L07 Tidy Data

Data Science I (STAT 301-1)

Contents

Overview

The goal of this lab is to learn what it means to be a "tidy" dataset and how to tidy messy datasets utilizing the `tidyr` package — a core member of the `tidyverse`. `tidyr` package home page

Datasets

All datasets are either defined inline or provided within the core `tidyverse` packages (`table1`, `table2`, `table4a`, `table4a`, `who`)

Exercises

Please complete the following exercises. Be sure your solutions are clearly indicated and that the document is neatly formatted.

```
library(tidyverse)
```

Load Packages

Exercise 1 (Website: 12.2.1 Ex. 2 — modified)

Follow these four steps to compute the `rate` per 10,000 once using only `table2` and again using `table4a + table4b`:

1. Extract the number of TB cases per country per year.
2. Extract the matching population numbers per country per year.
3. Divide case numbers by population numbers and multiply by 10000.
4. Store in a new `tibble`.

How does this process compare to using `table1`, which is a tidy dataset, to compute `rate` per 10,000?

```
#first using table2
case_per_year <- table2 %>%
  filter(type == 'cases') %>%
```

```r
  group_by(year, country) %>%
  summarise(total_cases = sum(count))

pop_per_year <- table2 %>%
  filter(type == 'population') %>%
  group_by(year, country) %>%
  summarise(total_pop = sum(count))

rate <- full_join(case_per_year, pop_per_year)
rate %>%
  mutate(rate = 10000*total_cases/total_pop)
```

```
## # A tibble: 6 x 5
## # Groups:   year [2]
##    year country     total_cases  total_pop  rate
##   <int> <chr>             <int>      <int> <dbl>
## 1  1999 Afghanistan         745   19987071 0.373
## 2  1999 Brazil            37737  172006362 2.19
## 3  1999 China            212258 1272915272 1.67
## 4  2000 Afghanistan        2666   20595360 1.29
## 5  2000 Brazil            80488  174504898 4.61
## 6  2000 China            213766 1280428583 1.67
```

Next using `table4a` and `table4b`

```r
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")

rate2 <- left_join(tidy4a, tidy4b)
```

```
## Joining, by = c("country", "year")
```

```r
rate2 %>%
  mutate(rate = 10000*cases/population)
```

