Package ‘matsindf’

March 22, 2020

Type Package

Title Matrices in Data Frames

Version 0.3.4

Date 2020-03-21

Maintainer Matthew Heun <matthew.heun@me.com>

Description Provides functions to collapse a tidy data frame into matrices in a data frame and expand a data frame of matrices into a tidy data frame.

License MIT + file LICENSE

Language en-US

Encoding UTF-8

LazyData true

RoxygenNote 7.1.0

Depends R (>= 2.10)

Imports dplyr, magrittr, matsbyname, purrr, rlang, rlist, tibble, tidyr

Suggests ggplot2, knitr, rmarkdown, testthat, covr

VignetteBuilder knitr

URL https://github.com/MatthewHeun/matsindf

BugReports https://github.com/MatthewHeun/matsindf/issues

NeedsCompilation no

Author Matthew Heun [aut, cre] (<https://orcid.org/0000-0002-7438-214X>)

Repository CRAN

Date/Publication 2020-03-22 04:00:02 UTC

R topics documented:

`add_UKEnergy2000_matnames` *Add a column of matrix names to tidy data frame*

Description

Add a column of matrix names to tidy data frame

Usage

```
add_UKEnergy2000_matnames(
  .DF,
  ledger_side_colname = "Ledger.side",
  energy_colname = "E.ktoe",
  supply_side = "Supply",
  consumption_side = "Consumption",
  matname_colname = "matname",
  U_name = "U",
  V_name = "V",
  Y_name = "Y"
)
```

Arguments

`.DF` a data frame with `ledger_side_colname` and `energy_colname`.

`ledger_side_colname` the name of the column in `.DF` that contains ledger side (a string). Default is `"Ledger.side"`.

`energy_colname` the name of the column in `.DF` that contains energy values (a string). Default is `"E.ktoe"`.

supply_side the identifier for items on the supply side of the ledger (a string). Default is "Supply".

consumption_side
the identifier for items on the consumption side of the ledger (a string). Default is "Consumption".

matname_colname
the name of the output column containing the name of the matrix in which this row belongs (a string). Default is "UVY".

U_name the name for the use matrix (a string). Default is "U".

V_name the name for the make matrix (a string). Default is "V".

Y_name the name for the final demand matrix (a string). Default is "Y".

Value

.DF with an added column, UVY_colname.

Examples

```
matsindf:::add_UKEnergy2000_matnames(UKEnergy2000)
```

---

add_UKEnergy2000_row_col_meta
Add row, column, row type, and column type metadata

Description

Add row, column, row type, and column type metadata

Usage

```
add_UKEnergy2000_row_col_meta(
  .DF,
  matname_colname = "matname",
  U_name = "U",
  V_name = "V",
  Y_name = "Y",
  product_colname = "Product",
  flow_colname = "Flow",
  industry_type = "Industry",
  product_type = "Product",
  sector_type = "Sector",
  rowname_colname = "rowname",
  colname_colname = "colname",
  rowtype_colname = "rowtype",
  coltype_colname = "coltype"
)
```

Arguments

.DF a data frame containing matname_colname.

matname_colname the name of the column in .DF that contains names of matrices (a string). Default is "matname".

U_name the name for use matrices (a string). Default is "U".

V_name the name for make matrices (a string). Default is "V".

Y_name the name for final demand matrices (a string). Default is "Y".

product_colname the name of the column in .DF where Product names is found (a string). Default is "Product".

flow_colname the name of the column in .DF where Flow information is found (a string). The Flow column usually contains the industries involved in this flow. Default is "Flow".

industry_type the name that identifies production industries and and transformation processes (a string). Default is "Industry".

product_type the name that identifies energy carriers (a string). Default is "Product".

sector_type the name that identifies final demand sectors (a string). Default is "Sector".

rowname_colname the name of the output column that contains row names for matrices (a string). Default is "rowname".

colname_colname the name of the output column that contains column names for matrices (a string). Default is "colname".

rowtype_colname the name of the output column that contains row types for matrices (a string). Default is "rowtype".

coltype_colname the name of the output column that contains column types for matrices (a string). Default is "coltype".

