Package 'COINr'

September 16, 2021

Type Package

Title Composite Indicator Construction and Analysis

Version 0.5.5

Maintainer William Becker <william.becker@bluefoxdata.eu>

Description A comprehensive high-level package for composite indicator construction and analysis. It is a ``development environment''
for composite indicators and scoreboards, which includes utilities for construction (indicator selection, denomination, imputation,
data treatment, normalisation, weighting and aggregation) and analysis (multivariate analysis, correlation plotting, short cuts for
principal component analysis, global sensitivity analysis, and more). A composite indicator is completely encapsulated inside a single
hierarchical list called a ``COIN''. This allows a fast and efficient work flow, as well as making quick copies, testing methodological variations and making comparisons.
It also includes many plotting options, both statistical (scatter plots, distribution plots) as well as for presenting results (maps,
bar charts, radar charts, and more).
Finally, three Shiny apps are available which enable fast data exploration, results presentation, and checking the effects of
altering weights.
Full documentation is found in the online book at <https://bluefoxr.github.io/COINrDoc/>, as well as the vignette.

License MIT + file LICENSE

Encoding UTF-8

URL https://bluefoxr.github.io/COINrDoc/

BugReports https://github.com/bluefoxr/COINr/issues

LazyData true

RoxygenNote 7.1.1

Imports dplyr (>= 1.0.4), magrittr (>= 2.0.1), reshape2 (>= 1.4.4),
openxlsx (>= 4.2.3), corrplot (>= 0.84), plotly (>= 4.9.3),
matrixStats (>= 0.58.0), stats, rlang (>= 0.4.10), purrr (>=
0.3.4), tidyr (>= 1.1.2), stringr (>= 1.4.0), e1071 (>= 1.7-4),

tibble (>= 3.0.6), shiny (>= 1.6.0), reactable (>= 0.2.3), ggplot2 (>= 3.3.3), readxl (>= 1.3.1), Amelia, rmarkdown

Depends R (>= 4.0.0)

Suggests spelling, knitr

VignetteBuilder knitr

Language en-GB

NeedsCompilation no

Author William Becker [aut, cre, cph] (<https://orcid.org/0000-0002-6467-4472>)

Repository CRAN

Date/Publication 2021-09-16 08:20:01 UTC

R topics documented:

`aggregate` *Aggregate indicators*

Description

Takes indicator data and a specified structure and hierarchically aggregates according to the structure specified in `IndMeta`. Uses a variety of aggregation methods as specified by `agtype`, which can be different for each level of aggregation (see `agtype_by_level`).

Usage

```
aggregate(
  COIN,
  agtype = "arith_mean",
  agweights = NULL,
  dset = NULL,
  agtype_bylevel = NULL,
  agfunc = NULL,
  out2 = NULL
)
```

Arguments

COIN	COIN object
agtype	The type of aggregation method. One of either: • "arith_mean" - weighted arithmetic mean • "median" - weighted median • "geom_mean" - weighted geometric mean • "harm_mean" - weighted harmonic mean • "copeland" - weighted Copeland method • "custom" - a custom function - see agfunc • "mixed" - a different aggregation method for each level. In this case, aggregation methods are specified as any of the previous options using the agtype_bylevel argument.
agweights	The weights to use in the aggregation. This can either be: NULL, in which case it will use the weights that were attached to IndMeta and AggMeta in assemble() (if they exist), or A character string which corresponds to a named list of weights stored in .$Parameters$Weights. You can either add these manually or through rew8r(). E.g. entering agweights = "Original" will use the original weights read in on assembly. This is equivalent to agweights = NULL. Or, a data frame of weights to use in the aggregation.
dset	Which data set (contained in COIN object) to use
agtype_bylevel	A character vector with aggregation types for each level. Note that if this is specified, agtype *must* be specified as agtype = "mixed", otherwise agtype_by_level will be ignored.
agfunc	A custom function to use for aggregation if agtype = "custom", of the type $y = f(x, w)$, where y is a scalar aggregated value and x and w are vectors of indicator values and weights respectively. Ensure that NAs are handled (e.g. set na.rm = T) if your data has missing values.
out2	Where to output the results. If "COIN" (default for COIN input), appends to updated COIN, otherwise if "df" outputs to data frame.

Value

An updated COIN containing the new aggregated data set at .$Data$Aggregated.

Examples

```
# assemble a COIN first
ASEM <- assemble(IndData = ASEMIndData, IndMeta = ASEMIndMeta, AggMeta = ASEMAggMeta)
# normalise the data
ASEM <- normalise(ASEM, dset = "Raw")
# aggregate the data
ASEM <- COINr::aggregate(ASEM, agtype="arith_mean", dset = "Normalised")
# check aggregated data set exists
stopifnot(!is.null(ASEM$Data$Aggregated))
```

ASEMAggMeta *ASEM aggregate metadata*

Description

This contains all the metadata for the aggregate groups, including the names, weights and codes. See the ASEM Portal for further information and detailed description of each indicator, and COINr documentation for the formatting of this data set.

Usage

```
ASEMAggMeta
```

Format

A data frame with 8 rows and 9 variables:

Source

https://composite-indicators.jrc.ec.europa.eu/asem-sustainable-connectivity/repository

---

ASEMIndData *ASEM raw indicator data*

Description

A data set containing raw values of indicators for 51 countries, groups and denominators. See the ASEM Portal for further information and detailed description of each indicator, and COINr documentation for the formatting of this data set.

Usage

```
ASEMIndData
```

Format

A data frame with 51 rows and 60 variables.

Source

https://composite-indicators.jrc.ec.europa.eu/asem-sustainable-connectivity/repository

---

`ASEMIndMeta` *ASEM indicator metadata*

Description

This contains all metadata for ASEM indicators, including names, weights, directions, etc. See the ASEM Portal for further information and detailed description of each indicator, and COINr documentation for the formatting of this data set.

Usage

```
ASEMIndMeta
```

Format

A data frame with 49 rows and 9 variables

Source

https://bluefoxr.github.io/COINrDoc/coins-the-currency-of-coinr.html#aggregation-metadata

---

`assemble` *Build COIN object*

Description

This takes the raw data provided by the user and puts it into a list format (COIN object) that is recognised by COINr. It also checks whether there are any syntax errors in the data provided. Optionally, you can exclude or include indicators using the `include` and `exclude` arguments. Note that if an indicator is specified in BOTH `include` and `exclude`, it will be excluded.

Usage

```
assemble(
  IndData,
  IndMeta,
  AggMeta,
  include = NULL,
  exclude = NULL,
  preagg = NULL,
  use_year = NULL,
  impute_latest = FALSE
)
```

Arguments

IndData	A data frame of indicator data.
IndMeta	A data frame containing auxiliary information for each indicator
AggMeta	A data frame specifying the names and weights of each aggregation group
include	Optional argument specifying a subset of indicator codes to include (default all indicators included)
exclude	Optional argument specifying a subset of indicator codes to exclude (default none excluded)
preagg	Set to TRUE if you want to assemble a COIN using pre-aggregated data (typically for ex-post analysis)
use_year	If IndData includes a Year column, and there are multiple observations for each unit (one per year), this can be set to a target year. For example, setting use_year = 2020 will filter IndData to only include points from 2020. Keeping in mind that a COIN represents a single year of data.
impute_latest	Logical: if TRUE, imputes missing data points using most recent value from previous years. If FALSE (default) simply extracts the data frame as is. This only works if !is.null(use_year) and there are previous years of data available (before use_year). Currently does not support imputation using future values or interpolation.

Details

A "COIN" is an S3 class which is a structured list of indicator data, metadata, results and methodology which is used throughout COINr. COINs are a convenient way to store all variables relating to the composite indicator in a single named object. This keeps the workspace tidy, but also allows fast and concise calls to functions, as well as copying COINs to introduce methodological variations, and enables complex operations such as global sensitivity analysis (see sensitivity().

For general information on COINs see the COINr vignette as well as the relevant chapter in the COINr online documentation.

For details on copying, adjusting and comparing COINs see the COINr chapter on adjustments and comparisons.

Value

A "COIN" (list) formatted to the specifications of COINr. Note that the COIN object is an S3 class. It doesn't impose restrictions on the structure of the list.

Examples

```
# build the ASEM COIN
ASEM <- assemble(IndData = ASEMIndData, IndMeta = ASEMIndMeta, AggMeta = ASEMAggMeta)
```

BoxCox *Box Cox transformation*

Description

Simple Box Cox, with no optimisation of lambda. See COINr online documentation for more details.

Usage

```
BoxCox(x, lambda, makepos = TRUE)
```

Arguments

x	A vector or column of data to transform
lambda	The lambda parameter of the Box Cox transform
makepos	If TRUE (default) makes all values positive by subtracting the minimum and adding 1.

Value

A vector of length `length(x)` with transformed values.

See Also

- `treat()` Outlier treatment

Examples

```
# get a column of data with outliers
x <- ASEMIndData$Tariff
# Apply Box Cox
xBox <- BoxCox(x, lambda = 2)
# plot one against the other
plot(x, xBox)
```

build_ASEM *Builds ASEM example*

Description

A short cut function for building the ASEM COIN. This builds the ASEM COIN up to and including aggregated results. See the ASEM Portal for the underlying data set and online documentation for more information on COINs.

Usage

```
build_ASEM()
```

Value

COIN class object (a list) of the ASEM index, as well as information printed to the console - see `assemble()`.

Examples

```
# Build the ASEM COIN
ASEM <- build_ASEM()
```

checkData *Detailed unit data check and screener by data availability*

Description

Gives detailed tables of data availability, and optionally screens units based on a data availability threshold and presence of zeros. Units can be optionally "forced" to be included or excluded, making exceptions for the data availability threshold.

Usage

```
checkData(
  COIN,
  dset = NULL,
  ind_thresh = NULL,
  zero_thresh = NULL,
  unit_screen = "none",
  Force = NULL,
  out2 = "COIN"
)
```

Arguments

COIN	The COIN object
dset	The data set to be checked/screened
ind_thresh	A data availability threshold used for flagging low data and screening units if unit_screen != "none". Default 0.66. Specify as a fraction.
zero_thresh	As ind_thresh but for non-zero values. Defaults to 0.05, i.e. it will flag any units with less than 5% non-zero values (equivalently more than 95% zero values).
unit_screen	Specifies whether and how to screen units based on data availability or zero values. • If set to "none" (default), does not screen any units. • If set to "byNA", screens units with data availability below ind_thresh • If set to "byzeros", screens units with non-zero values below zero_thresh • If set to "byNAandzeros", screens units based on either of the previous two criteria being true. • If you simply want to force a unit or units to be excluded (without any other screening), use the Force argument and set unit_screen = TRUE. unit_screen != "none" outputs a new data set .$Data$Screened.
Force	A data frame with any additional countries to force inclusion or exclusion. First column is "UnitCode". Second column "Status" either "Include" or "Exclude" for each country to force.
out2	Where to output the results. If "COIN" (default for COIN input), appends to updated COIN, otherwise if "list" outputs to data frame.

Details

The two main criteria of interest are NA values, and zeros. The summary table gives percentages of NA values for each unit, across indicators, and percentage zero values (*as a percentage of non*-NA *values*). Each unit is flagged as having low data or too many zeros based on thresholds.

This function currently only supports COINs as inputs, not data frames.

Value

An updated COIN with data frames showing missing data in .$Analysis, and if unit_screen != "none" outputs a new data set .$Data$Screened. If out2 = "list" wraps missing data stats and screened data set into a list.

Examples

```
# build ASEM COIN
ASEM <- assemble(IndData = ASEMIndData, IndMeta = ASEMIndMeta, AggMeta = ASEMAggMeta)
# return stats to the COIN, plus screened data set, return to list
ScreenedData <- checkData(ASEM, dset = "Raw", unit_screen = "byNA",
ind_thresh = 0.9, out2 = "list")
# See which units were removed
print(ScreenedData$RemovedUnits)
```

coin2Excel *Write a COIN to Excel*

Description

Takes a COIN and writes all main data tables and other things to an Excel file. This uses the 'openxlsx' package.

Usage

```
coin2Excel(COIN, fname = "COINresults.xlsx")
```

Arguments

COIN	A COIN object
fname	The file name to write to, as a character string

Value

An Excel workbook with each table on a separate named tab.

Examples

```
## Here we write a COIN to Excel, but this is done to a temporary directory
## to avoid "polluting" the working directory when running automatic tests.
## In a real case, set fname to a directory of your choice.
ASEM <- build_ASEM()
# write to Excel in temporary directory
coin2Excel(ASEM, fname = paste0(tempdir(), "\\ASEM_results.xlsx"))
# spreadsheet is at:
print(paste0(tempdir(), "\\ASEM_results.xlsx"))
# now delete temporary file to keep things tidy in testing
unlink(paste0(tempdir(),"\\ASEM_results.xlsx"))
```

COINToolIn *Import data directly from COIN Tool*

Description

This provides a direct interface for reading a COIN Tool input deck and converting it to COINr. You need to provide a COIN Tool file, with the "Database" sheet properly compiled.

