Package 'dataquieR'

March 1, 2023

Title Data Quality in Epidemiological Research

Version 2.0.0

Description Data quality assessments guided by a 'data quality framework introduced by Schmidt and colleagues, 2021' <doi:10.1186/s12874-021-01252-7> target the data quality dimensions integrity, completeness, consistency, and accuracy. The scope of applicable functions rests on the availability of extensive metadata which can be provided in spreadsheet tables. Either standardized (e.g. as 'html5' reports) or individually tailored reports can be generated. For an introduction into the specification of corresponding metadata, please refer to the 'package website' <https://dataquality.ship-med.uni-greifswald.de/Annotation_of_Metadata.html>.

License BSD_2_clause + file LICENSE

URL https://dataquality.ship-med.uni-greifswald.de/

BugReports https://gitlab.com/libreumg/dataquier/-/issues

Depends R (>= 3.6.0)

Imports dplyr (>= 1.0.2), emmeans, ggplot2 (>= 3.4.0), lme4, lubridate, MASS, MultinomialCI, parallelMap, patchwork, R.devices, reshape, rlang, robustbase, qmrparser, utils, rio, scales

Suggests cli, whoami, lifecycle, anytime, cowplot (>= 0.9.4), digest, DT (>= 0.23), flexdashboard, flexsiteboard, htmltools, knitr, markdown, parallel, rmarkdown, rstudioapi, testthat (>= 3.0.0), tibble, vdiffr, pkgload, Rdpack, callr, colorspace, withr, plotly, ggvenn, htmlwidgets

VignetteBuilder knitr

Encoding UTF-8

KeepSource TRUE

Language en-US

RoxygenNote 7.2.3

NeedsCompilation no

Author University Medicine Greifswald [cph],
Elisa Kasbohm [aut] (<https://orcid.org/0000-0001-5261-538X>),
Joany Marino [aut] (<https://orcid.org/0000-0002-4657-3758>),
Adrian Richter [aut] (<https://orcid.org/0000-0002-3372-2021>),
Carsten Oliver Schmidt [aut] (<https://orcid.org/0000-0001-5266-9396>),
Stephan Struckmann [aut, cre] (<https://orcid.org/0000-0002-8565-7962>),
German Research Foundation (DFG SCHM 2744/3-1, SCHM 2744/9-1, SCHM 2744/3-4) [fnd],
National Research Data Infrastructure for Personal Health Data: (NFDI 13/1) [fnd],
European Union's Horizon 2020 programme (euCanSHare, grant agreement No. 825903) [fnd]

Maintainer Stephan Struckmann <stephan.struckmann@uni-greifswald.de>

Repository CRAN

Date/Publication 2023-03-01 16:30:02 UTC

R topics documented:

`..manual` *Holds parts of the manual at run-time*

Description

Holds parts of the manual at run-time

Usage

```
..manual
```

Format

An object of class `environment` of length 0.

`.util_internal_normalize_meta_data`
Make normalizations of v2.0 item_level metadata.

Description

Requires referred missing-tables being available by `prep_get_data_frame`.

Usage

```
.util_internal_normalize_meta_data(meta_data = "item_level", verbose = TRUE)
```

Arguments

meta_data	data.frame the old item-level-metadata
verbose	logical display all estimated decisions

.variable_arg_roles *Variable-argument roles*

Description

A Variable-argument role is the intended use of an argument of a indicator function – an argument that refers variables. In general for the table .variable_arg_roles, the suffix _var means one variable allowed, while _vars means more than one. The default sets of arguments for util_correct_variable_use/util_correct_variable_ are defined from the point of usage, e.g. if it could be, that NAs are in the list of variable names, the function should be able to remove certain response variables from the output and not disallow them by setting `allow_na` to `FALSE`.

Usage

```
.variable_arg_roles
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 14 rows and 8 columns.

See Also

`util_correct_variable_use()`

`util_correct_variable_use2()`

---

acc_distributions *Plots and checks for distributions*

Description

Data quality indicator checks "Unexpected location" and "Unexpected proportion" with histograms and, if a grouping variable is included, plots of empirical cumulative distributions for the subgroups.

Usage

```
acc_distributions(
  resp_vars = NULL,
  group_vars = NULL,
  study_data,
  meta_data,
  label_col,
  check_param = c("any", "location", "proportion"),
  plot_ranges = TRUE,
  flip_mode = "noflip"
)
```

Arguments

resp_vars	variable list the names of the measurement variables
group_vars	variable list the name of the observer, device or reader variable
study_data	data.frame the data frame that contains the measurements
meta_data	data.frame the data frame that contains metadata attributes of study data
label_col	variable attribute the name of the column in the metadata with labels of variables
check_param	enum any | location | proportion. Which type of check should be conducted (if possible): a check on the location of the mean or median value of the study data, a check on proportions of categories, or either of them if the necessary metadata is available.
plot_ranges	logical Should the plot show ranges and results from the data quality checks? (default: TRUE)
flip_mode	enum default | flip | noflip | auto. Should the plot be in default orientation, flipped, not flipped or auto-flipped. Not all options are always supported. In general, this con be controlled by setting the `roptions(dataquieR.flip_mode = ...)`. If called from `dq_report`, you can also pass `flip_mode` to all function calls or set them specifically using `specific_args`.

Value

A list with:

- `SummaryTable`: data.frame containing data quality checks for "Unexpected location" (`FLG_acc_ud_loc`) and "Unexpected proportion" (`FLG_acc_ud_prop`) for each response variable in `resp_vars`.
- `SummaryData`: a data.frame containing data quality checks for "Unexpected location" and / or "Unexpected proportion" for a report
- `SummaryPlotList`: list of ggplots for each response variable in `resp_vars`.

Algorithm of this implementation:

- If no response variable is defined, select all variables of type float or integer in the study data.
- Remove missing codes from the study data (if defined in the metadata).
- Remove measurements deviating from (hard) limits defined in the metadata (if defined).

- Exclude variables containing only NA or only one unique value (excluding NAs).
- Perform check for "Unexpected location" if defined in the metadata (needs a LOCATION_METRIC (mean or median) and LOCATION_RANGE (range of expected values for the mean and median, respectively)).
- Perform check for "Unexpected proportion" if defined in the metadata (needs PROPORTION_RANGE (range of expected values for the proportions of the categories)).
- Plot histogram(s).
- If group_vars is specified by the user, distributions within group-wise ecdf are presented.

See Also

Online Documentation

`acc_distributions_loc` *Plots and checks for distributions – Location*

Description

Data quality indicator checks "Unexpected location" and "Unexpected proportion" with histograms and, if a grouping variable is included, plots of empirical cumulative distributions for the subgroups.

Usage

```
acc_distributions_loc(
  resp_vars = NULL,
  study_data,
  meta_data,
  label_col,
  check_param = "location",
  plot_ranges = TRUE,
  flip_mode = "noflip"
)
```

Arguments

`resp_vars`	variable list the names of the measurement variables
`study_data`	data.frame the data frame that contains the measurements
`meta_data`	data.frame the data frame that contains metadata attributes of study data
`label_col`	variable attribute the name of the column in the metadata with labels of variables
`check_param`	enum any | location | proportion. Which type of check should be conducted (if possible): a check on the location of the mean or median value of the study data, a check on proportions of categories, or either of them if the necessary metadata is available.
`plot_ranges`	logical Should the plot show ranges and results from the data quality checks? (default: TRUE)

`flip_mode` enum default | flip | noflip | auto. Should the plot be in default orientation, flipped, not flipped or auto-flipped. Not all options are always supported. In general, this con be controlled by setting the `roptions(dataquieR.flip_mode = ...)`. If called from `dq_report`, you can also pass `flip_mode` to all function calls or set them specifically using `specific_args`.

Value

A list with:

- `SummaryTable`: data.frame containing data quality checks for "Unexpected location" (`FLG_acc_ud_loc`) and "Unexpected proportion" (`FLG_acc_ud_prop`) for each response variable in `resp_vars`.
- `SummaryData`: a data.frame containing data quality checks for "Unexpected location" and / or "Unexpected proportion" for a report
- `SummaryPlotList`: list of ggplots for each response variable in `resp_vars`.

Algorithm of this implementation:

- If no response variable is defined, select all variables of type float or integer in the study data.
- Remove missing codes from the study data (if defined in the metadata).
- Remove measurements deviating from (hard) limits defined in the metadata (if defined).
- Exclude variables containing only `NA` or only one unique value (excluding `NA`s).
- Perform check for "Unexpected location" if defined in the metadata (needs a LOCATION_METRIC (mean or median) and LOCATION_RANGE (range of expected values for the mean and median, respectively)).
- Perform check for "Unexpected proportion" if defined in the metadata (needs PROPORTION_RANGE (range of expected values for the proportions of the categories)).
- Plot histogram(s).
- If group_vars is specified by the user, distributions within group-wise ecdf are presented.

See Also

- acc_distributions
- Online Documentation

`acc_distributions_loc_ecdf`

Plots and checks for distributions – Location, ECDF

Description

Data quality indicator checks "Unexpected location" and "Unexpected proportion" with histograms and, if a grouping variable is included, plots of empirical cumulative distributions for the subgroups.

Usage

```
acc_distributions_loc_ecdf(
  resp_vars = NULL,
  group_vars = NULL,
  study_data,
  meta_data,
  label_col,
  check_param = "location",
  plot_ranges = TRUE,
  flip_mode = "noflip"
)
```

Arguments

resp_vars	variable list the names of the measurement variables
group_vars	variable list the name of the observer, device or reader variable
study_data	data.frame the data frame that contains the measurements
meta_data	data.frame the data frame that contains metadata attributes of study data
label_col	variable attribute the name of the column in the metadata with labels of variables
check_param	enum any | location | proportion. Which type of check should be conducted (if possible): a check on the location of the mean or median value of the study data, a check on proportions of categories, or either of them if the necessary metadata is available.
plot_ranges	logical Should the plot show ranges and results from the data quality checks? (default: TRUE)
flip_mode	enum default | flip | noflip | auto. Should the plot be in default orientation, flipped, not flipped or auto-flipped. Not all options are always supported. In general, this con be controlled by setting the `roptions(dataquieR.flip_mode = ...)`. If called from `dq_report`, you can also pass `flip_mode` to all function calls or set them specifically using `specific_args`.

Value

A list with:

- `SummaryTable`: data.frame containing data quality checks for "Unexpected location" (`FLG_acc_ud_loc`) and "Unexpected proportion" (`FLG_acc_ud_prop`) for each response variable in `resp_vars`.
- `SummaryData`: a data.frame containing data quality checks for "Unexpected location" and / or "Unexpected proportion" for a report
- `SummaryPlotList`: list of ggplots for each response variable in `resp_vars`.

Algorithm of this implementation:

- If no response variable is defined, select all variables of type float or integer in the study data.
- Remove missing codes from the study data (if defined in the metadata).
- Remove measurements deviating from (hard) limits defined in the metadata (if defined).

- Exclude variables containing only NA or only one unique value (excluding NAs).
- Perform check for "Unexpected location" if defined in the metadata (needs a LOCATION_METRIC (mean or median) and LOCATION_RANGE (range of expected values for the mean and median, respectively)).
- Perform check for "Unexpected proportion" if defined in the metadata (needs PROPORTION_RANGE (range of expected values for the proportions of the categories)).
- Plot histogram(s).
- If group_vars is specified by the user, distributions within group-wise ecdf are presented.

See Also

- acc_distributions
- Online Documentation

`acc_distributions_prop`

Plots and checks for distributions – Proportion

Description

Data quality indicator checks "Unexpected location" and "Unexpected proportion" with histograms and, if a grouping variable is included, plots of empirical cumulative distributions for the subgroups.

Usage

```
acc_distributions_prop(
  resp_vars = NULL,
  study_data,
  meta_data,
  label_col,
  check_param = "proportion",
  plot_ranges = TRUE,
  flip_mode = "noflip"
)
```

Arguments

resp_vars	variable list the names of the measurement variables
study_data	data.frame the data frame that contains the measurements
meta_data	data.frame the data frame that contains metadata attributes of study data
label_col	variable attribute the name of the column in the metadata with labels of variables
check_param	enum any | location | proportion. Which type of check should be conducted (if possible): a check on the location of the mean or median value of the study data, a check on proportions of categories, or either of them if the necessary metadata is available.

plot_ranges — logical Should the plot show ranges and results from the data quality checks? (default: TRUE)

flip_mode — enum default | flip | noflip | auto. Should the plot be in default orientation, flipped, not flipped or auto-flipped. Not all options are always supported. In general, this con be controlled by setting the `roptions(dataquieR.flip_mode = ...)`. If called from `dq_report`, you can also pass `flip_mode` to all function calls or set them specifically using `specific_args`.

Value

A list with:

- `SummaryTable`: data.frame containing data quality checks for "Unexpected location" (`FLG_acc_ud_loc`) and "Unexpected proportion" (`FLG_acc_ud_prop`) for each response variable in `resp_vars`.
- `SummaryData`: a data.frame containing data quality checks for "Unexpected location" and / or "Unexpected proportion" for a report
- `SummaryPlotList`: list of ggplots for each response variable in `resp_vars`.

Algorithm of this implementation:

- If no response variable is defined, select all variables of type float or integer in the study data.
- Remove missing codes from the study data (if defined in the metadata).
- Remove measurements deviating from (hard) limits defined in the metadata (if defined).
- Exclude variables containing only `NA` or only one unique value (excluding `NA`s).
- Perform check for "Unexpected location" if defined in the metadata (needs a LOCATION_METRIC (mean or median) and LOCATION_RANGE (range of expected values for the mean and median, respectively)).
- Perform check for "Unexpected proportion" if defined in the metadata (needs PROPORTION_RANGE (range of expected values for the proportions of the categories)).
- Plot histogram(s).
- If group_vars is specified by the user, distributions within group-wise ecdf are presented.

See Also

- acc_distributions
- Online Documentation

`acc_distributions_prop_ecdf` *Plots and checks for distributions – Proportion, ECDF*

Description

Data quality indicator checks "Unexpected location" and "Unexpected proportion" with histograms and, if a grouping variable is included, plots of empirical cumulative distributions for the subgroups.

Usage

```
acc_distributions_prop_ecdf(
  resp_vars = NULL,
  group_vars = NULL,
  study_data,
  meta_data,
  label_col,
  check_param = "proportion",
  plot_ranges = TRUE,
  flip_mode = "noflip"
)
```

Arguments

`resp_vars`	variable list the names of the measurement variables
`group_vars`	variable list the name of the observer, device or reader variable
`study_data`	data.frame the data frame that contains the measurements
`meta_data`	data.frame the data frame that contains metadata attributes of study data
`label_col`	variable attribute the name of the column in the metadata with labels of variables
`check_param`	enum any | location | proportion. Which type of check should be conducted (if possible): a check on the location of the mean or median value of the study data, a check on proportions of categories, or either of them if the necessary metadata is available.
`plot_ranges`	logical Should the plot show ranges and results from the data quality checks? (default: TRUE)
`flip_mode`	enum default | flip | noflip | auto. Should the plot be in default orientation, flipped, not flipped or auto-flipped. Not all options are always supported. In general, this con be controlled by setting the `roptions(dataquieR.flip_mode = ...)`. If called from `dq_report`, you can also pass `flip_mode` to all function calls or set them specifically using `specific_args`.

Value

A list with:

- SummaryTable: data.frame containing data quality checks for "Unexpected location" (FLG_acc_ud_loc) and "Unexpected proportion" (FLG_acc_ud_prop) for each response variable in resp_vars.
- SummaryData: a data.frame containing data quality checks for "Unexpected location" and / or "Unexpected proportion" for a report
- SummaryPlotList: list of ggplots for each response variable in resp_vars.

Algorithm of this implementation:

- If no response variable is defined, select all variables of type float or integer in the study data.
- Remove missing codes from the study data (if defined in the metadata).
- Remove measurements deviating from (hard) limits defined in the metadata (if defined).
- Exclude variables containing only NA or only one unique value (excluding NAs).
- Perform check for "Unexpected location" if defined in the metadata (needs a LOCATION_METRIC (mean or median) and LOCATION_RANGE (range of expected values for the mean and median, respectively)).
- Perform check for "Unexpected proportion" if defined in the metadata (needs PROPORTION_RANGE (range of expected values for the proportions of the categories)).
- Plot histogram(s).
- If group_vars is specified by the user, distributions within group-wise ecdf are presented.

See Also

- acc_distributions
- Online Documentation

---

acc_end_digits *Extension of acc_shape_or_scale to examine uniform distributions of end digits*

Description

This implementation contrasts the empirical distribution of a measurement variables against assumed distributions. The approach is adapted from the idea of rootograms (Tukey (1977)) which is also applicable for count data (Kleiber and Zeileis (2016)).

Usage

```
acc_end_digits(resp_vars = NULL, study_data, meta_data, label_col = VAR_NAMES)
```

Arguments

resp_vars	variable the names of the measurement variables, mandatory
study_data	data.frame the data frame that contains the measurements
meta_data	data.frame the data frame that contains metadata attributes of study data
label_col	variable attribute the name of the column in the metadata with labels of variables

Value

a list with:

- SummaryData: data frame underlying the plot
- SummaryPlot: ggplot2 distribution plot comparing expected with observed distribution

ALGORITHM OF THIS IMPLEMENTATION:

- This implementation is restricted to data of type float or integer.
- Missing codes are removed from resp_vars (if defined in the metadata)
- The user must specify the column of the metadata containing probability distribution (currently only: normal, uniform, gamma)
- Parameters of each distribution can be estimated from the data or are specified by the user
- A histogram-like plot contrasts the empirical vs. the technical distribution

See Also

Online Documentation

---

acc_loess *Smoothes and plots adjusted longitudinal measurements*

Description

The following R implementation executes calculations for quality indicator Unexpected distribution wrt location (link). Local regression (LOESS) is a versatile statistical method to explore an averaged course of time series measurements (Cleveland, Devlin, and Grosse 1988). In context of epidemiological data, repeated measurements using the same measurement device or by the same examiner can be considered a time series. LOESS allows to explore changes in these measurements over time.

Usage

```
acc_loess(
  resp_vars,
  group_vars,
  time_vars,
  co_vars = NULL,
  min_obs_in_subgroup,
  label_col = NULL,
  study_data,
  meta_data,
  resolution = 180,
  se_line = list(color = "red", linetype = 2),
  plot_data_time,
  plot_format = "AUTO"
)
```

Arguments

`resp_vars`	variable the name of the continuous measurement variable
`group_vars`	variable the name of the observer, device or reader variable
`time_vars`	variable a variable identifying the variable with the time of measurement
`co_vars`	variable list a vector of covariables, e.g. age and sex for adjustment. Can be NULL (default) for no adjustment.
`min_obs_in_subgroup`	integer from=0. optional argument if `group_vars` are used. This argument specifies the minimum number of observations that is required to include a subgroup (level) of the group variable named by `group_vars` in the analysis. Subgroups with less observations are excluded. The default is `30`.
`label_col`	variable attribute the name of the column in the metadata with labels of variables
`study_data`	data.frame the data frame that contains the measurements
`meta_data`	data.frame the data frame that contains metadata attributes of study data
`resolution`	numeric how many timepoints have a standard error estimation
`se_line`	list standard error estimator line style, as arguments passed to `ggplot2::geom_line()`
`plot_data_time`	logical mark times with data values (caution, there may be many marks)
`plot_format`	enum AUTO | COMBINED | FACETS | BOTH. Return the LOESS plot as a combined plot or as facets plots one per group. BOTH will return both plot variants, AUTO will decide based on the number of observers.

Details

If `plot_data_time` is not set, it will be selected based on the number of data points per group: If more than 4000 points would be plotted for at least one group, the > 4000 marks will not be plotted.

Limitations

The application of LOESS usually requires model fitting, i.e. the smoothness of a model is subject to a smoothing parameter (span). Particularly in the presence of interval-based missing data

(USR_181), high variability of measurements combined with a low number of observations in one level of the group_vars the fit to the data may be distorted. Since our approach handles data without knowledge of such underlying characteristics, finding the best fit is complicated if computational costs should be minimal. The default of LOESS in R uses a span 0.75 which provides in most cases reasonable fits. The function above increases the fit to the data automatically if the minimum of observations in one level of the group_vars is higher than n=30.

Value

a list with:

- SummaryPlotList: list with two plots:
  - Loess_fits_facets: ggplot2 LOESS plot provides panels for each subject/object. The plot contains LOESS-smoothed curves for each level of the group_vars. The red dashed lines represent the confidence interval of a LOESS curve for the whole data.
  - Loess_fits_combined: ggplot2 LOESS plot combines all curves into one panel and is obtained by myloess$Loess_fits_combined. Given a low number of levels in the group_vars this plot eases comparisons. However, if number increases this plot may be too crowded and unclear.

See Also

Online Documentation

---

acc_margins *Estimate marginal means, see emmeans::emmeans*

Description

margins does calculations for quality indicator Unexpected distribution wrt location (link). Therefore we pursue a combined approach of descriptive and model-based statistics to investigate differences across the levels of an auxiliary variable.

CAT: Unexpected distribution w.r.t. location

Marginal means

Marginal means rests on model based results, i.e. a significantly different marginal mean depends on sample size. Particularly in large studies, small and irrelevant differences may become significant. The contrary holds if sample size is low.

Usage

```
acc_margins(
  resp_vars = NULL,
  group_vars = NULL,
  co_vars = NULL,
  threshold_type = NULL,
  threshold_value,
```

```
  min_obs_in_subgroup,
  study_data,
  meta_data,
  label_col
)
```

Arguments

resp_vars	variable the name of the continuous measurement variable
group_vars	variable list len=1-1. the name of the observer, device or reader variable
co_vars	variable list a vector of covariables, e.g. age and sex for adjustment
threshold_type	enum empirical | user | none. In case empirical is chosen a multiplier of the scale measure is used, in case of user a value of the mean or probability (binary data) has to be defined see Implementation and use of thresholds. In case of none, no thresholds are displayed and no flagging of unusual group levels is applied.
threshold_value	numeric a multiplier or absolute value see Implementation and use of thresholds
min_obs_in_subgroup	integer from=0. optional argument if a "group_var" is used. This argument specifies the minimum no. of observations that is required to include a subgroup (level) of the "group_var" in the analysis. Subgroups with less observations are excluded. The default is 5.
study_data	data.frame the data frame that contains the measurements
meta_data	data.frame the data frame that contains metadata attributes of study data
label_col	variable attribute the name of the column in the metadata with labels of variables

Details

Limitations

Selecting the appropriate distribution is complex. Dozens of continuous, discrete or mixed distributions are conceivable in the context of epidemiological data. Their exact exploration is beyond the scope of this data quality approach. The function above uses the help function util_dist_selection which discriminates four cases:

- continuous data
- binary data
- count data with <= 20 categories
- count data with > 20 categories

Nonetheless, only three different plot types are generated. The fourth case is treated as continuous data. This is in fact a coarsening of the original data but for the purpose of clarity this approach is chosen.

Value

a list with:

- SummaryTable: data frame underlying the plot
- SummaryData: data frame
- SummaryPlot: ggplot2 margins plot

See Also

Online Documentation

Examples

