Lesson notes | Advanced pivoting

Created by the GRAPH Courses team

November 2023

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, a non-profit headquartered at the University of Geneva Global Health Institute, and supported by the World Health Organization (WHO) and other partners

Intro

You know basic pivoting operations from long format datasets to wide format datasets and vice versa. However, as is often the case, basic manipulations are sometimes not enough for the wrangling you need to do. Let's now see the next level. Let's go !

Learning Objectives

1. Master complex pivoting from wide to long and long to wide
2. Know how to use separators as a pivoting tool

Packages

```
# Load packages
if(!require(pacman)) install.packages("pacman")
pacman::p_load(tidyverse, outbreaks, janitor, rio, here, knitr)
```

Datasets

We will introduce these datasets as we go along but here is an overview:

- Survey data from India on how much money patients spent on tuberculosis treatment

- Biomarker data from an enteropathogen study in Zambia
- A diet survey from Vietnam

Wide to long

Sometimes you have multiple kinds of wide data in the same table. Consider this artificial example of heights and weights for children over two years:

```
child_stats <-
  tibble::tribble(
    ~child, ~year1_height, ~year2_height, ~year1_weight, ~year2_weight,
       "A",        "80cm",        "85cm",         "5kg",        "10kg",
       "B",        "85cm",        "90cm",         "7kg",        "12kg",
       "C",        "90cm",       "100cm",         "6kg",        "14kg"
    )

child_stats
```

```
## # A tibble: 3 × 5
##   child year1_height year2_height year1_weight year2_weight
##   <chr> <chr>        <chr>        <chr>        <chr>
## 1 A     80cm         85cm         5kg          10kg
## 2 B     85cm         90cm         7kg          12kg
## 3 C     90cm         100cm        6kg          14kg
```

If you pivot all the measurement columns, you'll get overly long data:

```
child_stats %>%
  pivot_longer(2:5)
```

```
## # A tibble: 5 × 3
##   child name         value
##   <chr> <chr>        <chr>
## 1 A     year1_height 80cm
## 2 A     year2_height 85cm
## 3 A     year1_weight 5kg
## 4 A     year2_weight 10kg
## 5 B     year1_height 85cm
```

This is not what you (usually) want, because now you have two different kinds of data in the same column—weight and height.

To get the right shape, you'll need to use the `names_sep` argument and the ".value" identifier:

```
child_stats %>%
  pivot_longer(2:5,
               names_sep = "_",
               names_to = c("period", ".value"))
```

```
## # A tibble: 5 × 4
##   child period height weight
##   <chr> <chr>  <chr>  <chr>
## 1 A     year1  80cm   5kg
## 2 A     year2  85cm   10kg
## 3 B     year1  85cm   7kg
## 4 B     year2  90cm   12kg
## 5 C     year1  90cm   6kg
```

Now we have one row for each child-period, an appropriately long format!

What the code above is doing may not be clear, but you should already be able to answer the practice question below by pattern matching with our example. After the practice question, we will explain the `names_sep` argument and the ".value" identifier in more depth.

Consider this other artificial data set:

```
adult_stats <-
  tibble::tribble(
    ~adult,  ~year1_BMI,  ~year2_BMI,  ~year1_HIV,  ~year2_HIV,
       "A",          25,          30,  "Positive",  "Positive",
       "B",          34,          28,  "Negative",  "Positive",
       "C",          19,          17,  "Negative",  "Negative"
  )


adult_stats
```

```
## # A tibble: 3 × 5
##   adult year1_BMI year2_BMI year1_HIV year2_HIV
##   <chr>     <dbl>     <dbl> <chr>     <chr>
## 1 A            25        30 Positive  Positive
## 2 B            34        28 Negative  Positive
## 3 C            19        17 Negative  Negative
```

Pivot the data into a long format to get the following structure:

adult	year	BMI	HIV

PRACTICE

(in RMD)

```
Q_adult_long <-
  adult_stats %>%
  pivot_longer(_________)
```

The `child_stats` example above has numbers stored as characters [...]

