Data wrangling and merging I & II

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?

- Loading datasets in various formats (.dta, .RData, .xls/.xlsx, .csv)
- A format that compresses data (.feather)
- A glimpse of `tidyverse`

Tidyverse

The `tidyverse` is a "universe" that includes several packages that all follow the tidyverse logic when it comes to dealing, handling and wrangling the data.

The following figure is taken from this blog post and shows a part of the tidyverse – but this is not exhaustive. Instead of covering all packages (which will not be possible in this course), we will focus on the major components of tidyverse.

Before we delve more into tidyverse, let's briefly look at the hands on exercise that we had at the end of last session:

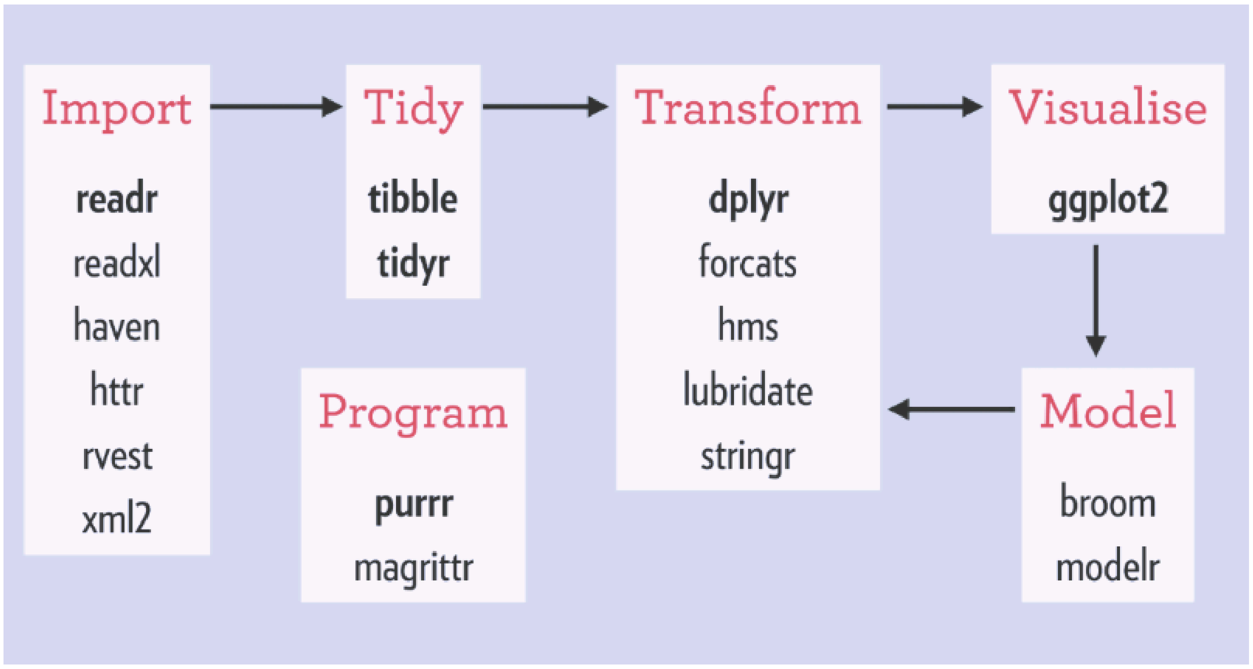

Figure 1: tidyverse

Hands on exercise

0. Read in the UNPKO dataset.

```
unpko <- read_dta("data/CMPS Mission Totals 1990-2011.dta")
```

1. Which countries are included in the dataset?
2. How many missions were on average deployed during the years?
3. Now we will use the `ucdp_dyadic` dataset. Reload the dataset (remember, the file is called "ucdp_dyadic_191.RData").

3.* Bonus with the `unpko` dataset: How many countries are covered in the dataset?

4. Subset the dataset to region == 3 only. Save the result in an object called `ucdp_dyad_region`.
5. Which (unique) countries are in the location variable? Can you make any sense of what the "3" in region stands for?

Important functions of `dplyr` and `tidyr` for some first descriptive analysis and data wrangling

Generate new variables: `mutate()`

If we want to generate a new variable, we use the function `mutate()`.

