Package 'manydata'

May 8, 2026

Title Many Global Governance Datacubes

Version 1.1.3

Date 2025-09-30

Description This is the core package offering a portal to the many packages universe. It includes functions to help researchers access, work across, and maintain ensembles of datasets on global governance called datacubes.

License CC BY 4.0

URL https://www.manydata.ch/

BugReports https://github.com/globalgov/manydata/issues

Depends R (>= 3.5.0), cli, dplyr, messydates (>= 0.5.0)

Imports caret, dtplyr, ggplot2 (>= 3.4.0), glmnet, httr, jsonlite, purrr, remotes, stringr, text2vec, tidyr

Suggests testthat, readr, knitr, rmarkdown, ggVennDiagram, rlang

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

Config/Needs/build roxygen2, devtools

Config/Needs/check covr, lintr, spelling

Config/Needs/website pkgdown

Config/testthat/parallel true

Config/testthat/edition 3

Config/testthat/start-first compare

NeedsCompilation no

Author James Hollway [cre, aut, ctb] (IHEID, ORCID: <https://orcid.org/0000-0002-8361-9647>), Henrique Sposito [aut, ctb] (IHEID, ORCID: <https://orcid.org/0000-0003-3420-6085>), Bernhard Bieri [ctb] (IHEID, ORCID: <https://orcid.org/0000-0001-5943-9059>),

Esther Peev [ctb] (IHEID, ORCID:
<https://orcid.org/0000-0002-9678-2777>),
Jael Tan [ctb] (IHEID, ORCID: <https://orcid.org/0000-0002-6234-9764>)

Maintainer James Hollway <james.hollway@graduateinstitute.ch>

Repository CRAN

Date/Publication 2025-09-30 12:50:02 UTC

Contents

`call_packages` *Call, download, and update many* packages*

Description

`call_packages()` finds and download other packages that belong to the many universe of packages. It allows users to rapidly access the names and other descriptive information of these packages. If users intend to download and install a package listed, they can type the package name within the function.

Usage

```
call_packages(package, develop = FALSE)
```

Arguments

package	A character vector of package name. For multiple packages, please declare package names as a vector (e.g. c("package1", "package2")).
develop	Would you like to download the develop version of the package? FALSE by default.

Value

`call_packages()` returns a tibble with the 'many packages' currently available. If one or more package names are provided, these will be installed from Github.

See Also

Other call_: `call_releases()`, `call_treaties()`

Examples

```
#call_packages()
#call_packages("manyenviron")
```

---

call_releases *Call releases historical milestones/releases*

Description

The function will take a data frame that details this information, or more usefully, a Github repository listing.

Usage

```
call_releases(repo, begin = NULL, end = NULL)
```

Arguments

repo	the github repository to track, e.g. "globalgov/manydata"
begin	When to begin tracking repository milestones. By default NULL, two months before the first release.
end	When to end tracking repository milestones. By default NULL, two months after the latest release.

Details

The function creates a project timeline graphic using ggplot2 with historical milestones and milestone statuses gathered from a specified GitHub repository.

Value

A ggplot graph object

Source

https://benalexkeen.com/creating-a-timeline-graphic-using-r-and-ggplot2/

See Also

Other call_: `call_packages()`, `call_treaties()`

Examples

```
#call_releases("globalgov/manydata")
#call_releases("manypkgs")
```

---

call_sources *Call sources and citations*

Description

These functions call any source or citation information that is available for a datacube or dataset. The function can be used on its own to the console, called during another function call such as `consolidate()` or `pluck()`, or is used to automatically and consistently populate help files.

Usage

```
call_sources(x)

call_citations(x, output = c("console", "help"))
```

Arguments

`x`	A datacube or dataset
`output`	Whether the output should be formatted for "console" or the "help" page.

`call_treaties` *Call treaties from 'many' datasets*

Description

Call treaties from 'many' datasets

Usage

```
call_treaties(
  dataset,
  treaty_type = NULL,
  variable = NULL,
  actor = NULL,
  key = "manyID"
)
```

Arguments

`dataset`	A dataset in a datacube from one of the many packages. NULL by default. That is, all datasets in the datacube are used. For multiple datasets, please declare datasets as a vector (e.g. c("dataset1", "dataset2")).
`treaty_type`	The type of treaties to be returned. NULL, by default. Other options are "bilateral" or "multilateral".
`variable`	Would you like to get one, or more, specific variables present in one or more datasets in the 'many' datacube? NULL by default. For multiple variables, please declare variable names as a vector.
`actor`	An actor variable in dataset. NULL by default. If declared, a tibble of the treaties and their member actors is returned.
`key`	A variable key to join datasets. 'manyID' by default.

