SPEC REU R Resources: Merging Data

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

January 2025

This second walkthrough in Module 3: Data Management II covers how to merge datasets in R using the `dplyr` and `tidyr` packages from the tidyverse. We'll explore the use of `rbind()` and `cbind()` functions, and we'll focus on merging datasets using functions like `full_join()`, `inner_join()`, `right_join()` and `left_join()`, based on common identifiers.

In this walkthrough, we will work with small, cleaned subsets from two common sources in international relations research: `cow_cleaned.csv` from the Correlates of War, and `wdi_milex_April2021.rds` from the World Development Indicators.

Initial Setup

Make sure to add a header to your R script, set your working directory to the data folder location, and load the required libraries and datasets.

Similar to Walkthrough 1, both datasets in this exercise are small, cleaned subsets of larger datasets. While they may not resemble the format typically encountered in research, they are useful in this context for learning how to merge data. To gain a deeper understanding of the datasets and their variables, refer to the Militarized Interstate Disputes (v5.0) codebook and the World Development Indicators data catalog to better understand the datasets and their variables. Keep in mind that since these datasets have been modified for this walkthrough, the variable names may differ slightly from those in the original codebooks.

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

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

# Load the data
cow <- read.csv("cow_cleaned.csv")
wdi <- readRDS("wdi_milex_April2021.rds")
```

Features of a "Tidy" Dataframe Recap

Before we begin, let's review the key features of a "tidy" dataframe:

- Each row is a unique observation.
- Each column is a variable with information about the observations.
- There is only one type of observation in each dataframe.

Merging Data

Merging data is a crucial part of data management. Often, the information needed for research comes from multiple sources, and combining them into one cohesive dataset allows for more comprehensive analysis and a broader picture of the real-world phenomena you're studying.

Before merging datasets, it's important to ensure both are in a tidy format to avoid errors during the process. This includes addressing missing data, correcting any inconsistencies, and filtering for relevant variables or observations based on your analysis needs. In this walkthrough, the datasets have already been cleaned and tidied, as our focus will be on learning key functions for efficiently merging data.

Understanding `rbind()` & `cbind()`

One simple and straightforward way to merge data in R is by using the functions `rbind()` (for row-wise binding) and `cbind()` (for column-wise binding). These functions allow you to combine datasets with the same structure, meaning their dimensions must align perfectly. Specifically, `rbind()` combines datasets by adding new rows, while `cbind()` adds new columns.

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

```
# Create vectors
vec1 <- c(1, 2, 3) # Create numeric vector
vec2 <- c("a", "a", "b") # Create character vector

# Combine vec1 and vec2 using cbind()
cbind(vec1, vec2)
```

```
##      vec1 vec2
## [1,] "1"  "a"
## [2,] "2"  "a"
## [3,] "3"  "b"
```

```
# Combine vec1 and vec2 using rbind()
rbind(vec1, vec2)
```

```
##      [,1] [,2] [,3]
## vec1 "1"  "2"  "3"
## vec2 "a"  "a"  "b"
```

However, for `rbind()` and `cbind()` to work properly, the datasets must have the same number of rows or columns, respectively. If they don't align, using these functions can result in dropped or mismatched data, leading to incorrect results.

```
# Combine vec1 and vec2 using cbind() and store it as object dat1
dat1 <- cbind(vec1, vec2)

# Create string vector
vec3 <- c(T, F, T, T)

# Combine dat1 and vec3 using cbind()
cbind(dat1, vec3)
```

```
##      vec1 vec2 vec3
## [1,] "1"  "a"  "TRUE"
## [2,] "2"  "a"  "FALSE"
## [3,] "3"  "b"  "TRUE"
```

```
# Mismatching observations
obs <- c("c", 4)
```

```r
# Combine dat1 and mismatching observations using rbind()
rbind(dat1, obs)
```

```
##     vec1 vec2
##     "1"  "a"
##     "2"  "a"
##     "3"  "b"
## obs "c"  "4"
```

As we can see, information gets dropped—specifically, the fourth observation in `vec3` is omitted. The same thing happens if the number of columns doesn't match when using `rbind()`. In such cases, it's better to use more flexible functions like `full_join()`, `inner_join()`, `right_join()`, or `left_join()` from the `dplyr` package.

Dataset Merging with `dplyr` Functions

Functions like `full_join()`, `left_join()`, `right_join()`, and `inner_join()` offer more versatile ways to combine datasets based on shared key columns. `dplyr` provides six join functions: `left_join()`, `inner_join()`, `right_join()`, `full_join()`, `semi_join()`, and `anti_join()`, and we'll focus on the first four, as they are the most commonly used. Here's an overview of how these functions work:

- `full_join()` retains all rows from both datasets, filling in `NA` values for any unmatched columns when there is no match between the key columns.

