Tidy Data

Jarred Robidoux

2023-02-18

Tidy Data

Introduction

In this cahpter, you will learn a consistent way to organise your data in R, and organization called **tidy data**. Getting you data into this format requires some upfront work, but that work pays off in the long term.

Prerequisites

In this chapter we'll focus on tidyr, a package that provides a bunch of tools to help tidy up your messy datasets. tidyr is a member of the core tidyverse

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.4.0     v purrr   1.0.1
## v tibble  3.1.8     v dplyr   1.1.0
## v tidyr   1.3.0     v stringr 1.5.0
## v readr   2.1.3     v forcats 1.0.0
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Tidy Data

You can represent the same underlying data in multiple ways. The example below shows the same data organized in four different ways. Each dataset shows the same values for four variables *country*, *year*, *population*, and *cases*, but each dataset organizes the values in a different way. Table 1

```
table1
```

```
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>       <dbl>  <dbl>      <dbl>
## 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
```

Table 2

```
table2
```

```
## # A tibble: 12 x 4
##    country      year type            count
##    <chr>       <dbl> <chr>           <dbl>
##  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
```

Table 3

```
table3
```

```
## # A tibble: 6 x 3
##   country      year rate
##   <chr>       <dbl> <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
```

Table 4a

```
table4a
```

```
## # A tibble: 3 x 3
##   country     `1999` `2000`
##   <chr>        <dbl>  <dbl>
## 1 Afghanistan    745   2666
## 2 Brazil       37737  80488
## 3 China       212258 213766
```

Table 4b

```
table4b
```

```
## # A tibble: 3 x 3
##   country         `1999`     `2000`
##   <chr>            <dbl>      <dbl>
## 1 Afghanistan   19987071   20595360
## 2 Brazil       172006362  174504898
## 3 China       1272915272 1280428583
```

There are three interrelated rules which make a dataset tidy: 1. Each variable must have its own column 2. Each observation must have its own row 3. Each value must have its own cell

These rules are interrelated because it's impossible to only satisfy two of the three. That interrelationship leads to an even simpler set of practical instructions: 1. Put each dataset in a tibble 2. Put each variable in a column

Exercises

1. Compute the *rate* for table2.

```
table2_cases <- table2 %>%
  filter(type == "cases")

table2_population <- table2 %>%
  filter(type == "population")

table2_mod <- tibble(
  country = table2_cases$country,
  year = table2_cases$year,
  cases = table2_cases$count,
  population = table2_population$count
)

table1 == table2_mod
```

```
##      country year cases population
## [1,]    TRUE TRUE  TRUE       TRUE
## [2,]    TRUE TRUE  TRUE       TRUE
## [3,]    TRUE TRUE  TRUE       TRUE
## [4,]    TRUE TRUE  TRUE       TRUE
## [5,]    TRUE TRUE  TRUE       TRUE
## [6,]    TRUE TRUE  TRUE       TRUE
```

```
table2_mod %>%
  mutate(rate = (cases / population) * 10000)
```

```
## # A tibble: 6 x 5
##   country      year  cases population  rate
##   <chr>       <dbl>  <dbl>      <dbl> <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
```

Pivoting

For most real analyses, you'll need to do some tidying.

1. One variable might be spread across multiple columns
2. One observation might be scattered across multiple rows

To fix these problems, you'll need the two most important functions in tidyr: *pivot_longer()* and *pivot_wider()*

pivot_longer()

To tidy a dataset like this, we need to *pivot* the offending columns into a new pair of variables. To describe that operation we need three parameters.

- The set of columns whose names are values, not variables. In this example, those are the columns 1999 and 2000.
- The name of the variable to move the column names to. Here it is *year*
- The name of the variable to move the column values to. Here it is *cases*

```
table4a %>%
  pivot_longer(c(`1999`, `2000`), names_to = "year", values_to = "cases")
```

```
## # A tibble: 6 x 3
##   country     year   cases
##   <chr>       <chr>  <dbl>
## 1 Afghanistan 1999     745
## 2 Afghanistan 2000    2666
## 3 Brazil      1999   37737
## 4 Brazil      2000   80488
## 5 China       1999  212258
## 6 China       2000  213766
```

pivot_longer() makes data sets longer by increasing the number of rows and decreasing the number of columns.

