Harmonise and integrate heterogeneous areal data with the R package `arealDB`.

Steffen Ehrmann[1,*], **Ralf Seppelt**[2], **Carsten Meyer**[1]

1 German Centre for Integrative Biodiversity Research (iDiv) Halle-Jena-Leipzig, Deutscher Platz 5e, 04103 Leipzig, Germany
2 UFZ – Helmholtz Centre for Environmental Research, Leipzig, Department Computational Landscape Ecology, Permoserstraße 15, 04318 Leipzig, Germany

* steffen.ehrmann@idiv.de

ABSTRACT

Areal data is a common data type to store information such as biodiversity inventories, socioeconomic censuses or cadastral surveys. Many research questions require that areal data are integrated from multiple heterogeneous sources. Inconsistent concepts, terms, definitions, or messy tables makes data wrangling an often tedious and error-prone process. A dedicated tool that assists in organising areal data still is lacking. Here, we introduce the R package `arealDB` that helps to harmonise and integrate heterogeneous areal data and associated geometries into a consistent database. The package is used to collect metadata, harmonise language and variable names, reshape messy into tidy data and integrate them in a standard data format. `arealDB` solves the specific problem of integrating disparate regional data sources on a given target variable, which may be published in different languages, with a different table arrangement or provided in various data formats. We guide the user step by step through the individual functions needed to integrate two such datasets using the example of the harvested area of soybean in Brazil and the USA. A database that has been built with `arealDB` is "tidy", and can thus be accessed easily with powerful and widespread tools such as the R meta-package `tidyverse`. Moreover, it is accompanied by provenance documentation that traces the full process of creation for each data point in the database. By offering easy-to-use tools for integrating areal data, `arealDB` promises substantial time-savings to database collation efforts, as well as quality-improvements to downstream scientific, monitoring, and management applications.

***K*eywords** disorganised messy data · interoperability · data integration · relational database · census and survey data · metadata · provenance documentation · zonal data

1 Introduction

Areal data capture phenomena of interest at the level of finite spatial units. They are an essential data type in many basic and applied research fields, for example, to project human populations or to analyse the spread of infectious diseases based on census or survey data, or to map global biodiversity patterns based on species checklists. Areal data also play a crucial role in various policy and management applications such as national progress reporting towards Sustainable Development Goals (SDGs), to assess the implications of macroeconomic policies based on international trade statistics or to document land ownership. Through illustrative maps in news or education media, areal data are also an everyday communication tool in civil society.

Many critical applications of areal data surpass the spatial, temporal or thematic scope of any unique data source, which makes it necessary to integrate heterogeneous sources into a single, more comprehensive database (Otto et al. [2015]). Efforts of harmonising and integrating areal data are carried out by numerous organisations such as the Food and Agriculture Organization, the World Bank Group and many smaller projects.

However, integrating areal data from disparate sources usually is a tedious and error-prone process (see Tab. 1). The approaches, languages [1], definitions, and formats used to collate and present data in distinct datasets are rarely identical, which frequently leads to inconsistencies. The most apparent problem lies in the fact that heterogeneous source data are typically not arranged according to a common standard, they are often available as *disorganised messy data*.

[1]http://www-ksl.stanford.edu/kst/what-is-an-ontology.html

"Disorganised" refers to data in a spreadsheet that is neither tidy nor a single rectangular table (Wickham [2014]). Such data may be organised according to some other arrangement that can be considered systematic, yet is "mis-organised" with respect to a database standard. From disorganised messy data we have to gather data that may be segregated into clusters that are spread across a spreadsheet, or variables that may be arranged in rows while other variables of the same dataset are available in columns, etc.

Table 1. List of issues, which have to be considered when building a database of areal data that are addressed by `arealDB`.

short	**class**	**challenge**
spatially match data	georeferencing	connect names of territorial units to spatial polygon data (geometries).
administrative reforms	alternative data	areal data refer to old territorial units.
distinct data sources	alternative data	several authorities collect areal data of the same territorial unit(s), possibly providing their own geometries.
disputed areas	alternative data	authorities provide their data for territorial units that are also claimed by other authorities.
territorial unit names	translation	territorial unit names do not follow the same standard.
ontology of variables	translation	variable names and values of categorical variables do not follow the same ontological standard.
translate terms	translation	data from different languages shall be integrated.
disorganised messy data	documentation	source data are not arranged according to the same format.
data harmonisation	documentation	provide a thorough documentation of how data were processed.
metadata	documentation	the database should be provided embedded in it's context.

To address the challenges in integrating heterogeneous areal data sources, we introduce the R software package `arealDB`. The package leads the user through the process of harmonising and integrating areal datasets and their corresponding geometries into a standardised database while ensuring internal consistency and metadata documentation.

We exemplify the functionality of `arealDB` by integrating two example datasets on the harvested area of soybean. The first dataset (Brazil) is provided in Portuguese language and accompanied by specific geometries, while the second datasets (USA) is provided in our target language English but does not come with geometries and instead refers to the names of counties. A readily available dataset that has been integrated from several nations can be a powerful tool to map temporal and spatial dynamics of production of the commodity crop. The tools presented here may also be used to process areal data of any other kind, such as socioeconomic census and survey data, species checklist data or national indicator data. In combining focal variables across various disciplines powerful applications are easily implemented, for instance to analyse how commodity crop production in combination with socioeconomic variables leads to land-use change.

Description

`arealDB` contains three groups of tools that reflect three stages of data management (Fig. 1):

1. Stage 1 with functions that set up a database while gathering initial thematic metadata.
2. Stage 2 with functions that transform downloaded files into a standardised format and gather metadata on those files (i.e. *register* them).
3. Stage 3 with functions that harmonise, based on the metadata of stage two, and integrate geometries and data tables into a standardised database (this process is called *normalising* in `arealDB`).

