Intro to R III

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

Cosima Meyer

November 8, 2019

Outline of a markdown file

This markdown file will be the first where we start using the markdown file structure consistently.

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
              "readxl", # to read in excel files (.xls and .xlsx)
              "readr", # to read in .csv files
              "haven", # to read in .dta files (also SAS and SPSS)
              "feather", # to compress large-scale datasets
              "WDI", # to access World Bank data
              "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)
```

```
# Set working directory
setwd() # This command sets the working directory
getwd() # This command shows you the present working directory
dir() # This command XX
```

Normally, we would import our datasets directly here. But since we're covering this aspect in today's session more in-depth, we discuss it in the next section.

Import data

As promised, we will learn how to import data. R can handle several file formats: .RData (R data format), .dta (Stata data format), .csv (comma separated values), .xls/.xlsx (Excel data format), and many more. The above listed file formats are the most common file types and we will consider them therefore here.

.RData

While UCDP offers the datasets in various formats but to showcase you how to read-in a .Rdata dataset, we will use the .Rdata format of the UCDP Dyadic Dataset. For this, we use R's built-in function `load()`.

```
load("data/ucdp_dyadic_191.RData")
```

.dta

We will use the data on UN peacekeeping personnel (from 1990-2011) by Jakob Kathman to learn how to read in .dta-file formats. We will use the `read_dta()` function from the `haven` package which is also part of the `tidyverse`.

```
# install.packages("haven")
library(haven) # Also already included in "tidyverse"

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

.csv

Here, we will use the .csv format of the UCDP Peace Agreement Dataset. We will use the `read_csv()` function from the `readr` package which is also part of the `tidyverse`.

```
# install.packages("readr")
library(readr) # Also already included in "tidyverse"

ucdp_peaceagreeement <- read_csv("data/ucdp-peace-agreements-191.csv")
```

.xls/.xlsx

We use the .xls format of the UCDP GED Dataset to learn how to load excel formats. We will use the `read_excel()` function from the `readxl` package which is also part of the `tidyverse`. Note, I slightly adjusted the original dataset and restricted it to African countries after 2009.

```
# install.packages("readxl")
library(readxl) # Also already included in "tidyverse"

# Read an excel object
ucdp_ged <- read_excel("data/ucdp_ged_subset.xls")
```

.feather

As we can see, the UCDP GED dataset is relatively large. It might be therefore a good idea to resort to a file format that allow us to read in big data sets quickly – like the newly developed file format feather. I

personally find feather very straight forward and helpful with large datasets (usually >500MB/1GB). Beyond, it's incredibly fast with over 600MB/s.

The basic commands are simple:

```
# install.packages("feather")
library(feather)

# Save a data.frame as a feather object
write_feather(ucdp_ged, "data/ucdp_ged.feather")
# "data.feather" stands for the path and file name,
# "df_combined" is our data.frame

# Read a feather object
ucdp_ged_fthr <- read_feather("data/ucdp_ged.feather")
```

As you might observe, loading the data now is significantly faster! You might observe an even greater difference when loading the full dataset with more than 150,000 observations. Once you downloaded this file and run it on your machine, you can try it out uncommenting and using the following lines of code:

```
# # Read an excel object
# ucdp_ged_original <- read_excel("data/ged191.xlsx")
#
# # Performance of readxl with a .xlsx data file
# system.time(ucdp_ged <- read_excel("data/ucdp_ged_subset.xls"))
#
# # Write as feather
# write_feather(ucdp_ged_original, "data/ucdp_ged_original.feather")
#
# # Performance of feather with a .feather data file
# system.time(ucdp_ged_fthr <- read_feather("data/ucdp_ged_original.feather"))
```

The following code checks the performance with "hard facts and numbers":

```
# Performance of readxl with a .xlsx data file
system.time(ucdp_ged <- read_excel("data/ucdp_ged_subset.xls"))
```

```
##    user  system elapsed
##   0.808   0.040   0.893
```

```
# Performance of feather with a .feather data file
system.time(ucdp_ged_fthr <- read_feather("data/ucdp_ged.feather"))
```

```
##    user  system elapsed
##   0.031   0.000   0.031
```

"Built-in" functions

Some data resources like the World Bank or Quality of Government, provide R packages that allow us to read in data automatically.

World Bank

We will use the World Bank dataset to retrieve information on the country's population size. *(Note, Hultman et al. (2014) used the disaggregated Composite Index of National Capabilities by Singer et al. (1972).)*

```r
# install.packages("WDI")
library(WDI)

wb_population <- WDI(indicator = "SP.POP.TOTL", # select the indicator
                     start = 1990, # define the start date
                     end = 2011) # define the end date
```

You can also further define which countries you want to include and add more indicators.

Quality of Government

There is also a package that allows us to directly access data from the Quality of Government. This package is on Github and therefore requires a slightly different installation.

```r
devtools::install_github("ropengov/rqog")
library(rqog)
```

We pretend that we are interested in the regime type of the countries and download all information from the Polity IV project

```r
basic <-
  read_qog(which_data = "basic", # to access the basic dataset
           # (you usually may want to access the "standard" dataset)
           data_type = "time-series") # select the data type
           # (we are interested in longitudinal data)

qog_polity <- basic %>% # call the basic dataset
  dplyr::filter(year %in% 1990:2011) %>%  # define the time frame
  dplyr::select(year, cname, p_polity2) # select the indicator
```

Bonus: Saving datasets without and with an automatic time stamp.