Tidy table 4b

```
table4b %>%
  pivot_longer(c(`1999`, `2000`), names_to = "year", values_to = "population")
```

```
## # A tibble: 6 x 3
##   country     year  population
##   <chr>       <chr>      <dbl>
```

```
## 1 Afghanistan 1999    19987071
## 2 Afghanistan 2000    20595360
## 3 Brazil      1999   172006362
## 4 Brazil      2000   174504898
## 5 China       1999  1272915272
## 6 China       2000  1280428583
```

To combine the tidied versions of table4a and table4b into a single tibble, we need to use *left_join*, which you'll learn about later

```
tidy4a <- table4a %>%
  pivot_longer(c(`1999`, `2000`), names_to = "year", values_to = "cases")

tidy4b <- table4b %>%
  pivot_longer(c(`1999`, `2000`), names_to = "year", values_to = "population")

left_join(tidy4a, tidy4b)
```

```
## Joining with `by = join_by(country, year)`
```

```
## # A tibble: 6 x 4
##   country     year   cases population
##   <chr>       <chr>  <dbl>      <dbl>
## 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
```

Wider

pivot_wider() is the opposite of *pivot_longer()*. You use it when an observation is scattered across multiple rows. For example, take *table2*: an observation is a country in a year, but each observation is spread across two rows.

```
table2 %>%
  pivot_wider(names_from = type, values_from = count)
```

```
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>       <dbl>  <dbl>      <dbl>
## 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
```

Exercises

1. Why does this code fail?

```
table4a %>%
  pivot_longer(c(`1999`, `2000`), names_to="year", values_to = "cases")
```

```
## # A tibble: 6 x 3
##   country     year   cases
##   <chr>       <chr>  <dbl>
## 1 Afghanistan 1999     745
## 2 Afghanistan 2000    2666
## 3 Brazil      1999   37737
## 4 Brazil      2000   80488
## 5 China       1999  212258
## 6 China       2000  213766
```

2. What would happen if you widen this table? Why? How could you add a new column to uniquely identify each value?

```
people <- tribble(
  ~name, ~names, ~values,
  "Phillip Woods", "age", 45,
  "Phillip Woods", "height", 186,
  "Phillip Woods", "age", 50,
  "Jessoca Cordero", "age", 37,
  "Jessica Cordero", "height", 156
)
```

3. Tidy the simple tibble below.

```
preg <- tribble(
  ~pregnant, ~male, ~female,
  "yes", NA, 10,
  "no", 20, 12
)

preg %>%
  pivot_longer(c(male, female), names_to = "sex", values_to = "count")
```

```
## # A tibble: 4 x 3
##   pregnant sex    count
##   <chr>    <chr>  <dbl>
## 1 yes      male      NA
## 2 yes      female    10
## 3 no       male      20
## 4 no       female    12
```

Separating and Uniting

So far you've learned how to tidy table2 and table4, but not table3. table3 has a different problem: we have one column (rate) that contains two variables (cases and population).

```
table3
```

```
## # A tibble: 6 x 3
##   country      year rate
##   <chr>       <dbl> <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
```

To fix this problem, we'll need the separate() function.

