Package 'manydata'

November 19, 2022

Title A Portal for Global Governance Data

Version 0.8.2

Date 2022-11-17

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://github.com/globalgov/manydata

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

Depends R (>= 3.5.0)

Imports tibble, dplyr, messydates, purrr, rlang, stringr, usethis, jsonlite, remotes, httr, ggplot2 (>= 3.4.0), migraph, cshapes, ggraph, janitor, tidyr, plyr, zoo

Suggests testthat, anytime, mice, withr, haven, readxl, readr, knitr, rmarkdown

Encoding UTF-8

LazyData true

RoxygenNote 7.2.1

VignetteBuilder knitr

NeedsCompilation no

Author James Hollway [cre, aut, ctb] (IHEID, <https://orcid.org/0000-0002-8361-9647>), Henrique Sposito [ctb] (IHEID, <https://orcid.org/0000-0003-3420-6085>), Bernhard Bieri [ctb] (IHEID, <https://orcid.org/0000-0001-5943-9059>), Esther Peev [ctb] (IHEID, <https://orcid.org/0000-0002-9678-2777>), Jael Tan [ctb] (IHEID, <https://orcid.org/0000-0002-6234-9764>)

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

Repository CRAN

Date/Publication 2022-11-19 13:00:10 UTC

R topics documented:

`coalesce_rows` *Get first non-missing*

Description

For use with dplyr::summarise, for example

Usage

```
coalesce_rows(x)
```

Arguments

x	A vector

Details

This function operates similarly to coalesce for columns, that is picking the first non-missing observation, but on observations rather than variables.

Value

A single value

Source

https://stackoverflow.com/questions/40515180/dplyr-how-to-find-the-first-non-missing-string-by-groups

Examples

```
dplyr::summarise(emperors$wikipedia, coalesce_rows(emperors$wikipedia))
coalesce_rows(emperors$wikipedia$Beg)
```

`consolidate` *Consolidate database into a single dataset*

Description

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

Usage

```
consolidate(
  database,
  rows = "any",
  cols = "any",
  resolve = "coalesce",
  key = "manyID"
)
```

Arguments

`database`	A database object from one of the many packages
`rows`	Which rows or units to retain. By default "any" (or all) units are retained, but another option is "every", which retains only those units that appear in all parent datasets.
`cols`	Which columns or variables to retain. By default "any" (or all) variables are retained, but another option is "every", which retains only those variables that appear in all parent datasets.
`resolve`	How should conflicts between observations be resolved? By default "coalesce", but other options include: "min", "max", "mean", "median", and "random". "coalesce" takes the first non-NA value. "max" takes the largest value. "min" takes the smallest value. "mean" takes the average value. "median" takes the median value. "random" takes a random value. For different variables to be resolved differently, you can specify the variables' names alongside how each is to be resolved in a list (e.g. `resolve = c(var1 = "min", var2 = "max")`). In this case, only the variables named will be resolved and returned.

key — An ID column to collapse by. By default "many_ID". 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 database (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

Text variables are dropped for more efficient consolidation.

Value

A single tibble/data frame.

Examples

```
consolidate(database = emperors, key = "ID")
consolidate(database = favour(emperors, "UNRV"), rows = "every",
cols = "every", resolve = "coalesce", key = "ID")
consolidate(database = emperors, rows = "any", cols = "every",
resolve = "min", key = "ID")
consolidate(database = emperors, rows = "every", cols = "any",
resolve = "max", key = "ID")
consolidate(database = emperors, rows = "every", cols = "every",
resolve = "median", key = "ID")
consolidate(database = emperors, rows = "every", cols = "every",
resolve = "mean", key = "ID")
consolidate(database = emperors, rows = "every", cols = "every",
resolve = "random", key = "ID")
consolidate(database = emperors, rows = "every", cols = "every",
resolve = c(Beg = "min", End = "max"), key = "ID")
consolidate(database = emperors, rows = "any", cols = "any",
resolve = c(Death = "max", Cause = "coalesce"),
key = c("ID", "Beg"))
```

---

db_profile *Database profile functions*

Description

Database profiling functions that returns confirmed, unique, missing, conflicting, or majority values in all (non-ID) variables in the datasets in a 'many' package database.

