Package 'tabshiftr'

May 13, 2020

Title Reshape Disorganised Messy Data

Version 0.1.2

Description Helps the user to build and register schema descriptions of disorganised (messy) tables. Disorganised tables are tables that are not in a topologically coherent form, where packages such as 'tidyr' could be used for reshaping. The schema description documents the arrangement of input tables and is used to reshape them into a standardised (tidy) output format.

URL https://github.com/EhrmannS/tabshiftr

BugReports https://github.com/EhrmannS/tabshiftr/issues

Depends R (>= 2.10)

Language en-gb

License GPL-3

Encoding UTF-8

LazyData true

Imports checkmate, rlang, tibble, dplyr, tidyr, magrittr, tidyselect, testthat, crayon, methods, purrr, stringr

RoxygenNote 7.1.0

Suggests knitr, rmarkdown, bookdown, readr, covr

VignetteBuilder knitr

NeedsCompilation no

Author Steffen Ehrmann [aut, cre] (<https://orcid.org/0000-0002-2958-0796>), Abdualmaged Alhemiary [ctb], Amelie Haas [ctb], Annika Ertel [ctb], Arne Rümmler [ctb] (<https://orcid.org/0000-0001-8637-9071>)

Maintainer Steffen Ehrmann <steffen@funroll-loops.de>

Repository CRAN

Date/Publication 2020-05-13 15:10:02 UTC

R topics documented:

`expect_valid_table` *Test for a valid table*

Description

This function is a collection of expectations which ensure that the output of `reorganise` is formally and contentwise correct. It is used in the tests of this package.

Usage

```
expect_valid_table(x = NULL, units = 1)
```

Arguments

`x`	a table to test.
`units`	the number of units in the output table (from 1 to 3)

Value

Either an error message of the invalid expectations, or the output of the last successful expectation.

getMetadata *Derive metadata from a schema description*

Description

This function scrutinises the schema description in response to the data to prepare metadata that are required when reshaping the input data.

Usage

```
getMetadata(data = NULL, schema = NULL)
```

Arguments

data	a list containing the values of selectData.
schema	the schema description that is the basis to derive metadata.

Value

A list of metadata for a single cluster, that simplifies reshaping the data.

getNames *Determine the names from a schema description*

Description

This function determines the specific column names that are required in the process of reshaping.

Usage

```
getNames(header = NULL, meta = NULL)
```

Arguments

header	the header from which to derive names.
meta	the output of getMetadata as basis to derive names.

Value

A vector of column names.

`makeSchema` *Make a schema description*

Description

This function checks whether the schema description is formally correct.

Usage

```
makeSchema(schema = NULL)
```

Arguments

`schema`	`[list(.)]` the list of schema information. This can contain the lists `clusters`, `meta`, `header` and `variables`.

Value

An object of class `schema`.

Setting up schema descriptions

The recommended strategy for setting up a schema description is the following recently.

1. Clarify which are the identifying variables and which are the measured variables and create a new entry for each of them in the schema.
2. Determine whether there are clusters and find the origin (top left cell) of each cluster. Follow the next steps for each cluster...
3. Determine which variable identifies clusters and provide that as cluster ID.
4. Determine for each identifying variable the following:
   - is the variable available at all? If not, provide the variable value for this cluster in 'value'.
   - all columns in which the variable *names* sit.
   - in case the variable is in several columns, determine additionally the row in which its *names* sit.
   - whether the variable is distinct from the main table.
   - whether the variable must be split off of another column.
5. Determine for each measured variable the following:
   - all columns in which the *values* of the variable sit.
   - the unit and conversion factor
   - in case the variable is not tidy, one of the three following cases should apply:
     (a) in case the variable is nested in a wide identifying variable, determine in addition to the columns in which the values sit also the rows in which the *variable name* sits.

(b) in case the names of the variable are given as a value of an identifying variable, give the column name as `key`, together with the respective name of the measured variable in `values`.
(c) in case the name of the variable is the ID of clusters, specify `key = "cluster"` and in `values` the cluster number the variable refers to.

See below for a more detailed description of the fields used in schemas or read the vignette.