  ```r
  full_join(data1, data2, by = "key_column")
  ```

- `left_join()` keeps all rows from the left dataset and adds matching rows from the right dataset, filling in `NA` values for the right dataset's columns where no match is found.

  ```r
  left_join(left_data, right_data, by = "key_column")
  ```

- `right_join()` keeps all rows from the right dataset and includes only matching rows from the left dataset, with unmatched columns filled in with `NA`.

  ```r
  right_join(left_data, right_data, by = "key_column")
  ```

- `inner_join()` keeps only the rows that have matching values in both datasets.

  ```r
  inner_join(left_data, right_data, by = "key_column")
  ```

These functions streamline the merging process, even if the dataset structures aren't perfectly aligned. However, choosing the right "by" variables is important for correctly matching rows in R. When selecting these variables, ensure you:

- Choose the columns that are common to both datasets you plan to merge.
- Select "by" variables that uniquely identify each row within the dataset to avoid ambiguity during the join.
- Confirm that the "by" variables share the same data type and format in both datasets.

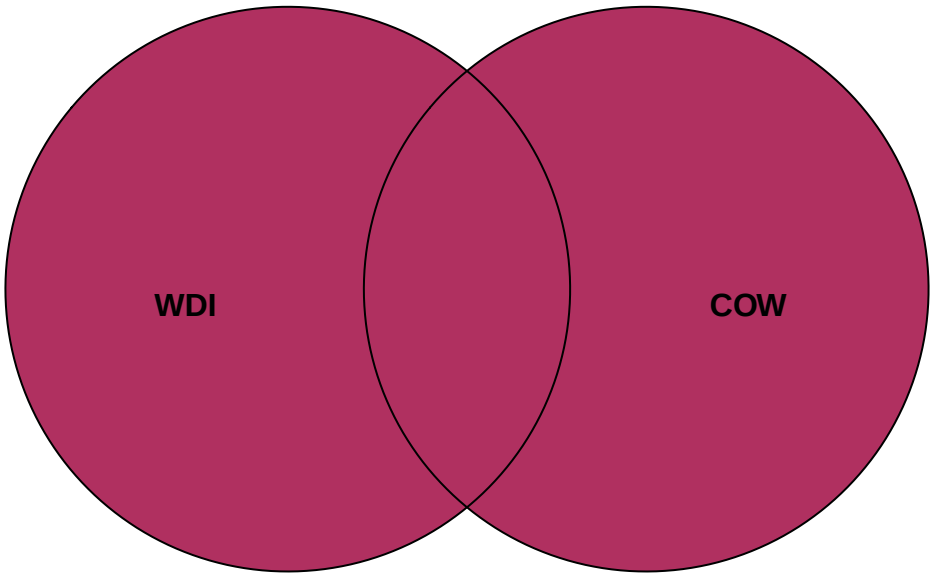
Before performing the join, check the "by" variables for missing values and data errors to avoid issues like duplicated rows or incorrect merges, which can complicate your analysis. For the datasets `cow` and `wdi`, we'll use `ccode` and `year` as our "by" variables, since these columns are included in both datasets, represent the same information, and are complete. In this examples, the datasets are cleaned and ready for use join functions, but we'll practice more in the groupwork and homework assignments.

Using `full_join()`

Let's start by merging the datasets using the `full_join()` function. This function retains all observations from both datasets, even if there are no matching rows in the other dataset. Below is a Venn diagram for

`full_join()`, which shows that all observations from both datasets are included. The overlapping area represents the rows that have matches on `ccode` and `year` in both datasets.