Usage

```
db_plot(database, key = "manyID", variable = "all", category = "all")

db_comp(database, key = "manyID", variable = "all", category = "all")
```

Arguments

database	A many database.
key	A variable key to join datasets by, "manyID" by default.
variable	Would you like to focus on one, or more, specific variables? 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", "conflicting", or "majority". For multiple variables, please declare categories as a vector.

Details

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

`db_plot()` plots the database profile.

`db_comp()` creates a tibble comparing the variables in a database.

Value

A plot, or a tibble, with the profile of the variables across all datasets in a "many" database. For multiple categories across multiple variables, the functions return all rows that contain at least one of the selected variables coded as one of the categories.

Examples

```
db_plot(database = emperors, key = "ID")
db_plot(database = emperors, key = "ID", variable = c("Beg", "End"))
db_plot(database = emperors, key = "ID", variable = c("Beg", "End"),
category = c("conflict", "unique"))


db_comp(database = emperors, key = "ID")
db_comp(database = emperors, key = "ID", variable = "Beg")
db_comp(database = emperors, key = "ID", variable = c("Beg", "End"),
category = "conflict")
db_comp(database = emperors, key = "ID", variable = c("Beg", "End"),
category = c("conflict", "unique"))
```

`emperors` *Emperors database documentation*

Description

Emperors database documentation

Usage

```
emperors
```

Format

The emperors database is a list that contains the following 3 datasets: wikipedia, UNRV, britannica. For more information and references to each of the datasets used, please use the `data_source()` and `data_contrast()`. functions.

wikipedia: A dataset with 68 observations and the following 15 variables: ID, Beg, 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, Beg, End, Birth, Death, FullName, Dynasty.

britannica: A dataset with 87 observations and the following 3 variables: ID, Beg, End.

Details

```
#> $wikipedia
#> ------------------------------------------------------------------------------
#> |  Column Name  |  Data Type  |  Observations  |  Missing  |  Missing (%)  |
#> ------------------------------------------------------------------------------
#> |      ID       |  character  |             68|     0     |       0       |
#> |      Beg      |    mdate    |             68|     0     |       0       |
#> |      End      |    mdate    |             68|     0     |       0       |
#> |   FullName    |  character  |             68|     0     |       0       |
#> |     Birth     |  character  |             68|     5     |     7.35      |
#> |     Death     |  character  |             68|     0     |       0       |
#> |   CityBirth   |  character  |             68|    17     |      25       |
#> | ProvinceBirth |  character  |             68|     0     |       0       |
#> |     Rise      |  character  |             68|     0     |       0       |
#> |     Cause     |  character  |             68|     0     |       0       |
#> |    Killer     |  character  |             68|     0     |       0       |
#> |    Dynasty    |  character  |             68|     0     |       0       |
#> |      Era      |  character  |             68|     0     |       0       |
#> |     Notes     |  character  |             68|    22     |     32.35     |
#> |     Verif     |  character  |             68|    57     |     83.82     |
#> ------------------------------------------------------------------------------
#>
#>
```

```
#> $UNRV
#> ------------------------------------------------------------------------------
#> |  Column Name  |  Data Type  |  Observations  |  Missing  |  Missing (%)  |
#> ------------------------------------------------------------------------------
#> |      ID       |  character  |             99|     0     |       0       |
#> |      Beg      |    mdate    |             99|     0     |       0       |
#> |      End      |    mdate    |             99|     0     |       0       |
#> |     Birth     |  character  |             99|     0     |       0       |
#> |     Death     |  character  |             99|     0     |       0       |
#> |   FullName    |  character  |             99|     5     |     5.05      |
#> |    Dynasty    |  character  |             99|    37     |     37.37     |
#> ------------------------------------------------------------------------------
#>
#>
#> $britannica
#> ------------------------------------------------------------------------------
#> |  Column Name  |  Data Type  |  Observations  |  Missing  |  Missing (%)  |
#> ------------------------------------------------------------------------------
#> |      ID       |  character  |             87|     0     |       0       |
#> |      Beg      |    mdate    |             87|     0     |       0       |
#> |      End      |    mdate    |             87|     0     |       0       |
#> ------------------------------------------------------------------------------
```

`favour` *Favour datasets in a database*

Description

Favour datasets in a database

Usage

```
favour(database, dataset)

favor(database, dataset)
```

Arguments

`database`	A many database
`dataset`	The name of one, or more, datasets within the database to be favoured over others.