In particular when using automatically downloaded datasets, it might make sense to save a the most recent version of the datasets with a time stamp. The following code shows how to a) generally save datasets in R and b) how to save datasets with a time stamp.

```r
# a) save datasets as RData files in R
save(wb_population, file = "data/wbpopulation.RData")
```

We generate an automatic time stamp using `paste0()`. `paste0()` allows us to combine several strings without spaces inbetween. What you can see here is that we combine "data/wbpopulation" with the current date (provided by `Sys.Date()`) and add the extension ".RData".

```r
# b) save datasets as RData files in R with automatic time stamps
name_wb <- paste0("data/wbpopulation-", Sys.Date(), ".RData")
name_wb
```

```
## [1] "data/wbpopulation-2019-11-07.RData"
```

We save this information in the object `name_wb` and use it so save wb_population as done above.

```r
save(wb_population, file = name_wb)
```

Hands on exercise

I provided you with three additional datasets (see below) – load these datasets by using the functions that we discussed above.

Political Terror Scale (PTS-2019.dta)

National Elections Across Democracy and Autocracy (NELDA) (NELDA.csv)

Electoral Contention and Violence (ECAV) (ECAV_V1.1.xls)

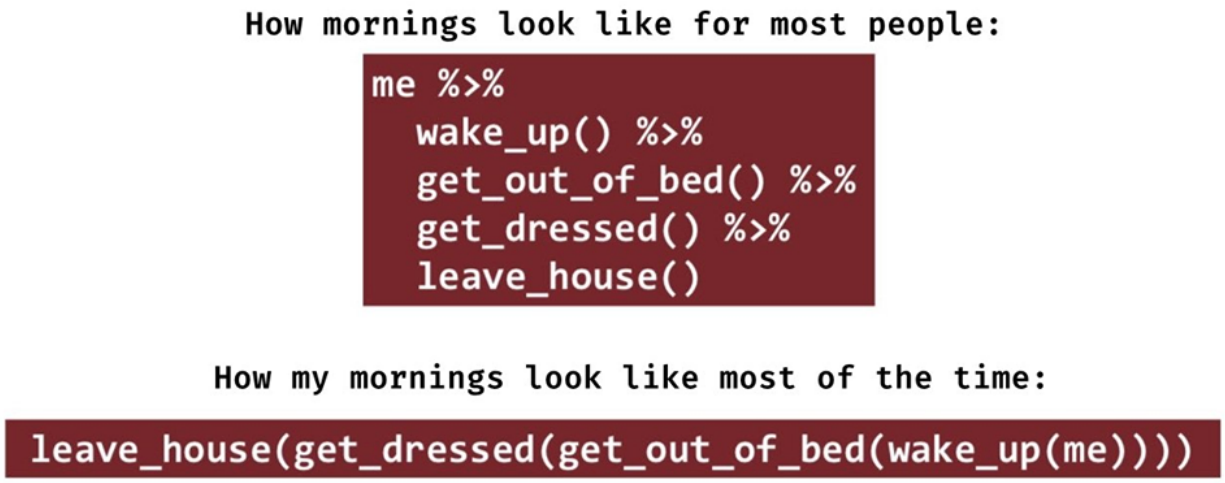

Figure 1: Base R vs. Tidyverse

Explore the data descriptively

To explore data descriptively, we will learn the basics of `tidyverse` today.

Tidyverse

The R universe (and the data management in R) basically builds upon two (seemingly contradictive) approaches: **base R** and **tidyverse**. These two approaches are often seen as two different philosophies. Base R is already implemented in R, whereas the `tidyverse` requires the user to load new packages. People often find base R unintuitive and hard to read. This is why Hadley Wickham developed and introduced the `tidyverse` – a more intuitive approach to manage and wrangle data. Code written before 2014 was usually written in base R whereas the tidyverse style is becoming more and the standard style.

If you look at Figure 1, you may observe that the two code "chuncks" are susbstantially different. What are differences that you observe?

We will build upon `tidyverse` throughout the course. The logic is fairly simple: As you can see in the graphic above, you have your main object (`me`) in `tidyverse` and you pipe (`%>%`) through this object by filtering, selecting, renaming, ... parts of it. To visualize this logic, the general structure exists:

```
dataset %>%
  select_what_you_need %>%
  do_something
```

In base R you would in contrast wrap the commands around your main object which makes it unnecessarily hard to follow the code. To get a better idea how tidyverse and its pipes work, we will do some first descriptive (exploratory) analysis.

We use our `unpko` dataset and retrieve information on the time span of the dataset. What could be one way to look at the data?

