# Lecture Assignment 12

Taiki Yamashita

2024-05-14

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.0     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

## 12.2.1 Question 1) —–

## Using prose, describe how the variables and observations are organised in each of the sample tables.

In table1, each row represents a (country, year) combination. The cases and population columns contain the values for those variables.

```
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
```

In table2, each row represents a (country, year, variable). The column count contains the values of variables cases and population in separte rows.

```
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
```

In table3, each row represents a (country, year) combination. The column rate gives the values of both cases and population.

```r
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 4 has been divided into two separate tables, namely table4a and table4b, each focusing on different variables. Table4a presents the data for cases, while table4b provides information on population. In both tables, countries are listed in rows, years in columns, and the individual cells denote the corresponding values of the variables for each country and year.

```r
table4a # values of cases
```

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

```r
table4b # values of population
```

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

### 12.2.1 Question 2) —–

### Compute the rate for table2, and table4a + table4b. You will need to perform four operations:

1) Extract the number of TB cases per country per year.

```
t2_cases <- filter(table2, type == "cases") %>%
  rename(cases = count) %>%
  arrange(country, year)
```

2) Extract the matching population per country per year.

```
t2_population <- filter(table2, type == "population") %>%
  rename(population = count) %>%
  arrange(country, year)
```

3) Divide cases by population, and multiply by 10000 and 4) Store back in the appropriate place.

Now we need to create a new data frame with population and cases column, and then calculate the cases per capita in a new column.

```
t2_cases_per_cap <- tibble(
  year = t2_cases$year,
  country = t2_cases$country,
  cases = t2_cases$cases,
  population = t2_population$population
) %>%
  mutate(cases_per_cap = (cases / population) * 10000) %>%
  select(country, year, cases_per_cap)
```

Now we have to store this new variable in the appropriate location. Therefore, we will add new rows to table2. For table4a and table4b, we will create a new table for cases per capita. We will name it table4c with country rows and year columns.

```
t2_cases_per_cap <- t2_cases_per_cap %>%
  mutate(type = "cases_per_cap") %>%
  rename(count = cases_per_cap)

bind_rows(table2, t2_cases_per_cap) %>%
  arrange(country, year, type, count)
```

```
## # A tibble: 18 x 4
##    country      year type            count
##    <chr>       <dbl> <chr>           <dbl>
##  1 Afghanistan  1999 cases         7.45e+2
##  2 Afghanistan  1999 cases_per_cap 3.73e-1
##  3 Afghanistan  1999 population    2.00e+7
##  4 Afghanistan  2000 cases         2.67e+3
```

```
##  5 Afghanistan  2000 cases_per_cap 1.29e+0
##  6 Afghanistan  2000 population    2.06e+7
##  7 Brazil       1999 cases         3.77e+4
##  8 Brazil       1999 cases_per_cap 2.19e+0
##  9 Brazil       1999 population    1.72e+8
## 10 Brazil       2000 cases         8.05e+4
## 11 Brazil       2000 cases_per_cap 4.61e+0
## 12 Brazil       2000 population    1.75e+8
## 13 China        1999 cases         2.12e+5
## 14 China        1999 cases_per_cap 1.67e+0
## 15 China        1999 population    1.27e+9
## 16 China        2000 cases         2.14e+5
## 17 China        2000 cases_per_cap 1.67e+0
## 18 China        2000 population    1.28e+9
```

```r
table4c <-
  tibble(
    country = table4a$country,
    `1999` = table4a[["1999"]] / table4b[["1999"]] * 10000,
    `2000` = table4a[["2000"]] / table4b[["2000"]] * 10000
  )
table4c
```

```
## # A tibble: 3 x 3
##   country     `1999` `2000`
##   <chr>        <dbl>  <dbl>
## 1 Afghanistan  0.373   1.29
## 2 Brazil       2.19    4.61
## 3 China        1.67    1.67
```

Which representation is easiest to work with? Which is hardest? Why? Table2 isn't the most user-friendly format. Since it lists cases and population separately, we needed to consolidate them into one table to calculate cases per capita. In contrast, table4a and table4b split cases and population, simplifying the calculation, though it required repeating the process for each row.

## 12.3.3 Question 1) —–

Why are pivot_longer() and pivot_wider() not perfectly symmetrical? Carefully consider the following example:

```r
stocks <- tibble(
  year   = c(2015, 2015, 2016, 2016),
  half  = c(   1,    2,     1,    2),
  return = c(1.88, 0.59, 0.92, 0.17)
)
stocks %>%
  pivot_wider(names_from = year, values_from = return) %>%
  pivot_longer(`2015`:`2016`, names_to = "year", values_to = "return")
```

```
## # A tibble: 4 x 3
##    half year  return
```

```
##   <dbl> <chr>  <dbl>
## 1     1 2015    1.88
## 2     1 2016    0.92
## 3     2 2015    0.59
## 4     2 2016    0.17
```

The symmetry between `pivot_longer()` and `pivot_wider()` is not perfect due to the loss of column type information when transitioning from wide to long format. With `pivot_longer()`, multiple columns with varying data types are merged into a single column, resulting in the loss of individual data type distinctions. Conversely, `pivot_wider()` derives column names from values within a column, always treating them as character values for `pivot_longer()`. Therefore, if the original variable for column names wasn't of character data type, the round-trip conversion may not accurately recreate the initial dataset.

```r
glimpse(stocks)
```

```
## Rows: 4
## Columns: 3
## $ year   <dbl> 2015, 2015, 2016, 2016
## $ half   <dbl> 1, 2, 1, 2
## $ return <dbl> 1.88, 0.59, 0.92, 0.17
```

pivot_wider() pivots the table to create a data frame with years as column names and values in return as column values.

```r
stocks %>%
  pivot_wider(names_from = year, values_from = return)
```

```
## # A tibble: 2 x 3
##    half `2015` `2016`
##   <dbl>  <dbl>  <dbl>
## 1     1   1.88   0.92
## 2     2   0.59   0.17
```

pivot_longer() unpivots the table, returning it to a tidy data frame with columns for half, year, and return.

```r
stocks %>%
  pivot_wider(names_from = year, values_from = return)%>%
  pivot_longer(`2015`:`2016`, names_to = "year", values_to = "return")
```

```
## # A tibble: 4 x 3
##    half year  return
##   <dbl> <chr>  <dbl>
## 1     1 2015    1.88
## 2     1 2016    0.92
## 3     2 2015    0.59
## 4     2 2016    0.17
```

In the new data frame, year has a data type of character than numeric. Instead, we can use the `names_transform` argument to `pivot_longer()`, which provides a function to coerce the column to a different data type.

```
stocks %>%
  pivot_wider(names_from = year, values_from = return)%>%
  pivot_longer(`2015`:`2016`, names_to = "year", values_to = "return",
               names_transform = list(year = as.numeric))
```

```
## # A tibble: 4 x 3
##    half  year return
##   <dbl> <dbl>  <dbl>
## 1     1  2015   1.88
## 2     1  2016   0.92
## 3     2  2015   0.59
## 4     2  2016   0.17
```

## 12.3.3 Question 2) ——

Why does this code fail?

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

This code fails because the column names 1999 and 2000 are not non-syntactic variable names. when selecting variables from a data frame, tidyverse functions will interpret numbers like 1999 and 2000 as column numbers. pivot_longer() tries to select the 1999th and 2000th column of the data frame.

Instead we can do this to fix the error…

```
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
```

## 12.3.3 Question 3) ——

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,
```

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

Using `pivot_wider()` to widen this data frame results in columns that contain lists of numeric vectors due to the lack of unique row identification by the name and key columns. We could solve this problem by adding a row with a distinct observation count for each combination of name and key.

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

```
## # A tibble: 5 x 4
## # Groups:   name, names [4]
##   name            names  values   obs
##   <chr>           <chr>   <dbl> <int>
## 1 Phillip Woods   age        45     1
## 2 Phillip Woods   height    186     1
## 3 Phillip Woods   age        50     2
## 4 Jessica Cordero age        37     1
## 5 Jessica Cordero height    156     1
```

```r
pivot_wider(people2, names_from="name", values_from = "values")
```

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