Usage

```
COINToolIn(fname, makecodes = FALSE, oldtool = FALSE)
```

Arguments

fname	The file name and path to read, e.g. "C:/Documents/COINToolFile.xlsx".
makecodes	Logical: if TRUE, will generate short indicator codes based on indicator names, otherwise if FALSE, will use COIN Tool indicator codes "Ind.01", etc. Currently only does this for indicators, not aggregation groups.
oldtool	Logical: if TRUE, compatible with old COIN Tool (pre-release, early 2019 or earlier). There are some minor differences on where the elements are found.

Details

The COIN Tool is an Excel-based tool for building composite indicators.

See COINr online documentation for more details and an example.

Value

A list containing:

- .$IndData A data frame of imported indicator data to be input as the IndData argument in assemble().
- .$IndMeta A data frame of imported indicator metadata to be input as the IndMeta argument in assemble().
- .$AggMeta A data frame of imported aggregation metadata to be input as the AggMeta argument in assemble().

See Also

- coin2Excel() Export COIN contents to Excel

Examples

```
## Not run:
## This example downloads a COIN Tool spreadsheet containing example data,
## saves it to a temporary directory, unzips, and reads into R. Finally it
## assembles it into a COIN.

# Make temp zip filename in temporary directory
tmpz <- tempfile(fileext = ".zip")
# Download an example COIN Tool file to temporary directory
# NOTE: the download.file() command may need its "method" option set to a
# specific value depending on the platform you run this on. You can also
# choose to download/unzip this file manually.
download.file("https://knowledge4policy.ec.europa.eu/sites/default/
files/coin_tool_v1_lite_exampledata.zip", tmpz)
# Unzip
CTpath <- unzip(tmpz, exdir = tempdir())
# Read COIN Tool into R
l <- COINToolIn(CTpath, makecodes = TRUE)
# We can finish by assembling it
CT_exampleCOIN <- assemble(l$IndData, l$IndMeta, l$AggMeta)
```

```
## End(Not run)
```

---

coin_win *Winsorisation helper function*

Description

To be used inside `treat()` to avoid repetitions. Winsorises a numerical vector of data.

Usage

```
coin_win(
  icol,
  winmax,
  winchange = TRUE,
  t_skew = 2,
  t_kurt = 3.5,
  icode = NULL
)
```

Arguments

`icol`	The vector of data to Winsorise
`winmax`	The maximum number of points to Winsorise for each indicator. If `NA`, will keep Winsorising until skewness and kurtosis thresholds achieved (but it is likely this will cause errors).
`winchange`	Logical: if `TRUE`, Winsorisation can change direction from one iteration to the next. Otherwise if `FALSE` (default), no change.
`t_skew`	Absolute skew threshold (default 2).
`t_kurt`	Kurtosis threshold (default 3.5).
`icode`	The indicator name - used for error messages in `treat()`.

Details

Outliers are identified according to skewness and kurtosis thresholds. The algorithm attempts to reduce the absolute skew and kurtosis by successively Winsorising points up to a specified limit.

The process is detailed in the COINr online documentation.

Value

A list containing:

- `.$icol` the vector of treated data
- `.$imax` the indices of elements of the vector that were Winsorised as high values
- `.$imax` the indices of elements of the vector that were Winsorised as low values
- `.$winz` the total number of Winsorised points

See Also

- `treat()` Outlier treatment

Examples

```
# get a column of data with outliers
x <- ASEMIndData$Tariff
# Winsorise up to five points
winlist <- coin_win(x, winmax = 5)
# check the differences
data.frame(
Orig = x,
Treated = winlist$icol,
Changes = ifelse(x == winlist$icol, "Same", "Treated"))
```

`colourTable` *Conditionally formatted table*

Description

Given a data frame, generates a conditionally-formatted html table using reactable. This function is used by `iplotTable()`. It is a quick wrapper for reactable::reactable.

Usage

```
colourTable(
  df,
  freeze1 = TRUE,
  sortcol = NULL,
  sortorder = "desc",
  searchable = TRUE,
  pagesize = 10,
  cell_colours = NULL,
  reverse_colours = FALSE
)
```

Arguments

`df`	A data frame to be displayed as a table.
`freeze1`	If `TRUE` (default), freezes the first column. This may be for example the unit name or code.
`sortcol`	A column name to sort the table by. Defaults to first numerical column. Set to `"none"` to disable.
`sortorder`	Either `"desc"` for sorted column to be sorted from high to low (default) or `"asc"` for the opposite.

searchable	If TRUE, includes a search box
pagesize	The number of rows to display on each page.
cell_colours	A character vector of colour codes (e.g. Hex codes) to use for the colour palette. Should be in order of low to high values. Defaults to a simple green palette. See grDevices::colorRamp() for more info.
reverse_colours	If TRUE, reverses the colour map - useful for rank tables where lowest numbers mean high scores.

Value

An interactive table generated by reactable.

See Also

- iplotTable() Interactive table of indicator data (from a COIN)

Examples

```
# some random data
df <-  as.data.frame(matrix(runif(12), 3, 4))
# a names column
df <- cbind(Rnames = letters[1:3], df)
# round it
df <- roundDF(df)
# make a table
colourTable(df)
```

---

compareDF *Compare two data frames*

Description

A custom function for comparing two data frames of indicator data, to see whether they match up, at a specified number of significant figures.

Usage

```
compareDF(df1, df2, matchcol, sigfigs = 5)
```

Arguments

df1	A data frame
df2	Another data frame
matchcol	A common column name that is used to match row order. E.g. this might be UnitCode.
sigfigs	The number of significant figures to use for matching numerical columns

Details

This function compares numerical and non-numerical columns to see if they match. Rows and columns can be in any order. The function performs the following checks:

- Checks that the two data frames are the same size
- Checks that column names are the same, and that the matching column has the same entries
- Checks column by column that the elements are the same, after sorting according to the matching column

It then summarises for each column whether there are any differences, and also what the differences are, if any.

This is intended to cross-check results. For example, if you run something in COINr and want to check indicator results against external calculations.

Value

A list with comparison results. List contains:

- `.$Same`: overall summary: if `TRUE` the data frames are the same according to the rules specified, otherwise `FALSE`.
- `.$Details`: details of each column as a data frame. Each row summarises a column of the data frame, saying whether the column is the same as its equivalent, and the number of differences, if any. In case the two data frames have differing numbers of columns and rows, or have differing column names or entries in `matchcol`, `.$Details` will simply contain a message to this effect.
- `.$Differences`: a list with one entry for every column which contains different entries. Differences are summarised as a data frame with one row for each difference, reporting the value from `df1` and its equivalent from `df2`.

Examples

```
# take a sample of indicator data (including the UnitCode column)
data1 <- ASEMIndData[c(2,12:15)]
# copy the data
data2 <- data1
# make a change: replace one value in data2 by NA
data2[1,2] <- NA
# compare data frames
compareDF(data1, data2, matchcol = "UnitCode")
```

compTable *Rank comparison table between two COINs*

Description

Takes two COINs, and generates a rank comparison between specified indicator/aggregates. COINs must share at least some common unit codes, and the indicator selected by `isel`.

Usage

```
compTable(
  COIN1,
  COIN2,
  dset = "Raw",
  isel,
  COINnames = NULL,
  sort_by = "AbsRankChange"
)
```

Arguments

`COIN1`	First COIN
`COIN2`	Second COIN
`dset`	The data set of interest
`isel`	The indicator/column of interest
`COINnames`	An optional character vector of the names of `COIN1` and `COIN2`, to be used in the table headers.
`sort_by`	If `"RankCOIN1"`, sorts by the indicator values of COIN1, if `"RankCOIN2"`, sorts by `COIN2`, if `"RankChange"`, sorts by rank change, and if `"AbsRankChange"` sorts by absolute rank change.

Value

A data frame with ranks and rank changes between two COINs.

See Also

- `compTableMulti()` Comparison table between multiple COINs

Examples

```
ASEM <- build_ASEM()
# Make a copy
ASEMAltNorm <- ASEM
# Edit .$Method
ASEMAltNorm$Method$normalise$ntype <- "borda"
```

```
# Regenerate
ASEMAltNorm <- regen(ASEMAltNorm, quietly = TRUE)
# compare
CT <- compTable(ASEM, ASEMAltNorm, dset = "Aggregated", isel = "Index")
```

`compTableMulti` *Rank tables between multiple COINs*

Description

Takes multiple COINs (two or more), and generates a rank comparison for a single indicator or aggregate.

Usage

```
compTableMulti(
  COINs,
  dset = "Aggregated",
  isel = "Index",
  tabtype = "Ranks",
  ibase = 1,
  sort_table = TRUE,
  extra_cols = NULL
)
```

Arguments

`COINs`	A list of COINs
`dset`	The data set to extract the indicator from (must be present in each COIN). Default `"Aggregated"`.
`isel`	Code of the indicator or aggregate to extract from each COIN (must be present in the specified data set of each COIN). Default `"Index"`.
`tabtype`	The type of table to generate - `"Ranks"`, `"Diffs"`, or `"AbsDiffs"`.
`ibase`	The index of the COIN to use as a base comparison
`sort_table`	If TRUE, sorts by the base COIN (`ibase`) (default).
`extra_cols`	A character vector of any extra columns to include from the COIN referenced by `ibase`. For example, this could include group columns.

Value

Rank comparison table as a data frame

See Also

- `compTable()` Comparison table between two COINs

Examples

```
ASEM <- build_ASEM()
# Make a copy
ASEMAltNorm <- ASEM
# Edit .$Method
ASEMAltNorm$Method$normalise$ntype <- "borda"
# Regenerate
ASEMAltNorm <- COINr::regen(ASEMAltNorm, quietly = TRUE)
# compare
ctable <- compTableMulti(list(ASEM, ASEMAltNorm), dset = "Aggregated", isel = "Index")

# add more COINs to the list to see more cols in the table...
```

---

copeland *Copeland scores*

Description

Aggregates a data frame of indicator values into a single column using the Copeland method. This function is used inside `aggregate()`, and calls `outrankMatrix()`.

Usage

```
copeland(ind_data, w = NULL)
```

Arguments

`ind_data`	A data frame or matrix of indicator data, with observations as rows and indicators as columns. No other columns should be present (e.g. label columns).
`w`	A vector of weights, which should have length equal to `ncol(ind_data)`. Weights are relative and will be re-scaled to sum to 1. If `w` is not specified, defaults to equal weights.

Value

Numeric vector of Copeland scores.

Examples

```
# get a sample of a few indicators
ind_data <- ASEMIndData[12:16]
# calculate outranking matrix
cop_results <- copeland(ind_data)
# check output
stopifnot(length(cop_results$Scores) == nrow(ind_data))
```

corrweightscat *Scatter plot of correlations against weights*

Description

Plots correlations on the x axis and weights on the y axis. Allows a selected highlighted point and a line showing low correlation boundary. This function is intended for use inside `rew8r()`.

Usage

```
corrweightscat(
  dat,
  facet = FALSE,
  acvar = NULL,
  linesw = FALSE,
  locorval = NULL,
  hicorval = NULL
)
```

Arguments

`dat`	Data frame with first col indicator codes, second is weights, third is correlations
`facet`	Logical: if `TRUE` creates subplots.
`acvar`	Active variable to highlight (one of the indicator codes)
`linesw`	Whether to plot a vertical line showing low correlation boundary
`locorval`	x value of low correlation line
`hicorval`	x value of high correlation line

Details

Since this plot is really only intended for use inside `rew8r()` no example is provided.

Value

A scatter plot generated using plotly, also outputs event data (the clicked indicator).

See Also

- `rew8r()` Interactive app for adjusting weights and seeing effects on correlations
- `getCorr()` Get correlations between indicators/levels

`denominate` *Denominate indicator data sets*

Description

Denominates (divides) indicators by other "denominator" indicators that are either input here or were attached as `"Den_*"` columns of `IndData` when assembling the COIN. This function can work either on COINs or on data frames.

Usage

```
denominate(
  obj,
  dset = NULL,
  specby = NULL,
  denomby = NULL,
  scaledenoms = NULL,
  denominators = NULL,
  out2 = "COIN"
)
```

Arguments

`obj`	COIN object or a data frame of indicator data to be denominated. If a data frame, must include a `UnitCode` column.
`dset`	The data set to denominate (only if COIN used as input)
`specby`	Selects the source of the specifications for denomination. • If `"metadata"`, uses the denominator column in `.$metadata`. • If `"user"`, takes a character vector of denominator codes (one for each indicator, with `NA` for indicators that should not be denominated, and in the same order as the indicators).
`denomby`	Character vector specifying which denominators to use for each indicator. Only used if `specby = "user"`. For indicators with no denomination, set elements to `NA`. Elements must be column names of denominators.
`scaledenoms`	This allows the possibility to scale denominators if needed. For example, if GDP is a denominator and is measured in dollars, dividing will create very small numbers (order 1e-10 and smaller) which could cause problems with numerical precision. This should be a named list of the form e.g. `list(Den_GDP = 1e-9)`, where the name is the denominator to be scaled, and the entry is a factor to multiply the denominator values by. In the example, this would multiply GDP values by 1e-9, which (if the original values are in dollars) would scale them to billions of dollars. The list can include more than one entry, corresponding to any denominators that are present.

denominators	A data frame of denominator data. Columns should be denominator data, with column names corresponding to entries in denomby. This must also include a UnitCode column to match units (ordering is unimportant, this is done inside the function). Ensure that the unit codes correspond to the unit codes in the indicator data.
out2	Where to output the results. If "COIN" (default for COIN input), appends to updated COIN, otherwise if "df" outputs to data frame.

