Reshaping Data

Dr. Osita Onyejekwe

09/14/2022

Reshaping data

As we have seen, having data in tidy format is what makes the tidyverse flow. 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.

gather

One of the most used functions in this package is gather, which converts wide data into tidy data. Let's see a simple example with a subset of the gapminder data. Here we have annual fertility rates for Germany and Korea in wide format:

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

```
## # A tibble: 2 x 57
##   country  ‘1960‘ ‘1961‘ ‘1962‘ ‘1963‘ ‘1964‘ ‘1965‘ ‘1966‘ ‘1967‘ ‘1968‘ ‘1969‘
##   <chr>     <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
## 1 Germany    2.41   2.44   2.47   2.49   2.49   2.48   2.44   2.37   2.28   2.17
## 2 South K~   6.16   5.99   5.79   5.57   5.36   5.16   4.99   4.85   4.73   4.62
## # ... with 46 more variables: ‘1970‘ <dbl>, ‘1971‘ <dbl>, ‘1972‘ <dbl>,
## #   ‘1973‘ <dbl>, ‘1974‘ <dbl>, ‘1975‘ <dbl>, ‘1976‘ <dbl>, ‘1977‘ <dbl>,
## #   ‘1978‘ <dbl>, ‘1979‘ <dbl>, ‘1980‘ <dbl>, ‘1981‘ <dbl>, ‘1982‘ <dbl>,
## #   ‘1983‘ <dbl>, ‘1984‘ <dbl>, ‘1985‘ <dbl>, ‘1986‘ <dbl>, ‘1987‘ <dbl>,
## #   ‘1988‘ <dbl>, ‘1989‘ <dbl>, ‘1990‘ <dbl>, ‘1991‘ <dbl>, ‘1992‘ <dbl>,
## #   ‘1993‘ <dbl>, ‘1994‘ <dbl>, ‘1995‘ <dbl>, ‘1996‘ <dbl>, ‘1997‘ <dbl>,
## #   ‘1998‘ <dbl>, ‘1999‘ <dbl>, ‘2000‘ <dbl>, ‘2001‘ <dbl>, ‘2002‘ <dbl>, ...
```

Recall that the gapminder data we used had a column named year and a column named fertility_rate. We would like to convert this subset into that format. We will use the gather function for this.

In the third argument of the gather function you specify the columns that will be *gathered*. The default is to gather all columns, so in most cases we have to specify the columns. Here we want columns 1960, 1961, up to 2015. The first argument sets the column/variable name that will hold the variable that is currently kept in the wide data column names. In our case it makes sense to set the name to year, but we can name it anything. The second argument sets the column/variable name that will hold the values in the column cells. In this case we call it fertility since this is what is stored in this file. Note that nowhere in this file does it tell us this is fertility data. Instead, this information was kept in the file name.
The gathering code looks like this:

```
new_tidy_data <- wide_data %>% gather(year, fertility, `1960`:`2015`)
```

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 South Korea 1960       6.16
## 3 Germany     1961       2.44
## 4 South Korea 1961       5.99
## 5 Germany     1962       2.47
## 6 South Korea 1962       5.79
```

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

```
new_tidy_data <- wide_data %>% gather(year, fertility, -country)
new_tidy_data
```

```
## # A tibble: 112 x 3
##    country     year  fertility
##    <chr>       <chr>     <dbl>
##  1 Germany     1960       2.41
##  2 South Korea 1960       6.16
##  3 Germany     1961       2.44
##  4 South Korea 1961       5.99
##  5 Germany     1962       2.47
##  6 South Korea 1962       5.79
##  7 Germany     1963       2.49
##  8 South Korea 1963       5.57
##  9 Germany     1964       2.49
## 10 South Korea 1964       5.36
## # ... with 102 more rows
```

This data looks a lot like the original `tidy_data` we used. There is just one minor difference. Can you spot it? Look at the data type of the year column:

```
library(dslabs)
data("gapminder")
tidy_data<- gapminder %>%
            filter(country %in% c("South Korea", "Germany")) %>%
            select(country, year, fertility)
            class(tidy_data$year)
```

