Download and harmonize FAOSTAT and WDI data: the FAOSTAT package

Filippo Gheri & Michael. C. J. Kao
Food and Agriculture Organization
of the United Nations
Edited by Paul Rougieux

Abstract

The aim of this document is to introduce the **FAOSTAT** package developed by the Food and Agricultural Organization of the United Nations. The package serves as an open gate way to FAOSTAT (the FAO extensive library of agricultural statistics) and WDI (World Development Indicators database of World Bank).

Retrieve, harmonize, and process official statistics is a thorny task. This paper will demonstrate how the **FAOSTAT** can alleviate some of these obstacles providing the possibility of downloading, harmonizing, and processing FAOSTAT and WDI data automatically through R. The use of open source software R brings tremendous amount of benefits, speeding up the production process and open up the data and methodology to the general public. In this small paper we will illustrate the process and demonstrate how the use of the package can increase transparency and sustainability.

Keywords: R, Official Statistics, FAOSTAT.

Contents

1. Introduction

In 2011 Adam Prakash and Mattieu Stigler in the ESS division of the Food and Agricultural Organization of the United Nations had the idea of utilizing R and LaTeX for the production of the FAO statistical yearbook. The initiative was taken in order to replace the labour intensive work with a streamline system which integrates data extraction, processing, analysis, and dissemination into one single comprehensive system. The **FAOSTAT** package is one of the first outputs of the new process, and it aims at facilitating the user in downloading, harmonizing, and processing statistics.

This paper will demonstrate how the **FAOSTAT** package is used to automatically download data from FAOSTAT and WDI, and to harmonize different data sources under a common country coding system. The goal is to provide a tool that facilitates the data collection from FAOSTAT and WDI, and helps the user in harmonizing different datasets.

2. Install the package

The package can be installed from the CRAN repository just like all other R packages.

```
if(!is.element("FAOSTAT", .packages(all.available = TRUE)))
   install.packages("FAOSTAT")
library(FAOSTAT)
```

The latest version of the package can also be installed from the following github repository:

```
if(!is.element("FAOSTAT", .packages(all.available = TRUE)))
    remotes::install_gitlab(repo="paulrougieux/faostatpackage/FAOSTAT")
library(FAOSTAT)
```

This documentation is also the vignette of the package.

```
help(package = "FAOSTAT")
vignette("FAOSTAT", package = "FAOSTAT")
```

3. Data collection

3.1. Download data from FAOSTAT

FAOSTAT is the largest agricultural database in the world. It contains data from land productivity to agricultural production and trade dating back from 1960 to the most recent available data. Detailed information on meta data, methods and standards can be found on the official website of FAOSTAT (http://www.fao.org/faostat/en/#data) and the Statistics Division (ESS) (http://www.fao.org/economic/ess/en/).

Load data from the bulk download repository

Explore the FAOSTAT website to find out the data you are interested in http://www.fao.org/faostat/en/#data Copy a "bulk download" url, for example they are located in the right menu on the "crops" page: http://www.fao.org/faostat/en/#data/QC.

In this example, the url is split in two parts: a common part `'url_bulk_site'` and a .zip file name part. In practice you can enter the full url directly as the `` `url_bulk` `` argument. Notice also that I have choosen to load global data in long format (normalized).

```
url_bulk_site <- "http://fenixservices.fao.org/faostat/static/bulkdownloads"
url_crops <- file.path(url_bulk_site, "Production_Crops_E_All_Data_(Normalized).zip")
url_forestry <- file.path(url_bulk_site, "Forestry_E_All_Data_(Normalized).zip")


# Create a folder to store the data
data_folder <- "data_raw"
dir.create(data_folder)

# Download the files
download_faostat_bulk(url_bulk = url_forestry, data_folder = data_folder)
download_faostat_bulk(url_bulk = url_crops, data_folder = data_folder)

# Read the files and assign them to data frames
production_crops <- read_faostat_bulk("data_raw/Production_Crops_E_All_Data_(Normalized).zip
```

```
forestry <- read_faostat_bulk("data_raw/Forestry_E_All_Data_(Normalized).zip")

# Save the data frame in the serialized RDS format for fast reuse later.
saveRDS(production_crops, "data_raw/production_crops_e_all_data.rds")
saveRDS(forestry,"data_raw/forestry_e_all_data.rds")
```

Load data through the FAOSTAT API (this method is deprecated since 2017)

In order to access to an indicator in FAOSTAT using the API, three pieces of information need to be provided: domain, item, and element codes. They are defined as:

Domain Code :
: The domain associated with the data. For example, production, trade, prices etc.