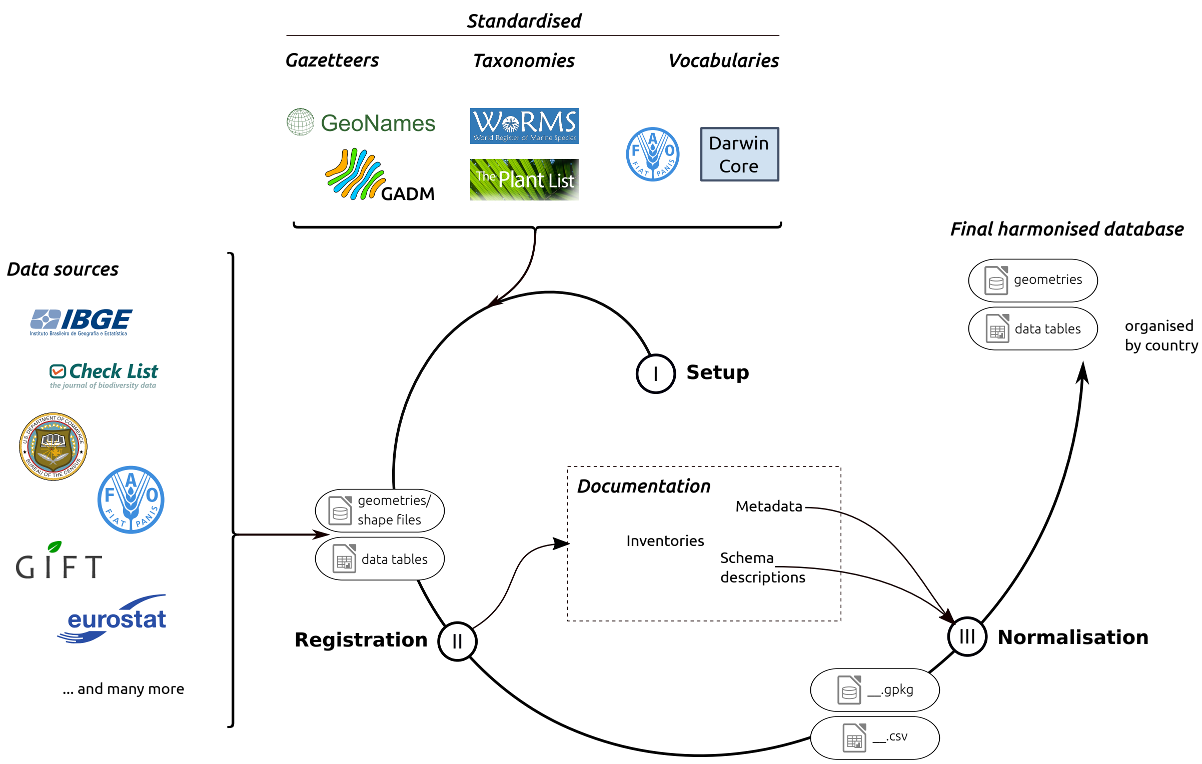

Figure 1. Flow-chart of the general workflow. In stage one initial metadata from already standardised databases are used to establish an ontological basis, then data tables and geometries that have been downloaded from various data sources are registered in stage two and eventually those files are harmonised and integrated (normalised) into the final database in stage three. The initial input data and data sources shown in this figure are only a subset of sources that can be handled with `arealDB`.

Project setup

An areal database is started in `arealDB` by a call to the function `setPath()`, which creates the directory structure into which the data will be organised (Fig. 2a). The files that shall be integrated into that database should be recorded for documenting the history of the database, in `arealDB` this happens via so-called inventory tables (Fig. 2b).

The variables handled with `arealDB` are distinguished into the groups *identifying variables* and *values variables*. Identifying variables are categorical and characterise observational units. Those are the obligatory variables 'timestep' and 'spatial unit' but also usecase-specific variables such as 'biological species', 'agricultural commodities' or 'socio-economic groups of people'. Values variables store the areal data of the respective observational units. To set up identifying variables, the function `setVariables()` is used (Fig. 2). This function creates the skeleton of two files per identifying variable, (1) an index table, which relates the variables' terms to an ID and to ancillary information and (2) a translation table, which relates terms in other languages and semantics to the target language. Both index and translation tables can be seeded with tabular information from other projects (Fig. 1). Sources of such tabular information may be, for example, gazetteers such as GeoNames [2] or the ancillary data in GADM [3], biological taxonomies such as the Plant List [4], WoRMS (Horton et al. [2019]) or the IUCN Red List [5], or specific standardised ontologies such as data products offered by FAOSTAT [6], the Darwin Core (Wieczorek et al. [2012]), the Humboldt Core (Guralnick et al. [2018]) or information from the Land Administration Domain Model (Lemmen et al. [2015]).

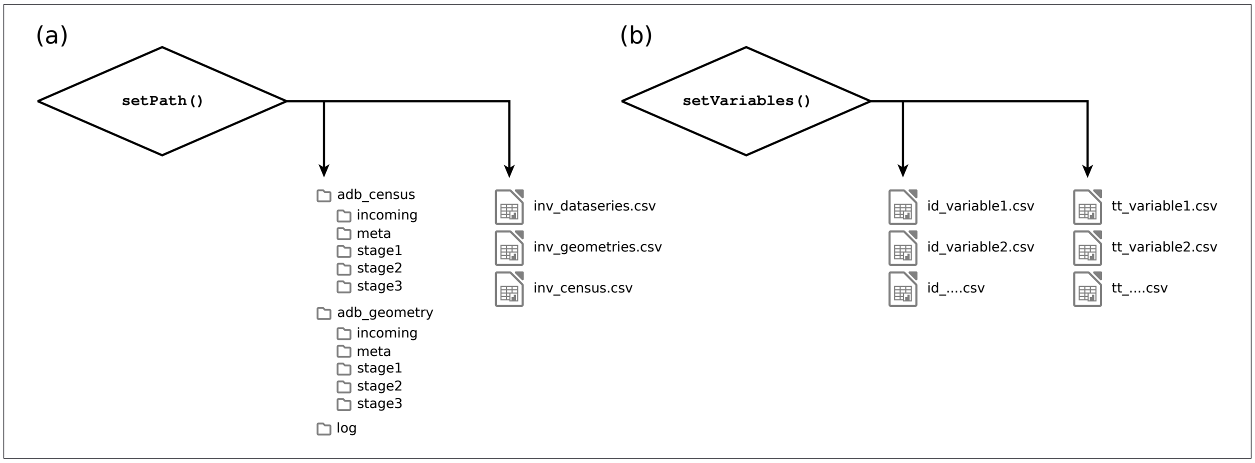

Figure 2. Flow-chart of the project setup. (a) The function `setPath()` initiates the project by creating a directory structure in which the files are stored and by creating the inventory tables for dataseries, geometries and census tables. **(b)** The function `setVariables()` creates index and translation tables for all variables that should be handled in this project.