Details

Certain datasets, or consolidated datacubes, in 'many' packages contains information on treaties which can be retrieved with `call_treaties()`.

Value

`call_treaties()` returns a tibble with a list of the agreements.

See Also

Other call_: `call_packages()`, `call_releases()`

Examples

```
membs <- dplyr::tibble(manyID = c("ROU-RUS[RFP]_1901A",
"ROU-RUS[RFP]_1901A", "GD16FI_1901A"),
stateID = c("ROU", "RUS", "DNK"),
Title = c("Convention Between Roumania And Russia Concerning Fishing
In The Danube And The Pruth",
"Convention Between Roumania And Russia Concerning Fishing
In The Danube And The Pruth",
"Convention Between The Governments Of Denmark And
The United Kingdom Of Great Britain
And Northern Ireland For Regulating The Fisheries
Of Their Respective Subjects Outside
Territorial Waters In The Ocean Surrounding The Faroe Islands"),
Begin = c("1901-02-22", "1901-02-22", "1901-06-24"))
call_treaties(membs)
call_treaties(membs, treaty_type = "bilateral",
variable = c("Title", "Begin"))
call_treaties(membs, variable = c("Title", "Begin"), actor = "stateID")
```

`code_extend` *Extending codes*

Description

These functions use text embeddings and multinomial logistic regression to suggest missing codes or flag potentially incorrect codes based on text data. Two approaches are provided: one using GloVe embeddings trained on the input text, and another using pre-trained BERT embeddings via the `{text}` package. Both functions require a vector of text (e.g., titles or descriptions) and a corresponding vector of categorical codes, with `NA` or empty strings indicating missing codes to be inferred. The functions train a multinomial logistic regression model using `glmnet` on the text embeddings of the entries with known codes, and then predict codes for the entries with missing codes. The functions also validate the model's performance on a holdout set and report per-class precision, recall, and F1-score. If no missing codes are present, the functions instead check existing codes for potential mismatches and report them.

Usage

```
code_extend_glove(titles, var, req_f1 = 0.8, rarity_threshold = 8)

code_extend_bert(titles, var, req_f1 = 0.8, rarity_threshold = 8, emb_texts)
```

Arguments

`titles`	A character vector of text entries (e.g., titles or descriptions).
`var`	A character vector of (categorical) codes that might be coded from the titles or texts. Entries with missing codes should be `NA_character_` or empty strings. The function will suggest codes for these entries. If no missing codes are present, the function will check existing codes for potential mismatches.

req_f1 The required macro-F1 score on the validation set before proceeding with inference. Default is 0.80.

rarity_threshold Minimum number of occurrences for a code to be included in training. Codes with fewer occurrences are excluded from training to ensure sufficient data for learning. Default is 8.

emb_texts For `code_extend_bert()`, pre-computed embeddings from `text::textEmbed()`. This avoids re-computing embeddings if they have already been computed. A Hugging Face model can be specified via the `model` argument. Default is "sentence-transformers/all-MiniLM-L6-v2". Other models can be used, but they should produce sentence-level embeddings.

Examples

```
titles <- paste(emperors$Wikipedia$CityBirth,
                emperors$Wikipedia$ProvinceBirth,
                emperors$Wikipedia$Rise,
                emperors$Wikipedia$Dynasty,
                emperors$Wikipedia$Cause)
var <- emperors$Wikipedia$Killer
var[var=="Unknown"] <- NA
var[var %in% c("Senate","Court Officials","Opposing Army")] <- "Enemies"
var[var %in% c("Fire","Lightning","Aneurism","Heart Failure")] <- "God"
var[var %in% c("Wife","Usurper","Praetorian Guard","Own Army")] <- "Friends"
glo <- code_extend_glove(titles,
           var)
```

`compare_categories` *Compare categories in 'many' datacubes*

Description

Compare categories in 'many' datacubes

Usage

```
compare_categories(
  datacube,
  dataset = "all",
  key = "manyID",
  variable = "all",
  category = "all"
)
```

Arguments

datacube	A datacube from one of the many packages.
dataset	A dataset in a datacube from one of the many packages. By default "all". That is, all datasets in the datacube are used. To select two or more datasets, please declare them as a vector.
key	A variable key to join datasets. 'manyID' by default.
variable	Would you like to focus on one, or more, specific variables present in one or more datasets in the 'many' datacube? By default "all". For multiple variables, please declare variable names as a vector.
category	Would you like to focus on one specific code category? By default "all" are returned. Other options include "confirmed", "unique", "missing", "conflict", or "majority". For multiple variables, please declare categories as a vector.