```
library(eulerr)

# Plot the Venn diagram for full join
plot(euler(c("WDI" = 3, "COW" = 3, "WDI&COW" = 1)),
     quantities = FALSE, fills = c("maroon", "maroon", "maroon"))
```



Now, let's code the merge:

```
# Merge by 'ccode' and 'year' using full_join()
full_data <- full_join(wdi, cow, by = c("ccode", "year"))%>%
  relocate(ccode, year)  ## Reorder the variables
```

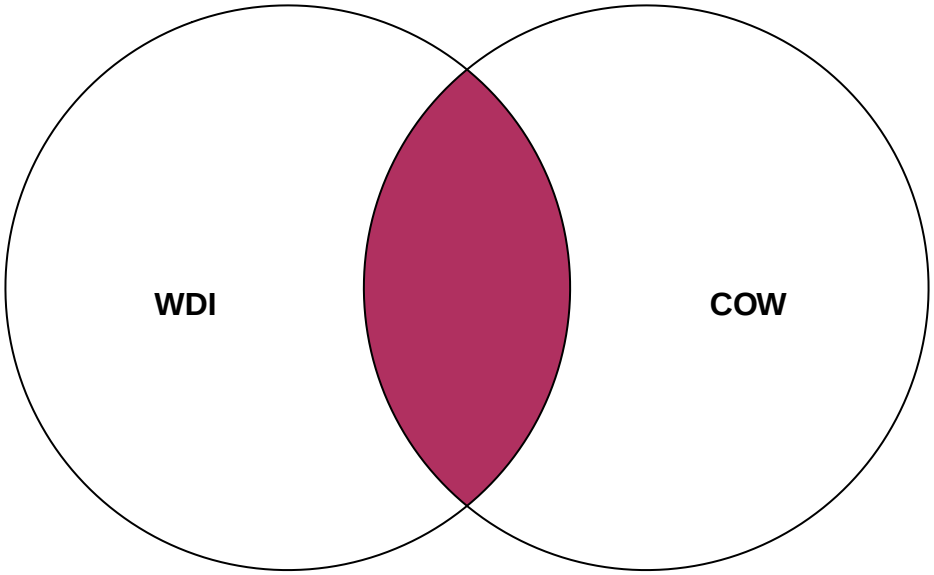
The merge produced the correct number of rows, but since both the `wdi2` and `cow2` datasets have a `milex` variable that isn't used in the merge, they appear as `milex.x` (from `wdi2`) and `milex.y` (from `cow2`). It's a good practice to rename these variables prior to merging the data, such as `milex_wdi` and `milex_cow` to clearly indicate their sources and avoid confusion.

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

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

## Use 'cow2' and 'wdi2' for the upcoming exercises

# Merge by 'ccode', 'year' and 'milex' using full_join()
full_data1 <- full_join(wdi, cow, by = c("ccode", "year", "milex"))%>%
  arrange(ccode, year, milex)  ## Reorder the variables
```

Using `inner_join()`

Let's use the `inner_join()` function to keep only the observations that have matching values in both datasets. This is useful when you want to focus on the data common to both datasets, excluding any entries that don't have matches. The following Venn diagram illustrates the data we retain after performing the `inner_join()`.

```r
plot(euler(c("WDI" = 3,"COW" = 3,"WDI&COW" = 1)),
     quantities = FALSE,fills = c("white","white","maroon"))
```



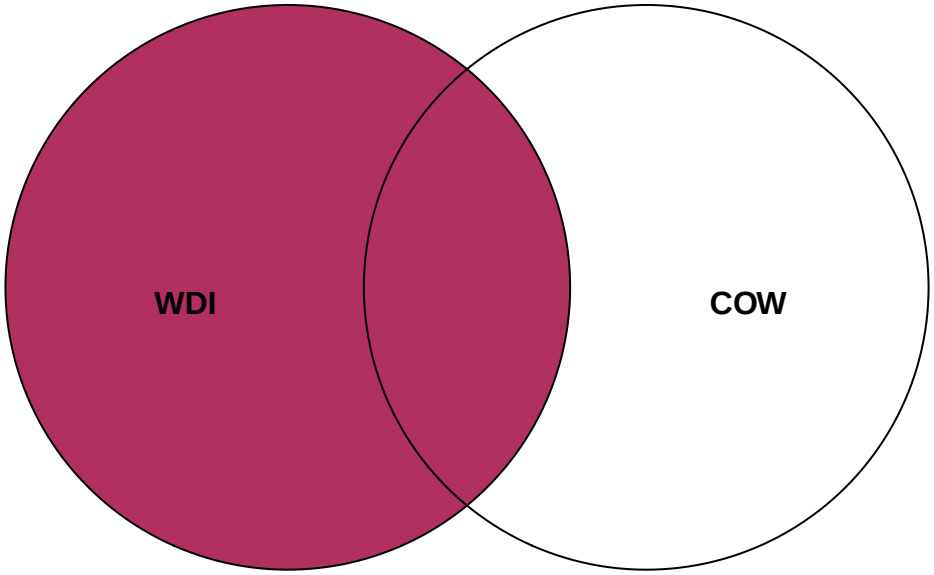
In this case, the function will retain only the observations where both `ccode` and `year` values are present in both the `wdi2` and `cow2` datasets.

```r
# Merge by 'ccode', 'year' using inner_join()
inner_data <- inner_join(wdi2, cow2, by = c("ccode", "year")) %>%
  arrange(ccode, year) ## Rearrange the variables
```

