075
##  Mean   : 5952   Mean   :30.3
##  3rd Qu.: 8928   3rd Qu.:42.0
##  date_of_onset        date_of_sample                district
##  Min.   :2014-05-18   Min.   :2014-05-23   Western Urban:3165
##  1st Qu.:2014-10-05   1st Qu.:2014-10-12   Port Loko    :1701
##  Median :2014-11-18   Median :2014-11-24   Western Rural:1522
##  Mean   :2014-12-04   Mean   :2014-12-10   Bombali      :1190
##  3rd Qu.:2015-01-11   3rd Qu.:2015-01-16   Kenema       : 780
##            chiefdom
##  W/Urban       :2274
```

```
##  W/Rural        :1146
##  Freetowm       : 891
##  Bombali Sebora: 665
##  Nongowa        : 527
```

A summary is considered a long format because it no longer has the structure one row = one observation unit.

Let's see an example where you keep a wide format, but you can the unique observation unit. This occurs when we count the number of patients in each district.

```
ebola_district_counts <-
  ebola %>%
  count(district)

ebola_district_counts
```

```
## # A tibble: 5 × 2
##   district     n
##   <fct>    <int>
## 1 Bo         606
## 2 Bombali   1190
## 3 Bonthe      84
## 4 Kailahun   570
## 5 Kambia     421
```

Now, one row = one observation, so we are still in a wide format, but the unique observation units are now the districts and not the patients themselves.

Let's now explore another, somewhat similar, data wrangling manipulation which will change your data format from "wide" to "long": grouping by multiple variables then counting number of recorded patients within groups.

```
ebola_yearly_counts <-
  ebola %>%
  mutate(year = lubridate::year(date_of_onset)) %>%
  group_by(district, year) %>%
  count()

ebola_yearly_counts
```

```
## # A tibble: 5 × 3
## # Groups:   district, year [5]
##   district  year     n
##   <fct>    <dbl> <int>
## 1 Bo        2014   397
## 2 Bo        2015   209
## 3 Bombali   2014  1070
## 4 Bombali   2015   120
## 5 Bonthe    2014     7
```

Here we start with extracting the year information (year of the date) from the variable `date_of_onset`. For this we use a useful function `year()` from the package {lubridate}, which you will see in more details in further lessons. We then group by `district` and `year` to count the number of patients recorded per district and per year.

Because of this nested grouping, we are generating a long format dataset: we do not have one row = one unique observation. Both districts and years take up multiple rows. Each district takes up 2 rows (because their count is calculated for 2 years: 14 x 2 = 28) and each year takes up 14 rows (because their count is calculated over 14 districts: 2 x 14 = 28).

This is a perfect illustration of how one data wrangling manipulation will preserve a wide format, while another will generate a long format.

What if we wanted to switch `ebola_yearly_counts` back to a wide format ? There is an easy function for this: `pivot_wider()` (similar to `pivot_longer()`).

Let's switch our data back to a wide format:

```
ebola_yearly_counts %>%
  pivot_wider(values_from = n,
              names_from = year)
```

```
## # A tibble: 5 × 3
## # Groups:   district [5]
##   district `2014` `2015`
##   <fct>     <int>  <int>
## 1 Bo          397    209
## 2 Bombali    1070    120
## 3 Bonthe        7     77
## 4 Kailahun    535     35
## 5 Kambia      127    294
```

`pivot_wider()` has two important arguments: `values_from` and `names_from`. `values_from` defines which values will become the core of the wide data format (in other words: which 1D vector will become a 2D matrix). `names_from` identifies which variable to use to define column names in the wide format.

To explain with our dataset `ebola_yearly_counts`. We define our observation unit as the districts. So to have a wide format, we need the variable `years` to become two columns: `2014` and `2015`, the unique values of the variable. We define this in `pivot_wider()` with `names_from`. This would then create a dataframe in the format districts x years, in other words, a matrix. You need to fill the matrix with values, hence the argument `values_from`, which indicates we want to fill the matrix with count values.

Now, imagine we wanted to see the years as our unique observation unit. We would do exactly the same, but our 2nd dimension to the wide format would be the `districts`:

```
ebola_yearly_counts %>%
  pivot_wider(values_from = n,
              names_from = district)
```