```
## Not run:
# runs spuriously slow on rhub
load(system.file("extdata/study_data.RData", package = "dataquieR"))
load(system.file("extdata/meta_data.RData", package = "dataquieR"))
co_vars <- c("AGE_0")
label_col <- LABEL
rvs <- c("DBP_0")
group_vars <- prep_map_labels(rvs, meta_data = meta_data, from = label_col,
  to = VAR_NAMES)
group_vars <- prep_map_labels(group_vars, meta_data = meta_data,
  to = GROUP_VAR_OBSERVER)
group_vars <- prep_map_labels(group_vars, meta_data = meta_data)
acc_margins(resp_vars = rvs,
            study_data = study_data,
            meta_data = meta_data,
            group_vars = group_vars,
            label_col = label_col,
            co_vars = co_vars)

## End(Not run)
```

---

acc_multivariate_outlier

Calculate and plot Mahalanobis distances

Description

A standard tool to detect multivariate outliers is the Mahalanobis distance. This approach is very helpful for the interpretation of the plausibility of a measurement given the value of another. In this approach the Mahalanobis distance is used as a univariate measure itself. We apply the same rules for the identification of outliers as in univariate outliers:

- the classical approach from Tukey: $1.5 * IQR$ from the 1st (Q_{25}) or 3rd (Q_{75}) quartile.
- the $6 * \sigma$ approach, i.e. any measurement of the Mahalanobis distance not in the interval of $\bar{x} \pm 3 * \sigma$ is considered an outlier.

- the approach from Hubert for skewed distributions which is embedded in the R package **robustbase**
- a completely heuristic approach named σ-gap.

For further details, please see the vignette for univariate outlier.

Usage

```
acc_multivariate_outlier(
  variable_group = NULL,
  id_vars = NULL,
  label_col,
  n_rules = 4,
  max_non_outliers_plot = 10000,
  criteria = c("tukey", "sixsigma", "hubert", "sigmagap"),
  study_data,
  meta_data
)
```

Arguments

`variable_group`	variable list the names of the continuous measurement variables building a group, for that multivariate outliers make sense.
`id_vars`	variable optional, an ID variable of the study data. If not specified row numbers are used.
`label_col`	variable attribute the name of the column in the metadata with labels of variables
`n_rules`	numeric from=1 to=4. the no. of rules that must be violated to classify as outlier
`max_non_outliers_plot`	integer from=0. Maximum number of non-outlier points to be plot. If more points exist, a subsample will be plotted only. Note, that sampling is not deterministic.
`criteria`	set tukey | sixsigma | hubert | sigmagap. a vector with methods to be used for detecting outliers.
`study_data`	data.frame the data frame that contains the measurements
`meta_data`	data.frame the data frame that contains metadata attributes of study data

Value

a list with:

- `SummaryTable`: data.frame underlying the plot
- `SummaryPlot`: ggplot2 outlier plot
- `FlaggedStudyData` data.frame contains the original data frame with the additional columns `tukey`, `sixsigma`, `hubert`, and `sigmagap`. Every observation is coded 0 if no outlier was detected in the respective column and 1 if an outlier was detected. This can be used to exclude observations with outliers.

ALGORITHM OF THIS IMPLEMENTATION:

- Implementation is restricted to variables of type float
- Remove missing codes from the study data (if defined in the metadata)
- The covariance matrix is estimated for all variables from `variable_group`
- The Mahalanobis distance of each observation is calculated $MD_i^2 = (x_i - \mu)^T \Sigma^{-1} (x_i - \mu)$
- The four rules mentioned above are applied on this distance for each observation in the study data
- An output data frame is generated that flags each outlier
- A parallel coordinate plot indicates respective outliers

List function.

See Also

Online Documentation

`acc_robust_univariate_outlier`

Identify univariate outliers by four different approaches

Description

A classical but still popular approach to detect univariate outlier is the boxplot method introduced by Tukey 1977. The boxplot is a simple graphical tool to display information about continuous univariate data (e.g., median, lower and upper quartile). Outliers are defined as values deviating more than $1.5 \times IQR$ from the 1st (Q25) or 3rd (Q75) quartile. The strength of Tukey's method is that it makes no distributional assumptions and thus is also applicable to skewed or non mound-shaped data Marsh and Seo, 2006. Nevertheless, this method tends to identify frequent measurements which are falsely interpreted as true outliers.

A somewhat more conservative approach in terms of symmetric and/or normal distributions is the $6 * \sigma$ approach, i.e. any measurement not in the interval of $mean(x) + / - 3 * \sigma$ is considered an outlier.

Both methods mentioned above are not ideally suited to skewed distributions. As many biomarkers such as laboratory measurements represent in skewed distributions the methods above may be insufficient. The approach of Hubert and Vandervieren 2008 adjusts the boxplot for the skewness of the distribution. This approach is implemented in several R packages such as `robustbase::mc` which is used in this implementation of `dataquieR`.

Another completely heuristic approach is also included to identify outliers. The approach is based on the assumption that the distances between measurements of the same underlying distribution should homogeneous. For comprehension of this approach:

- consider an ordered sequence of all measurements.
- between these measurements all distances are calculated.

- the occurrence of larger distances between two neighboring measurements may than indicate a distortion of the data. For the heuristic definition of a large distance $1 * \sigma$ has been been chosen.

Note, that the plots are not deterministic, because they use ggplot2::geom_jitter.

Usage

```
acc_robust_univariate_outlier(
  resp_vars = NULL,
  label_col,
  study_data,
  meta_data,
  exclude_roles,
  n_rules = length(unique(criteria)),
  max_non_outliers_plot = 10000,
  criteria = c("tukey", "sixsigma", "hubert", "sigmagap")
)
```

Arguments

resp_vars	variable list the name of the continuous measurement variable
label_col	variable attribute the name of the column in the metadata with labels of variables
study_data	data.frame the data frame that contains the measurements
meta_data	data.frame the data frame that contains metadata attributes of study data
exclude_roles	variable roles a character (vector) of variable roles not included
n_rules	integer from=1 to=4. the no. of rules that must be violated to flag a variable as containing outliers. The default is 4, i.e. all.
max_non_outliers_plot	integer from=0. Maximum number of non-outlier points to be plot. If more points exist, a subsample will be plotted only. Note, that sampling is not deterministic.
criteria	set tukey | sixsigma | hubert | sigmagap. a vector with methods to be used for detecting outliers.

Details

Hint: *The function is designed for unimodal data only.*

Value

a list with:

- SummaryTable: data.frame with the columns Variables, Mean, SD, Median, Skewness, Tukey (N), 6-Sigma (N), Hubert (N), Sigma-gap (N), Most likely (N), To low (N), To high (N) Grading
- SummaryPlotList: ggplot univariate outlier plots

ALGORITHM OF THIS IMPLEMENTATION:

- Select all variables of type float in the study data
- Remove missing codes from the study data (if defined in the metadata)
- Remove measurements deviating from limits defined in the metadata
- Identify outlier according to the approaches of Tukey (Tukey 1977), SixSigma (-Bakar et al. 2006), Hubert (Hubert and Vandervieren 2008), and SigmaGap (heuristic)
- A output data frame is generated which indicates the no. of possible outlier, the direction of deviations (to low, to high) for all methods and a summary score which sums up the deviations of the different rules
- A scatter plot is generated for all examined variables, flagging observations according to the no. of violated rules (step 5).

See Also

acc_univariate_outlier

`acc_shape_or_scale` *Compare observed versus expected distributions*

Description

This implementation contrasts the empirical distribution of a measurement variables against assumed distributions. The approach is adapted from the idea of rootograms (Tukey 1977) which is also applicable for count data (Kleiber and Zeileis 2016).

Usage

```
acc_shape_or_scale(
  resp_vars,
  dist_col,
  guess,
  par1,
  par2,
  end_digits,
  label_col,
  study_data,
  meta_data,
  flip_mode = "noflip"
)
```

Arguments

resp_vars	variable the name of the continuous measurement variable
dist_col	variable attribute the name of the variable attribute in meta_data that provides the expected distribution of a study variable
guess	logical estimate parameters
par1	numeric first parameter of the distribution if applicable
par2	numeric second parameter of the distribution if applicable
end_digits	logical internal use. check for end digits preferences
label_col	variable attribute the name of the column in the metadata with labels of variables
study_data	data.frame the data frame that contains the measurements
meta_data	data.frame the data frame that contains metadata attributes of study data
flip_mode	enum default | flip | noflip | auto. Should the plot be in default orientation, flipped, not flipped or auto-flipped. Not all options are always supported. In general, this con be controlled by setting the `roptions(dataquieR.flip_mode = ...)`. If called from `dq_report`, you can also pass `flip_mode` to all function calls or set them specifically using `specific_args`.

Value

a list with:

- `SummaryData`: data.frame underlying the plot
- `SummaryPlot`: ggplot2 probability distribution plot
- `SummaryTable`: data.frame with the columns `Variables` and `GRADING`

ALGORITHM OF THIS IMPLEMENTATION:

- This implementation is restricted to data of type float or integer.
- Missing codes are removed from resp_vars (if defined in the metadata)
- The user must specify the column of the metadata containing probability distribution (currently only: normal, uniform, gamma)
- Parameters of each distribution can be estimated from the data or are specified by the user
- A histogram-like plot contrasts the empirical vs. the technical distribution

See Also

Online Documentation

`acc_univariate_outlier` *Identify univariate outliers by four different approaches*

Description

A classical but still popular approach to detect univariate outlier is the boxplot method introduced by Tukey 1977. The boxplot is a simple graphical tool to display information about continuous univariate data (e.g., median, lower and upper quartile). Outliers are defined as values deviating more than $1.5 \times IQR$ from the 1st (Q25) or 3rd (Q75) quartile. The strength of Tukey's method is that it makes no distributional assumptions and thus is also applicable to skewed or non mound-shaped data Marsh and Seo, 2006. Nevertheless, this method tends to identify frequent measurements which are falsely interpreted as true outliers.

A somewhat more conservative approach in terms of symmetric and/or normal distributions is the $6 * \sigma$ approach, i.e. any measurement not in the interval of $mean(x) + / - 3 * \sigma$ is considered an outlier.

Both methods mentioned above are not ideally suited to skewed distributions. As many biomarkers such as laboratory measurements represent in skewed distributions the methods above may be insufficient. The approach of Hubert and Vandervieren 2008 adjusts the boxplot for the skewness of the distribution. This approach is implemented in several R packages such as `robustbase::mc` which is used in this implementation of `dataquieR`.

Another completely heuristic approach is also included to identify outliers. The approach is based on the assumption that the distances between measurements of the same underlying distribution should homogeneous. For comprehension of this approach:

- consider an ordered sequence of all measurements.
- between these measurements all distances are calculated.
- the occurrence of larger distances between two neighboring measurements may than indicate a distortion of the data. For the heuristic definition of a large distance $1 * \sigma$ has been been chosen.

Note, that the plots are not deterministic, because they use ggplot2::geom_jitter.

Usage

```
acc_univariate_outlier(
  resp_vars = NULL,
  label_col,
  study_data,
  meta_data,
  exclude_roles,
  n_rules = length(unique(criteria)),
  max_non_outliers_plot = 10000,
  criteria = c("tukey", "sixsigma", "hubert", "sigmagap")
)
```

Arguments

resp_vars	variable list the name of the continuous measurement variable
label_col	variable attribute the name of the column in the metadata with labels of variables
study_data	data.frame the data frame that contains the measurements
meta_data	data.frame the data frame that contains metadata attributes of study data
exclude_roles	variable roles a character (vector) of variable roles not included
n_rules	integer from=1 to=4. the no. of rules that must be violated to flag a variable as containing outliers. The default is 4, i.e. all.
max_non_outliers_plot	integer from=0. Maximum number of non-outlier points to be plot. If more points exist, a subsample will be plotted only. Note, that sampling is not deterministic.
criteria	set tukey | sixsigma | hubert | sigmagap. a vector with methods to be used for detecting outliers.

Details

Hint: *The function is designed for unimodal data only.*

Value

a list with:

- SummaryTable: data.frame with the columns Variables, Mean, SD, Median, Skewness, Tukey (N), 6-Sigma (N), Hubert (N), Sigma-gap (N), Most likely (N), To low (N), To high (N) Grading
- SummaryPlotList: ggplot univariate outlier plots

ALGORITHM OF THIS IMPLEMENTATION:

- Select all variables of type float in the study data
- Remove missing codes from the study data (if defined in the metadata)
- Remove measurements deviating from limits defined in the metadata
- Identify outlier according to the approaches of Tukey (Tukey 1977), SixSigma (-Bakar et al. 2006), Hubert (Hubert and Vandervieren 2008), and SigmaGap (heuristic)
- A output data frame is generated which indicates the no. of possible outlier, the direction of deviations (to low, to high) for all methods and a summary score which sums up the deviations of the different rules
- A scatter plot is generated for all examined variables, flagging observations according to the no. of violated rules (step 5).

See Also

- acc_robust_univariate_outlier
- Online Documentation

`acc_varcomp` *Estimates variance components*

Description

Variance based models and intraclass correlations (ICC) are approaches to examine the impact of so-called process variables on the measurements. This implementation is model-based.

NB: The term ICC is frequently used to describe the agreement between different observers, examiners or even devices. In respective settings a good agreement is pursued. ICC-values can vary between `[-1;1]` and an ICC close to 1 is desired (Koo and Li 2016, Müller and Büttner 1994).

However, in multi-level analysis the ICC is interpreted differently. Please see Snijders et al. (Sniders and Bosker 1999). In this context the proportion of variance explained by respective group levels indicate an influence of (at least one) level of the respective group_vars. An ICC close to 0 is desired.

Usage

```
acc_varcomp(
  resp_vars = NULL,
  group_vars,
  co_vars = NULL,
  min_obs_in_subgroup = 30,
  min_subgroups = 5,
  label_col = NULL,
  threshold_value = 0.05,
  study_data,
  meta_data
)
```

Arguments

`resp_vars`	variable list the names of the continuous measurement variables
`group_vars`	variable list the names of the resp. observer, device or reader variables
`co_vars`	variable list a vector of covariables, e.g. age and sex for adjustment
`min_obs_in_subgroup`	integer from=0. optional argument if a "group_var" is used. This argument specifies the minimum no. of observations that is required to include a subgroup (level) of the "group_var" in the analysis. Subgroups with fewer observations are excluded. The default is 30.
`min_subgroups`	integer from=0. optional argument if a "group_var" is used. This argument specifies the minimum no. of subgroups (levels) included "group_var". If the variable defined in "group_var" has fewer subgroups it is not used for analysis. The default is 5.
`label_col`	variable attribute the name of the column in the metadata with labels of variables

threshold_value
: numeric from=0 to=1. a numerical value ranging from 0-1

study_data
: data.frame the data frame that contains the measurements

meta_data
: data.frame the data frame that contains metadata attributes of study data

Value

a list with:

- SummaryTable: data frame with ICCs per rvs
- ScalarValue_max_icc: maximum variance contribution value by group_vars
- ScalarValue_argmax_icc: variable with maximum variance contribution by group_vars

ALGORITHM OF THIS IMPLEMENTATION:

- This implementation is yet restricted to data of type float.
- Missing codes are removed from resp_vars (if defined in the metadata)
- Deviations from limits, as defined in the metadata, are removed
- A linear mixed-effects model is estimated for resp_vars using co_vars and group_vars for adjustment.
- An output data frame is generated for group_vars indicating the ICC.

See Also

Online Documentation

Examples

```
## Not run:
# runs spuriously slow on rhub
load(system.file("extdata/study_data.RData", package = "dataquieR"))
load(system.file("extdata/meta_data.RData", package = "dataquieR"))
co_vars <- c("SEX_0", "AGE_0")
min_obs_in_subgroup <- 30
min_subgroups <- 3
label_col <- LABEL
rvs <- c("DBP_0", "SBP_0")
group_vars <- prep_map_labels(rvs, meta_data = meta_data, from = label_col,
  to = VAR_NAMES)
group_vars <- prep_map_labels(group_vars, meta_data = meta_data,
  to = GROUP_VAR_OBSERVER)
group_vars <- prep_map_labels(group_vars, meta_data = meta_data)
acc_varcomp(
  resp_vars = rvs, group_vars = group_vars, co_vars = co_vars,
  min_obs_in_subgroup = min_obs_in_subgroup,
  min_subgroups = min_subgroups, label_col = label_col,
  study_data = study_data, meta_data = meta_data
)
```

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

`as.data.frame.dataquieR_resultset`
Convert a full `dataquieR` *report to a* `data.frame`

Description

converts a dataquieR report to a data.frame. Intended for use in pipelines.

Usage

```
## S3 method for class 'dataquieR_resultset'
as.data.frame(x, ...)
```

Arguments

`x`	dataquieR report
`...`	not used

Value

a data.frame with one row per indicator call, one column `implementationform` naming the called indicator function, one column per function argument and one additional column containing the results of each call as a list.

`as.list.dataquieR_resultset`
Convert a full `dataquieR` *report to a* `list`

Description

converts a dataquieR report to a list. Intended for use in pipelines.

Usage

```
## S3 method for class 'dataquieR_resultset'
as.list(x, ...)
```

Arguments

`x`	dataquieR report
`...`	arguments passed to pipeline_recursive_result

Value

a list with one element per indicator call. Each element is an encapsulated sub-list as described in pipeline_recursive_result

---

cause_label_df *Data frame with labels for missing- and jump-codes*

Description

data.frame with the following columns:

- CODE_VALUE: numeric Missing code (the number)
- CODE_LABEL: character a label for the missing code
- CODE_CLASS: enum JUMP | MISSING. Class of the missing code.
- resp_vars: character optional, if a missing code is specific for some variables, it is listed for each such variable with one entry in resp_vars, If NA, the code is assumed shared among all variables. For v1.0 metadata, you need to refer to VAR_NAMES here.

---

check_table *Data frame with contradiction rules*

Description

Two versions exist, the newer one is used by con_contradictions_redcap and is described here., the older one used by con_contradictions_redcap is described here.

---

com_item_missingness *Summarize missingness columnwise (in variable)*

Description

Item-Missingness (also referred to as item nonresponse (De Leeuw et al. 2003)) describes the missingness of single values, e.g. blanks or empty data cells in a data set. Item-Missingness occurs for example in case a respondent does not provide information for a certain question, a question is overlooked by accident, a programming failure occurs or a provided answer were missed while entering the data.

Usage

```
com_item_missingness(
  study_data,
  meta_data,
  resp_vars = NULL,
  label_col,
  show_causes = TRUE,
  cause_label_df,
  include_sysmiss = TRUE,
  threshold_value,
  suppressWarnings = FALSE,
  assume_consistent_codes = TRUE,
  expand_codes = assume_consistent_codes,
  drop_levels = TRUE,
  expected_observations = c("HIERARCHY", "ALL", "SEGMENT"),
  pretty_print = TRUE
)
```

Arguments

study_data	data.frame the data frame that contains the measurements
meta_data	data.frame the data frame that contains metadata attributes of study data
resp_vars	variable list the name of the measurement variables
label_col	variable attribute the name of the column in the metadata with labels of variables
show_causes	logical if TRUE, then the distribution of missing codes is shown
cause_label_df	data.frame missing code table. If missing codes have labels the respective data frame can be specified here or in the metadata as assignments, see cause_label_df
include_sysmiss	logical Optional, if TRUE system missingness (NAs) is evaluated in the summary plot
threshold_value	numeric from=0 to=100. a numerical value ranging from 0-100
suppressWarnings	logical warn about consistency issues with missing and jump lists
assume_consistent_codes	logical if TRUE and no labels are given and the same missing/jump code is used for more than one variable, the labels assigned for this code are treated as being be the same for all variables.
expand_codes	logical if TRUE, code labels are copied from other variables, if the code is the same and the label is set somewhere
drop_levels	logical if TRUE, do not display unused missing codes in the figure legend.
expected_observations	enum HIERARCHY | ALL | SEGMENT. If ALL, all observations are expected to comprise all study segments. If SEGMENT, the PART_VAR is expected to point to a variable with values of 0 and 1, indicating whether the variable was

	expected to be observed for each data row. If HIERARCHY, this is also checked recursively, so, if a variable points to such a participation variable, and that other variable does has also a PART_VAR entry pointing to a variable, the observation of the initial variable is only expected, if both segment variables are 1.
pretty_print	logical If FALSE, produce a table that can easily be processed further, because some cells feature two numbers (absolute and percentage) otherwise.

Value

a list with:

- SummaryTable: data frame about item missingness per response variable
- SummaryPlot: ggplot2 heatmap plot, if show_causes was TRUE
- ReportSummaryTable: data frame underlying SummaryPlot

ALGORITHM OF THIS IMPLEMENTATION:

- Lists of missing codes and, if applicable, jump codes are selected from the metadata
- The no. of system missings (NA) in each variable is calculated
- The no. of used missing codes is calculated for each variable
- The no. of used jump codes is calculated for each variable
- Two result dataframes (1: on the level of observations, 2: a summary for each variable) are generated
- *OPTIONAL:* if show_causes is selected, one summary plot for all resp_vars is provided

See Also

Online Documentation

---

com_qualified_item_missingness

Compute Indicators for Qualified Item Missingness

Description

Compute Indicators for Qualified Item Missingness

Usage

```
com_qualified_item_missingness(
  resp_vars,
  study_data,
  meta_data,
  label_col = NULL,
  expected_observations = c("HIERARCHY", "ALL", "SEGMENT")
)
```

Arguments

resp_vars	variable list the name of the measurement variables
study_data	data.frame the data frame that contains the measurements
meta_data	data.frame the data frame that contains metadata attributes of study data
label_col	variable attribute the name of the column in the metadata with labels of variables
expected_observations	enum HIERARCHY | ALL | SEGMENT. Report the number of observations expected using the old PART_VAR concept. See com_item_missingness for an explanation.

Value

list list with entries:

Examples

```
## Not run:
prep_load_workbook_like_file("inst/extdata/Metadata_example_v3-6.xlsx")
clean <- prep_get_data_frame("item_level")
clean <- subset(clean, `Metadata name` == "Example" &
  !dataquieR:::util_empty(VAR_NAMES))
clean$`Metadata name` <- NULL
clean[, "MISSING_LIST_TABLE"] <- "missing_matchtable1"
prep_add_data_frames(item_level = clean)
clean <- prep_get_data_frame("missing_matchtable1")
clean <- clean[clean$`Metadata name` == "Example", , FALSE]
clean <-
  clean[suppressWarnings(as.character(as.integer(clean$CODE_VALUE)) ==
    as.character(clean$CODE_VALUE)), , FALSE]
clean$CODE_VALUE <- as.integer(clean$CODE_VALUE)
clean <- clean[!is.na(clean$`Metadata name`), , FALSE]
clean$`Metadata name` <- NULL
prep_add_data_frames(missing_matchtable1 = clean)
ship <- prep_get_data_frame("ship")
number_of_mis <- ceiling(nrow(ship) / 20)
resp_vars <- sample(colnames(ship), ceiling(ncol(ship) / 20), FALSE)
mistab <- prep_get_data_frame("missing_matchtable1")
valid_replacement_codes <-
  mistab[mistab$CODE_INTERPRET != "P", "CODE_VALUE",
    drop =
    TRUE] # sample only replacement codes on item level. P uses the actual
          # values
for (rv in resp_vars) {
  values <- sample(as.numeric(valid_replacement_codes), number_of_mis,
    replace = TRUE)
  if (inherits(ship[[rv]], "POSIXct")) {
    values <- as.POSIXct(values, origin = min(as.POSIXct(Sys.Date()), 0))
  }
  ship[sample(seq_len(nrow(ship)), number_of_mis, replace = FALSE), rv] <-
    values
```