Data registration

An aspect that increases the quality of an integrated database is provenance documentation because errors that may arise in a final database can be traced back to the specific source datasets or certain modification processes based on this documentation. Documenting provenance requires that the input state of a dataset, as well as procedural metadata, which become available as a side-product in the evolution from messy to tidy data, is known.

Thus, the second step in integrating databases with `arealDB` is to create an inventory of the relevant files and to record initial metadata, such as original file names and file locations (this process is called *registering* in `arealDB`). It is, furthermore, required to document a file's structure in custom schema descriptions for data tables (see, for example, Mäs et al. [2018]. It records the file-specific peculiarities such as the position (columns and rows) of the data components, which allows automatically reshaping/normalising the files in the next step.

Dataseries are a particular series of data that are provided by the same data provider, ideally including both geometries and data tables and a proper link between the two. Datasets from the same dataseries are typically stored in the same format and follow the same organisational logic. A new dataseries is registered with the function `regDataseries()` in the inventory table `inv_dataseries.csv` (Fig. 3) as the first items because geometries and data tables refer to them.

New items to include in an areal database retain at the first stage their original name and their original arrangement. At stage two they are then stored in *csv* and *geopackage* format and are assigned standardised names. The functions `regGeometries()` and `regTable()` are provided with a set of metadata that are checked for consistency and inserted into the inventory tables `inv_geometries.csv` and `inv_tables.csv` (Fig. 3b and 3c). Both functions oversee, first, that the individual files are transformed to the target format, second, that the correct standard names are assigned, and third, that all files are stored in the correct directory. The functions then create IDs for all items and insert the provided metadata into the inventory tables, where they are looked up in data normalisation'.

None of the inventory tables ever need to be modified manually. All inventory information is gathered via the `reg*()` functions, which check for consistency and that that information can be used without problems later in the workflow.

[2]http://www.geonames.org
[3]http://www.gadm.org
[4]http://www.theplantlist.org
[5]http://www.iucnredlist.org
[6]http://www.fao.org/faostat

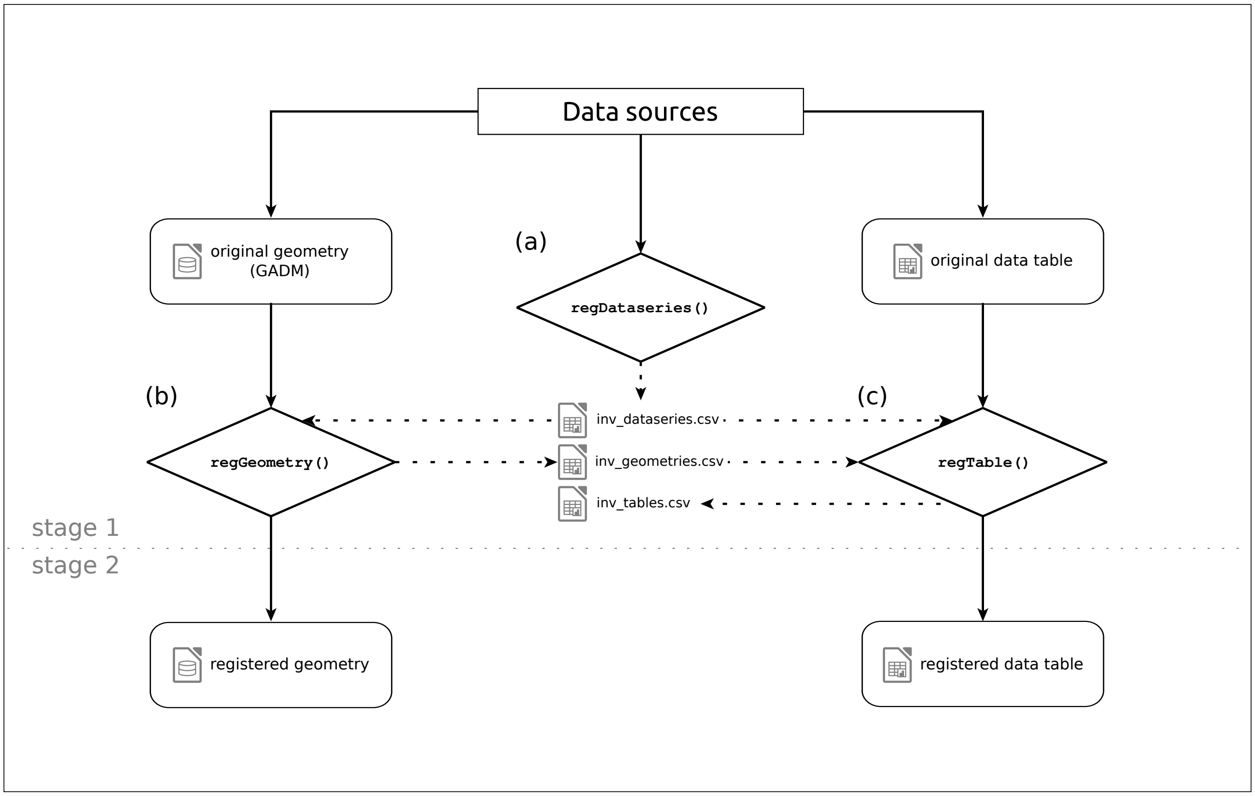

Figure 3. Flow-chart of the registration procedure. **(a)** The function `regDataseries()` is used to document the various dataseries that are provided by the data source. **(b)** Then the function `regGeometry()` is used to register all geometry files that have been downloaded. **(c)** Finally, the function `regTable()` is used to register all census tables that have been downloaded, and to relate the census tables to dataseries and geometries. The registered files are stored in the folders `"/adb_geometries/stage2"` and `"/adb_tables/stage2"`.