```
## # A tibble: 2 × 15
## # Groups:   year [2]
##    year    Bo Bombali Bonthe Kailahun Kambia Kenema Koinadugu  Kono
##   <dbl> <int>   <int>  <int>    <int>  <int>  <int>     <int> <int>
## 1  2014   397    1070      7      535    127    641       142   328
## 2  2015   209     120     77       35    294    139        15   223
## # … with 6 more variables: Moyamba <int>, `Port Loko` <int>,
## #   Pujehun <int>, Tonkolili <int>, `Western Rural` <int>, …
```

Here our unique observation units (our rows) are now the years (2014, 2015). Our 2nd dimension (our columns) are the districts. Our matrix input are the count values organized in the format years x districts.

That's it! We'll see more complex examples in the next lesson...

Above, using Gapminder's data on the number of infant deaths, we transformed the wide format dataset `infant_deaths_wide` into the long format dataset `infant_deaths_long`. Now, using `pivot_wider()`, let's switch it back to its wide format again. We want the unique observation unit to be the countries and to have a countries x years matrix, filled with the death counts.


```
# Q_infant_death_wide_countries <-
#   infant_death_long %>%
#   pivot_wider(___________)
```

Using again `pivot_wider()` and again `infant_death_long`, let's switch it back to its wide format but this time we want the unique observation unit to be the years and to have a years x countries matrix, filled with the death counts.

```
# Q_infant_death_wide_years <-
#   infant_death_long %>%
#   pivot_wider(___________)
```

Why is long data better for analysis?

Above we mentioned that long data is best for a majority of data analysis tasks. Now we can justify why.

Filtering grouped data

First, let's talk about filtering grouped data. Some filter operations are difficult to do on wide data.

Here is an example taking the infant deaths dataset. We want to answer the following question: **For each country, which year had the highest number of child deaths?**

This is how we would do so with the long format of the data :

```
infant_deaths_long %>%
  group_by(country) %>%
  filter(deaths_count == max(deaths_count))
```

```
## # A tibble: 5 × 3
## # Groups:   country [5]
##   country              year  deaths_count
##   <chr>                <chr>        <dbl>
## 1 Afghanistan          x2010        74600
## 2 Angola               x2010        79100
## 3 Albania              x2010          420
## 4 United Arab Emirates x2011          687
## 5 Argentina            x2010         9550
```

Easy right? We can easily see that Afghanistan had its maximal infant death count in 2010.

If you wanted to do the same thing with wide data, you would need some weird functions like `rowwise()`:

```
infant_deaths_wide %>%
  rowwise() %>%
  mutate(max_count = max(x2010, x2011, x2012, x2013, x2014, x2015))
```

```
## # A tibble: 5 × 8
## # Rowwise:
##   country              x2010 x2011 x2012 x2013 x2014 x2015 max_count
##   <chr>                <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>     <dbl>
## 1 Afghanistan          74600 72000 69500 67100 64800 62700     74600
## 2 Angola               79100 76400 73700 71200 69000 67200     79100
## 3 Albania                420   384   354   331   313   301       420
```

```
## 4 United Arab Emirates   683   687   686   681   672   658        687
## 5 Argentina             9550  9230  8860  8480  8100  7720       9550
```

Okay it works, but we still don't know which year is attached to that value in `max_count`. We would have to take that value and index it back to its respective year column... what a hassle! There are solutions to find this but all are very painful. Why make your life complicated when you can just pivot to long format and use the beauty of `group_by()` and `filter()`?

Here we use a special {dplyr} function: `rowwise()`. `rowwise()` is a function which allows further operations to be applied on the rows rather than on the columns. As you can see, here `mutate()` creating a maximum value column, takes the maximum of each row (not the maximum of each column.

SIDE NOTE

Without `rowwise()` you would get this :

```
infant_deaths_wide %>%
  mutate(max_count = max(x2010, x2011, x2012, x2013, x2014,
         x2015))
```

```
## # A tibble: 5 × 8
##   country              x2010 x2011 x2012 x2013 x2014 x2015
max_count
##   <chr>                <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
<dbl>
## 1 Afghanistan          74600 72000 69500 67100 64800 62700
1170000
## 2 Angola               79100 76400 73700 71200 69000 67200
1170000
## 3 Albania                420   384   354   331   313   301
1170000
## 4 United Arab Emirates   683   687   686   681   672   658
1170000
## 5 Argentina             9550  9230  8860  8480  8100  7720
1170000
```

The max count over all columns.

PRACTICE

(in RMD)

Using the Eurostat dataset, `euro_births_wide`, wrangle to have the maximal birth count over all years, per country. (like above, you should think of pivoting, then using `group_by()` and `filter()`)