```
}
com_qualified_item_missingness(resp_vars = NULL, ship, "item_level", LABEL)
com_qualified_item_missingness(resp_vars = "Diabetes Age onset", ship,
  "item_level", LABEL)
com_qualified_item_missingness(resp_vars = NULL, "study_data", "meta_data",
  LABEL)
study_data <- ship
meta_data <- prep_get_data_frame("item_level")
label <- LABEL

## End(Not run)
```

`com_qualified_segment_missingness` *Compute Indicators for Qualified Segment Missingness*

Description

Compute Indicators for Qualified Segment Missingness

Usage

```
com_qualified_segment_missingness(
  study_data,
  meta_data,
  label_col = NULL,
  meta_data_segment,
  expected_observations = c("HIERARCHY", "ALL", "SEGMENT")
)
```

Arguments

`study_data`	data.frame the data frame that contains the measurements
`meta_data`	data.frame the data frame that contains metadata attributes of study data
`label_col`	variable attribute the name of the column in the metadata with labels of variables
`meta_data_segment`	data.frame Segment level metadata
`expected_observations`	enum HIERARCHY | ALL | SEGMENT. Report the number of observations expected using the old `PART_VAR` concept. See com_item_missingness for an explanation.

Value

list list with entries:

Examples

```
## Not run:
prep_load_workbook_like_file("inst/extdata/Metadata_example_v3-6.xlsx")
clean <- prep_get_data_frame("item_level")
clean <- subset(clean, `Metadata name` == "Example" &
  !dataquieR:::util_empty(VAR_NAMES))
clean$`Metadata name` <- NULL
clean <- rbind(clean, data.frame(
VAR_NAMES = "part_seg_interview",
LABEL = "PART_SEG_INTERVIEW",
VALUE_LABELS = NA_character_,
DATA_TYPE = "integer",
UNIT = NA_character_,
SCALE_LEVEL = NA_character_,
STANDARDIZED_VOCABULARY_TABLE = NA_character_,
MISSING_LIST = NA_character_,
JUMP_LIST = NA_character_,
MISSING_LIST_TABLE = "missing_matchtable1",
MISSING_CODED = NA,
HARD_LIMITS = NA_character_,
SOFT_LIMITS = NA_character_,
DETECTION_LIMITS = NA_character_,
CONTRADICTIONS = NA_character_,
DISTRIBUTION = NA_character_,
DECIMALS = NA_character_,
DATA_ENTRY_TYPE = NA_character_,
 GROUP_VAR_OBSERVER = NA_character_,
 GROUP_VAR_DEVICE = NA_character_,
 TIME_VAR = NA_character_,
 STUDY_SEGMENT = NA_character_,
 PARTICIPATION_VAR = NA_character_,
 VARIABLE_ROLE = NA_character_,
 VARIABLE_ORDER = NA_character_,
 ELEMENT_HOMOGENITY_CHECKTYPE = NA_character_,
 UNIVARIATE_OUTLIER_CHECKTYPE = NA_character_,
 LOCATION_METRIC = NA_character_,
 LOCATION_RANGE = NA_character_,
 PROPORTION_RANGE = NA_character_,
 KEY_REPEATED_MEASURES = NA_character_,
 REPEATED_MEASURES_GOLDSTANDARD = NA_character_,
 CO_VARS = NA_character_
))
prep_add_data_frames(item_level = clean)
clean <- prep_get_data_frame("segment_level")
clean <- subset(clean, `Metadata name` == "Example" &
  !dataquieR:::util_empty(STUDY_SEGMENT))
clean$`Metadata name` <- NULL
prep_add_data_frames(segment_level = clean)
clean <- prep_get_data_frame("missing_matchtable1")
clean <- clean[clean$`Metadata name` == "Example", , FALSE]
clean <- clean[suppressWarnings(as.character(as.integer(clean$CODE_VALUE)) ==
  as.character(clean$CODE_VALUE)), , FALSE]
```

```
clean$CODE_VALUE <- as.integer(clean$CODE_VALUE)
clean <- clean[!is.na(clean$`Metadata name`), , FALSE]
clean$`Metadata name` <- NULL
prep_add_data_frames(missing_matchtable1 = clean)
ship <- prep_get_data_frame("ship")
ship$part_seg_interview <-
  sample(as.numeric(prep_get_data_frame("missing_matchtable1")$CODE_VALUE),
    nrow(ship), replace = TRUE)
com_qualified_segment_missingness(ship, "item_level",
        LABEL, "segment_level")

## End(Not run)
```

`com_segment_missingness` *Summarizes missingness for individuals in specific segments*

Description

This implementation can be applied in two use cases::

1. participation in study segments is not recorded by respective variables, e.g. a participant's refusal to attend a specific examination is not recorded.
2. participation in study segments is recorded by respective variables.

Use case *(1)* will be common in smaller studies. For the calculation of segment missingness it is assumed that study variables are nested in respective segments. This structure must be specified in the static metadata. The R-function identifies all variables within each segment and returns TRUE if all variables within a segment are missing, otherwise FALSE.

Use case *(2)* assumes a more complex structure of study data and meta data. The study data comprise so-called intro-variables (either TRUE/FALSE or codes for non-participation). The column `PART_VAR` in the metadata is filled by variable-IDs indicating for each variable the respective intro-variable. This structure has the benefit that subsequent calculation of item missingness obtains correct denominators for the calculation of missingness rates.

Usage

```
com_segment_missingness(
  study_data,
  meta_data,
  group_vars = NULL,
  strata_vars = NULL,
  label_col,
  threshold_value,
  direction,
  color_gradient_direction,
  expected_observations = c("HIERARCHY", "ALL", "SEGMENT"),
  exclude_roles = "process"
)
```

Arguments

`study_data`	data.frame the data frame that contains the measurements
`meta_data`	data.frame the data frame that contains metadata attributes of study data
`group_vars`	variable the name of a variable used for grouping, defaults to *NULL* for not grouping output
`strata_vars`	variable the name of a variable used for stratification, defaults to NULL for not grouping output
`label_col`	variable attribute the name of the column in the metadata with labels of variables
`threshold_value`	numeric from=0 to=100. a numerical value ranging from 0-100
`direction`	enum low | high. "high" or "low", i.e. are deviations above/below the threshold critical. This argument is deprecated and replaced by *color_gradient_direction*.
`color_gradient_direction`	enum above | below. "above" or "below", i.e. are deviations above or below the threshold critical? (default: above)
`expected_observations`	enum HIERARCHY | ALL | SEGMENT. If ALL, all observations are expected to comprise all study segments. If SEGMENT, the `PART_VAR` is expected to point to a variable with values of 0 and 1, indicating whether the variable was expected to be observed for each data row. If HIERARCHY, this is also checked recursively, so, if a variable points to such a participation variable, and that other variable does has also a `PART_VAR` entry pointing to a variable, the observation of the initial variable is only expected, if both segment variables are 1.
`exclude_roles`	variable roles a character (vector) of variable roles not included

Details

Implementation and use of thresholds:

This implementation uses one threshold to discriminate critical from non-critical values. If direction is above than all values below the threshold_value are normal (displayed in dark blue in the plot and flagged with GRADING = 0 in the dataframe). All values above the threshold_value are considered critical. The more they deviate from the threshold the displayed color shifts to dark red. All critical values are highlighted with GRADING = 1 in the summary data frame. By default, highest values are always shown in dark red irrespective of the absolute deviation.

If direction is below than all values above the threshold_value are normal (displayed in dark blue, GRADING = 0).

Hint:

This function does not support a `resp_vars` argument but `exclude_roles` to specify variables not relevant for detecting a missing segment.

List function.

Value

a list with:

- SummaryData: data frame about segment missingness
- SummaryPlot: ggplot2 heatmap plot: a heatmap-like graphic that highlights critical values depending on the respective threshold_value and direction.

See Also

Online Documentation

`com_unit_missingness` *Counts all individuals with no measurements at all*

Description

This implementation examines a crude version of unit missingness or unit-nonresponse (Kalton and Kasprzyk 1986), i.e. if all measurement variables in the study data are missing for an observation it has unit missingness.

The function can be applied on stratified data. In this case strata_vars must be specified.

Usage

```
com_unit_missingness(
  study_data,
  meta_data,
  id_vars = NULL,
  strata_vars = NULL,
  label_col
)
```

Arguments

study_data	data.frame the data frame that contains the measurements
meta_data	data.frame the data frame that contains metadata attributes of study data
id_vars	variable list optional, a (vectorized) call of ID-variables that should not be considered in the calculation of unit- missingness
strata_vars	variable optional, a string or integer variable used for stratification
label_col	variable attribute the name of the column in the metadata with labels of variables

Details

This implementations calculates a crude rate of unit-missingness. This type of missingness may have several causes and is an important research outcome. For example, unit-nonresponse may be selective regarding the targeted study population or technical reasons such as record-linkage may cause unit-missingness.

It has to be discriminated form segment and item missingness, since different causes and mechanisms may be the reason for unit-missingness.

Hint:
This function does not support a `resp_vars` argument but `id_vars`, which have a roughly inverse logic behind: id_vars with values do not prevent a row from being considered missing, because an ID is the only hint for a unit that elsewise would not occur in the data at all.
List function.

Value

A list with:

- `FlaggedStudyData`: data.frame with id-only-rows flagged in a column `Unit_missing`
- `SummaryData`: data.frame with numbers and percentages of unit missingness

See Also

Online Documentation

`contradiction_functions`
contradiction_functions

Description

Detect abnormalities help functions

Usage

```
contradiction_functions
```

Format

An object of class `list` of length 11.

Details

2 variables:

- `A_not_equal_B`, if `A != B`
- `A_greater_equal_B`, if `A >= B`
- `A_greater_than_B`, if `A > B`
- `A_less_than_B`, if `A < B`
- `A_less_equal_B`, if `A <= B`
- `A_present_not_B`, if `A & is.na(B)`
- `A_present_and_B`, if `A & !(is.na(B))`
- `A_present_and_B_levels`, if `A & B  %in% {set of levels}`
- `A_levels_and_B_gt_value`, if `A %in% {set of levels} & B >  value`
- `A_levels_and_B_lt_value`, if `A %in% {set of levels} & B <  value`
- `A_levels_and_B_levels`, if `A %in% {set of levels} & B %in% {set of levels}`

contradiction_functions_descriptions *description of the contradiction functions*

Description

description of the contradiction functions

Usage

```
contradiction_functions_descriptions
```

Format

An object of class `list` of length 11.

con_contradictions *Checks user-defined contradictions in study data*

Description

This approach considers a contradiction if impossible combinations of data are observed in one participant. For example, if age of a participant is recorded repeatedly the value of age is (unfortunately) not able to decline. Most cases of contradictions rest on comparison of two variables.

Important to note, each value that is used for comparison may represent a possible characteristic but the combination of these two values is considered to be impossible. The approach does not consider implausible or inadmissible values.

Usage

```
con_contradictions(
  resp_vars = NULL,
  study_data,
  meta_data,
  label_col,
  threshold_value,
  check_table,
  summarize_categories = FALSE
)
```

Arguments

`resp_vars`	variable list the name of the measurement variables
`study_data`	data.frame the data frame that contains the measurements
`meta_data`	data.frame the data frame that contains metadata attributes of study data
`label_col`	variable attribute the name of the column in the metadata with labels of variables
`threshold_value`	numeric from=0 to=100. a numerical value ranging from 0-100
`check_table`	data.frame contradiction rules table. Table defining contractions. See details for its required structure.
`summarize_categories`	logical Needs a column 'tag' in the `check_table`. If set, a summary output is generated for the defined categories plus one plot per category. `inheritParams acc_distributions`

Details

Algorithm of this implementation::

- Select all variables in the data with defined contradiction rules (static metadata column CONTRADICTIONS)
- Remove missing codes from the study data (if defined in the metadata)
- Remove measurements deviating from limits defined in the metadata
- Assign label to levels of categorical variables (if applicable)
- Apply contradiction checks on predefined sets of variables
- Identification of measurements fulfilling contradiction rules. Therefore two output data frames are generated:
  - on the level of observation to flag each contradictory value combination, and
  - a summary table for each contradiction check.
- A summary plot illustrating the number of contradictions is generated.

List function.

Value

If `summarize_categories` is `FALSE`: A list with:

- `FlaggedStudyData`: The first output of the contradiction function is a data frame of similar dimension regarding the number of observations in the study data. In addition, for each applied check on the variables an additional column is added which flags observations with a contradiction given the applied check.
- `SummaryTable`: The second output summarizes this information into one data frame. This output can be used to provide an executive overview on the amount of contradictions. This output is meant for automatic digestion within pipelines.
- `SummaryData`: The third output is the same as `SummaryTable` but for human readers.
- `SummaryPlot`: The fourth output visualizes summarized information of `SummaryData`.

if summarize_categories is TRUE, other objects are returned: one per category named by that category (e.g. "Empirical") containing a result for contradictions within that category only. Additionally, in the slot all_checks a result as it would have been returned with summarize_categories set to FALSE. Finally, a slot SummaryData is returned containing sums per Category and an according ggplot in SummaryPlot.

See Also

Online Documentation

Examples

```
## Not run:
load(system.file("extdata", "meta_data.RData", package = "dataquieR"))
load(system.file("extdata", "study_data.RData", package = "dataquieR"))
check_table <- read.csv(system.file("extdata",
  "contradiction_checks.csv",
  package = "dataquieR"
),
header = TRUE, sep = "#"
)
check_table[1, "tag"] <- "Logical"
check_table[1, "Label"] <- "Becomes younger"
check_table[2, "tag"] <- "Empirical"
check_table[2, "Label"] <- "sex transformation"
check_table[3, "tag"] <- "Empirical"
check_table[3, "Label"] <- "looses academic degree"
check_table[4, "tag"] <- "Logical"
check_table[4, "Label"] <- "vegetarian eats meat"
check_table[5, "tag"] <- "Logical"
check_table[5, "Label"] <- "vegan eats meat"
check_table[6, "tag"] <- "Empirical"
check_table[6, "Label"] <- "non-veg* eats meat"
check_table[7, "tag"] <- "Empirical"
check_table[7, "Label"] <- "Non-smoker buys cigarettes"
check_table[8, "tag"] <- "Empirical"
check_table[8, "Label"] <- "Smoker always scrounges"
check_table[9, "tag"] <- "Logical"
check_table[9, "Label"] <- "Cuff didn't fit arm"
check_table[10, "tag"] <- "Empirical"
check_table[10, "Label"] <- "Very mature pregnant woman"
label_col <- "LABEL"
threshold_value <- 1
con_contradictions(
  study_data = study_data, meta_data = meta_data, label_col = label_col,
  threshold_value = threshold_value, check_table = check_table
)
check_table[1, "tag"] <- "Logical, Age-Related"
check_table[10, "tag"] <- "Empirical, Age-Related"
con_contradictions(
  study_data = study_data, meta_data = meta_data, label_col = label_col,
  threshold_value = threshold_value, check_table = check_table
```

```
)
con_contradictions(
  study_data = study_data, meta_data = meta_data, label_col = label_col,
  threshold_value = threshold_value, check_table = check_table,
  summarize_categories = TRUE
)

## End(Not run)
```

`con_contradictions_redcap` *Checks user-defined contradictions in study data*

Description

This approach considers a contradiction if impossible combinations of data are observed in one participant. For example, if age of a participant is recorded repeatedly the value of age is (unfortunately) not able to decline. Most cases of contradictions rest on comparison of two variables.

Important to note, each value that is used for comparison may represent a possible characteristic but the combination of these two values is considered to be impossible. The approach does not consider implausible or inadmissible values.

Usage

```
con_contradictions_redcap(
  study_data,
  meta_data,
  label_col,
  threshold_value,
  meta_data_cross_item = "cross-item_level",
  use_value_labels,
  summarize_categories = FALSE
)
```

Arguments

`study_data`	data.frame the data frame that contains the measurements
`meta_data`	data.frame the data frame that contains metadata attributes of study data
`label_col`	variable attribute the name of the column in the metadata with labels of variables
`threshold_value`	numeric from=0 to=100. a numerical value ranging from 0-100
`meta_data_cross_item`	data.frame contradiction rules table. Table defining contractions. See details for its required structure.

use_value_labels
: logical If set to TRUE, labels can be used in the REDCap syntax to specify contraction checks for categorical variables. If set to FALSE, contractions have to be specified using the coded values. In case that this argument is not set in the function call, it will be set to TRUE if the metadata contains a column VALUE_LABELS which is not empty.

  inheritParams acc_distributions

summarize_categories
: logical Needs a column 'CONTRADICTION_TYPE' in the meta_data_cross_item. If set, a summary output is generated for the defined categories plus one plot per category.

Details

Algorithm of this implementation::

- Remove missing codes from the study data (if defined in the metadata)
- Remove measurements deviating from limits defined in the metadata
- Assign label to levels of categorical variables (if applicable)
- Apply contradiction checks (given as REDCap-like rules in a separate metadata table)
- Identification of measurements fulfilling contradiction rules. Therefore two output data frames are generated:
  - on the level of observation to flag each contradictory value combination, and
  - a summary table for each contradiction check.
- A summary plot illustrating the number of contradictions is generated.

List function.

Value

If summarize_categories is FALSE: A list with:

- FlaggedStudyData: The first output of the contradiction function is a data frame of similar dimension regarding the number of observations in the study data. In addition, for each applied check on the variables an additional column is added which flags observations with a contradiction given the applied check.
- SummaryData: The second output summarizes this information into one data frame. This output can be used to provide an executive overview on the amount of contradictions.
- VariableGroupTable: A subset of SummaryData used within the pipeline.
- SummaryPlot: The third output visualizes summarized information of SummaryData.

If summarize_categories is TRUE, other objects are returned: One per category named by that category (e.g. "Empirical") containing a result for contradiction checks within that category only. Additionally, in the slot all_checks, a result as it would have been returned with summarize_categories set to FALSE. Finally, a slot SummaryData is returned containing sums per Category and an according ggplot in SummaryPlot.

See Also

Online Documentation

Examples

```
## Not run:  # slow
load(system.file("extdata", "meta_data.RData", package = "dataquieR"))
load(system.file("extdata", "study_data.RData", package = "dataquieR"))
meta_data_cross_item <- prep_get_data_frame("meta_data_v2|cross-item_level")
label_col <- "LABEL"
threshold_value <- 1
con_contradictions_redcap(
  study_data = study_data, meta_data = meta_data, label_col = label_col,
  threshold_value = threshold_value, meta_data_cross_item = meta_data_cross_item
)
con_contradictions_redcap(
  study_data = study_data, meta_data = meta_data, label_col = label_col,
  threshold_value = threshold_value, meta_data_cross_item = meta_data_cross_item,
  summarize_categories = TRUE
)

## End(Not run)
```

---

con_detection_limits *Detects variable values exceeding detection limits*

Description

Inadmissible numerical values can be of type integer or float. This implementation requires the definition of intervals in the metadata to examine the admissibility of numerical study data.

This helps identify inadmissible measurements according to hard limits (for multiple variables).

Usage

```
con_detection_limits(
  resp_vars = NULL,
  label_col,
  study_data,
  meta_data,
  limits = c("DETECTION_LIMITS", "HARD_LIMITS", "SOFT_LIMITS"),
  flip_mode = "flip"
)
```

Arguments

resp_vars	variable list the name of the measurement variables
label_col	variable attribute the name of the column in the metadata with labels of variables
study_data	data.frame the data frame that contains the measurements
meta_data	data.frame the data frame that contains metadata attributes of study data
limits	enum HARD_LIMITS | SOFT_LIMITS | DETECTION_LIMITS. what limits from metadata to check for

`flip_mode` enum default | flip | noflip | auto. Should the plot be in default orientation, flipped, not flipped or auto-flipped. Not all options are always supported. In general, this con be controlled by setting the `roptions(dataquieR.flip_mode = ...)`. If called from `dq_report`, you can also pass `flip_mode` to all function calls or set them specifically using `specific_args`.

Details

Algorithm of this implementation::

- Remove missing codes from the study data (if defined in the metadata)
- Interpretation of variable specific intervals as supplied in the metadata.
- Identification of measurements outside defined limits. Therefore two output data frames are generated:
  - on the level of observation to flag each deviation, and
  - a summary table for each variable.
- A list of plots is generated for each variable examined for limit deviations. The histogram-like plots indicate respective limits as well as deviations.
- Values exceeding limits are removed in a data frame of modified study data

For con_detection_limits, The default for the limits argument differs and is here "DETECTION_LIMITS"

Value

a list with:

- `FlaggedStudyData` data.frame related to the study data by a 1:1 relationship, i.e. for each observation is checked whether the value is below or above the limits.
- `SummaryTable` data.frame summarizes limit deviations for each variable.
- `SummaryPlotList` list of ggplots The plots for each variable are either a histogram (continuous) or a barplot (discrete).
- `ModifiedStudyData` data.frame If the function identifies limit deviations, the respective values are removed in `ModifiedStudyData`.
- `ReportSummaryTable`: heatmap-like data frame about limit violations

See Also

- con_limit_deviations
- Online Documentation

`con_hard_limits` *Detects variable values exceeding hard limits*

Description

Inadmissible numerical values can be of type integer or float. This implementation requires the definition of intervals in the metadata to examine the admissibility of numerical study data.

This helps identify inadmissible measurements according to hard limits (for multiple variables).

Usage

```
con_hard_limits(
  resp_vars = NULL,
  label_col,
  study_data,
  meta_data,
  limits = c("HARD_LIMITS", "SOFT_LIMITS", "DETECTION_LIMITS"),
  flip_mode = "flip"
)
```

Arguments

`resp_vars`	variable list the name of the measurement variables
`label_col`	variable attribute the name of the column in the metadata with labels of variables
`study_data`	data.frame the data frame that contains the measurements
`meta_data`	data.frame the data frame that contains metadata attributes of study data
`limits`	enum HARD_LIMITS | SOFT_LIMITS | DETECTION_LIMITS. what limits from metadata to check for
`flip_mode`	enum default | flip | noflip | auto. Should the plot be in default orientation, flipped, not flipped or auto-flipped. Not all options are always supported. In general, this con be controlled by setting the `roptions(dataquieR.flip_mode = ...)`. If called from `dq_report`, you can also pass `flip_mode` to all function calls or set them specifically using `specific_args`.

Details

Algorithm of this implementation::

- Remove missing codes from the study data (if defined in the metadata)
- Interpretation of variable specific intervals as supplied in the metadata.
- Identification of measurements outside defined limits. Therefore two output data frames are generated:
  - on the level of observation to flag each deviation, and
  - a summary table for each variable.

- A list of plots is generated for each variable examined for limit deviations. The histogram-like plots indicate respective limits as well as deviations.
- Values exceeding limits are removed in a data frame of modified study data

For con_detection_limits, The default for the limits argument differs and is here "DETECTION_LIMITS"

Value

a list with:

- `FlaggedStudyData` data.frame related to the study data by a 1:1 relationship, i.e. for each observation is checked whether the value is below or above the limits.
- `SummaryTable` data.frame summarizes limit deviations for each variable.
- `SummaryPlotList` list of ggplots The plots for each variable are either a histogram (continuous) or a barplot (discrete).
- `ModifiedStudyData` data.frame If the function identifies limit deviations, the respective values are removed in `ModifiedStudyData`.
- `ReportSummaryTable`: heatmap-like data frame about limit violations

See Also

- con_limit_deviations
- Online Documentation

---

con_inadmissible_categorical

Detects variable levels not specified in metadata

Description

For each categorical variable, value lists should be defined in the metadata. This implementation will examine, if all observed levels in the study data are valid.

Usage