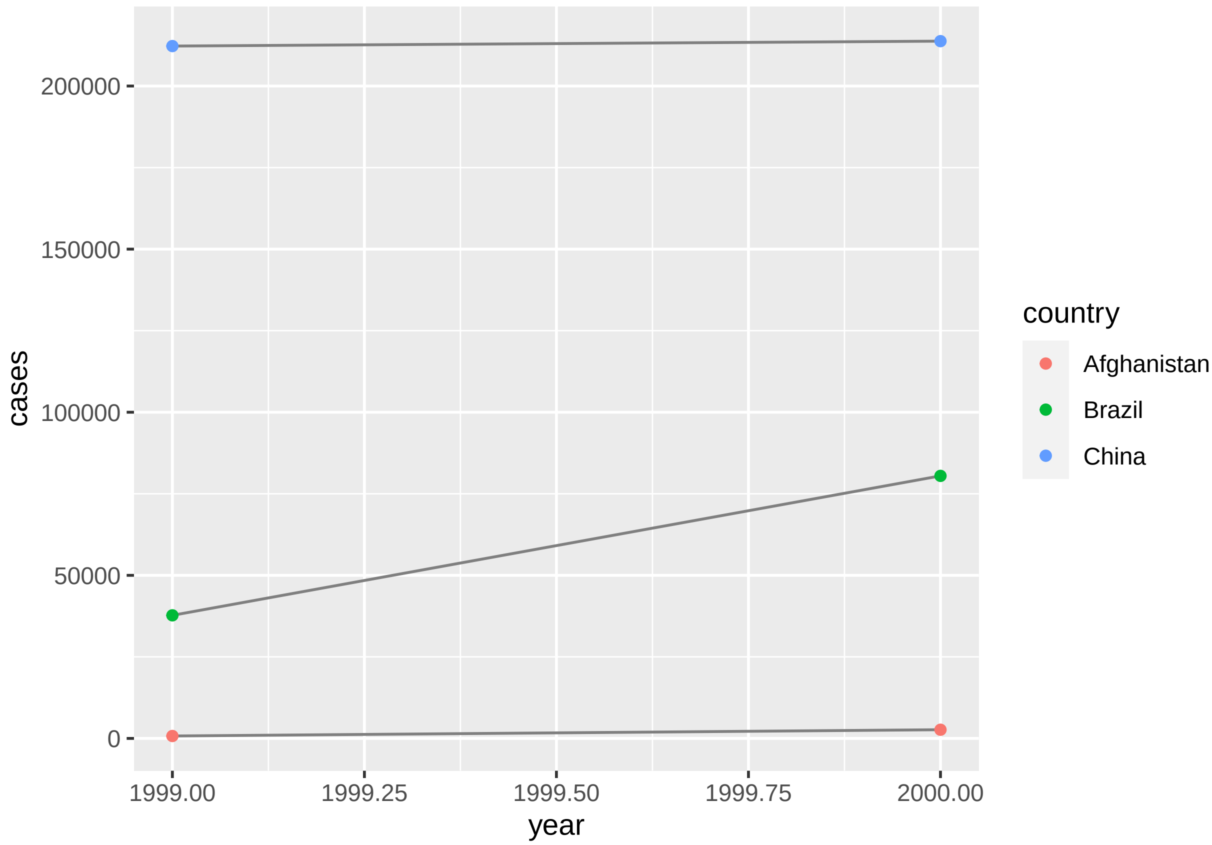
```
## # A tibble: 6 x 5
##   country     year   cases population  rate
##   <chr>       <chr>  <int>      <int> <dbl>
## 1 Afghanistan 1999     745   19987071 0.373
## 2 Afghanistan 2000    2666   20595360 1.29
## 3 Brazil      1999   37737  172006362 2.19
## 4 Brazil      2000   80488  174504898 4.61
## 5 China       1999  212258 1272915272 1.67
## 6 China       2000  213766 1280428583 1.67
```

Using `table1` would be a lot easier, as you wouldn't have to filter by type.

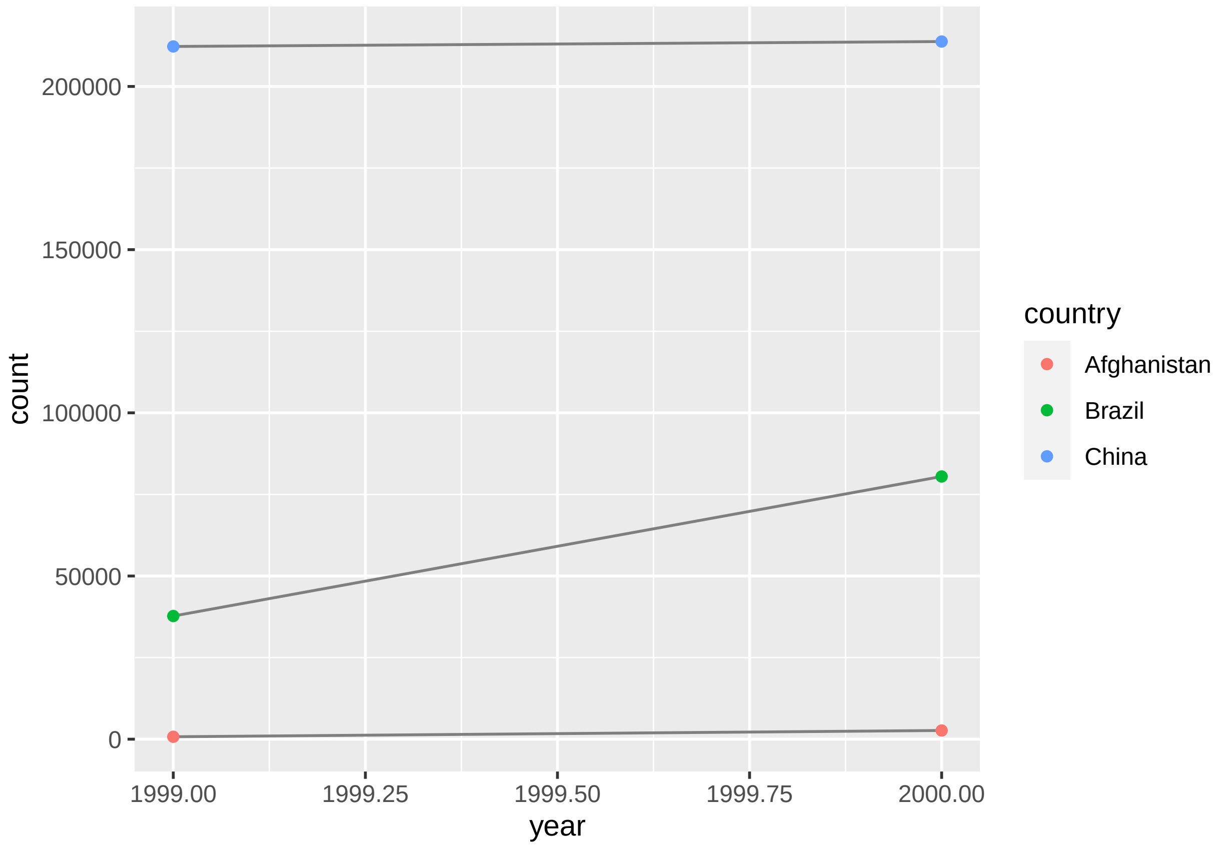
Exercise 2 (Website: 12.2.1 Ex. 3)