Details

The dataset declared becomes the reference for the first non NA value. If more than one dataset is declared, please list datasets in increasing order of importance (.i.e. last dataset should be favoured over previous).

Value

The database with datasets re-ordered accordingly

Examples

```
favour(emperors, "UNRV")
favour(emperors, c("wikipedia", "UNRV", "britannica"))
```

`get_packages` *Find and download packages in the many packages universe*

Description

Find and download packages in the many packages universe.

Usage

```
get_packages(pkg)
```

Arguments

pkg	A character vector of package names or number of a package. To download multiple packages at once, please declare package names as a vector (e.g. c("pkg1", "pkg2)).

Details

The function 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 by simply calling the function. If users intend to download and install a package from the universe, they can type the package name within the function.

Value

If no package name is provided, this function prints a table (tibble) to the console with details on packages that are currently available within the many universe. This includes the name and description of the package, the latest installed and release version number, and the latest release date. It also include a list of numbers which orders the package and can be used to load the respective package instead of the name. If one or more package names are provided, these will be installed from Github.

Examples

```
#get_packages()
```

`network_map` *Plot geographical networks*

Description

Creates a plot of the a unimodal geographical network.

Usage

```
network_map(object, date, theme = "light")
```

Arguments

`object`	Unimodal geographical network. Needs to contain ISO3c country IDs.
`date`	String date at which the network snapshot was taken (e.g. "2010-01-01"). Used by `{cshapes}` to plot the correct map. Date can be between 1886 and 2021.
`theme`	Theme you would like to use to plot the graph. Available themes are "light", "dark", and "earth".

Value

A map of a country level geographical network.

Examples

```
# Plot a network of environmental agreements signed in 2010
# extracted from {manyenviron} using data from ECOLEX.
# Light theme
membership <- migraph::as_igraph(data.frame(
  from = c("ETH", "ETH", "ETH", "ETH", "UKR", "UKR",
           "MOZ", "MOZ", "JPN", "JPN"),
  to = c("GNQ", "KEN", "TZA", "RWA", "CHN", "POL",
         "COL", "NZL", "MNE", "LKA")))
network_map(membership, date = "2010-01-01", theme = "light") +
  ggplot2::labs(title = "Sample of International Environmental Treaties 2010",
                subtitle = "Ecolex data",
                caption = "Created with love by {}")
# Earth theme
network_map(membership, date = "2010-01-01", theme = "earth") +
  ggplot2::labs(title = "International Environmental Treaties 2010",
                subtitle = "Ecolex data",
                caption = "Created with love by {}")
```

plot_releases *A plotting function that visualises historical milestones/releases*

Description

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

Usage

```
plot_releases(repo)
```

Arguments

repo	the github repository to track, e.g. "globalgov/manydata"

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/

Examples

```
#plot_releases("globalgov/manydata")
```

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"))
```

report *Set of data structure exploration functions for users*

Description

The report family of functions allows users to quickly get information about and compare several aspects of a package in the many packages universe, and its' databases and datasets.

Usage

```
data_source(pkg, database = NULL, dataset = NULL)

data_contrast(pkg, database = NULL, dataset = NULL)

data_evolution(pkg, database, dataset, preparation_script = FALSE)

