Package: manydata (via r-universe)

March 21, 2025

Title A Portal for Global Governance Data

Version 1.0.0

Date 2025-03-20

Description This is the core package for the many packages universe.
It includes functions to help researchers work with and
contribute to event datasets on global governance.

License CC BY 4.0

URL `https://globalgov.github.io/manydata`, `https://manydata.ch/`

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

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

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

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

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

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

Config/pak/sysreqs git libicu-dev libssl-dev

Repository https://globalgov.r-universe.dev

RemoteUrl https://github.com/globalgov/manydata

RemoteRef HEAD

RemoteSha c9c24f514dc23ecd1eb8ccdac5776d08e24cc62e

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")
```

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_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 = "manyID")
```

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.

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/topic/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.

`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

Resolving multiple observations of the same variable into one

Usage

```
resolve_coalesce(.data, vars)

resolve_unite(.data, vars)

resolve_min(.data, vars)

resolve_max(.data, vars)

resolve_random(.data, vars)

resolve_precision(.data, vars)
```

Arguments

`.data`	A data frame or tibble containing the variables.
`vars`	A vector of variables from `.data` to be resolved or converged.

Unite

Note that uniting always returns a character/string vector. Values are separated by commas and a set is contained within braces.

Examples

```
test <- data.frame(bloop.x = c(1,6,NA),
                   bloop.y = c(2,NA,3), bloop = c(NA,3,4))
resolve_coalesce(test)
resolve_unite(test)
resolve_min(test)
resolve_max(test)
resolve_random(test)
resolve_precision(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))
```

`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