Recreate the plot below showing the change in cases over time using `table2` instead of `table1`. What do you need to do first?

```r
# Change over time in number of TB cases by country
ggplot(table1, aes(year, cases)) +
  geom_line(aes(group = country), colour = "grey50") +
  geom_point(aes(colour = country))
```




```r
table2 %>%
  filter(type == 'cases') %>%
  ggplot(aes(year, count)) +
  geom_line(aes(group = country), colour = 'grey50') +
  geom_point(aes(colour = country))
```



First you need to filter so that only the cases are plotted, not the populations.

Exercise 3 (Website: 12.3.3 Ex. 2)

Why does the provided code fail? Fix it.

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

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

You need backticks on the 1999 and 2000, or else R doesn't recognize them as variable names.

Exercise 4 (Website: 12.3.3 Ex. 3)

What would happen if you use pivot_wider() on this table? Add a new column to fix the problem.

```
people <- tribble(
  ~name,             ~key,    ~value,
  #-----------------|--------|------
```

```r
  "Phillip Woods",   "age",       45,
  "Phillip Woods",   "height",   186,
  "Phillip Woods",   "age",       50,
  "Jessica Cordero", "age",       37,
  "Jessica Cordero", "height",   156
)

people %>%
  pivot_wider(names_from = key, values_from = value)
```

```
## Warning: Values are not uniquely identified; output will contain list-cols.
## * Use `values_fn = list` to suppress this warning.
## * Use `values_fn = length` to identify where the duplicates arise
## * Use `values_fn = {summary_fun}` to summarise duplicates
```

```
## # A tibble: 2 x 3
##   name            age       height
##   <chr>           <list>    <list>
## 1 Phillip Woods   <dbl [2]> <dbl [1]>
## 2 Jessica Cordero <dbl [1]> <dbl [1]>
```

I believe it is not working because Phillip Woods has two values for `age`, and R doesn't know where to put the second value. I will make a variable called `obs` that accounts for the two ages of Phillip Woods.

```r
people2 <- people %>%
  group_by(name, key) %>%
  mutate(obs = row_number())

people2 %>%
  pivot_wider(names_from = "name", values_from = "value")
```

```
## # A tibble: 3 x 4
## # Groups:   key [2]
##   key      obs `Phillip Woods` `Jessica Cordero`
##   <chr>  <int>           <dbl>             <dbl>
## 1 age        1              45                37
## 2 height     1             186               156
## 3 age        2              50                NA
```

This table still doesn't look great despite the fix.

Exercise 5 (Website: 12.3.3 Ex. 4)

Tidy the simple tibble below. Do you need to make it wider or longer? What are the variables in your tidy version?

