Tidy Data and Tidying Data

D.Gerard & J. Wall

2020-03-15

# Learning Objectives

* Identify tidy and non-tidy data
* Learn to make your data tidy:
  + “Pivotting” which converts between long and wide forms with pivot\_longer(), pivot\_wider()
  + Splitting and combining character columns:
    - Use separate() and extract() to pull a single character column into multiple columns;
    - use unite() to combine multiple columns into a single character column.
  + Make implicit missing values explicit with complete()
  + Make explicit missing values implicit with values\_drop\_na()
  + Replace missing values with next/previous value with fill(), or a known value with replace\_na()

# Resources

* Chapter 12 of [RDS](https://r4ds.had.co.nz/tidy-data.html)
* [Data Import Cheat Sheet](https://github.com/rstudio/cheatsheets/blob/master/data-import.pdf)
* [Article by Hadley Wickham](http://vita.had.co.nz/papers/tidy-data.pdf)
* [Tidyr Vignette](https://tidyr.tidyverse.org/articles/pivot.html)
* [Pivot Vignette](https://tidyr.tidyverse.org/articles/pivot.html)
* [Separate Vignette](https://tidyr.tidyverse.org/reference/separate.html)
* [Unite Vignette](https://tidyr.tidyverse.org/reference/unite.html)

# Tidy Data

* Recall:
  + Observations/units/subjects/individuals/cases: objects described by a set of data (e.g. cars, people, countries).
  + Variable: describes some characteristic of the units (e.g. mpg, age, GDP).
  + Each unit has a single value of each variable (e.g. 20 mpg, 31 years old, million).
* Tidy Data:
  + One observation per row.
  + One variable per column.
  + One value per cell.
* Hadley’s visualization:
* 
* We will use the tidyr package (a member of the tidyverse) to make data tidy.
* library(tidyverse)
* Frequent Characteristics of “Nontidy” Data
  + Column headers are values, not variable names.
  + Multiple variables are stored in one column.
  + Variables are stored in both rows and columns.
  + Multiple types of observational units are stored in the same table.
  + A single observational unit is stored in multiple tables.

Look at table1, table2, table3, table4a, table4b. Which one is tidy? Why?

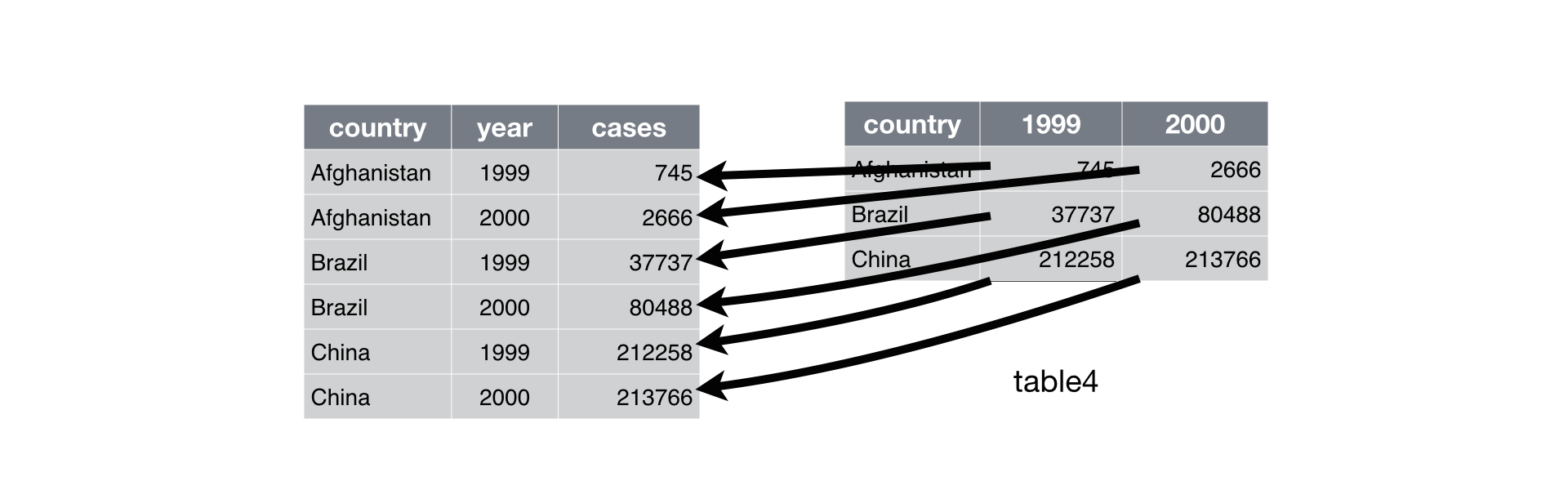
It is easy to analyze and plot your data with tidy data …. is why it is called the tidyverse.

```r  
table1  
```  
  
```  
## # A tibble: 6 x 4  
## country year cases population  
## <chr> <int> <int> <int>  
## 1 Afghanistan 1999 745 19987071  
## 2 Afghanistan 2000 2666 20595360  
## 3 Brazil 1999 37737 172006362  
## 4 Brazil 2000 80488 174504898  
## 5 China 1999 212258 1272915272  
## 6 China 2000 213766 1280428583  
```  
  
```r  
table2  
```  
  
```  
## # A tibble: 12 x 4  
## country year type count  
## <chr> <int> <chr> <int>  
## 1 Afghanistan 1999 cases 745  
## 2 Afghanistan 1999 population 19987071  
## 3 Afghanistan 2000 cases 2666  
## 4 Afghanistan 2000 population 20595360  
## 5 Brazil 1999 cases 37737  
## 6 Brazil 1999 population 172006362  
## 7 Brazil 2000 cases 80488  
## 8 Brazil 2000 population 174504898  
## 9 China 1999 cases 212258  
## 10 China 1999 population 1272915272  
## 11 China 2000 cases 213766  
## 12 China 2000 population 1280428583  
```  
  
```r  
table3  
```  
  
```  
## # A tibble: 6 x 3  
## country year rate   
## \* <chr> <int> <chr>   
## 1 Afghanistan 1999 745/19987071   
## 2 Afghanistan 2000 2666/20595360   
## 3 Brazil 1999 37737/172006362   
## 4 Brazil 2000 80488/174504898   
## 5 China 1999 212258/1272915272  
## 6 China 2000 213766/1280428583  
```  
  
```r  
table4a  
```  
  
```  
## # A tibble: 3 x 3  
## country `1999` `2000`  
## \* <chr> <int> <int>  
## 1 Afghanistan 745 2666  
## 2 Brazil 37737 80488  
## 3 China 212258 213766  
```  
  
```r  
table4b  
```  
  
```  
## # A tibble: 3 x 3  
## country `1999` `2000`  
## \* <chr> <int> <int>  
## 1 Afghanistan 19987071 20595360  
## 2 Brazil 172006362 174504898  
## 3 China 1272915272 1280428583  
```  
  
```r  
# rate per 10,000  
table1 %>%  
 mutate(rate = cases/population\*10000)  
```  
  
```  
## # A tibble: 6 x 5  
## country year cases population rate  
## <chr> <int> <int> <int> <dbl>  
## 1 Afghanistan 1999 745 19987071 0.373  
## 2 Afghanistan 2000 2666 20595360 1.29   
## 3 Brazil 1999 37737 172006362 2.19   
## 4 Brazil 2000 80488 174504898 4.61   
## 5 China 1999 212258 1272915272 1.67   
## 6 China 2000 213766 1280428583 1.67  
```  
  
```r  
# cases per year  
table1 %>%  
 count(year, wt=cases)  
```  
  
```  
## # A tibble: 2 x 2  
## year n  
## \* <int> <int>  
## 1 1999 250740  
## 2 2000 296920  
```  
  
```r  
# graph change over time  
ggplot(table1, aes(year, cases)) +  
 geom\_line(aes(group = country, color = country)) +  
 geom\_point(aes(color = country))  
```  
  
![](tidyr\_files/figure-docx/tidytables-1.png)