As you saw in the previous lesson, you can easily extract the numbers from the output long data frame in our example using the `parse_number()` function from readr:

```
child_stats_long <-
  child_stats %>%
  pivot_longer(2:5,
               names_sep = "_",
               names_to = c("period", ".value"))

child_stats_long
```

SIDE NOTE

```
## # A tibble: 5 × 4
##   child period height weight
##   <chr> <chr>  <chr>  <chr>
## 1 A     year1  80cm   5kg
## 2 A     year2  85cm   10kg
## 3 B     year1  85cm   7kg
## 4 B     year2  90cm   12kg
## 5 C     year1  90cm   6kg
```

```
child_stats_long %>%
  mutate(height = parse_number(height),
         weight = parse_number(weight))
```

```
## # A tibble: 5 × 4
##   child period height weight
##   <chr> <chr>   <dbl>  <dbl>
## 1 A     year1      80      5
## 2 A     year2      85     10
## 3 B     year1      85      7
## 4 B     year2      90     12
## 5 C     year1      90      6
```

Understanding `names_sep` and ".value"

Now let's break down the `pivot_longer()` call we saw above a bit more:

```
child_stats
```

```
## # A tibble: 3 × 5
##   child year1_height year2_height year1_weight year2_weight
##   <chr> <chr>        <chr>        <chr>        <chr>
## 1 A     80cm         85cm         5kg          10kg
## 2 B     85cm         90cm         7kg          12kg
## 3 C     90cm         100cm        6kg          14kg
```

```
child_stats %>%
  pivot_longer(2:5,
               names_sep = "_",
               names_to = c("period", ".value"))
```

```
## # A tibble: 5 × 4
##   child period height weight
##   <chr> <chr>  <chr>  <chr>
## 1 A     year1  80cm   5kg
## 2 A     year2  85cm   10kg
## 3 B     year1  85cm   7kg
## 4 B     year2  90cm   12kg
## 5 C     year1  90cm   6kg
```

Notice that the column names in the original `child_stats` data frame (`year1_height`, `year2_height` and so on) are made of three parts:

- the period being referenced: e.g. "year1"
- an underscore separator, "_";
- and the type of value recorded "height" or "weight"

We can make a table with these parts:

column_name	period	separator	".value"
year1_height	year1	_	height
year2_height	year2	_	height
year1_weight	year1	_	weight
year2_weight	year2	_	weight

`names_sep = "_"`:

This is the separator between the period indicator (year) and the values (year and weight) recorded.

If we have a different separator, this argument would change. For example, if the separator were an empty space, " ", you would have `names_sep = " "`, as seen in the example below:

```
child_stats_space_sep <-
  tibble::tribble(
    ~child, ~`yr1 height`, ~`yr2 height`, ~`yr1 weight`, ~`yr2 weight`,
       "A",        "80cm",        "85cm",         "5kg",        "10kg",
       "B",        "85cm",        "90cm",         "7kg",        "12kg",
       "C",        "90cm",       "100cm",         "6kg",        "14kg"
    )

child_stats_space_sep %>%
  pivot_longer(2:5,
               names_sep = " ",
               names_to = c("period", ".value"))
```

```
## # A tibble: 5 × 4
##   child period height weight
##   <chr> <chr>  <chr>  <chr>
## 1 A     yr1    80cm   5kg
## 2 A     yr2    85cm   10kg
## 3 B     yr1    85cm   7kg
## 4 B     yr2    90cm   12kg
## 5 C     yr1    90cm   6kg
```

`names_to = c("period", ".value")`

Next, the names_to argument indicates how the data should be reshaped. We passed a vector of two character strings , "period" and the ".value" to this argument. Let's consider each in turn:

The "period" string indicated that we want to move the data from each year (or period) into a separate row Note that there is nothing special about the word "period" used here; we could change this to any other string. So instead of "period", you could have written "time" or "year_of_measurement" or anything else:

```
child_stats %>%
  pivot_longer(2:5,
               names_sep = "_",
               names_to = c("year_of_measurement", ".value"))
```

```
## # A tibble: 5 × 4
##   child year_of_measurement height weight
##   <chr> <chr>               <chr>  <chr>
## 1 A     year1               80cm   5kg
## 2 A     year2               85cm   10kg
## 3 B     year1               85cm   7kg
## 4 B     year2               90cm   12kg
## 5 C     year1               90cm   6kg
```

Now, **the ".value" placeholder** is a special indicator, that tells `pivot_longer()` to make a separate column for every distinct value that appears after the separator. In our example, these distinct values are "height" and "weight".