```
con_inadmissible_categorical(
  resp_vars = NULL,
  study_data,
  meta_data,
  label_col,
  threshold_value = 0
)
```

Arguments

`resp_vars`	variable list the name of the measurement variables
`study_data`	data.frame the data frame that contains the measurements
`meta_data`	data.frame the data frame that contains metadata attributes of study data
`label_col`	variable attribute the name of the column in the metadata with labels of variables
`threshold_value`	numeric from=0 to=100. a numerical value ranging from 0-100.

Details

Algorithm of this implementation::

- Remove missing codes from the study data (if defined in the metadata)
- Interpretation of variable specific VALUE_LABELS as supplied in the metadata.
- Identification of measurements not corresponding to the expected categories. Therefore two output data frames are generated:
  - on the level of observation to flag each undefined category, and
  - a summary table for each variable.
- Values not corresponding to defined categories are removed in a data frame of modified study data

Value

a list with:

- `SummaryData`: data frame summarizing inadmissible categories with the columns:
  - `Variables`: variable name/label
  - `OBSERVED_CATEGORIES`: the categories observed in the study data
  - `DEFINED_CATEGORIES`: the categories defined in the metadata
  - `NON_MATCHING`: the categories observed but not defined
  - `NON_MATCHING_N`: the number of observations with categories not defined
  - `NON_MATCHING_N_PER_CATEGORY`: the number of observations for each of the unexpected categories
  - `GRADING`: indicator TRUE/FALSE if inadmissible categorical values were observed (more than indicated by the `threshold_value`)
- `SummaryTable`: data frame for the `dataquieR` pipeline reporting the number and percentage of inadmissible categorical values
- `ModifiedStudyData`: study data having inadmissible categories removed
- `FlaggedStudyData`: study data having cases with inadmissible categories flagged

See Also

Online Documentation

`con_limit_deviations` *Detects variable values exceeding limits defined in metadata*

Description

Inadmissible numerical values can be of type integer or float. This implementation requires the definition of intervals in the metadata to examine the admissibility of numerical study data.

This helps identify inadmissible measurements according to hard limits (for multiple variables).

Usage

```
con_limit_deviations(
  resp_vars = NULL,
  label_col,
  study_data,
  meta_data,
  limits = c("HARD_LIMITS", "SOFT_LIMITS", "DETECTION_LIMITS"),
  flip_mode = "flip"
)
```

Arguments

`resp_vars`	variable list the name of the measurement variables
`label_col`	variable attribute the name of the column in the metadata with labels of variables
`study_data`	data.frame the data frame that contains the measurements
`meta_data`	data.frame the data frame that contains metadata attributes of study data
`limits`	enum HARD_LIMITS | SOFT_LIMITS | DETECTION_LIMITS. what limits from metadata to check for
`flip_mode`	enum default | flip | noflip | auto. Should the plot be in default orientation, flipped, not flipped or auto-flipped. Not all options are always supported. In general, this con be controlled by setting the `roptions(dataquieR.flip_mode = ...)`. If called from `dq_report`, you can also pass `flip_mode` to all function calls or set them specifically using `specific_args`.

Details

Algorithm of this implementation::

- Remove missing codes from the study data (if defined in the metadata)
- Interpretation of variable specific intervals as supplied in the metadata.
- Identification of measurements outside defined limits. Therefore two output data frames are generated:
  - on the level of observation to flag each deviation, and
  - a summary table for each variable.

- A list of plots is generated for each variable examined for limit deviations. The histogram-like plots indicate respective limits as well as deviations.
- Values exceeding limits are removed in a data frame of modified study data

For con_detection_limits, The default for the limits argument differs and is here "DETECTION_LIMITS"

Value

a list with:

- `FlaggedStudyData` data.frame related to the study data by a 1:1 relationship, i.e. for each observation is checked whether the value is below or above the limits.
- `SummaryTable` data.frame summarizes limit deviations for each variable.
- `SummaryPlotList` list of ggplots The plots for each variable are either a histogram (continuous) or a barplot (discrete).
- `ModifiedStudyData` data.frame If the function identifies limit deviations, the respective values are removed in `ModifiedStudyData`.
- `ReportSummaryTable`: heatmap-like data frame about limit violations

See Also

- con_detection_limits
- Online Documentation

`con_soft_limits` *Detects variable values exceeding soft limits*

Description

Inadmissible numerical values can be of type integer or float. This implementation requires the definition of intervals in the metadata to examine the admissibility of numerical study data.

This helps identify inadmissible measurements according to hard limits (for multiple variables).

Usage

```
con_soft_limits(
  resp_vars = NULL,
  label_col,
  study_data,
  meta_data,
  limits = c("SOFT_LIMITS", "DETECTION_LIMITS", "HARD_LIMITS"),
  flip_mode = "flip"
)
```

Arguments

resp_vars	variable list the name of the measurement variables
label_col	variable attribute the name of the column in the metadata with labels of variables
study_data	data.frame the data frame that contains the measurements
meta_data	data.frame the data frame that contains metadata attributes of study data
limits	enum HARD_LIMITS | SOFT_LIMITS | DETECTION_LIMITS. what limits from metadata to check for
flip_mode	enum default | flip | noflip | auto. Should the plot be in default orientation, flipped, not flipped or auto-flipped. Not all options are always supported. In general, this con be controlled by setting the `roptions(dataquieR.flip_mode = ...)`. If called from `dq_report`, you can also pass `flip_mode` to all function calls or set them specifically using `specific_args`.

Details

Algorithm of this implementation::

- Remove missing codes from the study data (if defined in the metadata)
- Interpretation of variable specific intervals as supplied in the metadata.
- Identification of measurements outside defined limits. Therefore two output data frames are generated:
  - on the level of observation to flag each deviation, and
  - a summary table for each variable.
- A list of plots is generated for each variable examined for limit deviations. The histogram-like plots indicate respective limits as well as deviations.
- Values exceeding limits are removed in a data frame of modified study data

For con_detection_limits, The default for the limits argument differs and is here "DETECTION_LIMITS"

Value

a list with:

- `FlaggedStudyData` data.frame related to the study data by a 1:1 relationship, i.e. for each observation is checked whether the value is below or above the limits.
- `SummaryTable` data.frame summarizes limit deviations for each variable.
- `SummaryPlotList` list of ggplots The plots for each variable are either a histogram (continuous) or a barplot (discrete).
- `ModifiedStudyData` data.frame If the function identifies limit deviations, the respective values are removed in `ModifiedStudyData`.
- `ReportSummaryTable`: heatmap-like data frame about limit violations

See Also

- con_limit_deviations
- Online Documentation

dataquieR *The* dataquieR *package about Data Quality in Epidemiological Research*

Description

For a quick start please read dq_report and maybe the vignettes or the package's website.

dataquieR_resultset *Internal constructor for the internal class dataquieR_resultset.*

Description

creates an object of the class dataquieR_resultset.

Usage

```
dataquieR_resultset(...)
```

Arguments

...	properties stored in the object

Details

The class features the following methods:

- as.data.frame.dataquieR_resultset, * as.list.dataquieR_resultset, * print.dataquieR_resultset, * summary.dataquieR_resultset

Value

an object of the class dataquieR_resultset.

See Also

dq_report

`dataquieR_resultset2` *Internal constructor for the internal class dataquieR_resultset2.*

Description

creates an object of the class dataquieR_resultset2.

Usage

```
dataquieR_resultset2(...)
```

Arguments

...	properties stored in the object

Details

The class features the following methods:

- `[[.dataquieR_resultset2]`
- `dim.dataquieR_resultset2`
- `dimnames.dataquieR_resultset2`
- `print.dataquieR_resultset2`
- `resnames.dataquieR_resultset2`
- `summary.dataquieR_resultset2`

Value

an object of the class dataquieR_resultset2.

See Also

dq_report2

dataquieR_resultset_verify *Verify an object of class dataquieR_resultset*

Description

stops on errors

Usage

```
dataquieR_resultset_verify(list_to_verify)
```

Arguments

`list_to_verify`	object to be checked

Value

`invisible(TRUE)` – stops on errors.

DATA_TYPES *Data Types*

Description

Data Types of Study Data:
In the metadata, the following entries are allowed for the variable attribute DATA_TYPE:

Usage

```
DATA_TYPES
```

Format

An object of class `list` of length 4.

Details

- `integer` for integer numbers
- `string` for text/string/character data
- `float` for decimal/floating point numbers
- `datetime` for timepoints

Data Types of Function Arguments:
As function arguments, dataquieR uses additional type specifications:

- `numeric` is a numerical value (float or integer), but it is not an allowed `DATA_TYPE` in the metadata. However, some functions may accept `float` or `integer` for specific function arguments. This is, where we use the term `numeric`.
- `enum` allows one element out of a set of allowed options similar to match.arg
- `set` allows a subset out of a set of allowed options similar to match.arg with `several.ok = TRUE`.
- `variable` Function arguments of this type expect a character scalar that specifies one variable using the variable identifier given in the meta data attribute `VAR_NAMES` or, if `label_col` is set, given in the meta data attribute given in that argument. Labels can easily be translated using prep_map_labels
- `variable list` Function arguments of this type expect a character vector that specifies variables using the variable identifiers given in the meta data attribute `VAR_NAMES` or, if `label_col` is set, given in the meta data attribute given in that argument. Labels can easily be translated using prep_map_labels

See Also

integer string

`DATA_TYPES_OF_R_TYPE` *All available data types, mapped from their respective R types*

Description

All available data types, mapped from their respective R types

Usage

```
DATA_TYPES_OF_R_TYPE
```

Format

An object of class `list` of length 14.

See Also

`prep_dq_data_type_of`

DF_ELEMENT_COUNT *Data frame level metadata attribute name*

Description

Number of expected data elements in a data frame. numeric. Check only conducted if number entered

Usage

```
DF_ELEMENT_COUNT
```

Format

An object of class `character` of length 1.

See Also

meta_data_dataframe

DF_ID_REF_TABLE *Data frame level metadata attribute name*

Description

The name of the data frame containing the reference IDs to be compared with the IDs in the study data set.

Usage

```
DF_ID_REF_TABLE
```

Format

An object of class `character` of length 1.

See Also

meta_data_dataframe

DF_ID_VARS *Data frame level metadata attribute name*

Description

All variables that are to be used as one single ID variable (combined key) in a data frame.

Usage

```
DF_ID_VARS
```

Format

An object of class `character` of length 1.

See Also

meta_data_dataframe

DF_NAME *Data frame level metadata attribute name*

Description

Name of the data frame

Usage

```
DF_NAME
```

Format

An object of class `character` of length 1.

See Also

meta_data_dataframe

DF_RECORD_CHECK *Data frame level metadata attribute name*

Description

The type of check to be conducted when comparing the reference ID table with the IDs delivered in the study data files.

Usage

```
DF_RECORD_CHECK
```

Format

An object of class `character` of length 1.

See Also

meta_data_dataframe

DF_RECORD_COUNT *Data frame level metadata attribute name*

Description

Number of expected data records in a data frame. numeric. Check only conducted if number entered

Usage

```
DF_RECORD_COUNT
```

Format

An object of class `character` of length 1.

See Also

meta_data_dataframe

DF_UNIQUE_ID *Data frame level metadata attribute name*

Description

Defines expectancies on the uniqueness of the IDs across the rows of a data frame, or the number of times some ID can be repeated.

Usage

```
DF_UNIQUE_ID
```

Format

An object of class `character` of length 1.

See Also

meta_data_dataframe

DF_UNIQUE_ROWS *Data frame level metadata attribute name*

Description

Specifies whether identical data is permitted across rows in a data frame (excluding ID variables)

Usage

```
DF_UNIQUE_ROWS
```

Format

An object of class `character` of length 1.

See Also

meta_data_dataframe

dim.dataquieR_resultset2 *Get the dimensions of a* dq_report2 *result*

Description

Get the dimensions of a dq_report2 result

Usage

```
## S3 method for class 'dataquieR_resultset2'
dim(x)
```

Arguments

x	a dataquieR_resultset2 result

Value

dimensions

dimensions *Names of DQ dimensions*

Description

a vector of data quality dimensions. The supported dimensions are Completeness, Consistency and Accuracy.

Usage

```
dimensions
```

Format

An object of class character of length 3.

Value

Only a definition, not a function, so no return value

See Also

Data Quality Concept

dimnames.dataquieR_resultset2 *Names of a* dataquieR *report object (v2.0)*

Description

Names of a dataquieR report object (v2.0)

Usage

```
## S3 method for class 'dataquieR_resultset2'
dimnames(x)
```

Arguments

x	the result object

Value

the names

DISTRIBUTIONS *All available probability distributions for acc_shape_or_scale*

Description

- uniform For uniform distribution
- normal For Gaussian distribution
- GAMMA For a gamma distribution

Usage

```
DISTRIBUTIONS
```

Format

An object of class list of length 3.

dq_report *Generate a full DQ report*

Description

Generate a full DQ report

Usage

```
dq_report(
  study_data,
  meta_data,
  label_col = NULL,
  meta_data_segment = "segment_level",
  meta_data_dataframe = "dataframe_level",
  ...,
  dimensions = c("Completeness", "Consistency"),
  dont_modify_study_data_by = c("con_soft_limits", "con_detection_limits"),
  strata_attribute,
  strata_vars,
  cores = list(mode = "socket", logging = FALSE, cpus = util_detect_cores(),
    load.balancing = TRUE),
  specific_args = list(),
  author = prep_get_user_name(),
  debug_parallel = FALSE
)
```

Arguments

study_data	data.frame the data frame that contains the measurements
meta_data	data.frame the data frame that contains metadata attributes of study data
label_col	variable attribute the name of the column in the metadata with labels of variables
meta_data_segment	data.frame – optional: Segment level metadata
meta_data_dataframe	data.frame – optional: Data frame level metadata
...	arguments to be passed to all called indicator functions if applicable.
dimensions	dimensions Vector of dimensions to address in the report. Allowed values in the vector are Completeness, Consistency, and Accuracy. The generated report will only cover the listed data quality dimensions. Accuracy is computational expensive, so this dimension is not enabled by default. Completeness should be included, if Consistency is included, and Consistency should be included, if Accuracy is included to avoid misleading detections of e.g. missing codes as outliers, please refer to the data quality concept for more details. Integrity is always included.

`dont_modify_study_data_by`
: character list of functions, which are not allowed to modify study data downstream the pipeline, e.g., to avoid, that even soft limit violations are removed.

`strata_attribute`
: character variable of a variable attribute coding study segments. Values other than leaving this empty or passing STUDY_SEGMENT are not yet supported. Stratification is not yet fully supported, please use dq_report_by.

`strata_vars`
: character name of variables to stratify the report on, such as "study_center". Not yet supported, please use dq_report_by.

`cores`
: integer number of cpu cores to use or a named list with arguments for parallelMap::parallelStart or NULL, if parallel has already been started by the caller.

`specific_args`
: list named list of arguments specifically for one of the called functions, the of the list elements correspond to the indicator functions whose calls should be modified. The elements are lists of arguments.

`author`
: character author for the report documents.

`debug_parallel`
: logical print blocks currently evaluated in parallel

Details

See dq_report_by for a way to generate stratified or splitted reports easily.

Value

a dataquieR_resultset. Can be printed creating a RMarkdown-report.

See Also

- as.data.frame.dataquieR_resultset, * as.list.dataquieR_resultset, * print.dataquieR_resultset, * summary.dataquieR_resultset
- dq_report_by

Examples

```
## Not run:  # really long-running example.
load(system.file("extdata", "study_data.RData", package = "dataquieR"))
load(system.file("extdata", "meta_data.RData", package = "dataquieR"))
report <- dq_report(study_data, meta_data, label_col = LABEL) # most easy use
report <- dq_report(study_data, meta_data,
  label_col = LABEL, dimensions =
    c("Completeness", "Consistency", "Accuracy"),
  check_table = read.csv(system.file("extdata",
    "contradiction_checks.csv",
    package = "dataquieR"
  ), header = TRUE, sep = "#"),
  show_causes = TRUE,
  cause_label_df = prep_get_data_frame("meta_data_v2|missing_table")
)
save(report, file = "report.RData") # careful, this contains the study_data
report <- dq_report(study_data, meta_data,
```

```
  label_col = LABEL,
  check_table = read.csv(system.file("extdata",
    "contradiction_checks.csv",
    package = "dataquieR"
  ), header = TRUE, sep = "#"),
  specific_args = list(acc_univariate_outlier = list(resp_vars = "K")),
    resp_vars = "SBP_0"
)
report <- dq_report(study_data, meta_data,
  label_col = LABEL,
  check_table = read.csv(system.file("extdata",
    "contradiction_checks.csv",
    package = "dataquieR"
  ), header = TRUE, sep = "#"),
  specific_args = list(acc_univariate_outlier = list(resp_vars = "DBP_0")),
    resp_vars = "SBP_0"
)
report <- dq_report(study_data, meta_data,
  label_col = LABEL,
  check_table = read.csv(system.file("extdata",
    "contradiction_checks.csv",
    package = "dataquieR"
  ), header = TRUE, sep = "#"),
  specific_args = list(acc_univariate_outlier = list(resp_vars = "DBP_0")),
  resp_vars = "SBP_0", cores = NULL
)
rp1 <- dq_report("ship", "ship_meta",
  meta_data_segment = "meta_data_segment",
  meta_data_dataframe = "meta_data_dataframe",
  label_col = LABEL)

## End(Not run)
```

---

dq_report2 *Generate a full DQ report, v2*

Description

Generate a full DQ report, v2

Usage

```
dq_report2(
  study_data,
  meta_data = "item_level",
  label_col = LABEL,
  meta_data_segment = "segment_level",
  meta_data_dataframe = "dataframe_level",
  meta_data_cross_item = "cross-item_level",
  ...,
```

```
  dimensions = c("Completeness", "Consistency"),
  cores = list(mode = "socket", logging = FALSE, cpus = util_detect_cores(),
    load.balancing = TRUE),
  specific_args = list(),
  author = prep_get_user_name(),
  user_info = NULL,
  debug_parallel = FALSE,
  resp_vars = character(0),
  filter_indicator_functions = character(0),
  filter_result_slots = character(0)
)
```

Arguments

study_data	data.frame the data frame that contains the measurements
meta_data	data.frame the data frame that contains metadata attributes of study data
label_col	variable attribute the name of the column in the metadata with labels of variables
meta_data_segment	data.frame – optional: Segment level metadata
meta_data_dataframe	data.frame – optional: Data frame level metadata
meta_data_cross_item	data.frame – optional: Cross-item level metadata
...	arguments to be passed to all called indicator functions if applicable.
dimensions	dimensions Vector of dimensions to address in the report. Allowed values in the vector are Completeness, Consistency, and Accuracy. The generated report will only cover the listed data quality dimensions. Accuracy is computational expensive, so this dimension is not enabled by default. Completeness should be included, if Consistency is included, and Consistency should be included, if Accuracy is included to avoid misleading detections of e.g. missing codes as outliers, please refer to the data quality concept for more details. Integrity is always included.
cores	integer number of cpu cores to use or a named list with arguments for parallelMap::parallelStart or NULL, if parallel has already been started by the caller.
specific_args	list named list of arguments specifically for one of the called functions, the of the list elements correspond to the indicator functions whose calls should be modified. The elements are lists of arguments.
author	character author for the report documents.
user_info	list additional info stored with the report, e.g., comments, title, ...
debug_parallel	logical print blocks currently evaluated in parallel
resp_vars	variable list the name of the measurement variables for the report. If missing, all variables will be used. Only item level indicator functions are filtered, so far.
filter_indicator_functions	character regular expressions, only if an indicator function's name matches one of these, it'll be used for the report. If of length zero, no filtering is performed.

filter_result_slots
: character regular expressions, only if an indicator function's result's name matches one of these, it'll be used for the report. If of length zero, no filtering is performed.

Details

See dq_report_by for a way to generate stratified or splitted reports easily.

Value

a dataquieR_resultset2. Can be printed creating a RMarkdown-report.

See Also

- as.data.frame.dataquieR_resultset, * as.list.dataquieR_resultset, * print.dataquieR_resultset, * summary.dataquieR_resultset
- dq_report_by

Examples

```
## Not run:
prep_load_workbook_like_file("inst/extdata/meta_data_v2.xlsx")
meta_data <- prep_get_data_frame("item_level")
meta_data_cross <- prep_get_data_frame("cross-item_level")
x <- dq_report2("study_data", dimensions = NULL, label_col = "LABEL")
xx <- pbapply::pblapply(x, util_eval_to_dataquieR_result, env = environment())
xx <- pbapply::pblapply(tail(x), util_eval_to_dataquieR_result, env = environment())
xx <- parallel
cat(vapply(x, deparse1, FUN.VALUE = character(1)), sep = "\n", file = "all_calls.txt")
rstudioapi::navigateToFile("all_calls.txt")
eval(x$`acc_multivariate_outlier.Blood pressure checks`)

## End(Not run)
```

---

dq_report_by *Generate a stratified full DQ report*

Description

Generate a stratified full DQ report

Usage

```
dq_report_by(
  study_data,
  meta_data,
  label_col,
  meta_data_split = STUDY_SEGMENT,
```

```
  study_data_split,
  ...,
  v1.0 = FALSE
)
```

Arguments

study_data	data.frame the data frame that contains the measurements
meta_data	data.frame the data frame that contains metadata attributes of study data
label_col	variable attribute the name of the column in the metadata with labels of variables
meta_data_split	variable attribute name of a meta data attribute to split the report in sections of variables, e.g. all blood- pressure. By default, reports are split by STUDY_SEGMENT if available.
study_data_split	variable Name of a study variable to stratify the report by, e.g. the study centers.
...	passed through to dq_report
v1.0	logical if TRUE, create old dq_report reports, use dq_report2, otherwise

See Also

dq_report

Examples

```
## Not run:  # really long-running example.
prep_load_workbook_like_file("inst/extdata/meta_data_extended.xlsx")
rep <- dq_report_by("study_data", label_col =
  LABEL, study_data_split = "CENTER_0")
rep <- dq_report_by("study_data",
  label_col = LABEL, study_data_split = "CENTER_0",
  meta_data_split = NULL
)

## End(Not run)
```

---

html_dependency_report_dt

HTML Dependency for report headers in DT::datatable

Description

HTML Dependency for report headers in DT::datatable

Usage

```
html_dependency_report_dt()
```

Value

the dependency

`html_dependency_vert_dt` *HTML Dependency for vertical headers in* `DT::datatable`

Description

HTML Dependency for vertical headers in `DT::datatable`

Usage

```
html_dependency_vert_dt()
```

Value

the dependency

`int_all_datastructure_dataframe` *Wrapper function to check for studies data structure*

Description

This function tests for unexpected elements and records, as well as duplicated identifiers and content. The unexpected element record check can be conducted by providing the number of expected records or an additional table with the expected records. It is possible to conduct the checks by study segments or to consider only selected segments.

Usage