```
## [1] "integer"
```

```
             class(new_tidy_data$year)
```

```
## [1] "character"
```

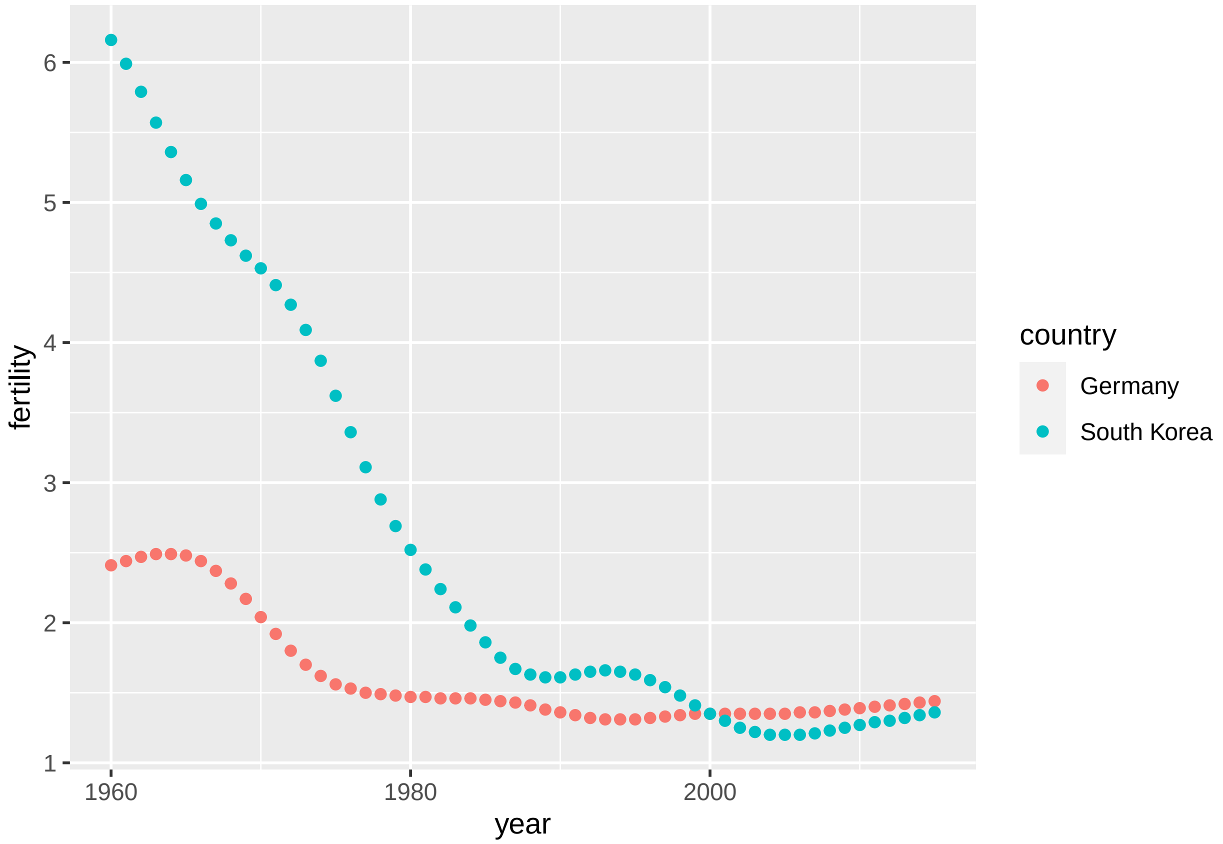
The gather function assumes that column names are characters. So we need a bit more wrangling before we are ready to make a plot. We need to convert the column to numbers. The gather function has an argument for that, the convert argument:

```
new_tidy_data <- wide_data %>%
                 gather(year, fertility, -country, convert= TRUE)
                 class(new_tidy_data$year)
```

```
## [1] "integer"
```

We could have also used the mutate and as.numeric functions. Now that the data is tidy we can use the same ggplot as before:

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



spread

As we will see in later examples it is sometimes useful for data wrangling purposes to convert tidy data into wide data. We often use this as an intermediate step in tidying up data. The spread function is basically the inverse of **gather**. The first argument tells **spread** which variable will be used as the column names. The second argument specifies which variable to use to fill out the cells:

```
new_wide_data <- new_tidy_data %>% spread(year, fertility)
                 select(new_wide_data, country, `1960`:`1967`)
```

```
## # A tibble: 2 x 9
##   country     ‘1960‘ ‘1961‘ ‘1962‘ ‘1963‘ ‘1964‘ ‘1965‘ ‘1966‘ ‘1967‘
##   <chr>        <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
## 1 Germany       2.41   2.44   2.47   2.49   2.49   2.48   2.44   2.37
## 2 South Korea   6.16   5.99   5.79   5.57   5.36   5.16   4.99   4.85
```

separate

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

