SPEC Lab REU R Resources: Data Management II- Merging & Reshaping Data

Alix Ziff, Gaea Morales, Zachary Johnson, and Ben Graham.

April 2021

This workshop provides an introduction to reshaping datasets to make them "tidy" and merging datasets together in `R` using the `dplyr` and `tidyr` packages, both part of the `tidyverse`. We will work with data drawn from two sources often used in international relations research: *Correlates of War* and *World Development Indicators*. The two files you need are cow_cleaned.csv & wdi_milex_April2021.rds

Recall that in a "tidy" dataframe:

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

2). Each column is a variable with information about the observations (e.g., information about the country-years)

3). There is only one type of observation in each dataframe (i.e. we don't have information about country-years and country-days in the same dataframe)

Before you get started, write a detailed header in your R script and save your R script. Then set your working directory to the Training Data folder.

Let's load our packages.

```r
library(tidyverse)
```

We're going to read in data from the World Development Indicators and name the data object *wdi*. Take a look at the data summary. Do the same for the Correlates of War data but name the object *cow*.

```r
cow <- read.csv("cow_cleaned.csv")
wdi <- readRDS("wdi_milex_April2021.rds")

summary(wdi$year)
summary(cow$year)
```

Merging Data

Exercise: rbind & cbind issues

In principle, we can merge data using the cbind() and rbind() functions that we are already familiar with.

```r
vec1 <- c(1, 2, 3) #creating an object with numerical values 1-3
vec2 <- c("a", "a", "b") #creating an object with character string a,a,b

cbind(vec1, vec2) #putting these two vectors together as variables into a dataframe

rbind(vec1, vec2) #putting these two vectors together as observations into a dataframe
```

Helpful Hint: Remember that rows are observations and columns are variables!

However, this requires the data to align in its dimensions. In addition, for cbind() the ordering of the rows has to be the exact same in each dataframe to achieve correct merging; for rbind() the ordering of the columns has to be the same in both dataframes to achieve the right output.

And what happens if one dataframe has more observations or more variables than the other?

```
dat1 <- cbind(vec1, vec2)
vec3 <- c(T, F, T, T) #adding another character string vector
cbind(dat1, vec3) #adding in vector 3 (with 4 values not 3) to the dat1 data frame
```

```
## Warning in cbind(dat1, vec3): number of rows of result is not a multiple of
## vector length (arg 2)
```

Right, information gets dropped. The fourth observation in vector 3 gets omitted. The same things happens if we have the wrong number of columns for a row bind.

```
obs <- c("c", 4)
rbind(dat1, obs)
```

Sheesh, this world is bad enough to make a person buy STATA…

tidyr saves the day!

We are going to use the `full_join()` function in tidyr to merge instead. `full_join()', along with its companions,`inner_join()’ and `left_join()`, merge two dataframes together, based on one or more variables that:

1). Are present in both dataframes you want to merge together

2). Uniquely identify each observation in each dataframe.

So the trick in working with these join commands is to select the proper "by" variable(s)

Try to figure out what happens to our data when we do the following:

```
wrong <- full_join(wdi, cow, by = "ccode") #merging wdi and cow by ccode

alsowrong <- full_join(wdi, cow) #let R try to figure out what to merge by
```

```
## Joining, by = c("ccode", "year", "milex")
```

We have way too many observations and duplicates.

Now let's `full_join()` properly!

```
#merging wdi & cow by ccode AND year, because together they uniquely identify observations
newdata <- full_join(wdi, cow, by = c("ccode", "year"))%>%
  relocate(ccode, year) #reordering the variables at the end to make it pretty


head(newdata) #looking at the first few values of our new dataset
nrow(wdi) #double checking the number of rows in WDI
nrow(cow) #double checking the number of rows in COW
nrow(newdata) #double checking the number of rows in our new dataset
```

How many rows do we have here and why?

So the merge worked OK. However, things aren't as good as they could be. Both the wdi data and the cow data have military expenditures measures called milex. Now we have a milex.x variable and a milex.y variable. Which is fine, but we have to remember the the .x version came from wdi, which was listed first in our join function and the .y version came from cow, which was listed second in our join. I am going to forget that!

Before we merge datasets together, it is usually better if we name the variables in such a way that we can tell where they came from later. In this case, we should rename the variables to something like milex_wdi and milex_cow. When you prep new data for use in the SPEC lab, we always use _[SUFFIX] to identify what data source each variable comes from.

```
cow2 <- cow %>%
  rename(milex_cow = milex)

wdi2 <- wdi %>%
  rename(milex_wdi = milex)
```

Let's use the function `inner_join()` to only keep the observations that are present in both datasets. You will see this leaves us with fewer observations than 'full_join()'

```
innerData <- inner_join(wdi2, cow2, by = c("ccode", "year")) %>%
  arrange(ccode, year)

head(innerData)
nrow(innerData)
nrow(newdata)
```

Helpful Hint: Remember that if we have a country for example, that is included in one dataset but not the other it will be dropped using the `inner_join` function.

Now, we'll use the function `left_join()` to keep only the observations in the first dataset listed in the join command, in this case, wdi.

```
leftData <- left_join(wdi2, cow2, by = c("ccode", "year"))

nrow(wdi) == nrow(leftData)
```

The only reason we were able to merge the cow and wdi datasets is that they were both already tidy, and they both already had the same unique identifiers for their observations. In each dataset, each observation is a country-year, and those unique observations are identified by the same variables, *cccode* and *year*.

But data don't always come to us tidy. There is a LOT of work to clean these data to prepare them to be merged together. In fact, Data Management IIA is all about how we do that in the lab, working with Gleditsch-Ward numbers (gwno) as our numeric country codes.

One key step in making tidy data is to make sure that each row in our dataframe represents a single observation and each column represents a variable. To do that, we often need to reshape datasets.

Reshaping Data

We use the `pivot_longer()` and `pivot_wider()` commands, which have replaced the (very similar) `gather()` and `spread()` commands.

We use `pivot_longer()` in situations where there are multiple observations in each row of data. We only want information about *one* observation in each row.

The `pivot_longer()` reduces the number of columns and increases the number of rows, hence making the dataframe *longer*. 'pivot_longer()' takes multiple columns and reduces them to two new simplified variables: one column with the index (the old column names) and one column with the values.

```
longData <- pivot_longer(leftData, #this is the dataframe we are going to reshape
    names_to = "data_origin", #naming the new column, which has the old column names as values
    values_to = "milex", #naming the new column that will have the values in it
    c(milex_wdi, milex_cow)) %>% #the columns we want to pull down as rows.
    #We need to grab these columns as a single object. Concatenate is one way to do this.
    select(ccode, year, data_origin, milex) #keep just the variables we want

head(longData)
```

So now the *data_origin* column tells us where the information comes from and *milex* gives us the values for miltary expenditures, which come from the *milex_wdi* and *milex_cow* variables.

If we wanted to reverse what we did with the `pivot_longer` function, and make the dataset wider we would pull data from a variable and create separate columns for each unique value.

In practice, we often use `pivot_wider()` to tidy dataframes that have multiple rows of information for each observation. We only want one row of information for each observation.

```
wideData <- pivot_wider(longData,
                        names_from = data_origin,
                        values_from = milex)
head(wideData)
```

Helpful Hint: Whereas wide data (`pivot_wider()` formally know as `spread()`) is indexed by two variables (from the column and the row), long data (`pivot_longer()` the function formally known as `gather()`) is just indexed from the row.