Value

.DF with additional columns named rowname_colname, colname_colname, rowtype_colname, and coltype_colname.

Examples

```
UKEnergy2000 %>%
  matsindf:::add_UKEnergy2000_matnames(.) %>%
  matsindf:::add_UKEnergy2000_row_col_meta(.)
```

collapse_to_matrices *Collapse a "tidy" data frame to matrices in a data frame (matsindf)*

Description

A "tidy" data frame contains information that can be collapsed into matrices, including columns for matrix names, row names, column names, row types, column types, and values (entries in matrices). These column names are specified as strings by the `matnames`, `rownames`, `colnames`, `rowtypes`, `coltypes`, and `values` arguments to collapse_to_matrices, respectively. A **matsindf**-style matrix has named rows and columns. In addition, **matsindf**-style matrices have "types" for row and column information, such as "Commodities", "Industries", "Products", or "Machines". The row and column types for the **matsindf**-style matrices are stored as attributes on the matrix (`rowtype` and `coltype`), which can be accessed with the `rowtype` and `coltype` functions of the **matsbyname** package. Row and column types are both respected and propagated by the various `_byname` functions of the **matsbyname** package. Use the `*_byname` functions when you do operations on the **matsindf**-style matrices. The **matsindf**-style matrices will be stored in a column with same name as the incoming `values` column. This function is similar to `nest`, which stores data frames into a cell of a data frame. With `collapse_to_matrices`, matrices are created. This function is similar to `summarise` in that groups are respected. (In fact, calls to this function may not work properly unless grouping is provided. Errors of the form "Error: Duplicate identifiers for rows ..." are usually fixed by grouping `.DF` prior to calling this function.) The usual approach is to `group_by` the `matnames` column and any other columns to be preserved in the output. Note that execution is halted if any of `rownames`, `rowtypes`, `colnames`, `coltypes`, or `values` is a grouping variable. `rowtypes` and `coltypes` should be the same for all rows of the same matrix; execution is halted if that is not the case. `spread`ing the output by `matnames` may be necessary before calculations are done on the matrices. See the example.

Usage

```
collapse_to_matrices(
  .DF,
  matnames = "matnames",
  matvals = "matvals",
  rownames = "rownames",
  colnames = "colnames",
  rowtypes = "rowtypes",
  coltypes = "coltypes"
)
```

Arguments

`.DF`	the "tidy" data frame
`matnames`	a string identifying the column in `.DF` containing matrix names for matrices to be created. Default is "`matnames`".
`matvals`	a string identifying the column in `.DF` containing values to be inserted into the matrices to be created. This will also be the name of the column in the output

	containing matrices formed from the data in the matvals column. Default is "matvals".
rownames	a string identifying the column in .DF containing row names for matrices to be created. Default is "rownames".
colnames	a string identifying the column in .DF containing column names for matrices to be created. Default is "colnames".
rowtypes	optional string identifying the column in .DF containing the type of values in rows of the matrices to be created. Default is "rowtypes".
coltypes	optional string identifying the column in .DF containing the type of values in columns of the matrices to be created Default is "coltypes".

Details

Groups are not preserved on output.

Value

a data frame with matrices in columns

See Also

nest and summarise.

