Data wrangling and merging IV

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
              "lubridate", # to deal with date variable
              "zoo" # for data wrangling
              )

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

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

Tricky merging tasks

In this session, we will showcase two more advanced data wrangling and merging examples:

1. Generating a panel-like data structure
2. Generating post-conflict data

1. Generating a panel-like data structure

We will first read in the data. We will again use our restricted UCDP GED dataset.

```
ucdp_ged_africa_fthr <- read_feather("ucdp_ged_africa.feather")
```

Since it takes a very long time to run all the commands and would logically also exceed the working capacity of our R Studio Server, we restrict our sample to a random subset. If you wish to replicate all steps by yourself, just comment the following code lines and execute the rest.

```
# Randomly sample 1000 observations.
ucdp_ged_africa_fthr <- sample_n(ucdp_ged_africa_fthr, 1000)
```

As you might have recognized, the UCDP data's format is event-based. This means, the unit of analysis is based on a start and an end date. We can also see this if we have a look at the data:

```
ucdp_ged_africa_fthr %>%
  head(5)
```

```
## # A tibble: 5 x 12
##   conflict_new_id country  year date_start          date_end
##             <dbl> <chr>   <dbl> <dttm>              <dttm>
## 1             287 Burundi  2000 2000-01-19 00:00:00 2000-01-19 00:00:00
## 2             386 Algeria  2003 2003-10-17 00:00:00 2003-10-17 00:00:00
## 3             573 Maurit~  2009 2009-06-23 00:00:00 2009-06-23 00:00:00
## 4             297 Nigeria  2017 2017-08-07 00:00:00 2017-08-07 00:00:00
## 5           11346 Libya    2011 2011-10-15 00:00:00 2011-10-15 00:00:00
## # ... with 7 more variables: deaths_a <dbl>, deaths_b <dbl>,
## #   deaths_civilians <dbl>, deaths_unknown <dbl>, low <dbl>, best <dbl>,
## #   high <dbl>
```

As we can see, the time frames are usually rather short and on a daily basis, sometimes they cover several days, weeks or months. We need to generate a panel format dataset to capture all this information. This example is leaned on the code here.

```
# Generate unique episode ids for each conflict episode
ucdp_ged_africa_fthr$episode_id <-
  as.factor(
    paste(
      ucdp_ged_africa_fthr$conflict_new_id,
      ucdp_ged_africa_fthr$date_start,
      sep = "-"
    )
  )
levels(ucdp_ged_africa_fthr$episode_id) <-
  seq(1, nlevels(ucdp_ged_africa_fthr$episode_id))
```

If there are NAs in the date_end variable, we would need to fill these in. But since there are no NAs (the latest date ist 2018-12-31), we do not need to do anything here.

```
#  Check if date_end is NA
sum(is.na(ucdp_ged_africa_fthr$date_end))
```

```
## [1] 0
```

```
# Check for maximum date
ucdp_ged_africa_fthr %>%
  dplyr::summarise(max(date_end))
```

```
## # A tibble: 1 x 1
##   `max(date_end)`
##   <dttm>
## 1 2018-12-28 00:00:00
```

Now we will need to extract years and months from our date variables.

```
# Generate year based on date_start and date_end
ucdp_ged_africa_fthr <- ucdp_ged_africa_fthr %>%
  dplyr::mutate(year = lubridate::year(date_start),
                year2 = lubridate::year(date_end))

# Generate month based on date_end and date_end
ucdp_ged_africa_fthr <- ucdp_ged_africa_fthr %>%
  dplyr::mutate(month = lubridate::month(date_start),
```

```
                month2 = lubridate::month(date_end))

# Add day variable (only including 1) to the dataset
ucdp_ged_africa_fthr <- ucdp_ged_africa_fthr %>%
  dplyr::mutate(day = rep(1, nrow(ucdp_ged_africa_fthr)))

# Generate combined date variable with the package lubridate
ucdp_ged_africa_fthr$date <-
  with(ucdp_ged_africa_fthr, lubridate::ymd(sprintf('%04d%02d%02d', year, month, day)))
ucdp_ged_africa_fthr$date2 <-
  with(ucdp_ged_africa_fthr, lubridate::ymd(sprintf('%04d%02d%02d', year2, month2, day)))
```

Now we can expand the dataset to a panel that runs from the conflict's start year-month to the last year-month observed in the data.

```
# Write function to expand the data
paneldata <-
  function(x)
    with(x, data.frame(episode_id, date = seq(date, date2, by = "month")))

# Apply the function to our data (do it row-wise)
ucdp_ged_africa_fthr2 <- ucdp_ged_africa_fthr %>%
  rowwise() %>%
  do(paneldata(.))
```

We now received a new dataset that contains the variables `date` and `episode_id`.