Item Code :
: These are the codes relating to the commodity or product group such as wheat, almonds, and aggregated item like "total cereals".

Element Code :
: Lastly, this is the quantity/unit or data collection type associated with the commodity. Typical elements are quantity, value or production/extraction rate.

An interactive function `FAOsearch` has been provided for the user to identify the respective codes. An object `.LastSearch` will be assigned after the search and can be used by both the `getFAO` and `getFAOtoSYB` functions as the sole argument to download the data.

```
## Use the interective function to search the codes.
FAOsearch()

## Use the result of the search to download the data.
test = getFAO(query = .LastSearch)
```

The `getFAOtoSYB` is a wrapper for the `getFAO` to batch download data, it supports error recovery and stores the status of the download. The function also splits the data downloaded into entity level and regional aggregates, saving time for the user. Query results from `FAOsearch` can also be used.

```
## A demonstration query
FAOquery.df = data.frame(varName = c("arableLand", "cerealExp", "cerealProd"),
                         domainCode = c("RL", "TP", "QC"),
                         itemCode = c(6621, 1944, 1717),
                         elementCode = c(5110, 5922, 5510),
                         stringsAsFactors = FALSE)

## Download the data from FAOSTAT
FAO.lst = with(FAOquery.df,
    getFAOtoSYB(name = varName, domainCode = domainCode,
                itemCode = itemCode, elementCode = elementCode,
                useCHMT = TRUE, outputFormat = "wide"))
FAO.lst$entity[, "arableLand"] = as.numeric(FAO.lst$entity[, "arableLand"])
```

The object returned is a list of length three, these are entity level data, aggregates and the download status. The function supports both long and wide format.

In some cases multiple China are provided. In the FAOSTAT database for example, the trade domain provides data on China mainland (faostat country code = 41), Taiwan (faostat country code = 214) and China plus Taiwan (faostat country code = 357). In some other datasets it is also possible to find China plus Taiwan plus Macao plus Hong Kong (faostat country code = 351). The `CHMT` function avoids double counting if multiple China are detected by removing the more aggregated entities if detected. The default in `getFAOtoSYB` is to use `CHMT` when possible. It is important to perform this check before the aggregation step in order to avoid duble counting. This means that not necessarely this operation needs to be done at the data collection stage. This can be done also at a later stage using the `FAOcheck` function (or the `CHMT` function directly).

```
FAOchecked.df = FAOcheck(var = FAOquery.df$varName, year = "Year",
                         data = FAO.lst$entity, type = "multiChina",
                         take = "simpleCheck")
```

3.2. Download data from World Bank

The World Bank also provides an API where data from the World Bank and various international organization are made public. More information about the data and the API can be found at http://data.worldbank.org/

The authors are aware of the **WDI** package, but we have wrote this function before the recent update of the package with additional functionalities. We have plans to integrate with the **WDI** package to avoid confusion for the users.

```
## Download World Bank data and meta-data
WB.lst = getWDItoSYB(indicator = c("SP.POP.TOTL", "NY.GDP.MKTP.CD"),
                     name = c("totalPopulation", "GDPUSD"),
                     getMetaData = TRUE, printMetaData = TRUE)
```

The output is similar to the object generated by `getFAOtoSYB` except that if the argument `getMetaData` is specified as TRUE then the meta data is also downloaded and saved. The function `getWDItoSYB` relies on `getWDI` and `getWDImetaData` functions.

One point to note here, it is usually unlikely to reconstruct the world aggregates provided by the World Bank based on the data provided. The reason is that the aggregate contains Taiwan when available, yet the statistics for Taiwan are not published.