open_codebook(pkg, database, dataset)
```

Arguments

pkg	character string of the many package to report data on. Required input.
database	vector of character strings of the many package to report data on a specific database in a many package If NULL, the function returns a summary of all databases in the many package NULL by default for `data_source()` and `data_contrast()`.
dataset	character string of the many package to report data on a specific dataset in a specific database of a many package If NULL and database is specified, returns database level metadata. NULL by default for `data_source()` and `data_contrast()`.
preparation_script	Would you like to open the preparation script for the dataset? By default false.

Details

`data_source()` displays names of the database/datasets and source material of data in a many package.

`data_contrast()` displays information about databases and datasets contained in them. Namely the number of unique ID's, the percentage of missing data, the number of observations, the number of variables, the minimum beginning date and the maximum ending date as well as the most direct URL to the original dataset.

`data_evolution()` enables users to access the differences between raw data and the data made available to them in one of the 'many' packages.

`open_codebook()` opens the original codebook of the specified dataset to allow users to look up the original coding rules. Note that no original codebook might exist for certain datasets. In the latter case, please refer to the source URL provided with each dataset by running `manydata::data_contrast()` as further information on coding rules available online.

Value

A dataframe with the data sources

A list with the desired metadata to compare various datasets in a many package.

Either the data comparison between raw and available data or the preparation script detailing all the steps taken to prepare raw data before making it available in one of the 'many' packages.

Opens a pdf version of the original codebook of the specified dataset, if available.

Examples

```
data_source(pkg = "manydata")


data_contrast(pkg = "manydata")


data_evolution(pkg = "manydata", database = "emperors",
dataset = "wikipedia")
#data_evolution(pkg = "manytrade", database = "agreements",
#dataset = "GPTAD")
```

---

retrieve_treaty *Retrieve international treaties*

Description

Some databases and datasets across the 'many* packages' (e.g. manyenviron) contain a myriad of information on international treaties governing an international domain. Researchers can, for example, use `retrieve_bilaterals()` to retrieve which countries have signed bilateral agreements in a respective year. Alternatively, researchers can use `retrieve_multilaterals()` to retrieve the titles of all multilateral agreements signed in the past 10 years. Alternatively, researchers can retrieve treaties that modify, amend, or expand other treaties with `retrieve_links()`. Or, even, researchers can retrieve membership lists of countries part to a certain treaty with `retrieve_membership_list()`. Finally, researchers can retrieve treaty texts available in 'many' datasets with `retrieve_texts()`. To retrieve information from several datasets in a database, researchers can `consolidate()` a database into one dataset with some combination of the rows, columns, and observations before getting the desired information.

Usage

```
retrieve_bilaterals(dataset)

retrieve_multilaterals(dataset)

retrieve_membership_list(dataset, actor = "StateID", treaty_type = NULL)
```

```
retrieve_links(dataset, treaty_type = NULL)

retrieve_texts(dataset, treaty_type = NULL)
```

Arguments

dataset	A dataset from one of the many packages.
actor	An actor variable. "StateID", by default.
treaty_type	The type of treaties to be returned. NULL, by default. Other options are "bilateral" or "multilateral".

Value

A tibble of bilateral agreements.

A tibble of multilateral agreements.

A tibble of manyIDs and countries part of the treaty.

A tibble of manyIDs and their links.

A tibble of manyIDs and their texts.

Examples

```
membs <- tibble::tibble(manyID = c("ROU-RUS[RFP]_1901A",
"ROU-RUS[RFP]_1901E", "GD16FI_1901A"),
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"),
Beg = c("1901-02-22", "1901-02-22", "1901-06-24"))
retrieve_bilaterals(membs)
membs <- tibble::tibble( manyID = c("ROU-RUS[RFP]_1901A",
"ROU-RUS[RFP]_1901A", "GD16FI_1901A"),
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"),
Beg = c("1901-02-22", "1901-02-22", "1901-06-24"))
retrieve_multilaterals(membs)
membs <- tibble::tibble(StateID = c("ROU", "RUS", "DNK"),
manyID = c("ROU-RUS[RFP]_1901A", "ROU-RUS[RFP]_1901A", "GD16FI_1901A"))
retrieve_membership_list(dataset = membs)
```

```
membs <- tibble::tibble(manyID = c("ROU-RUS[RFP]_1901A",
"ROU-RUS[RFP]_1901A:ROU-RUS[RFP]_1901A",
"GD16FI_1901A"))
retrieve_links(dataset = membs)
membs <- tibble::tibble(manyID = c("ROU-RUS[RFP]_1901A",
"ROU-RUS[RFP]_1901A:ROU-RUS[RFP]_1901A",
"GD16FI_1901A"),
Text = c("treaty 1", "treaty 2", "treaty 3"))
retrieve_texts(dataset = membs)
#retrieve_texts(dataset = manyenviron::agreements$HUGGO)
```

`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 = Beg)
```

Index