Details

Confirmed values are the same in all datasets in datacube. Unique values appear once in datasets in datacube. Missing values are missing in all datasets in datacube. Conflict values are different in the same number of datasets in datacube. Majority values have the same value in multiple, but not all, datasets in datacube.

See Also

Other compare_: compare_dimensions(), compare_missing(), compare_overlap()

Examples

```
compare_categories(emperors, key = "ID")
compare_categories(datacube = emperors, dataset = c("wikipedia", "UNRV"),
key = "ID", variable = c("Beg", "End"), category = c("conflict", "unique"))
plot(compare_categories(emperors, key = "ID"))
plot(compare_categories(datacube = emperors, dataset = c("wikipedia", "UNRV"),
key = "ID", variable = c("Beg", "End"), category = c("conflict", "unique")))
```

---

compare_diff *Compare two datasets for differences*

Description

Compare two datasets for differences

Usage

```
compare_new(.data1, .data2, by = "ID")

compare_diff(
  .data1,
  .data2,
  by = "ID",
  exclude = c("Title", "Coder", "Comments"),
  diff_threshold = 0
)
```

Arguments

.data1	First dataset to compare
.data2	Second dataset to compare
by	Column name to join on (default is "ID")
exclude	Character vector of column names to exclude from comparison. By default, "Title", "Coder", and "Comments" are excluded.
diff_threshold	Integer specifying the minimum number of differing columns for a row to be included in the output. Default is 0, meaning any difference will be included. Set to 3 to only show rows with at least 3 differing columns.

Details

This function uses dplyr::anti_join to find rows in .data1 that are not present in .data2. If no differences are found, a message is printed and NULL is returned. If differences are found, they are returned as a data frame.

Value

A data frame with the differences found

Examples

```
## Not run:
df1 <- data.frame(ID = 1:5, Value = letters[1:5])
df2 <- data.frame(ID = 3:7, Value = letters[3:7])
compare_new(df1, df2)
compare_new(df1, df1)

## End(Not run)
compare_diff(emperors$Wikipedia, emperors$Britannica)
```

`compare_dimensions` *Compare dimensions for 'many' data*

Description

Compare dimensions for 'many' data

Usage

```
compare_dimensions(datacube, dataset = "all")
```

Arguments

`datacube`	A datacube from one of the many packages.
`dataset`	A dataset in a datacube from one of the many packages. By default, "all". That is, all datasets in the datacube are used. To select two or more datasets, please declare them as a vector.

Details

`compare_dimensions()` compares the number of observations, variables, the earliest date, and the latest date in all observations for datasets in a 'many' datacube.

Value

`compare_dimensions()` returns a tibble with information about each dataset including the number of observations, the number of variables, the earliest date, and the latest date in all observations.

See Also

Other compare_: `compare_categories()`, `compare_missing()`, `compare_overlap()`

Examples

```
compare_dimensions(emperors)
```

compare_missing *Compare missing observations for 'many' data*

Description

Compare missing observations for 'many' data

Usage

```
compare_missing(datacube, dataset = "all", variable = "all")
```

Arguments

datacube	A datacube from one of the many packages.
dataset	A dataset in a datacube from one of the many packages. NULL by default. That is, all datasets in the datacube are used. To select two or more datasets, please declare them as a vector.
variable	Would you like to focus on one, or more, specific variables present in one or more datasets in the 'many' datacube? By default "all". For multiple variables, please declare variable names as a vector.

Details

`compare_missing()` compares the missing observations for variables in each dataset in a 'many' datacube.

Value

`compare_missing()` returns a tibble with information about each dataset including the number of observations, the number of variables, the earliest date, and the latest date in all observations.