```
# Q_euro_births_max <- "_____"
#
```

Summarizing grouped data

Like filtering, most summarizing operations will be difficult to do on wide data. If we pause and think why it is so hard to filter or summarize on wide data, we realize that it is because the wide data prevents us from grouping our data. A data format where each row is a unique observation unit is a data format where each group is a unique observation unit. A long format allows to group data.

For example, if you want to ask: **For each country, between 2015 and 2021, what was the mean number of infant deaths and the standard deviation (variation) in deaths ?**

With long data it is simple:

```
infant_deaths_long %>%
  group_by(country) %>%
  summarize(mean_deaths = mean(deaths_count),
            sd_deaths = sd(deaths_count))
```

```
## # A tibble: 5 × 3
##   country             mean_deaths sd_deaths
##   <chr>                     <dbl>     <dbl>
## 1 Afghanistan               68450    4466.
## 2 Albania                     350.     45.2
## 3 Algeria                   21033.    484.
## 4 Angola                    72767.   4513.
## 5 Antigua and Barbuda          10.7     0.816
```

With wide data, on the other hand, finding the mean is doable...

```
infant_deaths_wide %>%
  rowwise() %>%
  mutate(mean_deaths = sum(x2010, x2011, x2012,
                           x2013, x2014, x2015, na.rm = T)/6)
```

```
## # A tibble: 5 × 8
## # Rowwise:
##   country              x2010 x2011 x2012 x2013 x2014 x2015 mean_deaths
##   <chr>                <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>       <dbl>
## 1 Afghanistan          74600 72000 69500 67100 64800 62700       68450
## 2 Angola               79100 76400 73700 71200 69000 67200       72767.
## 3 Albania                420   384   354   331   313   301         350.
## 4 United Arab Emirates   683   687   686   681   672   658         678.
## 5 Argentina             9550  9230  8860  8480  8100  7720        8657.
```

But standard deviation would be more difficult.

Using the Eurostat dataset, `euro_births_wide`, wrangle to have the mean number and variation (standard deviation) of birth over all years, per country. (like above, you should think of pivoting, then using `group_by()` and `summarize()`)

```
# Q_euro_births_mean_sd <- "_____"
#
```

Joining datasets

Joining related datasets is often easier with long data.

To illustrate this, let us load another dataset: the number of children born each year (also another jewel dataset from Gapminder!).

```
infant_births_wide <-
    read_csv(here("data/gapminder_new_births.csv"))

infant_births_wide
```

```
## # A tibble: 5 × 7
##   country                x2010   x2011   x2012   x2013   x2014   x2015
##   <chr>                  <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
## 1 Aruba                   1110    1090    1070    1060    1050    1040
## 2 Afghanistan          1100000 1090000 1090000 1080000 1080000 1080000
## 3 Angola               1010000 1030000 1060000 1080000 1100000 1130000
## 4 Albania                35000   35900   37000   38000   38900   39500
## 5 United Arab Emirates   91200   94300   96500   97900   98400   98300
```

The dataset looks very similar to the one about infant deaths, only this time, we are viewing the counts of births.

You may want to join this information on births with the number of deaths. With this joint dataset, we could calculate the infant mortality rate for each year

With long data these operations are easy. First pivot new data to long:

```
infant_births_long <-
  infant_births_wide %>%
  pivot_longer(cols = x2010:x2015,
               values_to = "births_count",
               names_to = "year")

infant_births_long
```

```
## # A tibble: 5 × 3
##   country year  births_count
##   <chr>   <chr>        <dbl>
## 1 Aruba   x2010         1110
## 2 Aruba   x2011         1090
## 3 Aruba   x2012         1070
## 4 Aruba   x2013         1060
## 5 Aruba   x2014         1050
```

Then join using `left_join()`:

```
infant_deaths_long %>%
  left_join(infant_births_long)
```

```
## # A tibble: 5 × 4
##   country     year  deaths_count births_count
##   <chr>       <chr>        <dbl>        <dbl>
## 1 Afghanistan x2010        74600      1100000
## 2 Afghanistan x2011        72000      1090000
## 3 Afghanistan x2012        69500      1090000
## 4 Afghanistan x2013        67100      1080000
## 5 Afghanistan x2014        64800      1080000
```

The mortality rate is the number of deaths on the number of births. You can easily calculate the mortality rate for each year using `mutate()`.