4. Merge data from different sources

Merge is a typical data manipulation step in daily work yet a non-trivial exercise especially when working with different data sources. The built in *mergeSYB* function enables one to merge data from different sources as long as the country coding system is identified. Currently the following country coding translation are supported and included in the internal data set FAOcountryProfile of the package:

- United Nations M49 country standard [UN_CODE]
  https://unstats.un.org/unsd/methods/m49/m49.htm/
- FAO country code scheme [FAOST_CODE]
  https://termportal.fao.org/faonocs/appl/

- FAO Global Administrative Unit Layers (GAUL).[ADM0_CODE]
- ISO 3166-1 alpha-2 [ISO2_CODE]
  https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2
- ISO 3166-1 alpha-2 (World Bank) [ISO2_WB_CODE]
  https://data.worldbank.org/
- ISO 3166-1 alpha-3 [ISO3_CODE]
  http://en.wikipedia.org/wiki/ISO_3166-1_alpha-3
- ISO 3166-1 alpha-3 (World Bank) [ISO3_WB_CODE]
  https://data.worldbank.org/

Data from any sources employ country classification listed above can be supplied to `mergeSYB` in order to obtain a single merged data. However, the column name of the country coding scheme is required to be the same as the name in square bracket, the responsibility of identifying the coding system lies with the user.

Nevertheless, often only the name of the country is provided and thus merge is not possible or inaccurate based on names. We have provided a function to obtain country codes based on the names matched. In order to avoid matching with the wrong code, the function only attempts to fill in countries which have exact match.

```
## Just a demonstration
Demo = WB.lst$entity[, c("Country", "Year", "totalPopulation")]
demoResult = fillCountryCode(country = "Country", data = Demo,
    outCode = "ISO2_WB_CODE")

## Countries have not been filled in.
unique(demoResult[is.na(demoResult$ISO2_WB_CODE), "Country"])
```

We have not implemented a regular expression match for the identification reason listed below. From the above example we can see that both China and Sudan are not filled in, the identification of Sudan prior to 2011 and China should be carefully examined.

Below we list some commonly observed problems when merging data from different sources.

4.1. Identification problem

Due to the fact that different organization are bounded by different political agenda, the users need to be aware of the precise definition and legal recognition of countries. Take example, the China provided by the World Bank does not include Taiwan, Hong Kong and Macao. On the other hand, FAO provides not only a single China (FAO = 41), but also China plus Taiwan (FAO = 357) depending on the context. In addition, it is common to observed statistics for China (ISO2 = CN or ISO3 = CHN) which includes Taiwan, Hong Kong and Macao. The default translates China from other country coding scheme to Mainland China (FAO = 41) and is not matched in `fillCountryCode`.

4.2. Representation problem

Moreover, the situation is further complicated by disputed territories or economic union such as Kosovo and Belgium-Luxembourg which does not have representation under particular country coding system.

```
## Countries which are not listed under the ISO2 international standard.
FAO.df = translateCountryCode(data = FAOchecked.df, from = "FAOST_CODE",
    to = "ISO2_CODE")

## Countries which are not listed under the UN M49 system.
WB.df = translateCountryCode(data = WB.lst$entity, from = "ISO2_WB_CODE",
    to = "UN_CODE")
```

4.3. Transition problem

Finally, the discontinuity and transition of countries further increases the complexity of the data. The South Sudan was recognized by the United Nations on the 9th of July 2011, however, the statistics reported by The republic of the Sudan in the same year can also includes data for South Sudan thus failing the mutually exclusive test. Moreover, sources which uses ISO standard country code have no way to distinguish between the new and old Sudan (SD and SDN are used for both entity) which causes problem in merge with country system that distingiushes the entity.

Finally, if historical aggregates are computed then a region composition which does not backtrack in time will result in an aggregate which is incorrect. For more details about historical and transitional countries please refer to http://unstats.un.org/unsd/methods/m49/m49chang.htm

Given the lack of an internationally recognized standard which incorporates all these properties, we suggests the use of the FAO country standard and region profile shipped with the package which addresses most of these problems.

```
merged.df = mergeSYB(FAOchecked.df, WB.lst$entity, outCode = "FAOST_CODE")
```

5. Scale data to basic unit

The functions `translateUnit` and `scaleUnit` help the user in scaling the data to the basic unit. The function `translateUnit` allows to translate multipliers from character names into numbers and vice versa. This is really useful because multipliers in metadata are usually provided in character names. The function `scaleUnit` allows to scale the variables in the dataset using the multipliers provided by the user. It is always important to work with variables in basic units, especially if new variables need to be generated. As a matter of fact, when a large set of new variables need to be generated out of the raw variables, it is already quite difficult to deal with the unit of measurements. For this reason it is better to avoid the multipliers issue a priori by scaling all the raw variables to their basic unit.

```
multipliers = data.frame(Variable = c("arableLand", "cerealExp", "cerealProd",
                                      "totalPopulation", "GDPUSD"),
                         Multipliers = c("thousand", NA, NA, NA, NA),
                         stringsAsFactors = FALSE)
multipliers[, "Multipliers"] =
  as.numeric(translateUnit(multipliers[, "Multipliers"]))
preConstr.df = scaleUnit(merged.df, multipliers)
```