See Also

Other compare_: `compare_categories()`, `compare_dimensions()`, `compare_overlap()`

Examples

```
compare_missing(emperors)
plot(compare_missing(emperors))
```

compare_overlap *Compare the overlap between datasets in 'many' datacubes*

Description

Compare the overlap between datasets in 'many' datacubes

Usage

```
compare_overlap(datacube, dataset = "all", key = NULL)
```

Arguments

datacube	A datacube from one of the many packages.
dataset	A dataset in a datacube from one of the many packages. By default "all". That is, all datasets in the datacube are used.
key	A variable key to join datasets. 'manyID' by default.

Details

`compare_overlap()` compares the overlap between "key" observations in each dataset in a 'many' datacube.

Value

`compare_overlap()` returns a tibble with information about each dataset and the number of overlapping observations.

See Also

Other compare_: `compare_categories()`, `compare_dimensions()`, `compare_missing()`

Examples

```
compare_overlap(emperors, key = "ID")
plot(compare_overlap(emperors, key = "ID"))
```

`consolidate` *Consolidate datacube into a single dataset*

Description

This function consolidates a set of datasets in a 'many*' package datacube into a single dataset with some combination of the rows, columns, and observations of the datasets in the datacube.

Usage

```
consolidate(
  datacube,
  join = c("full", "inner", "left"),
  resolve = "coalesce",
  key = NULL
)
```

Arguments

`datacube`	A datacube from one of the many packages
`join`	Which join procedure to use. By default "full" so that all observations are retained, but other options include "left" for basing the consolidated dataset on observations present in the first dataset (reorder the datasets to favour another dataset), and "inner" for a consolidated dataset that includes only observations that are present in all datasets.
`resolve`	Choice how (potentially conflicting) values from shared variables should be resolved. Options include: • "coalesce" (default): uses first non-NA value (if available) for each observation, essentially favouring the order the datasets are in in the datacube. • "unite": combines the unique values for each observation across datasets as a set (separated by commas and surrounded by braces), which can be useful for retaining information. • "random": selects values at random from among the observations from each dataset that observed that variable, of particular use for exploring the implications of dataset-related variation. • "precise": selects the value that has the highest precision from among the observations from each dataset (see `resolving_precision()`), which favours more precise data. • "min", "max": these options return the minimum or maximum values respectively, which can be useful for conservative temporal fixing. To resolve variables by different functions, pass the argument a vector (e.g. `resolve = c(var1 = "min", var2 = "max")`). Unnamed variables will be resolved according to the default ("coalesce").

key — An ID column to collapse by. By default "manyID". Users can also specify multiple key variables in a list. For multiple key variables, the key variables must be present in all the datasets in the datacube (e.g. `key = c("key1", "key2")`). For equivalent key columns with different names across datasets, matching is possible if keys are declared (e.g. `key = c("key1" = "key2")`). Missing observations in the key variable are removed.

Details

The function includes separate arguments for the rows and columns, as well as for how to resolve conflicts for observations across datasets. This provides users with considerable flexibility in how they combine data. For example, users may wish to stick to units that appear in every dataset but include variables coded in any dataset, or units that appear in any dataset but only those variables that appear in every dataset. Even then there may be conflicts, as the actual unit-variable observations may differ from dataset to dataset. We offer a number of resolve methods that enable users to choose how conflicts between observations are resolved.

Text variables are dropped for more efficient consolidation.

Value

A single tibble/data frame.

Examples

```
consolidate(emperors, join = "full", resolve = "coalesce", key = "ID")
consolidate(emperors, join = "inner", resolve = "min", key = "ID")
consolidate(emperors, join = "left", resolve = "max", key = "ID")
```

---

describe — *Data reports for datacubes and datasets with 'mdate' variables*

Description

These functions provide meta level descriptions of datacubes or datasets. `mreport()` creates a properly formatted data report for datasets which contain 'mdate' class objects, alongside other object classes. `describe_datacube()` prints a text description of the datasets in a datacube.

Usage

```
mreport(data)

describe_datacube(datacube)
```

Arguments

data — A {tibble} or a {data.frame}.

datacube — A datacube

Details

'mreport' displays the variable's name, the variable type, the number of observations per variable, the number of missing observations for variable, and the percentage of missing observations in variable.

Value

A data report of class 'mreport'.