Examples

```
library(dplyr)
library(tidyr)
library(tibble)
ptype <- "Products"
itype <- "Industries"
tidy <- data.frame(Country = c( "GH",  "GH",  "GH",  "GH",  "GH",  "GH",  "GH",
                                "US",  "US",  "US",  "US", "GH", "US"),
                   Year    = c(  1971,  1971,  1971,  1971,  1971,  1971,  1971,
                                 1980,  1980,  1980,  1980, 1971, 1980),
                   matrix  = c(   "U",   "U",   "E",   "E",   "E",   "V",   "V",
                                  "U",   "U",   "E",   "E", "eta", "eta"),
                   row     = c( "c 1", "c 2", "c 1", "c 2", "c 2", "i 1", "i 2",
                               "c 1", "c 1", "c 1", "c 2", NA, NA),
                   col     = c( "i 1", "i 2", "i 1", "i 2", "i 3", "c 1", "c 2",
                               "i 1", "i 2", "i 1", "i 2", NA, NA),
                   rowtypes = c( ptype, ptype, ptype, ptype, ptype, itype, itype,
                                 ptype, ptype, ptype, ptype, NA, NA),
                   coltypes = c( itype, itype, itype, itype, itype, ptype, ptype,
                                 itype, itype, itype, itype, NA, NA),
                   vals  = c(    11   ,  22,    11 ,    22 ,    23 ,    11 ,    22 ,
                                 11 ,    12 ,    11 ,    22,    0.2, 0.3)
) %>% group_by(Country, Year, matrix)
mats <- collapse_to_matrices(tidy, matnames = "matrix", matvals = "vals",
                             rownames = "row", colnames = "col",
                             rowtypes = "rowtypes", coltypes = "coltypes")
mats %>% spread(key = matrix, value = vals)
```

everything_except *Get symbols for all columns except ...*

Description

This convenience function performs a set difference between the columns of `.DF` and the variable names (or symbols) given in `...`. The return value is a list of symbols.

Usage

```
everything_except(.DF, ..., .symbols = TRUE)
```

Arguments

`.DF`	a data frame whose variable names are to be differenced
`...`	a string, strings, vector of strings, or list of strings representing column names to be subtracted from the names of `.DF`
`.symbols`	a boolean that defines the return type: `TRUE` for symbols, `FALSE` for strings

Value

a vector of symbols (when `symbols = TRUE`) or strings (when `symbol = FALSE`) containing all variables names except those given in `...`

Examples

```
DF <- data.frame(a = c(1, 2), b = c(3, 4), c = c(5, 6))
everything_except(DF, "a", "b")
everything_except(DF, "a", "b", symbols = FALSE)
everything_except(DF, c("a", "b"))
everything_except(DF, list("a", "b"))
```

---

expand_to_tidy *Expand a "tidy" data frame with matsindf-style matrices to a "tidy" data frame with each matrix entry as an observation*

Description

A data frame with **matsindf**-style matrices contains matrices with names `matnames` in the column specified by `matvals`). An IO-style matrix has named rows and columns. In addition, **matsindf**-style matrices have "types" for row and column information, such as "Commodities", "Industries", "Products", or "Machines".

Usage

```
expand_to_tidy(
  .DF,
  matnames = "matnames",
  matvals = "matvals",
  rownames = "rownames",
  colnames = "colnames",
  rowtypes = "rowtypes",
  coltypes = "coltypes",
  drop = NA
)
```

Arguments

`.DF`	the data frame containing **matsindf**-style matrices. (`.DF` may also be a named list of matrices, in which case names of the matrices are taken from the names of items in the list and list items are expected to be matrices.)
`matnames`	name of the column in `.DF` containing matrix names (a string). Default is `"matnames"`.
`matvals`	name of the column in `.DF` containing IO-style matrices or constants (a string), This will also be the name of the column containing matrix entries in the output data frame. Default is `"matvals"`.
`rownames`	name for the output column of row names (a string). Default is `"rownames"`.
`colnames`	name for the output column of column names (a string). Default is `"colnames"`.
`rowtypes`	optional name for the output column of row types (a string). Default is `"rowtypes"`.
`coltypes`	optional name for the output column of column types (a string) Default is `"coltypes"`.
`drop`	if specified, the value to be dropped from output, For example, `drop = 0` will cause `0` entries in the matrices to be deleted from output. If `NA`, no values are dropped from output. Default is `NA`.