The ".value" string cannot be arbitrarily replaced. For example, this won't work:

```
child_stats %>%
  pivot_longer(2:5,
               names_sep = "_",
               names_to = c("period", "values"))
```

```
## # A tibble: 5 × 4
##   child period values value
##   <chr> <chr>  <chr>  <chr>
## 1 A     year1  height 80cm
## 2 A     year2  height 85cm
## 3 A     year1  weight 5kg
## 4 A     year2  weight 10kg
## 5 B     year1  height 85cm
```

To restate the point, the ".value" placeholder is tells `pivot_longer()` that we want to separate out the "height" and "weight" values into separate columns, because there are the two value types that occur after the "_" separator in the column names.

This means that if you had a wide dataset with three types of values, you would get separated-out columns, one for each value type. For example, consider the mock dataset below which shows children's records, at two time points, for the following variables:

- age in months,
- body fat %
- bmi

```
child_stats_three_values <-
  tibble::tribble(
  ~child,  ~t1_age,  ~t2_age, ~t1_fat, ~t2_fat, ~t1_bmi, ~t2_bmi,
     "a",  "5mths",  "8mths",   "13%",   "15%",      14,      15,
     "b",  "7mths",  "9mths",   "15%",   "17%",      16,      18
  )
child_stats_three_values
```

```
## # A tibble: 2 × 7
##   child t1_age t2_age t1_fat t2_fat t1_bmi t2_bmi
##   <chr> <chr>  <chr>  <chr>  <chr>   <dbl>  <dbl>
## 1 a     5mths  8mths  13%    15%        14     15
## 2 b     7mths  9mths  15%    17%        16     18
```

Here, in the column names there are three value types occurring after the "_" separator: `age`, `fat` and `bmi`; the ".value" string tells `pivot_longer()` to make a new column for each value type:

```
child_stats_three_values %>%
  pivot_longer(2:7,
               names_sep = "_",
               names_to = c("time", ".value")
               )
```

```
## # A tibble: 4 × 5
##   child time  age   fat     bmi
##   <chr> <chr> <chr> <chr> <dbl>
## 1 a     t1    5mths 13%      14
## 2 a     t2    8mths 15%      15
## 3 b     t1    7mths 15%      16
## 4 b     t2    9mths 17%      18
```

A pediatrician records the following information for a set of children over two years:

- head circumference;
- neck circumference; and
- hip circumference

all in centimeters.

The output table resembles the below:

```
growth_stats <-
  tibble::tribble(

~child,~yr1_head,~yr2_head,~yr1_neck,~yr2_neck,~yr1_hip,~yr2_hip,
       "a",       45,       48,       23,       24,      51,
52,
       "b",       48,       50,       24,       26,      52,
52,
       "c",       50,       52,       24,       27,      53,
54
    )

growth_stats
```


```
## # A tibble: 3 × 7
##   child yr1_head yr2_head yr1_neck yr2_neck yr1_hip yr2_hip
##   <chr>    <dbl>    <dbl>    <dbl>    <dbl>   <dbl>   <dbl>
## 1 a           45       48       23       24      51      52
## 2 b           48       50       24       26      52      52
## 3 c           50       52       24       27      53      54
```

Pivot the data into a long format to get the following structure:

child	year	head	neck	hip

```
Q_growth_stats_long <-
  growth_stats %>%
  pivot_longer(_________)
```

Value type *before* the separator

In all the example we have used so far, the column names were constructed such that value type came after the separator (Recall our table:

column_name	period	separator	".value"
year1_height	year1	_	height
year2_height	year2	_	height
year1_weight	year1	_	weight
year2_weight	year2	_	weight

)

But of course, the column names could be constructed differently, with the value types coming before the separator, as in this example:

```
child_stats2 <-
  tibble::tribble(
    ~child, ~height_year1, ~height_year2, ~weight_year1, ~weight_year2,
       "A",        "80cm",        "85cm",         "5kg",        "10kg",
       "B",        "85cm",        "90cm",         "7kg",        "12kg",
       "C",        "90cm",       "100cm",         "6kg",        "14kg"
    )

child_stats2
```

```
## # A tibble: 3 × 5
##   child height_year1 height_year2 weight_year1 weight_year2
##   <chr> <chr>        <chr>        <chr>        <chr>
## 1 A     80cm         85cm         5kg          10kg
## 2 B     85cm         90cm         7kg          12kg
## 3 C     90cm         100cm        6kg          14kg
```

Here, the value types (height and weight) come before the "_" separator.

