Data wrangling and merging III

Complexities in analyzing conflicts: Data wrangling and data management in R

Cosima Meyer

November 13, 2019

Preparation: Remove documents, install packages, load data sets

```
# Remove all objects from R's memory
rm(list=ls())
```

```
# We will use the following code to install all packages
packages <- c("tidyverse", # to load tidyverse
              "dplyr",# to load dplyr
              "haven", # to read in .dta files (also SAS and SPSS)
              "WDI", # to access World Bank data
              "feather", # to compress large-scale datasets
              "countrycode", # to generate country/continent/region codes
              "skimr", # for summary statistics
              "devtools" # to install packages from Github
              )

# Install uninstalled packages
lapply(packages[!(packages %in% installed.packages())], install.packages)

# Load all packages to library
lapply(packages, library, character.only = TRUE)
```

What did we learn last time?

- Tidyverse: `tidyr` and `dplyr`
- How can you have a first look at data?
  - `dataset %>% View()`
  - `dataset %>% dplyr::slice(1:4) %>% dplyr::glimpse()`
  - `dataset %>% str()`
  - `dataset %>% skimr::skim()`
  - `dataset %>% head()`
  - …
- How can you install a package?
  - `install.packages("packageName")` and `library(packageName)`
- Merging and group work
- Homework covered data cleaning (in particular also date variables) and dealing with **missing values** – what are your main take aways?

Merging

The following steps mimick the steps that we learned last time. Again, we will showcase the merging procedure with the paper by Hultman et al. (2014). One of our goals for this class is to replicate (some of) the figures in the article (Figure 1-3). To do this, we need information on **UN PKO** and **battle-related deaths**. We rely on the datasets by the UNPKO data by Jakob Kathman. and the UCDP GED Dataset

The general merging procedure follows these steps:

1. Read in data
2. (Look at data)
3. Identify unique identifier (usually country and year)
4. Merge
5. Check if everything went well

Following this guideline, **we will read in the data sets first**.

```
# Read in a .dta object
unpko <- haven::read_dta("data/CMPS Mission Totals 1990-2011.dta")
```

Since the GED data is too large for the R Studio Cloud, we will work again with a restricted dataset (only African countries and only variables that we really need). If you want to replicate the steps later by yourself for a different purpose, I also added you the full (commented) code in last week's file so that you can use it. For the purpose of showcasing the merging procedure, we will use the newly generated subsetted dataset and read it in:

```
# Read a feather object
ucdp_ged_africa_fthr <- read_feather("ucdp_ged_africa.feather")
```

As the next step, we will now have a first look at the data. This allows us to get a better idea of what we are dealing with. We will skip this step this time since we have done this last week. There are various ways how to have a first look at the data and you can look up a few approaches in last week's script.

As you remember from the Hultman et al. (2014) paper, they only focused on the African continent. We now have more countries spanning around the globe. One first step might therefore be to restrict our dataset to the African continent only. To do so, we have two possible approaches: 1) Select countries on the African continent manually, or, if we want to automatize these steps a bit, 2) rely on a pre-coded continent/region variable.

We rely on a pre-coded variable for the continent. As our dataset does not contain this information, we rely on the package `countrycode` and its function `countrycode()` to generate this information (I already loaded the package for you). You can look up the syntax of this function using `? countrycode()`.

```
unpko <- unpko %>% # We overwrite our dataset
  dplyr::mutate(region = countrycode(missioncountry, "country.name", "continent"))
```

```
## Warning in countrycode(missioncountry, "country.name", "continent"): Some values were not matched unambi
```

```
# and apply the countrycode() function to generate our new variable called "region"
```

We get the warning that "Some values were not matched unambiguously: , Kosovo, Yugoslavia". We need to code these observations manually.