Data normalisation

The third and final step to integrate areal data consists of harmonising and reshaping the output of stage two (this process is called *normalising* in `arealDB` (Codd [1990])). At stage two, there is no guarantee that territorial units in geometries and data tables have compatible names or that areal data are georeferenced, that variables are provided in the same language across several sources, or that source data tables are provided in a compatible arrangement. A harmonised and normalised database would allow employing relatively simple (procedural) algorithms to extract data from all sources at once. It drastically minimises the effort on, and thus error-sources from, coming up with data source-specific extraction procedures for analysis.

Both geometries and census tables are normalised with help of the function `normalise()`, which internally calls the functions `normGeometry()` and `normTable()`. Geometries are typically provided as shape or geopackage files, which have already been optimised for interoperability, and where it is thus sufficient to know which columns in the attribute table contain names of the territorial units. Data tables are however less standardised and thus potentially vastly more complex or messy, and hence require schema descriptions. The schema descriptions, which have been recorded in step two, can be thought of as a list that documents accurate positions of variables held in data tables (see Supplement 1 for details). The `norm*()` functions use those schema descriptions to reshape the data.

Finally, the areal data are linked to geometries (i.e., they are georeferenced). This step is strictly speaking not part of data normalisation anymore; georeferencing is described here nevertheless because it is carried out automatically while normalising. Georeferencing in `arealDB` requires, first of all, an initial geometry dataset from which the *administrative hierarchy ID (ahID)* can be constructed, and which must thus include information on the hierarchical arrangement of the territorial units. The ahID is unique per territorial unit at each administrative level so that areal data can be linked unambiguously to respective geometries. Areal data that come without geometries have to be linked to the initial geometries to have a georeference. When areal data come with geometries, they already have a georeference. However, those geometries must be matched with the initial geometries to ensure spatial consistency. This process is carried out internally via the function `matchUnits()`.

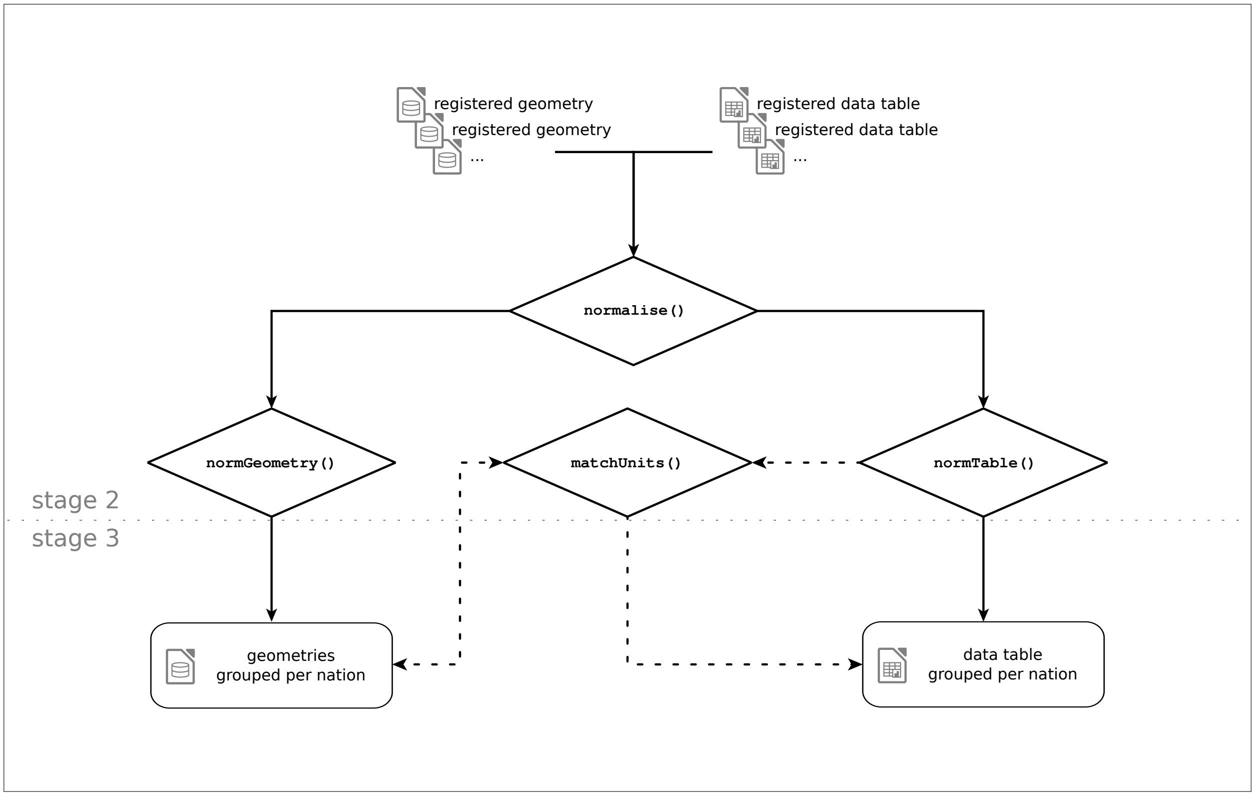

Figure 4. Flow-chart of the normalisation procedure. The function `normalise()` detects all relevant files of geometries or data tables in stage two. It then calls the respective function that carry out the normalisation. The function `normGeometry()` groups geometries per nation and creates the *administrative hierarchy ID (ahID)*. The function `normTable()` reshapes the data tables into tidy format, calls `matchUnits()` to assign ahID to the areal data and groups the tables per nation. The normalised files are stored in the folders `"/adb_geometries/stage3"` and `"/adb_tables/stage3"`.