Examples

```
mreport(emperors)
```

`emperors` *Emperors datacube documentation*

Description

The `emperors` datacube is a list containing 3 datasets: Wikipedia, UNRV, and Britannica

Usage

```
emperors
```

Format

Wikipedia: A dataset with 68 observations and the following 15 variables: ID, Begin, End, FullName, Birth, Death, CityBirth, ProvinceBirth, Rise, Cause, Killer, Dynasty, Era, Notes, Verif.

UNRV: A dataset with 99 observations and the following 7 variables: ID, Begin, End, Birth, Death, FullName, Dynasty.

Britannica: A dataset with 87 observations and the following 3 variables: ID, Begin, End.

Details

```
#> $Wikipedia
#> ----------------------------------------------------------
#> |  Variable    |  Class  |  Obs  |  Missing  |  Miss %  |
#> ----------------------------------------------------------
#> |ID            |character|     69|          0|         0|
#> |Begin         |mdate    |     69|          0|         0|
#> |End           |mdate    |     69|          0|         0|
#> |FullName      |character|     68|          1|      1.45|
#> |Birth         |mdate    |     63|          6|       8.7|
#> |Death         |mdate    |     68|          1|      1.45|
#> |CityBirth     |character|     51|         18|     26.09|
#> |ProvinceBirth|character|     68|          1|      1.45|
#> |Rise          |character|     68|          1|      1.45|
```

```
#> |Cause        |character|     68|          1|      1.45|
#> |Killer       |character|     68|          1|      1.45|
#> |Dynasty      |character|     68|          1|      1.45|
#> |Era          |character|     68|          1|      1.45|
#> |Notes        |character|     46|         23|     33.33|
#> ----------------------------------------------------------
#>
#>
#> $UNRV
#> ----------------------------------------------------------
#> |  Variable  |  Class  |  Obs  |  Missing  |  Miss %  |
#> ----------------------------------------------------------
#> |ID          |character|     98|          0|         0|
#> |Begin       |mdate    |     98|          0|         0|
#> |End         |mdate    |     98|          0|         0|
#> |Birth       |mdate    |     74|         24|     24.49|
#> |Death       |mdate    |     98|          0|         0|
#> |FullName    |character|     93|          5|       5.1|
#> |Dynasty     |character|     61|         37|     37.76|
#> ----------------------------------------------------------
#>
#>
#> $Britannica
#> ----------------------------------------------------------
#> |  Variable  |  Class  |  Obs  |  Missing  |  Miss %  |
#> ----------------------------------------------------------
#> |ID          |character|     87|          0|         0|
#> |Begin       |mdate    |     87|          0|         0|
#> |End         |mdate    |     87|          0|         0|
#> ----------------------------------------------------------
```

URL

- Wikipedia: https://en.wikipedia.org/wiki/List_of_Roman_emperors
- UNRV: https://www.unrv.com/government/emperor.php
- Britannica: https://www.britannica.com/place/list-of-Roman-emperors-2043294

Mapping

- wikipedia: Variable Mapping

from	*to*
name	ID
reign.start	Begin
reign.end	End
name.full	FullName
birth	Birth
death	Death

birth.cty	CityBirth
birth.prv	ProvinceBirth
rise	Rise
cause	Cause
killer	Killer
dynasty	Dynasty
era	Era
notes	Notes
verif.who	Verif

- UNRV: Variable Mapping

from	*to*
'Common Name'	ID
Beg	Begin
'Full Name/Imperial Name'	FullName
'Dynasty/Class/Notes'	Dynasty

- britannica: Variable Mapping

from	*to*
Name	ID
reign_start	Begin
reign_end	End

Source

- Wikipedia, 'List_of_Roman_emperors', https://en.wikipedia.org/wiki/List_of_Roman_emperors, Accessed on 2021-07-22.
- United Nations of Roma Victrix, 'Roman Emperor list', https://www.unrv.com/government/emperor.php, Accessed on 2021-07-22.
- Britannica, 'List of Roman emperors', https://www.britannica.com/topic/list-of-Roman-emperors-2043294, Accessed on 2021-07-22.

`filter_datacube` *Filtering datacube datasets to a certain date*

Description

Filtering datacube datasets to a certain date

Usage

```
filter_datacube(datacube, date = Sys.Date())
```

Arguments

datacube	A datacube, i.e. a list of data frames with Begin and End date variables.
date	A date (of class Date or character) at which to filter the datacube.