Details

Typically, the aim here is to convert extensive (size-related) variables into intensive variables(comparable between units of different sizes). There is also the option scaledenoms to scale denominators to avoid very small or very large numbers resulting.

This function expects that denominators$UnitCode contains all unit codes found in the data frame to be denominated. Unused unit codes (rows) in denominators will be ignored. Note that some national-level denominator data is available inside COINr at COINr::WorldDenoms.

See online documentation for further details and examples.

Value

If out2 = COIN and obj is a COIN, returns an updated COIN object, with new dataset .$Data$Denominated of denominated indicators. Otherwise returns a data frame of denominated indicator data.

See Also

- WorldDenoms A data set of some common national-level denominators.

Examples

```
# assemble ASEM COIN
ASEM <- assemble(IndData = ASEMIndData, IndMeta = ASEMIndMeta, AggMeta = ASEMAggMeta)
# denominate using specs present on assembly
ASEM <- denominate(ASEM, dset = "Raw")

# OR, use function on data frame
# Get a sample of indicator data (note must be indicators plus a "UnitCode" column)
IndData <- ASEMIndData[c("UnitCode", "Goods", "Services", "FDI")]
# Also get some denominator data
Denoms <- ASEMIndData[c("UnitCode", "Den_Pop", "Den_GDP")]
# Denominate one by the other
IndDataDenom <- denominate(IndData, denomby = c("Den_GDP", NA, "Den_Pop"), denominators = Denoms)
```

effectiveWeight *Effective weights*

Description

This calculates the effective weights of each element in the indicator hierarchy. This is useful for understanding e.g. the true weight of each indicator in the framework and is also used in `plotframework()`.

Usage

```
effectiveWeight(COIN)
```

Arguments

`COIN`	COIN object, or list with first entry is the indicator metadata, second entry is the aggregation metadata

Value

A list with effective weights, as well as a data frame with labels and parents for the sunburst plot.

Examples

```
# build ASEM COIN
ASEM <- assemble(IndData = ASEMIndData, IndMeta = ASEMIndMeta, AggMeta = ASEMAggMeta)
# get effective weights
effwts <- effectiveWeight(ASEM)
```

extractYear *Impute panel data*

Description

Given a data frame of the `IndData` format, with a `Year` column, imputes any missing data using the latest available year. This function is used inside `assemble()`.

Usage

```
extractYear(IndData, use_year, impute_latest = FALSE)
```

Arguments

IndData	A data frame of indicator data, containing a Year column and with multiple observations for each unit code.
use_year	The year of data to extract and impute.
impute_latest	Logical: if TRUE, imputes missing data points using most recent value from previous years. If FALSE (default) simply extracts the data frame as is.

Details

This expects a data frame in the IndData format, i.e. it should at least have a UnitCode column, and a Year column, as well as other columns that are to be imputed. It also presumes that there are multiple observations for each unit code, i.e. one per year. It then searches for any missing values in the target year, and replaces them with the equivalent points from previous years. It will replace using the most recently available point.

Value

A list containing:

- .$IndDataImp: An IndData format data frame from the specified year (use_year), with missing data imputed using previous years (where possible).
- .$DataYears: A data frame in the same format as IndData, where each entry shows which year each data point came from. Points where there was no missing data will have use_year, imputed points will have the corresponding year used to impute, and any points in .$IndDataImp which are still NA will be be NA.
- .$ImpTable: A data frame where each row is a point that was successfully imputed. This is a filtered and arranged version of .$DataYears that focuses only on the imputed points.
- .$NImputed: The number of imputed points.

See Also

- assemble() Assemble a COIN - this function optionally calls extractYear().
- impute() Impute data using other imputation options (not using panel data).

Examples

```
# artificial example using ASEM data
# We only have one year of data so we copy it and "pretend" that they are from different years
# First, introduce 3 NAs
dat2018 <- ASEMIndData
dat2018[2, 12] <- NA
dat2018[3, 13] <- NA
dat2018[4, 14] <- NA
# Now make copy, pretending it is the previous year
dat2017 <- ASEMIndData
dat2017$Year <- 2017
# This df still has one missing point
dat2017[4, 14] <- NA
```

```
# Finally we have a 2016 data frame where none of the previous points are missing
dat2016 <- ASEMIndData
dat2016$Year <- 2016
# We can now put them together
IndData <- rbind(dat2018, dat2017, dat2016)
# And extract the 2018 data, with missing data taken from previous years
Imp <- extractYear(IndData, 2018, impute_latest = TRUE)
# View which points have been imputed and the years of data used
Imp$ImpTable
```

`geoMean` *Weighted geometric mean*

Description

Weighted geometric mean of a vector. `NA` are skipped by default. This function is used inside `aggregate()`.

Usage

```
geoMean(x, w = NULL)
```

Arguments

`x`	A numeric vector of positive values.
`w`	A vector of weights, which should have length equal to `length(x)`. Weights are relative and will be re-scaled to sum to 1. If `w` is not specified, defaults to equal weights.

Value

The geometric mean, as a numeric value.

Examples

```
# a vector of values
x <- 1:10
# a vector of weights
w <- runif(10)
# weighted geometric mean
geoMean(x,w)
```

`geoMean_rescaled` *Rescaled weighted geometric mean*

Description

NOTE this function is not really in use but is kept here for the moment. Not sure it is very useful.

Usage

```
geoMean_rescaled(x, w = NULL)
```

Arguments

`x`	A numeric vector of positive values.
`w`	A vector of weights, which should have length equal to `length(x)`. Weights are relative and will be re-scaled to sum to 1. If `w` is not specified, defaults to equal weights.

Details

Weighted geometric mean of a vector. Here, any zero or negative values are automatically dealt with by re-scaling the data to be all positive, i.e. it shifts so that the minimum is equal to 0.1.

Note that this could be better achieved by normalising first. However, following default normalisation between 0 and 100, this function offers a quick way to test the effect of a geometric mean, for example in a sensitivity analysis, and avoids bugs arising.

Value

Rescaled weighted geometric mean, as a numeric value.

Examples

```
# a vector of values
x <- 1:10
# a vector of weights
w <- runif(10)
# rescaled weighted geometric mean
geoMean_rescaled(x,w)
```

getCorr *Get different types of correlation matrices*

Description

Helper function for getting correlations between indicators. This retrieves subsets of correlation matrices between different aggregation levels, in different formats.

Usage

```
getCorr(
  COIN,
  dset,
  icodes = NULL,
  aglevs = NULL,
  cortype = "pearson",
  pval = 0.05,
  withparent = TRUE,
  grouplev = NULL
)
```

Arguments

COIN	The COIN object
dset	The target data set
icodes	An optional list of character vectors where the first entry specifies the indicator/aggregate codes to correlate against the second entry (also a specification of indicator/aggregate codes).
aglevs	The aggregation levels to take the two groups of indicators from. See getIn() for details. Defaults to indicator level.
cortype	The type of correlation to calculate, either "pearson", "spearman", or "kendall".
pval	The significance level for including correlations. Correlations with $p > pval$ will be returned as NA. Default 0.05. Set to 0 to disable this.
withparent	If TRUE, and aglev[1] != aglev[2], will only return correlations of each row with its parent.
grouplev	The aggregation level to group correlations by if aglev[1] == aglev[2]. By default, groups correlations into the aggregation level above. Set to 0 to disable grouping and return the full matrix.

Details

Note that this function can only call correlations within the same data set (i.e. only one data set in .$Data).

Value

A data frame of correlation values in long format. Correlations with $p > pval$ will be returned as NA.

See Also

- plotCorr() Plot correlation matrices of indicator subsets

Examples

```
# build ASEM COIN
ASEM <- assemble(IndData = ASEMIndData, IndMeta = ASEMIndMeta, AggMeta = ASEMAggMeta)
# correlations of indicators in Political pillar
corrs <- getCorr(ASEM, dset = "Raw", icodes = "Political", aglevs = 1)
```

---

getCronbach *Cronbach's alpha*

Description

Calculates Cronbach's alpha, a measure of statistical reliability. Cronbach's alpha is a simple measure of "consistency" of a data set, where a high value implies higher reliability/consistency.

Usage

```
getCronbach(
  COIN,
  dset = "Raw",
  icodes = NULL,
  aglev = NULL,
  use = "pairwise.complete.obs"
)
```

Arguments

COIN	A COIN or a data frame containing only numerical columns of data.
dset	The data set to check the consistency of.
icodes	Indicator codes if a subset of dset is requested
aglev	The aggregation level to take icodes from (see getIn() for details)
use	Argument to pass to stats::cor to calculate the covariance matrix. Default "pairwise.complete.obs".

Details

This function simply returns Cronbach's alpha. If you want a lot more details on reliability, the 'psych' package has a much more detailed analysis.

Value

Cronbach alpha as a numerical value.

Examples

```
# build ASEM COIN
ASEM <- assemble(IndData = ASEMIndData, IndMeta = ASEMIndMeta, AggMeta = ASEMAggMeta)
# get Cronbach of indicators in Physical pillar
getCronbach(ASEM, dset = "Raw", icodes = "Physical", aglev = 1)
```

---

getIn *Get subsets of indicator data*

Description

This function does a number of things that are useful for many COINr functions and operations. First, it checks to see what kind of input object is input. Then, it selects the indicator data according to the specs supplied.

Usage

```
getIn(obj, dset = "Raw", icodes = NULL, aglev = NULL, justnumeric = TRUE)
```

Arguments

obj	An input object. The function can handle either the COIN object, or a data frame. The data frame should have each column as an indicator, and optional columns `UnitCode` and `UnitName` which specify the code (or name) of each unit. Any columns except these latter two will be treated as indicators. Any other type of object will return an error.
dset	If input is a COIN object, this specifies which data set in `.$Data` to use.
icodes	An optional character vector of indicator codes to subset the indicator data. Usefully, can also refer to an aggregation group name, and data will be subsetted accordingly. NOTE does not work with multiple aggregate group names.
aglev	The aggregation level to take indicator data from. Integer from 1 (indicator level) to N (top aggregation level, typically the index).
justnumeric	Logical: if `TRUE`, removes any non-numeric columns from `ind_data_only`. Otherwise keeps all except those.

Details

For example, specifying `dset = "Raw"` and `icodes = c("Ind1","Ind5")`, it will return the indicator columns named "Ind1" and "Ind5" (if they exist), in the format described below. `icodes` can be indicators or aggregation groups, and can call multiple groups.

You can also specify which aggregation level to target, using the `aglev` argument. See examples below, and in particular the COINr online documentation.

`getIn()` is used by many COINr functions for plotting, accessing and reporting subsets of indicator data.

Value

A list with the following entries:

- `.$IndCodes` The indicator codes
- `.$IndNames` The indicator names (if a COIN object is input)
- `.$ind_data` A data frame of indicator data, according to the input specifications, including any unit codes, names and groups
- `.$ind_data_only` A data frame, as above, but without unit codes, names, groups.
- `.$UnitCodes` Unit codes of selected data set.
- `.$otype` Object type (a string: either `"COINobj"` or `"df"`).

Examples

```
# assemble ASEM COIN
ASEM <- assemble(IndData = ASEMIndData, IndMeta = ASEMIndMeta, AggMeta = ASEMAggMeta)
# get indicator data from Social pillar
SocData <- getIn(ASEM, dset = "Raw", icodes = "Social", aglev = 1)
# Indicator codes
SocData$IndCodes
# Indicator data (no other columns)
SocData$ind_data_only
```

---

getPCA *Perform PCA on a COIN*

Description

Performs Principle Component Analysis (PCA) on a specified data set and subset of indicators or aggregation groups. Returns weights corresponding to the first principal component, i.e the weights that maximise the variance explained by the linear combination of indicators.

Usage

```
getPCA(
  COIN,
  dset = "Raw",
  icodes = NULL,
  aglev = NULL,
  nowarnings = FALSE,
  out2 = "COIN"
)
```

Arguments

COIN	An input object. The function can handle either the COIN object, or a data frame. The data frame should have each column as an indicator, and an optional column "UnitCode" which specifies the code (or name) of each unit. Any other type of object will return an error.
dset	If input is a COIN object, this specifies which data set in .$Data to use.
icodes	An optional character vector of indicator codes to subset the indicator data. Usefully, can also refer to an aggregation group name, and data will be sub-setted accordingly. NOTE does not work with multiple aggregate group names.
aglev	The aggregation level to take indicator data from. Integer from 1 (indicator level) to N (top aggregation level, typically the index).
nowarnings	If FALSE (default), will give warnings where missing data are found. Set to TRUE to suppress these warnings.
out2	If the input is a COIN object, this controls where to send the output. If "COIN", it sends the results to the COIN object, otherwise if "list", outputs to a separate list.

Details

Note that getPCA() is simply a quick wrapper for stats::prcomp() which makes PCA on COINs quicker. See COINr online documentation for more details and examples.

Value

If out2 = "COIN", results are appended to the COIN object. Specifically:

- A new set of PCA weights is added to .$Parameters$Weights
- A list is added to .$Analysis containing PCA weights (loadings) of the first principle component, and the output of stats::prcomp, for each aggregation group found in the targeted level. If out2 = "list" the same outputs are contained in a list.