Application

arealDB may be helpful to anybody who works with areal data, whether it is merely for combining a couple of relatively small scale datasets, when it comes to harmonising a complex geospatial database project that already exists according to the standards defined here or for integrating vast citizen science based datasets of areal data. Users can, moreover, profit from arealDB via harmonised database compilation efforts led by research teams, such as WorldPop (Stevens et al. [2015]), GIFT (Weigelt et al. [2019]) and many others (Otto et al. [2015], König et al. [2019], Lloyd et al. [2019]), from statistical agencies, such as Eurostat, or international organizations, such as the Food and Agriculture Organization or the World Bank Group.

The package can recently be installed via the below code and it is versioned (Ehrmann [2019]).

```
library(devtools)
devtools::install_git(url = "https://gitlab.com/luckinet/software/arealDB")
```

In the following we outline an example of all steps that are required to build a harmonised areal database. The proper documentation of any function that comes with this package can be retrieved after installation, for example, via the command ?setPath for the function of that name.

Stage one (Fig. 2) consists of initiating a project at a particular root path via setPath() followed by providing tables of standardised variables or vocabularies via setVariables().

```
library(arealDB)
library(readr)
library(magrittr)

setPath(root = ".../root/directory/of/newProject")

read_csv(file = "data/fao_commodities.csv", col_types = "iclciccc") %>%
  setVariables(variable = "commodities",
               pid = "faoID",
               target = "simpleName")

read_csv(file = "data/nation_translations.csv", col_types = "cccDi") %>%
  setVariables(variable = "nations",
               type = "tt",
               origin = "origin",
               target = "target")
```

Listing 1. Create id and translation tables

At stage two, arealDB requires to document the provenance of a database, and this starts with all the dataseries from which input data are taken, via regDataseries().

```
regDataseries(name = "gadm",
              description = "Database of Global Administrative Areas",
              website = "https://gadm.org/index.html",
              update = TRUE)

regDataseries(name = "ibge",
              description = "Instituto Brasileiro de Geografia e Estatistica",
              website = "https://sidra.ibge.gov.br/tabela/5457",
              update = TRUE)

regDataseries(name = "usda",
              description = "US Dept. of Agriculture",
              website = "https://www.nass.usda.gov/Quick_Stats/Lite/index.php",
              update = TRUE)
```

Listing 2. Register dataseries

Dataseries are provided to the functions regGeometry() and regTable() to establish the link between geometries and data tables. The function regGeometry() requires a dataseries of geometries in the argument gSeries = ..., such as the GADM dataset. Both, regGeometry() and regTable() require additional arguments, as shown below, to gather metadata that are required later-on to reshape the respective data into the harmonised database.

```
regGeometry(nation = "NAME_0",
            gSeries = "gadm",
            level = 1,
            layer = "level0",
            nameCol = "NAME_0",
            archive = "gadm36_levels_gpkg.zip|gadm36_levels.gpkg",
            update = TRUE)

regGeometry(nation = "Brazil",
            gSeries = "ibge",
            level = 2,
            nameCol = "NM_ESTADO",
            archive = "br_unidades_da_federacao.zip|BRUFE250GC_SIR.shp",
            update = TRUE)
```

Listing 3. Register geometries

The function `regTable()` connects data tables and geometries via `dSeries = ...` as the dataseries of tables and `gSeries = ...` as the dataseries of geometries to which the tables are connected. For example, while IBGE provides both, geometries and data tables, this is not the case for USDA, which requires to fall back to the GADM geometries.

```
regTable(nation = "Brazil",
         subset = "soy",
         dSeries = "ibge", gSeries = "ibge",
         level = 3,
         variable = c("harvested_area"),
         algo = 1,
         begin = 1990, end = 2017,
         archive = "tabela5457_harvested.csv",
         update = TRUE)

regTable(nation = "United States of America",
         subset = "soy",
         dSeries = "usda", gSeries = "gadm",
         level = 3,
         variable = c("harvested_area"),
         algo = 1,
         begin = 1990, end = 2017,
         archive = "soybean_us_county_1990_2017.csv",
         update = TRUE)
```

Listing 4. Register data tables

Stage three consists of harmonising the data based on the metadata gathered in stage two. Either a call is carried out to the functions `normGeometry()` and `normTable()`, or to the convenience function `normalise()`, which internally calls the former two functions based on the argument `what = ...`. In the former case, each file needs to be handled manually, i.e., the path to each file needs to be provided (this is not shown here). In the latter case, the function `normalise()` gathers all files that are unprocessed at stage two to enqueue and process one after the other.

It is often required that disorganised messy data have to be handled, as there are recently no standards defined for how the vast range of potential input data ought to be organised. To normalise those data tables, a schema description is required, an example of which is shown for the USDA dataset below. This dataset is already in tidy form and thus we use the schema description to tell the function `normTable()` which data is found where. The function `normTable()` uses this information to reshape the dataset into the form that is required for the harmonised final database. Additional information on schema description, and how to set them up, are noted in Supplement 1).

```
normalise(what = "geometries",
          nation = c("brazil", "united states"),
          update = TRUE, verbose = FALSE)

meta_usda1 <-
  list(clusters = list(top = NULL, left = NULL, width = NULL, height = NULL,
                       id = NULL),
       variables = list(territories =
                          list(type = "id", name = NULL, form = "long",
```

```
                              row = NULL, col = c(6, 10), rel = FALSE),
                       period =
                         list(type = "id", name = "year", form = "long",
                              row = NULL, col = 2, rel = FALSE),
                       commodities =
                         list(type = "id", name = NULL, form = "long",
                              row = NULL, col = 16, rel = FALSE),
                       harvested =
                         list(type = "values", unit = "ha", factor = 0.4046,
                              row = NULL, col = 19, rel = FALSE,
                              id = NULL, level = NULL)))

normalise(what = "tables",
          nation = c("brazil", "united states"),
          faoID = list(commodities = "simpleName"),
          update = TRUE, verbose = FALSE)
```

Listing 5. Normalise the data

Discussion

We have developed the R package `arealDB`, which provides so far missing software for harmonising and integrating areal data across a wide range of heterogeneous sources into a single, consistent database. We described the three stages 'project setup', 'data registration' and 'data normalisation', following the default workflow of `arealDB`. This workflow results in a tidy areal database in which each of the variables follows the same semantic logic in the same language and where areal data refers to spatial units that are consistently matched (Tab. 2). In the following, we discuss how the outlined challenges have been solved and which limitations remain.

