Package 'arealDB'

July 3, 2023

Type Package

Title Harmonise and Integrate Heterogeneous Areal Data

Description Many relevant applications in the environmental and socioeconomic sciences use areal data, such as biodiversity checklists, agricultural statistics, or socioeconomic surveys. For applications that surpass the spatial, temporal or thematic scope of any single data source, data must be integrated from several heterogeneous sources. Inconsistent concepts, definitions, or messy data tables make this a tedious and error-prone process. 'arealDB' tackles those problems and helps the user to integrate a harmonised databases of areal data. Read the paper at Ehrmann, Seppelt & Meyer (2020) <doi:10.1016/j.envsoft.2020.104799>.

Version 0.6.3

URL https://github.com/luckinet/arealDB

BugReports https://github.com/luckinet/arealDB/issues

Depends R (>= 2.10)

Language en-gb

License GPL-3

Encoding UTF-8

Imports checkmate, dplyr, magrittr, readr, rlang, sf, stringr, tibble, tidyr, tidyselect, tabshiftr, ontologics, purrr, rmapshaper, progress

Suggests testthat, knitr, rmarkdown, bookdown, covr

RoxygenNote 7.2.3

VignetteBuilder knitr

NeedsCompilation no

Author Steffen Ehrmann [aut, cre] (<https://orcid.org/0000-0002-2958-0796>), Arne Rümmler [aut, ctb] (<https://orcid.org/0000-0001-8637-9071>), Felipe Melges [ctb] (<https://orcid.org/0000-0003-0833-8973>), Carsten Meyer [aut] (<https://orcid.org/0000-0003-3927-5856>)

Maintainer Steffen Ehrmann <steffen.ehrmann@posteo.de>

Repository CRAN

Date/Publication 2023-07-03 10:00:02 UTC

R topics documented:

`getColTypes` *Get the column types of a tibble*

Description

(internal function not for user interaction)

Usage

```
getColTypes(input = NULL)
```

Arguments

`input`	`[tibble(1)]` tibble from which to get column types.

`makeExampleDB` *Build an example database*

Description

This function helps setting up an example database up until a certain step.

Usage

```
makeExampleDB(path = NULL, until = NULL, verbose = FALSE)
```

Arguments

path	[character(1)] The database gets created by default in tempdir(), but if you want it in a particular location, specify that in this argument.
until	[character(1)] The database building step in terms of the function names until which the example database shall be built, one of "start_arealDB", "regDataseries", "regGeometry", "regTable", "normGeometry" or "normTable".
verbose	[logical(1)] be verbose about building the example database (default FALSE).

Details

Setting up a database with an R-based tool can appear to be cumbersome and too complex and thus intimidating. By creating an example database, this functions allows interested users to learn step by step how to build a database of areal data. Moreover, all functions in this package contain verbose information and ask for information that would be missing or lead to an inconsistent database, before a failure renders hours of work useless.

Value

No return value, called for the side effect of creating an example database at the specified path.

Examples

```
if(dev.interactive()){
# to build the full example database
makeExampleDB(path = paste0(tempdir(), "/newDB"))

# to make the example database until a certain step
makeExampleDB(path = paste0(tempdir(), "/newDB"), until = "regDataseries")

}
```

---

matchOntology *Match target terms with an ontology*

Description

This function takes a table to replace the values of various columns with harmonised values listed in the project specific gazetteer.

Usage

```
matchOntology(
  table = NULL,
  columns = NULL,
  dataseries = NULL,
  ontology = NULL,
  verbose = FALSE,
  beep = NULL
)
```

Arguments

table	[character(1)] a table that contains columns that should be harmonised by matching with the gazetteer.
columns	[character(1)] the columns containing the concepts
dataseries	[character(1)] the source dataseries from which territories are sourced.
ontology	[onto] either a path where the ontology/gazetteer is stored, or an already loaded ontology.
verbose	[‘logical(1)‘][logical] whether or not to give detailed information on the process of this function.
beep	[integerish(1)] Number specifying what sound to be played to signal the user that a point of interaction is reached by the program, see beep.

Value

Returns a table that resembles the input table where the target columns were translated according to the provided ontology.