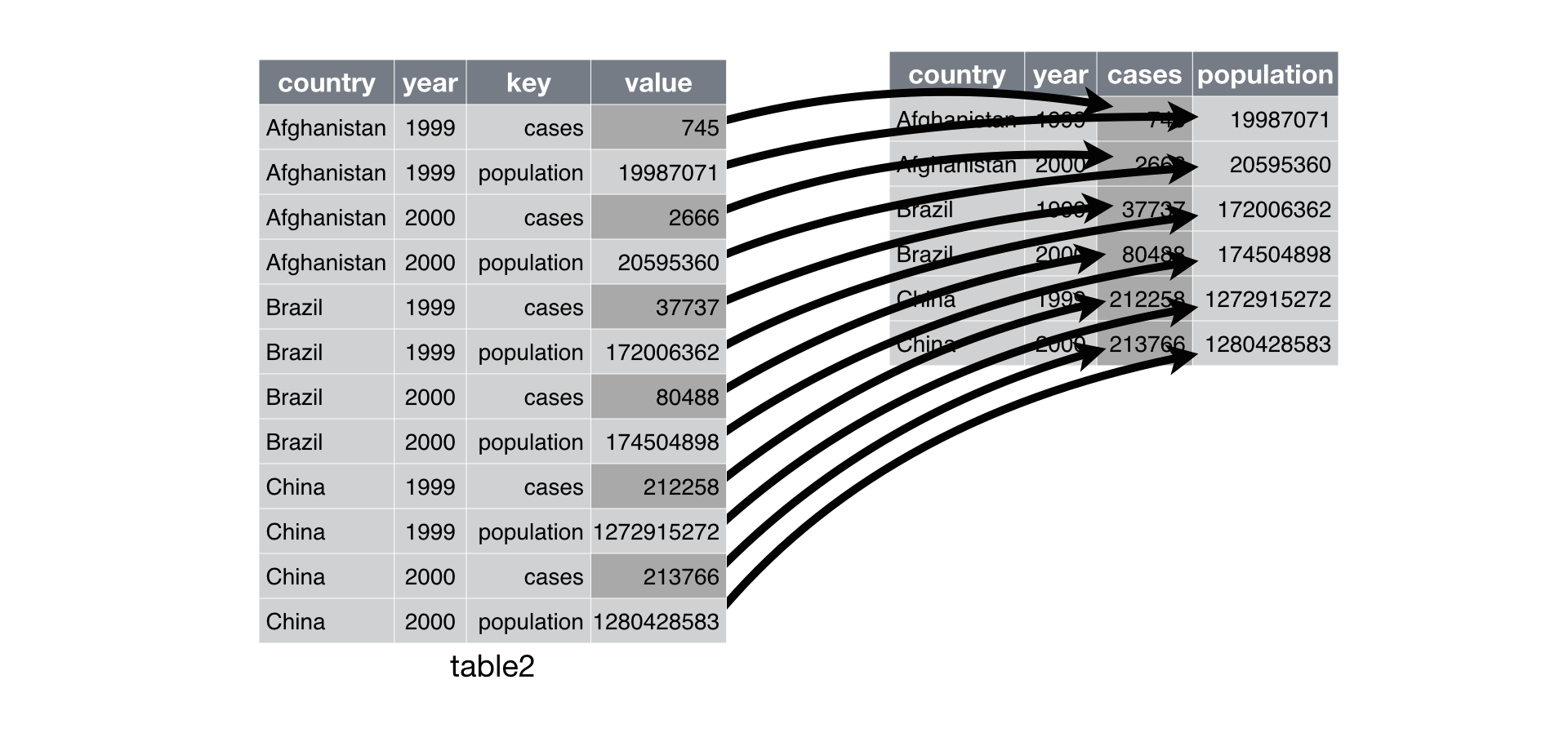
* Example of tidy data:
* tidyr::table1
* ## # A tibble: 6 x 4  
  ## country year cases population  
  ## <chr> <int> <int> <int>  
  ## 1 Afghanistan 1999 745 19987071  
  ## 2 Afghanistan 2000 2666 20595360  
  ## 3 Brazil 1999 37737 172006362  
  ## 4 Brazil 2000 80488 174504898  
  ## 5 China 1999 212258 1272915272  
  ## 6 China 2000 213766 1280428583
  + Variables: Country, Year, Cases, Population
  + Units: locationtime
* Untidy data: Each observational unit is spread across multiple rows
* print(tidyr::table2, n = 12)
* ## # A tibble: 12 x 4  
  ## country year type count  
  ## <chr> <int> <chr> <int>  
  ## 1 Afghanistan 1999 cases 745  
  ## 2 Afghanistan 1999 population 19987071  
  ## 3 Afghanistan 2000 cases 2666  
  ## 4 Afghanistan 2000 population 20595360  
  ## 5 Brazil 1999 cases 37737  
  ## 6 Brazil 1999 population 172006362  
  ## 7 Brazil 2000 cases 80488  
  ## 8 Brazil 2000 population 174504898  
  ## 9 China 1999 cases 212258  
  ## 10 China 1999 population 1272915272  
  ## 11 China 2000 cases 213766  
  ## 12 China 2000 population 1280428583
* Untidy data: Two variables are in one column
* tidyr::table3
* ## # A tibble: 6 x 3  
  ## country year rate   
  ## \* <chr> <int> <chr>   
  ## 1 Afghanistan 1999 745/19987071   
  ## 2 Afghanistan 2000 2666/20595360   
  ## 3 Brazil 1999 37737/172006362   
  ## 4 Brazil 2000 80488/174504898   
  ## 5 China 1999 212258/1272915272  
  ## 6 China 2000 213766/1280428583
* Untidy data: Data are spread across two data frames. Within each data frame, multiple observations are in one row.
* tidyr::table4a
* ## # A tibble: 3 x 3  
  ## country `1999` `2000`  
  ## \* <chr> <int> <int>  
  ## 1 Afghanistan 745 2666  
  ## 2 Brazil 37737 80488  
  ## 3 China 212258 213766
* tidyr::table4b
* ## # A tibble: 3 x 3  
  ## country `1999` `2000`  
  ## \* <chr> <int> <int>  
  ## 1 Afghanistan 19987071 20595360  
  ## 2 Brazil 172006362 174504898  
  ## 3 China 1272915272 1280428583
* Sometimes it is easy to determine the units and the variables.
* Sometimes it is very hard and you need to talk to the data collectors to find out.
* We want tidy data because R easily manipulates vectors. So in the long run it will make your life easier to first make data tidy.
* **Exercise 1** The following built-in datasets are not tidy. For each one, describe why it is not tidy and write out what the first five entries would look like once it is in a tidy format.
  1. relig\_income
  2. billboard
  3. us\_rent\_income