Fields of a schema

The following section contains a list of all the fields recently evaluated in a schema. The information is split up into the four sub-sections *clusters*, *header*, *meta* and *variables*.

There is hardly any limit to how data can be arranged in a spreadsheet, apart from the apparent organisation into a lattice of cells. However, it is often the case that data are gathered into topologically coherent chunks. Those chunks are what is considered 'cluster' in arealDB. Clusters are described by the properties:

- `row` [`integerish(1)`]:
  The vertical cell values of the top-left cell of each cluster. This can also be a vector of values in case there are several clusters.
- `col` [`integerish(1)`]:
  The horizontal cell value of the top-left cell of each cluster. This can also be a vector of values in case there are several clusters.
- `width` [`integerish(1)`]:
  The width of each cluster in cells. This can also be a vector of values in case there are several clusters
- `height` [`integerish(1)`]:
  The height of each cluster in cells. This can also be a vector of values in case there are several clusters
- `id` [`character(1)`]:
  When data are clustered, it is often the case that the data are segregated according to one of the variables of interest. In such cases, this variable needs to be registered as cluster ID.

The slot `header` describes in which rows the header informations are stored and how they should be treated. It contains the properties

- `row` [`integerish(.)`]
  The rows in which the header information are stored.
- `rel` [`logical(1)`]
  Whether or not the values in `row` are relative to the cluster positions or whether they refer to the overall table.
- `merge` [`logical(1)`]
  When there is more than one row, this determines whether or not those rows should be merged.

The slot `meta` describes information concerning the values in a spreadsheet. It contains the properties

- `del` [`character(.)`]
  The symbol(s) that are used as delimiter in the table to reorganise.

- `dec` [`character(.)`]
  The symbol(s) that are used as decimal symbol in the table to reorganise.
- `na` [`character(.)`]
  The symbol(s) that are used as "not available" values in the table to reorganise.

Each element in the slot `variables` is a list that describes one of the variables that shall be comprised in the final database. Variables are either so-called *identifying variables*, which indicate observation units, or *measured variables*, which carry the observed values. Identifying variables are described by the properties:

- `type` [`character(1)`]:
  The value `"id"` signals that this is an identifying variable.
- `col` [`integerish(1)`]:
  The column(s) in which the variable values are recorded.
- `row` [`integerish(1)`]:
  The row(s) in which the variable values are recorded.
- `split` [`character(1)`]:
  A regular expression that matches the part of values that are supposed to be part of the variable, when the cells contain more than one value (for example separated by a ",").
- `dist` [`character(1)`]:
  Whether or not the variable is distinct from a cluster. This is the case when the variable is not systematically available for all clusters and thus needs to be registered separately from the clusters.
- `rel` [`logical(1)`]:
  Whether or not the values in `row` and `col` are relative to the cluster positions or whether they refer to the overall table.

Measured variables are described by the properties:

- `type` [`character(1)`]:
  the value `"measured"` signals that this is a measured variable.
- `unit` [`character(1)`]:
  The unit in which the values shall be recorded in the database.
- `factor` [`character(1)`]:
  A factor to transform the values to `unit`. For instance, if values are recorded in acres, but shall be contained in the database in hectare, the factor would be 0.40468.
- `row` [`integerish(1)`]:
  The row(s) in which the variable values are recorded.
- `col` [`integerish(1)`]:
  The column(s) in which the variable values are recorded.
- `rel` [`logical(1)`]:
  Whether or not the values in `row` and `col` are relative to the cluster positions or whether they refer to the overall table.
- `key` [`character(1)`]:
  If the variable is recorded, together with other variables, so that the variable names are listed in one column and the respective values are listed in another column, give here the name of the column that contains the variable names.

- value [character(1)]:
  If the variable is recorded, together with other variables, so that the variable names are listed in one column and the respective values are listed in another column, give here the level in the names column that stands for the values of this variable.
- dist [character(1)]:
  Whether or not the variable is distinct from a cluster. This is the case when the variable is not systematically available for all clusters and thus needs to be registered separately from the clusters.