In a last step we can then merge our data on these variables (they are present in both datasets).

```
# Check for duplicates
ucdp_ged_africa_fthr2 %>%
  dplyr::distinct(episode_id, date)
```

```
## Source: local data frame [1,183 x 2]
## Groups: <by row>
##
## # A tibble: 1,183 x 2
##    episode_id date
##    <fct>      <date>
##  1 192        2000-01-01
##  2 528        2003-10-01
##  3 934        2009-06-01
##  4 228        2017-08-01
##  5 24         2011-10-01
##  6 555        2011-07-01
##  7 338        2007-06-01
##  8 869        1993-05-01
##  9 293        2016-09-01
## 10 537        2004-06-01
## # ... with 1,173 more rows
```

```
ucdp_ged_africa_fthr %>%
  dplyr::distinct(episode_id, date)
```

```
## # A tibble: 989 x 2
##    episode_id date
##    <fct>      <date>
```

```
##  1 192        2000-01-01
##  2 528        2003-10-01
##  3 934        2009-06-01
##  4 228        2017-08-01
##  5 24         2011-10-01
##  6 555        2011-07-01
##  7 338        2007-06-01
##  8 869        1993-05-01
##  9 293        2016-09-01
## 10 537        2004-06-01
## # ... with 979 more rows
```

As we can see, we do have duplicates. We thus need to aggregate (as we've learned in our first session today).

```
# Aggregate
ucdp_ged_africa_fthr2 <- ucdp_ged_africa_fthr2 %>%
  dplyr::distinct(episode_id, date)

ucdp_ged_africa_fthr <- ucdp_ged_africa_fthr %>%
  group_by(episode_id, date) %>%
  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)
)

# Merge both datasets
ucdp_ged_combined <- ucdp_ged_africa_fthr2 %>%
  dplyr::left_join(ucdp_ged_africa_fthr, by=c("date", "episode_id"))

ucdp_ged_combined <- ucdp_ged_combined %>%
  dplyr::arrange(episode_id, date)
```

```
ucdp_ged_combined %>%
  head(20)
```

```
## Source: local data frame [20 x 12]
## Groups: <by row>
##
## # A tibble: 20 x 12
##    episode_id date       country date_start          date_end
##    <fct>      <date>     <chr>   <dttm>              <dttm>
##  1 1          1993-07-01 South ~ 1993-07-01 00:00:00 1993-07-31 00:00:00
##  2 2          1994-05-01 South ~ 1994-05-03 00:00:00 1994-05-03 00:00:00
##  3 3          1998-02-01 Sierra~ 1998-02-15 00:00:00 1998-06-30 00:00:00
##  4 3          1998-03-01 <NA>    NA                  NA
##  5 3          1998-04-01 <NA>    NA                  NA
##  6 3          1998-05-01 <NA>    NA                  NA
##  7 3          1998-06-01 <NA>    NA                  NA
```

```
##  8 4          1998-03-01 Sierra~ 1998-03-01 00:00:00 1998-03-26 00:00:00
##  9 5          1999-01-01 Sierra~ 1999-01-09 00:00:00 1999-01-09 00:00:00
## 10 6          2002-02-01 Nigeria 2002-02-11 00:00:00 2002-02-15 00:00:00
## 11 7          2012-07-01 Nigeria 2012-07-22 00:00:00 2012-07-22 00:00:00
## 12 8          2010-09-01 Nigeria 2010-09-26 00:00:00 2010-09-27 00:00:00
## 13 9          2012-01-01 Somalia 2012-01-17 00:00:00 2012-01-17 00:00:00
## 14 10         2013-06-01 Egypt   2013-06-29 00:00:00 2013-06-29 00:00:00
## 15 11         2013-07-01 Egypt   2013-07-05 00:00:00 2013-07-05 00:00:00
## 16 12         2013-08-01 Egypt   2013-08-06 00:00:00 2013-08-06 00:00:00
## 17 13         2013-11-01 Egypt   2013-11-22 00:00:00 2013-11-22 00:00:00
## 18 14         2014-02-01 Egypt   2014-02-14 00:00:00 2014-02-14 00:00:00
## 19 15         2015-09-01 Centra~ 2015-09-28 00:00:00 2015-09-28 00:00:00
## 20 16         2014-07-01 South ~ 2014-07-10 00:00:00 2014-07-10 00:00:00
## # ... with 7 more variables: deaths_a <dbl>, deaths_b <dbl>,
## #   deaths_civilians <dbl>, deaths_unknown <dbl>, low <dbl>, best <dbl>,
## #   high <dbl>
```