`normGeometry` *Normalise geometries*

Description

Harmonise and integrate geometries into a standardised format

Usage

```
normGeometry(
  input = NULL,
  pattern = NULL,
  query = NULL,
  thresh = 10,
  outType = "gpkg",
  priority = "ontology",
  beep = NULL,
  simplify = FALSE,
  update = FALSE,
  verbose = FALSE
)
```

Arguments

input	[character(1)] path of the file to normalise. If this is left empty, all files at stage two as subset by pattern are chosen.
pattern	[character(1)] an optional regular expression. Only dataset names which match the regular expression will be processed.
query	[character(1)] part of the SQL query (starting from WHERE) used to subset the input geometries, for example where NAME_0 = 'France'. The first part of the query (where the layer is defined) is derived from the meta-data of the currently handled geometry.
thresh	[integerish(1)]
outType	[character(1)] the output file-type, see st_drivers for a list. If a file-type supports layers, they are stored in the same file, otherwise the different layers are provided separately. For an R-based workflow, "rds" could be an efficient option.
priority	[character(1)] how to match the new geometries with the already harmonised database. This can either be • "spatial": where all territories are intersected spatially or • "ontology": where territories are matched by comparing their name with the ontology and those that do not match are intersected spatially, • "both": where territories are matched with the ontology and spatially, and conflicts are indicated
beep	[integerish(1)] Number specifying what sound to be played to signal the user that a point of interaction is reached by the program, see beep.
simplify	[logical(1)] whether or not to simplify geometries.

update [logical(1)]
whether or not the physical files should be updated (TRUE) or the function should merely return the geometry inventory of the handled files (FALSE, default). This is helpful to check whether the metadata specification and the provided file(s) are properly specified.

verbose [logical(1)]
be verbose about what is happening (default FALSE). Furthermore, you can use suppressMessages to make this function completely silent.

Details

To normalise geometries, this function proceeds as follows:

1. Read in input and extract initial metadata from the file name.
2. In case filters are set, the new geometry is filtered by those.
3. The territorial names are matched with the gazetteer to harmonise new territorial names (at this step, the function might ask the user to edit the file 'matching.csv' to align new names with already harmonised names).
4. Loop through every nation potentially included in the file that shall be processed and carry out the following steps:
   - In case the geometries are provided as a list of simple feature POLYGONS, they are dissolved into a single MULTIPOLYGON per main polygon.
   - In case the nation to which a geometry belongs has not yet been created at stage three, the following steps are carried out:
     (a) Store the current geometry as basis of the respective level (the user needs to make sure that all following levels of the same dataseries are perfectly nested into those parent territories, for example by using the GADM dataset)
   - In case the nation to which the geometry belongs has already been created, the following steps are carried out:
     (a) Check whether the new geometries have the same coordinate reference system as the already existing database and re-project the new geometries if this is not the case.
     (b) Check whether all new geometries are already exactly matched spatially and stop if that is the case.
     (c) Check whether the new geometries are all within the already defined parents, and save those that are not as a new geometry.
     (d) Calculate spatial overlap and distinguish the geometries into those that overlap with more and those with less than thresh.
     (e) For all units that did match, copy gazID from the geometries they overlap.
     (f) For all units that did not match, rebuild metadata and a new gazID.
   - If update = TRUE, store the processed geometry at stage three.
5. Move the geometry to the folder '/processed', if it is fully processed.

Value

This function harmonises and integrates so far unprocessed geometries at stage two into stage three of the geospatial database. It produces for each main polygon (e.g. nation) in the registered geometries a spatial file of the specified file-type.

See Also

Other normalise functions: `normTable()`

Examples

```
if(dev.interactive()){
  library(sf)

  # build the example database
  makeExampleDB(until = "regGeometry", path = tempdir())

  # normalise all geometries ...
  normGeometry(nation = "estonia", update = TRUE)

  # ... and check the result
  st_layers(paste0(tempdir(), "/adb_geometries/stage3/Estonia.gpkg"))
  output <- st_read(paste0(tempdir(), "/adb_geometries/stage3/Estonia.gpkg"))
}
```

`normTable` *Normalise data tables*

Description

Harmonise and integrate data tables into standardised format

Usage

```
normTable(
  input = NULL,
  pattern = NULL,
  ontoMatch = NULL,
  outType = "rds",
  beep = NULL,
  update = FALSE,
  verbose = FALSE
)
```

Arguments

`input`	[`character(1)`] path of the file to normalise. If this is left empty, all files at stage two as subset by `pattern` are chosen.
`pattern`	[`character(1)`] an optional regular expression. Only dataset names which match the regular expression will be processed.
`ontoMatch`	[`character(.)`] name of the column(s) that shall be matched with an ontology (defined in `start_arealDB`).

outType [logical(1)]
the output file-type, currently implemented options are either **.csv* (more exchangeable for a workflow based on several programs) or **.rds* (smaller and less error-prone data-format but can only be read by R efficiently).

beep [integerish(1)]
Number specifying what sound to be played to signal the user that a point of interaction is reached by the program, see beep.

update [logical(1)]
whether or not the physical files should be updated (TRUE) or the function should merely return the new object (FALSE, default). This is helpful to check whether the metadata specification and the provided file(s) (translation and ID tables) are properly specified.

verbose [logical(1)]
be verbose about translating terms (default FALSE). Furthermore, you can use suppressMessages to make this function completely silent.