See Also

- stats::prcomp Principle component analysis

Examples

```
# build ASEM COIN
ASEM <- assemble(IndData = ASEMIndData, IndMeta = ASEMIndMeta,
AggMeta = ASEMAggMeta)
# get PCA results for pillar groups inside "Sust" (sustainability) sub-index
# (warnings about missing data are suppressed)
PCAres <- getPCA(ASEM, dset = "Raw", icodes = "Sust",
aglev = 1, nowarnings = TRUE, out2 = "list")
# summarise PCA results for Social pillar
summary(PCAres$PCAresults$Social$PCAres)
```

---

getResults *Results summary tables*

Description

Generates fast results tables, either attached to the COIN or as a data frame.

Usage

```
getResults(
  COIN,
  tab_type = "Summary",
  use = "scores",
  order_by = NULL,
  nround = 2,
  out2 = "df"
)
```

Arguments

COIN	The COIN object, or a data frame of indicator data
tab_type	The type of table to generate. Either "Summary", "Aggregates", "Full", or "FullWithDenoms".
use	Either "scores" (default) or "ranks".
order_by	A code of the indicator or aggregate to sort the table by. If not specified, defaults to the highest aggregate level, i.e. the index in most cases.
nround	The number of decimal places to round numerical values to. Defaults to 2.
out2	If "df", outputs a data frame (tibble). Else if "COIN" attaches to .$Results in an updated COIN.

Details

Although results are available in a COIN in .$Data, the format makes it difficult to quickly present results. This function generates results tables that are suitable for immediate presentation, i.e. sorted by index or other indicators, and only including relevant columns. Scores are also rounded by default, and there is the option to present scores or ranks.

Value

If `out2 = "df"`, the results table is returned as a data frame. If `out2 = "COIN"`, this function returns an updated COIN with the results table attached to `.$Results`.

See Also

- `resultsDash()` Interactive results dashboard
- `coin2Excel()` Export results to Excel

Examples

```
# build ASEM COIN up to aggregation
ASEM <- build_ASEM()
# results table of scores for index and aggregates (excluding indicator scores)
dfResults <- getResults(ASEM, tab_type = "Aggregates", out2 = "df")
```

---

getStats *Get table of indicator statistics for any data set*

Description

Takes a COIN or data frame and returns a table of statistics for each column, including max, min, median, mean, standard deviation, kurtosis, etc. Flags indicators with possible outliers, and checks for collinearity with other indicators and denominators (if any). Also checks number of unique values and percentage of zeros. Also returns correlation matrices and a table of outliers, as a list.

Usage

```
getStats(
  COIN,
  icodes = NULL,
  dset = "Raw",
  out2 = "COIN",
  cortype = "pearson",
  t_skew = 2,
  t_kurt = 3.5,
  t_colin = 0.9,
  t_denom = 0.7,
  t_missing = 65,
  IQR_coef = 1.5
)
```

Arguments

COIN	A COIN object or data frame of indicator data
icodes	A character vector of indicator names to analyse. Defaults to all indicators.
dset	The data set to analyse.
out2	Where to output the results: if "COIN" (default), appends to the COIN, otherwise if "list", outputs to a separate list.
cortype	The type of correlation to calculate, either "pearson", "spearman", or "kendall". See stats::cor.
t_skew	Skewness threshold.
t_kurt	Kurtosis threshold.
t_colin	Collinearity threshold (absolute value of correlation).
t_denom	High correlation with denominator threshold.
t_missing	Missing data threshold, in percent.
IQR_coef	Interquartile range coefficient, used for identifying outliers.

Value

If out2 = "COIN" (default), results are appended to the COIN in .$Analysis, otherwise if out2 = "list", outputs to a separate list. In both cases, the result is a list containing:

- A data frame of statistics for each indicator column
- A data frame indicating which points may be considered outliers according to the interquartile range
- A data frame of correlations between indicators
- A data frame of correlations between indicators and any denominators present in .$Input$Denominators

Examples

```
# build ASEM COIN
ASEM <- assemble(IndData = ASEMIndData, IndMeta = ASEMIndMeta, AggMeta = ASEMAggMeta)
# get list of stats from raw data set
stat_list <- getStats(ASEM, dset = "Raw", out2 = "list")
```

---

getStrengthNWeak *Generate strengths and weaknesses for a specified unit*

Description

Generates a table of strengths and weaknesses for a selected unit, based on ranks.

Usage

```
getStrengthNWeak(COIN, usel = NULL, topN = 5, bottomN = 5, withcodes = TRUE)
```

Arguments

COIN	A COIN
usel	A selected unit code
topN	The top N indicators to report
bottomN	The bottom N indicators to report
withcodes	If TRUE (default), also includes a column of indicator codes. Setting to FALSE may be more useful in generating reports, where codes are not helpful.

Details

This currently only works at the indicator level. Indicators with NA values for the selected unit are ignored. Strengths and weaknesses mean the top N-ranked indicators for the selected unit. Effectively, this takes the rank that the selected unit has in each indicator, sorts the ranks, and takes the top N highest and lowest.

NOTE: this function currently requires data to be aggregated. This restriction may be relaxed at some point.

Value

A list containing a data frame `.$Strengths`, and a data frame `.$Weaknesses`. Each data frame has columns with indicator code, name, rank and value (for the selected unit).

See Also

- `getUnitReport()` Automatic unit report as html, pdf or Word
- `getUnitSummary()` Summary of scores for a given unit

Examples

```
# build ASEM COIN up to aggregation
ASEM <- build_ASEM()
# generate top 5 strengths and weaknesses for GBR
getStrengthNWeak(ASEM, usel = "GBR")
```

---

getUnitReport *Generate unit report*

Description

Generates a scorecard for a given unit using an R Markdown template.

Usage

```
getUnitReport(
  COIN,
  usel,
  out_type = ".html",
  outdir = NULL,
  rmd_template = NULL
)
```

Arguments

COIN	A COIN
usel	A selected unit code, or a character vector of unit codes (for multiple reports).
out_type	A string specifying the output type. Can be either ".docx" (Word), ".pdf" or ".html". IMPORTANT: if the template includes interactive plots (e.g. the iplot() functions from COINr), writing to .docx or .pdf will not work *unless* you have installed the webshot package. To do this, run: install.packages("webshot") webshot::install_phantomjs()
outdir	Character string specifying the output directory (defaults to current working directory).
rmd_template	A character string specifying the full file path to an R Markdown template which is used to generate the report. If this is not specified, defaults to COINr's inbuilt template example.

Details

Most likely you will want to customise the template which can be found in the COINr installed package directory under /UnitReport. Currently, a few examples are given, such as some charts and basic summary statistics.

This function will render the unit report to either pdf, html or word doc. As mentioned below, if you have HTML widgets such as interactive plotly plots, or COINr iplot() functions, you will need to install the webshot package to be able to render to pdf or word formats.

To customise the template, copy the .rmd template found in /UnitReport and alter it, then point the rmd_template argument to your new template.

Note that this function is particularly useful for generating a large number of reports, e.g. we can generate reports for all units at once using a for loop, purrr::map or apply() or similar.

Value

Markdown document rendered to HTML, pdf or Word. This function requires Pandoc to be installed. If Pandoc is not found, then it returns a warning and a printed message (string).

Examples

```
# build ASEM COIN up to aggregation
ASEM <- build_ASEM()
# Generate a unit report for NZ
```

```
# This is written to the temporary directory to avoid polluting other directories
# during automated testing.
# It will be deleted at the end of the R session.
# Normally you would set the directory to somewhere else to save the resulting files
getUnitReport(ASEM, usel = "NZL", out_type = ".html", outdir = tempdir())
# You can find this file in the temporary directory:
print(tempdir())
# We will now delete the file to keep things tidy in testing
unlink(paste0(tempdir(),"\\NZL_report.html"))
```

`getUnitSummary` *Generate unit summary table*

Description

Generates a summary table for a single unit. This is mostly useful in unit reports.

Usage

```
getUnitSummary(COIN, usel, aglevs)
```

Arguments

`COIN`	A COIN
`usel`	A selected unit code
`aglevs`	The aggregation levels to display results from.

Details

This returns the scores and ranks for each indicator/aggregate as specified in `aglevs`. It orders the table so that the highest aggregation levels are first. This means that if the index level is included, it will be first.

Value

A summary table as a data frame, containing scores and ranks for specified indicators/aggregates.

See Also

- `getUnitReport()` Automatic unit report as html, pdf or Word
- `getStrengthNWeak()` Top N-ranking indicators for a given unit

Examples

```
# build ASEM COIN up to aggregation
ASEM <- build_ASEM()
# generate unit summary for NZ - index and sub-indexes only
getUnitSummary(ASEM, usel ="NZL", aglevs = c(4,3))
```

harMean *Weighted harmonic mean*

Description

Weighted harmonic mean of a vector. NA are skipped by default. This function is used inside aggregate().

Usage

```
harMean(x, w = NULL)
```

Arguments

x	A numeric vector of positive values.
w	A vector of weights, which should have length equal to length(x). Weights are relative and will be re-scaled to sum to 1. If w is not specified, defaults to equal weights.

Value

Weighted harmonic mean, as a numeric value.

Examples

```
# a vector of values
x <- 1:10
# a vector of weights
w <- runif(10)
# weighted harmonic mean
harMean(x,w)
```

hicorrSP *Highly correlated indicators in the same aggregation group*

Description

This returns a data frame of any highly correlated indicators within the same aggregation group. The level of the aggregation group can be controlled by the grouplev argument.

Usage

```
hicorrSP(
  COIN,
  dset = "Normalised",
  hicorval = 0.9,
  cortype = "pearson",
  grouplev = NULL
)
```

Arguments

COIN	Data frame with first col indicator codes, second is weights, third is correlations
dset	The data set to use for correlations.
hicorval	A threshold to flag high correlation. Default 0.9.
cortype	The type of correlation, either "pearson" (default), "spearman" or "kendall". See stats::cor.
grouplev	The level to group indicators in. E.g. if grouplev = 2 it will look for high correlations between indicators that belong to the same group in Level 2.

Value

A data frame with one entry for every indicator pair that is highly correlated within the same group, at the specified level. Pairs are only reported once, i.e. only uses the upper triangle of the correlation matrix.

See Also

- rew8r() Interactive app for adjusting weights and seeing effects on correlations
- getCorr() Get correlations between indicators/levels

Examples

```
# Assemble ASEM COIN
ASEM <- assemble(IndData = ASEMIndData, IndMeta = ASEMIndMeta, AggMeta = ASEMAggMeta)
# check for any within-pillar correlations of > 0.7
hicorrSP(ASEM, dset = "Raw", hicorval = 0.7, , grouplev = 2)
```

---

impute *Impute missing data*

Description

Imputation of missing data data sets using a variety of methods (see imtype). This also includes the possibility of imputing by grouping variables, i.e. columns of IndData that are prefaced by "Group_".

Usage

```
impute(
  COIN,
  imtype = NULL,
  dset = NULL,
  groupvar = NULL,
  EMaglev = NULL,
  out2 = "COIN"
)
```

Arguments

COIN	A COIN or a data frame
imtype	The type of imputation method. Either: • "agg_mean" (the mean of normalised indicators inside the aggregation group), • "agg_median" (the median of normalised indicators inside the aggregation group), • "ind_mean" (the mean of all the other units in the indicator), • "ind_median" (the median of all the other units in the indicator), • "indgroup_mean" (the mean of all the other units in the indicator, in the same group), • "indgroup_median" (the median of all the other units in the indicator, in the same group), • "EM" (expectation maximisation algorithm via AMELIA package, currently without bootstrapping) • "none" (no imputation, returns original data set)
dset	The data set in .$Data to impute
groupvar	The name of the column to use for by-group imputation. Only applies when imtype is set to a group option.
EMaglev	The aggregation level to use if imtype = "EM".
out2	Where to output the imputed data frame. If "COIN" (default for COIN input), creates a new data set .$Data$Imputed. Otherwise if "df" outputs directly to a data frame.

Details

See online documentation for further details and examples.

Value

If out2 = "COIN" (default for COIN input), creates a new data set .$Data$Imputed. Otherwise if out2 = "df" outputs directly to a data frame.

Examples

```
# assemble the COIN
ASEM <- assemble(IndData = ASEMIndData, IndMeta = ASEMIndMeta, AggMeta = ASEMAggMeta)
# Check how many missing data points are in raw data set
sum(is.na(ASEM$Data$Raw))
# impute data using Asia/Europe group mean
DataImputed <- impute(ASEM, dset = "Raw", imtype = "indgroup_mean", groupvar = "Group_EurAsia",
out2 = "df")
# See how many missing data points we have in the imputed data
sum(is.na(DataImputed))
# check no missing data
stopifnot(sum(is.na(DataImputed))==0)
```

---

indChange *Add and remove indicators*

Description

A shortcut function to add and remove indicators. This will make the relevant changes and recalculate the index if asked. Adding and removing is done relative to the current set of indicators used in calculating the index results. Any indicators that are added must of course be present in `.$Input$Original` (in both `IndData` and `IndMeta`).

