21 Reshaping data

Soyoung Park

Pusan National University
Department of Statistics

Reshaping data

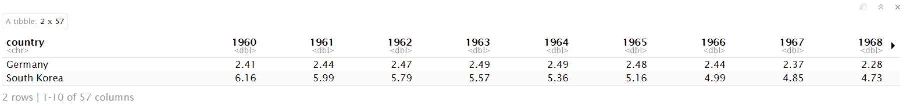
After the first step in the data analysis process, importing data, a common next step is to reshape the data into a form that facilitates the rest of the analysis. The **tidyr** package includes several functions that are useful for tidying data.

Reshaping data

We will use the fertility wide format dataset described in next section as an example.

```
library(tidyverse)
library(dslabs)
path <- system.file("extdata", package="dslabs")
filename <- file.path(path,
                      "fertility-two-countries-example.csv")
wide_data <- read_csv(filename)
```

Messy data



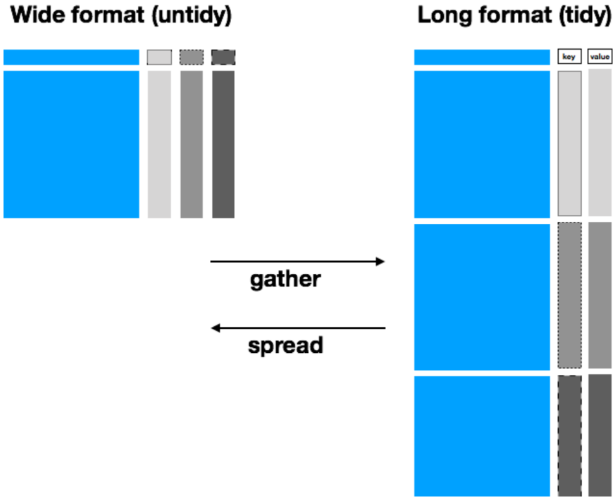
A tibble: 2 x 57

country <chr>	1960 <dbl>	1961 <dbl>	1962 <dbl>	1963 <dbl>	1964 <dbl>	1965 <dbl>	1966 <dbl>	1967 <dbl>	1968 <dbl>
Germany	2.41	2.44	2.47	2.49	2.49	2.48	2.44	2.37	2.28
South Korea	6.16	5.99	5.79	5.57	5.36	5.16	4.99	4.85	4.73

2 rows | 1-10 of 57 columns

Gather/Pivot_longer

Gather/Pivot_longer.. spread/pivot_wider



`pivot_longer`

One of the most used functions in the **tidyr** package is `pivot_longer`, which is useful for converting wide data into tidy data.

Here we want to reshape the `wide_data` dataset so that each row represents a fertility observation, which implies we need three columns to store the year, country, and the observed value. In its current form, data from different years are in different columns with the year values stored in the column names.

Through the `names_to` and `values_to` argument we will tell `pivot_longer` the column names we want to assign to the columns containing the current column names and observations, respectively. In this case a better choice for these two arguments would be `year` and `fertility`.

pivot_longer

The default is to pivot all columns so, in most cases, we have to specify the columns. In our example we want columns 1960, 1961 up to 2015.

```r
new_tidy_data <- pivot_longer(wide_data,
                              `1960`:`2015`,
                              names_to = "year",
                              values_to = "fertility")
```

We can also use the pipe like this:

```r
new_tidy_data <- wide_data %>%
  pivot_longer(`1960`:`2015`,
               names_to = "year",
               values_to = "fertility")
```

pivot_longer

We can see that the data have been converted to tidy format with columns `year` and `fertility`:

```
head(new_tidy_data)
#> # A tibble: 6 x 3
#>   country year  fertility
#>   <chr>   <chr>     <dbl>
#> 1 Germany 1960       2.41
#> 2 Germany 1961       2.44
#> 3 Germany 1962       2.47
#> 4 Germany 1963       2.49
#> 5 Germany 1964       2.49
#> # ... with 1 more row
```

pivot_longer

and that each year resulted in two rows since we have two countries and this column was not pivoted. A somewhat quicker way to write this code is to specify which column will **not** include in the pivot, rather than all the columns that will be pivoted:

```r
new_tidy_data <- wide_data %>%
  pivot_longer(-country,
               names_to = "year",
               values_to = "fertility")
```

pivot_longer

The `new_tidy_data` object looks like the original `tidy_data` we defined this way

```r
data("gapminder")
tidy_data <- gapminder %>%
  filter(country %in% c("South Korea", "Germany") &
           !is.na(fertility)) %>%
  select(country, year, fertility)
```

pivot_longer

with just one minor difference. Can you spot it? Look at the data type of the year column:

```
class(tidy_data$year)
#> [1] "integer"
class(new_tidy_data$year)
#> [1] "character"
```