# intro to pivot\_longer and pivot\_wider

## When one variable spread across multiple columns, use pivot\_longer()

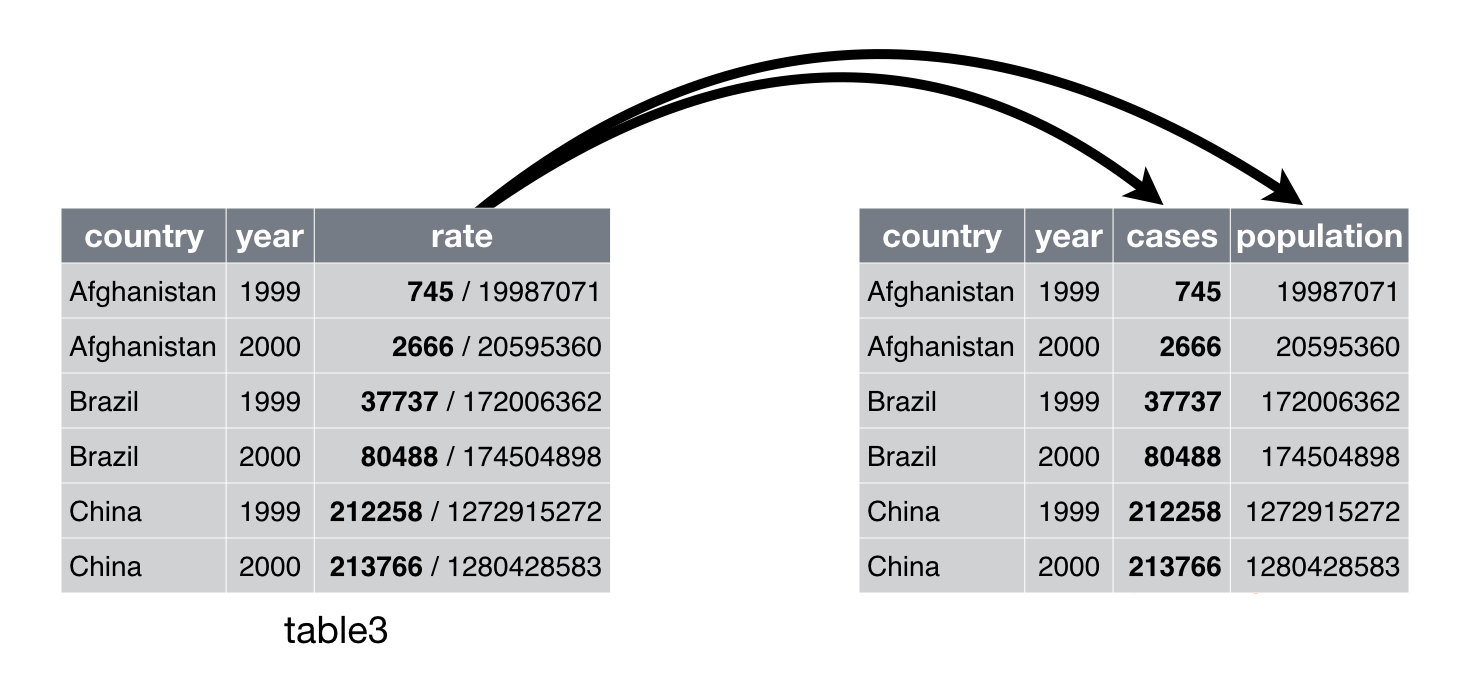
* Column names are actually *values* of a variable
* table4a and table4b
* Solution: pivot\_longer()
* Hadley’s visualization:
* 
* Specify
  1. The columns that are values, not variables,
  2. The name of the variable that will take the values of the column names (names\_to), and
  3. The name of the variable that will take the values spread in the cells (values\_to).
* table4a
* ## # A tibble: 3 x 3  
  ## country `1999` `2000`  
  ## \* <chr> <int> <int>  
  ## 1 Afghanistan 745 2666  
  ## 2 Brazil 37737 80488  
  ## 3 China 212258 213766
* tidy4a <- table4a %>%  
   pivot\_longer(cols = c(`1999`,`2000`), names\_to = "year",   
   values\_to = "cases")
* **Exercise 2** Try on your own to do the same thing to table4b. We will learn next class how to join these two data frames next week. But the code is
* full\_join(tidy4a, tidy4b)
* ## Joining, by = c("country", "year")
* ## # A tibble: 6 x 4  
  ## country year cases population  
  ## \* <chr> <chr> <int> <int>  
  ## 1 Afghanistan 1999 745 19987071  
  ## 2 Afghanistan 2000 2666 20595360  
  ## 3 Brazil 1999 37737 172006362  
  ## 4 Brazil 2000 80488 174504898  
  ## 5 China 1999 212258 1272915272  
  ## 6 China 2000 213766 1280428583
* **Exercise 3**: Tidy built-in dataset relig\_income
* ## # A tibble: 180 x 3  
  ## religion income count  
  ## <chr> <chr> <dbl>  
  ## 1 Agnostic <$10k 27  
  ## 2 Agnostic $10-20k 34  
  ## 3 Agnostic $20-30k 60  
  ## 4 Agnostic $30-40k 81  
  ## 5 Agnostic $40-50k 76  
  ## 6 Agnostic $50-75k 137  
  ## 7 Agnostic $75-100k 122  
  ## 8 Agnostic $100-150k 109  
  ## 9 Agnostic >150k 84  
  ## 10 Agnostic Don't know/refused 96  
  ## # ... with 170 more rows
* **Exercise 4**: gather the monkeymem data frame (available at <https://dcgerard.github.io/stat_412_612/data/monkeymem.csv>). The cell values represent identification accuracy of some objects (in percent of 20 trials).
* ## Parsed with column specification:  
  ## cols(  
  ## Monkey = col\_character(),  
  ## Treatment = col\_character(),  
  ## Week2 = col\_double(),  
  ## Week4 = col\_double(),  
  ## Week8 = col\_double(),  
  ## Week12 = col\_double(),  
  ## Week16 = col\_double()  
  ## )
* ## # A tibble: 90 x 4  
  ## Monkey Treatment Week Percent  
  ## <chr> <chr> <chr> <dbl>  
  ## 1 Spank Control Week2 95  
  ## 2 Spank Control Week4 75  
  ## 3 Spank Control Week8 80  
  ## 4 Spank Control Week12 65  
  ## 5 Spank Control Week16 70  
  ## 6 Chim Control Week2 85  
  ## 7 Chim Control Week4 75  
  ## 8 Chim Control Week8 55  
  ## 9 Chim Control Week12 75  
  ## 10 Chim Control Week16 85  
  ## # ... with 80 more rows