Table 2. The header of a final table of areal data. All variables are tidy, `tabID` and `geoID` document the origin of each observation and `ahID` refers to geometries that are harmonised and integrated in the spatial part of the areal database (not shown here).

ID	tabID	geoID	id varaible ahID	id varaible timestep	id varaible commodity	values varaible production
1	1	1	070017008	2016	maize	15000
2	1	1	070017008	2016	wheat	12000
3	1	1	070017008	2017	maize	14000
...						

Georeferencing data

In `arealDB`, every single areal dataset may be linked to individual geometries. This procedure avoids potentially erroneous assumptions when georeferencing areal data on a single, overarching source of geometries, but requires that the individual geometries are sorted into an initial hierarchical structure of territorial units so that the correct geometries match with each another. Often, a simple spatial join of geometries that overlap is sufficient to match geometries (which is computed with functions of the R package `sf` (Pebesma [2018]) in `arealDB`). Geometries are matched, by default, only if they overlap with more than 90%. Thus, in some cases no proper match can be found, either because two geometries are in fact not the same or despite they are the same but with a small overlap.

For example, let there be four territorial units and an administrative reform, where two of those units would be merged into a new unit. For data later than the reform, the ahID of the two untouched units is still valid, but the two modified units are no longer valid. `arealDB` solves this issue by assigning a new, fifth ahID to the then third unit. This new ahID would then be matched with the areal data that are valid from this reform onwards, while the old ahIDs would not be used for data beyond this date. Territorial units that do not match because of any other reason, for example due to a too low overlap, are treated the same. Eventually, the resulting database could be regarded as a mere list of geometries that are matched with specific areal data, which makes the sole assumption that geometries that have a spatial join have the same ahID.

Another difficulty lies in names that are shared by various distinct territorial units located in different nations, or at different administrative levels. In `arealDB`, the names of territorial units are matched hierarchically, i.e., only at the administrative level at which the units are valid and only within the valid parent unit, which is usually given by the context. This procedure ensures, for example, that areal data from cities are not accidentally assigned to municipalities with the same name and that statistics that are valid for a unit in a particular nation are not erroneously assigned to a unit with the same name in another nation. To summarise, in `arealDB` territorial units are matched according to their spatial information and not according to their names.

Alternative versions of data

Alternative versions of data may occur whenever more than one authority provides areal data for territorial units, for example, in disputed areas or when different measurement campaigns record data. An issue with alternative data typically arises when distinct authorities disagree about the values of the phenomenon in focus, and statistical inconsistencies occur.

This issue is handled in `arealDB` likewise by providing the option that every single areal dataset may be linked to individual geometries. When each areal dataset refers to specific geometries, alternative versions of data can dwell next to one another. Areal data from distinct, perhaps disputed sources can thus be assigned to all parties that claim possession or responsibility, with the same ahID for the same territorial unit. The integrated database can be grouped by ahID of the units in question and can be summarised by whichever routine is deemed adequate, to resolve statistical inconsistencies before further analyses.

Translating terms

When handling data from sources that span large spatial extents, they are likely presented in different languages, which have to be harmonised. However, terms may be provided not only in different languages (sensu stricto) but also with distinct semantic meaning. For example, the concept of *patch of land that is dominated by grassy vegetation and on which cattle graze*, could be called pasture (British English), but also rangeland (American English), and is called pastagem in Portuguese. The language translation `"pastagem <-> pasture"` from Portuguese to English is functionally similar to the semantic translation `"rangeland <-> pasture"`. Both examples are cases of many-to-one translations because terms in different languages (sensu lato) refer to the same term in the target language. Additionally, in a particular dataset, the term 'pasture' may have been used for a habitat that would best be described as 'grassland', as would become evident from a potentially available dataset description. Here, an individual term refers to different concepts, depending on where it originates, which constitutes a one-to-many semantic translation.

The function `translateTerms()` manages all translations by comparing new terms individually and explicitly with the translation tables that have been created in step one. The user is provided with an interface that suggests a range of terms pre-selected from the translation table via fuzzy matching. Then, the missing translations have to be provided by the user, so that finally the new terms can be compared against the look-up section of a translation table to check for consistent translation.

Many-to-one translations are handled quite straightforwardly, the target value is repeated in the column `target` and terms that refer to it are recorded in the column `origin`. One-to-many translations need to be provided, in the column `tabID`, with the areal dataset from which the terms originate (Tab. 3).

Table 3. A translation table that includes (a) many-to-one translations (lines 1 and 2) and a case of one-to-many translations (line 3), as well as (b) language (line 1) and semantic (lines 2 and 3) translations of the term 'pasture'.

origin	target	source	ID	notes
pastagem	pasture	tabID	1	
rangeland	pasture	tabID	2	
pasture	grassland	tabID	3	
...				

Documenting metadata

It is often the case that data are provided without suitable metadata that document provenance or other aspects that allow evaluation of data quality (Henzen et al. [2013]). All data management, including the processes employed in `arealDB` to harmonise and reshape areal data, is prone to (human) error. For example, translation of terms may be erroneous, false columns or rows may be selected for reshaping, the unit of values may be misinterpreted or a false file may be processed. Hence, the process of tidying messy data is of the utmost interest in assessing the quality of a database.

To avoid errors due to unsuitable translations, one can build on previously outlined standardised ontologies, such as the Humboldt Core (Guralnick et al. [2018]) or the FAO Commodity list [7], as outlined in 'project setup'. Moreover, `arealDB` records information on the origin and modification of areal data (in the form of schema descriptions) and inserts IDs that identify the source of each data point into the database (see Tab. 2)). Finally, all metadata provided by the user are checked for consistency via the R package `checkmate` (Lang [2017]).

All of this increases transparency by allowing to trace back inconsistencies that might show only later on in the analysis pipeline, and thus invokes more trust when repurposing databases assembled with `arealDB` for different downstream applications (Henzen et al. [2013]).