Usage

```
indChange(COIN, add = NULL, drop = NULL, regen = FALSE)
```

Arguments

`COIN`	COIN object
`add`	A character vector of indicator codes to add (must be present in the original input data)
`drop`	A character vector of indicator codes to remove (must be present in the original input data)
`regen`	Logical (default): if `TRUE`, automatically regenerates the results based on the new specs Otherwise, just updates the `.$Method$assemble` parameters. This latter might be useful if you want to Make other changes before re-running using the `regen()` function.

Value

An updated COIN, with regenerated results if `regen = TRUE`.

See Also

- regen() regenerate a COIN
- compTable() compare two different COINs
- compTableMulti() compare multiple COINs

Examples

```
# build ASEM example
ASEM <- build_ASEM()
# remove one indicator and regenerate results
ASEM2 <- indChange(ASEM, drop = "UNVote", regen = TRUE)
# compare the differences
CT <- compTable(ASEM, ASEM2, dset = "Aggregated", isel = "Index")
```

---

indDash *Indicator visualisation dashboard*

Description

Generates an interactive visualisation of one or two indicators at a time. Requires Shiny and an active R session. This dashboard is useful for quickly exploring indicator data, and seeing e.g. an untreated indicator distribution against its treated equivalent.

Usage

```
indDash(COIN)
```

Arguments

COIN	The COIN object

Details

This function requires an interactive R session. Otherwise it will simply generate a message to that effect.

Value

Interactive app (if running an interactive R session). This app is purely exploratory and does not return anything back to R.

Examples

```
# Only run in interactive mode
if(interactive()){
# build ASEM COIN
ASEM <- build_ASEM()
# view dashboard
indDash(ASEM)
}
```

---

iplotBar *Bar chart*

Description

Generates an interactive bar chart. This function is simply a wrapper for the **plotly** bar chart function, but accesses the COIN object to get the relevant indicator. Also has click event data for Shiny. Allows construction of stacked bar charts which show underlying components (for aggregated data only), and plots of only specified groups.

Usage

```
iplotBar(
  COIN,
  dset = "Raw",
  isel = NULL,
  usel = NULL,
  aglev = NULL,
  stack_children = FALSE,
  from_group = NULL
)
```

Arguments

COIN	The COIN object, or a data frame of indicator data.
dset	The data set to plot.
isel	The selected indicator code or aggregate (does not support multiple indicators)
usel	A character vector of unit codes to highlight on the bar chart (optional)
aglev	The aggregation level to collect the indicator data from (this needs to be specified)
stack_children	If TRUE, produces a stacked bar chart with any children of isel. In this case, usel is ignored. This only works if dset = "Aggregated" and aglev > 1.
from_group	Filters the bar chart to a specified group using a group column that is present in the specified data set. Specified as list(group_variable = selected_group).

Value

Interactive bar chart generated by plotly.

See Also

- iplotMap() bar chart of indicator or aggregate
- resultsDash() interactive dashboard of indicator data

Examples

```
# assemble ASEM COIN
ASEM <- assemble(IndData = ASEMIndData, IndMeta = ASEMIndMeta, AggMeta = ASEMAggMeta)
# plot Flights indicator
iplotBar(ASEM, dset = "Raw", isel = "Flights", aglev = 1)
```

---

iplotCorr *Correlation heatmap*

Description

Plots an interactive heatmap of a correlation matrix. Currently this only works with the aggregated data set, i.e. you need to have aggregated the data first before using this.

Usage

```
iplotCorr(
  COIN,
  aglevs = NULL,
  insig = FALSE,
  levs = TRUE,
  grouprects = TRUE,
  flagcolours = TRUE,
  corthresh = NULL,
  showvals = TRUE,
  cortype = "pearson",
  useweights = NULL
)
```

Arguments

COIN	The COIN object
aglevs	A two length vector specifying which level to plot against which level. E.g. c(2,4) for COIN plots sub-pillars against sub-indexes. If NULL, plots everything against everything.
insig	Logical: if TRUE, all correlation values are plotted; if FALSE (default), does not plot insignificant correlations.

levs	Logical: if TRUE, plots lines showing the division between different levels. Only works if aglevs = NULL.
grouprects	Logical: if TRUE, plots rectangles showing aggregation groups.
flagcolours	If TRUE uses a discrete colour scale specified by corthresh. Otherwise uses a continuous colour map.
corthresh	A named list specifying the colour thresholds to use if flagcolours = TRUE. Entries should specify correlation thresholds and can specify any of clow, cmid and chi. Anything below clow will be coloured red. Anything between clow and cmid will be grey. Anything between cmid and chigh will be blue. Anything above chigh will be green. Default is list(clow = -0.4, cmid = 0.4, chigh = 0.85). You can specify a subset of these and the others will revert to defaults.
showvals	Logical: if TRUE, overlays correlation values on each square.
cortype	The type of correlation: either "pearson" (default), "spearman" or "kendall". See stats::cor.
useweights	An optional list of weights to use (this is used mainly in the rew8r() app).

Details

This is a slightly involved wrapper for **plotly**. It allows plotting any level against any other, and outputs correlation heat maps as HTML widgets. It has some flexibility regarding grouping of indicators, colouring, treatment of insignificant correlations, and the correlation type. Explore the arguments and see.

Value

A **plotly** correlation map (figure).

See Also

- plotCorr() Static correlation heat maps
- rew8r() Interactive app for adjusting weights and seeing effects on correlations
- getCorr() Get correlations between indicators/levels

Examples

```
# build ASEM COIN up to aggregation
ASEM <- build_ASEM()
# correlation heatmap of pillars against sub-indexes
iplotCorr(ASEM, aglevs = c(2,3))
```

`iplotIndDist` *Interactive indicator distribution plots*

Description

Generates a JavaScript distribution plot of a single indicator, using **plotly**. Plot can be embedded e.g. in HTML documents, websites, etc, or used for more interactive data exploration. This only plots one indicator at a time - for multiple plots you can use `plotIndDist()`.

Usage

```
iplotIndDist(
  COIN,
  dset = "Raw",
  icodes = NULL,
  ptype = "Violin",
  aglev = 1,
  axlims = NULL
)
```

Arguments

`COIN`	The COIN object, or a data frame of indicator data
`dset`	The source data set to use for indicator data (if input is COIN object)
`icodes`	A character vector of a single indicator name or aggregate name to plot.
`ptype`	The type of plot to produce. Currently supports `"Violin"` and `"Histogram"`.
`aglev`	The aggregation level to extract the indicator data from. Defaults to indicator level (1)
`axlims`	Optional parameter specifying axis limits. Useful mainly for matching with another plot.

Value

Plots generated with **plotly**. These can be edited further with **plotly** commands.

Examples

```
# build ASEM COIN
ASEM <- assemble(IndData = ASEMIndData, IndMeta = ASEMIndMeta, AggMeta = ASEMAggMeta)
# plot renewable energy indicator
iplotIndDist(ASEM, "Raw", "Renew", ptype = "Violin")
```

`iplotIndDist2` *Interactive indicator distribution plots for two indicators simultaneously*

Description

Generates a JavaScript distribution plot of two indicators, using Plotly. Plot can be embedded e.g. in HTML documents, websites, etc, or used for more interactive data exploration.

Usage

```
iplotIndDist2(
  COIN,
  dsets = "Raw",
  icodes = NULL,
  ptype = "Scatter",
  aglevs = 1
)
```

Arguments

`COIN`	The COIN, or a data frame of indicator data
`dsets`	The source data sets to use for indicator data (if input is COIN object). If the source data sets are the same, this can be a single character string, otherwise, a character vector, e.g. `c("Raw","Treated")`.
`icodes`	A character vector of two indicator codes to plot (corresponding to the two `dsets` specified)
`ptype`	The type of plot to produce. Currently supports `"Histogram"` and `"Scatter"`.
`aglevs`	The aggregation level to extract the indicator data from. Defaults to indicator level (1). This also can be specified as a vector if the two indicators are from different levels.

Value

Plots generated with **plotly**. These can be edited further with **plotly** commands.

Examples

```
# build ASEM COIN
ASEM <- assemble(IndData = ASEMIndData, IndMeta = ASEMIndMeta, AggMeta = ASEMAggMeta)
# plot CO2 against renewable energy indicator
iplotIndDist2(ASEM, dsets = "Raw", icodes = c("Renew", "CO2"))
```

`iplotMap` *Choropleth map*

Description

Generates an interactive choropleth map of specified indicator data. Only works on national level data (i.e. one point per country), with countries labelled by ISO alpha-3 codes. This function is simply a quick wrapper for the **plotly** choropleth map function.

Usage

```
iplotMap(COIN, dset = "Raw", isel)
```

Arguments

`COIN`	The COIN object, or a data frame of indicator data.
`dset`	The data set to plot.
`isel`	The selected indicator code or aggregate

Value

Interactive map generated by plotly.

See Also

- `iplotBar()` bar chart of indicator or aggregate
- `resultsDash()` interactive dashboard of indicator data

Examples

```
# assemble ASEM COIN
ASEM <- assemble(IndData = ASEMIndData, IndMeta = ASEMIndMeta, AggMeta = ASEMAggMeta)
# map CO2 indicator
iplotMap(ASEM, dset = "Raw", isel = "CO2")
```

iplotRadar *Radar chart*

Description

Generates an interactive radar chart for a specified unit or set of units.

Usage

```
iplotRadar(
  COIN,
  dset = "Raw",
  usel = NULL,
  aglev = NULL,
  isel = NULL,
  addstat = "none",
  statgroup = NULL,
  statgroup_name = NULL
)
```

Arguments

COIN	The COIN object, or a data frame of indicator data.
dset	The data set to use in the table
usel	Character vector of unit code(s) to plot data from
aglev	The selected aggregation level to take indicator data from, where 1 is the base indicator level, and 2, 3 etc are higher aggregation levels
isel	The indicator or aggregation code(s) to plot
addstat	Adds the statistic of the scores in each dimension as a separate trace. If "mean" adds the overall mean for each dimension/indicator. If "median" adds the overall median. If "groupmean" or "groupmedian", adds the group mean or median respectively of the first unit specified in usel, using the group specified by statgroup. Default "none", i.e. no extra trace. Using a group mean or median won't make sense unless all of selected units are from the same group.
statgroup	A grouping variable (must be present in dset) if addstat = "groupmean" or "groupmedian"
statgroup_name	An optional name to display for statgroup. In the legend this will appear as e.g. "statgroup_name group mean". Defaults to statgroup.

Details

This function uses **plotly** to generate a radar chart for showing one or more units, compared using a specified set of indicators. Optionally, you can add mean/median or group mean/median as an extra trace. The point being to show how a particular unit compares to its peers.

Value

Interactive radar chart generated using plotly.

See Also

- `resultsDash()` Interactive results dashboard.

Examples

```
# build ASEM COIN up to aggregation
ASEM <- build_ASEM()
# radar chart of Austria vs. China in Political indicators
iplotRadar(ASEM, dset = "Aggregated", usel = c("AUT", "CHN"), isel = "Political", aglev = 1)
```

---

iplotTable *Results table*

Description

Generates an interactive table of data. For use in e.g. Shiny or HTML documents.

Usage

```
iplotTable(
  COIN,
  dset = "Raw",
  isel = NULL,
  aglev = NULL,
  nround = 1,
  extra_cols = FALSE
)
```

Arguments

`COIN`	The COIN object, or a data frame of indicator data.
`dset`	The data set to use in the table
`isel`	The selected indicator codes (default all in `dset`)
`aglev`	The aggregation level to select data from
`nround`	The number of decimal places to round scores to (default 1).
`extra_cols`	If `FALSE` (default), excludes group columns and similar. If `TRUE`, includes them.

Details

This function is a wrapper for the **reactable** package and offers a fast way to make interactive tables. It also applies conditional formatting (colouring by cell value), and sorts by the first column by default. Like other **COINr** functions, it can target subsets of indicators.

Value

An interactive table generated by reactable.

See Also

- `colourTable()` Conditionally-formatted table for any data frame
- `resultsDash()` interactive dashboard of indicator data
- `getResults()` results summary tables

Examples

```
# assemble ASEM COIN
ASEM <- assemble(IndData = ASEMIndData, IndMeta = ASEMIndMeta, AggMeta = ASEMAggMeta)
# table of indicators in "Poltical" pillar
iplotTable(ASEM, dset = "Raw", isel = "Political", aglev = 1)
```

`is.coin` *Check if an object is a COIN*

Description

Returns TRUE if an input object is a COIN, otherwise FALSE if not.

Usage

```
is.coin(obj)
```

Arguments

`obj`	An input object to test

Value

Logical: TRUE if input is a COIN, otherwise FALSE

See Also

- `getIn()` Get subset of indicator data from either a COIN or data frame.
- `assemble()` Assemble a COIN from indicator data and metadata

Examples

```
# build the ASEM COIN
ASEM <- assemble(IndData = ASEMIndData, IndMeta = ASEMIndMeta, AggMeta = ASEMAggMeta)
# check class
stopifnot(is.coin(ASEM))
```

`loggish` *Log-type transformation of a vector*

Description

This applies various simple transformations, to be used by the `treat()` function. This function is probably not very useful on its own because it requires `params`, a list of parameters which are used to output the type and status of treatment applied.