## When an observation is scattered over multiple rows, use pivot\_wider()

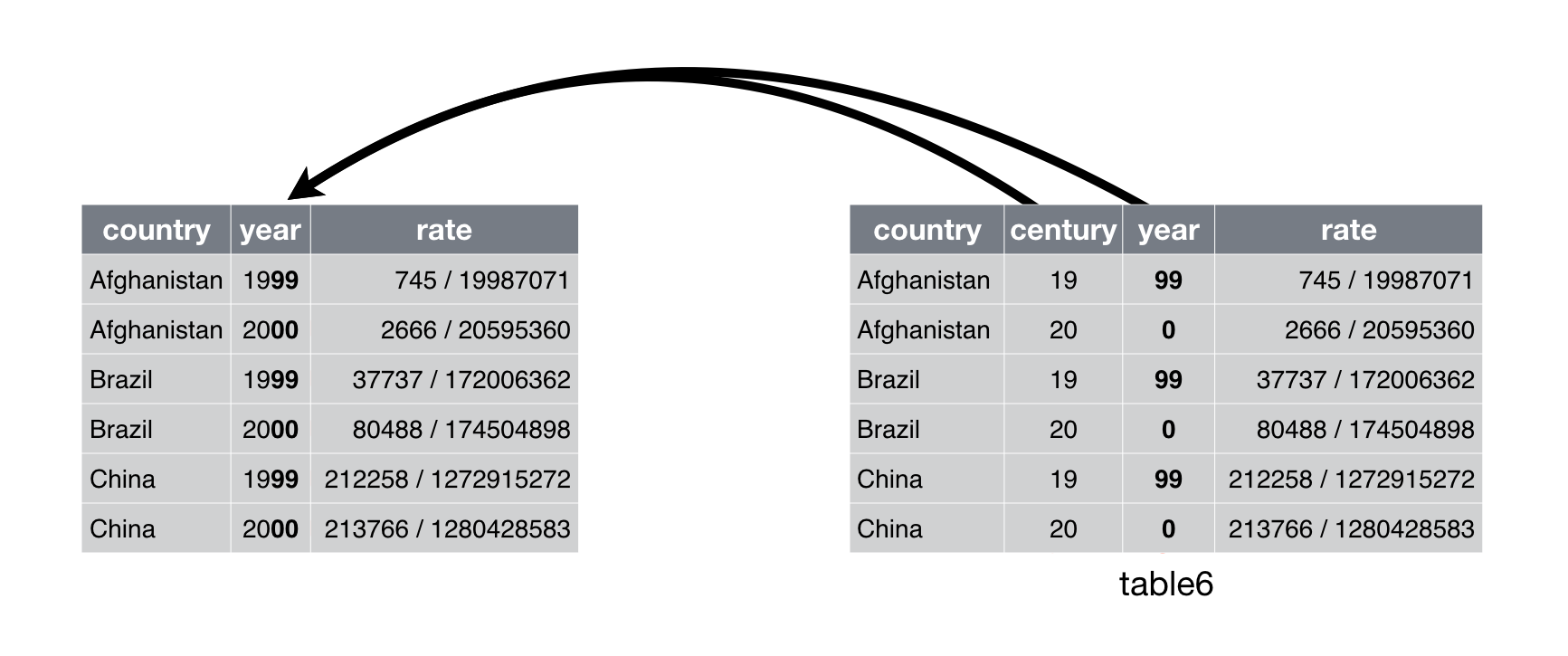
* pivot\_wider() makes a dataset wider by increasing the number of columns and decreasing the number of rows.
* One of the columns contains the *names* of the variables
* table2
* Solution: pivot\_wider()
* Hadley’s visualization:
* 
* Specify
  1. The column from which to pull the variable names (names\_from),
  2. The column from which to get the associated values of the variables.
* table2
* ## # A tibble: 12 x 4  
  ## country year type count  
  ## <chr> <int> <chr> <int>  
  ## 1 Afghanistan 1999 cases 745  
  ## 2 Afghanistan 1999 population 19987071  
  ## 3 Afghanistan 2000 cases 2666  
  ## 4 Afghanistan 2000 population 20595360  
  ## 5 Brazil 1999 cases 37737  
  ## 6 Brazil 1999 population 172006362  
  ## 7 Brazil 2000 cases 80488  
  ## 8 Brazil 2000 population 174504898  
  ## 9 China 1999 cases 212258  
  ## 10 China 1999 population 1272915272  
  ## 11 China 2000 cases 213766  
  ## 12 China 2000 population 1280428583
* table2 %>% pivot\_wider(names\_from = type, values\_from = count)
* ## # A tibble: 6 x 4  
  ## country year cases population  
  ## <chr> <int> <int> <int>  
  ## 1 Afghanistan 1999 745 19987071  
  ## 2 Afghanistan 2000 2666 20595360  
  ## 3 Brazil 1999 37737 172006362  
  ## 4 Brazil 2000 80488 174504898  
  ## 5 China 1999 212258 1272915272  
  ## 6 China 2000 213766 1280428583
* **Exercise 5**: Tidy the fish\_encounters dataset of fish spotting by monitoring stations. Make the NA into 0 using the option values\_fill = list(seen = 0)
* ## # A tibble: 19 x 12  
  ## fish Release I80\_1 Lisbon Rstr Base\_TD BCE BCW BCE2 BCW2 MAE MAW  
  ## <fct> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int>  
  ## 1 4842 1 1 1 1 1 1 1 1 1 1 1  
  ## 2 4843 1 1 1 1 1 1 1 1 1 1 1  
  ## 3 4844 1 1 1 1 1 1 1 1 1 1 1  
  ## 4 4845 1 1 1 1 1 0 0 0 0 0 0  
  ## 5 4847 1 1 1 0 0 0 0 0 0 0 0  
  ## 6 4848 1 1 1 1 0 0 0 0 0 0 0  
  ## 7 4849 1 1 0 0 0 0 0 0 0 0 0  
  ## 8 4850 1 1 0 1 1 1 1 0 0 0 0  
  ## 9 4851 1 1 0 0 0 0 0 0 0 0 0  
  ## 10 4854 1 1 0 0 0 0 0 0 0 0 0  
  ## 11 4855 1 1 1 1 1 0 0 0 0 0 0  
  ## 12 4857 1 1 1 1 1 1 1 1 1 0 0  
  ## 13 4858 1 1 1 1 1 1 1 1 1 1 1  
  ## 14 4859 1 1 1 1 1 0 0 0 0 0 0  
  ## 15 4861 1 1 1 1 1 1 1 1 1 1 1  
  ## 16 4862 1 1 1 1 1 1 1 1 1 0 0  
  ## 17 4863 1 1 0 0 0 0 0 0 0 0 0  
  ## 18 4864 1 1 0 0 0 0 0 0 0 0 0  
  ## 19 4865 1 1 1 0 0 0 0 0 0 0 0
* **Exercise 6**: Spread the flowers1 data frame (available at <https://dcgerard.github.io/stat_412_612/data/flowers1.csv>). Hint: use read\_csv2() to read in the dataset.
* ## Using ',' as decimal and '.' as grouping mark. Use read\_delim() for more control.
* ## Parsed with column specification:  
  ## cols(  
  ## Time = col\_double(),  
  ## replication = col\_double(),  
  ## Variable = col\_character(),  
  ## Value = col\_double()  
  ## )
* ## # A tibble: 24 x 4  
  ## Time replication Flowers Intensity  
  ## <dbl> <dbl> <dbl> <dbl>  
  ## 1 1 1 62.3 150  
  ## 2 1 2 77.4 150  
  ## 3 1 3 55.3 300  
  ## 4 1 4 54.2 300  
  ## 5 1 5 49.6 450  
  ## 6 1 6 61.9 450  
  ## 7 1 7 39.4 600  
  ## 8 1 8 45.7 600  
  ## 9 1 9 31.3 750  
  ## 10 1 10 44.9 750  
  ## # ... with 14 more rows

# separate and unite functions

## separate() splits a column into multiple columns.

* Problem: One column contains two (or more) variables.
* table3
* Solution: separate()
* Hadley’s visualization:
* 
* Specify:
  1. The column that contains two (or more) variables,
  2. A character vector of the new names of the variables, and
  3. The character that separates variables (or the position that separates variables) The default is the first non-numeric character.
* table3
* ## # A tibble: 6 x 3  
  ## country year rate   
  ## \* <chr> <int> <chr>   
  ## 1 Afghanistan 1999 745/19987071   
  ## 2 Afghanistan 2000 2666/20595360   
  ## 3 Brazil 1999 37737/172006362   
  ## 4 Brazil 2000 80488/174504898   
  ## 5 China 1999 212258/1272915272  
  ## 6 China 2000 213766/1280428583
* table3 %>%  
   separate(rate, into = c("cases","population"))
* ## # A tibble: 6 x 4  
  ## country year cases population  
  ## <chr> <int> <chr> <chr>   
  ## 1 Afghanistan 1999 745 19987071   
  ## 2 Afghanistan 2000 2666 20595360   
  ## 3 Brazil 1999 37737 172006362   
  ## 4 Brazil 2000 80488 174504898   
  ## 5 China 1999 212258 1272915272  
  ## 6 China 2000 213766 1280428583
* table3 %>%  
   separate(rate, into = c("cases","population"), sep="/")
* ## # A tibble: 6 x 4  
  ## country year cases population  
  ## <chr> <int> <chr> <chr>   
  ## 1 Afghanistan 1999 745 19987071   
  ## 2 Afghanistan 2000 2666 20595360   
  ## 3 Brazil 1999 37737 172006362   
  ## 4 Brazil 2000 80488 174504898   
  ## 5 China 1999 212258 1272915272  
  ## 6 China 2000 213766 1280428583
* table3 %>%  
   separate(  
   rate,   
   into = c("cases","population"),  
   convert = TRUE # to convert to numeric if desired  
   )
* ## # A tibble: 6 x 4  
  ## country year cases population  
  ## <chr> <int> <int> <int>  
  ## 1 Afghanistan 1999 745 19987071  
  ## 2 Afghanistan 2000 2666 20595360  
  ## 3 Brazil 1999 37737 172006362  
  ## 4 Brazil 2000 80488 174504898  
  ## 5 China 1999 212258 1272915272  
  ## 6 China 2000 213766 1280428583
* table5 <- table3 %>%  
   separate(year, into=c("century", "year"), sep=2)
* **Exercise 7:** Tidy the dataset flowers2.csv (available at <https://dcgerard.github.io/stat_412_612/data/flowers2.csv> ) by turning the one column into 3 separate columns.
* ## Parsed with column specification:  
  ## cols(  
  ## `Flowers/Intensity;Time` = col\_character()  
  ## )
* ## # A tibble: 24 x 3  
  ## Flowers Intensity Time   
  ## <chr> <chr> <chr>  
  ## 1 62.3 150 1   
  ## 2 77.4 150 1   
  ## 3 55.3 300 1   
  ## 4 54.2 300 1   
  ## 5 49.6 450 1   
  ## 6 61.9 450 1   
  ## 7 39.4 600 1   
  ## 8 45.7 600 1   
  ## 9 31.3 750 1   
  ## 10 44.9 750 1   
  ## # ... with 14 more rows