```
int_all_datastructure_dataframe(
  meta_data_dataframe = "dataframe_level",
  meta_data = "item_level"
)
```

Arguments

`meta_data_dataframe`	data.frame the data frame that contains the metadata for the data frame level, mandatory
`meta_data`	data.frame the data frame that contains metadata attributes of the study data, mandatory. The metadata data frame is assumed to contain the information from all the studies. this is needed to know the `VAR_NAMES`, i.e., the column names used in data frames and known from the metadata.

Value

a list with

- DataframeTable: data frame with selected check results, used for the data quality report.

Examples

```
## Not run:
out_dataframe <- int_all_datastructure_dataframe(
  meta_data_dataframe = "meta_data_dataframe",
  meta_data = "ship_meta"
)
md0 <- prep_get_data_frame("ship_meta")
md0
md0$VAR_NAMES
md0$VAR_NAMES[[1]] <- "Id" # is this missmatch reported -- is the data frame
                           # also reported, if nothing is wrong with it
out_dataframe <- int_all_datastructure_dataframe(
  meta_data_dataframe = "meta_data_dataframe",
  meta_data = md0
)

# This is the "normal" procedure for inside pipeline
# but outside this function  checktype is exact by default
options(dataquieR.RECORD_MISSMATCH_CHECKTYPE = "subset_u")
lapply(setNames(nm = prep_get_data_frame("meta_data_dataframe")$DF_NAME),
  int_sts_element_dataframe, meta_data = md0)
md0$VAR_NAMES[[1]] <-
  "id" # is this missmatch reported -- is the data frame also reported,
       # if nothing is wrong with it
lapply(setNames(nm = prep_get_data_frame("meta_data_dataframe")$DF_NAME),
  int_sts_element_dataframe, meta_data = md0)
options(dataquieR.RECORD_MISSMATCH_CHECKTYPE = "exact")

## End(Not run)
```

---

int_all_datastructure_segment

Wrapper function to check for segment data structure

Description

This function tests for unexpected elements and records, as well as duplicated identifiers and content. The unexpected element record check can be conducted by providing the number of expected records or an additional table with the expected records. It is possible to conduct the checks by study segments or to consider only selected segments.

Usage

```
int_all_datastructure_segment(
  meta_data_segment = "segment_level",
  study_data,
  meta_data = "item_level"
)
```

Arguments

meta_data_segment	data.frame the data frame that contains the metadata for the segment level, mandatory
study_data	data.frame the data frame that contains the measurements, mandatory.
meta_data	data.frame the data frame that contains metadata attributes of the study data, mandatory.

Value

a list with

- SegmentTable: data frame with selected check results, used for the data quality report.

Examples

```
## Not run:
out_segment <- int_all_datastructure_segment(
  meta_data_segment = "meta_data_segment",
  study_data = "ship",
  meta_data = "ship_meta"
)

study_data <- cars
meta_data <- dataquieR::prep_create_meta(VAR_NAMES = c("speedx", "distx"),
  DATA_TYPE = c("integer", "integer"), MISSING_LIST = "|", JUMP_LIST = "|",
  STUDY_SEGMENT = c("Intro", "Ex"))

out_segment <- int_all_datastructure_segment(
  meta_data_segment = "meta_data_segment",
  study_data = study_data,
  meta_data = meta_data
)

## End(Not run)
```

`int_datatype_matrix` *Check declared data types of metadata in study data*

Description

Checks data types of the study data and for the data type declared in the metadata

Usage

```
int_datatype_matrix(
  resp_vars = NULL,
  study_data,
  meta_data,
  split_segments = FALSE,
  label_col,
  max_vars_per_plot = 20,
  threshold_value = 0
)
```

Arguments

`resp_vars`	variable the names of the measurement variables, if missing or `NULL`, all variables will be checked
`study_data`	data.frame the data frame that contains the measurements
`meta_data`	data.frame the data frame that contains metadata attributes of study data
`split_segments`	logical return one matrix per study segment
`label_col`	variable attribute the name of the column in the metadata with labels of variables
`max_vars_per_plot`	integer from=0. The maximum number of variables per single plot.
`threshold_value`	numeric from=0 to=100. percentage failing conversions allowed to still classify a study variable convertible. `inheritParams acc_distributions`

Details

This is a preparatory support function that compares study data with associated metadata. A prerequisite of this function is that the no. of columns in the study data complies with the no. of rows in the metadata.

For each study variable, the function searches for its data type declared in static metadata and returns a heatmap like matrix indicating data type mismatches in the study data.

List function.

Value

a list with:

- SummaryTable: data frame about the applicability of each indicator function (each function in a column). its integer values can be one of the following four categories: 0. Non-matching datatype, 1. Matching datatype,
- SummaryPlot: ggplot2 heatmap plot, graphical representation of SummaryTable
- DataTypePlotList: list of plots per (maybe artificial) segment
- ReportSummaryTable: data frame underlying SummaryPlot

Examples

```
## Not run:
load(system.file("extdata/meta_data.RData", package = "dataquieR"), envir =
  environment())
load(system.file("extdata/study_data.RData", package = "dataquieR"), envir =
  environment())
appmatrix <- int_datatype_matrix(study_data = study_data,
                                 meta_data = meta_data,
                                 label_col = LABEL)

## End(Not run)
```

---

int_duplicate_content *Check for duplicated content*

Description

This function tests for duplicates entries in the data set. It is possible to check duplicated entries by study segments or to consider only selected segments.

Usage

```
int_duplicate_content(level = c("dataframe", "segment"), ...)
```

Arguments

level	character a character vector indicating whether the assessment should be conducted at the study level (level = "dataframe") or at the segment level (level = "segment").
...	Depending on level, passed to either util_int_duplicate_content_segment or util_int_duplicate_content_dataframe

Value

a list. Depending on level, see util_int_duplicate_content_segment or util_int_duplicate_content_dataframe for a description of the outputs.

int_duplicate_ids *Check for duplicated IDs*

Description

This function tests for duplicates entries in identifiers. It is possible to check duplicated identifiers by study segments or to consider only selected segments.

Usage

```
int_duplicate_ids(level = c("dataframe", "segment"), ...)
```

Arguments

`level`	character a character vector indicating whether the assessment should be conducted at the study level (level = "dataframe") or at the segment level (level = "segment").
`...`	Depending on `level`, passed to either util_int_duplicate_ids_segment or util_int_duplicate_ids_dataframe

Value

a list. Depending on `level`, see util_int_duplicate_ids_segment or util_int_duplicate_ids_dataframe for a description of the outputs.

int_part_vars_structure *Detect Expected Observations*

Description

For each participant, check, if an observation was expected, given the `PART_VARS` from item-level metadata

Usage

```
int_part_vars_structure(
  study_data,
  meta_data,
  label_col = LABEL,
  expected_observations = c("HIERARCHY", "SEGMENT"),
  disclose_problem_paprt_var_data = FALSE
)
```

Arguments

study_data	study_data must have all relevant PART_VARS to avoid false-positives on PART_VARS missing from study_data
meta_data	meta_data must be complete to avoid false positives on non-existing PART_VARS
label_col	character mapping attribute colnames(study_data) vs. meta_data[label_col]
expected_observations	enum HIERARCHY | SEGMENT. How should PART_VARS be handled: - SEGMENT: if PART_VAR is 1, an observation is expected - HIERARCHY: the default, if the PART_VAR is 1 for this variable and also for all PART_VARS of PART_VARS up in the hierarchy, an observation is expected.
disclose_problem_paprt_var_data	logical show the problematic data (PART_VAR only)

Value

empty list, so far – the function only warns.

---

int_sts_element_dataframe

Determine missing and/or superfluous data elements

Description

Depends on dataquieR.RECORD_MISSMATCH_CHECKTYPE option, see there – # TODO: Rename this option and find out, how to document and link it here using Roxygen.

Usage

```
int_sts_element_dataframe(study_data, meta_data = "item_level")
```

Arguments

study_data	data.frame the data frame that contains the measurements
meta_data	data.frame the data frame that contains metadata attributes of study data

Value

list with names lots:

- DataframeData: data frame with the unexpected elements check results.
- DataframeTable: data.frame table with all errors, used for the data quality report: - MISSING: meta_data or study_data: where is the element missing - PCT_int_sts_element: Percentage of element mismatches - NUM_int_sts_element: Number of element mismatches - resp_vars: affected element names

`int_sts_element_segment` *Checks for element set*

Description

Checks for element set

Usage

```
int_sts_element_segment(study_data, meta_data = "item_level")
```

Arguments

`study_data`	data.frame the data frame that contains the measurements, mandatory.
`meta_data`	data.frame the data frame that contains metadata attributes of the study data, mandatory.

Value

a list with

- `SegmentData`: data frame with the unexpected elements check results. - `Segment`: name of the corresponding segment, if applicable, `ALL` otherwise
- `SegmentTable`: data frame with the unexpected elements check results, used for the data quality report. - `Segment`: name of the corresponding segment, if applicable, `ALL` otherwise

Examples

```
## Not run:
study_data <- cars
meta_data <- dataquieR::prep_create_meta(VAR_NAMES = c("speedx", "distx"),
  DATA_TYPE = c("integer", "integer"), MISSING_LIST = "|", JUMP_LIST = "|",
  STUDY_SEGMENT = c("Intro", "Ex"))
options(dataquieR.RECORD_MISSMATCH_CHECKTYPE = "none")
int_sts_element_segment(study_data, meta_data)
options(dataquieR.RECORD_MISSMATCH_CHECKTYPE = "exact")
int_sts_element_segment(study_data, meta_data)
study_data <- cars
meta_data <- dataquieR::prep_create_meta(VAR_NAMES = c("speedx", "distx"),
  DATA_TYPE = c("integer", "integer"), MISSING_LIST = "|", JUMP_LIST = "|",
  STUDY_SEGMENT = c("Intro", "Intro"))
options(dataquieR.RECORD_MISSMATCH_CHECKTYPE = "none")
int_sts_element_segment(study_data, meta_data)
options(dataquieR.RECORD_MISSMATCH_CHECKTYPE = "exact")
int_sts_element_segment(study_data, meta_data)
study_data <- cars
meta_data <- dataquieR::prep_create_meta(VAR_NAMES = c("speed", "distx"),
  DATA_TYPE = c("integer", "integer"), MISSING_LIST = "|", JUMP_LIST = "|",
```

```
  STUDY_SEGMENT = c("Intro", "Intro"))
options(dataquieR.RECORD_MISSMATCH_CHECKTYPE = "none")
int_sts_element_segment(study_data, meta_data)
options(dataquieR.RECORD_MISSMATCH_CHECKTYPE = "exact")
int_sts_element_segment(study_data, meta_data)

## End(Not run)
```

`int_unexp_elements` *Check for unexpected data element count*

Description

This function contrasts the expected element number in each study in the metadata with the actual element number in each study data frame.

Usage

```
int_unexp_elements(identifier_name_list, data_element_count)
```

Arguments

`identifier_name_list`
: character a character vector indicating the name of each study data frame, mandatory.

`data_element_count`
: integer an integer vector with the number of expected data elements, mandatory.

Value

a list with

- `DataframeData`: data frame with the results of the quality check for unexpected data elements
- `DataframeTable`: data frame with selected unexpected data elements check results, used for the data quality report.

Examples

```
## Not run:
study_tables <- list(
"sd1" = readRDS(system.file("extdata", "ship_subset1.RDS",
                    package = "dataquieR")),
"sd2" = readRDS(system.file("extdata", "ship_subset2.RDS",
                    package = "dataquieR"))
)

prep_add_data_frames(data_frame_list = study_tables)

int_unexp_elements(
```

```
 identifier_name_list = c("sd1", "sd2"),
 data_element_count = c(30, 29)
)

## End(Not run)
```

`int_unexp_records_dataframe` *Check for unexpected data record count at the data frame level*

Description

This function contrasts the expected record number in each study in the metadata with the actual record number in each study data frame.

Usage

```
int_unexp_records_dataframe(identifier_name_list, data_record_count)
```

Arguments

`identifier_name_list`
: character a character vector indicating the name of each study data frame, mandatory.

`data_record_count`
: integer an integer vector with the number of expected data records per study data frame, mandatory.

Value

a list with

- `DataframeData`: data frame with the results of the quality check for unexpected data elements
- `DataframeTable`: data frame with selected unexpected data elements check results, used for the data quality report.

`int_unexp_records_segment` *Check for unexpected data record count within segments*

Description

This function contrasts the expected record number in each study segment in the metadata with the actual record number in each segment data frame.

Usage

```
int_unexp_records_segment(
  study_segment,
  data_record_count,
  study_data,
  meta_data
)
```

Arguments

`study_segment`	character a character vector indicating the name of each study data frame, mandatory.
`data_record_count`	integer an integer vector with the number of expected data records, mandatory.
`study_data`	data.frame the data frame that contains the measurements, mandatory.
`meta_data`	data.frame the data frame that contains metadata attributes of the study data, mandatory.

Details

The current implementation does not take into account jump or missing codes, the function is rather based on checking whether NAs are present in the study data

Value

a list with

- `SegmentData`: data frame with the results of the quality check for unexpected data elements
- `SegmentTable`: data frame with selected unexpected data elements check results, used for the data quality report.

Examples

```
## Not run:
study_data <- readRDS(system.file("extdata", "ship.RDS", package = "dataquieR"))
meta_data <- readRDS(system.file("extdata", "ship_meta.RDS", package = "dataquieR"))

int_unexp_records_segment(
  study_segment = c("PART_STUDY", "PART_INTERVIEW"),
  data_record_count = c(3000, 1100),
  study_data = study_data,
  meta_data = meta_data
)

## End(Not run)
```

`int_unexp_records_set` *Check for unexpected data record set*

Description

This function tests that the identifiers match a provided record set. It is possible to check for unexpected data record sets by study segments or to consider only selected segments.

Usage

```
int_unexp_records_set(level = c("dataframe", "segment"), ...)
```

Arguments

`level`	character a character vector indicating whether the assessment should be conducted at the study level (level = "dataframe") or at the segment level (level = "segment").
`...`	Depending on `level`, passed to either util_int_unexp_records_set_segment or util_int_unexp_records_set_dataframe

Value

a list. Depending on `level`, see util_int_unexp_records_set_segment or util_int_unexp_records_set_dataframe for a description of the outputs.

Examples

```
## Not run:
study_data <- readRDS(system.file("extdata", "ship.RDS",
  package = "dataquieR"
))
meta_data <- readRDS(system.file("extdata", "ship_meta.RDS",
  package = "dataquieR"
```

```
))
md1_segment <- readRDS(system.file("extdata", "meta_data_segment.RDS",
  package = "dataquieR"
))
ids_segment <- readRDS(system.file("extdata", "meta_data_ids_segment.RDS",
  package = "dataquieR"
))

# TODO: update examples
int_unexp_records_set(
  level = "segment",
  identifier_name_list = c("INTERVIEW", "LABORATORY"),
  valid_id_table_list = ids_segment,
  meta_data_record_check = md1_segment[,
    c("STUDY_SEGMENT", "SEGMENT_RECORD_CHECK")],
  study_data = study_data,
  meta_data = meta_data,
  meta_data_level = md1_segment
)

## End(Not run)
```

`menu_env` `.menu_env` – *an environment for HTML menu creation*

Description

used by the dq_report2-pipeline

Usage

```
.menu_env
```

Format

An object of class `environment` of length 3.

`menu_env-menu` *Generate the menu for a report*

Description

Generate the menu for a report

Arguments

`pages`	encapsulated `list` with report pages as `tagList` objects, its names are the desired file names

Value

the html-taglist for the menu

---

menu_env_drop_down *Creates a drop-down menu*

Description

Creates a drop-down menu

Arguments

title	name of the entry in the main menu
menu_description	description, displayed, if the main menu entry itself is clicked
...	the sub-menu-entries
id	id for the entry, defaults to modified title

Value

html div object

---

menu_env_menu_entry *Create a single menu entry*

Description

Create a single menu entry

Arguments

title	of the entry
id	linked href, defaults to modified title. can be a word, then a single-page-link with an anchor tag is created.
...	additional arguments for the menu link

Value

html-a-tag object

meta_data *Data frame with metadata about the study data on variable level*

Description

Variable level metadata.

See Also

further details on variable level metadata.

meta_data_segment

meta_data_dataframe

meta_data_dataframe *Well known columns on the* meta_data_dataframe *sheet*

Description

Meta data describing data delivered on one data frame/table sheet, e.g., a full questionnaire, not its items.

meta_data_env .meta_data_env – *an environment for easy metadata access*

Description

used by the dq_report2-pipeline

Usage

```
.meta_data_env
```

Format

An object of class environment of length 6.

See Also

meta_data_env_id_vars meta_data_env_co_vars meta_data_env_time_vars meta_data_env_group_vars

meta_data_env_co_vars *Extract co-variables for a given item*

Description

Extract co-variables for a given item

Arguments

entity	vector of item-identifiers

Value

a vector with co-variables for each entity-entry, having the explode attribute set to FALSE

See Also

meta_data_env

meta_data_env_group_vars *Extract group variables for a given item*

Description

Extract group variables for a given item

Arguments

entity	vector of item-identifiers

Value

a vector with possible group-variables (can be more than one per item) for each entity-entry, having the explode attribute set to TRUE

See Also

meta_data_env

`meta_data_env_id_vars` *Extract id variables for a given item or variable group*

Description

Extract id variables for a given item or variable group

Extract selected outlier criteria for a given item or variable group

Extract outlier rules-number-threshold for a given item or variable group

Arguments

`entity`	vector of item- or variable group identifiers

Details

In the environment, `target_meta_data` should be set either to `item_level` or to `cross-item_level`.

In the environment, `target_meta_data` should be set either to `item_level` or to `cross-item_level`.

In the environment, `target_meta_data` should be set either to `item_level` or to `cross-item_level`.

Value

a vector with id-variables for each entity-entry, having the `explode` attribute set to `FALSE`

a vector with id-variables for each entity-entry, having the `explode` attribute set to `FALSE`

a vector with id-variables for each entity-entry, having the `explode` attribute set to `FALSE`

See Also

meta_data_env

meta_data_env

meta_data_env

`meta_data_env_time_vars`
Extract measurement time variable for a given item

Description

Extract measurement time variable for a given item

Arguments

`entity`	vector of item-identifiers

Value

a vector with time-variables (usually one per item) for each entity-entry, having the `explode` attribute set to `TRUE`

See Also

meta_data_env

`meta_data_segment` *Well known columns on the* `meta_data_segment` *sheet*

Description

Meta data describing study segments, e.g., a full questionnaire, not its items.

`nres` *return the number of result slots in a report*

Description

return the number of result slots in a report

Usage

```
nres(x)
```

Arguments

`x`	the `dataquieR` report (v2.0)

Value

the number of used result slots

`pipeline_recursive_result` *Convert a pipeline result data frame to named encapsulated lists*

Description

This function converts a data frame to a recursive list structure based on columns selected for grouping

Usage

```
pipeline_recursive_result(
  call_plan_with_results,
  result_groups = setdiff(colnames(call_plan_with_results), c(NA, "results",
    "resp_vars"))
)
```

Arguments

`call_plan_with_results`	data.frame result from pipeline_vectorized
`result_groups`	character arguments to group by

Details

The data frame columns for the arguments of a certain row/computation from the calling plan translate to levels in the encapsulated list hierarchy. The order of the levels can be specified in the `result_groups` argument.

Value

a list with:

- first argument's values in `result_groups`, each containing second's argument's values as a similar list recursively

Examples

```
## Not run:
call_plan_with_results <- structure(list(
  resp_vars =
    c(
      "SBP_0", "DBP_0", "VO2_CAPCAT_0",
      "BSG_0"
    ), group_vars = c(
    "USR_BP_0", "USR_BP_0", "USR_VO2_0",
    "USR_BP_0"
  ), co_vars = list("SEX_0", "SEX_0", "SEX_0", "SEX_0")
```

```
),
class = "data.frame", row.names = c(
  NA,
  -4L
)
)
call_plan_with_results[["results"]] <-
  list(NA, 2, "Hello", ggplot2::ggplot())
result_groups <-
  colnames(call_plan_with_results)[2:(ncol(call_plan_with_results) - 1)]
pipeline_recursive_result(call_plan_with_results, result_groups)
pipeline_recursive_result(call_plan_with_results, rev(result_groups))

## End(Not run)
```

`pipeline_vectorized` *Call (nearly) one "Accuracy" function with many parameterizations at once automatically*

Description

This is a function to automatically call indicator functions of the "Accuracy" dimension in a vectorized manner with a set of parameterizations derived from the metadata.

Usage

```
pipeline_vectorized(
  fct,
  resp_vars = NULL,
  study_data,
  meta_data,
  label_col,
  ...,
  key_var_names,
  cores = list(mode = "socket", logging = FALSE, load.balancing = TRUE),
  variable_roles = list(resp_vars = list(VARIABLE_ROLES$PRIMARY,
    VARIABLE_ROLES$SECONDARY), group_vars = VARIABLE_ROLES$PROCESS),
  result_groups,
  use_cache = FALSE,
  compute_plan_only = FALSE
)
```

Arguments

`fct`	function function to call
`resp_vars`	variable list the name of the measurement variables, if NULL (default), all variables are used.
`study_data`	data.frame the data frame that contains the measurements

meta_data	data.frame the data frame that contains metadata attributes of study data
label_col	variable attribute the name of the column in the metadata with labels of variables
...	additional arguments for the function
key_var_names	character character vector named by arguments to be filled by meta data GROUP_VAR- entries as follows: c(group_vars = GROUP_VAR_OBSERVER) – may be missing, then all possible combinations will be analyzed. Cannot contain resp_vars.
cores	integer number of cpu cores to use or a named list with arguments for parallelMap::parallelStart or NULL, if parallel has already been started by the caller.
variable_roles	list restrict each function argument (referred to by its name matching a name in names(variable_roles)) to variables of the role given here.
result_groups	character columns to group results to encapsulated lists or NULL receive a data frame with all call arguments and their respective results in a column 'result' – see pipeline_recursive_result
use_cache	logical set to FALSE to omit re-using already distributed study- and metadata on a parallel cluster
compute_plan_only	logical set to TRUE to omit computations and return only the compute plan filled with planned evaluations. used in pipelines.

Details

The function to call is given as first argument. All arguments of the called functions can be given here, but pipline_vectorized can derive technically possible values (most of them) from the metadata, which can be controlled using the arguments key_var_names and variable_roles. The function returns an encapsulated list by default, but it can also return a data.frame. See also pipeline_recursive_result for these two options. The argument use_cache controls, whether the input data (study_data and meta_data) should be passed around, if running in parallel or being distributed beforehand to the compute nodes. All calls will be done in parallel, if possible. This can be configured, see argument cores below.

If the function is called in a larger framework (such as dq_report), then compute_plan_only controls, not to actually call functions but return a data.frame with parameterizations of "Accuracy" functions only. Also in such a scenario, one may want not to start and stop a cluster with entry and leaving of pipeline_vectorized but use an existing cluster. This can be achieved by setting the cores argument NULL.

Value

- if result_groups is set, a list with:
  - first argument's values in result_groups, each containing second's argument's values as a similar list recursively;
- if result_groups is not set, a data frame with one row per function call, all the arguments of each call in its columns and a column results providing the function calls' results.

Examples