To do this, we first check which cases are affected (" , " indicates that we have observations with *missings* for a country).

```
unpko %>%
  dplyr::filter(is.na(region)) # %>%
```

```
## # A tibble: 584 x 17
##    mission missioncountry missionccode  year month yearmon troop police
##    <chr>   <chr>                 <dbl> <dbl> <dbl>   <dbl> <dbl>  <dbl>
##  1 LBB     ""                       NA  1997     4   1997.    23      0
##  2 LBB     ""                       NA  1997     5   1997.    23      0
##  3 UNFOR   ""                       NA  1996     1   1996.  1999      0
##  4 UNFOR   ""                       NA  1996     2   1996.   240      0
##  5 UNMIK   Kosovo                  347  1999     6   1999.     8     37
##  6 UNMIK   Kosovo                  347  1999     7   1999.    30    368
##  7 UNMIK   Kosovo                  347  1999     8   1999.    34    897
##  8 UNMIK   Kosovo                  347  1999     9   1999.     0   1552
##  9 UNMIK   Kosovo                  347  1999    10   1999.     0   1728
## 10 UNMIK   Kosovo                  347  1999    11   1999.     0   1850
## # ... with 574 more rows, and 9 more variables: militaryobservers <dbl>,
## #   total <dbl>, total2 <dbl>, monthlytotpers <dbl>,
## #   monthlytrooppers <dbl>, monthlypolicepers <dbl>,
## #   monthlymilobspers <dbl>, numberofmissions <dbl>, region <chr>
```

```
  # View()

# As we can see, we do have observations with missings in the `missioncountry` variable.
# We will first code the `region` variable for Kosovo and Yugoslavia manually.
# We use the `ifelse()` function. The logic is as follows: `ifelse(test, yes, no)`.
# Or, in plain words: If an object fulfills a certain value/logical mode (`test`),
# then do whatever is in `yes`. If not, do whatever is in `no`.

# We will see this with the following example:
unpko <- unpko %>%
```

```r
  dplyr::mutate(region = ifelse(missioncountry == "Kosovo", "Europe", region))

# We will du this again with `Yugoslavia`.
unpko <- unpko %>%
  dplyr::mutate(region = ifelse(missioncountry == "Yugoslavia", "Europe", region))

# We will use the mission names (`mission`) to identify the mission countries.
# To do this, we will first need to look up the distinct missions.
unpko %>%
    dplyr::filter(is.na(region)) %>%
    dplyr::distinct(mission) # and select only unique missions
```

```
## # A tibble: 5 x 1
##   mission
##   <chr>
## 1 LBB
## 2 UNFOR
## 3 UNPF
## 4 UNPROFOR
## 5 UNTSO
```

```r
# These 5 missions have no countryname. We can simply look them up (or know them by heart).
#
# - LBB: ?
# - UNFOR: ? United Forces
# - UNPF: United Nation Peace Forces
# - UNPROFOR: Bosnia and Herzegovina, Croatia, the Federal Republic of Yugoslavia
#   (Serbia and Montenegro) and the former Yugoslav Republic of Macedonia
#   between Feb 1992 - March 1995
# - UNTSO: United Nations Truce Supervision Organization (UNTSO)
#
# As we see, it is hard to locate these missions geographically in a single country.
# Luckily, none of these observations seems to be directly located on the African continent.
# If we were interested in one of these specific missions, we would need to further investigate
# and make rigorous coding decisions. For our purpose, we can simply drop these observations.

# We will use the command [`drop_na()`](https://tidyr.tidyverse.org/reference/drop_na.html)
# from the `tidyr` package to drop observations with missing values.

unpko <- unpko %>%
  tidyr::drop_na(region)
```

We will now restrict our dataset to the African continent.

```r
unpko_africa <- unpko %>%
  dplyr::filter(region == "Africa")
```

UCDP GED

In a similar manner we would also look at the GED dataset. Since we've done this last week, we will skip this this time.