Let's say we want to generate a variable that combines `troop` and `police`. We will call this variable `armed_forces`.

```
unpko <- unpko %>%
  dplyr::mutate(armed_forces = troop + police)
```

We can also use mutate for more difficult operations and combine it with other functions – but more on this below.

Descriptive statistics (`mean()`, `max()`, `min()`, ...)

Assume we want to know how many `troops` we have on average per mission country.

```
unpko %>%
  dplyr::group_by(missioncountry) %>%
  dplyr::summarise(troop_avg = mean(troop))
```

```
## # A tibble: 38 x 2
##    missioncountry           troop_avg
##    <chr>                        <dbl>
##  1 ""                           NA
##  2 Afghanistan                   0
##  3 Angola                       NA
##  4 Bosnia and Herzegovina        1.71
##  5 Burundi                    1317.
##  6 Cambodia                     NA
##  7 Central African Republic    927.
##  8 Chad                       1298.
##  9 Costa Rica                   NA
## 10 Croatia                    4411.
## # ... with 28 more rows
```

Or let's say we want to know the maximum of military observers in South Sudan:

```
unpko %>%
  dplyr::filter(missioncountry == "South Sudan") %>%
  dplyr::summarise(max_milobs = max(militaryobservers))
```

```
## # A tibble: 1 x 1
##   max_milobs
##        <dbl>
## 1        207
```

Summary statistics: `skim()`

If we want a more comprehensive overview, the `skimr` package and the `skim()` funciton provides a nice summary statistic.

```r
skimr::skim(unpko)
```

```
## Skim summary statistics
##  n obs: 4414
##  n variables: 17
##
## -- Variable type:character ------------------------------------------------------------
##        variable missing complete    n min max empty n_unique
##         mission       0     4414 4414   3   9     0       71
##  missioncountry       0     4414 4414   0  32   343       38
##
## -- Variable type:numeric ------------------------------------------------------------
##           variable missing complete    n     mean       sd      p0
##       armed_forces     278     4136 4414  3313.15  5550.18    0
##  militaryobservers     278     4136 4414   113.4    152.94    0
##       missionccode     343     4071 4414   521.06   189.71   41
##              month       0     4414 4414     6.46     3.44    1
##  monthlymilobspers     278     4136 4414  1984.69   544.79  738
##  monthlypolicepers     278     4136 4414  6970.35  4157.04 1020
##     monthlytotpers     138     4276 4414 59099.59 29486.68 9924
##    monthlytrooppers     278     4136 4414 51729.48 25151.71 9053
##   numberofmissions       0     4414 4414    16.97     2.87    7
##             police     278     4136 4414   388.1    864.42    0
##              total     139     4275 4414  3357.08  5566.87    1
##             total2     139     4275 4414  3357.08  5566.87    1
##              troop     278     4136 4414  2925.06  5196.86    0
##               year       0     4414 4414  2001.8      6.24 1990
##            yearmon       0     4414 4414  2001.87     6.24 1990.01
##       p25      p50      p75     p100     hist
##      7      926     4106.75 39285
##      2       44      192     1039
##    372      540      652      860
##      3        6        9       12
##   1629     2087     2353     2921
##   2940     6754    10629    14703
##  35546    63928    84309    1e+05
##  28932    58571    74277.5  86571
##     16       17       19       22
##      0        6      232.5   5511
##    130     1035     4144    39922
##    130     1035     4144    39922
##      0      223     3799.25 38614
##   1997     2002     2007     2012
##   1997.01  2002.07  2007.1   2012.07
```

Generate some proportional overview: `count()`

`count()` allows us to generate a quasi proportion table for the different values of our variables. We will use the `count()` function as a way to count the number of observations per missioncountry

```r
unpko %>%
  dplyr::count(missioncountry)
```

```
## # A tibble: 38 x 2
##    missioncountry               n
##    <chr>                    <int>
##  1 ""                         343
##  2 Afghanistan                118
##  3 Angola                     133
##  4 Bosnia and Herzegovina      82
##  5 Burundi                     97
##  6 Cambodia                    24
##  7 Central African Republic    29
##  8 Chad                        41
##  9 Costa Rica                  25
## 10 Croatia                     42
## # ... with 28 more rows
```

And we now want to sort them so that we've the highest value first.

```
unpko %>%
  dplyr::count(missioncountry, sort = TRUE)
```

```
## # A tibble: 38 x 2
##    missioncountry     n
##    <chr>          <int>
##  1 ""               343
##  2 Syria            275
##  3 Cyprus           271
##  4 India            271
##  5 Lebanon          271
##  6 Morocco          251
##  7 Iraq             229
##  8 Georgia          189
##  9 Haiti            167
## 10 Kosovo           158
## # ... with 28 more rows
```

As we can see, 343 missions have no mission country. Why is this the case? We will deal with this later once we're looking more closely into the merging procedure.