```
## Not run:  # really long-running example
load(system.file("extdata/study_data.RData", package = "dataquieR"))
load(system.file("extdata/meta_data.RData", package = "dataquieR"))
a <- pipeline_vectorized(
  fct = acc_margins, study_data = study_data,
  meta_data = meta_data, label_col = LABEL,
  key_var_names = c(group_vars = GROUP_VAR_OBSERVER)
)
b <- pipeline_vectorized(
  fct = acc_margins, study_data = study_data,
  meta_data = meta_data, label_col = LABEL
)
b_adj <-
  pipeline_vectorized(
    fct = acc_margins, study_data = study_data,
    meta_data = meta_data, label_col = LABEL, co_vars = c("SEX_1", "AGE_1")
  )
c <- pipeline_vectorized(
  fct = acc_loess, study_data = study_data,
  meta_data = meta_data, label_col = LABEL,
  variable_roles = list(
    resp_vars = list(VARIABLE_ROLES$PRIMARY),
    group_vars = VARIABLE_ROLES$PROCESS
  )
)
d <- pipeline_vectorized(
  fct = acc_loess, study_data = study_data,
  meta_data = meta_data, label_col = LABEL,
  variable_roles = list(
    resp_vars = list(VARIABLE_ROLES$PRIMARY, VARIABLE_ROLES$SECONDARY),
    group_vars = VARIABLE_ROLES$PROCESS
  )
)
e <- pipeline_vectorized(
  fct = acc_margins, study_data = study_data,
  meta_data = meta_data, label_col = LABEL,
  key_var_names = c(group_vars = GROUP_VAR_OBSERVER), co_vars = "SEX_0"
)

f <- pipeline_vectorized(
  fct = acc_margins, study_data = study_data,
  meta_data = meta_data, label_col = LABEL,
  key_var_names = c(group_vars = GROUP_VAR_OBSERVER), co_vars = "SEX_0",
  result_groups = NULL
)
pipeline_recursive_result(f)
g <- pipeline_vectorized(
  fct = acc_margins, study_data = study_data,
  meta_data = meta_data, label_col = LABEL,
  key_var_names = c(group_vars = GROUP_VAR_OBSERVER), co_vars = "SEX_0",
  result_groups = c("co_vars")
```

```
)
g1 <- pipeline_vectorized(
  fct = acc_margins, study_data = study_data,
  meta_data = meta_data, label_col = LABEL,
  key_var_names = c(group_vars = GROUP_VAR_OBSERVER), co_vars = "SEX_0",
  result_groups = c("group_vars")
)
g2 <- pipeline_vectorized(
  fct = acc_margins, study_data = study_data,
  meta_data = meta_data, label_col = LABEL,
  key_var_names = c(group_vars = GROUP_VAR_OBSERVER), co_vars = "SEX_0",
  result_groups = c("group_vars", "co_vars")
)
g3 <- pipeline_vectorized(
  fct = acc_margins, study_data = study_data,
  meta_data = meta_data, label_col = LABEL,
  key_var_names = c(group_vars = GROUP_VAR_OBSERVER), co_vars = "SEX_0",
  result_groups = c("co_vars", "group_vars")
)
g4 <- pipeline_vectorized(
  fct = acc_margins, study_data = study_data,
  meta_data = meta_data, label_col = LABEL,
  co_vars = "SEX_0", result_groups = c("co_vars")
)
meta_datax <- meta_data
meta_datax[9, "GROUP_VAR_DEVICE"] <- "v00011"
g5 <- pipeline_vectorized(
  fct = acc_margins, study_data = study_data,
  meta_data = meta_datax, label_col = LABEL,
  co_vars = "SEX_0", result_groups = c("co_vars")
)
g6 <- pipeline_vectorized(
  fct = acc_margins, study_data = study_data,
  meta_data = meta_datax, label_col = LABEL,
  co_vars = "SEX_0", result_groups = c("co_vars", "group_vars")
)

## End(Not run)
```

`prep_add_cause_label_df` *Convert missing codes in metadata format v1.0 and a missing-cause-table to v2.0 missing list / jump list assignments*

Description

The function has to working modes. If `replace_meta_data` is `TRUE`, by default, if `cause_label_df` contains a column named `resp_vars`, then the missing/jump codes in `meta_data[, c(MISSING_CODES, JUMP_CODES)]` will be overwritten, otherwise, it will be labeled using the `cause_label_df`.

Usage

```
prep_add_cause_label_df(
  meta_data = "item_level",
  cause_label_df,
  label_col = VAR_NAMES,
  assume_consistent_codes = TRUE,
  replace_meta_data = ("resp_vars" %in% colnames(cause_label_df))
)
```

Arguments

`meta_data`	data.frame the data frame that contains metadata attributes of study data.
`cause_label_df`	data.frame missing code table. If missing codes have labels the respective data frame can be specified here, see cause_label_df
`label_col`	variable attribute the name of the column in the metadata with labels of variables
`assume_consistent_codes`	logical if TRUE and no labels are given and the same missing/jump code is used for more than one variable, the labels assigned for this code will be the same for all variables.
`replace_meta_data`	logical if `TRUE`, ignore existing missing codes and jump codes and replace them with data from the `cause_label_df`. Otherwise, copy the labels from `cause_label_df` to the existing code columns.

Details

If a column `resp_vars` exists, then rows with a value in `resp_vars` will only be used for the corresponding variable.

Value

data.frame updated metadata including all the code labels in missing/jump lists

See Also

prep_extract_cause_label_df

`prep_add_data_frames` *Add data frames to the pre-loaded / cache data frame environment*

Description

These can be referred to by their names, then, wherever `dataquieR` expects a data.frame – just pass a character instead. If this character is not found, `dataquieR` would additionally look for files with the name and for `URLs`. You can also refer to specific sheets of a workbook or specific object from an `RData` by appending a pipe symbol and its name. A second pipe symbol allows to extract certain columns from such sheets (but they will remain data frames).

Usage

```
prep_add_data_frames(..., data_frame_list = list())
```

Arguments

...	data frames, if passed with names, these will be the names of these tables in the data frame environment. If not, then the names in the calling environment will be used.
data_frame_list	a named list with data frames. Also these will be added and names will be handled as for the ... argument.

Value

data.frame invisible(the cache environment)

See Also

prep_load_workbook_like_file

prep_get_data_frame

---

prep_add_missing_codes

Insert missing codes for NA*s based on rules*

Description

Insert missing codes for NAs based on rules

Usage

```
prep_add_missing_codes(
  resp_vars,
  study_data,
  meta_data,
  label_col,
  rules,
  use_value_labels,
  overwrite = FALSE
)
```

Arguments

`resp_vars`	variable list the name of the measurement variables to be modified, all from `rules`, if omitted
`study_data`	data.frame the data frame that contains the measurements
`meta_data`	data.frame the data frame that contains metadata attributes of study data
`label_col`	variable attribute the name of the column in the metadata with labels of variables
`rules`	data.frame with the columns: • `resp_vars`: Variable, whose `NA`-values should be replaced by jump codes • `CODE_CLASS`: Either `MISSING` or `JUMP`: Is the currently described case an expected missing value (`JUMP`) or not (`MISSING`) • `CODE_VALUE`: The jump code or missing code • `CODE_LABEL`: A label describing the reason for the missing value • `RULE`: A rule in `REDcap` style (see, e.g., `REDcap` help, `REDcap` how-to), and `REDcap` branching logic that describes cases for the missing
`use_value_labels`	logical In rules for factors, use the value labels, not the codes. Defaults to `TRUE`, if any `VALUE_LABELS` are given in the metadata.
`overwrite`	logical Also insert missing codes, if the values are not `NA`

Value

a `list` with the entries:

- `ModifiedStudyData`: Study data with `NA`s replaced by the `CODE_VALUE`
- `ModifiedMetaData`: Metadata having the new codes amended in the columns `JUMP_LIST` or `MISSING_LIST`, respectively

Examples

```
## Not run:
load(system.file("extdata", "study_data.RData", package = "dataquieR"))
load(system.file("extdata", "meta_data.RData", package = "dataquieR"))
vn <- subset(r$ModifiedMetaData, LABEL == "PREGNANT_0", VAR_NAMES)[[1]]
rules <- tibble::tribble(
  ~resp_vars, ~CODE_CLASS, ~CODE_LABEL, ~CODE_VALUE, ~RULE,
  "PREGNANT_0", "JUMP", "No pregnancies in males", "9999", '[SEX_0]=1',
 )
 r <- prep_add_missing_codes(NA, study_data, meta_data,
   label_col = "LABEL", rules, use_value_labels = FALSE)
 subset(r$ModifiedMetaData, LABEL == "PREGNANT_0", JUMP_LIST)
 subset(meta_data, LABEL == "PREGNANT_0", JUMP_LIST)
 table(study_data[[vn]])
 table(r$ModifiedStudyData[[vn]])
 r <- prep_add_missing_codes(NA, study_data, meta_data,
   label_col = "LABEL", rules, use_value_labels = FALSE, overwrite = TRUE)
 table(study_data[[vn]])
 table(r$ModifiedStudyData[[vn]])
```

```
rules <- tibble::tribble(
  ~resp_vars, ~CODE_CLASS, ~CODE_LABEL, ~CODE_VALUE, ~RULE,
  "PREGNANT_0", "JUMP", "No pregnancies in males", "9999", '[SEX_0]="males"',
 )
 r <- prep_add_missing_codes(NA, study_data, meta_data,
   label_col = "LABEL", rules, use_value_labels = TRUE, overwrite = FALSE)
 table(study_data[[vn]])
 table(r$ModifiedStudyData[[vn]])

rules <- tibble::tribble(
  ~resp_vars, ~CODE_CLASS, ~CODE_LABEL, ~CODE_VALUE, ~RULE,
  "PREGNANT_0", "JUMP", "No pregs in males", "9999", '[v00002]="males"',
 )
 r <- prep_add_missing_codes(NA, study_data, meta_data,
   label_col = "LABEL", rules, use_value_labels = TRUE, overwrite = FALSE)
 table(study_data[[vn]])
 table(r$ModifiedStudyData[[vn]])
 devtools::load_all(".")

study_data$v00002 <- ifelse(study_data$v00002 == "0", "females", "males")
meta_data[meta_data$LABEL == "SEX_0", "VALUE_LABELS"] <- "females|males"
rules <- tibble::tribble(
  ~resp_vars, ~CODE_CLASS, ~CODE_LABEL, ~CODE_VALUE, ~RULE,
  "PREGNANT_0", "JUMP", "No pregnancies in males", "9999", '[v00002]="males"',
)
r <- prep_add_missing_codes(NA, study_data, meta_data,
                    label_col = "LABEL", rules, use_value_labels = TRUE, overwrite = FALSE)
table(study_data[[vn]])
table(r$ModifiedStudyData[[vn]])

## End(Not run)
```

---

prep_add_to_meta *Support function to augment metadata during data quality reporting*

Description

adds an annotation to static metadata

Usage

```
prep_add_to_meta(
  VAR_NAMES,
  DATA_TYPE,
  LABEL,
  VALUE_LABELS,
  meta_data = "item_level",
  ...
)
```

Arguments

VAR_NAMES	character Names of the Variables to add
DATA_TYPE	character Data type for the added variables
LABEL	character Labels for these variables
VALUE_LABELS	character Value labels for the values of the variables as usually pipe separated and assigned with =: `1 = male | 2 = female`
meta_data	data.frame the metadata to extend
...	Further defined variable attributes, see prep_create_meta

Details

Add metadata e.g. of transformed/new variable This function is not yet considered stable, but we already export it, because it could help. Therefore, we have some inconsistencies in the formals still.

Value

a data frame with amended meta data.

---

prep_apply_coding *Re-Code labels with their respective codes according to the meta_data*

Description

Re-Code labels with their respective codes according to the meta_data

Usage

```
prep_apply_coding(study_data, meta_data = "item_level")
```

Arguments

study_data	data.frame the data frame that contains the measurements
meta_data	data.frame the data frame that contains metadata attributes of study data

Value

data.frame modified study data with labels replaced by the codes

prep_check_meta_data_dataframe *Verify and normalize meta data on data frame level*

Description

if possible, mismatching data types are converted ("true" becomes TRUE)

Usage

```
prep_check_meta_data_dataframe(meta_data_dataframe = "dataframe_level")
```

Arguments

meta_data_dataframe
: data.frame data frame or path/url of a metadata sheet for the data frame level

Details

missing columns are added, filled with NA, if this is valid, i.e., n.a. for DF_NAME as the key column

Value

standardized metadata sheet as data frame

Examples

```
## Not run:
mds <- prep_check_meta_data_dataframe("ship_meta_dataframe|dataframe_level") # also converts
print(mds)
prep_check_meta_data_dataframe(mds)
mds1 <- mds
mds1$DF_RECORD_COUNT <- NULL
print(prep_check_meta_data_dataframe(mds1)) # fixes the missing column by NAs
mds1 <- mds
mds1$DF_UNIQUE_ROWS[[2]] <- "xxx" # not convertible
# print(prep_check_meta_data_dataframe(mds1)) # fail
mds1 <- mds
mds1$DF_UNIQUE_ID[[2]] <- 12 # not yet supported
# print(prep_check_meta_data_dataframe(mds1)) # fail

## End(Not run)
```

prep_check_meta_data_segment *Verify and normalize meta data on segment level*

Description

if possible, mismatching data types are converted ("true" becomes TRUE)

Usage

```
prep_check_meta_data_segment(meta_data_segment = "segment_level")
```

Arguments

meta_data_segment
: data.frame data frame or path/url of a metadata sheet for the segment level

Details

missing columns are added, filled with NA, if this is valid, i.e., n.a. for STUDY_SEGMENT as the key column

Value

standardized metadata sheet as data frame

Examples

```
## Not run:
mds <- prep_check_meta_data_segment("ship_meta_v2|segment_level") # also converts
print(mds)
prep_check_meta_data_segment(mds)
mds1 <- mds
mds1$SEGMENT_RECORD_COUNT <- NULL
print(prep_check_meta_data_segment(mds1)) # fixes the missing column by NAs
mds1 <- mds
mds1$SEGMENT_UNIQUE_ROWS[[2]] <- "xxx" # not convertible
# print(prep_check_meta_data_segment(mds1)) # fail

## End(Not run)
```

`prep_check_meta_names` *Checks the validity of meta data w.r.t. the provided column names*

Description

This function verifies, if a data frame complies to meta data conventions and provides a given richness of meta information as specified by `level`.

Usage

```
prep_check_meta_names(meta_data = "item_level", level, character.only = FALSE)
```

Arguments

`meta_data`	data.frame the data frame that contains metadata attributes of study data
`level`	enum level of requirement (see also VARATT_REQUIRE_LEVELS). set to `NULL` to deactivate the check of richness.
`character.only`	logical a logical indicating whether level can be assumed to be character strings.

Details

Note, that only the given level is checked despite, levels are somehow hierarchical.

Value

a logical with:

- invisible(TRUE). In case of problems with the meta data, a condition is raised (`stop()`).

Examples

```
## Not run:
prep_check_meta_names(data.frame(VAR_NAMES = 1, DATA_TYPE = 2,
                      MISSING_LIST = 3))

prep_check_meta_names(
  data.frame(
    VAR_NAMES = 1, DATA_TYPE = 2, MISSING_LIST = 3,
    LABEL = "LABEL", VALUE_LABELS = "VALUE_LABELS",
    JUMP_LIST = "JUMP_LIST", HARD_LIMITS = "HARD_LIMITS",
    GROUP_VAR_OBSERVER = "GROUP_VAR_OBSERVER",
    GROUP_VAR_DEVICE = "GROUP_VAR_DEVICE",
    TIME_VAR = "TIME_VAR",
    PART_VAR = "PART_VAR",
    STUDY_SEGMENT = "STUDY_SEGMENT",
    LOCATION_RANGE = "LOCATION_RANGE",
    LOCATION_METRIC = "LOCATION_METRIC",
    PROPORTION_RANGE = "PROPORTION_RANGE",
    MISSING_LIST_TABLE = "MISSING_LIST_TABLE",
```

```
    CO_VARS = "CO_VARS",
    LONG_LABEL = "LONG_LABEL"
  ),
  RECOMMENDED
)

prep_check_meta_names(
  data.frame(
    VAR_NAMES = 1, DATA_TYPE = 2, MISSING_LIST = 3,
    LABEL = "LABEL", VALUE_LABELS = "VALUE_LABELS",
    JUMP_LIST = "JUMP_LIST", HARD_LIMITS = "HARD_LIMITS",
    GROUP_VAR_OBSERVER = "GROUP_VAR_OBSERVER",
    GROUP_VAR_DEVICE = "GROUP_VAR_DEVICE",
    TIME_VAR = "TIME_VAR",
    PART_VAR = "PART_VAR",
    STUDY_SEGMENT = "STUDY_SEGMENT",
    LOCATION_RANGE = "LOCATION_RANGE",
    LOCATION_METRIC = "LOCATION_METRIC",
    PROPORTION_RANGE = "PROPORTION_RANGE",
    DETECTION_LIMITS = "DETECTION_LIMITS", SOFT_LIMITS = "SOFT_LIMITS",
    CONTRADICTIONS = "CONTRADICTIONS", DISTRIBUTION = "DISTRIBUTION",
    DECIMALS = "DECIMALS", VARIABLE_ROLE = "VARIABLE_ROLE",
    DATA_ENTRY_TYPE = "DATA_ENTRY_TYPE",
    CO_VARS = "CO_VARS",
    VARIABLE_ORDER = "VARIABLE_ORDER", LONG_LABEL =
      "LONG_LABEL", recode = "recode",
      MISSING_LIST_TABLE = "MISSING_LIST_TABLE"
  ),
  OPTIONAL
)

# Next one will fail
try(
  prep_check_meta_names(data.frame(VAR_NAMES = 1, DATA_TYPE = 2,
    MISSING_LIST = 3), TECHNICAL)
)

## End(Not run)
```

---

prep_clean_labels *Support function to scan variable labels for applicability*

Description

Adjust labels in meta_data to be valid variable names in formulas for diverse r functions, such as `glm` or `lme4::lmer`.

Usage

```
prep_clean_labels(label_col, meta_data = "item_level", no_dups = FALSE)
```

Arguments

label_col	character label attribute to adjust or character vector to adjust, depending on meta_data argument is given or missing.
meta_data	data.frame meta data frame: If label_col is a label attribute to adjust, this is the meta data table to process on. If missing, label_col must be a character vector with values to adjust.
no_dups	logical disallow duplicates in input or output vectors of the function, then, prep_clean_labels would call stop() on duplicated labels.

Details

Currently, labels as given by label_col arguments in the most functions are directly used in formula, so that they become natural part of the outputs, but different models expect differently strict syntax for such formulas, especially for valid variable names. prep_clean_labels removes all potentially inadmissible characters from variable names (no guarantee, that some exotic model still rejects the names, but minimizing the number of exotic characters). However, variable names are modified, may become unreadable or indistinguishable from other variable names. For the latter case, a stop call is possible, controlled by the no_dups argument.

A warning is emitted, if modifications were necessary.

Value

a data.frame with:

- if meta_data is set, a list with:
  - modified meta_data[, label_col] column
- if meta_data is not set, adjusted labels that then were directly given in label_col

Examples

```
## Not run:
meta_data1 <- data.frame(
  LABEL =
    c(
      "syst. Blood pressure (mmHg) 1",
      "1st heart frequency in MHz",
      "body surface (\\u33A1)"
    )
)
print(meta_data1)
print(prep_clean_labels(meta_data1$LABEL))
meta_data1 <- prep_clean_labels("LABEL", meta_data1)
print(meta_data1)

## End(Not run)
```

prep_create_meta *Support function to create data.frames of metadata*

Description

Create a meta data frame and map names. Generally, this function only creates a data.frame, but using this constructor instead of calling `data.frame(..., stringsAsFactors = FALSE)`, it becomes possible, to adapt the metadata data.frame in later developments, e.g. if we decide to use classes for the metadata, or if certain standard names of variable attributes change. Also, a validity check is possible to implement here.

Usage

```
prep_create_meta(..., stringsAsFactors = FALSE, level, character.only = FALSE)
```

Arguments

`...`	named column vectors, names will be mapped using WELL_KNOWN_META_VARIABLE_NAMES, if included in WELL_KNOWN_META_VARIABLE_NAMES can also be a data frame, then its column names will be mapped using WELL_KNOWN_META_VARIABLE_NAMES
`stringsAsFactors`	logical if the argument is a list of vectors, a data frame will be created. In this case, `stringsAsFactors` controls, whether characters will be auto-converted to Factors, which defaults here always to false independent from the default.stringsAsFactors.
`level`	enum level of requirement (see also VARATT_REQUIRE_LEVELS) set to `NULL`, if not a complete metadata frame is created.
`character.only`	logical a logical indicating whether level can be assumed to be character strings.

Details

For now, this calls data.frame, but it already renames variable attributes, if they have a different name assigned in WELL_KNOWN_META_VARIABLE_NAMES, e.g. `WELL_KNOWN_META_VARIABLE_NAMES$RECODE` maps to `recode` in lower case.

NB: `dataquieR` exports all names from WELL_KNOWN_META_VARIABLE_NAME as symbols, so `RECODE` also contains `"recode"`.

Value

a data frame with:

- meta data attribute names mapped and
- meta data checked using prep_check_meta_names and do some more verification about conventions, such as check for valid intervals in limits)

See Also

WELL_KNOWN_META_VARIABLE_NAMES

prep_datatype_from_data

Get data types from data

Description

Get data types from data

Usage

```
prep_datatype_from_data(resp_vars = colnames(study_data), study_data)
```

Arguments

resp_vars	variable names of the variables to fetch the data type from the data
study_data	data.frame the data frame that contains the measurements Hint: Only data frames supported, no URL or file names.

Value

vector of data types

Examples

```
## Not run:
dataquieR::prep_datatype_from_data(cars)

## End(Not run)
```

prep_deparse_assignments

Convert two vectors from a code-value-table to a key-value list

Description

Convert two vectors from a code-value-table to a key-value list

Usage

```
prep_deparse_assignments(codes, labels, split_char = SPLIT_CHAR)
```

Arguments

codes	integer codes
labels	character labels, same length as codes
split_char	character split character character to split code assignments

Value

a vector with assignment strings for each row of `cbind(codes, labels)`

---

prep_dq_data_type_of *Get the dataquieR* DATA_TYPE *of* x

Description

Get the dataquieR DATA_TYPE of x

Usage

```
prep_dq_data_type_of(x)
```

Arguments

x	object to define the dataquieR data type of

Value

the dataquieR data type as listed in DATA_TYPES

See Also

DATA_TYPES_OF_R_TYPE

---

prep_expand_codes *Expand code labels across variables*

Description

Code labels are copied from other variables, if the code is the same and the label is set only for some variables

Usage

```
prep_expand_codes(
  meta_data = "item_level",
  suppressWarnings = FALSE,
  mix_jumps_and_missings = FALSE
)
```

Arguments

meta_data	data.frame the data frame that contains metadata attributes of study data
suppressWarnings	logical show warnings, if labels are expanded
mix_jumps_and_missings	logical ignore the class of the codes for label expansion, i.e., use missing code labels as jump code labels, if the values are the same.

Value

data.frame an updated meta data frame.

Examples

```
## Not run:
load(system.file("extdata", "meta_data.RData", package = "dataquieR"))
meta_data$JUMP_LIST[meta_data$VAR_NAMES == "v00003"] <- "99980 = NOOP"
md <- prep_expand_codes(meta_data)
md$JUMP_LIST
md$MISSING_LIST
md <- prep_expand_codes(meta_data, mix_jumps_and_missings = TRUE)
md$JUMP_LIST
md$MISSING_LIST
load(system.file("extdata", "meta_data.RData", package = "dataquieR"))
meta_data$MISSING_LIST[meta_data$VAR_NAMES == "v00003"] <- "99980 = NOOP"
md <- prep_expand_codes(meta_data)
md$JUMP_LIST
md$MISSING_LIST