Usage

```
loggish(x, ltype, params = NULL)
```

Arguments

`x`	A vector or column of data to transform
`ltype`	The type of log transformation - see `deflog` in `treat()`.
`params`	Some extra parameters to pass. These parameters mostly concern internal messages for `treat()` and this can be left unspecified unless `ltype == "boxcox"`, in which case there should be a parameter `params$boxlam` specified (see `BoxCox()`). However, if you wish to use a Box Cox transformation, it is better to use `BoxCox()` directly.

Value

A list with

- `.$x` is the transformed vector of data
- `.$Flag` is a flag of the type of treatment specified (used inside `treat()`)
- `.$Treatment` the treatment applied (used inside `treat()`)
- `.$TreatSpec` the treatment specified (used inside `treat()`)

See Also

- `treat()` Outlier treatment

Examples

```
# get a column of data with outliers
x <- ASEMIndData$Tariff
# apply a GII transformation
xdash <- loggish(x, ltype = "GIIlog")
# plot one against the other
plot(x, xdash$x)
```

names2Codes *Generate short codes from long names*

Description

Given a character vector of long names (probably with spaces), generates short codes. Intended for use when importing from the COIN Tool.

Usage

```
names2Codes(cvec, maxword = 2, maxlet = 4)
```

Arguments

cvec	A character vector of names
maxword	The maximum number of words to use in building a short name (default 2)
maxlet	The number of letters to take from each word (default 4)

Value

A corresponding character vector, but with short codes, and no duplicates.

See Also

- COINToolIn() Import data from the COIN Tool (Excel).

Examples

```
# generate codes for indicators in the ASEM data set (first five only)
names2Codes(ASEMIndMeta$IndName[1:5], maxlet = 3)
```

noisyWeights *Noisy replications of weights*

Description

Given a set of weights, this function returns multiple replicates of the weights, with added noise. This is intended for use in uncertainty and sensitivity analysis.

Usage

```
noisyWeights(w, noise_specs, Nrep)
```

Arguments

w	A data frame of weights, in the format found in .$Parameters$Weights.
noise_specs	a data frame with columns: • AgLevel: The aggregation level to apply noise to • NoiseFactor: The size of the perturbation: setting e.g. 0.2 perturbs by +/- 20% of nominal values.
Nrep	The number of weight replications to generate.

Details

Weights are expected to be in a long data frame format with columns Aglevel, Code and Weight, as used inside COINs.

Noise is added using the noise_specs argument, which is specified by a data frame with columns AgLevel and NoiseFactor. The aggregation level refers to number of the aggregation level to target while the NoiseFactor refers to the size of the perturbation. If e.g. a row is AgLevel = 1 and NoiseFactor = 0.2, this will allow the weights in aggregation level 1 to deviate by +/- 20% of their nominal values (the values in w).

Value

A list of Nrep sets of weights (data frames).

See Also

- sensitivity() Perform global sensitivity or uncertainty analysis on a COIN

Examples

```
# build ASEM COIN
ASEM <- assemble(IndData = ASEMIndData, IndMeta = ASEMIndMeta,
AggMeta = ASEMAggMeta)

# generate 2 sets of weights based on original ASEM weights,
# perturbed by +/-20% only at indicator level
wlist <- noisyWeights(ASEM$Parameters$Weights$Original,
noise_specs = data.frame(AgLevel = 1, NoiseFactor = 0.2), Nrep = 2)
```

---

normalise *Normalise indicator data sets*

Description

A dataset of indicators is normalised using one of several methods. This function also supports custom normalisation.

Usage

```
normalise(
  COIN,
  ntype = "minmax",
  npara = NULL,
  dset = NULL,
  directions = NULL,
  individual = NULL,
  indiv_only = NULL,
  out2 = NULL
)
```

Arguments

COIN	Either the COIN object, or a data frame of indicator data
ntype	The type of normalisation method. Either "minmax", "zscore", "scaled", "goalposts", "rank", "borda", "prank", "fracmax", "dist2targ", "dist2ref", "dist2max", "custom" or "none". See the online documentation.
npara	Supporting object for ntype. This is a list of the form list(ntype = parameters_for_ntype). So, if ntype = "minmax", npara could be list(minmax = c(0,100)) to scale into the 0 to 100 interval. If ntype = "zscore", npara could be list(zscore = c(0,1)) to scale to mean zero and standard deviation 1. This means you can store parameters for more than one normalisation type side by side, which helps in comparisons, adjustments, and sensitivity analyses.
dset	The data set to normalise.
directions	A vector specifying the direction assigned to each indicator. Needs to be the same length as the number of indicators, or the number of indicators in icodes, if specified.
individual	A list of named lists specifying individual normalisation to apply to specific indicators. Should be structured as follows: The name of each sub-list should be the indicator code. The the list elements are: • .$ntype is the type of normalisation to apply • .$npara is a corresponding object or parameters that are used by ntype, in the same format as npara above.
indiv_only	Logical: if FALSE (default), indicators not specified in individual are subjected to default normalisation. Otherwise if TRUE they are not normalised.
out2	Where to output the results. If "COIN" (default for COIN input), appends to updated COIN, otherwise if "df" outputs to data frame.

Details

Normalisation refers to the operation of bringing variables (indicators) onto a common scale. This is typically done by matching one or more indicator statistics. For example, the *min-max* method operates a linear transformation to make the minimum and maximum values of each indicator to be equal. The *z-score* method makes the standard deviation and variance equal. And so on.

This function supports a range of normalisation methods - see `ntype`. Some of these require supporting parameters or similar - to see full details check the online documentation.

Indicators can also be each normalised by a different method. See `individual`.

Value

If `out2 = "COIN"` (default for COIN input), returns an updated COIN object with a data frame `.$Data$Normalised` added. Else if `out2 = "df"` outputs a data frame of normalised data.

Examples

```
# build ASEM COIN
ASEM <- assemble(IndData = ASEMIndData, IndMeta = ASEMIndMeta, AggMeta = ASEMAggMeta)
# directly normalise raw data using min-max, onto 0-10 interval
ASEM <- normalise(ASEM, dset = "Raw", ntype = "minmax", npara = list(minmax = c(0,10)))
# Check: get indicator data first
NormData <- getIn(ASEM, dset = "Normalised")$ind_data_only
# ensure that min is 0 and max is 10 for all columns
stopifnot(
all(apply(NormData, MARGIN = 2, min, na.rm = TRUE) == 0),
all(apply(NormData, MARGIN = 2, max, na.rm = TRUE) == 10)
)
```

---

outrankMatrix *Outranking matrix*

Description

Constructs an outranking matrix based on a data frame of indicator data and corresponding weights. This function is used inside `aggregate()`.

Usage

```
outrankMatrix(ind_data, w = NULL)
```

Arguments

`ind_data`	A data frame or matrix of indicator data, with observations as rows and indicators as columns. No other columns should be present (e.g. label columns).
`w`	A vector of weights, which should have length equal to `ncol(ind_data)`. Weights are relative and will be re-scaled to sum to 1. If `w` is not specified, defaults to equal weights.

Value

An outranking matrix with nrow(ind_data rows and columns (matrix class).

Examples

```
# get a sample of a few indicators
ind_data <- COINr::ASEMIndData[12:16]
# calculate outranking matrix
ORmatrix <- outrankMatrix(ind_data)
# check output
stopifnot(is.matrix(ORmatrix), nrow(ORmatrix) == nrow(ind_data))
```

---

plotCorr *Static heatmaps of correlation matrices*

Description

Generates heatmaps of correlation matrices using **ggplot2**. This enables correlating any set of indicators against any other, and supports calling named aggregation groups of indicators. The `withparent` argument generates tables of correlations only with parents of each indicator. Also supports discrete colour maps using `flagcolours`, different types of correlation, and groups plots by higher aggregation levels.

Usage

```
plotCorr(
  COIN,
  dset = "Raw",
  icodes = NULL,
  aglevs = 1,
  cortype = "pearson",
  withparent = "parent",
  grouplev = NULL,
  showvals = TRUE,
  flagcolours = FALSE,
  flagthresh = c(-0.4, 0.3, 0.9),
  pval = 0.05,
  out2 = "fig"
)
```

Arguments

`COIN`	The COIN object
`dset`	The target data set.
`icodes`	An optional list of character vectors where the first entry specifies the indicator/aggregate codes to correlate against the second entry (also a specification of indicator/aggregate codes)
`aglevs`	The aggregation levels to take the two groups of indicators from. See `getIn()` for details.

cortype	The type of correlation to calculate, either "pearson", "spearman", or "kendall" (see stats::cor()).
withparent	If aglev[1] != aglev[2], and equal "parent" will only plot correlations of each row with its parent (default). If "family", plots the lowest aggregation level in aglevs against all its parent levels. If "none" plots the full correlation matrix.
grouplev	The aggregation level to group correlations by if aglev[1] == aglev[2]. By default, groups correlations into the aggregation level above. Set to 0 to disable grouping and plot the full matrix.
showvals	If TRUE, shows correlation values. If FALSE, no values shown.
flagcolours	If TRUE, uses discrete colour map with thresholds defined by flagthresh. If FALSE uses continuous colour map.
flagthresh	A 3-length vector of thresholds for highlighting correlations, if flagcolours = TRUE. flagthresh[1] is the negative threshold. Below this value, values will be flagged red. flagthresh[2] is the "weak" threshold. Values between flagthresh[1] and flagthresh[2] are coloured grey. flagthresh[3] is the "high" threshold. Anything between flagthresh[2] and flagthresh[3] is flagged "OK", and anything above flagthresh[3] is flagged "high".
pval	The significance level for plotting correlations. Correlations with $p < pval$ will be shown, otherwise they will be plotted as white squares. Set to 0 to disable this.
out2	If "fig" returns a plot, if "dflong" returns the correlation matrix in long form, if "dfwide", returns the correlation matrix in wide form. The last option here is probably useful if you want to present a table of the data in a report.

Details

This function calls getCorr().

Note that this function can only call correlations within the same data set (i.e. only one data set in .$Data).

Value

If out2 = "fig" returns a plot generated with **ggplot2**. These can be edited further with **ggplot2** commands. If out2 = "dflong" returns the correlation matrix as a data frame in long form, if out2 = "dfwide", returns the correlation matrix in wide form. The last option here is probably useful if you want to present a table of the data in a report.

See Also

- getCorr() Getting correlation matrices of indicator subsets

Examples

```
# build ASEM COIN
ASEM <- assemble(IndData = ASEMIndData, IndMeta = ASEMIndMeta, AggMeta = ASEMAggMeta)
# correlation data frame of indicators in connectivity sub-index, grouped by pillar
```

```
corrs <- plotCorr(ASEM, dset = "Raw", icodes = "Conn", aglevs = 1,
showvals = F, out2 = "dflong")
# NOTE to create a plot instead set out2 = "fig"
```

`plotframework` *Interactive sunburst plot of index structure*

Description

Plots the structure of the index using a sunburst plot using **plotly**. Output can be used as an interactive plot in html documents, e.g. via R Markdown.

Usage

```
plotframework(COIN)
```

Arguments

`COIN`	COIN object, or list with first entry is the indicator metadata, second entry is the aggregation metadata

Details

Note that this plot is sensitive to the *order* of the elements. If you use `assemble()` and input a COIN, this plot should work automatically. If you input a list, you should make sure that the indicator metadata is ordered by descending order of the hierarchy (i.e. highest level, working downwards).

Value

Interactive sunburst plot built using **plotly**. This can be edited further with **plotly** commands.

Examples

```
# build ASEM COIN
ASEM <- assemble(IndData = ASEMIndData, IndMeta = ASEMIndMeta, AggMeta = ASEMAggMeta)
# plot framework
plotframework(ASEM)
```

`plotIndDist` *Static indicator distribution plots*

Description

Plots indicator distributions using box plots, dot plots, violin plots, violin-dot plots, and histograms. Supports plotting multiple indicators by calling aggregation groups.

Usage

```
plotIndDist(
  COINobj,
  dset = "Raw",
  icodes = NULL,
  aglev = 1,
  type = "Box",
  ntype = NULL,
  npara = NULL
)
```

Arguments

`COINobj`	The COIN object, or a data frame of indicator data
`dset`	The source data set to use for indicator data (if input is COIN object)
`icodes`	A character vector of indicator names to plot. Defaults to all indicators.
`aglev`	The aggregation level to extract the indicator data from. Defaults to indicator level (1).
`type`	The type of plot. Currently supported `"Box"`, `"Dot"`, `"Violin"`, `"Violindot"`, `"Histogram"`.
`ntype`	The type of normalisation to apply. If `NULL`, no normalisation applied, otherwise specify using `ntype` options in `normalise()`.
`npara`	Optional parameters to pass to `normalise()` if normalisation required.

Details

This function also optionally normalises indicators so they can be compared more easily side by side. For this purpose it calls `normalise()` - see `ntype` and `npara` arguments.

See COINr online documentation and `getIn()` for more information on accessing/plotting groups.

Value

Plots generated with **ggplot2**. These can be edited further with **ggplot2** commands.

Examples

```
# build ASEM COIN
ASEM <- assemble(IndData = ASEMIndData, IndMeta = ASEMIndMeta, AggMeta = ASEMAggMeta)
# plot indicators in Physical pillar
plotIndDist(ASEM, type = "Box", dset = "Raw", icodes = "Physical")
```

---

plotSA *Plot sensitivity indices*

Description

Plots sensitivity indices as bar or pie charts.