```
infant_deaths_long %>%
  left_join(infant_births_long) %>%
  mutate(infant_mortality_rate = deaths_count/births_count,
         infant_mortality_rate_per_cent = infant_mortality_rate * 100)
```

```
## # A tibble: 5 × 6
##   country     year  deaths_count births_count infant_mortality_rate
##   <chr>       <chr>        <dbl>        <dbl>                  <dbl>
## 1 Afghanistan x2010        74600      1100000                 0.0678
## 2 Afghanistan x2011        72000      1090000                 0.0661
## 3 Afghanistan x2012        69500      1090000                 0.0638
## 4 Afghanistan x2013        67100      1080000                 0.0621
## 5 Afghanistan x2014        64800      1080000                 0.06
## # … with 1 more variable: infant_mortality_rate_per_cent <dbl>
```

Continuing the use of Gapminder, we will load in 2 more datasets that you will then join together.

The first is a count of HIV cases per year and per country.

```
hiv_counts <-
    read_csv(here("data/gapminder_hiv.csv"))

hiv_counts
```

```
## # A tibble: 5 × 23
##   country     `1990` `1991` `1992` `1993` `1994` `1995`
`1996` `1997`
##   <chr>       <chr>  <chr>  <chr>  <chr>  <chr>  <chr>
<chr>  <chr>
## 1 Afghanistan 60     60     60     60     60     150    150
150
## 2 Angola      600    1200   1800   2500   3300   4300
5300   6300
## 3 Argentina   3000   3200   3400   3600   3800   3500
3200   3100
## 4 Armenia     60     60     60     60     60     60     60
60
## 5 Australia   350    600    600    600    600    600    600
350
## # … with 14 more variables: `1998` <chr>, `1999` <chr>,
`2000` <chr>,
## #   `2001` <chr>, `2002` <chr>, `2003` <chr>, `2004` <chr>,
…
```

The second is a count of malaria cases per year and per country.

```
malaria_counts <-
    read_csv(here("data/gapminder_malaria.csv"))

malaria_counts
```

```
## # A tibble: 5 × 18
##   country     `1990` `1991` `1992` `1993` `1994` `1995`
`1996` `1997`
##   <chr>       <chr>   <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
<dbl> <chr>
## 1 Afghanistan <NA>       NA     NA     NA     22     NA
NA <NA>
## 2 Angola      <NA>       NA     NA     NA     NA     NA
NA <NA>
## 3 Argentina   <NA>       NA     NA     NA     NA     NA
NA <NA>
## 4 Armenia     <NA>       NA     NA     NA     NA     NA
NA <NA>
## 5 Azerbaijan  <NA>       NA     NA     NA     NA     NA
NA <NA>
```

```
## # … with 9 more variables: `1998` <dbl>, `1999` <chr>,
`2000` <chr>,
## #   `2001` <dbl>, `2002` <chr>, `2003` <chr>, `2004` <chr>,
…
```

We are going to join these two datasets together after pivoting them into long format.