## End(Not run)
```

---

prep_extract_cause_label_df

Extract all missing/jump codes from metadata and export a cause-label-data-frame

---

Description

Extract all missing/jump codes from metadata and export a cause-label-data-frame

Usage

```
prep_extract_cause_label_df(meta_data = "item_level", label_col = VAR_NAMES)
```

Arguments

meta_data	data.frame the data frame that contains metadata attributes of study data
label_col	variable attribute the name of the column in the metadata with labels of variables

Value

list with the entries

- `meta_data` data.frame a data frame that contains updated metadata
- `cause_label_df` data.frame missing code table. If missing codes have labels the respective data frame are specified here, see cause_label_df

See Also

prep_add_cause_label_df

`prep_get_data_frame` *Read data from files/URLs*

Description

data_frame_name can be a file path or an URL you can append a pipe and a sheet name for Excel files or object name e.g. for `RData` files. Numbers may also work. All file formats supported by your `rio` installation will work.

Usage

```
prep_get_data_frame(data_frame_name, .data_frame_list = .dataframe_environment)
```

Arguments

`data_frame_name`
: character name of the data frame to read, see details

`.data_frame_list`
: environment cache for loaded data frames

Details

The data frames will be cached automatically, you can define an alternative environment for this using the argument `.data_frame_list`, and you can purge the cache using prep_purge_data_frame_cache.

Use prep_add_data_frames to manually add data frames to the cache, e.g., if you have loaded them from more complex sources, before.

Value

data.frame a data frame

See Also

prep_add_data_frames

prep_load_workbook_like_file

Examples

```
## Not run:
bl <- as.factor(prep_get_data_frame(
  paste0("https://www.rki.de/DE/Content/InfAZ/N/Neuartiges_Coronavirus",
    "/Projekte_RKI/COVID-19_Todesfaelle.xlsx?__blob=",
    "publicationFile|COVID_Todesfälle_BL|Bundesland"))[[1]])

n <- as.numeric(prep_get_data_frame(paste0(
  "https://www.rki.de/DE/Content/InfAZ/N/Neuartiges_Coronavirus/",
  "Projekte_RKI/COVID-19_Todesfaelle.xlsx?__blob=",
  "publicationFile|COVID_Todesfälle_BL|Anzahl verstorbene",
  " COVID-19 Fälle"))[[1]])
plot(bl, n)
# Working names would be to date (2022-10-21), e.g.:
#
# https://www.rki.de/DE/Content/InfAZ/N/Neuartiges_Coronavirus/ \
#    Projekte_RKI/COVID-19_Todesfaelle.xlsx?__blob=publicationFile
# https://www.rki.de/DE/Content/InfAZ/N/Neuartiges_Coronavirus/  \
#    Projekte_RKI/COVID-19_Todesfaelle.xlsx?__blob=publicationFile|2
# https://www.rki.de/DE/Content/InfAZ/N/Neuartiges_Coronavirus/ \
#    Projekte_RKI/COVID-19_Todesfaelle.xlsx?__blob=publicationFile|name
# study_data
# ship
# meta_data
# ship_meta
#
prep_get_data_frame("meta_data | meta_data")
prep_get_data_frame(file.path(system.file(package = "dataquieR"),
  "extdata", "meta_data.RData"))
prep_get_data_frame(file.path(system.file(package = "dataquieR"),
  "extdata", "meta_data.RData|meta_data"))

## End(Not run)
```

---

prep_get_user_name *Return the logged-in User's Full Name*

Description

If `whoami` is not installed, the user name from `Sys.info()` is returned.

Usage

```
prep_get_user_name()
```

Details

Can be overridden by options or environment:

`options(FULLNAME = "Stephan Struckmann")`

`Sys.setenv(FULLNAME = "Stephan Struckmann")`

Value

character the user's name

`prep_link_escape` *Prepare a label as part of a link for* `RMD` *files*

Description

Prepare a label as part of a link for `RMD` files

Usage

```
prep_link_escape(s, html = FALSE)
```

Arguments

`s`	the label
`html`	prepare the label for direct `HTML` output instead of `RMD`

Value

the escaped label

`prep_list_dataframes` *List Loaded Data Frames*

Description

List Loaded Data Frames

Usage

```
prep_list_dataframes()
```

Value

names of all loaded data frames

prep_load_workbook_like_file *Pre-load a file with named (usually more than) one table(s)*

Description

These can thereafter be referred to by their names only. Such files are, e.g., spreadsheet-workbooks or RData-files.

Usage

```
prep_load_workbook_like_file(file)
```

Arguments

file	the file name to load.

Details

Note, that this function in contrast to prep_get_data_frame does neither support URLs nor selecting specific sheets/columns from a file.

Value

data.frame invisible(the cache environment)

See Also

prep_add_data_frames

prep_get_data_frame

Examples

```
## Not run:
file_name <-
  system.file("extdata", "meta_data_extended.xlsx", package = "dataquieR")
prep_load_workbook_like_file(file_name)
prep_get_data_frame(
  "dataframe_level") # dataframe_level is a sheet in the file

## End(Not run)
```

prep_map_labels *Support function to allocate labels to variables*

Description

Map variables to certain attributes, e.g. by default their labels.

Usage

```
prep_map_labels(
  x,
  meta_data = "item_level",
  to = LABEL,
  from = VAR_NAMES,
  ifnotfound,
  warn_ambiguous = FALSE
)
```

Arguments

x	character variable names, character vector, see parameter from
meta_data	data.frame meta data frame, if, as a dataquieR developer, you do not have **item-level-metadata**, you should use util_map_labels instead to avoid consistency checks on for item-level meta_data.
to	character variable attribute to map to
from	character variable identifier to map from
ifnotfound	list A list of values to be used if the item is not found: it will be coerced to a list if necessary.
warn_ambiguous	logical print a warning if mapping variables from from to to produces ambiguous identifiers.

Details

This function basically calls colnames(study_data) <- meta_data$LABEL, ensuring correct merging/joining of study data columns to the corresponding meta data rows, even if the orders differ. If a variable/study_data-column name is not found in meta_data[[from]] (default from = VAR_NAMES), either stop is called or, if ifnotfound has been assigned a value, that value is returned. See mget, which is internally used by this function.

The function not only maps to the LABEL column, but to can be any metadata variable attribute, so the function can also be used, to get, e.g. all HARD_LIMITS from the metadata.

Value

a character vector with:

- mapped values

Examples

```
## Not run:
meta_data <- prep_create_meta(
  VAR_NAMES = c("ID", "SEX", "AGE", "DOE"),
  LABEL = c("Pseudo-ID", "Gender", "Age", "Examination Date"),
  DATA_TYPE = c(DATA_TYPES$INTEGER, DATA_TYPES$INTEGER, DATA_TYPES$INTEGER,
                DATA_TYPES$DATETIME),
  MISSING_LIST = ""
)
stopifnot(all(prep_map_labels(c("AGE", "DOE"), meta_data) == c("Age",
                                                    "Examination Date")))

## End(Not run)
```

---

prep_merge_study_data *Merge a list of study data frames to one (sparse) study data frame*

Description

Merge a list of study data frames to one (sparse) study data frame

Usage

```
prep_merge_study_data(study_data_list)
```

Arguments

study_data_list	list the list

Value

data.frame study_data

---

prep_meta_data_v1_to_item_level_meta_data *Convert item-level metadata from v1.0 to v2.0*

Description

This function is idempotent..

Usage

```
prep_meta_data_v1_to_item_level_meta_data(
  meta_data = "item_level",
  verbose = TRUE,
  label_col = LABEL,
  cause_label_df
)
```

Arguments

meta_data	data.frame the old item-level-metadata
verbose	logical display all estimated decisions
label_col	variable attribute the name of the column in the metadata with labels of variables
cause_label_df	data.frame missing code table, see cause_label_df. Optional. If this argument is given, you can add missing code tables.

Details

The options("dataquieR.force_item_specific_missing_codes") (default FALSE) tells the system, to always fill in res_vars columns to the MISSING_LIST_TABLE, even, if the column already exists, but is empty.

Value

data.frame the updated metadata

---

prep_min_obs_level *Support function to identify the levels of a process variable with minimum number of observations*

Description

utility function to subset data based on minimum number of observation per level

Usage

```
prep_min_obs_level(study_data, group_vars, min_obs_in_subgroup)
```

Arguments

study_data	data.frame the data frame that contains the measurements
group_vars	variable list the name grouping variable
min_obs_in_subgroup	integer optional argument if a "group_var" is used. This argument specifies the minimum no. of observations that is required to include a subgroup (level) of the "group_var" in the analysis. Subgroups with less observations are excluded. The default is 30.

Details

This functions removes observations having less than `min_obs_in_subgroup` distinct values in a group variable, e.g. blood pressure measurements performed by an examiner having less than e.g. 50 measurements done. It displays a warning, if samples/rows are removed and returns the modified study data frame.

Value

a data frame with:

- a subsample of original data

`prep_pmap` *Support function for a parallel* `pmap`

Description

parallel version of `purrr::pmap`

Usage

```
prep_pmap(.l, .f, ..., cores = 0)
```

Arguments

`.l`	data.frame with one call per line and one function argument per column
`.f`	`function` to call with the arguments from `.l`
`...`	additional, static arguments for calling `.f`
`cores`	number of cpu cores to use or a (named) list with arguments for parallelMap::parallelStart or NULL, if parallel has already been started by the caller. Set to 0 to run without parallelization.

Value

list of results of the function calls

Author(s)

Aurèle

S Struckmann

See Also

`purrr::pmap`

Stack Overflow post

prep_prepare_dataframes *Prepare and verify study data with metadata*

Description

This function ensures, that a data frame ds1 with suitable variable names study_data and meta_data exist as base data.frames.

Usage

```
prep_prepare_dataframes(
  .study_data,
  .meta_data,
  .label_col,
  .replace_hard_limits,
  .replace_missings,
  .sm_code = NULL,
  .allow_empty = FALSE
)
```

Arguments

.study_data	if provided, use this data set as study_data
.meta_data	if provided, use this data set as meta_data
.label_col	if provided, use this as label_col
.replace_hard_limits	replace HARD_LIMIT violations by NA, defaults to FALSE.
.replace_missings	replace missing codes, defaults to TRUE
.sm_code	missing code for NAs, if they have been re-coded by util_combine_missing_lists
.allow_empty	allow ds1 to be empty. i.e., 0 rows and/or 0 columns

Details

This function defines ds1 and modifies study_data and meta_data in the environment of its caller (see eval.parent). It also defines or modifies the object label_col in the calling environment. Almost all functions exported by dataquieR call this function initially, so that aspects common to all functions live here, e.g. testing, if an argument meta_data has been given and features really a data.frame. It verifies the existence of required metadata attributes (VARATT_REQUIRE_LEVELS). It can also replace missing codes by NAs, and calls prep_study2meta to generate a minimum set of metadata from the study data on the fly (should be amended, so on-the-fly-calling is not recommended for an instructive use of dataquieR).

The function also detects tibbles, which are then converted to base-R data.frames, which are expected by dataquieR.

Different from the other utility function that work in the caller's environment, so it modifies objects in the calling function. It defines a new object `ds1`, it modifies `study_data` and/or `meta_data` and `label_col`.

Value

`ds1` the study data with mapped column names

See Also

acc_margins

Examples

```
## Not run:
acc_test1 <- function(resp_variable, aux_variable,
                      time_variable, co_variables,
                      group_vars, study_data, meta_data) {
  prep_prepare_dataframes()
  invisible(ds1)
}
acc_test2 <- function(resp_variable, aux_variable,
                      time_variable, co_variables,
                      group_vars, study_data, meta_data, label_col) {
  ds1 <- prep_prepare_dataframes(study_data, meta_data)
  invisible(ds1)
}
environment(acc_test1) <- asNamespace("dataquieR")
# perform this inside the package (not needed for functions that have been
# integrated with the package already)

environment(acc_test2) <- asNamespace("dataquieR")
# perform this inside the package (not needed for functions that have been
# integrated with the package already)
acc_test3 <- function(resp_variable, aux_variable, time_variable,
                      co_variables, group_vars, study_data, meta_data,
                      label_col) {
  prep_prepare_dataframes()
  invisible(ds1)
}
acc_test4 <- function(resp_variable, aux_variable, time_variable,
                      co_variables, group_vars, study_data, meta_data,
                      label_col) {
  ds1 <- prep_prepare_dataframes(study_data, meta_data)
  invisible(ds1)
}
environment(acc_test3) <- asNamespace("dataquieR")
# perform this inside the package (not needed for functions that have been
# integrated with the package already)

environment(acc_test4) <- asNamespace("dataquieR")
# perform this inside the package (not needed for functions that have been
```

```
# integrated with the package already)
load(system.file("extdata/meta_data.RData", package = "dataquieR"))
load(system.file("extdata/study_data.RData", package = "dataquieR"))
try(acc_test1())
try(acc_test2())
acc_test1(study_data = study_data)
try(acc_test1(meta_data = meta_data))
try(acc_test2(study_data = 12, meta_data = meta_data))
print(head(acc_test1(study_data = study_data, meta_data = meta_data)))
print(head(acc_test2(study_data = study_data, meta_data = meta_data)))
print(head(acc_test3(study_data = study_data, meta_data = meta_data)))
print(head(acc_test3(study_data = study_data, meta_data = meta_data,
  label_col = LABEL)))
print(head(acc_test4(study_data = study_data, meta_data = meta_data)))
print(head(acc_test4(study_data = study_data, meta_data = meta_data,
  label_col = LABEL)))
try(acc_test2(study_data = NULL, meta_data = meta_data))

## End(Not run)
```

`prep_purge_data_frame_cache` *Title*

Description

Title

Usage

```
prep_purge_data_frame_cache()
```

Value

TODO

`prep_study2meta` *Guess a meta data frame from study data.*

Description

Guess a minimum meta data frame from study data. Minimum required variable attributes are:

Usage

```
prep_study2meta(
  study_data,
  level = c(VARATT_REQUIRE_LEVELS$REQUIRED, VARATT_REQUIRE_LEVELS$RECOMMENDED),
  cumulative = TRUE,
  convert_factors = FALSE
)
```

Arguments

study_data	data.frame the data frame that contains the measurements
level	enum levels to provide (see also VARATT_REQUIRE_LEVELS)
cumulative	logical include attributes of all levels up to level
convert_factors	logical convert the

Details

```
dataquieR:::util_get_var_att_names_of_level(VARATT_REQUIRE_LEVELS$REQUIRED)
#>      VAR_NAMES      DATA_TYPE   MISSING_LIST
#>    "VAR_NAMES"    "DATA_TYPE" "MISSING_LIST"
```

The function also tries to detect missing codes.

Value

a meta_data data frame

---

prep_title_escape *Prepare a label as part of a title text for* RMD *files*

Description

Prepare a label as part of a title text for RMD files

Usage

```
prep_title_escape(s, html = FALSE)
```

Arguments

s	the label
html	prepare the label for direct HTML output instead of RMD

Value

the escaped label

prep_valuelabels_from_data

Get value labels from data

Description

Detects factors and converts them to compatible metadata/study data.

Usage

```
prep_valuelabels_from_data(resp_vars = colnames(study_data), study_data)
```

Arguments

resp_vars	variable names of the variables to fetch the value labels from the data
study_data	data.frame the data frame that contains the measurements

Value

a list with:

- VALUE_LABELS: vector of value labels and modified study data
- ModifiedStudyData: study data with factors as integers

Examples

```
## Not run:
dataquieR::prep_datatype_from_data(iris)

## End(Not run)
```

print.dataquieR_result

Print a dataquieR result returned by pipeline_vectorized

Description

Print a dataquieR result returned by pipeline_vectorized

Usage

```
## S3 method for class 'dataquieR_result'
print(x, ...)
```

Arguments

x	list a dataquieR result from pipeline_vectorized
...	passed to print. Additionally, the argument slot may be passed to print only specific sub-results.

Value

see print

---

print.dataquieR_resultset

Generate a RMarkdown-based report from a dataquieR report

Description

Generate a RMarkdown-based report from a dataquieR report

Usage

```
## S3 method for class 'dataquieR_resultset'
print(x, dir, view = TRUE, self_contained = FALSE, ...)
```

Arguments

x	dataquieR report.
dir	character directory to store the rendered report's files, a temporary one, if omitted. Directory will be created, if missing, files may be overwritten inside that directory
view	logical display the report
self_contained	logical create a single page application HTML file. This may be quite big and hard to render for your web-browser.
...	additional arguments: • template: Report template to use, not yet supported. • chunk_error: display error messages in report • chunk_warning: display warnings in report • output_format: output format to use, see rmarkdown::render – currently, html based formats are supported by the default template. If set, the argument self_contained will be ignored. • chunk_echo: display R code in report • chunk_message: display message outputs in report

Value

file name of the generated report

print.dataquieR_resultset2 *Generate a RMarkdown-based report from a dataquieR report*

Description

Generate a RMarkdown-based report from a dataquieR report

Usage

```
## S3 method for class 'dataquieR_resultset2'
print(x, dir, view = TRUE, disable_plotly = FALSE, ...)
```

Arguments

x	dataquieR report v2.
dir	character directory to store the rendered report's files, a temporary one, if omitted. Directory will be created, if missing, files may be overwritten inside that directory
view	logical display the report
disable_plotly	logical do not use plotly, even if installed
...	additional arguments:

Value

file names of the generated report's HTML files

print.interval *print implementation for the class* interval

Description

such objects, for now, only occur in RECCap rules, so this function is meant for internal use, mostly – for now.

Usage

```
## S3 method for class 'interval'
print(x, ...)
```

Arguments

x	interval objects to print
...	not used yet

Value

the printed object

See Also

base::print

`print.ReportSummaryTable`

print implementation for the class `ReportSummaryTable`

Description

Use this function to print results objects of the class `ReportSummaryTable`.

Usage

```
## S3 method for class 'ReportSummaryTable'
print(
  x,
  relative,
  dt = FALSE,
  fillContainer = FALSE,
  displayValues = FALSE,
  view = TRUE,
  ...,
  flip_mode = "auto"
)
```

Arguments

`x`	`ReportSummaryTable` objects to print
`relative`	logical normalize the values in each column by division by the `N` column.
`dt`	logical use `DT::datatables`, if installed
`fillContainer`	logical if `dt` is `TRUE`, control table size, see `DT::datatables`.
`displayValues`	logical if `dt` is `TRUE`, also display the actual values
`view`	logical if `view` is `FALSE`, do not print but return the output, only
`...`	not used, yet
`flip_mode`	enum default | flip | noflip | auto. Should the plot be in default orientation, flipped, not flipped or auto-flipped. Not all options are always supported. In general, this con be controlled by setting the `roptions(dataquieR.flip_mode = ...)`. If called from `dq_report`, you can also pass `flip_mode` to all function calls or set them specifically using `specific_args`.

Value

the printed object

See Also

base::print

`pro_applicability_matrix` *Check applicability of DQ functions on study data*

Description

Checks applicability of DQ functions based on study data and metadata characteristics

Usage

```
pro_applicability_matrix(
  study_data,
  meta_data,
  split_segments = FALSE,
  label_col,
  max_vars_per_plot = 20,
  meta_data_segment,
  meta_data_dataframe,
  flip_mode = "noflip"
)
```

Arguments

`study_data`	data.frame the data frame that contains the measurements
`meta_data`	data.frame the data frame that contains metadata attributes of study data
`split_segments`	logical return one matrix per study segment
`label_col`	variable attribute the name of the column in the metadata with labels of variables
`max_vars_per_plot`	integer from=0. The maximum number of variables per single plot.
`meta_data_segment`	data.frame – optional: Segment level metadata
`meta_data_dataframe`	data.frame – optional: Data frame level metadata
`flip_mode`	enum default | flip | noflip | auto. Should the plot be in default orientation, flipped, not flipped or auto-flipped. Not all options are always supported. In general, this con be controlled by setting the `roptions(dataquieR.flip_mode = ...)`. If called from `dq_report`, you can also pass `flip_mode` to all function calls or set them specifically using `specific_args`.

Details

This is a preparatory support function that compares study data with associated metadata. A prerequisite of this function is that the no. of columns in the study data complies with the no. of rows in the metadata.

For each existing R-implementation, the function searches for necessary static metadata and returns a heatmap like matrix indicating the applicability of each data quality implementation.

In addition, the data type defined in the metadata is compared with the observed data type in the study data.

Value

a list with:

- `SummaryTable`: data frame about the applicability of each indicator function (each function in a column). its integer values can be one of the following four categories: 0. Non-matching datatype + Incomplete metadata, 1. Non-matching datatype + complete metadata, 2. Matching datatype + Incomplete metadata, 3. Matching datatype + complete metadata, 4. Not applicable according to data type
- `ApplicabilityPlot`: ggplot2 heatmap plot, graphical representation of `SummaryTable`
- `ApplicabilityPlotList`: list of plots per (maybe artificial) segment
- `ReportSummaryTable`: data frame underlying `ApplicabilityPlot`

Examples

```
## Not run:
load(system.file("extdata/meta_data.RData", package = "dataquieR"), envir =
  environment())
load(system.file("extdata/study_data.RData", package = "dataquieR"), envir =
  environment())
appmatrix <- pro_applicability_matrix(study_data = study_data,
                                      meta_data = meta_data,
                                      label_col = LABEL)

## End(Not run)
```

`rbind.ReportSummaryTable` *Combine* `ReportSummaryTable` *outputs*

Description

Using this `rbind` implementation, you can combine different heatmap-like results of the class `ReportSummaryTable`.

Usage

```
## S3 method for class 'ReportSummaryTable'
rbind(...)
```

Arguments

...	ReportSummaryTable objects to combine.

See Also

base::rbind.data.frame

---

resnames *Return names of result slots (e.g., 3rd dimension of dataquieR results)*

Description

Return names of result slots (e.g., 3rd dimension of dataquieR results)

Usage

```
resnames(x)
```

Arguments

x	the objects

Value

character vector with names

---

resnames.dataquieR_resultset2 *Return names of result slots (e.g., 3rd dimension of dataquieR results)*

Description

Return names of result slots (e.g., 3rd dimension of dataquieR results)

Usage

```
## S3 method for class 'dataquieR_resultset2'
resnames(x)
```

Arguments

x	the objects

Value

character vector with names

SEGMENT_ID_TABLE *Segment level metadata attribute name*

Description

The name of the data frame containing the reference IDs to be compared with the IDs in the targeted segment.

Usage

```
SEGMENT_ID_TABLE
```

Format

An object of class `character` of length 1.

See Also

meta_data_segment

SEGMENT_ID_VARS *Segment level metadata attribute name*

Description

All variables that are to be used as one single ID variable (combined key) in a segment.

Usage

```
SEGMENT_ID_VARS
```

Format

An object of class `character` of length 1.

See Also

meta_data_segment

SEGMENT_PART_VARS *Segment level metadata attribute name*

Description

The name of the segment participation status variable

Usage

```
SEGMENT_PART_VARS
```

Format

An object of class `character` of length 1.

See Also

meta_data_segment

SEGMENT_RECORD_CHECK *Segment level metadata attribute name*

Description

The type of check to be conducted when comparing the reference ID table with the IDs in a segment.

Usage