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

preg %>%
  pivot_longer(c(male, female), names_to = 'Gender', values_to = 'Count')
```

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

You need to make it longer. I made new variables called `Gender` and `Count`.

Exercise 6 (Website: 12.4.3 Ex. 1)

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

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

```
## 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
```

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

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

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

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

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

```
## Warning: Expected 3 pieces. Missing pieces filled with `NA` in 1 rows [2].

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

```
tibble(x = c("a,b,c", "d,e", "f,g,i")) %>%
  separate(x, c("one", "two", "three"), fill = 'right')
```

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

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

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

`extra` controls what happens if there are too many pieces. Setting it to `'warn'` (the default), shows the warning and drops extra values, to `'drop'` drops extra values without a warning, and to `'merge'` only splits at most `length(into)` times. Note how `'warn'` and `'drop'` omitted the 'g' but `merge` added it. `fill` is similar, except it controls what happens when there aren't enough pieces. Setting it to `'warn'` is the default, which will fill from the right. You can also set it to `'right'` and `'left'`.

Exercise 7 (Website: 12.4.3 Ex. 2)

Both `unite()` and `separate()` have a `remove` argument. What does it do? Why would you set it to `FALSE`?
`remove` removes the input column from the output data frame. If you don't want to lose that column, set it to `FALSE`.

Exercise 8 (Website: 12.6)

The case study of data from the *2014 World Health Organization Global Tuberculosis Report* produces a lot of useful data (`who`) — `data` sub-directory contains the codebook for `who`. However, the format is difficult to work with, so the authors walk you through the process of tidying the data. The tidying process uses this nice concise code:

```
who %>%
  gather(code, value, new_sp_m014:newrel_f65, na.rm = TRUE) %>%
  mutate(code = stringr::str_replace(code, "newrel", "new_rel")) %>%
  separate(code, c("new", "var", "sexage")) %>%
  select(-new, -iso2, -iso3) %>%
  separate(sexage, c("sex", "age"), sep = 1)
```

```
## # A tibble: 76,046 x 6
##    country      year var   sex   age   value
##    <chr>       <int> <chr> <chr> <chr> <int>
##  1 Afghanistan  1997 sp    m     014       0
##  2 Afghanistan  1998 sp    m     014      30
##  3 Afghanistan  1999 sp    m     014       8
##  4 Afghanistan  2000 sp    m     014      52
```

```
## 5 Afghanistan 2001 sp m 014 129
## 6 Afghanistan 2002 sp m 014 90
## 7 Afghanistan 2003 sp m 014 127
## 8 Afghanistan 2004 sp m 014 139
## 9 Afghanistan 2005 sp m 014 151
## 10 Afghanistan 2006 sp m 014 193
## # ... with 76,036 more rows
```

Insert a comment following each # in the code below that explains the purpose or objective of the line of code directly below it.

```
who_tidy <- who %>%
  #  condenses the columns from `new_sp_m014` to `newrel_f65` into two new variables called `'code'` an
  pivot_longer(cols = c(new_sp_m014:newrel_f65),
               names_to = "code",
               values_to = "value",
               values_drop_na = TRUE) %>%
  #replace values with 'newrel' to 'new_rel', for consistency.
  mutate(code = stringr::str_replace(code, "newrel", "new_rel")) %>%
  # separates 'code' into three new variables, 'new', 'var', and 'sexage', based on underscore separato
  separate(code, c("new", "var", "sexage")) %>%
  #delete 'new', 'iso2', and 'iso3' variables
  select(-new, -iso2, -iso3) %>%
  #Separate 'sexage' variable into 'sex' and 'age', separate in between first and second index
  separate(sexage, c("sex", "age"), sep = 1)

who_tidy
```

```
## # A tibble: 76,046 x 6
##    country      year var   sex   age   value
##    <chr>       <int> <chr> <chr> <chr> <int>
##  1 Afghanistan  1997 sp    m     014       0
##  2 Afghanistan  1997 sp    m     1524     10
##  3 Afghanistan  1997 sp    m     2534      6
##  4 Afghanistan  1997 sp    m     3544      3
##  5 Afghanistan  1997 sp    m     4554      5
##  6 Afghanistan  1997 sp    m     5564      2
##  7 Afghanistan  1997 sp    m     65        0
##  8 Afghanistan  1997 sp    f     014       5
##  9 Afghanistan  1997 sp    f     1524     38
## 10 Afghanistan  1997 sp    f     2534     36
## # ... with 76,036 more rows
```