```
path     <- system.file("extdata", package = "dslabs")
filename <- file.path(path, "life-expectancy-and-fertility-two-countries-example.csv")
raw_dat <- read_csv(filename)
select(raw_dat, 1:5)
```

```
## # A tibble: 2 x 5
##   country    ‘1960_fertility‘ ‘1960_life_exp~‘ ‘1961_fertility‘ ‘1961_life_exp~‘
##   <chr>                 <dbl>            <dbl>            <dbl>            <dbl>
## 1 Germany                2.41             69.3             2.44             69.8
## 2 South Kor~             6.16             53.0             5.99             53.8
```

First note that the data is in wide format. Second, note that now there are values for two variables with the column names encoding which column represents which variable. We can start the data wrangling with the `gather` function, but we should no longer use the column name `year` for the new column since since it also contains the variable type. We will call it `key`, the default, for now:

```
dat <- raw_dat %>% gather(key, value, -country)
head(dat)
```

```
## # A tibble: 6 x 3
##   country     key                  value
##   <chr>       <chr>                <dbl>
## 1 Germany     1960_fertility        2.41
## 2 South Korea 1960_fertility        6.16
## 3 Germany     1960_life_expectancy 69.3
## 4 South Korea 1960_life_expectancy 53.0
## 5 Germany     1961_fertility        2.44
## 6 South Korea 1961_fertility        5.99
```

The result is not exactly what we refer to as tidy since each observation is associated with two rows instead of one. 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 `key` column into the year and the variable type. Note that the entries in this column separate the year from the variable name with an underscore:

```
dat$key[1:5]
```

```
## [1] "1960_fertility"       "1960_fertility"       "1960_life_expectancy"
## [4] "1960_life_expectancy" "1961_fertility"
```

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. So a first attempt at this is:

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

Because "_" is the default separator we actually can simply write:

```
dat %>% separate(key, c("year", "variable_name"))
```

```
## Warning: Expected 2 pieces. Additional pieces discarded in 112 rows [3, 4, 7, 8,
## 11, 12, 15, 16, 19, 20, 23, 24, 27, 28, 31, 32, 35, 36, 39, 40, ...].
```

```
## # A tibble: 224 x 4
##    country     year  variable_name value
##    <chr>       <chr> <chr>         <dbl>
##  1 Germany     1960  fertility      2.41
##  2 South Korea 1960  fertility      6.16
##  3 Germany     1960  life          69.3
##  4 South Korea 1960  life          53.0
##  5 Germany     1961  fertility      2.44
##  6 South Korea 1961  fertility      5.99
##  7 Germany     1961  life          69.8
##  8 South Korea 1961  life          53.8
##  9 Germany     1962  fertility      2.47
## 10 South Korea 1962  fertility      5.79
## # ... with 214 more rows
```

However, we run into a problem. Note that we receive the warning `Too many values at 112 locations:` and that the `life_exepectancy` 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. Here we tell it to fill the column on the right:

```
dat %>% separate(key,
                 c("year", "first_variable_name", "second_variable_name"), fill =     "right")
```

```
## # A tibble: 224 x 5
##    country     year  first_variable_name second_variable_name value
##    <chr>       <chr> <chr>               <chr>                <dbl>
##  1 Germany     1960  fertility           <NA>                  2.41
##  2 South Korea 1960  fertility           <NA>                  6.16
##  3 Germany     1960  life                expectancy           69.3
##  4 South Korea 1960  life                expectancy           53.0
##  5 Germany     1961  fertility           <NA>                  2.44
##  6 South Korea 1961  fertility           <NA>                  5.99
##  7 Germany     1961  life                expectancy           69.8
##  8 South Korea 1961  life                expectancy           53.8
##  9 Germany     1962  fertility           <NA>                  2.47
## 10 South Korea 1962  fertility           <NA>                  5.79
## # ... with 214 more rows
```

However, if we read the `separate` help file we find that a better approach is to merge the last two variables when there is an extra separation:

```
dat %>% separate(key,
                 c("year", "variable_name"), sep = "_" , extra = "merge")
```