## unite is the inverse of separate()

* Problem: One variable spread across multiple columns.
* Solution: unite()
* Hadley’s visualization:
* 
* Much less common problem.
* table5 %>% unite(new, century, year)
* ## # A tibble: 6 x 3  
  ## country new rate   
  ## <chr> <chr> <chr>   
  ## 1 Afghanistan 19\_99 745/19987071   
  ## 2 Afghanistan 20\_00 2666/20595360   
  ## 3 Brazil 19\_99 37737/172006362   
  ## 4 Brazil 20\_00 80488/174504898   
  ## 5 China 19\_99 212258/1272915272  
  ## 6 China 20\_00 213766/1280428583
* table5 %>% unite(new, century, year, sep = "")
* ## # A tibble: 6 x 3  
  ## country new rate   
  ## <chr> <chr> <chr>   
  ## 1 Afghanistan 1999 745/19987071   
  ## 2 Afghanistan 2000 2666/20595360   
  ## 3 Brazil 1999 37737/172006362   
  ## 4 Brazil 2000 80488/174504898   
  ## 5 China 1999 212258/1272915272  
  ## 6 China 2000 213766/1280428583
* **Exercise 8**: Re-unite the data frame you separated from the flowers2 exercise. Use a comma for the separator.
* ## # A tibble: 24 x 2  
  ## `Flowers,Intensity` Time   
  ## <chr> <chr>  
  ## 1 62.3,150 1   
  ## 2 77.4,150 1   
  ## 3 55.3,300 1   
  ## 4 54.2,300 1   
  ## 5 49.6,450 1   
  ## 6 61.9,450 1   
  ## 7 39.4,600 1   
  ## 8 45.7,600 1   
  ## 9 31.3,750 1   
  ## 10 44.9,750 1   
  ## # ... with 14 more rows

# Dealing with Missing Values

Two types of missing values:

* explicitly missing (NA)
* implicitly missing (just row missing)

## Make NA’s explicit using complete()

First we will look at ways to make a missing row explicit The complete() function fills in NA’s where needed so each combination of columns is in the output

```r  
stocks <- tibble(  
 year = c(2015, 2015, 2015, 2015, 2016, 2016, 2016),  
 qtr = c( 1, 2, 3, 4, 2, 3, 4),  
 return = c(1.88, 0.59, 0.35, NA, 0.92, 0.17, 2.66)  
)  
stocks  
```  
  
```  
## # A tibble: 7 x 3  
## year qtr return  
## <dbl> <dbl> <dbl>  
## 1 2015 1 1.88  
## 2 2015 2 0.59  
## 3 2015 3 0.35  
## 4 2015 4 NA   
## 5 2016 2 0.92  
## 6 2016 3 0.17  
## 7 2016 4 2.66  
```  
  
```r  
stocks %>%  
 complete(year,qtr)  
```  
  
```  
## # A tibble: 8 x 3  
## year qtr return  
## <dbl> <dbl> <dbl>  
## 1 2015 1 1.88  
## 2 2015 2 0.59  
## 3 2015 3 0.35  
## 4 2015 4 NA   
## 5 2016 1 NA   
## 6 2016 2 0.92  
## 7 2016 3 0.17  
## 8 2016 4 2.66  
```

Here is a tibble where the item\_id and item\_name go together and we need to have all items showing for each group.

```r  
df <- tibble(  
 group = c(1:2, 1),  
 item\_id = c(1:2, 2),  
 item\_name = c("a", "b", "b"),  
 value1 = 1:3,  
 value2 = 4:6  
)  
df  
```  
  
```  
## # A tibble: 3 x 5  
## group item\_id item\_name value1 value2  
## <dbl> <dbl> <chr> <int> <int>  
## 1 1 1 a 1 4  
## 2 2 2 b 2 5  
## 3 1 2 b 3 6  
```

Note that there is no row for group 2 with item\_id = 1, item\_name = a. We specify that we want to expand the group to accomodate the patterns present in item\_id and item\_name. In this case, those are sort of equivalent in that the item\_name goes with the item\_id. Since they are equivalent, we use the nesting to tell the code to group those together. We also used the fill option to specify that missing value1’s should be given a 0 rather than an NA.

```r  
df %>% complete(group, nesting(item\_id, item\_name),   
 fill = list(value1 = 0))  
```  
  
```  
## # A tibble: 4 x 5  
## group item\_id item\_name value1 value2  
## <dbl> <dbl> <chr> <dbl> <int>  
## 1 1 1 a 1 4  
## 2 1 2 b 3 6  
## 3 2 1 a 0 NA  
## 4 2 2 b 2 5  
```

* **Exercise 9** In the following dataset, turn the implicit missing values to explicit.
* output <- tibble(  
   treatment = c("a", "b", "a", "c", "b"),  
   gender = factor(c("M", "F", "F", "M", "M"), levels = c("M", "F", "O")),  
   return = c(1.5, 0.75, 0.5, 1.8, NA)  
  )  
  output
* ## # A tibble: 5 x 3  
  ## treatment gender return  
  ## <chr> <fct> <dbl>  
  ## 1 a M 1.5   
  ## 2 b F 0.75  
  ## 3 a F 0.5   
  ## 4 c M 1.8   
  ## 5 b M NA
* ## # A tibble: 9 x 3  
  ## treatment gender return  
  ## <chr> <fct> <dbl>  
  ## 1 a M 1.5   
  ## 2 a F 0.5   
  ## 3 a O NA   
  ## 4 b M NA   
  ## 5 b F 0.75  
  ## 6 b O NA   
  ## 7 c M 1.8   
  ## 8 c F NA   
  ## 9 c O NA

## Make explicit NA’s implicit using values\_drop\_na

To turn explicit missing values to implicit missing values, you can use values\_drop\_na = TRUE as an option to pivot\_longer()

```r  
stocks %>%  
 pivot\_wider(  
 names\_from = year,   
 values\_from = return) %>%  
 pivot\_longer(  
 cols = c(`2015`, `2016`),  
 names\_to = "year",  
 values\_to = "return",   
 values\_drop\_na = TRUE)  
```  
  
```  
## # A tibble: 6 x 3  
## qtr year return  
## <dbl> <chr> <dbl>  
## 1 1 2015 1.88  
## 2 2 2015 0.59  
## 3 2 2016 0.92  
## 4 3 2015 0.35  
## 5 3 2016 0.17  
## 6 4 2016 2.66  
```

## fill()

A similar option uses fill to carry forward the previous value to any missing spots.

```r  
stocks %>%  
 fill(return)  
```  
  
```  
## # A tibble: 7 x 3  
## year qtr return  
## <dbl> <dbl> <dbl>  
## 1 2015 1 1.88  
## 2 2015 2 0.59  
## 3 2015 3 0.35  
## 4 2015 4 0.35  
## 5 2016 2 0.92  
## 6 2016 3 0.17  
## 7 2016 4 2.66  
```

## replace\_na()

We can also use the replace\_na() function to replace the NA values in our tibble with other things.

```r  
df2 <- df %>% complete(group, nesting(item\_id, item\_name))  
df2 %>% replace\_na(list(value1 = 0, value2 = "unknown"))  
```  
  
```  
## # A tibble: 4 x 5  
## group item\_id item\_name value1 value2   
## <dbl> <dbl> <chr> <dbl> <chr>   
## 1 1 1 a 1 4   
## 2 1 2 b 3 6   
## 3 2 1 a 0 unknown  
## 4 2 2 b 2 5  
```

# names\_pattern()

* Problem: The value of the variable is part of the column name.
* Solution: names\_pattern() option of pivot\_longer()
* preg <- read\_csv("./Data/preg.csv")
* ## Parsed with column specification:  
  ## cols(  
  ## name = col\_character(),  
  ## treatmenta = col\_double(),  
  ## treatmentb = col\_double()  
  ## )
* preg
* ## # A tibble: 3 x 3  
  ## name treatmenta treatmentb  
  ## <chr> <dbl> <dbl>  
  ## 1 John Smith NA 18  
  ## 2 Jane Doe 4 1  
  ## 3 Mary Johnson 6 7

tidy preg

```r  
preg2 <- preg %>% pivot\_longer(  
 cols = starts\_with("treatment"),  
 names\_to = "treatment",  
 values\_to = "result",  
 names\_pattern = "treatment(.)")  
  
preg2  
```  
  
```  
## # A tibble: 6 x 3  
## name treatment result  
## <chr> <chr> <dbl>  
## 1 John Smith a NA  
## 2 John Smith b 18  
## 3 Jane Doe a 4  
## 4 Jane Doe b 1  
## 5 Mary Johnson a 6  
## 6 Mary Johnson b 7  
```