The `pivot_longer` function assumes that column names are characters.

pivot_longer

So we need a bit more wrangling before we are ready to make a plot. We need to convert the year column to be numbers:

```
new_tidy_data <- wide_data %>%
  pivot_longer(-country, names_to = "year",
               values_to = "fertility") %>%
  mutate(year = as.integer(year))
```

Note that we could have also used the `mutate` and `as.numeric`.

pivot_longer

Now that the data is tidy, we can use this relatively simple ggplot code:

```
new_tidy_data %>%
  ggplot(aes(year, fertility, color = country)) +
  geom_point()
```

pivot_wider

As we will see in later examples, it is sometimes useful for data wrangling purposes to convert tidy data into wide data. The pivot_wider function is basically the inverse of pivot_longer.

```
new_wide_data <- new_tidy_data %>%
  pivot_wider(names_from = year, values_from = fertility)
```

pivot_wider

```
select(new_wide_data, country, `1960`:`1967`)
#> # A tibble: 2 x 9
#>   country     `1960` `1961` `1962` `1963` `1964` `1965` `19
#>   <chr>        <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <d
#> 1 Germany       2.41   2.44   2.47   2.49   2.49   2.48   2
#> 2 South Korea   6.16   5.99   5.79   5.57   5.36   5.16   4
```

Similar to `pivot_wider`, `names_from` and `values_from` default to `name` and `value`.

`separate`

The data wrangling shown above was simple compared to what is usually required. In our example spreadsheet files, we include an illustration that is slightly more complicated. It contains two variables: life expectancy and fertility. However, the way it is stored is not tidy and, as we will explain, not optimal.

separate

```
path <- system.file("extdata", package = "dslabs")

filename <-
  "life-expectancy-and-fertility-two-countries-example.csv"
filename <-  file.path(path, filename)

raw_dat <- read_csv(filename)
select(raw_dat, 1:5)
#> # A tibble: 2 x 5
#>   country     `1960_fertility` `1960_life_expectancy` `196
#>   <chr>                  <dbl>                  <dbl>
#> 1 Germany                 2.41                   69.3
#> 2 South Korea             6.16                   53.0
#> # ... with 1 more variable: 1961_life_expectancy <dbl>
```

`separate`

First, note that the data is in wide format.

Second, notice that this table includes values for two variables, fertility and life expectancy, with the column name encoding which column represents which variable.

Encoding information in the column names is not recommended but, unfortunately, it is quite common. We will put our wrangling skills to work to extract this information and store it in a tidy fashion.

We can start the data wrangling with the `pivot_longer` function, but we should no longer use the column name `year` for the new column since it also contains the variable type.

separate

We will call it `name`, the default, for now:

```
dat <- raw_dat %>% pivot_longer(-country)
head(dat)
#> # A tibble: 6 x 3
#>   country name                 value
#>   <chr>   <chr>                <dbl>
#> 1 Germany 1960_fertility        2.41
#> 2 Germany 1960_life_expectancy 69.3
#> 3 Germany 1961_fertility        2.44
```

separate

The result is not exactly what we refer to as tidy since each observation is associated with two, not one, rows.

We want to have the values from the two variables, fertility and life expectancy, in two separate columns.

The first challenge to achieve this is to separate the `name` column into the year and the variable type.

separate

Notice that the entries in this column separate the year from the variable name with an underscore:

```
dat$name[1:5]
#> [1] "1960_fertility"        "1960_life_expectancy" "1961_fer
#> [4] "1961_life_expectancy" "1962_fertility"
```

`separate`

Encoding multiple variables in a column name is such a common problem that the **readr** package includes a function to separate these columns into two or more.