Examples

```
filter_datacube(emperors, date = "0100")
```

---

find *Find elements within manydata*

Description

Find elements within manydata

Usage

```
find_ID(df, id_col = "ID")

find_common_ID(..., id_col = "ID")

find_duplicates(df, id_col = "ID")
```

Arguments

df	A data frame to be scored.
id_col	The name of the column containing IDs. Default is "ID".
...	Data frames to compare

Examples

```
find_duplicates(emperors$Wikipedia)
```

---

find_year *Creates Numerical IDs from Signature Dates*

Description

Agreements should have a unique identification number that is meaningful, we condense their signature dates to produce this number.

Usage

```
find_year(date)
```

Arguments

date	A date variable

Value

A character vector with condensed dates

Examples

```
## Not run:
IEADB <- dplyr::slice_sample(manyenviron::agreements$IEADB, n = 10)
code_dates(IEADB$Title)

## End(Not run)
```

---

pluck *Selects a single dataset from a datacube*

Description

This function is reexported/wrapped from the {purrr} package. It allows users to select a single dataset from one of the datacubes available across the 'many* packages'. It additionally invites users to cite the selected dataset.

Usage

```
pluck(.x, ..., .default = NULL)
```

Arguments

.x	The datacube
...	The name of the dataset in the datacube
.default	Value to use if target is NULL or absent.

Value

The selected dataset

Examples

```
pluck(emperors, "UNRV")
```

recollect *Pastes unique string vectors*

Description

For use with dplyr::summarise, for example

Usage

```
recollect(x, collapse = "_")
```

Arguments

x	A vector
collapse	String indicating how elements separated

Details

This function operates similarly to reunite, but instead of operating on columns/observations, it pastes together unique rows/observations.

Value

A single value

Examples

```
data <- data.frame(ID = c(1,2,3,3,2,1))
data1 <- data.frame(ID = c(1,2,3,3,2,1), One = c(1,NA,3,NA,2,NA))
recollect(data$ID)
recollect(data1$One)
```

repaint *Fills missing data by lookup*

Description

Fills missing data where known by other observations with the same id/index

Usage

```
repaint(df, id, var)
```

Arguments

df	a dataframe
id	a string identifying a column in the dataframe for indexing
var	a string identifying a column or columns in the dataframe to be filled

Value

A dataframe

Examples

```
data <- data.frame(ID = c(1,2,3,3,2,1),
                   One = c(1,NA,3,NA,2,NA),
                   Two = c(NA,"B",NA,"C",NA,"A"))
repaint(data, "ID", c("One","Two"))
```

---

resolving *Resolving multiple observations of the same variable into one*

Description

This family of functions provides row-wise summarization for data frames or tibbles, returning a single value per row based on specified columns. They are useful for tasks like extracting typical or summary values from multiple variables, simplifying wide data structures, and imputing representative values.

Usage

```
resolve_unite(.data, vars, na.rm = TRUE)

resolve_coalesce(.data, vars)

resolve_min(.data, vars, na.rm = TRUE)

resolve_max(.data, vars, na.rm = TRUE)

resolve_random(.data, vars, na.rm = TRUE)

resolve_precision(.data, vars)

resolve_mean(.data, vars, na.rm = TRUE)

resolve_mode(.data, vars, na.rm = TRUE)

resolve_median(.data, vars, na.rm = TRUE)

resolve_consensus(.data, vars, na.rm = TRUE)
```

Arguments

`.data`	A data frame or tibble containing the variables.
`vars`	A vector of variables from `.data` to be resolved or converged. If this argument is left unspecified, then all variables will be merged together.
`na.rm`	Logical whether missing values (NAs) should be removed before operation of the function. Note that unlike how the `na.rm` argument operates in functions in base R, e.g. `max()`, here the default is TRUE.

Unite

Uniting returns all the unique values as a set, separated by commas and contained within braces. Note that uniting always returns a character/string vector, which enables it to accommodate different classes of variables. The order of the values reflects their first appearance; that is, they are not ordered by increasing value.

Coalesce

Coalescing returns a vector of the first non-missing values found when reading the variables from left to right. That is, missing values in the first vector may be filled by observations in the second vector, or later vectors if the second vector also misses an observation for that cell. Variables can be reordered manually.