As we can see, it worked. However, we have missings that we need to fill. We will use the `na.locf()` function from the `zoo` package. It carries the last observation forward and overwrites NAs but stops once a non-NA observation is reached.

```
ucdp_ged_combined <- ucdp_ged_combined %>%
  group_by(episode_id) %>%
  dplyr::mutate(
    country = na.locf(country),
    date_start = na.locf(date_start),
    date_end = na.locf(date_end),
    deaths_a = na.locf(deaths_a),
    deaths_b = na.locf(deaths_b),
    deaths_civilians = na.locf(deaths_civilians),
    deaths_unknown = na.locf(deaths_unknown),
    low = na.locf(low),
    best = na.locf(best),
    high = na.locf(high))
```

```
## Warning: Grouping rowwise data frame strips rowwise nature
```

If we sort our panel data now based on a country-year level, we will see that we still have duplicates in our dataset. This is because we generated our panel data based on the episode-level. It is plausible that multiple conflict episodes happen within one single country at one single day/week/month/year. If we want to further proceed with our data, we will then (again) need to aggregate our data – but this time to a country-year level.

We will follow the steps discussed in today's first session and last week's Friday session.

We have our country variable which contains the full name of the country. We will recode this country name in an ISO3 code using the `countrycode()` function. To execute this command on the full dataset may take some time because the dataset is sufficiently larger this time.

```
# Generate a ccode variable
ucdp_ged_combined <- ucdp_ged_combined %>% # We overwrite our dataset
  dplyr::mutate(ccode = countrycode(country, "country.name", "iso3c"))
```

In a next step, we need to generate again the `year` variable.

```
# Generate year based on date
ucdp_ged_combined <- ucdp_ged_combined %>%
  dplyr::mutate(year = lubridate::year(date))
```

We now can use the `year` and `ccode` variable to aggregate the dataset on a country-year-level.

```
ucdp_combined_agg <- ucdp_ged_combined %>%
  group_by(ccode, year) %>%
  dplyr::summarise(
          country = first(country),
          date_start = min(date_start),
          date_end = max(date_end),
          deaths_a = mean(deaths_a),
          deaths_b = mean(deaths_b),
          deaths_civilians = mean(deaths_civilians),
          deaths_unknown = mean(deaths_unknown),
          low = mean(low),
          best = mean(best),
          high = mean(high)
)
```

We need to pay attention to how we best aggregate the data. Does the sum of the deaths make sense? Or is it rather the mean? These are questions that need to be carefully answered every time we conduct a new aggregation.

Once we have the panel data format dataset, we can proceed with merging steps as we have done before.

2. Generating post-conflict data

UCDP offers us conflict periods but no post-conflict periods. We therefore need to code these post-conflict period ourselves. The following steps show you one way how to do this. For this example here, we will proceed with our previously generated `ucdp_combined_agg` data.

Ideally, we would have a dataset that contains information on all the countries and the years in our targeted sample. The World Bank is usually one go-to. Otherwise, we could also manually create this information with two separate vectors (`ccode` and `years`) that we would then combine to a `data.frame`. Remember, we have done this in our first session.

To showcase how we do it, we will manually create the required information.

