SPEC REU R Resources: Tidying and Reshaping Data

Alix Ziff, Gaea Morales, Zachary Johnson, Marie Zaragoza, Claudia Salas Gimenez, Ben Graham

January 2025

In this module, we will continue to expand our data management skills by focusing on the essential concepts of tidying, reshaping and merging datasets. Utilizing key functions from the `dplyr` and `tidyr` packages in the tidyverse, such as `pivot_longer()`, `pivot_wider()`, and `left_join()`, this walkthrough will explore how to organize data efficiently, making it suitable for analysis.

The datasets used in this walkthrough, `cow_milex_untidy_1.rds` and `cow_milex_tidy_2.rds`, are small, cleaned subsets from the `Militarized Interstate Disputes (v5.0)` dataset from the Correlates of War project.

Initial Setup

Before diving into data manipulation, ensure your R script includes a detailed header, your working directory is set for easy access to the datasets, and the necessary libraries and datasets are loaded.

Note: We'll use both `read_csv` and `read.csv` from the `readr` package to load the `cow_milex_untidy_1.rds` dataset. Using `read.csv` will automatically add an "X" to the start of column names to ensure that all columns conform to R's syntactic rules for variable names, which do not allow numbers as column names. This also simplifies our reshaping process by making it easier to select columns with numerical names or special characters for transformation.

```r
# Set working directory
#setwd("YourFolderPath")

# Load required libraries
library(tidyverse)
library(readr)

# Load the data
cow1 <- read_csv("cow_milex_untidy_1.csv")
cow1a <-read.csv("cow_milex_untidy_1.csv")
## read.csv will add an `X` to the start of column names
cow2 <- read.csv("cow_milex_untidy_2.csv")
## read.csv names the first column `X` instead of '1...'

## We'll use both cow1 and cow1a to show how the pivot_longer()
## function changes depending on the structure of the data
```

Examining the Data

First, take a look at the structure of `cow1` and `cow2`.

```r
# Display the first six rows of the 'cow1' data object
head(cow1)
```

```r
# Display the first six rows of the 'cow2' data object
head(cow2)
```

In this dataset, each row represents a country (indicated by `ccode`), and each column represents a year with data on that country's military expenditures in that given year.

These datasets represent small, cleaned subsets from a larger dataset, `Militarized Interstate Disputes (v5.0)`, from the Correlates of War project. Make sure to refer to the codebook to better understand the dataset and its variables. The codebook provides descriptions for all variables in the `Militarized Interstate Disputes (v5.0)` dataset, including those in the subsets we're using. Can you identify what each variable represents in this context?

Tidying Data

One of the most crucial steps in data management is ensuring that your data is "tidy." Tidy data is structured in a way that is intuitive, easy to manipulate, and ready for analysis. It involves reshaping the data so that:

- Each row is a unique observation (e.g., a specific country in a specific year like a country-year like the U.S. in 1985 or Zimbabwe in 2019).

1. Each column is a single variable with information about the observations.
2. The dataset is organized around a single type of observational unit (i.e. we don't have information about country-years and country-days in the same dataframe).

To better grasp what a tidy dataframe looks like, consider the `cow_example` dataset.

```r
# Load the data
cow_example <- read_csv("cow_example.csv")

# Display dataset
head(cow_example)
```

Here, each row represents a unique country-year observation, detailing attributes like military expenditures or personnel.

For a different example, consider the `lewis_example` from a YouGov public poll in October 2020, where participants were asked about establishing the LEWIS registry to track police officers dismissed for misconduct.

```r
# Load the data
lewis_example <- read_csv("lewis_example.csv")

# Display dataset
head(lewis_example)
```

In this dataset, each row represents a single poll participant, with each column capturing specific details about the participant's demographics and opinions, such as birth year, gender, race, and political affiliation. Thus, the data is structured in a tidy format!

Reshaping Data

We can make dataframes tidy by reshaping data using functions like `pivot_longer()` or `pivot_wider()`.

Using `pivot_longer()`

`pivot_longer()` transforms dataframes to "long" format by increasing the number of rows and decreasing the number of columns. This function is ideal when multiple observations are stacked in one row, and we need to separate them to have information about one observation in each row.

For this first assignment, we will use the `cow1` data and select the year columns by applying the `matches("^(19|20)")` function, which captures any column names starting with 19 or 20. This ensures we correctly identify all the year columns in the dataset.

```r
# Convert wide data format to long format
longData <- cow1 %>%
  pivot_longer(cols = matches("^(19|20)"),
               ## Match columns that start with '19' or '20'
               names_to = "year",
               ## Create 'year' variable to store all the year columns
               values_to = "milex") %>%
               ## Create 'milex' variable to store values for military expenditures
  select(ccode, year, milex)
  ## Keep only the 'ccode', 'year', and 'milex' columns

# Display dataset
head(longData)
```

This reorganization produces a streamlined dataframe with only the `ccode`, `year`, and `milex` columns, ensuring each row distinctly represents military expenditures for a specific country and year. There is only one piece of information per row for each observation.

For this second method, we'll be using `cow1a` and selecting columns that refer to years with the code `matches("^X(19|20)")`. An additional difference from the previous example on how to use `pivot_longer()` is the addition of the `names_pattern()` function, which separates the actual year data from the prefix "X". The `names_pattern` parameter uses a regular expression (`"X(\\d+)"`) to extract just the numeric portion from column names that start with "X", transforming them into a usable year format for easier analysis.

```r
# Convert wide data format to long format
longData2 <- cow1a %>%
  pivot_longer(cols = matches("^X(19|20)"),
               ## Select columns that start with 'X19' or 'X20'
               names_to = "year",
               values_to = "milex",
               names_pattern = "X(\\d+)") %>%
               ## Use 'names_pattern' to extract the years from the column names
  select(ccode, year, milex)
  ## Keep only the 'ccode', 'year', and 'milex' columns

# Display dataset
head(longData2)
```

Using `pivot_wider()`

Conversely, `pivot_wider()` is used to create "wide" dataframes by decreasing the number of rows and increasing the number of columns. It's not used as often as `pivot_longer()`, but `pivot_wider()` is useful when tidying dataframes that have information from multiple variables for each observation. We only want one row of information for each observation.

Take a look at `cow2`. Each observation (a unique country-year) is represented by two rows, with the `value` column containing information for two variables: military expenditures and per capita expenditures. Our goal is to restructure the data by creating separate columns for `milex` and `milexper`, ensuring each country-year is consolidated into a single row.

Note: Before reshaping, we need to remove the "X" column on the left, as it's an index column with no useful information and isn't needed for your reshaping operation. If we don't handle it, it will interfere with the reshaping process and prevent us from reducing the rows as intended.

```r
# Convert wide data format to wide format
wideData <- cow2 %>%
  select(-X) %>%
  # Remove the column "X" so that we can reshape these data.
  pivot_wider(names_from = variable,
              values_from = value)

# Display dataset
head(wideData)
```

```
## # A tibble: 6 x 4
##   ccode  year    milex milper
##   <int> <int>    <int>  <int>
## 1     2  1967 75448000   3380
## 2     2  1968 80732000   3550
## 3     2  1969 81446000   3460
## 4     2  1970 77827008   3070
## 5     2  1971 74862000   2720
## 6     2  1972 77639000   2323
```

Conclusion

This document introduced key concepts of tidying and reshaping data using functions like `pivot_longer()` and `pivot_wider()`. Next, we'll explore how to merge datasets, and you'll have the opportunity to practice these techniques in the groupwork and homework assignments. Mastering these functions will help you efficiently organize data, making it easier to analyze and visualize.