Details

Names for output columns are specified in the `rownames`, `colnames`, `rowtypes`, and `coltypes`, arguments. The entries of the **matsindf**-style matrices are stored in an output column named `values`.

Value

a tidy data frame containing expanded **matsindf**-style matrices

Examples

```
library(dplyr)
library(matsbyname)
ptype <- "Products"
itype <- "Industries"
tidy <- data.frame(Country  = c( "GH",  "GH",  "GH",  "GH",  "GH",  "GH",  "GH",
                                 "US",  "US",  "US",  "US", "GH", "US"),
```

```
                Year      = c( 1971,  1971,  1971,  1971,  1971,  1971,  1971,
                                 1980,  1980,  1980,  1980, 1971, 1980),
                matrix    = c(  "U",   "U",   "Y",   "Y",   "Y",   "V",   "V",
                                 "U",   "U",   "Y",   "Y", "eta", "eta"),
                row       = c(  "c1",  "c2",  "c1",  "c2",  "c2",  "i1",  "i2",
                                 "c1",  "c1",  "c1",  "c2", NA, NA),
                col       = c(  "i1",  "i2",  "i1",  "i2",  "i3",  "c1",  "c2",
                                 "i1",  "i2",  "i1",  "i2", NA, NA),
                rowtypes  = c( ptype, ptype, ptype, ptype, ptype, itype, itype,
                                 ptype, ptype, ptype, ptype, NA, NA),
                coltypes  = c(itype, itype, itype, itype, itype, ptype, ptype,
                                itype, itype, itype, itype, NA, NA),
                vals  = c(    11   ,  22,    11 ,    22 ,    23 ,    11 ,    22 ,
                                11 ,    12 ,    11 ,    22,   0.2, 0.3)) %>%
  group_by(Country, Year, matrix)
mats <- collapse_to_matrices(tidy, matnames = "matrix", rownames = "row", colnames = "col",
                             rowtypes = "rowtypes", coltypes = "coltypes",
                             matvals = "vals") %>%
  ungroup()
expand_to_tidy(mats, matnames = "matrix", matvals = "vals",
                     rownames = "rows", colnames = "cols",
                     rowtypes = "rt",   coltypes = "ct")
expand_to_tidy(mats, matnames = "matrix", matvals = "vals",
                     rownames = "rows", colnames = "cols",
                     rowtypes = "rt",   coltypes = "ct", drop = 0)
```

---

group_by_everything_except

Group by all variables except some

Description

This is a convenience function that allows grouping of a data frame by all variables (columns) except those variables specified in '...'.

Usage

```
group_by_everything_except(.DF, ..., .add = FALSE, .drop = FALSE)
```

Arguments

`.DF`	a data frame to be grouped
`...`	a string, strings, vector of strings, or list of strings representing column names to be excluded from grouping
`.add`	When '.add = FALSE', the default, 'dplyr::group_by()' will override existing groups. To add to the existing groups, use '.add = TRUE'.
`.drop`	When '.drop = TRUE', empty groups are dropped.

Value

a grouped version of '.DF'

Examples

```
library(dplyr)
DF <- data.frame(a = c(1, 2), b = c(3, 4), c = c(5, 6))
group_by_everything_except(DF) %>% group_vars()
group_by_everything_except(DF, NULL) %>% group_vars()
group_by_everything_except(DF, c()) %>% group_vars()
group_by_everything_except(DF, list()) %>% group_vars()
group_by_everything_except(DF, c) %>% group_vars()
group_by_everything_except(DF, "a") %>% group_vars()
group_by_everything_except(DF, "c") %>% group_vars()
group_by_everything_except(DF, c("a", "c")) %>% group_vars()
group_by_everything_except(DF, c("a")) %>% group_vars()
group_by_everything_except(DF, list("a")) %>% group_vars()
```

---

index_column *Index a column in a data frame by groups relative to an initial year*

Description

This function indexes (by ratio) variables in `vars_to_index` to the first time in `time_var` or to `index_time` (if specified). Groups in `.DF` are both respected and required. Neither `var_to_index` nor `time_var` can be in the grouping variables.