Details

To normalise data tables, this function proceeds as follows:

1. Read in input and extract initial metadata from the file name.
2. Employ the function tabshiftr::reorganise to reshape input according to the respective schema description.
3. The territorial names are matched with the gazetteer to harmonise new territorial names (at this step, the function might ask the user to edit the file 'matching.csv' to align new names with already harmonised names).
4. Harmonise territorial unit names.
5. If update = TRUE, store the processed data table at stage three.

Value

This function harmonises and integrates so far unprocessed data tables at stage two into stage three of the areal database. It produces for each main polygon (e.g. nation) in the registered data tables a file that includes all thematic areal data.

See Also

Other normalise functions: normGeometry()

Examples

```
if(dev.interactive()){
  # build the example database
  makeExampleDB(until = "normGeometry", path = tempdir())

  # normalise all available data tables ...
  normTable(update = TRUE)
```

```
  # ... and check the result
  output <- readRDS(paste0(tempdir(), "/adb_tables/stage3/Estonia.rds"))
}
```

---

regDataseries *Register a new dataseries*

Description

This function registers a new dataseries of both, geometries or areal data into the geospatial database. This contains the name and relevant meta-data of a dataseries to enable provenance tracking and reproducability.

Usage

```
regDataseries(
  name = NULL,
  description = NULL,
  homepage = NULL,
  licence_link = NULL,
  licence_path = NULL,
  notes = NULL,
  update = FALSE,
  overwrite = FALSE
)
```

Arguments

name	[character(1)] the dataseries abbreviation or name.
description	[character(1)] the "long name" or "brief description" of the dataseries.
homepage	[character(1)] the homepage of the data provider where the dataseries or additional information can be found.
licence_link	[character(1)] link to the licence or the webpage from which the licence was copied.
licence_path	[character(1)] path to the local file in which the licence text is stored.
notes	[character(1)] optional notes.
update	[logical(1)] whether or not the file 'inv_dataseries.csv' should be updated (obligatory to continue registering geometries or tables associated to this dataseries).
overwrite	[logical(1)] whether or not the dataseries to register shall overwrite a potentially already existing older version.

Value

Returns a tibble of the new entry that is appended to 'inv_dataseries.csv' in case `update = TRUE`.

See Also

Other register functions: `regGeometry()`, `regTable()`

Examples

```
if(dev.interactive()){
  # start the example database
  makeExampleDB(until = "match_gazetteer", path = tempdir())

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

---

regGeometry *Register a new geometry entry*

Description

This function registers a new geometry of territorial units into the geospatial database.

Usage

```
regGeometry(
  ...,
  subset = NULL,
  gSeries = NULL,
  label = NULL,
  layer = NULL,
  archive = NULL,
  archiveLink = NULL,
  nextUpdate = NULL,
  updateFrequency = NULL,
  notes = NULL,
  update = FALSE,
  overwrite = FALSE
)
```

Arguments

`...`	[character(1)] optional named argument selecting the main territory into which this geometry is nested. The name of this must be a class of the gazetteer and the value must be one of the territory names of that class, e.g. *nation = "Estonia"*.
`subset`	[character(1)] optional argument to specify which subset the file contains. This could be a subset of territorial units (e.g. only one municipality) or of a target variable.
`gSeries`	[character(1)] the name of the geometry dataseries (see `regDataseries`).
`label`	[list(.)] list of as many columns as there are in common in the ontology and this geometry. Must be of the form `list(class = columnName)`, with 'class' as the class of the ontology corresponding to the respective column name in the geometry.
`layer`	[character] the name of the file's layer from which the geometry should be created (if applicable).
`archive`	[character(1)] the original file (perhaps a *.zip) from which the geometry emerges.
`archiveLink`	[character(1)] download-link of the archive.
`nextUpdate`	[character(1)] value describing the next anticipated update of this dataset (in YYYY-MM-DD format).
`updateFrequency`	[character(1)] value describing the frequency with which the dataset is updated, according to the ISO 19115 Codelist, MD_MaintenanceFrequencyCode. Possible values are: 'continual', 'daily', 'weekly', 'fortnightly', 'quarterly', 'biannually', 'annually', 'asNeeded', 'irregular', 'notPlanned', 'unknown', 'periodic', 'semimonthly', 'biennially'.
`notes`	[character(1)] optional notes that are assigned to all features of this geometry.
`update`	[logical(1)] whether or not the file 'inv_geometries.csv' should be updated.
`overwrite`	[logical(1)] whether or not the geometry to register shall overwrite a potentially already existing older version.

Details

When processing geometries to which areal data shall be linked, carry out the following steps:

1. Determine the main territory (such as a nation, or any other polygon), a `subset` (if applicable), the dataseries of the geometry and the ontology `label`, and provide them as arguments to this function.

2. Run the function.
3. Export the shapefile with the following properties:
   - Format: GeoPackage
   - File name: What is provided as message by this function
   - CRS: EPSG:4326 - WGS 84
   - make sure that 'all fields are exported'
4. Confirm that you have saved the file.

Value

Returns a tibble of the entry that is appended to 'inv_geometries.csv' in case `update = TRUE`.

See Also

Other register functions: `regDataseries()`, `regTable()`

Examples