Separate

separate() pulls apart one column into multiple columns, by splitting wherever a separator character appears.

```
table3 %>%
  separate(rate, into = c("cases", "population"))
```

```
## # A tibble: 6 x 4
##   country      year cases  population
##   <chr>       <dbl> <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
```

Notice the column types: you'll see that cases and population are character columns.

```
table3 %>%
  separate(rate, into = c("cases", "population"), convert = TRUE)
```

```
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>       <dbl>  <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
```

Unite

unite() is the inverse of *separate()*: it combines multiple columns into a single column.

```
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
```

In this case we also need to use the *sep* argument. The default will place an underscore (_) between the values from different columns

```
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
```

Excerises

1. What do the *extra* and *fill* arguments do in *separate()*? Experiment with the various options for the following two toy datasets.

```
tibble(x=c("a,b,c", "d,e,f,g", "h,i,j")) %>%
  separate(x, c("one", "two", "three"), extra = "warn", fill = "left")
```

```
## Warning: Expected 3 pieces. Additional pieces discarded in 1 rows [2].
```

```
## # A tibble: 3 x 3
##   one   two   three
##   <chr> <chr> <chr>
## 1 a     b     c
## 2 d     e     f
## 3 h     i     j
```

```
?separate
```

```
## starting httpd help server ... done
```

The *extra* and *fill* arguments control what happens to the warning message and missing values when there are too many/not enough pieces to register into columns using *separate()*

2. Both *unite()* and *separate()* have a remove argument. What does it do? Why would you set it to false?

```
table5 %>%
  unite(new, century, year, remove = TRUE)
```

```
## # 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
```

By setting *remove = TRUE* in both *unite()* and *separate()*, you will be excluding the input columns from the output. This looks much cleaner when you do this, hence why *remove = TRUE* is the default.

Case Study

To finish off the chapter, let's pull together everything you've learned to tackle a realistic data tidying problem. The tidyr::who data set contains tuberculosis (TB) cases broken down by year, country, age, gender, and diagnosis method.

```
who
```

```
## # A tibble: 7,240 x 60
##    country     iso2  iso3   year new_s~1 new_s~2 new_s~3 new_s~4 new_s~5 new_s~6
##    <chr>       <chr> <chr> <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
##  1 Afghanistan AF    AFG    1980      NA      NA      NA      NA      NA      NA
##  2 Afghanistan AF    AFG    1981      NA      NA      NA      NA      NA      NA
##  3 Afghanistan AF    AFG    1982      NA      NA      NA      NA      NA      NA
##  4 Afghanistan AF    AFG    1983      NA      NA      NA      NA      NA      NA
##  5 Afghanistan AF    AFG    1984      NA      NA      NA      NA      NA      NA
##  6 Afghanistan AF    AFG    1985      NA      NA      NA      NA      NA      NA
##  7 Afghanistan AF    AFG    1986      NA      NA      NA      NA      NA      NA
##  8 Afghanistan AF    AFG    1987      NA      NA      NA      NA      NA      NA
##  9 Afghanistan AF    AFG    1988      NA      NA      NA      NA      NA      NA
## 10 Afghanistan AF    AFG    1989      NA      NA      NA      NA      NA      NA
## # ... with 7,230 more rows, 50 more variables: new_sp_m65 <dbl>,
## #   new_sp_f014 <dbl>, new_sp_f1524 <dbl>, new_sp_f2534 <dbl>,
## #   new_sp_f3544 <dbl>, new_sp_f4554 <dbl>, new_sp_f5564 <dbl>,
## #   new_sp_f65 <dbl>, new_sn_m014 <dbl>, new_sn_m1524 <dbl>,
## #   new_sn_m2534 <dbl>, new_sn_m3544 <dbl>, new_sn_m4554 <dbl>,
## #   new_sn_m5564 <dbl>, new_sn_m65 <dbl>, new_sn_f014 <dbl>,
## #   new_sn_f1524 <dbl>, new_sn_f2534 <dbl>, new_sn_f3544 <dbl>, ...
```

```r
who1 <- who %>%
  pivot_longer(
    cols = new_sp_f014:newrel_f65,
    names_to = "key",
    values_to = "cases",
    values_drop_na = TRUE
  )

who2 <- who1 %>%
  mutate(key = stringr::str_replace(key, "newrel", "new_rel"))

who3 <- who2 %>%
  separate(key, c("new", "type", "sexage"), sep = "_")

who4 <- who3 %>%
  select(-new, -iso2, -iso3)
```