Usage

```
index_column(
  .DF,
  var_to_index,
  time_var = "Year",
  index_time = NULL,
  indexed_var = paste0(var_to_index, suffix),
  suffix = "_indexed"
)
```

Arguments

`.DF`	the data frame in which the variables are contained
`var_to_index`	the column name representing the variable to be indexed (a string)
`time_var`	the name of the column containing time information. Default is "`Year`".
`index_time`	the time to which data in `var_to_index` are indexed. If `NULL` (the default), `index_time` is set to the first time of each group.
`indexed_var`	the name of the indexed variable. Default is "`<<var_to_index>>_<<suffix>>`".
`suffix`	the suffix to be appended to the indexed variable. Default is "`_indexed`".

Details

Note that this function works when the variable to index is a column of numbers or a column of matrices.

Value

a data frame with same number of rows as .DF and the following columns: grouping variables of .DF, var_to_index, time_var, and one additional column containing indexed var_to_index named with the value of indexed_var.

Examples

```
library(dplyr)
library(tidyr)
DF <- data.frame(Year = c(2000, 2005, 2010), a = c(10, 15, 20), b = c(5, 5.5, 6)) %>%
  gather(key = name, value = var, a, b) %>%
  group_by(name)
index_column(DF, var_to_index = "var", time_var = "Year", suffix = "_ratioed")
index_column(DF, var_to_index = "var", time_var = "Year", indexed_var = "now.indexed")
index_column(DF, var_to_index = "var", time_var = "Year", index_time = 2005,
             indexed_var = "now.indexed")

  DF %>%
    ungroup() %>%
    group_by(name, var) %>%
   index_column(var_to_index = "var", time_var = "Year") # Fails! Do not group on var_to_index.
  DF %>%
    ungroup() %>%
    group_by(name, Year) %>%
   index_column(var_to_index = "var", time_var = "Year") # Fails! Do not group on time_var.
```

---

matsindf_apply *Apply a function to a matsindf data frame (and more)*

Description

Applies FUN to .dat or performs the calculation specified by FUN on numbers or matrices. FUN must return a named list.

Usage

```
matsindf_apply(.dat = NULL, FUN, ...)
```

Arguments

.dat	a list of named items or a data frame
FUN	the function to be applied to .dat
...	named arguments to be passed by name to FUN.

Details

If is.null(.dat) and ... are all named numbers or matrices of the form argname = m, ms are passed to FUN by argnames. The return value is a named list provided by FUN. The arguments in ... are not included in the output.

If is.null(.dat) and ... are all lists of numbers or matrices of the form argname = l, FUN is Mapped across the various ls to obtain a list of named lists returned from FUN. The return value is a data frame whose rows are the top-level lists returned from FUN and whose column names are the names of the list items returned from FUN. Columns of .dat are not included in the return value.

If !is.null(.dat) and ... are all named character strings of the form argname = string, argnames are expected to be names of arguments to FUN, and strings are expected to be column names in .dat. The return value is .dat with additional columns (at right) whose names are the names of list items returned from FUN. When .dat contains columns whose names are same as columns added at the right, a warning is emitted.

.dat can be a list of named items in which case a list will be returned.

If items in .dat have same names are arguments to FUN, it is not necessary to specify any arguments in .... matsindf_apply assumes that the appropriately-named items in .dat are intended to be arguments to FUN. When an item name appears in both ... and .dat, ... takes precedence.

NULL arguments in ... are ignored for the purposes of deciding whether all arguments are numbers, matrices, lists of numbers of matrices, or named character strings. However, all NULL arguments are passed to FUN, so FUN should be able to deal with NULL arguments appropriately.