```
## # A tibble: 224 x 4
##    country     year  variable_name   value
##    <chr>       <chr> <chr>           <dbl>
##  1 Germany     1960  fertility        2.41
##  2 South Korea 1960  fertility        6.16
##  3 Germany     1960  life_expectancy 69.3
##  4 South Korea 1960  life_expectancy 53.0
##  5 Germany     1961  fertility        2.44
##  6 South Korea 1961  fertility        5.99
##  7 Germany     1961  life_expectancy 69.8
##  8 South Korea 1961  life_expectancy 53.8
##  9 Germany     1962  fertility        2.47
## 10 South Korea 1962  fertility        5.79
## # ... with 214 more rows
```

This achieves the separation we wanted. However, we are not done yet. We need to create a column for each variable. As we learned, the `spread` function can do this:

```
dat %>% separate(key,
                 c("year", "variable_name"), sep = "_" , extra = "merge") %>%
        spread(variable_name, value)
```

```
## # A tibble: 112 x 4
##    country year  fertility life_expectancy
##    <chr>   <chr>     <dbl>           <dbl>
##  1 Germany 1960       2.41            69.3
##  2 Germany 1961       2.44            69.8
##  3 Germany 1962       2.47            70.0
##  4 Germany 1963       2.49            70.1
##  5 Germany 1964       2.49            70.7
##  6 Germany 1965       2.48            70.6
##  7 Germany 1966       2.44            70.8
##  8 Germany 1967       2.37            71.0
##  9 Germany 1968       2.28            70.6
## 10 Germany 1969       2.17            70.5
## # ... with 102 more rows
```

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`, i.e. unite two columns into one. So, although this is *not* an optimal approach, had we used this command to separate:

```
dat %>% separate(key,
                 c("year", "variable_name", "second_variable_name"), fill = "right")
```

```
## # A tibble: 224 x 5
##    country     year  variable_name second_variable_name value
##    <chr>       <chr> <chr>         <chr>                <dbl>
##  1 Germany     1960  fertility     <NA>                  2.41
##  2 South Korea 1960  fertility     <NA>                  6.16
##  3 Germany     1960  life          expectancy           69.3 
##  4 South Korea 1960  life          expectancy           53.0 
##  5 Germany     1961  fertility     <NA>                  2.44
##  6 South Korea 1961  fertility     <NA>                  5.99
##  7 Germany     1961  life          expectancy           69.8 
##  8 South Korea 1961  life          expectancy           53.8 
##  9 Germany     1962  fertility     <NA>                  2.47
## 10 South Korea 1962  fertility     <NA>                  5.79
## # ... with 214 more rows
```

we can achieve the same final result by uniting the second and third column like this:

```
dat %>%
  separate(key, c("year", "first_variable_name", "second_variable_name"), fill = "right") %>%
  unite(variable_name, first_variable_name, second_variable_name, sep = "_")
```

```
## # A tibble: 224 x 4
##    country     year  variable_name   value
##    <chr>       <chr> <chr>           <dbl>
##  1 Germany     1960  fertility_NA     2.41
##  2 South Korea 1960  fertility_NA     6.16
##  3 Germany     1960  life_expectancy 69.3 
##  4 South Korea 1960  life_expectancy 53.0 
##  5 Germany     1961  fertility_NA     2.44
##  6 South Korea 1961  fertility_NA     5.99
##  7 Germany     1961  life_expectancy 69.8 
##  8 South Korea 1961  life_expectancy 53.8 
##  9 Germany     1962  fertility_NA     2.47
## 10 South Korea 1962  fertility_NA     5.79
## # ... with 214 more rows
```

Then spreading the columns:

```
dat %>%
  separate(key, c("year", "first_variable_name", "second_variable_name"), fill = "right") %>%
  unite(variable_name, first_variable_name, second_variable_name, sep = "_") %>%
  spread(variable_name, value) %>%
  rename(fertility = fertility_NA)
```

```
## # A tibble: 112 x 4
##    country year  fertility life_expectancy
##    <chr>   <chr>     <dbl>           <dbl>
##  1 Germany 1960       2.41            69.3
##  2 Germany 1961       2.44            69.8
##  3 Germany 1962       2.47            70.0
##  4 Germany 1963       2.49            70.1
##  5 Germany 1964       2.49            70.7
##  6 Germany 1965       2.48            70.6
##  7 Germany 1966       2.44            70.8
##  8 Germany 1967       2.37            71.0
##  9 Germany 1968       2.28            70.6
## 10 Germany 1969       2.17            70.5
## # ... with 102 more rows
```