Apart from the data, the `separate` function takes three arguments: the name of the column to be separated, the names to be used for the new columns, and the character that separates the variables.

separate

So, a first attempt at this is:

```
dat %>% separate(name, c("year", "name"), "_")
```

Because _ is the default separator assumed by separate, we do not have to include it in the code:

```
dat %>% separate(name, c("year", "name"))
```

The function does separate the values, but we run into a new problem.

`separate`

When running the above code, we receive the warning `Too many values at 112 locations:` and that the `life_expectancy` variable is truncated to `life`.

This is because the `_` is used to separate `life` and `expectancy`, not just year and variable name.

We could add a third column to catch this and let the `separate` function know which column to *fill* in with missing values, `NA`, when there is no third value.

separate

Here we tell it to fill the column on the right:

```
var_names <- c("year", "first_variable_name", "second_variable
dat %>% separate(name, var_names, fill = "right")
```

separate

However, a better approach is to merge the last two variables when there is an extra separation:

```
dat %>% separate(name, c("year", "name"), extra = "merge")
```

This achieves the separation we wanted. However, we are not done yet.

separate

We need to create a column for each variable. As we learned, the `pivot_wider` function can do this:

```
dat %>%
  separate(name, c("year", "name"), extra = "merge") %>%
  pivot_wider()
```

The data is now in tidy format with one row for each observation with three variables: year, fertility, and life expectancy.

unite

It is sometimes useful to do the inverse of separate, unite two columns into one.

Suppose that we did not know about extra and used this command to separate:

```
var_names <- c("year", "first_variable_name",
               "second_variable_name")
dat %>%
  separate(name, var_names, fill = "right")
```

unite

We can achieve the same final result by uniting the second and third columns, then pivoting the columns and renaming `fertility_NA` to `fertility`:

```
dat %>%
  separate(name, var_names, fill = "right") %>%
  unite(name, first_variable_name, second_variable_name) %>%
  pivot_wider() %>%
  rename(fertility = fertility_NA)
```

Exercises

1. Run the following command to define the `co2_wide` object:

```r
co2_wide <- data.frame(matrix(co2, ncol = 12, byrow = TRUE)) %
  setNames(1:12) %>%
  mutate(year = as.character(1959:1997))
```

Use the `pivot_longer` function to wrangle this into a tidy dataset. Call the column with the `CO2` measurements co2 and call the month column `month`. Call the resulting object `co2_tidy`.

Exercises

2. Plot CO2 versus month with a different curve for each year using this code:

```
co2_tidy %>% ggplot(aes(month, co2, color = year)) + geom_line
```

If the expected plot is not made, it is probably because `co2_tidy$month` is not numeric:

```
class(co2_tidy$month)
```

Rewrite your code to make sure the month column is numeric. Then make the plot.

Exercises

3. What do we learn from this plot?

a) CO2 measures increase monotonically from 1959 to 1997.

b) CO2 measures are higher in the summer and the yearly average increased from 1959 to 1997.

c) CO2 measures appear constant and random variability explains the differences.

d) CO2 measures do not have a seasonal trend.

Exercises

4. Now load the `admissions` data set, which contains admission information for men and women across six majors and keep only the admitted percentage column:

```
data(admissions)
dat <- admissions %>% select(-applicants)
```

If we think of an observation as a major, and that each observation has two variables (men admitted percentage and women admitted percentage) then this is not tidy. Use the `pivot_wider` function to wrangle into tidy shape: one row for each major.

Exercises

5. Now we will try a more advanced wrangling challenge. We want to wrangle the admissions data so that for each major we have 4 observations: `admitted_men`, `admitted_women`, `applicants_men` and `applicants_women`. The *trick* we perform here is actually quite common: first use `pivot_longer` to generate an intermediate data frame and then `pivot_wider` to obtain the tidy data we want. We will go step by step in this and the next two exercises.

Use the `pivot_longer` function to create a `tmp` data.frame with a column containing the type of observation admitted or `applicants`. Call the new columns `name` and `value`.

Exercises

6. Now you have an object `tmp` with columns `major`, `gender`, `name` and `value`. Note that if you combine the `name` and `gender`, we get the column names we want: `admitted_men`, `admitted_women`, `applicants_men` and `applicants_women`. Use the function `unite` to create a new column called `column_name`.

Exercises

7. Now use the `pivot_wider` function to generate the tidy data with four variables for each major.

8. Now use the pipe to write a line of code that turns `admissions` to the table produced in the previous exercise.