Outlook

`arealDB` allows to further democratise global efforts of integrating areal data, also at the sub-national level. The standards suggested here enable areal databases that are structurally interoperable with one another, without requiring a central authority. However, the endeavour of integrating databases also across knowledge domains requires advances in ontological standardisation. Various fields are developing so-called essential variables (Reyers et al. [2017]), and `arealDB` allows to build on those advances to come up with areal databases that are interoperable also ontologically.

A database that is built using `arealDB` is a relational database, despite it not being provided as SQL or PostGIS data structure. A possible extension, which we may include into the package in the future, is thus a convenience option to choose between different forms of output.

Moreover, based on the `httr` package (Wickham [2018]), it is possible to develop further functions or packages that download information from various sources that provide a stable API or otherwise standardised access. In combining such efforts with the strict metadata documentation and schema descriptions of `arealDB`, an automated pipeline for downloading and integrating areal data to stage three quality would be possible.

References

EF Codd. *The Relational Model for Database Management: Version 2*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990. ISBN 0-201-14192-2.

Steffen Ehrmann. arealDB: Manage areal census and survey data, September 2019. URL `https://doi.org/10.5281/zenodo.3407297`.

Robert Guralnick, Ramona Walls, and Walter Jetz. Humboldt core–toward a standardized capture of biological inventories for biodiversity monitoring, modeling and assessment. *Ecography*, 41(5):713–725, 2018. doi: 10.1111/ecog.02942.

Christin Henzen, Stephan Mäs, and Lars Bernard. *Provenance Information in Geodata Infrastructures*, pages 133–151. Springer, 2013. ISBN 978-3-319-00614-7. doi: 10.1007/978-3-319-00615-4_8.

T. Horton, A. Kroh, S. Ahyong, N. Bailly, C.B. Boyko, S.N. Brandão, S. Gofas, J.N.A. Hooper, F. Hernandez, O. Holovachov, J. Mees, T.N. Molodtsova, G. Paulay, W. Decock, S. Dekeyzer, T. Lanssens, L. Vandepitte, B. Vanhoorne, K. Verfaille, R. Adlard, P. Adriaens, S. Agatha, K.J. Ahn, N. Akkari, B. Alvarez, G. Anderson, M.V. Angel, C. Arango, T. Artois, S. Atkinson, R. Bank, A. Barber, J.P. Barbosa, I. Bartsch, D. Bellan-Santini, J. Bernot, A. Berta, T.N. Bezerra, R. Bieler, S. Blanco, I. Blasco-Costa, M. Blazewicz, P. Bock, R. Böttger-Schnack, P. Bouchet, N. Boury-Esnault, G. Boxshall, R. Bray, B. Breure, N.L. Bruce, S. Cairns, P. Cárdenas, E. Carstens, B.K. Chan, T.Y. Chan, L. Cheng, M. Churchill, C.O. Coleman, A.G. Collins, L. Corbari, R. Cordeiro, A. Cornils, M. Coste, M.J. Costello, K.A. Crandall, F. Cremonte, T. Cribb, S. Cutmore, F. Dahdouh-Guebas, M. Daly, M. Daneliya, J.C. Dauvin, P. Davie, C. De Broyer, S. De Grave, V. de Mazancourt, N.J. de Voogd, P. Decker, W. Decraemer, D. Defaye, J.L. d'Hondt, S. Dippenaar, M. Dohrmann, J. Dolan, D. Domning, R. Downey, L. Ector, U. Eisendle-Flöckner, M. Eitel, S.C.d. Encarnação, H. Enghoff, J. Epler, C. Ewers-Saucedo, M. Faber, S. Feist, D. Figueroa, J. Finn, C. Fišer, E. Fordyce, W. Foster, J.H. Frank, C. Fransen, H. Furuya, H. Galea, O. Garcia-Alvarez, R. Garic, R. Gasca, S. Gaviria-Melo, S. Gerken, D. Gibson, R. Gibson, J. Gil, A. Gittenberger, C. Glasby, A. Glover, S.E. Gómez-Noguera, D. González-Solís, D. Gordon, M. Grabowski, C. Gravili, J.M.. Guerra-García, R. Guidetti, M.D. Guiry, K.A. Hadfield, E. Hajdu, J. Hallermann, B.W. Hayward, E. Hendrycks, D. Herbert, A. Herrera Bachiller, J.s. Ho, M. Hodda, J. Høeg, B. Hoeksema, R. Houart, L. Hughes, M. Hyžný, L.F.M. Iniesta, T. Iseto, S. Ivanenko, M. Iwataki, R. Janssen, G. Jarms, D. Jaume, K. Jazdzewski, C.D. Jersabek, P. Jóźwiak, A. Kabat, Y. Kantor, I. Karanovic, B. Karthick, Y.H. Kim, R. King, P.M. Kirk, M. Klautau, J.P. Kociolek, F. Köhler, J. Kolb, A. Kotov, A. Kremenetskaia, R.M. Kristensen, M. Kulikovskiy, S. Kullander, G. Lambert, D. Lazarus, F. Le Coze, S. LeCroy, D. Leduc, E.J. Lefkowitz, R. Lemaitre,