How can our `pivot_longer()` command accommodate this? Simple! Just swap the order of the vector given to the `names_to` argument:

So instead of `names_to = c("time", ".value")`, you would have `names_to = c(".value", "time")`:

```
child_stats2 %>%
  pivot_longer(2:5,
               names_sep = "_",
               names_to = c(".value", "time"))
```

```
## # A tibble: 5 × 4
##   child time  height weight
##   <chr> <chr> <chr>  <chr>
## 1 A     year1 80cm   5kg
## 2 A     year2 85cm   10kg
## 3 B     year1 85cm   7kg
## 4 B     year2 90cm   12kg
## 5 C     year1 90cm   6kg
```

And that's it!

PRACTICE

(in RMD)

Consider the following data set from Zambia about enteropathogens and their biomarkers.

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

enteropathogens_zambia_wide
```

```
## # A tibble: 5 × 7
##      ID LPS_1 LPS_2  LBP_1 LBP_2 IFABP_1 IFABP_2
##   <dbl> <dbl> <dbl>  <dbl> <dbl>   <dbl>   <dbl>
## 1  1002  222.  390. 38414. 6840.  1294.     610.
## 2  1003  181.   NA  26888.   NA     22.5     NA
## 3  1004  257.  221. 49183. 5426.     0       0
## 4  1005   NA   369.    NA  1938.     0     1010.
## 5  1006  275.   NA  61758.   NA      0       NA
```

This data frame has the following columns:

- LPS_1 and LPS_2: lipopolysaccharide levels, measured by Pyrochrome LAL, in EU/mL
- LBP_1 and LBP_2: LPS binding protein levels, in pg/mL
- IFABP_1 and IFAPB_2: intestinal-type fatty acid binding protein levels, in pg/mL

Pivot the dataset so that it resembles the following structure

ID	sample_count	LPS	LBP	IFABP

```
enteropathogens_zambia_wide %>%
   pivot_longer(_____________)
```

A non-time-series example

So far we have been using person-period (time series) datasets to illustrate the idea of complex pivots with multiple value types.

But as we have mentioned, not all reshape-requiring datasets are time series data. Let's see a quick non-time-series example […]

You might measure the height (cm) and weight (kg) of a series of parental couples in a table like this:

```
family_stats <-
  tibble::tribble(
  ~couple, ~father_height, ~father_weight, ~mother_height, ~mother_weight,
      "a",            180,            80,            160,             70,
      "b",            185,            90,            150,             76,
      "c",            182,            93,            143,             78
  )
family_stats
```

```
## # A tibble: 3 × 5
##   couple father_height father_weight mother_height
##   <chr>          <dbl>         <dbl>         <dbl>
## 1 a                180            80           160
## 2 b                185            90           150
## 3 c                182            93           143
## # ℹ 1 more variable: mother_weight <dbl>
```

Here we have two different types of values (weight and height) for each person in the couple.

To pivot this to one-row per person, we'll again need the `names_sep` and `names_to` arguments:

```
family_stats %>%
  pivot_longer(2:5,
               names_sep  = "_",
               names_to = c("person", ".value"))
```

```
## # A tibble: 5 × 4
##   couple person height weight
##   <chr>  <chr>   <dbl>  <dbl>
## 1 a      father    180     80
## 2 a      mother    160     70
## 3 b      father    185     90
## 4 b      mother    150     76
## 5 c      father    182     93
```

The separator is an underscore, "_", so we used `names_sep = "_"` and because the value types come after the separator, the ".value" identifier was placed second in the `names_to` argument.

Escaping the dot separator

A special example may crop up when you try to pivot a dataset where the separator is a period.

```
child_stats_dot_sep <-
  tibble::tribble(
    ~child, ~year1.height, ~year2.height, ~year1.weight, ~year2.weight,
       "A",        "80cm",        "85cm",         "5kg",        "10kg",
       "B",        "85cm",        "90cm",         "7kg",        "12kg",
       "C",        "90cm",       "100cm",         "6kg",        "14kg"
    )

child_stats_dot_sep %>%
  pivot_longer(2:5,
               names_to = c("period", ".value"),
               names_sep = "\\.")
```

```
## # A tibble: 5 × 4
##   child period height weight
##   <chr> <chr>  <chr>  <chr>
## 1 A     year1  80cm   5kg
## 2 A     year2  85cm   10kg
## 3 B     year1  85cm   7kg
## 4 B     year2  90cm   12kg
## 5 C     year1  90cm   6kg
```

There we used the string "\." to indicate a dot "." because the "." is a special character in R, and sometimes needs to be escaped

Consider again the adult_stats data you saw above. Now the column names have been changed slightly.