Sort dataset: Combination of `select()` and `everything()`

Sometimes we want to reorder the variables in a dataset. Let's say we want the `country` and the `year` first, but keep everything else as it is we would write:

```
unpko <- unpko %>%
  dplyr::select(missioncountry, year, everything())
```

Merging

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**.

Remember, we used the `haven` package and its `read_dta()` function to read in .dta data.

```r
# Read in a .dta object
unpko <- 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 will also add you the full (commented) code below so that you can use it.

```r
# # Read in data
# ucdp_ged <- read_excel("ged191.xlsx") # loaded as ucdp_ged (takes a while)
#
# # The variable "country" gives us information on the country
# ucdp_ged <- ucdp_ged %>% # We overwrite our dataset
#   dplyr::mutate(region = countrycode(country, "country.name", "continent"))
# # and apply the countrycode() function to generate
# # our new variable called "region"
#
# # We get the following message: Some values were not matched unambiguously:
# # Yemen (North Yemen)
#
# # Since this country is not part of the African continent, we can ignore it.
#
# # Restrict to the African continent
# ucdp_ged_africa <- ucdp_ged %>%
#   dplyr::filter(region == "Africa")
#
# # We also have so many information in this dataset but we do not need all of them.
# # We therefore just keep the variables that we really need using the select command
# ucdp_ged_africa <- ucdp_ged_africa %>%
#   dplyr::select(country, year, date_start, date_end, deaths_a, deaths_b,
#   deaths_civilians, deaths_unknown, low, best, high)
# # here we select the variables that we really need
#
# # Save a data.frame as a feather object
# write_feather(ucdp_ged_africa, "ucdp_ged_africa.feather")
# # "data.feather" stands for the path and file name,
# # "df_combined" is our data.frame
```

We will use the newly generated subsetted dataset and read it in:

```r
# 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.

UNPKO

Which variables are included in the dataset?

```r
names(unpko)
```

```
##  [1] "mission"            "missioncountry"     "missionccode"
##  [4] "year"               "month"              "yearmon"
##  [7] "troop"              "police"             "militaryobservers"
## [10] "total"              "total2"             "monthlytotpers"
## [13] "monthlytrooppers"   "monthlypolicepers" "monthlymilobspers"
## [16] "numberofmissions"
```

Alternatively, we can also have a `glimpse()` at the data:

```r
unpko %>% # address dataset
  dplyr::slice(1:4) %>% # take row 1-4
  dplyr::glimpse() # get a glimpse
```

```
## Observations: 4
## Variables: 16
## $ mission           <chr> "BINUB", "BINUB", "BINUB", "BINUB"
## $ missioncountry    <chr> "Burundi", "Burundi", "Burundi", "Burundi"
## $ missionccode      <dbl> 516, 516, 516, 516
## $ year              <dbl> 2007, 2007, 2007, 2007
## $ month             <dbl> 1, 2, 3, 4
## $ yearmon           <dbl> 2007.01, 2007.02, 2007.03, 2007.04
## $ troop             <dbl> 0, 0, 0, 0
## $ police            <dbl> 11, 11, 11, 11
## $ militaryobservers <dbl> 0, 0, 3, 4
## $ total             <dbl> 11, 11, 14, 15
## $ total2            <dbl> 11, 11, 14, 15
## $ monthlytotpers    <dbl> 82003, 82751, 83071, 83271
## $ monthlytrooppers  <dbl> 70252, 70715, 70839, 71027
## $ monthlypolicepers <dbl> 9219, 9444, 9596, 9565
## $ monthlymilobspers <dbl> 2532, 2592, 2636, 2679
## $ numberofmissions  <dbl> 19, 20, 20, 20
```

Which years are covered by the dataset?

```r
unpko %>%
  dplyr::arrange(year) %>%
  dplyr::distinct(year) # select only the unique years
```

```
## # A tibble: 23 x 1
##     year
##    <dbl>
##  1  1990
##  2  1991
##  3  1992
##  4  1993
##  5  1994
##  6  1995
##  7  1996
## 8  1997
## 9  1998
## 10 1999
## # ... with 13 more rows
```

It covers 1990 - 2012.