Examples

```
# define outline of the cluster(s)
theClusters <- list(row = c(1, 8, 8),
                    col = c(1, 1, 4),
                    width = 3,
                    height = 6,
                    id = "territories")

# identify the row(s) where the header is
theHeader <- list(row = 1, rel = TRUE)

# document identifying variables
idVars <- list(
  territories =
    list(type = "id", row = 1, col = 1, rel = TRUE),
  year =
    list(type = "id", row = c(3:6), col = 4, dist = TRUE),
  commodities =
    list(type = "id", col = 1, rel = TRUE))

# document measured variables
measuredVars <- list(
  harvested =
    list(type = "measured", unit = "ha", factor = 1,
         col = 2, rel = TRUE),
  production =
    list(type = "measured", unit = "t", factor = 1,
         col = 3, rel = TRUE))

# make the schema
mySchema <- list(clusters = theClusters,
                 header = theHeader,
                 variables = c(idVars, measuredVars))

makeSchema(schema = mySchema)
```

---

reorganise *Reorganise a table*

Description

This function takes a disorganised messy table and rearranges columns and rows into a tidy dataset that can be sorted into the areal database.

Usage

```
reorganise(input = NULL, schema = NULL)
```

Arguments

input	[data.frame(1)] table to reorganise.
schema	[symbol(1)] the schema description for reorganising input.

Value

A (tidy) table which is the result of employing schema on input.

Examples

```
# read in a disorganised messy dataset
library(readr)
ds <- system.file("test_datasets", package = "tabshiftr")
input <- read_csv(file = paste0(ds, "/table13.csv"),
                  col_names = FALSE, col_types = cols(.default = "c"))
input

# put together schema description (see makeSchema function)
mySchema <- makeSchema(schema = list(
  clusters =
    list(row = c(1, 8, 8), col = c(1, 1, 4), width = 3, height = 6,
         id = "territories"),
  header = list(row = 1, rel = TRUE),
  variables =
    list(territories =
           list(type = "id", row = 1, col = 1, rel = TRUE),
         year =
           list(type = "id", row = c(3:6), col = 4, dist = TRUE),
         commodities =
           list(type = "id", col = 1, rel = TRUE),
         harvested =
           list(type = "measured", unit = "ha", factor = 1,
                col = 2, rel = TRUE),
         production =
           list(type = "measured", unit = "t", factor = 1,
                col = 3, rel = TRUE))
))

# get the tidy output
```

```
reorganise(input, mySchema)
```

---

schema-class *The* schema *class (S4) and its methods*

Description

A schema stores the information of where which information is stored in a table of data.

Details

The slot variables typically contains several lists that each record the metadata of the respective variables.

Slots

cluster [list(1)]
: description of clusters of data.

header [codelist(1)]
: description of the header.

meta [list(1)]
: description of the metadata.

variables [named list(.)]
: description of variables.

---

schema_default *Default template of a schema description*

Description

Default template of a schema description

Usage

```
schema_default
```

Format

The object of class schema describes at which position in a table which information can be found. It contains the four slots clusters, header, meta and variables.

The default schema description contains an example of an identifying and a values variables. Further identifying and values variables would be added when more variables are contained in a table.

selectData *Select data chunks from a spreadsheet*

Description

This function builds a list of those chunks that contain the data.

Usage

```
selectData(input = NULL, clusters = NULL, header = NULL)
```

Arguments

input	the raw table from which to select clusters
clusters	the "clusters" slot of a schema description.
header	the "header" slot of a schema description.

Value

A list of the rows and columns in the spreadsheet that contain data plus the header and the data, per cluster.

show,schema-method *Print the* schema

Description

Print the schema

Usage

```
## S4 method for signature 'schema'
show(object)
```

Arguments

object	[schema] the schema to print.

`updateSchema` *Check schema description for consistency*

Description

This function takes an input table and a respective input schema and ensures that the cluster specification is complete and that column and row values are consistent.

Usage

```
updateSchema(input = NULL, schema = NULL)
```

Arguments

`input`	an input for which to check a schema description.
`schema`	the schema description.

Value

An updated schema description that fulfills formal requirements to be processed by `reorganise`.

Index