```
if(dev.interactive()){
  # build the example database
  makeExampleDB(until = "regDataseries", path = tempdir())

  # The GADM dataset comes as *.7z archive
  regGeometry(gSeries = "gadm",
              label = list(al1 = "NAME_0"),
              layer = "example_geom1",
              archive = "example_geom.7z|example_geom1.gpkg",
              archiveLink = "https://gadm.org/",
              nextUpdate = "2019-10-01",
              updateFrequency = "quarterly",
              update = TRUE)

  # The second administrative level in GADM contains names in the columns
  # NAME_0 and NAME_1
  regGeometry(gSeries = "gadm",
              label = list(al1 = "NAME_0", al2 = "NAME_1"),
              layer = "example_geom2",
              archive = "example_geom.7z|example_geom2.gpkg",
              archiveLink = "https://gadm.org/",
              nextUpdate = "2019-10-01",
              updateFrequency = "quarterly",
              update = TRUE)
}
```

regTable *Register a new areal data table*

Description

This function registers a new areal data table into the geospatial database.

Usage

```
regTable(
  ...,
  subset = NULL,
  dSeries = NULL,
  gSeries = NULL,
  label = NULL,
  begin = NULL,
  end = NULL,
  schema = NULL,
  archive = NULL,
  archiveLink = NULL,
  nextUpdate = NULL,
  updateFrequency = NULL,
  metadataLink = NULL,
  metadataPath = NULL,
  notes = NULL,
  update = FALSE,
  overwrite = FALSE
)
```

Arguments

...	[character(1)] name and value of the topmost unit under which the table shall be registered. The name of this must be a class of the gazetteer and the value must be one of the territory names of that class, e.g. *nation = "Estonia"*.
subset	[character(1)] optional argument to specify which subset the file contains. This could be a subset of territorial units (e.g. only one municipality) or of a target variable.
dSeries	[character(1)] the dataseries of the areal data (see regDataseries).
gSeries	[character(1)] optionally, the dataseries of the geometries, if the geometry dataseries deviates from the dataseries of the areal data (see regDataseries).
label	[integerish(1)] the label in the onology this geometry should correspond to.

`begin` [integerish(1)]
the date from which on the data are valid.

`end` [integerish(1)]
the date until which the data are valid.

`schema` [list(1)]
the schema description of the table to read in (must have been placed in the global environment before calling it here).

`archive` [character(1)]
the original file from which the boundaries emerge.

`archiveLink` [character(1)]
download-link of the archive.

`nextUpdate` [character(1)]
when does the geometry dataset get updated the next time (format restricted to: YYYY-MM-DD).

`updateFrequency`
[character(1)]
value describing the frequency with which the dataset is updated, according to the ISO 19115 Codelist, MD_MaintenanceFrequencyCode. Possible values are: 'continual', 'daily', 'weekly', 'fortnightly', 'quarterly', 'biannually', 'annually', 'asNeeded', 'irregular', 'notPlanned', 'unknown', 'periodic', 'semimonthly', 'biennially'.

`metadataLink` [character(1)]
if there is already metadata existing: link to the meta dataset.

`metadataPath` [character(1)]
if an existing meta dataset was downloaded along the data: the path where it is stored locally.

`notes` [character(1)]
optional notes.

`update` [logical(1)]
whether or not the file 'inv_tables.csv' should be updated.

`overwrite` [logical(1)]
whether or not the geometry to register shall overwrite a potentially already existing older version.

Details

When processing areal data tables, carry out the following steps:

1. Determine the main territory (such as a nation, or any other polygon), a `subset` (if applicable), the ontology `label` and the dataseries of the areal data and of the geometry, and provide them as arguments to this function.
2. Provide a `begin` and `end` date for the areal data.
3. Run the function.
4. (Re)Save the table with the following properties:
    - Format: csv

- Encoding: UTF-8
- File name: What is provided as message by this function
- make sure that the file is not modified or reshaped. This will happen during data normalisation via the schema description, which expects the original table.

5. Confirm that you have saved the file.

Every areal data dataseries (`dSeries`) may come as a slight permutation of a particular table arrangement. The function `normTable` expects internally a schema description (a list that describes the position of the data components) for each data table, which is saved as `paste0("meta_", dSeries, TAB_NUMBER)`. See package `tabshiftr`.

Value

Returns a tibble of the entry that is appended to 'inv_tables.csv' in case `update = TRUE`.

See Also

Other register functions: `regDataseries()`, `regGeometry()`

Examples