Which countries are included in the dataset?

```r
unpko %>%
    dplyr::distinct(missioncountry) # and select only unique mission countries
```

```
## # A tibble: 38 x 1
##    missioncountry
##    <chr>
##  1 Burundi
##  2 Central African Republic
##  3 Bosnia and Herzegovina
##  4 ""
##  5 Haiti
##  6 Ivory Coast
##  7 Guatemala
##  8 Chad
##  9 Morocco
## 10 Angola
## # ... with 28 more rows
```

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()`.

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

```r
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>,
## #   monthlytroopers <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 %>%
  dplyr::mutate(region = ifelse(missioncountry == "Kosovo", "Europe", region))
```

In plain words, we ask R to check if the `missioncountry` variable has the value `"Kosovo"`. If this is the case, it should assign `"Europe"` to the variable `region`. If this is not the case, it should simply print the observation that is already present in the `region` variable.

We will du this again with `Yugoslavia`.

```
unpko <- unpko %>%
  dplyr::mutate(region = ifelse(missioncountry == "Yugoslavia", "Europe", region))
```

We can also combine both commands in one command using the OR condition (|):

```
unpko_test <- unpko %>%
  dplyr::mutate(region = ifelse(missioncountry == "Kosovo" | missioncountry == "Yugoslavia",
                                "Europe", region))
```

If we check now again, we see that we have only regions without a region code left that have no observation in `missioncountry`. Do you remember that 343 missions had no mission country assigned?

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

```
## # A tibble: 343 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 UNPF    ""                       NA  1996     1   1996.   159    115
##  6 UNPF    ""                       NA  1996     2   1996.   106    195
##  7 UNPF    ""                       NA  1996     3   1996.   228     13
##  8 UNPF    ""                       NA  1996     4   1996.   122    192
##  9 UNPF    ""                       NA  1996     5   1996.   542      0
## 10 UNPF    ""                       NA  1996     6   1996.   276      0
## # ... with 333 more rows, and 9 more variables: militaryobservers <dbl>,
## #   total <dbl>, total2 <dbl>, monthlytotpers <dbl>,
## #   monthlytroopers <dbl>, monthlypolicepers <dbl>,
## #   monthlymilobspers <dbl>, numberofmissions <dbl>, region <chr>
```

```
  # View()
```

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
```

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()` from the `tidyr` package to drop observations with missing values.

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

If we want to check if there are still missings in the `region` variable, we simply use the following code:

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

```
## # A tibble: 0 x 17
## # ... with 17 variables: mission <chr>, missioncountry <chr>,
## #   missionccode <dbl>, year <dbl>, month <dbl>, yearmon <dbl>,
## #   troop <dbl>, police <dbl>, militaryobservers <dbl>, total <dbl>,
## #   total2 <dbl>, monthlytotpers <dbl>, monthlytroopers <dbl>,
## #   monthlypolicepers <dbl>, monthlymilobspers <dbl>,
## #   numberofmissions <dbl>, region <chr>
```

Or we could also count if there are still missing values:

```
unpko %>%
  dplyr::summarise(count = sum(is.na(region)))
```

```
## # A tibble: 1 x 1
##   count
##   <int>
## 1     0
```

There are no missings left.

We will now restrict our dataset to the African continent.

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

UCDP GED

We will now look at the GED dataset in a similar fashion.

Which variables are included in the dataset?

Which years are covered by the dataset?

It covers 1989 - 2018.

Which countries are included in the dataset?

We already have only African countries in the dataset because we restricted the dataset before.

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

# For the UCDP GED dataset
```

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 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 <- unpko_africa %>% # generate new dataset
  left_join(ucdp_ged_africa_fthr, 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
```