```
adult_stats_dot_sep <-
  tibble::tribble(
    ~adult,  ~`BMI.year1`,  ~`BMI.year2`,  ~`HIV.year1`,
~`HIV.year2`,
       "A",           25,           30,    "Positive",
"Positive",
       "B",           34,           28,    "Negative",
"Positive",
       "C",           19,           17,    "Negative",
"Negative"
  )


adult_stats_dot_sep
```

```
## # A tibble: 3 × 5
##   adult BMI.year1 BMI.year2 HIV.year1 HIV.year2
##   <chr>     <dbl>     <dbl> <chr>     <chr>
## 1 A            25        30 Positive  Positive
```

```
## 2 B            34        28 Negative  Positive
## 3 C            19        17 Negative  Negative
```

Again, pivot the data into a long format to get the following structure:

adult	year	BMI	HIV

```
Q_adult2_long <-
  adult_stats_dot_sep %>%
  pivot_longer(_________)
```

What to do when you don't have a neat separator ?

Sometimes you do not have a neat separator.

Consider this survey data from India that looked at how much money patients spent on tuberculosis treatment:

```
tb_visits <- read_csv(here("data/india_tb_pathways_and_costs_data.csv")) %>%
  clean_names() %>%
  select(id, first_visit_location, first_visit_cost, second_visit_location,
second_visit_cost, third_visit_location, third_visit_cost)

tb_visits
```

```
## # A tibble: 5 × 7
##       id first_visit_location first_visit_cost
##    <dbl> <chr>                           <dbl>
## 1 100202 GH                                  0
## 2 100396 Pvt. docto                       1500
## 3 100590 Pvt. docto                       2000
## 4 100687 Pvt. hospi                      20000
## 5 100784 Pvt. docto                       1000
## # ℹ 4 more variables: second_visit_location <chr>,
## #   second_visit_cost <dbl>, third_visit_location <chr>, …
```

It does not have a neat separator between the time indicators (first, second, third) and the value type (cost, location). That is, rather than something like "firstvisit_location", we have instead "first_visit_location", so the underscore is used for two purposes. For this reason, if you try our usual pivot strategy, you will get an error:

```
tb_visits %>%
  pivot_longer(2:7,
               names_to = c("visit_count", ".value"),
               names_sep = "_")
```

```
Error in `pivot_longer_spec()`:
! Can't combine `first_visit_location` <character> and `first_visit_cost`
<double>.
Run `rlang::last_error()` to see where the error occurred.
```

The most direct way to reshape this dataset successfully would be to use special "regex" (string manipulation), but you likely have not learned this yet!

So for now, the solution we recommend is to manually rename your columns to insert a clear separator, "__":

```
tb_visits_renamed <-
  tb_visits %>%
  rename(first__visit_location = first_visit_location,
         first__visit_cost = first_visit_cost,
         second__visit_location = second_visit_location,
         second__visit_cost= second_visit_cost,
         third__visit_location = third_visit_location,
         third__visit_cost = third_visit_cost)

tb_visits_renamed
```

```
## # A tibble: 5 × 7
##       id first__visit_location first__visit_cost
##    <dbl> <chr>                              <dbl>
## 1 100202 GH                                     0
## 2 100396 Pvt. docto                          1500
## 3 100590 Pvt. docto                          2000
## 4 100687 Pvt. hospi                         20000
## 5 100784 Pvt. docto                          1000
## # ℹ 4 more variables: second__visit_location <chr>,
## #   second__visit_cost <dbl>, …
```

Now we can try the pivot:

```
tb_visits_long <-
  tb_visits_renamed %>%
  pivot_longer(2:7,
               names_to = c("visit_count", ".value"),
               names_sep = "__")
tb_visits_long
```

```
## # A tibble: 5 × 4
##       id visit_count visit_location visit_cost
##    <dbl> <chr>       <chr>                <dbl>
## 1 100202 first       GH                       0
## 2 100202 second      <NA>                     0
## 3 100202 third       <NA>                     0
```

```
## 4 100396 first       Pvt. docto           1500
## 5 100396 second      Pvt. clini           1000
```

Now let's polish the data frame:

```
tb_visits_long %>%
  # remove nonexistent entries
  filter(!visit_location == "") %>%
  # give significant naming to the visit_count values
  mutate(visit_count = case_when(visit_count == "first" ~ 1,
                                 visit_count == "second" ~ 2,
                                 visit_count == "third" ~ 3)) %>%
  # ensure visit_cost is numerical
  mutate(visit_cost = as.numeric(visit_cost))
```

```
## # A tibble: 5 × 4
##       id visit_count visit_location visit_cost
##    <dbl>       <dbl> <chr>                <dbl>
## 1 100202           1 GH                       0
## 2 100396           1 Pvt. docto            1500
## 3 100396           2 Pvt. clini            1000
## 4 100396           3 Pvt. hospi            2500
## 5 100590           1 Pvt. docto            2000
```

Above, we first remove the entries where we do not have the visit location information (i.e. we filter out the rows where the visit location variable is set to ""). We then convert to numeric values the visit count variable, where the strings `"first"` to `"third"` are converted to numerical entries 1 to 3. Finally, we ensure the variable of visit cost is numeric using `mutate()` and the helper function `as.numeric()`.

We will use a survey data about diet from Vietnam. Women in Hanoi were interviewed about their food shopping, and this was used to create nutrition profiles for each women. Here we will use a subset of this data for 61 households who came for 2 visits, recording:

- `enerc_kcal_w_1`: the consumed energy from ingredient/food (Kcal) during the first visit (with _2 for the second visit)
- `dry_w_1`: the consumed dry from ingredient/food (g) during the first visit (with _2 for the second visit)
- `water_w_1`: the consumed water from ingredient/food (g) during the first visit (with _2 for the second visit)

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

diet_diversity_vietnam_wide
```

```
## # A tibble: 5 × 9
##   household_id enerc_kcal_w_1 enerc_kcal_w_2 dry_w_1
dry_w_2
##          <dbl>          <dbl>          <dbl>   <dbl>
<dbl>
## 1          348          2268.          1386.    548.
281.
## 2          354          2775.          1240.    600.
284.
## 3           53          3104.          2075.    646.
451.
## 4           18          2802.          2146.    620.
807.
## 5          211          1298.          1191.    269.
288.
## # ℹ 4 more variables: water_w_1 <dbl>, water_w_2 <dbl>,
## #   fat_w_1 <dbl>, fat_w_2 <dbl>
```

You should first distinguish if we have a neat operator or not. Based on this, rename your columns if necessary. Then bring the different visit records (1 and 2) into a sole column for energy, fat weight, water weight and dry weight. In other words, pivot the dataset into long format of this form:

household_id	visit	enerc_kcal_w	dry_w	water_w	fat_w

```
Q_diet_diversity_vietnam_long <-
  diet_diversity_vietnam_wide %>%
  pivot_longer(_________)
```

Long to wide

We just saw how to do some complex operations wide to long, which we saw in the previous lesson is essential for plotting and wrangling. Let's see the opposite transformation.

It could be useful to put long to wide to do different transformations, filters, and processing NAs. In this format, your measurements / collected data become the columns of the data set.

Let's take the Zambia enteropathogen data, and this time, let's take the original ! Indeed, what you were handling before was a dataset **prepared for you**, in a wide format. **The original dataset is long** and we will now see the data preparation I did beforehand, behind the scenes. You're almost becoming the teacher of this lesson ;)

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

```
## # A tibble: 5 × 5
##      ID group   LPS    LBP  IFABP
##   <dbl> <dbl> <dbl>  <dbl>  <dbl>
## 1  1002     1  222. 38414. 1294.
## 2  1002     2  390.  6840.  610.
## 3  1003     1  181. 26888.   22.5
## 4  1004     2  221.  5426.    0
## 5  1004     1  257. 49183.    0
```

This is how we convert it from long to wide:

```
enteropathogens_zambia_wide <-
  enteropathogens_zambia_long %>%
  pivot_wider(
    names_from = group,
    values_from = c(LPS, LBP, IFABP)
  )

enteropathogens_zambia_wide
```

```
## # A tibble: 5 × 7
##      ID LPS_1 LPS_2  LBP_1 LBP_2 IFABP_1 IFABP_2
##   <dbl> <dbl> <dbl>  <dbl> <dbl>   <dbl>   <dbl>
## 1  1002  222.  390. 38414. 6840.  1294.     610.
## 2  1003  181.   NA  26888.   NA     22.5     NA
## 3  1004  257.  221. 49183. 5426.     0        0
## 4  1005   NA   369.    NA  1938.     0     1010.
## 5  1006  275.   NA  61758.   NA      0       NA
```

You can see that the values of the variable `group` (1 or 2) are added to the values' names (LPS, LBP, IFABP) to create the new columns representing different group data: for example, LPS_1 and LPS_2.