```
if(dev.interactive()){
  # build the example database
  makeExampleDB(until = "regGeometry", path = tempdir())

  # the schema description for this table
  library(tabshiftr)

  schema_madeUp <-
    setIDVar(name = "al1", columns = 1) %>%
    setIDVar(name = "year", columns = 2) %>%
    setIDVar(name = "commodities", columns = 3) %>%
    setObsVar(name = "harvested",
              factor = 1, columns = 4) %>%
    setObsVar(name = "production",
              factor = 1, columns = 5)

  regTable(nation = "Estonia",
           subset = "barleyMaize",
           dSeries = "madeUp",
           gSeries = "gadm",
           level = 1,
           begin = 1990,
           end = 2017,
           schema = schema_madeUp,
           archive = "example_table.7z|example_table1.csv",
           archiveLink = "...",
           nextUpdate = "2019-10-01",
           updateFrequency = "quarterly",
           metadataLink = "...",
           metadataPath = "my/local/path",
```

```
            update = TRUE)
}
```

`start_arealDB` *Set the root path*

Description

Initiate a geospatial database or register a database that exists at the root path.

Usage

```
start_arealDB(root = NULL, gazetteer = NULL, top = NULL, ontology = NULL)

setPath(root = NULL)
```

Arguments

`root`	[`character(1)`] path to the root directory that contains or shall contain an areal database.
`gazetteer`	[`character(1)`] path to the gazetteer that holds the (hierarchical) information of territorial units used in this database.
`top`	[`character(1)`] the label of the class in the gazetteer that represents the top-most unit (e.g. country) of the areal database that shall be started.
`ontology`	[`list(.)`] named list with the path(s) of ontologies, where the list name identifies the variable that shall be matched with the ontology at the path.

Details

This is the first function that is run in a project, as it initiates the areal database by creating the default sub-directories and initial inventory tables. When a database has already been set up, this function is used to register that path in the options of the current R session.

Value

No return value, called for the side effect of creating the directory structure of the new areal database and tables that contain the database metadata.

Functions

- `setPath()`: deprecated way of starting an areal database

Examples

```
start_arealDB(root = paste0(tempdir(), "/newDB"),
              gazetteer = paste0(tempdir(), "/newDB/territories.rds"),
              top = "al1",
              ontology = list(var = paste0(tempdir(), "/newDB/ontology.rds")))

getOption("adb_path"); getOption("gazetteer_path")
```

---

testCompressed *Test whether a file is a compressed file*

Description

(internal function not for user interaction)

Usage

```
testCompressed(x)
```

Arguments

x	[character(1)] the file name.

Details

This function looks at the file-extension and if it is one of `.gz`, `.bz2`, `.tar` `.zip`, `.tgz`, `.gzip` or `.7z`, it returns the value `TRUE`.

---

updateTable *Update a table*

Description

Update any inventory, index or translation table of an areal database (internal function not meant for user interaction).

Usage

```
updateTable(index = NULL, name = NULL, matchCols = NULL, backup = TRUE)
```

Arguments

index	[tibble(1)] the table to use as update.
name	[character(1)] name of the table that shall be updated.
matchCols	[character(.)] the columns in the old file by which to match.
backup	[logical(1)] whether or not to store the old table in the log directory.

Value

No return value, called for the side-effect of storing a table in a specified location

Index