# Multiple variables stored in one column

Look at the built-in dataset who.

```r  
who  
```  
  
```  
## # A tibble: 7,240 x 60  
## country iso2 iso3 year new\_sp\_m014 new\_sp\_m1524 new\_sp\_m2534 new\_sp\_m3544  
## <chr> <chr> <chr> <int> <int> <int> <int> <int>  
## 1 Afghan~ AF AFG 1980 NA NA NA NA  
## 2 Afghan~ AF AFG 1981 NA NA NA NA  
## 3 Afghan~ AF AFG 1982 NA NA NA NA  
## 4 Afghan~ AF AFG 1983 NA NA NA NA  
## 5 Afghan~ AF AFG 1984 NA NA NA NA  
## 6 Afghan~ AF AFG 1985 NA NA NA NA  
## 7 Afghan~ AF AFG 1986 NA NA NA NA  
## 8 Afghan~ AF AFG 1987 NA NA NA NA  
## 9 Afghan~ AF AFG 1988 NA NA NA NA  
## 10 Afghan~ AF AFG 1989 NA NA NA NA  
## # ... with 7,230 more rows, and 52 more variables: new\_sp\_m4554 <int>,  
## # new\_sp\_m5564 <int>, new\_sp\_m65 <int>, new\_sp\_f014 <int>,  
## # new\_sp\_f1524 <int>, new\_sp\_f2534 <int>, new\_sp\_f3544 <int>,  
## # new\_sp\_f4554 <int>, new\_sp\_f5564 <int>, new\_sp\_f65 <int>,  
## # new\_sn\_m014 <int>, new\_sn\_m1524 <int>, new\_sn\_m2534 <int>,  
## # new\_sn\_m3544 <int>, new\_sn\_m4554 <int>, new\_sn\_m5564 <int>,  
## # new\_sn\_m65 <int>, new\_sn\_f014 <int>, new\_sn\_f1524 <int>,  
## # new\_sn\_f2534 <int>, new\_sn\_f3544 <int>, new\_sn\_f4554 <int>,  
## # new\_sn\_f5564 <int>, new\_sn\_f65 <int>, new\_ep\_m014 <int>,  
## # new\_ep\_m1524 <int>, new\_ep\_m2534 <int>, new\_ep\_m3544 <int>,  
## # new\_ep\_m4554 <int>, new\_ep\_m5564 <int>, new\_ep\_m65 <int>,  
## # new\_ep\_f014 <int>, new\_ep\_f1524 <int>, new\_ep\_f2534 <int>,  
## # new\_ep\_f3544 <int>, new\_ep\_f4554 <int>, new\_ep\_f5564 <int>,  
## # new\_ep\_f65 <int>, newrel\_m014 <int>, newrel\_m1524 <int>,  
## # newrel\_m2534 <int>, newrel\_m3544 <int>, newrel\_m4554 <int>,  
## # newrel\_m5564 <int>, newrel\_m65 <int>, newrel\_f014 <int>,  
## # newrel\_f1524 <int>, newrel\_f2534 <int>, newrel\_f3544 <int>,  
## # newrel\_f4554 <int>, newrel\_f5564 <int>, newrel\_f65 <int>  
```

Note that country, iso2, iso3, and year are already variables, so they can be left as is. But the columns from new\_sp\_m014 to newrel\_f65 encode four variables in their names:

1. The new\_/new prefix indicates these are counts of new cases. This dataset only contains new cases, so we’ll ignore it here because it’s constant.
2. The next two letters describe the type of TB:
   * rel stands for cases of relapse
   * ep stands for cases of extrapulmonary TB
   * sn stands for cases of pulmonary TB that could not be diagnosed by a pulmonary smear (smear negative)
   * sp stands for cases of pulmonary TB that could be diagnosed be a pulmonary smear (smear positive)
3. The sixth letter gives the sex of TB patients. The dataset groups cases by males (m) and females (f).
4. The remaining numbers gives the age group. The dataset groups cases into seven age groups:
   * 014 = 0 – 14 years old
   * 1524 = 15 – 24 years old
   * 2534 = 25 – 34 years old
   * 3544 = 35 – 44 years old
   * 4554 = 45 – 54 years old
   * 5564 = 55 – 64 years old
   * 65 = 65 or older

We can break these variables up by specifying multiple column names in names\_to, and then either providing names\_sep or names\_pattern. Here names\_pattern is the most natural fit. It has a similar interface to extract: you give it a regular expression containing groups (defined by ()) and it puts each group in a column.

```r  
# separate using regular expression  
who\_tidy <- who %>% pivot\_longer(  
 cols = new\_sp\_m014:newrel\_f65,  
 names\_to = c("diagnosis", "gender", "age"),  
 names\_pattern = "new\_?(.\*)\_(.)(.\*)", # or use names\_sep for separator  
 values\_to = "count"  
)  
head(who\_tidy)  
```  
  
```  
## # A tibble: 6 x 8  
## country iso2 iso3 year diagnosis gender age count  
## <chr> <chr> <chr> <int> <chr> <chr> <chr> <int>  
## 1 Afghanistan AF AFG 1980 sp m 014 NA  
## 2 Afghanistan AF AFG 1980 sp m 1524 NA  
## 3 Afghanistan AF AFG 1980 sp m 2534 NA  
## 4 Afghanistan AF AFG 1980 sp m 3544 NA  
## 5 Afghanistan AF AFG 1980 sp m 4554 NA  
## 6 Afghanistan AF AFG 1980 sp m 5564 NA  
```

We could go one step further and specify the types of the gender and age columns. This is good practice when you have categorical variables with a known set of values. Let’s also get rid of all the NA counts.

```r  
# separate using regular expression and add types for gender and ordered type for age  
who\_tidy <- who %>% pivot\_longer(  
 cols = new\_sp\_m014:newrel\_f65,  
 names\_to = c("diagnosis", "gender", "age"),  
 names\_pattern = "new\_?(.\*)\_(.)(.\*)", # or use names\_sep for separator  
 values\_to = "count",  
 values\_drop\_na = TRUE,  
 names\_ptypes = list(  
 gender = factor(levels = c("f", "m")),  
 age = factor(  
 levels = c("014", "1524", "2534", "3544", "4554", "5564", "65"),   
 ordered = TRUE)  
 )  
)  
head(who\_tidy)  
```  
  
```  
## # A tibble: 6 x 8  
## country iso2 iso3 year diagnosis gender age count  
## <chr> <chr> <chr> <int> <chr> <fct> <ord> <int>  
## 1 Afghanistan AF AFG 1997 sp m 014 0  
## 2 Afghanistan AF AFG 1997 sp m 1524 10  
## 3 Afghanistan AF AFG 1997 sp m 2534 6  
## 4 Afghanistan AF AFG 1997 sp m 3544 3  
## 5 Afghanistan AF AFG 1997 sp m 4554 5  
## 6 Afghanistan AF AFG 1997 sp m 5564 2  
```