To do so, we will first turn back to our `ucdp_combined_agg` dataset and see which countries and years are included in our dataset. We will do it the same way as done before – by looking at the distinct countries and years. But this time, we will store the information in a new object – called `countries` and `years` – repsectively. We need the `pull()` function to save our data in a new vector.

```
countries <- ucdp_combined_agg %>%
  group_by(ccode) %>%
  dplyr::distinct(ccode) %>%
  dplyr::pull()
```

```
years <- ucdp_combined_agg %>%
  group_by(year) %>%
  dplyr::distinct(year) %>%
  dplyr::arrange(year) %>%
  dplyr::pull()
```

As we can see, the `countries` variable covers 35 countries in Africa and the `years` variable spans from 1989 to 2018. We will use this information to generate our new data.frame.

```
# Generate a vector that contains all the years
years2 <- rep(years, length(countries))
```

```
# Generate a vector that contains all the countries
countries2 <- sort(rep(countries, length(years)))

# Combine `years2` and `countries2`
pseudo_data <- data.frame(years2, countries2)
```

Before we move on, we will quickly rename the variables.

```
pseudo_data <- pseudo_data %>%
  dplyr::rename(year = years2,
                ccode = countries2) %>%
  # We also save our ccode variable as a character
  dplyr::mutate(ccode = as.character(ccode))
```

In a next step, we need to generate a conflict variable in our `ucdp_combined_agg` dataset.

```
# Generate conflict variable
ucdp_combined_agg <- ucdp_combined_agg %>%
  dplyr::mutate(conflict = 1)
```

We will first check for unique observations.

```
# Double check for unique observations
ucdp_combined_agg_test <- ucdp_combined_agg %>%
  dplyr::distinct(ccode, year)

pseudo_data_test <- pseudo_data %>%
  dplyr::distinct(ccode, year)
```

Since we have no duplicates, we can proceed and merge both `ucdp_combined_agg` and our `pseudo_data` dataset.

```
ucdp_pseudo <- pseudo_data %>%
  dplyr::left_join(ucdp_combined_agg, by = c("ccode", "year"))
```

As you can see, we have joined based on our `pseudo_data` dataset. We therefore have various NAs in our dataset. This is absolutely plausible since these periods are either pre-conflict or post-conflict. In a next step, we will now proceed and label these periods as being either pre- or post-conflict.

1. Replace conflict with 0 if it is NA

```
# Replace conflict with 0 if it is NA
ucdp_pseudo <- ucdp_pseudo %>%
  dplyr::mutate(conflict = ifelse(is.na(conflict), 0, 1))

# Check if it worked
ucdp_pseudo %>%
  dplyr::count(conflict)
```

```
## # A tibble: 2 x 2
##   conflict     n
##      <dbl> <int>
## 1        0   839
## 2        1   331
```

Since we built our combined dataset based on the UCDP data, we actually do not need this step. But let's say you want to replicate these steps with the World Bank data, we will need these steps. I therefore included them here for you.

```r
# Only keep countries where a civil war occurred at least once
ucdp_pseudo <- ucdp_pseudo %>%
  group_by(ccode) %>%
  dplyr::filter(any(conflict == 1))
```

Now that we only have countries in our sample that have for sure the potential to become post-conflict at one point, we can proceed with the next steps. We will now generate the post-conflict variable.

```r
# Generate post-conflict variable
ucdp_pseudo <- ucdp_pseudo %>%
  dplyr::mutate(postconflict = ifelse(conflict==1, 0, 1))
```