As this approach does not produce nice results, we use a combination of `arrange()` and `View()`. Remember, the shortcut to generate the pipe operator (`%>%`) is Cmd + Shift + M (Mac); Ctrl + Shift + M (Windows).

```
unpko %>% # select the dataset
  arrange(year) # %>% # we sort the year variable in ascending order
```

```
## # A tibble: 4,414 x 16
##    mission missioncountry missionccode  year month yearmon troop police
##    <chr>   <chr>                 <dbl> <dbl> <dbl>   <dbl> <dbl>  <dbl>
##  1 ONUCA   Costa Rica               94  1990     1   1990.    NA     NA
##  2 ONUCA   Costa Rica               94  1990     2   1990.    NA     NA
##  3 ONUCA   Costa Rica               94  1990     3   1990.    NA     NA
##  4 ONUCA   Costa Rica               94  1990     4   1990.    NA     NA
##  5 ONUCA   Costa Rica               94  1990     5   1990.    NA     NA
##  6 ONUCA   Costa Rica               94  1990     6   1990.    NA     NA
##  7 ONUCA   Costa Rica               94  1990     7   1990.    NA     NA
##  8 ONUCA   Costa Rica               94  1990     8   1990.    NA     NA
##  9 ONUCA   Costa Rica               94  1990     9   1990.    NA     NA
## 10 ONUCA   Costa Rica               94  1990    10   1990.    NA     NA
## # ... with 4,404 more rows, and 8 more variables: militaryobservers <dbl>,
## #   total <dbl>, total2 <dbl>, monthlytotpers <dbl>,
## #   monthlytrooppers <dbl>, monthlypolicepers <dbl>,
## #   monthlymilobspers <dbl>, numberofmissions <dbl>
```

```
  # View()
```

If I additionally want to see just the unique years, I need to add the function `distinct()`.

```
unpko %>% # select the dataset
  arrange(year) %>% # we sort the year variable in ascending order
  dplyr::distinct(year) # and select only 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
```

As we can see, the dataset covers 1990 - 2012.

If we want to subset the dataset to Burundi only, we could use the following code:

```
unpko_bdi <- unpko %>% # save it in "unpko_bdi"
  dplyr::filter(missioncountry == "Burundi") # only look at "Burundi"
```

Note, we need to use a double equation sign (==) because we are selecting on a conditionality (similar to an if-function).

If we want to look at the average troop deployment in Burundi per year, we use our previously generated dataset `unpko_bdi`.

```
unpko_bdi %>%
  dplyr::group_by(year) %>% # group by year
  dplyr::summarise(mean_troop_deployment = mean(troop))
```

```
## # A tibble: 9 x 2
##    year mean_troop_deployment
##   <dbl>                 <dbl>
## 1  2004                 4032.
## 2  2005                 5261.
## 3  2006                 3307.
## 4  2007                    0
## 5  2008                    0
## 6  2009                    0
## 7  2010                    0
## 8  2011                    0
## 9  2012                    0
```

```
  # generate mean troop deployment with summarise()
  # and mean()
```

2007-2012 show 0. We can use the `View()` command to double-check if there are no values.

```
unpko_bdi %>%
  filter(year %in% c(2007, 2008, 2009, 2010, 2011, 2012)) # %>%
```

```
## # A tibble: 67 x 16
##    mission missioncountry missionccode  year month yearmon troop police
##    <chr>   <chr>                 <dbl> <dbl> <dbl>   <dbl> <dbl>  <dbl>
##  1 BINUB   Burundi                 516  2007     1   2007.     0     11
##  2 BINUB   Burundi                 516  2007     2   2007.     0     11
##  3 BINUB   Burundi                 516  2007     3   2007.     0     11
##  4 BINUB   Burundi                 516  2007     4   2007.     0     11
##  5 BINUB   Burundi                 516  2007     5   2007.     0     11
##  6 BINUB   Burundi                 516  2007     6   2007.     0     11
##  7 BINUB   Burundi                 516  2007     7   2007.     0     11
##  8 BINUB   Burundi                 516  2007     8   2007.     0     12
##  9 BINUB   Burundi                 516  2007     9   2007.     0     12
## 10 BINUB   Burundi                 516  2007    10   2007.     0     12
## # ... with 57 more rows, and 8 more variables: militaryobservers <dbl>,
## #   total <dbl>, total2 <dbl>, monthlytotpers <dbl>,
## #   monthlytrooppers <dbl>, monthlypolicepers <dbl>,
## #   monthlymilobspers <dbl>, numberofmissions <dbl>
```

```
                # filter for year
  # View() # View
```

As we can see, there is 0 personnel deployed.

Hands on exercise

1. Which countries are included in the `unpko` dataset?
2. How many missions were on average deployed during the years in the `unpko` dataset?
3. *Bonus with the `unpko` dataset: How many countries are covered in the dataset?
4. Now we will use the `ucdp_dyadic` dataset. Reload the dataset (remember, the file is called "ucdp_dyadic_191.RData").
5. Subset the dataset to region == 3 only. Save the result in an object called `ucdp_dyad_region`.

6. Which (unique) countries are in the location variable? Can you make any sense of what the "3" in region stands for?

Further input

- A Tidyverse Cookbook
- Stackoverflow