```
# Q_malaria_hiv_comparison <-
#   hiv_counts_long %>%
#   left_join(malaria_counts_long)
```

After joinging them, filter for the country of Vietnam and for the year 2000. In 2000, did Vietnam have more cases of malaria or HIV ? Your answer should be a string (either "malaria" or "HIV").

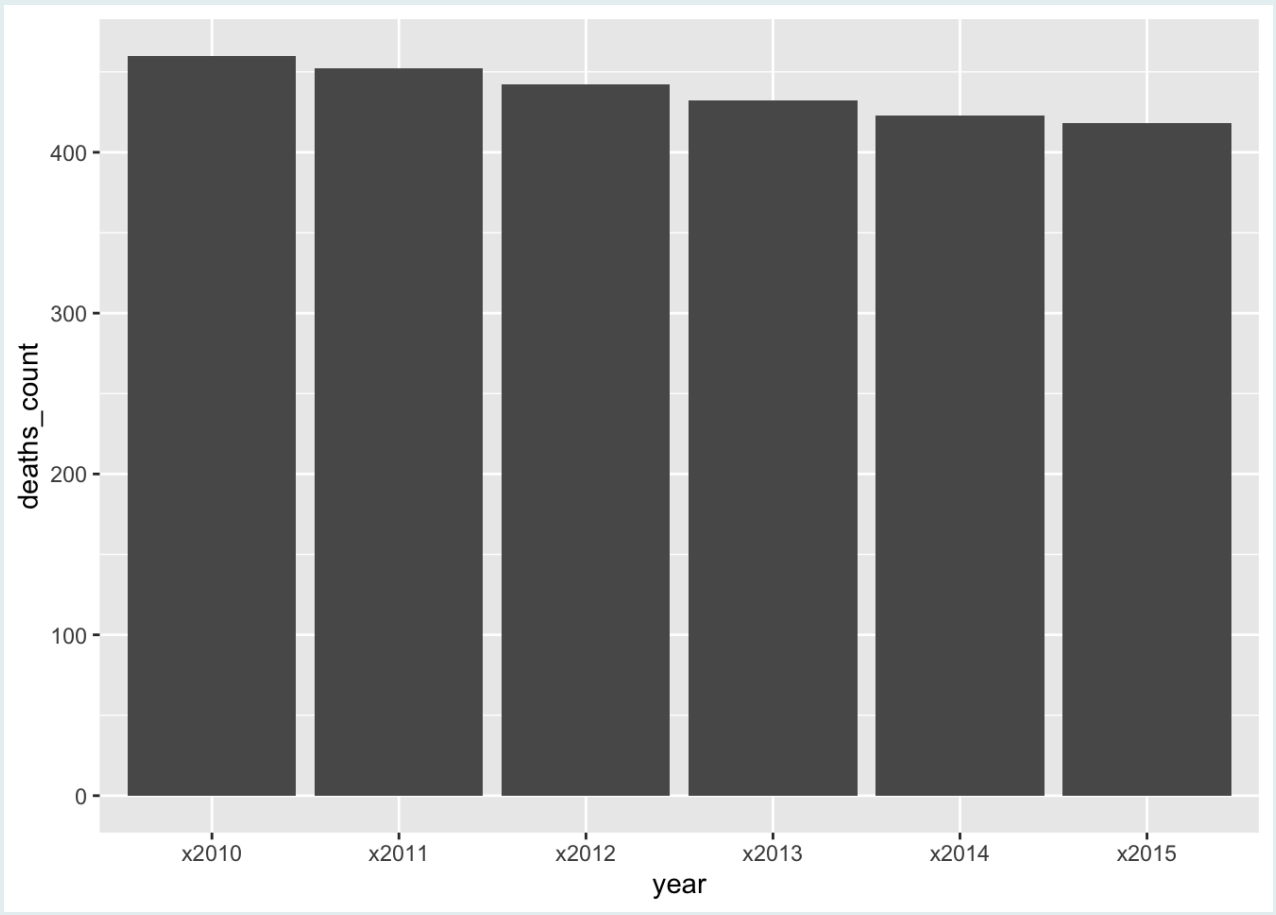
```
# Q_Vietnam_2000 <- "_______"
```

Plotting

Finally, one of the data analysis tasks that is MOST hindered by wide formats is plotting. You may not yet have any prior knowledge of {ggplot} and how to plot so we will see the figures without going in depth with the code. What you need to remember is: many plots with with ggplot are also only possible with long-format data

Consider again the infant_deaths data `infant_deaths_long`. We will plot the number of deaths for Belgium per year:

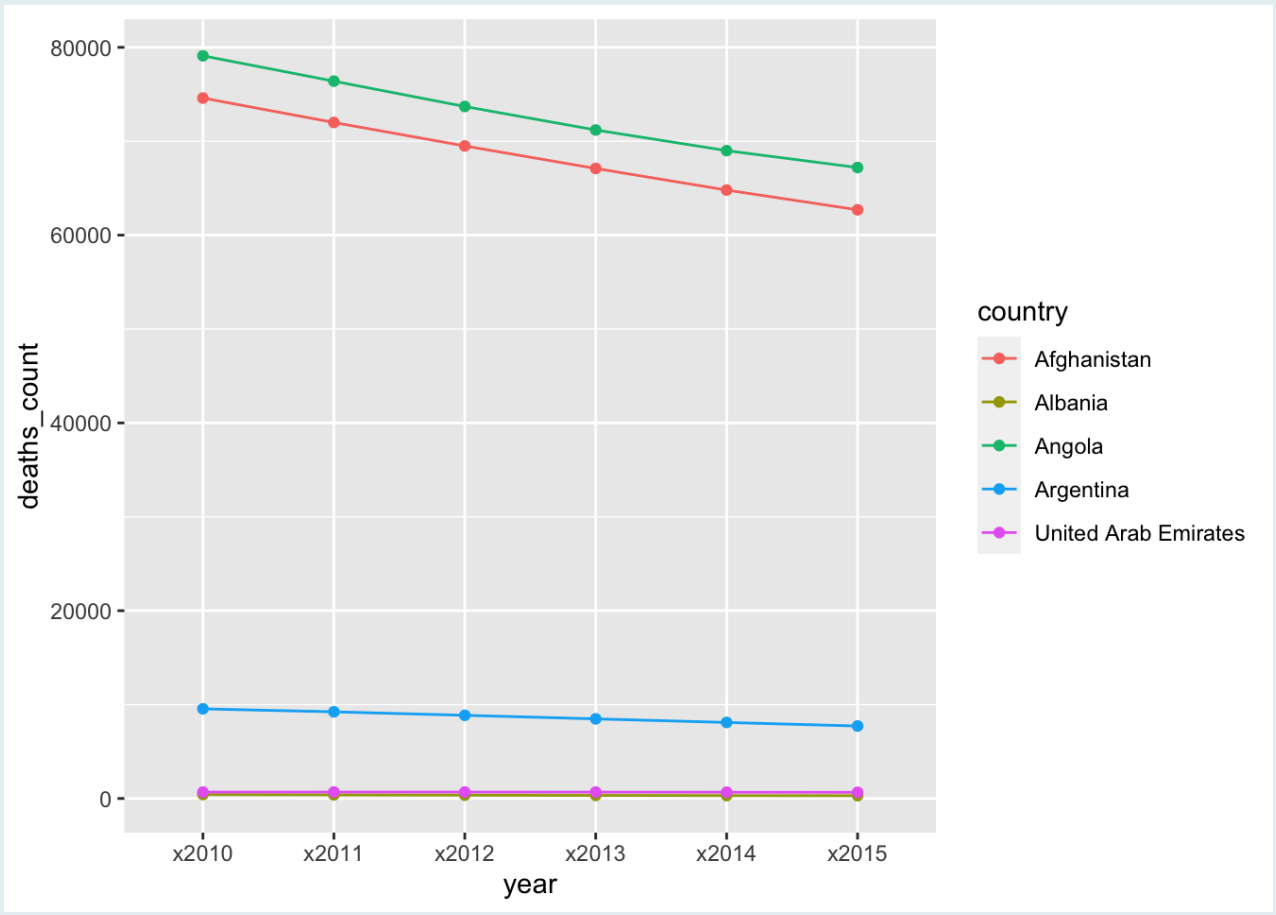
```
infant_deaths_long %>%
  filter(country == "Belgium") %>%
  ggplot() +
  geom_col(aes(x = year, y = deaths_count))
```



The plotting works because we can give the variable `year` for the x-axis. In the long format, `year` is a variable variable of its own. In the wide format, each year is a column and your counts are a matrix, not a vector. Plots need vectors of data!

Another plot that would not be possible without a long format:

```
infant_deaths_long %>%
  head(30) %>%
  ggplot(aes(x = year, y = deaths_count, group = country, color = country)) +
  geom_line() +
  geom_point()
```



Once again, the reason is the same, we need to tell the plot what to use as an x-axis and a y-axis and it is necessary to have these variables in their own column (as organized in the long format).

Wrap Up !

You have now explored different datasets and how they are either in a long or wide format. In the end, it's just about how you present the information. Sometimes one format will be more convenient, and other times another could be best. Now, you are no longer limited by the format of your data: don't like it? change it !

Contributors

The following team members contributed to this lesson:

KENE DAVID NWOSU

Data analyst, the GRAPH Network
Passionate about education

LAURE VANCAUWENBERGHE

Data analyst, the GRAPH Network
A firm believer in science for good, striving to ally programming, health and education

– title: "Pivoting" credits: "This document serves as an accompaniment for a lesson found on https://thegraphcourses.org.

The GRAPH Courses is a project of the Global Research and Analyses for Public Health (GRAPH) Network, with the support of the World Health Organization (WHO) and other partners" date: "November 2022" author: "The GRAPH Courses team" –