If .dat is present, ... contains strings, and one of the ... strings is not the name of a column in .dat, FUN is called WITHOUT the argument whose column is missing. I.e., that argument is treated as missing. If FUN works despite the missing argument, execution proceeds. If FUN cannot handle the missing argument, an error will occur in FUN.

Value

a named list or a data frame. (See details.)

Examples

```
library(matsbyname)
example_fun <- function(a, b){
  return(list(c = sum_byname(a, b), d = difference_byname(a, b)))
}
# Single values for arguments
matsindf_apply(FUN = example_fun, a = 2, b = 2)
# Matrices for arguments
a <- 2 * matrix(c(1,2,3,4), nrow = 2, ncol = 2, byrow = TRUE,
              dimnames = list(c("r1", "r2"), c("c1", "c2")))
b <- 0.5 * a
matsindf_apply(FUN = example_fun, a = a, b = b)
# Single values in lists are treated like columns of a data frame
matsindf_apply(FUN = example_fun, a = list(2, 2), b = list(1, 2))
# Matrices in lists are treated like columns of a data frame
matsindf_apply(FUN = example_fun, a = list(a, a), b = list(b, b))
# Single numbers in a data frame
DF <- data.frame(a = c(4, 4, 5), b = c(4, 4, 4))
```

```
matsindf_apply(DF, FUN = example_fun, a = "a", b = "b")
# By default, arguments to FUN come from DF
matsindf_apply(DF, FUN = example_fun)
# Now put some matrices in a data frame.
DF2 <- data.frame(a = I(list(a, a)), b = I(list(b,b)))
matsindf_apply(DF2, FUN = example_fun, a = "a", b = "b")
# All arguments to FUN are supplied by named items in .dat
matsindf_apply(list(a = 1, b = 2), FUN = example_fun)
# All arguments are supplied by named arguments in ..., but mix them up.
# Note that the named arguments override the items in .dat
matsindf_apply(list(a = 1, b = 2, z = 10), FUN = example_fun, a = "z", b = "b")
# Warning is issued when an output item has same name as an input item.
matsindf_apply(list(a = 1, b = 2, c = 10), FUN = example_fun, a = "c", b = "b")
```

`matsindf_apply_types` *Determine types of ... argument for matsindf_apply*

Description

This is a convenience function that returns a logical list for the types of `...` with components named `dots_present`, `all_dots_num`, `all_dots_mats`, `all_dots_list`, `all_dots_vect`, and `all_dots_char`.

Usage

```
matsindf_apply_types(...)
```

Arguments

`...`	the list of arguments to be checked

Details

When arguments are present in `...`, `dots_present` is `TRUE` but `FALSE` otherwise. When all items in `...` are single numbers, `all_dots_num` is `TRUE` and all other list members are `FALSE`. When all items in `...` are matrices, `all_dots_mats` is `TRUE` and all other list members are `FALSE`. When all items in `...` are lists, `all_dots_list` is `TRUE` and all other list members are `FALSE`. When all items in `...` are vectors (including lists), `all_dots_vect` is `TRUE`. When all items in `...` are character strings, `all_dots_char` is `TRUE` and all other list members are `FALSE`.

Value

A logical list with components named `dots_present`, `all_dot_num`, `all_dots_mats`, `all_dots_list`, and `all_dots_char`.

Examples

```
matsindf_apply_types(a = 1, b = 2)
matsindf_apply_types(a = matrix(c(1, 2)), b = matrix(c(2, 3)))
matsindf_apply_types(a = c(1, 2), b = c(3, 4), c = c(5, 6))
matsindf_apply_types(a = list(1, 2), b = list(3, 4), c = list(5, 6))
matsindf_apply_types(a = "a", b = "b", c = "c")
```

---

mat_to_rowcolval   *Convert a matrix to a data frame with rows, columns, and values.*

Description

This function "expands" a matrix into a tidy data frame with a values column and factors for row names, column names, row types, and column types. Optionally, values can be dropped.