Min and Max

These functions return a vector containing each row's minimum or maximum value. Note that these functions work not only on numeric and date vectors, but also on character string vectors. For character data, these functions will return the shortest or longest strings, respectively, in each row.

Random

This function returns a vector of values selected randomly from among the values contained in each row. Note that by default `na.rm = TRUE`, which means that missing data will not be selected at random by default, which can also change the probability distribution by each row. Where `na.rm = FALSE`, the probability of each value being selected is uniform.

Precision

This function returns a vector that maximises the precision of the values in each row. For numeric vectors, precision is expressed in significant digits, such that 1.01 would be more precise than 1. For character vectors, precision is expressed in terms of the character length proportional to the max character length in the row. This applies also to messydates, meaning precision is expressed in the lowest level date component specified, such that 2008-10 would be more precise than 2008, and 2008-10-10 would be more precise still.

Mean and median

These functions return a vector of the means or medians, respectively, of the values in each row.

Consensus

This function returns a vector of consensus values, i.e. where there is no variation in values by each row. If the values (excluding missing values by default) are not equivalent, then an NA is returned for that row.

Examples

```
test <- data.frame(preferred_dataset = c(1,6,NA),
                   more_comprehensive = c(1,3,3),
                   precise_where_available = c(NA,3.3,4.1))
test
resolve_unite(test)
resolve_coalesce(test)
resolve_min(test)
resolve_max(test)
resolve_random(test)
resolve_precision(test)
resolve_mean(test)
resolve_mode(test)
resolve_median(test)
resolve_consensus(test)
```

---

reunite *Pastes unique string vectors*

Description

A vectorised function for use with dplyr's mutate, etc

Usage

```
reunite(..., sep = "_")
```

Arguments

...	Variables to pass to the function, currently only two at a time
sep	Separator when vectors reunited, by default "_"

Value

A single vector with unique non-missing information

Examples

```
data <- data.frame(fir=c(NA, "two", "three", NA),
                   sec=c("one", NA, "three", NA), stringsAsFactors = FALSE)
transmutate(data, single = reunite(fir, sec))
```

scores *Scoring Functions for Data Quality Checks*

Description

A set of functions to assess various aspects of data quality. including a comprehensive dataset score as well as individual scores for specific data quality dimensions such as date consistency, duplicates, recency, frequency, time, coding, comments, sources, missing values, and variables.

According to the literature, data quality can be assessed by checking for consistency, completeness, accuracy, timeliness, and uniqueness of the data. Consistency means that the data is logically coherent, completeness means that all required data is present, accuracy means that the data is correct and reliable, timeliness means that the data is up-to-date, and uniqueness means that there are no duplicate records.

Usage

```
score_dataset(df)

score_obs_no(df)

score_var_no(df)

score_completeness(df)

score_date_consistency(df)

score_date_scope(df)

score_obs_info(df, id_col = "ID")

score_coding(df)

score_comments(df)

score_var_info(df)
```

Arguments

df	A data frame to be scored.
id_col	The name of the column containing IDs. Default is "ID".

Details

These functions are designed to help assess the quality of data in a data frame. Each function checks a specific aspect of the data and returns a score or a message indicating the quality of that aspect. The functions include:

- score_date_consistency: Proportion of invalid date pairs (End <= Begin).
- score_duplicates: Proportion of duplicate IDs.

References

Wang, R. Y., & Strong, D. M. (1996). Beyond accuracy: What data quality means to data consumers. Journal of Management Information Systems, 12(4), 5-34.

Examples

```
score_dataset(emperors)
score_obs_no(emperors)
score_var_no(emperors)
score_completeness(emperors)
score_date_consistency(emperors)
score_date_scope(emperors)
score_obs_info(emperors)
score_var_info(emperors)
```

---

transmutate *Drop only columns used in formula*

Description

A function between dplyr's transmute and mutate

Usage

```
transmutate(.data, ...)
```

Arguments

.data	Data frame to pass to the function
...	Variables to pass to the function

Value

Data frame with mutated variables and none of the variables used in the mutations, but, unlike dplyr::transmute(), all other unnamed variables.

Source

https://stackoverflow.com/questions/51428156/dplyr-mutate-transmute-drop-only-the-columns-used-in-the-formula

Examples

```
pluck(emperors, "Wikipedia")
transmutate(emperors$Wikipedia, Beginning = Begin)
```

Index