Using `left_join()`

Alternatively, the `left_join()` function keeps all rows from the left dataset (`wdi2` in this case) and includes matching rows from the right dataset (`cow2`). If there are no matches, the unmatched columns from the right dataset will be filled with `NA`.

```r
plot(euler(c("WDI" = 3,"COW" = 3,"WDI&COW" = 1)),
     quantities = FALSE,fills=c("maroon","white","maroon"))
```



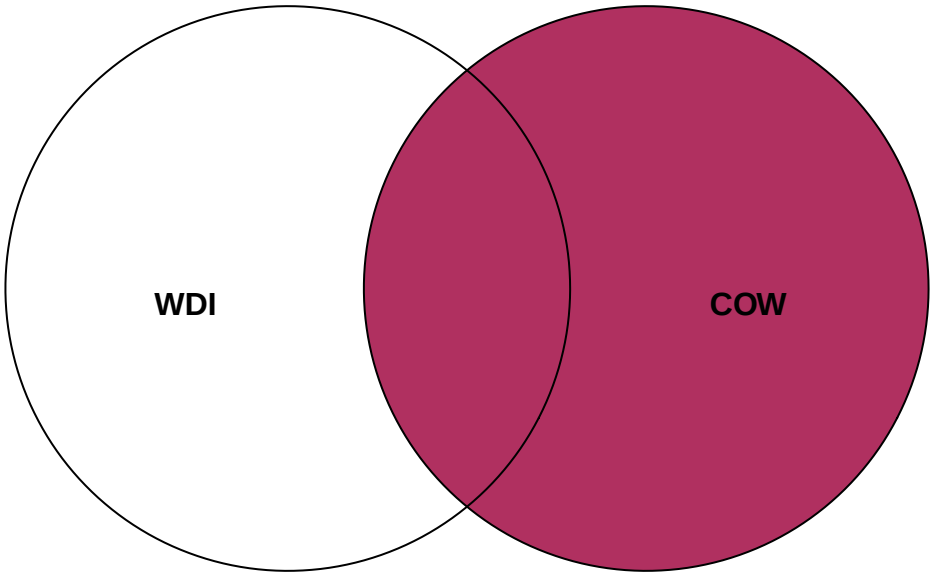
Now, let's apply `left_join()` t merge the datasets:

```
# Merge by 'ccode', 'year' using left_join()
left_data <- left_join(wdi2, cow2, by = c("ccode", "year"))
```

Using `right_join()`

The `right_join()` function works similarly to `left_join()`, but it keeps all rows from the right dataset (`cow2`) and includes matching rows from the left dataset (`wdi2`). If there are no matches in the left dataset, the unmatched columns will be filled with `NA`.

```
plot(euler(c("WDI" = 3,"COW" = 3,"WDI&COW" = 1)),
     quantities = FALSE, fills=c("white","maroon","maroon"))
```




```
# Merge by 'ccode', 'year' using right_join()
right_data <- right_join(wdi2, cow2, by = c("ccode", "year"))
```

Conclusion

In this walkthrough, we covered how to use `rbind()`, `cbind()`, and how to merge datasets using `full_join()`, `inner_join()`, `left_join()`, and `right_join()`. These functions help combine data efficiently when shared identifiers are present, making tidy and well-structured data essential for successful merges.

While the datasets here were set up for easy merging by country and year, this won't always be the case. For regional or urban data, a one-to-many merge might be needed, where smaller units (like cities) are matched with larger ones (like countries). Regardless of the structure, successful merging always depends on shared variables. We'll continue practicing these concepts in future modules and assignments.

We'll keep building on these skills through groupwork and homework assignments, where you'll refine your ability to tidy, reshape, and merge data.