Exercise 9 (Website: 12.6.1 Ex. 1)

In the WHO case study, the authors set `na.rm = TRUE` to make it easier to check that they had the correct values. Is this reasonable? Think about how missing values are represented in this dataset. Are there implicit missing values? What's the difference between `NA` and zero in this dataset? **The appropriateness of using `na.rm = TRUE` depends on what the `NA`'s mean. Do they mean no cases of TB or just missing data? To test if it is no cases, I will search for 0's in the dataset.**

```
who_tidy %>%
  filter(value == 0)
```

```
## # A tibble: 11,080 x 6
```

```
##    country      year var   sex   age   value
##    <chr>       <int> <chr> <chr> <chr> <int>
##  1 Afghanistan  1997 sp    m     014       0
##  2 Afghanistan  1997 sp    m     65        0
##  3 Afghanistan  1997 sp    f     5564      0
##  4 Afghanistan  2007 sn    m     014       0
##  5 Afghanistan  2007 sn    m     1524      0
##  6 Afghanistan  2007 sn    m     2534      0
##  7 Afghanistan  2007 sn    m     3544      0
##  8 Afghanistan  2007 sn    m     4554      0
##  9 Afghanistan  2007 sn    m     5564      0
## 10 Afghanistan  2007 sn    m     65        0
## # ... with 11,070 more rows
```

Obviously there are zeros, which tells me that the `NA`'s represent missing data, in which case it would be appropriate to set `na.rm = TRUE`.

```
dim(who_tidy)
```

```
## [1] 76046     6
```

```
dim(complete(who_tidy, country, year))
```

```
## [1] 80008     6
```

From this analysis, we see that completing the missing values from country and year adds rows to the dataset, indicating that there are implicit missing values.

Exercise 10 (Website: 12.6.1 Ex. 2)

In the code from the WHO case study, what happens if you neglect the `mutate()` step? (`mutate(key = stringr::str_replace(key, "newrel", "new_rel"))`)

```
who_no_mutate <- who %>%
  pivot_longer(cols = c(new_sp_m014:newrel_f65),
               names_to = "code",
               values_to = "value",
               values_drop_na = TRUE) %>%
  separate(code, c("new", "var", "sexage")) %>%
  select(-new, -iso2, -iso3) %>%
  separate(sexage, c("sex", "age"), sep = 1)
```

```
## Warning: Expected 3 pieces. Missing pieces filled with `NA` in 2580 rows [243,
## 244, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 903,
## 904, 905, 906, ...].
```

When you don't use `mutate`, `'newrel'` will not have the same format at the other codes, and thus won't be separated properly into `'new'`, `'var'`, and `'sexage'`. Thus, we'll get errors like we saw in Exercise 6.

Exercise 11 (Website: 12.6.1 Ex. 3)

The authors of the WHO case study claimed that `iso2` and `iso3` were redundant with `country`. Confirm this claim.

```
who %>%
  count(country, iso2, iso3) %>%
  count(country) %>%
  filter(n > 1)
```

```
## # A tibble: 0 x 2
## # ... with 2 variables: country <chr>, n <int>
```

There are no ovservations with more than one distinct combination of `country`, `iso2`, and `iso3`, thus they are redundant.

Exercise 12 (Website: 12.6.1 Ex. 4)

For each level of `country`, `year`, and `sex`, compute the total number of cases of TB (using the WHO case study data). Construct an informative visualization.

```
who_tidy %>%
  group_by(country, year, sex) %>%
  summarize(n = sum(value)) %>%
  filter(n > 50000) %>%
  ggplot(mapping = aes(x = year, y = n, color = country)) +
  geom_line(alpha = 0.5) +
  facet_wrap(~sex) +
  theme(legend.position = 'bottom')
```