We are considering this "advanced" pivoting because we are pivoting wider several variables at the same time, but as you can see, the syntax is quite simple—the same arguments are used as we did with the simpler pivots in the previous lesson—`names_from` and `values_from`.

Let's see another example, using the diet survey data from Vietnam that you manipulated previously:

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

```
## # A tibble: 5 × 6
##   visit_number household_id enerc_kcal_w dry_w water_w fat_w
##          <dbl>        <dbl>        <dbl> <dbl>   <dbl> <dbl>
## 1            1          348        2268.  548.   4219.  78.4
## 2            1          354        2775.  600.   2376. 115.
## 3            1           53        3104.  646.   2808. 127.
## 4            1           18        2802.  620.   3457.  87.4
## 5            1          211        1298.  269.   2584.  47.8
```

Here we will use the `visit_number` variable to create new variable for energy, water, fat and dry content of foods recorded at different visits:

```
diet_diversity_vietnam_wide <-
  diet_diversity_vietnam_long %>%
  pivot_wider(
    names_from = visit_number,
    values_from = c(enerc_kcal_w, dry_w, water_w, fat_w)
  )

diet_diversity_vietnam_wide
```

```
## # A tibble: 5 × 9
##   household_id enerc_kcal_w_1 enerc_kcal_w_2 dry_w_1 dry_w_2
##          <dbl>          <dbl>          <dbl>   <dbl>   <dbl>
## 1          348          2268.          1386.    548.    281.
## 2          354          2775.          1240.    600.    284.
## 3           53          3104.          2075.    646.    451.
## 4           18          2802.          2146.    620.    807.
## 5          211          1298.          1191.    269.    288.
## # ℹ 4 more variables: water_w_1 <dbl>, water_w_2 <dbl>,
## #   fat_w_1 <dbl>, fat_w_2 <dbl>
```

You can see that the values of the variable `visit_number` (1 or 2) are added to the values' names (`energy_kcal_w`, `dry_w`, `fat_w`, `water_w`) to create the new columns representing different group data: for example, `water_w_1` and `water_w_2`. We have pivoted to wide format all of these variables at the same time. Now each weight measure per visit is represented as a single variable (i.e. column) in the dataset.

With this format, it is easy to sum together the energy intake per household for example:

```
diet_diversity_vietnam_wide %>%
  select(household_id, enerc_kcal_w_1, enerc_kcal_w_2) %>%
  mutate(total_energy_kcal = enerc_kcal_w_1 + enerc_kcal_w_2) %>%
  arrange(household_id)
```

```
## # A tibble: 5 × 4
##   household_id enerc_kcal_w_1 enerc_kcal_w_2
##          <dbl>          <dbl>          <dbl>
## 1           14          1040.          1663.
## 2           17          2100.          1286.
## 3           18          2802.          2146.
## 4           22          3187.          1582.
## 5           24          2359.          2026.
## # ℹ 1 more variable: total_energy_kcal <dbl>
```

However, you could get something similar in the long format:

```
diet_diversity_vietnam_long %>%
  group_by(household_id) %>%
  summarize(total_energy = sum(enerc_kcal_w))
```

```
## # A tibble: 5 × 2
##   household_id total_energy
##          <dbl>        <dbl>
## 1           14        2704.
## 2           17        3386.
## 3           18        4948.
## 4           22        4769.
## 5           24        4385.
```

Take `tb_visits_long` dataset that we manipulated above and pivot it back to a wide format.

```
Q_tb_visit_wide <-
  tb_visits_long %>%
  pivot_wider(_________)
```

Wrap Up !

You data wrangling skills have just been enhanced with advanced pivoting. This skill will often prove essential when handling real world data. I have no doubt you will soon put it

into practice. It is also essential, as we have seen, for plotting. So I hope pivoting will be of use not only for your wrangling, but also for your plotting tasks.

Contributors

The following team members contributed to this lesson:

KENE DAVID NWOSU

Data analyst, the GRAPH Network
Passionate about world improvement

LAURE VANCAUWENBERGHE

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

CAMILLE BEATRICE VALERA

Project Manager and Scientific Collaborator, The GRAPH Network

References