After having a quick look at the data, we now need to identify a unique identifier. Typical identifiers are usually a *geograpical location* and a *time variable*. For today's session we will use the information on the *country* and on the *year*. Both datasets have a country variable (`unpko_africa`: `missioncountry`, `ucdp_ged_africa_fthr`: `country`). Both variables contain full country names. Because spelling inconsistencies would lead to non-matching in the merging procedure, it is always advised to choose **unique** identifiers. We will create these identifiers with the function `countrycode()` that we've learned above. We will now generate ISO3 country codes in alphabetic. We simply replace `"continent"` (remember, we used it to generate our continent variable earlier) with `"iso3c"`.

```
# For the UNPKO dataset
unpko_africa <- unpko_africa %>%
  dplyr::mutate(ccode = countrycode(missioncountry, "country.name", "iso3c"))

# For the UCDP GED dataset
ucdp_ged_africa_fthr <- ucdp_ged_africa_fthr %>%
  dplyr::mutate(ccode = countrycode(country, "country.name", "iso3c"))
```

Since we receive no error/warning messages, everything seemed to have worked perfectly.

Both datasets have a variable called `year` that gives us information on the year.

Now we are ALMOST all set for the merging.

Last week we did not conduct one essential step that is related to our step on identifying *unique* identifiers. We need to make sure that each identifier (in our case a `ccode year` combination). only appears **once** in the dataset. In other words, we need to make sure that our identifiers (`ccode year`) uniquely identify **one** observation in each dataset.

We first double-check how many *unique* combinations we have in our datasets:

```
unpko_africa %>%
  dplyr::distinct(ccode, year)
```

```
## # A tibble: 130 x 2
##    ccode  year
##    <chr> <dbl>
##  1 BDI    2007
##  2 BDI    2008
##  3 BDI    2009
##  4 BDI    2010
##  5 BDI    2011
##  6 BDI    2012
##  7 CAF    2000
##  8 CIV    2003
##  9 CIV    2004
## 10 CAF    1998
## # ... with 120 more rows
```

```
ucdp_ged_africa_fthr %>%
  dplyr::distinct(ccode,year)
```

```
## # A tibble: 701 x 2
##    ccode  year
##    <chr> <dbl>
```

```
##   1 MLI    2013
##   2 COD    2004
##   3 COD    2007
##   4 COD    2013
##   5 SOM    2010
##   6 TCD    2003
##   7 SDN    2011
##   8 CAF    2012
##   9 ETH    2012
##  10 DZA    2012
## # ... with 691 more rows
```

As we can see, each combination appears fewer times than our complete dataset. In fact, our `unpko_africa` dataset only has only 130 unique observations (as opposed to its current size of 1450 observations). Why is this the case? The dataset is on a monthly mission country basis. One possibility to reduce our dataset to unique observations only is to aggregate our dataset (similar to the `collapse` function in Stata). We will use a combination of `dplyr::summarise()` and `group_by()` on each dataset.

```r
# aggregate it to the country-year-level
unpko_africa_agg <- unpko_africa %>%
  group_by(ccode, year) %>%
  dplyr::summarise(mission = first(mission),
         missioncountry = first(missioncountry),
         month = min(month),
         yearmon = min(yearmon),
         troop = mean(troop),
         police = mean(police),
         militaryobservers = mean(militaryobservers),
         total = mean(total),
         total2 = mean(total2),
         monthlytotpers = mean(monthlytotpers),
         monthlytrooppers = mean(monthlytrooppers),
         monthlypolicepers = mean(monthlypolicepers),
         monthlymilobspers = mean(monthlymilobspers),
         numberofmissions = mean(numberofmissions)
)

# aggregate it to the country-year-level
ucdp_ged_africa_fthr_agg <- ucdp_ged_africa_fthr %>%
  group_by(ccode, year) %>%
  dplyr::summarise(
         country = first(country),
         date_start = min(date_start),
         date_end = max(date_end),
         deaths_a = sum(deaths_a),
         deaths_b = sum(deaths_b),
         deaths_civilians = sum(deaths_civilians),
         deaths_unknown = sum(deaths_unknown),
         low = sum(low),
         best = sum(best),
         high = sum(high)
)
```

We need to carefully decide whether we want to use `mean()`, `max()`, `min()`, or `sum()`.