Usage

```
plotSA(SAresults, ptype = "bar")
```

Arguments

SAresults	A list of sensitivity/uncertainty analysis results from sensitivity().
ptype	Type of plot to generate - either "bar", "pie" or "box".

Details

To use this function you first need to run sensitivity(). Then enter the resulting list as the SAresults argument here. See COINr online documentation for more details.

Value

A plot of sensitivity indices generated by ggplot2.

See Also

- sensitivity() Perform global sensitivity or uncertainty analysis on a COIN
- plotSARanks() Plot confidence intervals on ranks following a sensitivity analysis

Examples

```
# build ASEM COIN up to aggregation
ASEM <- build_ASEM()

# define noise to be applied to weights
nspecs <- data.frame(AgLevel = c(2,3), NoiseFactor = c(0.25,0.25))

# create list specifying assumptions to vary and alternatives
SAspecs <- list(
```

```
  normalise = list(ntype = c("minmax", "rank", "dist2max")),
  weights = list(NoiseSpecs = nspecs, Nominal = "Original")
)

# run uncertainty analysis
# here we set N deliberately much lower than normal to enable quick testing
# Would recommend in a practical case to increase to perhaps 500 (more is always better)
SAresults <- sensitivity(ASEM, v_targ = "Index",
                         SA_specs = SAspecs,
                         N = 5,
                         SA_type = "SA")

# Plot results as a bar chart
plotSA(SAresults, ptype = "bar")
```

`plotSARanks` *Plot ranks from an uncertainty/sensitivity analysis*

Description

Plots the ranks resulting from an uncertainty and sensitivity analysis, in particular plots the median, and 5th/95th percentiles of ranks.

Usage

```
plotSARanks(SAresults, plot_units = NULL, order_by = "nominal")
```

Arguments

`SAresults`	A list of sensitivity/uncertainty analysis results from `sensitivity()`.
`plot_units`	A character vector of units to plot. Defaults to all units. You can also set to `"top10"` to only plot top 10 units, and `"bottom10"` for bottom ten.
`order_by`	If set to `"nominal"`, orders the rank plot by nominal ranks (i.e. the original ranks prior to the sensitivity analysis). Otherwise if `"median"`, orders by median ranks.

Details

To use this function you first need to run `sensitivity()`. Then enter the resulting list as the `SAresults` argument here. See COINr online documentation for more details.

Value

A plot of rank confidence intervals, generated by 'ggplot2'.

See Also

- sensitivity() Perform global sensitivity or uncertainty analysis on a COIN
- plotSA() Plot sensitivity indices following a sensitivity analysis.

Examples

```
# build ASEM COIN up to aggregation
ASEM <- build_ASEM()

# define noise to be applied to weights
nspecs <- data.frame(AgLevel = c(2,3), NoiseFactor = c(0.25,0.25))

# create list specifying assumptions to vary and alternatives
SAspecs <- list(
  impute = list(imtype = c("indgroup_mean", "ind_mean", "none")),
  normalise = list(ntype = c("minmax", "rank", "dist2max")),
  weights = list(NoiseSpecs = nspecs, Nominal = "Original")
)

# run uncertainty analysis
# here we set N deliberately much lower than normal to enable quick testing
# Would recommend in a practical case to increase to perhaps 500 (more is always better)
SAresults <- sensitivity(ASEM, v_targ = "Index",
                         SA_specs = SAspecs,
                         N = 20,
                         SA_type = "UA")

# plot rank intervals
plotSARanks(SAresults)
```

---

rankDF *Convert a data frame to ranks*

Description

Replaces all numerical columns of a data frame with their ranks. Uses sport ranking, i.e. ties share the highest rank place. Ignores non-numerical columns. See rank().

Usage

```
rankDF(df)
```

Arguments

df	A data frame

Value

A data frame equal to the data frame that was input, but with any numerical columns replaced with ranks.

See Also

- `roundDF()` Round a data frame to a specified number of decimals.

Examples

```
# some random data, with a column of characters
df <- data.frame(RName = c("A", "B", "C"),
Score1 = runif(3), Score2 = runif(3))
# convert to ranks
rankDF(df)
```

`regen` *Regenerate COIN object*

Description

Function to regenerate the results of the COIN, using the methodological parameters stored in `.$Method`. This function calls the construction functions of COINr in the order that they are found in `.$Method`, along with any custom code found in `.$Method$Custom`.

Usage

```
regen(COINold, quietly = FALSE)
```

Arguments

`COINold`	COIN object containing specifications on how to regenerate.
`quietly`	Logical: if `TRUE` suppresses all messages from COINr functions (warnings may still occur though).

Details

Note that while sets of weights will be passed to the regenerated COIN, anything in `.$Analysis` will be removed and will have to be recalculated.

For more details on regeneration of COINs, comparisons and adjustments, see the online documentation.

Value

An updated COIN object, with all data sets recalculated according to specifications in `.$Method`. Weight sets will be passed through.

See Also

- compTable() compare two different COINs
- compTableMulti() compare multiple COINs

Examples

```
ASEM <- build_ASEM()
# Make a copy
ASEMAltNorm <- ASEM
# Edit .$Method
ASEMAltNorm$Method$normalise$ntype <- "borda"
# Regenerate
ASEMAltNorm <- regen(ASEMAltNorm, quietly = TRUE)
```

---

removeElements *Check the effect of removing indicators or aggregates*

Description

This is an analysis function for seeing what happens when elements of the composite indicator are removed. This can help with "what if" experiments and acts as different measure of the influence of each indicator or aggregate.

Usage

```
removeElements(COIN, aglev, isel, quietly = FALSE)
```

Arguments

COIN	The COIN, which must be constructed up to and including the aggregation step.
aglev	The level at which to remove elements. For example, aglev = 1 would check the effect of removing each indicator, one at a time. aglev = 2 would check the effect of removing each of the aggregation groups above the indicator level, one at a time.
isel	A character string indicating the indicator or aggregate code to extract from each iteration. I.e. normally this would be set to the index code to compare the ranks of the index upon removing each indicator or aggregate. But it can be any code that is present in .$Data$Aggregated.
quietly	Logical: if FALSE (default) will output to the console an indication of progress. Might be useful when iterating over many indicators. Otherwise set to TRUE to shut this up.

Details

One way of looking at indicator "importance" in a composite indicator is via correlations. A different way is to see what happens if we remove the indicator completely from the framework. If removing an indicator or a whole aggregation of indicators results in very little rank change, it is one indication that perhaps it is not necessary to include it. Emphasis on *one*: there may be many other things to take into account.

This function works by successively setting the weight of each indicator or aggregate to zero. If the analysis is performed at the indicator level, it creates a copy of the COIN, sets the weight of the first indicator to zero, regenerates the results, and compares to the nominal results (results when no weights are set to zero). It repeats this for each indicator in turn, such that each time one indicator is set to zero weights, and the others retain their original weights. The output is a series of tables comparing scores and ranks (see Value).

Note that "removing the indicator" here means more precisely "setting its weight to zero". In most cases the first implies the second, but check that the aggregation method that you are using satisfies this relationship. For example, if the aggregation method does not use any weights, then setting the weight to zero will have no effect.

Value

A list with elements as follows:

- `.$Scores`: a data frame where each column is the scores for each unit, with indicator/aggregate corresponding to the column name removed. E.g. `.$Scores$Ind1` gives the scores resulting from removing "Ind1".
- `.$Ranks`: as above but ranks
- `.$RankDiffs`: as above but difference between nominal rank and rank on removing each indicator/aggregate
- `.$RankAbsDiffs`: as above but absolute rank differences
- `.$MeanAbsDiffs`: as above, but the mean of each column. So it is the mean (over units) absolute rank change resulting from removing each indicator or aggregate.

See Also

- `compTable()` Comparison table between two COINs
- `compTableMulti()` Comparison table between multiple COINs

Examples

```
# check the effect of removing ASEM sub-pillars, one at a time
# First build ASEM index
ASEM <- build_ASEM()
# now run check at sub-index level (level 3), on index scores/ranks
CheckPillars <- removeElements(ASEM, 3, "Index")
# summary by pillar
CheckPillars$MeanAbsDiff
# have a look at the rest of the output list to see more details.
```

`replaceDF` *Replace multiple values in a data frame*

Description

Given a data frame (or vector), this function replaces values according to a look up table or dictionary. In COINr this may be useful for exchanging categorical data with numeric scores, prior to assembly. Or for changing codes.

Usage

```
replaceDF(df, lookup)
```

Arguments

`df`	A data frame or a vector
`lookup`	A data frame with columns `old` (the values to be replaced) and `new` the values to replace with. See details.

Details

The lookup data frame must not have any duplicated values in the `old` column. This function looks for exact matches of elements of the `old` column and replaces them with the corresponding value in the `new` column. For each row of `lookup`, the class of the old value must match the class of the new value. This is to keep classes of data frames columns consistent. If you wish to replace with a different class, you should convert classes in your data frame before using this function.

Value

A data frame with replaced values

See Also

- `assemble()` Assemble a COIN - this function optionally calls `extractYear()`.
- `rankDF()` Replace numeric columns of a data frame with ranks.
- `roundDF()` Replace numeric columns of a data frame with rounded values.
- `compareDF()` Detailed comparison of two similar data frames.

Examples

```
# replace sub-pillar codes in ASEM indicator metadata
codeswap <- data.frame(old = c("Conn", "Sust"), new = c("SI1", "SI2"))
# swap codes in both indmeta and aggmeta
replaceDF(ASEMIndMeta, codeswap)
replaceDF(ASEMAggMeta, codeswap)
```

`resultsDash` *Results visualisation dashboard*

Description

Generates an interactive visualisation of results. Requires Shiny and an interactive R session.

Usage

```
resultsDash(COIN, dset = "Aggregated")
```

Arguments

`COIN`	The COIN object
`dset`	The initial data set to explore.

Details

This function provides a fast way to present and explore results in a COIN. It plots interactive bar charts of any indicator or aggregate, and maps if the unit codes are ISO alpha-3 country codes. It also includes an interactive results table, and the possibility to quickly compare units on a radar chart.

Value

Interactive app for exploring results (if running an interactive R session). Otherwise will simply generate a message.

See Also

- `indDash()` shiny dashboard for exploring indicator distributions
- `rew8r()` shiny dashboard for altering weights and visualising correlations

Examples

```
# To be run in an interactive R session...
if(interactive()){
# build ASEM up to results
ASEM <- build_ASEM()
# launch results dashboard
resultsDash(ASEM)
}
```

`rew8r` *Re-weight indicators*

Description

Interactive gadget which lets you adjust weights and see the effects. Weights can be saved with new names to the COIN object.

Usage

```
rew8r(COIN)
```

Arguments

`COIN`	COIN object

Details

Correlations between indicators, and between indicators and index, can inform about how well the composite indicator is conveying information in the underlying indicators. For example, a correlation of zero between an indicator and the index shows that knowing the index ranks, you have no indication of what the ranks of the underlying indicator are. Since one of the objectives of a composite indicator is to summarise the information in its underlying indicators, this can be a problem.

The correlation between indicators and index (and other levels) can be adjusted by changing weights. But the effect of changing weights can be hard to gauge. The `rew8r()` app allows you to play around with the weights at any level, and to see what happens to the resulting correlations of interest. It also demonstrates what happens to the results. Rather than changing weights and manually regenerating the results, the `rew8r()` app does all this for you. If you find a set or sets of weights that you like, you can also save it/them back to the COIN as a new weight-set(s). To do this, click "Save", and then at the end of the session, "Close app".

Consider that changing weights to "fix" correlations may result in unusual sets of weights that are hard to justify. This tool may be more useful as a curiosity, as part of a "what-if" analysis, or simply to better understand what is going on inside your index.

NOTE that you need to have aggregated your data first before using this. This app also requires an interactive R session to run.

Value

An updated COIN object with additional sets of weights in `.$Parameters$Weights`, if specified.

See Also

- `weightOpt()` Correlation-optimised weights
- `plotCorr()` Correlation plots

Examples

```
## Only run this example in interactive R sessions
if (interactive()) {
# build ASEM COIN up to aggregation
ASEM <- build_ASEM()
# launch app
ASEM <- rew8r(ASEM)
}
```

---

roundDF *Round down a data frame*

Description

Tiny function just to round down a data frame for display in a table.

Usage

```
roundDF(df, decimals = 2)
```

Arguments

df	A data frame to input
decimals	The number of decimal places to round to (default 2)

Value

A data frame, with any numeric columns rounded to the specified amount.

See Also

- rankDF() Replace data frame numbers with ranks.

Examples

```
roundDF( as.data.frame(matrix(runif(20),10,2)), decimals = 3)
```

SA_estimate *Estimate sensitivity indices*

Description

Post process a sample to obtain sensitivity indices. This function takes a univariate output which is generated as a result of running a Monte Carlo sample from SA_sample() through a system. Then it estimates sensitivity indices using this sample.

Usage

```
SA_estimate(yy, N, d, Nboot = NULL)
```

Arguments

yy	A vector of model output values, as a result of a $N(d+2)$ Monte Carlo design.
N	The number of sample points per dimension.
d	The dimensionality of the sample
Nboot	Number of bootstrap draws for estimates of confidence intervals on sensitivity indices. If this is not specified, bootstrapping is not applied.

Details

This function is built to be used inside sensitivity(). See COINr online documentation for more details.