This variable is quasi the direct opposite of our conflict variable (whenever conflict is 0, post-conflict becomes 1). As you may now easily realize, this variable is not very precise. Burkina Faso only becomes a conflict case in our sample in 2018 but is flagged already post-conflict before. We therefore need to define if this post-conflict variable identifies a post-conflict or a pre-conflict period.

```r
# Define if this post-conflict variable is a post-conflict
# or a pre-conflict

# Identify first row of group
ucdp_pseudo <- ucdp_pseudo %>%
  dplyr::mutate(first_ind = as.numeric(!duplicated(ccode)))

# Replace first_ind with NA if it is 0
ucdp_pseudo <- ucdp_pseudo %>%
  dplyr::mutate(first_ind = ifelse(first_ind == 0, NA, 1))

# First_ind becomes 0 if post-conflict is 0
# (we need this to identify non-post-conflict cases)
ucdp_pseudo <- ucdp_pseudo %>%
  dplyr::mutate(first_ind = ifelse(postconflict == 0, 0, first_ind))

# Now we can fill first_ind with na.locf() until first_ind becomes 0
ucdp_pseudo <- ucdp_pseudo %>%
  dplyr::mutate(first_ind = na.locf(first_ind))

# This variable is essentially our
# pre-conflict variable and we will rename it
ucdp_pseudo <- ucdp_pseudo %>%
  dplyr::rename(preconflict = first_ind)

# But it still has some problems
# Sometimes preconflict==1 and postconflict ==1
# we therefore need to replace it
ucdp_pseudo <- ucdp_pseudo %>%
  dplyr::mutate(postconflict = ifelse(preconflict==1, 0, postconflict))
```

Now that we have our `postconflict` and `preconflict` variables we will create a post-conflict. This count variable basically counts each post-conflict year and starts again with 1 if a country falls back into the conflict. This variable can be helpful when you want to analyze a sequential development. We will use the `with()` function in the following and rely on code in base R.

```r
# Generate post-conflict count
ucdp_pseudo$postconflict_count <-
  with(ucdp_pseudo, ave(postconflict == 1, cumsum(postconflict == 0), FUN = cumsum))
```

In a last step, we will check for inconsistencies to make sure that everything worked well.

```
# Check for inconsistencies
# Generate subset
try <- ucdp_pseudo %>%
  select(ccode, year, preconflict, conflict, postconflict)

# Test for overlaps with preconflict
try %>%
  dplyr::mutate(test1 = ifelse(preconflict == 1 &
                                 conflict == 1 | preconflict == 1 & postconflict == 1, 1, 0)) %>%
  dplyr::count(test1)
```

```
## # A tibble: 39 x 3
## # Groups:   ccode [39]
##    ccode test1     n
##    <chr> <dbl> <int>
##  1 AGO       0    30
##  2 BDI       0    30
##  3 BFA       0    30
##  4 CAF       0    30
##  5 CIV       0    30
##  6 CMR       0    30
##  7 COD       0    30
##  8 COG       0    30
##  9 DZA       0    30
## 10 EGY       0    30
## # ... with 29 more rows
```

```
# Test for overlaps with conflict
try %>%
  dplyr::mutate(test2 = ifelse(conflict == 1 &
                                 preconflict == 1 | conflict == 1 & postconflict == 1, 1, 0)) %>%
  dplyr::count(test2)
```

```
## # A tibble: 39 x 3
## # Groups:   ccode [39]
##    ccode test2     n
##    <chr> <dbl> <int>
##  1 AGO       0    30
##  2 BDI       0    30
##  3 BFA       0    30
##  4 CAF       0    30
##  5 CIV       0    30
##  6 CMR       0    30
##  7 COD       0    30
##  8 COG       0    30
##  9 DZA       0    30
## 10 EGY       0    30
## # ... with 29 more rows
```

```
# Test for overlaps with postconflict
try %>%
  dplyr::mutate(test3 = ifelse(postconflict==1 &
                                 preconflict == 1 | postconflict == 1 & conflict == 1, 1, 0)) %>%
  dplyr::count(test3)
```

```
## # A tibble: 39 x 3
## # Groups:   ccode [39]
##    ccode test3     n
##    <chr> <dbl> <int>
##  1 AGO       0    30
##  2 BDI       0    30
##  3 BFA       0    30
##  4 CAF       0    30
##  5 CIV       0    30
##  6 CMR       0    30
##  7 COD       0    30
##  8 COG       0    30
##  9 DZA       0    30
## 10 EGY       0    30
## # ... with 29 more rows
```

We are all set :-)

For next session

Please make sure that you have your dataset in a merged format.

We will then continue working with your merged datasets and learn how to create **visualizations with `ggplot2`** next session (**November 22, 2019**).

The **hackathon on Wednesday (November 27, 2019)** is dedicated to your group projects. It allows you to work concentrated on your projects, consult me individually for help and address open questions.