Usage

```
mat_to_rowcolval(
  .matrix,
  matvals = "matvals",
  rownames = "rownames",
  colnames = "colnames",
  rowtypes = "rowtypes",
  coltypes = "coltypes",
  drop = NA
)
```

Arguments

`.matrix`	the IO-style matrix to be converted to a data frame with rows, columns, and values
`matvals`	a string for the name of the output column containing values. Default is "`matvals`".
`rownames`	a string for the name of the output column containing row names. Default is "`rownames`".
`colnames`	a string for the name of the output column containing column names. Default is "`colnames`".
`rowtypes`	a string for the name of the output column containing row types. Default is "`rowtypes`".
`coltypes`	a string for the name of the output column containing column types. Default is "`coltypes`".
`drop`	if specified, the value to be dropped from output. Default is `NA`. For example, `drop = 0` will cause `0` entries in the matrices to be deleted from output. If `NA`, no values are dropped from output.

Value

a data frame with rows, columns, and values

Examples

```
library(matsbyname)
data <- data.frame(Country  = c("GH", "GH", "GH"),
                   rows = c( "c1",  "c1", "c2"),
                   cols = c( "i1",  "i2", "i2"),
                   rt = c("Commodities", "Commodities", "Commodities"),
                   ct = c("Industries", "Industries", "Industries"),
                   vals = c(  11   ,  12,   22 ))
data
A <- data %>%
  rowcolval_to_mat(rownames = "rows", colnames = "cols",
                   rowtypes = "rt",   coltypes = "ct", matvals = "vals")
A
mat_to_rowcolval(A, rownames = "rows", colnames = "cols",
                 rowtypes = "rt", coltypes = "ct", matvals = "vals")
mat_to_rowcolval(A, rownames = "rows", colnames = "cols",
                 rowtypes = "rt", coltypes = "ct", matvals = "vals", drop = 0)
# This also works for single values
mat_to_rowcolval(2, matvals = "vals",
                 rownames = "rows", colnames = "cols",
                 rowtypes = "rt", coltypes = "ct")
mat_to_rowcolval(0, matvals = "vals",
                 rownames = "rows", colnames = "cols",
                 rowtypes = "rt", coltypes = "ct", drop = 0)
```

`rowcolval_to_mat` *Collapse a tidy data frame into a matrix with named rows and columns*

Description

Columns not specified in one of `rownames`, `colnames`, `rowtype`, `coltype`, or `values` are silently dropped. `rowtypes` and `coltypes` are added as attributes to the resulting matrix (via `setrowtype` and `setcoltype`). The resulting matrix is a (under the hood) a data frame. If both `rownames` and `colnames` columns of `.DF` contain `NA`, it is assumed that this is a single value, not a matrix, in which case the value in the `values` column is returned.

Usage

```
rowcolval_to_mat(
  .DF,
  matvals = "matvals",
  rownames = "rownames",
  colnames = "colnames",
  rowtypes = "rowtypes",
  coltypes = "coltypes",
```

```
  fill = 0
)
```

Arguments

.DF	a tidy data frame containing columns for row names, column names, and values
matvals	the name of the column in .DF containing values with which to fill the matrix (a string). Default is "matvals".
rownames	the name of the column in .DF containing row names (a string). Default is "rownames".
colnames	the name of the column in .DF containing column names (a string). Default is "colnames".
rowtypes	an optional string identifying the types of information found in rows of the matrix to be constructed. Default is "rowtypes".
coltypes	an optional string identifying the types of information found in columns of the matrix to be constructed. Default is "coltypes".
fill	the value for missing entries in the resulting matrix. default is 0.