# Variables stored in both rows and columns

Get the data and look at it

```r  
weather <- read\_csv("./Data/weather.csv")  
```  
  
```  
## Parsed with column specification:  
## cols(  
## .default = col\_double(),  
## id = col\_character(),  
## element = col\_character(),  
## d9 = col\_logical(),  
## d12 = col\_logical(),  
## d18 = col\_logical(),  
## d19 = col\_logical(),  
## d20 = col\_logical(),  
## d21 = col\_logical(),  
## d22 = col\_logical(),  
## d24 = col\_logical()  
## )  
```  
  
```  
## See spec(...) for full column specifications.  
```  
  
```r  
head(weather)  
```  
  
```  
## # A tibble: 6 x 35  
## id year month element d1 d2 d3 d4 d5 d6 d7 d8  
## <chr> <dbl> <dbl> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 MX17~ 2010 1 tmax NA NA NA NA NA NA NA NA  
## 2 MX17~ 2010 1 tmin NA NA NA NA NA NA NA NA  
## 3 MX17~ 2010 2 tmax NA 27.3 24.1 NA NA NA NA NA  
## 4 MX17~ 2010 2 tmin NA 14.4 14.4 NA NA NA NA NA  
## 5 MX17~ 2010 3 tmax NA NA NA NA 32.1 NA NA NA  
## 6 MX17~ 2010 3 tmin NA NA NA NA 14.2 NA NA NA  
## # ... with 23 more variables: d9 <lgl>, d10 <dbl>, d11 <dbl>, d12 <lgl>,  
## # d13 <dbl>, d14 <dbl>, d15 <dbl>, d16 <dbl>, d17 <dbl>, d18 <lgl>,  
## # d19 <lgl>, d20 <lgl>, d21 <lgl>, d22 <lgl>, d23 <dbl>, d24 <lgl>,  
## # d25 <dbl>, d26 <dbl>, d27 <dbl>, d28 <dbl>, d29 <dbl>, d30 <dbl>, d31 <dbl>  
```

* daily weather data from the Global Historical Climatology Network
* one weather station (MX17004) in Mexico for five months in 2010.
* variables in columns for id, year and month, spread across columns for days of month
* variables spread across rows for minimum and maximum temperatures
* months with fewer than 31 days have structural missing values

Use pivot\_longer() to put the days all in one column

```r  
weather2 <- weather %>% pivot\_longer(  
 cols = starts\_with("d"),  
 names\_to = "day",  
 values\_to = "value",  
 names\_prefix = "d", # removes the prefix specified  
 names\_ptypes = list(day = integer()), #sets the value left to be an integer  
 values\_drop\_na = TRUE  
)  
```

Rearrange the data

```r  
weather3 <- weather2 %>%  
 select(id,year, month, day, element, value) %>%  
 arrange(id, year, month, day)  
weather3  
```  
  
```  
## # A tibble: 66 x 6  
## id year month day element value  
## <chr> <dbl> <dbl> <int> <chr> <dbl>  
## 1 MX17004 2010 1 30 tmax 27.8  
## 2 MX17004 2010 1 30 tmin 14.5  
## 3 MX17004 2010 2 2 tmax 27.3  
## 4 MX17004 2010 2 2 tmin 14.4  
## 5 MX17004 2010 2 3 tmax 24.1  
## 6 MX17004 2010 2 3 tmin 14.4  
## 7 MX17004 2010 2 11 tmax 29.7  
## 8 MX17004 2010 2 11 tmin 13.4  
## 9 MX17004 2010 2 23 tmax 29.9  
## 10 MX17004 2010 2 23 tmin 10.7  
## # ... with 56 more rows  
```

Use pivot\_wider to separate tmax and tmin into separate columns

```r  
weather4 <- weather3 %>% pivot\_wider(  
 names\_from = element,  
 values\_from = value  
)  
head(weather4)  
```  
  
```  
## # A tibble: 6 x 6  
## id year month day tmax tmin  
## <chr> <dbl> <dbl> <int> <dbl> <dbl>  
## 1 MX17004 2010 1 30 27.8 14.5  
## 2 MX17004 2010 2 2 27.3 14.4  
## 3 MX17004 2010 2 3 24.1 14.4  
## 4 MX17004 2010 2 11 29.7 13.4  
## 5 MX17004 2010 2 23 29.9 10.7  
## 6 MX17004 2010 3 5 32.1 14.2  
```

Look at resulting dataframe

```r  
summary(weather4)  
```  
  
```  
## id year month day   
## Length:33 Min. :2010 Min. : 1.000 Min. : 1.00   
## Class :character 1st Qu.:2010 1st Qu.: 4.000 1st Qu.: 5.00   
## Mode :character Median :2010 Median : 8.000 Median :14.00   
## Mean :2010 Mean : 7.212 Mean :14.88   
## 3rd Qu.:2010 3rd Qu.:10.000 3rd Qu.:26.00   
## Max. :2010 Max. :12.000 Max. :31.00   
## tmax tmin   
## Min. :24.10 Min. : 7.90   
## 1st Qu.:27.80 1st Qu.:13.40   
## Median :29.00 Median :15.00   
## Mean :29.19 Mean :14.65   
## 3rd Qu.:29.90 3rd Qu.:16.50   
## Max. :36.30 Max. :18.20  
```

Class exercise or homework:

Tidy the billboard dataset (built-in)

* First gather up all the week entries into a row for each week for each song (where there is an entry)
* Then, convert the week variable to a number and figure out the date corresponding to each week on the chart
* Sort the data by artist, track and week. Here are what your first entries should be (formatting can be different):

#> A tibble: 5,307 x 5  
# artist track date.entered week rank date  
 # <chr> <chr> <date> <int> <dbl> <date>  
 # 1 2 Pac Baby Don't Cry (Keep... 2000-02-26 1 87 2000-02-26  
 # 2 2 Pac Baby Don't Cry (Keep... 2000-02-26 2 82 2000-03-04  
 # 3 2 Pac Baby Don't Cry (Keep... 2000-02-26 3 72 2000-03-11  
 # 4 2 Pac Baby Don't Cry (Keep... 2000-02-26 4 77 2000-03-18  
 # 5 2 Pac Baby Don't Cry (Keep... 2000-02-26 5 87 2000-03-25  
 # 6 2 Pac Baby Don't Cry (Keep... 2000-02-26 6 94 2000-04-01  
 # 7 2 Pac Baby Don't Cry (Keep... 2000-02-26 7 99 2000-04-08  
 # 8 2Ge+her The Hardest Part Of ... 2000-09-02 1 91 2000-09-02  
 # 9 2Ge+her The Hardest Part Of ... 2000-09-02 2 87 2000-09-09  
 # 10 2Ge+her The Hardest Part Of ... 2000-09-02 3 92 2000-09-16  
 # … with 5,297 more rows

## # A tibble: 5,307 x 5  
## artist track week rank date   
## <chr> <chr> <int> <dbl> <date>   
## 1 2 Pac Baby Don't Cry (Keep... 1 87 2000-02-26  
## 2 2 Pac Baby Don't Cry (Keep... 2 82 2000-03-04  
## 3 2 Pac Baby Don't Cry (Keep... 3 72 2000-03-11  
## 4 2 Pac Baby Don't Cry (Keep... 4 77 2000-03-18  
## 5 2 Pac Baby Don't Cry (Keep... 5 87 2000-03-25  
## 6 2 Pac Baby Don't Cry (Keep... 6 94 2000-04-01  
## 7 2 Pac Baby Don't Cry (Keep... 7 99 2000-04-08  
## 8 2Ge+her The Hardest Part Of ... 1 91 2000-09-02  
## 9 2Ge+her The Hardest Part Of ... 2 87 2000-09-09  
## 10 2Ge+her The Hardest Part Of ... 3 92 2000-09-16  
## # ... with 5,297 more rows

## Multiple observations per row

You can usually recognise this case because the name of the column that you want to appear in the output is part of the column name in the input.

family <- tribble(  
 ~family, ~dob\_child1, ~dob\_child2, ~gender\_child1, ~gender\_child2,  
 1L, "1998-11-26", "2000-01-29", 1L, 2L,  
 2L, "1996-06-22", NA, 2L, NA,  
 3L, "2002-07-11", "2004-04-05", 2L, 2L,  
 4L, "2004-10-10", "2009-08-27", 1L, 1L,  
 5L, "2000-12-05", "2005-02-28", 2L, 1L,  
)  
family <- family %>% mutate\_at(vars(starts\_with("dob")), parse\_date)  
family

## # A tibble: 5 x 5  
## family dob\_child1 dob\_child2 gender\_child1 gender\_child2  
## <int> <date> <date> <int> <int>  
## 1 1 1998-11-26 2000-01-29 1 2  
## 2 2 1996-06-22 NA 2 NA  
## 3 3 2002-07-11 2004-04-05 2 2  
## 4 4 2004-10-10 2009-08-27 1 1  
## 5 5 2000-12-05 2005-02-28 2 1