[7] http://www.fao.org/economic/ess/ess-standards/commodity/en/

Y. Liu, A.N. Lörz, J. Lowry, T. Ludwig, N. Lundholm, E. Macpherson, L. Madin, C. Mah, B. Mamo, T. Mamos, R. Manconi, G. Mapstone, P.E. Marek, B. Marshall, D.J. Marshall, P. Martin, R. Mast, C. McFadden, S.J. McInnes, T. Meidla, K. Meland, K.L. Merrin, C. Messing, D. Miljutin, C. Mills, Ø. Moestrup, V. Mokievsky, F. Monniot, R. Mooi, A.C. Morandini, R. Moreira da Rocha, F. Moretzsohn, C. Morrow, J. Mortelmans, J. Mortimer, L. Musco, T.A. Neubauer, E. Neubert, B. Neuhaus, P. Ng, A.D. Nguyen, C. Nielsen, T. Nishikawa, J. Norenburg, T. O'Hara, D. Opresko, M. Osawa, H.J. Osigus, Y. Ota, B. Páll-Gergely, D. Patterson, H. Paxton, R. Peña-Santiago, V. Perrier, W. Perrin, I. Petrescu, B. Picton, J.F. Pilger, A.B. Pisera, D. Polhemus, G.C. Poore, M. Potapova, P. Pugh, G. Read, M. Reich, J.D. Reimer, H. Reip, M. Reuscher, J.W. Reynolds, I. Richling, F. Rimet, P. Ríos, M. Rius, D.C. Rogers, G. Rosenberg, K. Rützler, K. Sabbe, J. Saiz-Salinas, S. Sala, S. Santagata, S. Santos, E. Sar, A. Satoh, T. Saucède, H. Schatz, B. Schierwater, A. Schmidt-Rhaesa, S. Schneider, C. Schönberg, P. Schuchert, A.R. Senna, C. Serejo, S. Shaik, S. Shamsi, J. Sharma, W.A. Shear, N. Shenkar, A. Shinn, M. Short, J. Sicinski, P. Sierwald, E. Simmons, F. Sinniger, D. Sivell, B. Sket, H. Smit, N. Smit, N. Smol, J.F.. Souza-Filho, J. Spelda, W. Sterrer, E. Stienen, P. Stoev, S. Stöhr, M. Strand, E. Suárez-Morales, M. Summers, L. Suppan, C. Suttle, B.J. Swalla, S. Taiti, M. Tanaka, A.H. Tandberg, D. Tang, M. Tasker, J. Taylor, J. Taylor, A. Tchesunov, H. ten Hove, J.J. ter Poorten, J.D. Thomas, E.V. Thuesen, M. Thurston, B. Thuy, J.T. Timi, T. Timm, A. Todaro, X. Turon, S. Tyler, P. Uetz, J. Uribe-Palomino, S. Utevsky, J. Vacelet, D. Vachard, W. Vader, R. Väinölä, B. Van de Vijver, S.E. van der Meij, T. van Haaren, R.W. van Soest, A. Vanreusel, V. Venekey, M. Vinarski, R. Vonk, C. Vos, G. Walker-Smith, T.C. Walter, L. Watling, M. Wayland, T. Wesener, C.E. Wetzel, C. Whipps, K. White, U. Wieneke, D.M. Williams, G. Williams, R. Wilson, A. Witkowski, J. Witkowski, N. Wyatt, C. Wylezich, K. Xu, J. Zanol, W. Zeidler, and Z. Zhao. World register of marine species (worms). http://www.marinespecies.org, 2019. URL `http://www.marinespecies.org`. Accessed: 2019-09-11.

Christian König, Patrick Weigelt, Julian Schrader, Amanda Taylor, Jens Kattge, and Holger Kreft. Biodiversity data integration–the significance of data resolution and domain. *PLoS biology*, 17(3):e3000183, 2019. doi: 10.1371/journal.pbio.3000183.

Michel Lang. checkmate: Fast Argument Checks for Defensive R Programming. *The R Journal*, 9(1):437–445, 2017. doi: 10.32614/RJ-2017-028.

Christiaan Lemmen, Peter Van Oosterom, and Rohan Bennett. The land administration domain model. *Land use policy*, 49:535–545, 2015. doi: 10.1016/j.landusepol.2015.01.014.

Christopher T. Lloyd, Heather Chamberlain, David Kerr, Greg Yetman, Linda Pistolesi, Forrest R. Stevens, Andrea E. Gaughan, Jeremiah J. Nieves, Graeme Hornby, Kytt MacManus, Parmanand Sinha, Maksym Bondarenko, Alessandro Sorichetta, and Andrew J. Tatem. Global spatio-temporally harmonised datasets for producing high-resolution gridded population distribution datasets. *Big Earth Data*, 0(0):1–32, 2019. doi: 10.1080/20964471.2019.1625151.

Stephan Mäs, Daniel Henzen, Lars Bernard, Matthias Müller, Simon Jirka, and Ivo Senner. Generic schema descriptions for comma-separated values files of environmental data. In *The 21th AGILE International Conference on Geographic Information Science*, 2018. URL `https://agile-online.org/conference_paper/cds/agile_2018/shortpapers/118M%C3%A4s-ShortPaper.pdf`.

Ilona M Otto, Anne Biewald, Dim Coumou, Georg Feulner, Claudia Köhler, Thomas Nocke, Anders Blok, Albert Gröber, Sabine Selchow, David Tyfield, et al. Socio-economic data for global environmental change research. *Nature Climate Change*, 5(6):503, 2015. doi: 10.1038/nclimate2593.

Edzer Pebesma. Simple Features for R: Standardized Support for Spatial Vector Data. *The R Journal*, 10(1):439–446, 2018. doi: 10.32614/RJ-2018-009.

Belinda Reyers, Mark Stafford-Smith, Karl-Heinz Erb, Robert J Scholes, and Odirilwe Selomane. Essential variables help to focus sustainable development goals monitoring. *Current Opinion in Environmental Sustainability*, 26: 97–105, 2017. doi: 10.1016/j.cosust.2017.05.003.

Forrest R Stevens, Andrea E Gaughan, Catherine Linard, and Andrew J Tatem. Disaggregating census data for population mapping using random forests with remotely-sensed and ancillary data. *PloS one*, 10(2):e0107042, 2015. doi: 10.1371/journal.pone.0107042.

Patrick Weigelt, Christian König, and Holger Kreft. Gift-a global inventory of floras and traits for macroecology and biogeography. *BioRxiv*, page 535005, 2019. doi: 10.1101/535005.

Hadley Wickham. Tidy data. *Journal of Statistical Software*, 59(10):1–23, 2014. doi: 10.18637/jss.v059.i10.

Hadley Wickham. *httr: Tools for Working with URLs and HTTP*, 2018. URL `https://CRAN.R-project.org/package=httr`. R package version 1.4.0.

John Wieczorek, David Bloom, Robert Guralnick, Stan Blum, Markus Döring, Renato Giovanni, Tim Robertson, and David Vieglais. Darwin core: an evolving community-developed biodiversity data standard. *PloS One*, 7(1):e29715, 2012. doi: 10.1371/journal.pone.0029715.