Value

A list with the output variance, plus a data frame of first order and total order sensitivity indices for each variable, as well as bootstrapped confidence intervals if !is.null(Nboot).

See Also

- sensitivity() Perform global sensitivity or uncertainty analysis on a COIN
- SA_sample() Input design for estimating sensitivity indices

Examples

```
# This is a generic example rather than applied to a COIN (for reasons of speed)

# A simple test function
testfunc <- function(x){
x[1] + 2*x[2] + 3*x[3]
}

# First, generate a sample
X <- SA_sample(500, 3)
```

```
# Run sample through test function to get corresponding output for each row
y <- apply(X, 1, testfunc)

# Estimate sensitivity indices using sample
SAinds <- SA_estimate(y, N = 500, d = 3, Nboot = 1000)
SAinds$SensInd
# Notice that total order indices have narrower confidence intervals than first order.
```

SA_sample *Generate sample for sensitivity analysis*

Description

Generates an input sample for a Monte Carlo estimation of global sensitivity indices. Used in the sensitivity() function. The total sample size will be $N(d+2)$.

Usage

```
SA_sample(N, d)
```

Arguments

N	The number of sample points per dimension.
d	The dimensionality of the sample

Details

This function generates a Monte Carlo sample as described e.g. in the Global Sensitivity Analysis: The Primer book. See also COINr online documentation.

Value

A matrix with $N(d+2)$ rows and d columns.

See Also

- sensitivity() Perform global sensitivity or uncertainty analysis on a COIN.
- SA_estimate() Estimate sensitivity indices from system output, as a result of input design from SA_sample().

Examples

```
# sensitivity analysis sample for 3 dimensions with 100 points per dimension
X <- SA_sample(100, 3)
```

`sensitivity` *Sensitivity analysis*

Description

Performs global uncertainty and sensitivity analysis on a COIN.

Usage

```
sensitivity(
  COIN,
  v_targ,
  SA_specs,
  N,
  SA_type = "UA",
  NrepWeights = 1000,
  store_results = "results+params",
  Nboot = NULL,
  quietly = FALSE
)
```

Arguments

`COIN`	A COIN (this function does not support data frame input)
`v_targ`	The target variable to perform SA or UA on. Currently just supports one variable, which should be an indicator/aggregate code present in `.$Data$Aggregated`.
`SA_specs`	A list which specifies which variables to perturb, and which alternatives/distributions to use
`N`	The number of Monte Carlo replications.
`SA_type`	The type of analysis to run. `"UA"` runs an uncertainty analysis. `"SA"` runs a sensitivity analysis (which anyway includes an uncertainty analysis).
`NrepWeights`	The number of weight-replications to generate. Default 1000.
`store_results`	Which results to store • `"onlyresults"` only stores scores, ranks and rank statistics (e.g. mean, median, quantiles) • `"results+params"` (default) stores all of the above, plus a record of the parameter values used for each replication • `"results+method"` stores all of the above, plus the full `.$Method` list of each replication • `"results+COIN"` stores all results and the complete COIN of each replication (this could result in a very large list).
`Nboot`	Number of bootstrap draws for estimates of confidence intervals on sensitivity indices. If this is not specified, bootstrapping is not applied. Ignored if `SA_type = "UA"`.
`quietly`	If `FALSE` (default), gives progress messages. Set `TRUE` to suppress these.

Details

To perform a sensitivity or uncertainty analysis, you must specify *which* parameters/assumptions to vary and *what* their alternative values are. This is the `SA_specs` argument below. To understand how this works, please see the COINr online documentation.

The output of this function can be visualised with the functions `plotSARanks()` and `plotSA()`.

Value

Sensitivity analysis results as a list, containing:

- `.$Scores` a data frame with a row for each unit, and columns are the scores for each replication.
- `.$Parameters` a record of the parameters used for each iteration
- `.$Ranks` as `.$Scores` but for unit ranks
- `.$RankStats` summary statistics for ranks of each unit
- `.$Nominal` the nominal scores and ranks of each unit (i.e. from the original COIN)
- Some information on the time elapsed, average time, and the parameters perturbed.
- Depending on the setting of `store_results`, may also contain a list of Methods or a list of COINs for each replication.

See Also

- `plotSARanks()` Plot confidence intervals of ranks following UA or SA
- `plotSA()` Plot sensitivity indices following a sensitivity analysis

Examples

```
# build ASEM COIN up to aggregation
ASEM <- build_ASEM()

# define noise to be applied to weights
nspecs <- data.frame(AgLevel = c(2,3), NoiseFactor = c(0.25,0.25))

# create list specifying assumptions to vary and alternatives
SAspecs <- list(
  impute = list(imtype = c("indgroup_mean", "ind_mean", "none")),
  normalise = list(ntype = c("minmax", "rank", "dist2max")),
  weights = list(NoiseSpecs = nspecs, Nominal = "Original")
)

# run uncertainty analysis
# here we set N deliberately much lower than normal to enable quick testing
# Would recommend in a practical case to increase to perhaps 500 (more is always better)
SAresults <- sensitivity(ASEM, v_targ = "Index",
                         SA_specs = SAspecs,
                         N = 15,
                         SA_type = "UA")
```

```
# to run a sensitivity analysis set SA_type = "SA" (takes longer)
```

`treat` *Treatment of outliers*

Description

Takes the COIN object and Winsorises indicators where necessary or specified, or reverts to log transform or similar. This is done one indicator at a time.

Usage

```
treat(
  COIN,
  dset = NULL,
  winmax = NULL,
  winchange = NULL,
  deflog = NULL,
  boxlam = NULL,
  t_skew = NULL,
  t_kurt = NULL,
  individual = NULL,
  indiv_only = NULL,
  bypass_all = NULL
)
```

Arguments

`COIN`	The COIN object
`dset`	The data set to treat
`winmax`	The maximum number of points to Winsorise for each indicator. If `NA`, will keep Winsorising until skewness and kurtosis thresholds achieved (but it is likely this will cause errors).
`winchange`	Logical: if `TRUE` (default), Winsorisation can change direction from one iteration to the next. Otherwise if `FALSE`, no change.
`deflog`	The type of transformation to apply if Winsorisation fails. If `"log"`, use simple `log(x)` as log transform (note: indicators containing negative values will be skipped). If `"CTlog"`, will do `log(x-min(x) + a)`, where `a <-0.01*(max(x)-min(x))`, similar to that used in the COIN Tool. If `"CTlog_orig"`, this is exactly the COIN Tool log transformation, which is `log(x-min(x) + 1)`. If `"GIIlog"`, use GII log transformation. If "boxcox", performs a Box-Cox transformation. In this latter case, you should also specify `boxlam`. Finally, if `"none"`, will return the indicator untreated.

`boxlam` The lambda parameter of the Box-Cox transform.

`t_skew` Absolute skew threshold (default 2)

`t_kurt` Kurtosis threshold (default 3.5)

`individual` A data frame specifying individual treatment for each indicator, with each row corresponding to one indicator to be treated. Columns are:

- `IndCode` The code of the indicator to be treated.
- `Treat` The type of treatment to apply, one of `"win"` (Winsorise), `"log"` (log), `"GIIlog"` (GII log), `"CTlog"` (COIN Tool log), `"boxcox"` (Box Cox), or `"None"` (no treatment).
- `Winmax` The maximum number of points to Winsorise. Ignored if the corresponding entry in `"Treat"` is not `"win"`.
- `Thresh` Either `NA`, which means that Winsorisation will continue up to `winmax` with no checks on skew and kurtosis, or `"thresh"`, which uses the skew and kurtosis thresholds specified in `t_skew` and `t_kurt`.
- `boxlam` Lambda parameter for the Box Cox transformation

`indiv_only` Logical: if `TRUE`, only the indicators specified in `"individual"` are treated. If `FALSE`, all indicators are treated: any outside of `individual` will get default treatment.

`bypass_all` Logical: if `TRUE`, bypasses all data treatment and returns the original data. This is useful for sensitivity analysis and comparing the effects of turning data treatment on and off.

Details

Outliers are identified according to skewness and kurtosis thresholds. The algorithm attempts to reduce the absolute skew and kurtosis by successively Winsorising points up to a specified limit. If this limit is reached, it applies a nonlinear transformation.

The process is detailed in the COINr online documentation.

Value

If the input is a COIN, outputs an updated COIN with a new treated data set at `.$Data$Treated`, as well as information about the data treatment in `.$Analysis$Treated`. Else if the input is a data frame, outputs both the treated data set and the information about data treatment to a list.

See Also

- `indDash()` Interactive app for checking indicator distributions. Useful for comparing before/after data treatment.

Examples

```
# assemble ASEM COIN
ASEM <- assemble(IndData = ASEMIndData, IndMeta = ASEMIndMeta, AggMeta = ASEMAggMeta)
# treat raw data set, Winsorise up to a maximum of five points
ASEM <- treat(ASEM, dset = "Raw", winmax = 5)
# inspect what was done
```

```
ASEM$Analysis$Treated$TreatSummary
# check whether skew and kurtosis now within limits
ASEM$Analysis$Treated$StatTable$SK.outlier.flag
```

`weightOpt` *Weight optimisation*

Description

This function provides optimised weights to agree with a pre-specified vector of "target importances".

Usage

```
weightOpt(
  COIN,
  itarg,
  aglev,
  cortype = "pearson",
  optype = "balance",
  toler = NULL,
  maxiter = NULL,
  out2 = NULL
)
```

Arguments

`COIN`	COIN object
`itarg`	a vector of (relative) target importances. For example, `c(1,2,1)` would specify that the second indicator should be twice as "important" as the other two.
`aglev`	The aggregation level to apply the weight adjustment to.
`cortype`	The type of correlation to use - can be either `"pearson"`, `"spearman"` or `"kendall"`. See stats::cor.
`optype`	The optimisation type. Either `"balance"`, which aims to balance correlations according to a vector of "importances" specified by `itarg` (default), or `"infomax"` which aims to maximise overall correlations. *This latter option is experimental and may not yet work very well*.
`toler`	Tolerance for convergence. Defaults to 0.001 (decrease for more accuracy, increase if convergence problems).
`maxiter`	Maximum number of iterations. Default 500.
`out2`	Where to output the results. If `"COIN"` (default for COIN input), appends to updated COIN, creating a new list of weights in `.$Parameters$Weights`. Otherwise if `"list"` outputs to a list (default).

Details

This is a linear version of the weight optimisation proposed in this paper: doi: 10.1016/j.ecolind.2017.03.056. Weights are optimised to agree with a pre-specified vector of "importances". The optimised weights are returned back to the COIN.

See the chapter in the COINr online documentation for more details.

Value

If `out2 = "COIN"` returns an updated COIN object with a new set of weights in `.$Parameters$Weights`, plus details of the optimisation in `.$Analysis`. Else if `out2 = "list"` the same outputs (new weights plus details of optimisation) are wrapped in a list.

See Also

- `rew8r()` Interactive app for adjusting weights and seeing effects on correlations
- `getPCA()` PCA, including weights from PCA

Examples

```
# build ASEM COIN up to aggregation
ASEM <- build_ASEM()
# optimise sub-pillar weights to give equal correlations with index
ASEM <- weightOpt(ASEM, itarg = "equal", aglev = 3, out2 = "COIN")
```

---

weights2corr *Recalculate correlations and ranks based on new weights*

Description

This is a short cut function which takes a new set of indicator weights, and recalculates the COIN results based on these weights. It returns a summary of rankings and the correlations between indicators and index.

Usage

```
weights2corr(
  COIN,
  w,
  aglevs = NULL,
  icodes = NULL,
  cortype = "pearson",
  withparent = TRUE
)
```

Arguments

COIN	COIN object
w	Full data frame of weights for each level
aglevs	A 2-length vector with two aggregation levels to correlate against each other
icodes	List of two character vectors of indicator codes, corresponding to the two aggregation levels
cortype	Correlation type. Either "pearson" (default), "kendall" or "spearman". See stats::cor.
withparent	Logical: if TRUE, only correlates with the parent, e.g. sub-pillars are only correlated with their parent pillars and not others.

Details

This function is principally used inside rew8r(). The w argument should be a data frame of weights, of the same format as the data frames found in .$Parameters$Weights.

Value

A list where .$cr is a vector of correlations between each indicator and the index, and .$dat is a data frame of rankings, with unit code, and index, input and output scores

See Also

- rew8r() Interactive app for adjusting weights and seeing effects on correlations
- getCorr() Get correlations between indicators/levels

Examples

```
# build ASEM COIN up to aggregation
ASEM <- build_ASEM()
# get correlations between pillars (level 2) and index (level 4)
# original weights used just for demonstration, normally you would alter first.
l <- weights2corr(ASEM, ASEM$Parameters$Weights$Original, aglevs = c(2,4))
```

WorldDenoms *World denomination data*

Description

A small selection of common denominator indicators, which includes GDP, Population, Area, GDP per capita and income group. All data sourced from the World Bank as of Feb 2021 (data is typically from 2019). Note that this is intended as example data, and it would be a good idea to use updated data from the World Bank when needed. In this data set, country names have been altered slightly so as to include no accents - this is simply to make it more portable between distributions.

Usage

```
WorldDenoms
```

Format

A data frame with 249 rows and 7 variables.

Source

https://data.worldbank.org/

Index