6. Computing growth, and other derivatives

The function `constructSYB` allows to automatically construct four different types of indicators: shares, growth rates, indeces, and year to year changes through the functios `shConstruct`, `grConstruct`, `lsgr`, `geogr`, `indConstruct`, `chConstruct`, `chgr`. There are two types of growth rate shipped with the package, the least squares growth rate and the geometric growth rate. The least squares growth rate is used when the time series is of sufficient length. The default is at least 5 usable observations, however if the time series is sparse and more than 50% of the data are missing than the robust regression is used.

```
con.df = data.frame(STS_ID = c("arableLandPC", "arableLandShareOfTotal",
                               "totalPopulationGeoGR", "totalPopulationLsGR",
                               "totalPopulationInd", "totalPopulationCh"),
                    STS_ID_CONSTR1 = c(rep("arableLand", 2),
                                       rep("totalPopulation", 4)),
                    STS_ID_CONSTR2 = c("totalPopulation", NA, NA, NA, NA, NA),
                    STS_ID_WEIGHT = rep("totalPopulation", 6),
                    CONSTRUCTION_TYPE = c("share", "share", "growth", "growth",
                                          "index", "change"),
                    GROWTH_RATE_FREQ = c(NA, NA, 10, 10, NA, 1),
                    GROWTH_TYPE = c(NA, NA, "geo", "ls", NA, NA),
                    BASE_YEAR = c(NA, NA, NA, NA, 2000, NA),
                    AGGREGATION = rep("weighted.mean", 6),
                    THRESHOLD_PROP = rep(60, 6),
                    stringsAsFactors = FALSE)

postConstr.lst = with(con.df,
                      constructSYB(data = preConstr.df,
                                   origVar1 = STS_ID_CONSTR1,
                                   origVar2 = STS_ID_CONSTR2,
                                   newVarName = STS_ID,
                                   constructType = CONSTRUCTION_TYPE,
                                   grFreq = GROWTH_RATE_FREQ,
                                   grType = GROWTH_TYPE,
                                   baseYear = BASE_YEAR))
```

7. Compute aggregates

Aggregation is another important step that is commonly overlook. Many things need to taken into account. Aggregate composition, duplicated values, missing values, aggregation method, and aggregation rules are probably the most important ones. The result can vary due to the differences between the regional composition and the set of countries used. Furthermore, it is complicated by the amount of missing values which can render the aggregates incomparable. Given the missing values and diverging country sets, aggregation can only serve as approximates in order to inform the general situation of the region. The user has the possibility of choosing whether to apply a rule or not. The rule consists in computing the aggregate just if, for each aggregate, the available data represent (in terms of the weighting variable) a share greater than the threshold provided by the user. In case of aggregation method equal to "sum", the same rule is applied considering a weight equals to one for all the countries.

```
## Compute aggregates under the FAO continental region.
relation.df = FAOregionProfile[, c("FAOST_CODE", "UNSD_MACRO_REG")]

Macroregion.df = Aggregation(data = postConstr.lst$data,
                             relationDF = relation.df,
                             aggVar = c("arableLand", "totalPopulation",
                                        "arableLandPC"),
                             weightVar = c(NA, NA, "totalPopulation"),
                             aggMethod = c("sum", "sum", "weighted.mean"),
                             applyRules = TRUE,
                             keepUnspecified = TRUE,
                             unspecifiedCode = "NotClassified",
                             thresholdProp = c(rep(0.65,3)))
```

Acknowledgement

The authors owe a great debt to Adam Prakash, Guido Barbaglia, Amy Heyman, Amanda Gordon, Jacques Joyeux, Mattieu Stigler, and Markus Gesmann for their contribution to the package.

The authors would also like to express their profound gratitude to the directors Pietro Gennari and Josef Schimidhuber and the entire ESS division for their support.

Affiliation:

Filippo Gheri & Michael C.J. Kao
Economics and Social Statistics Division
Economic and Social Development Department
United Nations Food and Agriculture Organization
Viale delle Terme di Caracalla 00153 Rome, Italy
E-mail: filippo.gheri@fao.org, michael.kao@fao.org