Value

a matrix with named rows and columns and, optionally, row and column types

Examples

```
library(matsbyname)
library(dplyr)
data <- data.frame(Country  = c("GH", "GH", "GH"),
                   rows = c( "c 1",  "c 1", "c 2"),
                   cols = c( "i 1",  "i 2", "i 2"),
                   vals = c(   11  ,   12,    22 ))
A <- rowcolval_to_mat(data, rownames = "rows", colnames = "cols", matvals = "vals")
A
rowtype(A) # NULL, because types not set
coltype(A) # NULL, because types not set
B <- rowcolval_to_mat(data, rownames = "rows", colnames = "cols", matvals = "vals",
                            rowtypes  = "Commodities", coltypes  = "Industries")
B
C <- data %>% bind_cols(data.frame(rt = c("Commodities", "Commodities", "Commodities"),
                                  ct = c("Industries", "Industries", "Industries"))) %>%
  rowcolval_to_mat(rownames = "rows", colnames = "cols", matvals = "vals",
                   rowtypes = "rt", coltypes = "ct")
C
# Also works for single values if both the rownames and colnames columns contain NA
data2 <- data.frame(Country = c("GH"), rows = c(NA), cols = c(NA),
  rowtypes = c(NA), coltypes = c(NA), vals = c(2))
data2 %>% rowcolval_to_mat(rownames = "rows", colnames = "cols", matvals = "vals",
  rowtypes = "rowtypes", coltypes = "coltypes")
data3 <- data.frame(Country = c("GH"), rows = c(NA), cols = c(NA), vals = c(2))
data3 %>% rowcolval_to_mat(rownames = "rows", colnames = "cols", matvals = "vals")
```

```
# Fails when rowtypes or coltypes not all same. In data3, column rt is not all same.
data4 <- data %>% bind_cols(data.frame(rt = c("Commodities", "Industries", "Commodities"),
                                        ct = c("Industries", "Industries", "Industries")))
rowcolval_to_mat(data4, rownames = "rows", colnames = "cols",
                            matvals = "vals", rowtypes = "rt", coltypes = "ct")
```

UKEnergy2000 *Energy consumption in the UK in 2000*

Description

A dataset containing approximations to some of the energy flows in the UK in the year 2000. These data first appeared as the example in Figures 3 and 4 of M.K. Heun, A. Owen, and P.E. Brockway. A physical supply-use table framework for energy analysis on the energy conversion chain. Sustainability Research Institute Paper 111, University of Leeds, School of Earth and Environment, Sustainability Research Institute, Leeds, England, 13 November 2017.

Usage

```
UKEnergy2000
```

Format

A data frame with 36 rows and 7 variables:

Country country, GB (Great Britain, only one country)

Year year, 2000 (only one year)

Ledger.side Supply or Consumption

Flow.aggregation.point tells where each row should be aggregated

Flow the Industry or Sector involved in this flow

Product the energy product involved in this flow

E.ktoe magnitude of the energy flow in ktoe

Source

http://www.see.leeds.ac.uk/fileadmin/Documents/research/sri/workingpapers/sri-wp111.pdf

verify_cols_missing *Verify that column names in a data frame are not already present*

Description

In the Recca package, many functions add columns to an existing data frame. If the incoming data frame already contains columns with the names of new columns to be added, a name collision could occur, deleting the existing column of data. This function provides a way to quickly check whether newcols are already present in .DF.

Usage

```
verify_cols_missing(.DF, newcols)
```

Arguments

.DF	the data frame to which newcols are to be added
newcols	a single string, a single name, a vector of strings representing the names of new columns to be added to .DF, or a vector of names of new columns to be added to .DF

Details

This function terminates execution if a column of .DF will be overwritten by one of the newcols.

Value

NULL. This function should be called for its side effect of checking the validity of the names of newcols to be added to .DF.

Examples

```
df <- data.frame(a = c(1,2), b = c(3,4))
verify_cols_missing(df, "d") # Silent. There will be no problem adding column "d".
newcols <- c("c", "d", "a", "b")
verify_cols_missing(df, newcols) # Error: a and b are already in df.
```

Index