Note that we have two pieces of information (or values) for each child: their gender and their dob (date of birth). These need to go into separate columns in the result. Again we supply multiple variables to names\_to, using names\_sep to split up each variable name. Note the special name .value: this tells pivot\_longer() that that part of the column name specifies the “value” being measured (which will become a variable in the output).

family %>%   
 pivot\_longer(  
 -family,   
 names\_to = c(".value", "child"),   
 names\_sep = "\_",   
 values\_drop\_na = TRUE  
 )

## # A tibble: 9 x 4  
## family child dob gender  
## <int> <chr> <date> <int>  
## 1 1 child1 1998-11-26 1  
## 2 1 child2 2000-01-29 2  
## 3 2 child1 1996-06-22 2  
## 4 3 child1 2002-07-11 2  
## 5 3 child2 2004-04-05 2  
## 6 4 child1 2004-10-10 1  
## 7 4 child2 2009-08-27 1  
## 8 5 child1 2000-12-05 2  
## 9 5 child2 2005-02-28 1

Class exercise or homework: Do the same with the built-in dataset anscombe

## x1 x2 x3 x4 y1 y2 y3 y4  
## 1 10 10 10 8 8.04 9.14 7.46 6.58  
## 2 8 8 8 8 6.95 8.14 6.77 5.76  
## 3 13 13 13 8 7.58 8.74 12.74 7.71  
## 4 9 9 9 8 8.81 8.77 7.11 8.84  
## 5 11 11 11 8 8.33 9.26 7.81 8.47  
## 6 14 14 14 8 9.96 8.10 8.84 7.04  
## 7 6 6 6 8 7.24 6.13 6.08 5.25  
## 8 4 4 4 19 4.26 3.10 5.39 12.50  
## 9 12 12 12 8 10.84 9.13 8.15 5.56  
## 10 7 7 7 8 4.82 7.26 6.42 7.91  
## 11 5 5 5 8 5.68 4.74 5.73 6.89

## # A tibble: 44 x 3  
## set x y  
## <chr> <dbl> <dbl>  
## 1 1 10 8.04  
## 2 1 8 6.95  
## 3 1 13 7.58  
## 4 1 9 8.81  
## 5 1 11 8.33  
## 6 1 14 9.96  
## 7 1 6 7.24  
## 8 1 4 4.26  
## 9 1 12 10.8   
## 10 1 7 4.82  
## # ... with 34 more rows

Class exercise or homework: Tidy the world\_bank\_pop dataset

## # A tibble: 1,056 x 20  
## country indicator `2000` `2001` `2002` `2003` `2004` `2005` `2006`  
## <chr> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 ABW SP.URB.T~ 4.24e4 4.30e4 4.37e4 4.42e4 4.47e+4 4.49e+4 4.49e+4  
## 2 ABW SP.URB.G~ 1.18e0 1.41e0 1.43e0 1.31e0 9.51e-1 4.91e-1 -1.78e-2  
## 3 ABW SP.POP.T~ 9.09e4 9.29e4 9.50e4 9.70e4 9.87e+4 1.00e+5 1.01e+5  
## 4 ABW SP.POP.G~ 2.06e0 2.23e0 2.23e0 2.11e0 1.76e+0 1.30e+0 7.98e-1  
## 5 AFG SP.URB.T~ 4.44e6 4.65e6 4.89e6 5.16e6 5.43e+6 5.69e+6 5.93e+6  
## 6 AFG SP.URB.G~ 3.91e0 4.66e0 5.13e0 5.23e0 5.12e+0 4.77e+0 4.12e+0  
## 7 AFG SP.POP.T~ 2.01e7 2.10e7 2.20e7 2.31e7 2.41e+7 2.51e+7 2.59e+7  
## 8 AFG SP.POP.G~ 3.49e0 4.25e0 4.72e0 4.82e0 4.47e+0 3.87e+0 3.23e+0  
## 9 AGO SP.URB.T~ 8.23e6 8.71e6 9.22e6 9.77e6 1.03e+7 1.09e+7 1.15e+7  
## 10 AGO SP.URB.G~ 5.44e0 5.59e0 5.70e0 5.76e0 5.75e+0 5.69e+0 4.92e+0  
## # ... with 1,046 more rows, and 11 more variables: `2007` <dbl>, `2008` <dbl>,  
## # `2009` <dbl>, `2010` <dbl>, `2011` <dbl>, `2012` <dbl>, `2013` <dbl>,  
## # `2014` <dbl>, `2015` <dbl>, `2016` <dbl>, `2017` <dbl>

## # A tibble: 19,008 x 4  
## country indicator year value  
## <chr> <chr> <chr> <dbl>  
## 1 ABW SP.URB.TOTL 2000 42444  
## 2 ABW SP.URB.TOTL 2001 43048  
## 3 ABW SP.URB.TOTL 2002 43670  
## 4 ABW SP.URB.TOTL 2003 44246  
## 5 ABW SP.URB.TOTL 2004 44669  
## 6 ABW SP.URB.TOTL 2005 44889  
## 7 ABW SP.URB.TOTL 2006 44881  
## 8 ABW SP.URB.TOTL 2007 44686  
## 9 ABW SP.URB.TOTL 2008 44375  
## 10 ABW SP.URB.TOTL 2009 44052  
## # ... with 18,998 more rows

## # A tibble: 4 x 2  
## indicator n  
## \* <chr> <int>  
## 1 SP.POP.GROW 4752  
## 2 SP.POP.TOTL 4752  
## 3 SP.URB.GROW 4752  
## 4 SP.URB.TOTL 4752

## # A tibble: 19,008 x 5  
## country area variable year value  
## <chr> <chr> <chr> <chr> <dbl>  
## 1 ABW URB TOTL 2000 42444  
## 2 ABW URB TOTL 2001 43048  
## 3 ABW URB TOTL 2002 43670  
## 4 ABW URB TOTL 2003 44246  
## 5 ABW URB TOTL 2004 44669  
## 6 ABW URB TOTL 2005 44889  
## 7 ABW URB TOTL 2006 44881  
## 8 ABW URB TOTL 2007 44686  
## 9 ABW URB TOTL 2008 44375  
## 10 ABW URB TOTL 2009 44052  
## # ... with 18,998 more rows

## # A tibble: 9,504 x 5  
## country area year TOTL GROW  
## <chr> <chr> <chr> <dbl> <dbl>  
## 1 ABW URB 2000 42444 1.18   
## 2 ABW URB 2001 43048 1.41   
## 3 ABW URB 2002 43670 1.43   
## 4 ABW URB 2003 44246 1.31   
## 5 ABW URB 2004 44669 0.951   
## 6 ABW URB 2005 44889 0.491   
## 7 ABW URB 2006 44881 -0.0178  
## 8 ABW URB 2007 44686 -0.435   
## 9 ABW URB 2008 44375 -0.698   
## 10 ABW URB 2009 44052 -0.731   
## # ... with 9,494 more rows

### Specification by hand

Look at the construction dataset. We would like to have columns for units and one for region.

spec <- tribble(  
 ~.name, ~.value, ~units, ~region,   
 "1 unit", "n", "1", NA,   
 "2 to 4 units", "n", "2-4", NA,   
 "5 units or more", "n", "5+", NA,   
 "Northeast", "n", NA, "Northeast",   
 "Midwest", "n", NA, "Midwest",   
 "South", "n", NA, "South",   
 "West", "n", NA, "West",   
)  
pivot\_longer\_spec(construction, spec)

## # A tibble: 63 x 5  
## Year Month units region n  
## <dbl> <chr> <chr> <chr> <dbl>  
## 1 2018 January 1 <NA> 859  
## 2 2018 January 2-4 <NA> NA  
## 3 2018 January 5+ <NA> 348  
## 4 2018 January <NA> Northeast 114  
## 5 2018 January <NA> Midwest 169  
## 6 2018 January <NA> South 596  
## 7 2018 January <NA> West 339  
## 8 2018 February 1 <NA> 882  
## 9 2018 February 2-4 <NA> NA  
## 10 2018 February 5+ <NA> 400  
## # ... with 53 more rows

The pivotting spec allows us to be more precise about exactly how pivot\_longer(df, spec = spec) changes the shape of df: it will have nrow(df) \* nrow(spec) rows, and ncol(df) - nrow(spec) + ncol(spec) - 2 columns.