NOW we are all set for the merging.

We will use again the tidyverse and more specifically the package dplyr (see code on your cheat sheet for combining datasets). It offers various operators for merging datasets. The most frequently used are left_join(), right_join(), inner_join(), and full_join(). It always takes the arguments in the following order: left_join(dataset1, dataset2, by="common_identifier"). If you come from a Stata background, you might remember the merge results _merge==1 (from master dataset) and _merge==2 (from using dataset). You may also remember the different merge operators (m:1, 1:m, m:m). tidyverse does not differentiate between a *master* and a *using dataset*. Instead it joins datasets from left or right. If we execute a left_join() we would then logically only keep matching rows from dataset1 (which is left). An inner_join() command keeps only rows that match both datasets whereas a full_join() keeps all observations.

Let's think logically what we get and what we need. If we use the command left_join(), which data do we keep?

```
combined <- ucdp_ged_africa_fthr_agg %>% # generate new dataset
  left_join(unpko_africa_agg, by = c("year", "ccode"))
```

In this case we only keep the countries that had peacekeeping operations (and are present in the unpko dataset). Given that we want to replicate the figures from the paper, this basis sounds plausible. If your research question requires a different data basis, you need to use a different merging command.

Hands on exercise

Now it's your turn. Get together in your groups and follow the next steps. These steps are based on the merging procedure described above.

1. Present the following information to your group mates briefly: What is you (tentative) research question? What is your dependent variable? What is your independent variable? What is the data basis you plan to use? *(max. 5 minutes in total)*
2. I've uploaded (hopefully) all datasets that you will need. Please read in all datasets that you need. The idea is that you work together and generate a dataset that contains **all** information so that everyone of you can easily pick the pieces that s/he needs. *(max. 15 minutes in total)*

For a better overview, here's a short info on the datasets that I've prepared for you (you can download all datasets from ILIAS):

- UCDP GED (ged191.xlsx) – this is the full GED dataset
- Correlates of War for inter-state wars (Inter-StateWarData_v4.0.csv)
- Global Internal Displacement Database (idmc_displacement_all_dataset.xlsx)
- Religion and Armed Conflict (RELAC) data (Relac-JCRrep.xlsx)
- UNPKO by Kathman (CMPS Mission Totals 1990-2011.dta)
- UCDP Termination on conflict-level (ucdp-term-conf-2015.xlsx)
- State of Emergency Project (STEM_II.xlsx)
- ICOW Territorial Claims Data Set (ICOWdata.zip) - you need to download and unzip it first.
- SVAC data – SVAC Dataset CONFLICT-YEAR (Version 2.0)-November 2019 (SVAC_conflictyears_1989-2015.xlsx)
- CIRI (CIRI Data 1981_2011 2014.04.14.csv)
- Coca cultivation (RPT_CultivosIlicitos_2019-11-12–102958.xlsx)
- PRIO PETRODATA (Petrodata_offshore_V1.2.xlsx and Petrodata_Onshore_V1.2.xlsx)
- PRIO DIADATA (DIADATA Excel file.xlsx)

```
# Read in the data
```

3. Have a first look at the data. *(ca. 5-10 minutes)*

```
# Have a first look at the data
```

4. Now you need to decide on a merging procedure. That means that you need to make sure that you know which variable is your common identifier – do you need to recode something? If you have more than one dataset, which merging steps are most logical? *(ca. 10-30 minutes)*

```
# Identify a common identifier (do you need to recode something?)
```

5. Once you've answered all these questions, you're ready to merge! Decide on the type of merging that you want to conduct and merge the data. *(ca. 10-20 minutes)*

```
# Merge data
```

6. Double-check if the merging worked. You may want to have a look at the data and see if your new dataset looks good. *(ca. 10-20 minutes)*

```
# Double-check if merging worked
```

7. If you are already this far, you can now start exploring your data descriptively more in-depth. *(open end)*

```
# Explore your data
```