```
SEGMENT_RECORD_CHECK
```

Format

An object of class `character` of length 1.

See Also

meta_data_segment

SEGMENT_RECORD_COUNT *Segment level metadata attribute name*

Description

Number of expected data records in each segment. numeric. Check only conducted if number entered

Usage

```
SEGMENT_RECORD_COUNT
```

Format

An object of class `character` of length 1.

See Also

meta_data_segment

SEGMENT_UNIQUE_ROWS *Segment level metadata attribute name*

Description

Specifies whether identical data is permitted across rows in a segment (excluding ID variables)

Usage

```
SEGMENT_UNIQUE_ROWS
```

Format

An object of class `character` of length 1.

See Also

meta_data_segment

SPLIT_CHAR *Character used by default as a separator in meta data such as missing codes*

Description

This 1 character is according to our metadata concept "|".

Usage

```
SPLIT_CHAR
```

Format

An object of class `character` of length 1.

study_data *Data frame with the study data whose quality is being assessed*

Description

Study data is expected in wide format. If should contain all variables for all segments in one large table, even, if some variables are not measured for all observational utils (study participants).

summary.dataquieR_resultset *Summarize a dataquieR report*

Description

Summarizes a dataquieR report extracting all GRADING results.

Usage

```
## S3 method for class 'dataquieR_resultset'
summary(
  object,
  aspect = c("issue", "applicability", "error"),
  return_the_value = TRUE,
  ...
)
```

Arguments

`object`	dataquieR report.
`aspect`	what sort of issues to summarize
`return_the_value`	logical return the `GRADING` or error message, a color otherwise.
`...`	not used yet.

Value

a data.frame with one row per variable and one column per `GRADING` result. Each function providing a `GRADING` conforming to the standards is represented by a column. `GRADING` expresses the presence of a problem with `0` = no | `1` = yes

Examples

```
## Not run:
# runs spuriously slow on rhub
load(system.file("extdata/meta_data.RData", package = "dataquieR"), envir =
  environment())
load(system.file("extdata/study_data.RData", package = "dataquieR"), envir =
  environment())
report <- suppressWarnings(dq_report(
  variables = head(meta_data[[LABEL]], 5),
  study_data, meta_data,
  cores = 1,
  label_col = LABEL, dimensions =
  c( # for sake of speed, omit Accuracy here
       "Consistency")
))
x <- summary(report)

## End(Not run)
```

---

summary.dataquieR_resultset2

Generate a report summary table

Description

Generate a report summary table

Usage

```
## S3 method for class 'dataquieR_resultset2'
summary(
  object,
  aspect = c("applicability", "error", "issue"),
```

```
  FUN = util_get_html_cell_for_result,
  collapse = "\n<br />\n",
  ...
)
```

Arguments

object	a square result set
aspect	an aspect/problem category of results
FUN	function to apply to the cells of the result table
collapse	passed to FUN
...	not used

Value

a summary of a Square2 report

Examples

```
## Not run:
  util_html_table(summary(report, aspect = "error", FUN = util_get_html_cell_for_result),
       filter = "top", options = list(scrollCollapse = TRUE, scrollY = "75vh"),
       is_matrix_table = TRUE, rotate_headers = TRUE, output_format = "HTML"
  )

## End(Not run)
```

---

util_abbreviate *Abbreviate snake_case function names to shortened* CamelCase

---

Description

Abbreviate snake_case function names to shortened CamelCase

Usage

```
util_abbreviate(x)
```

Arguments

x	a vector of indicator function names

Value

abbreviations

util_adjust_geom_text_for_plotly *Place all geom_texts also in* plotly *right from the x position*

Description

Place all geom_texts also in plotly right from the x position

Usage

```
util_adjust_geom_text_for_plotly(plotly)
```

Arguments

plotly	the plotly

Value

modified plotly-built object

util_alias2caption *Create a caption from an alias name of a* dq_report2 *result*

Description

Create a caption from an alias name of a dq_report2 result

Usage

```
util_alias2caption(alias)
```

Arguments

alias	alias name

Value

caption

util_all_intro_vars_for_rv *Get all* PART_VARS *for a response variable (from item-level metadata)*

Description

Get all PART_VARS for a response variable (from item-level metadata)

Usage

```
util_all_intro_vars_for_rv(
  rv,
  study_data,
  meta_data,
  label_col = LABEL,
  expected_observations = c("HIERARCHY", "ALL", "SEGMENT")
)
```

Arguments

rv	character the response variable's name
study_data	study_data
meta_data	meta_data
label_col	character the metadata attribute to map meta_data on study_data based on colnames(study_data)
expected_observations	enum HIERARCHY | ALL | SEGMENT. How should PART_VARS be handled: - ALL: Ignore, all observations are expected - SEGMENT: if PART_VAR is 1, an observation is expected - HIERARCHY: the default, if the PART_VAR is 1 for this variable and also for all PART_VARS of PART_VARS up in the hierarchy, an observation is expected.

Value

character all PART_VARS for rv from item level metadata. For expected_observations = HIERARCHY, the more general PART_VARS (i.e., up, in the hierarchy) are more left in the vector, e.g.: PART_STUDY, PART_PHYSICAL_EXAMI

util_all_is_integer *convenience function to abbreviate all(util_is_integer(...))*

Description

convenience function to abbreviate all(util_is_integer(...))

Usage

```
util_all_is_integer(x)
```

Arguments

x	the object to test

Value

TRUE, if all entries are integer-like, FALSE otherwise

util_anytime_installed *Test, if package* anytime *is installed*

Description

Test, if package anytime is installed

Usage

```
util_anytime_installed()
```

Value

TRUE if anytime is installed.

See Also

requireNamespace

https://community.rstudio.com/t/how-can-i-make-testthat-think-i-dont-have-a-package-installed/33441/2

util_app_cd *utility function for the applicability of contradiction checks*

Description

Test for applicability of contradiction checks

Usage

```
util_app_cd(x, dta)
```

Arguments

x	data.frame metadata
dta	logical vector, 1=matching data type, 0 = non-matching data type

Value

factor 0-3 for each variable in metadata

- 0 data type mismatch and not applicable
- 1 data type mismatches but applicable
- 2 data type matches but not applicable
- 3 data type matches and applicable

util_app_con_contradictions_redcap *utility function for the applicability of contradiction checks*

Description

Test for applicability of contradiction checks

Usage

```
util_app_con_contradictions_redcap(x, dta)
```

Arguments

x	data.frame metadata
dta	logical vector, 1=matching data type, 0 = non-matching data type

Value

factor 0-3 for each variable in metadata

- 0 data type mismatch and not applicable
- 1 data type mismatches but applicable
- 2 data type matches but not applicable
- 3 data type matches and applicable

`util_app_dc` *utility function for the applicability of of distribution plots*

Description

Test for applicability of distribution plots

Usage

```
util_app_dc(x, dta)
```

Arguments

`x`	data.frame metadata
`dta`	logical vector, 1=matching data type, 0 = non-matching data type

Value

factor 0-3 for each variable in metadata

- 0 data type mismatch and not applicable
- 1 data type mismatches but applicable
- 2 data type matches but not applicable
- 3 data type matches and applicable
- 4 not applicable because of not suitable data type

util_app_dl *utility function to test for applicability of detection limits checks*

Description

Test for applicability of detection limits checks

Usage

```
util_app_dl(x, dta)
```

Arguments

x	data.frame metadata
dta	logical vector, 1=matching data type, 0 = non-matching data type

Value

factor 0-3 for each variable in metadata

- 0 data type mismatch and not applicable
- 1 data type mismatches but applicable
- 2 data type matches but not applicable
- 3 data type matches and applicable
- 4 not applicable because of not suitable data type

util_app_ed *utility function for the applicability of of end digits preferences checks*

Description

Test for applicability of end digits preferences checks

Usage

```
util_app_ed(x, dta)
```

Arguments

x	data.frame metadata
dta	logical vector, 1=matching data type, 0 = non-matching data type

Value

factor 0-3 for each variable in metadata

- 0 data type mismatch and not applicable
- 1 data type mismatches but applicable
- 2 data type matches but not applicable
- 3 data type matches and applicable
- 4 not applicable because of not suitable data type

`util_app_hl` *utility function to test for applicability of hard limits checks*

Description

Test for applicability of hard limits checks

Usage

```
util_app_hl(x, dta)
```

Arguments

`x`	data.frame metadata
`dta`	logical vector, 1=matching data type, 0 = non-matching data type

Value

factor 0-3 for each variable in metadata

- 0 data type mismatch and not applicable
- 1 data type mismatches but applicable
- 2 data type matches but not applicable
- 3 data type matches and applicable
- 4 not applicable because of not suitable data type

util_app_iac *utility function for the applicability of categorical admissibility*

Description

Test for applicability of categorical admissibility

Usage

```
util_app_iac(x, dta)
```

Arguments

x	data.frame metadata
dta	logical vector, 1=matching data type, 0 = non-matching data type

Value

factor 0-3 for each variable in metadata

- 0 data type mismatch and not applicable
- 1 data type mismatches but applicable
- 2 data type matches but not applicable
- 3 data type matches and applicable
- 4 not applicable because of not suitable data type

util_app_iav *utility function for the applicability of numeric admissibility*

Description

Test for applicability of numeric admissibility

Usage

```
util_app_iav(x, dta)
```

Arguments

x	data.frame metadata
dta	logical vector, 1=matching data type, 0 = non-matching data type

Value

factor 0-3 for each variable in metadata

- 0 data type mismatch and not applicable
- 1 data type mismatches but applicable
- 2 data type matches but not applicable
- 3 data type matches and applicable
- 4 not applicable because of not suitable data type

`util_app_im` *utility function applicability of item missingness*

Description

Test for applicability of item missingness

Usage

```
util_app_im(x, dta)
```

Arguments

`x`	data.frame metadata
`dta`	logical vector, 1=matching data type, 0 = non-matching data type

Value

factor 0-3 for each variable in metadata

- 0 data type mismatch and not applicable
- 1 data type mismatches but applicable
- 2 data type matches but not applicable
- 3 data type matches and applicable

util_app_loess *utility function for applicability of LOESS smoothed time course plots*

Description

Test for applicability of LOESS smoothed time course plots

Usage

```
util_app_loess(x, dta)
```

Arguments

x	data.frame metadata
dta	logical vector, 1=matching data type, 0 = non-matching data type

Value

factor 0-3 for each variable in metadata

- 0 data type mismatch and not applicable
- 1 data type mismatches but applicable
- 2 data type matches but not applicable
- 3 data type matches and applicable
- 4 not applicable because of not suitable data type

util_app_mar *utility function to test for applicability of marginal means plots*

Description

Test for applicability of detection limits checks

Usage

```
util_app_mar(x, dta)
```

Arguments

x	data.frame metadata
dta	logical vector, 1 = matching data type, 0 = non-matching data type

Value

factor 0-3 for each variable in metadata

- 0 data type mismatch and not applicable
- 1 data type mismatches but applicable
- 2 data type matches but not applicable
- 3 data type matches and applicable
- 4 not applicable because of not suitable data type

`util_app_mol` *utility function applicability of multivariate outlier detection*

Description

Test for applicability of multivariate outlier detection

Usage

```
util_app_mol(x, dta)
```

Arguments

`x`	data.frame metadata
`dta`	logical vector, 1=matching data type, 0 = non-matching data type

Value

factor 0-3 for each variable in metadata

- 0 data type mismatch and not applicable
- 1 data type mismatches but applicable
- 2 data type matches but not applicable
- 3 data type matches and applicable
- 4 not applicable because of not suitable data type

util_app_ol *utility function for the applicability of outlier detection*

Description

Test for applicability of univariate outlier detection

Usage

```
util_app_ol(x, dta)
```

Arguments

x	data.frame metadata
dta	logical vector, 1=matching data type, 0 = non-matching data type

Value

factor 0-3 for each variable in metadata

- 0 data type mismatch and not applicable
- 1 data type mismatches but applicable
- 2 data type matches but not applicable
- 3 data type matches and applicable
- 4 not applicable because of not suitable data type

util_app_sl *utility function to test for applicability of soft limits checks*

Description

Test for applicability of soft limits checks

Usage

```
util_app_sl(x, dta)
```

Arguments

x	data.frame metadata
dta	logical vector, 1=matching data type, 0 = non-matching data type

Value

factor 0-3 for each variable in metadata

- 0 data type mismatch and not applicable
- 1 data type mismatches but applicable
- 2 data type matches but not applicable
- 3 data type matches and applicable
- 4 not applicable because of not suitable data type

`util_app_sm` *utility function applicability of segment missingness*

Description

Test for applicability of segment missingness

Usage

```
util_app_sm(x, dta)
```

Arguments

`x`	data.frame metadata
`dta`	logical vector, 1=matching data type, 0 = non-matching data type

Value

factor 0-3 for each variable in metadata

- 0 data type mismatch and not applicable
- 1 data type mismatches but applicable
- 2 data type matches but not applicable
- 3 data type matches and applicable

util_app_sos *utility function applicability of distribution function's shape or scale check*

Description

Test for applicability of checks for deviation form expected probability distribution shapes/scales

Usage

```
util_app_sos(x, dta)
```

Arguments

x	data.frame metadata
dta	logical vector, 1=matching data type, 0 = non-matching data type

Value

factor 0-3 for each variable in metadata

- 0 data type mismatch and not applicable
- 1 data type mismatches but applicable
- 2 data type matches but not applicable
- 3 data type matches and applicable
- 4 not applicable because of not suitable data type

util_app_vc *utility applicability variance components*

Description

Test for applicability of ICC

Usage

```
util_app_vc(x, dta)
```

Arguments

x	data.frame metadata
dta	logical vector, 1=matching data type, 0 = non-matching data type

Value

factor 0-3 for each variable in metadata

- 0 data type mismatch and not applicable
- 1 data type mismatches but applicable
- 2 data type matches but not applicable
- 3 data type matches and applicable
- 4 not applicable because of not suitable data type

`util_assign_levlabs` *utility function to assign labels to levels*

Description

function to assign labels to levels of a variable

Usage

```
util_assign_levlabs(
  variable,
  string_of_levlabs,
  splitchar,
  assignchar,
  ordered = TRUE,
  variable_name = ""
)
```

Arguments

`variable`	vector vector with values of a study variable
`string_of_levlabs`	character len=1. value labels, e.g. `1 = no | 2 = yes`
`splitchar`	character len=1. splitting character(s) in `string_of_levlabs`, usually SPLIT_CHAR
`assignchar`	character len=1. assignment operator character(s) in `string_of_levlabs`, usually `=` or `:`
`ordered`	the function converts `variable` to a factor, by default to an ordered factor assuming LHS of assignments being meaningful numbers, e.g. `1 = low | 2 = medium | 3 = high`. If no special order is given, set `ordered` to `FALSE`, e.g. for `1 = male | 2 = female` or `1 = low | 2 = high | 3 = medium`.
`variable_name`	character the name of the variable being converted for warning messages

Value

a data.frame with labels assigned to categorical variables (if available)

util_as_color *convert a character to a specific ordered factor*

Description

convert a character to a specific ordered factor

Usage

```
util_as_color(x)
```

Arguments

x	color name

Value

factor

util_as_numeric *Convert factors to label-corresponding numeric values*

Description

Converts a vector factor aware of numeric values not being scrambled.

Usage

```
util_as_numeric(v, warn)
```

Arguments

v	the vector
warn	if not missing: character with error message stating conversion error

Value

the converted vector

util_attach_attr *Attach attributes to an object and return it*

Description

Attach attributes to an object and return it

Usage

```
util_attach_attr(x, ...)
```

Arguments

x	the object
...	named arguments, each becomes an attributes

Value

x, having the desired attributes attached

util_backtickQuote *utility function to set string in backticks*

Description

Quote a set of variable names with backticks

Usage

```
util_backtickQuote(x)
```

Arguments

x	variable names

Value

quoted variable names

util_bQuote *Put in back-ticks*

Description

also escape potential back-ticks in x

Usage

```
util_bQuote(x)
```

Arguments

x	a string

Value

x in back-ticks

util_check_data_type *Verify the data type of a value*

Description

Function to verify the data type of a value.

Usage

```
util_check_data_type(
  x,
  type,
  check_convertible = FALSE,
  threshold_value = 0,
  return_counts = FALSE
)
```

Arguments

x	the value
type	expected data type
check_convertible	logical also try, if a conversion to the declared data type would work.
threshold_value	numeric from=0 to=100. percentage failing conversions allowed if check_convertible is TRUE.
return_counts	logical return the counts instead of logical values about threshold violations.

Value

if `check_convertible` is `FALSE`, logical whether x is of the expected type if `check_convertible` is `TRUE` integer with the states `0, 1, 2`: 0 = Mismatch, not convertible 1 = Match 2 = Mismatch, but convertible

---

`util_check_group_levels`

Check data for observer levels

Description

Check data for observer levels

Usage

```
util_check_group_levels(
  study_data,
  group_vars,
  min_obs_in_subgroup = -Inf,
  max_obs_in_subgroup = +Inf,
  min_subgroups = -Inf,
  max_subgroups = +Inf
)
```

Arguments

`study_data`	data.frame the data frame that contains the measurements
`group_vars`	variable the name of the observer, device or reader variable
`min_obs_in_subgroup`	integer from=0. optional argument if `group_vars` are used. This argument specifies the minimum number of observations that is required to include a subgroup (level) of the group variable named by `group_vars` in the analysis. Subgroups with fewer observations are excluded.
`max_obs_in_subgroup`	integer from=0. optional argument if `group_vars` are used. This argument specifies the maximum number of observations that is required to include a subgroup (level) of the group variable named by `group_vars` in the analysis. Subgroups with more observations are excluded.
`min_subgroups`	integer from=0. optional argument if a "group_var" is used. This argument specifies the minimum no. of subgroups (levels) included "group_var". If the variable defined in "group_var" has fewer subgroups it is splitted for analysis.
`max_subgroups`	integer from=0. optional argument if a "group_var" is used. This argument specifies the maximum no. of subgroups (levels) included "group_var". If the variable defined in "group_var" has more subgroups it is splitted for analysis.

Value

modified study data frame

Examples

```
## Not run:
study_data <- prep_get_data_frame("study_data")
meta_data <- prep_get_data_frame("meta_data")
prep_prepare_dataframes(.label_col = LABEL)
util_check_group_levels(ds1, "CENTER_0")
dim(util_check_group_levels(ds1, "USR_BP_0", min_obs_in_subgroup = 400))

## End(Not run)
```

---

util_check_one_unique_value

Check for one value only

Description

utility function to identify variables with one value only.

Usage

```
util_check_one_unique_value(x)
```

Arguments

x	vector with values

Value

logical(1): TRUE, if – except NA – exactly only one value is observed in x, FALSE otherwise

---

util_combine_missing_lists

Combine missing-lists for a set of variables to be displayed in the same heat-map

---

Description

Combine missing-lists for a set of variables to be displayed in the same heat-map

Usage

```
util_combine_missing_lists(
  resp_vars,
  study_data,
  meta_data,
  label_col,
  include_sysmiss,
  cause_label_df,
  assume_consistent_codes = TRUE,
  expand_codes = assume_consistent_codes,
  suppressWarnings = FALSE
)
```

Arguments

`resp_vars`	variable list the name of the measurement variables
`study_data`	data.frame the data frame that contains the measurements
`meta_data`	data.frame the data frame that contains metadata attributes of study data
`label_col`	variable attribute the name of the column in the metadata with labels of variables
`include_sysmiss`	logical Optional, if TRUE system missingness (NAs) is evaluated in the summary plot
`cause_label_df`	data.frame missing code table. If missing codes have labels the respective data frame can be specified here, see cause_label_df
`assume_consistent_codes`	logical if TRUE and no labels are given and the same missing/jump code is used for more than one variable, the labels assigned for this code will be the same for all variables.
`expand_codes`	logical if TRUE, code labels are copied from other variables, if the code is the same and the label is set somewhere
`suppressWarnings`	logical warn about consistency issues with missing and jump lists

Value

a list with:

- `ModifiedStudyData`: data frame with re-coded (if needed) study data
- `cause_label_df`: data frame with re-coded missing codes suitable for all variables

util_compare_meta_with_study

Compares study data data types with the ones expected according to the metadata

Description

Utility function to compare data type of study data with those defined in metadata

Usage

```
util_compare_meta_with_study(
  sdf,
  mdf,
  label_col,
  check_convertible = FALSE,
  threshold_value = 0
)
```

Arguments

sdf	the data.frame of study data
mdf	the data.frame of associated static meta data
label_col	variable attribute the name of the column in the metadata with labels of variables
check_convertible	logical also try, if a conversion to the declared data type would work.
threshold_value	numeric from=0 to=100. percentage failing conversions allowed if check_convertible is TRUE.

Value

if check_convertible is FALSE, a binary vector (0, 1) if data type applies, if check_convertible is TRUE`` a vector with the states 0, 1, 2`: 0 = Mismatch, not convertible 1 = Match 2 = Mismatch, but convertible

util_condition_constructor_factory

Produce a condition function

Description

Produce a condition function

Usage

```
util_condition_constructor_factory(
  .condition_type = c("error", "warning", "message")
)
```

Arguments

.condition_type	character the type of the conditions being created and signaled by the function, "error", "warning", or "message"

---

util_coord_flip *return a flip term for* ggplot2 *plots, if desired.*

Description

return a flip term for ggplot2 plots, if desired.

Usage

```
util_coord_flip(w, h, p, ref_env, ...)
```

Arguments

w	width of the plot to determine its aspect ratio
h	height of the plot to determine its aspect ratio
p	the ggplot2 object, so far. If w or h are missing, p is used for an estimate on w and h, if both axes are discrete.
ref_env	environment of the actual entry function, so that the correct formals can be detected.
...	additional arguments for coord_flip or coord_cartesian

Value

coord_flip or coord_cartesian

util_copy_all_deps *Copy default dependencies to the report's lib directory*

Description

Copy default dependencies to the report's lib directory

Usage

```
util_copy_all_deps(dir, pages, ...)
```

Arguments

dir	report directory
pages	all pages to write
...	additional htmltools::htmlDependency objects to be added to all pages, also

Value

```
invisible(NULL)
```

util_correct_variable_use *Check referred variables*

Description

This function operates in the environment of its caller (using eval.parent, similar to Function like C-Preprocessor-Macros). Different from the other utility function that work in the caller's environment (util_prepare_dataframes), It has no side effects except that the argument of the calling function specified in arg_name is normalized (set to its default or a general default if missing, variable names being all white space replaced by NAs). It expects two objects in the caller's environment: ds1 and meta_data. meta_data is the meta data frame and ds1 is produced by a preceding call of util_prepare_dataframes using meta_data and study_data.

Usage

```
util_correct_variable_use(
  arg_name,
  allow_na,
  allow_more_than_one,
  allow_null,
  allow_all_obs_na,
  allow_any_obs_na,
  min_distinct_values,
```

```
  need_type,
  role = "",
  overwrite = TRUE,
  do_not_stop = FALSE,
  remove_not_found = TRUE
)

util_correct_variable_use2(
  arg_name,
  allow_na,
  allow_more_than_one,
  allow_null,
  allow_all_obs_na,
  allow_any_obs_na,
  min_distinct_values,
  need_type,
  role = arg_name,
  overwrite = TRUE,
  do_not_stop = FALSE,
  remove_not_found = TRUE
)
```

Arguments

arg_name	character Name of a function argument of the caller of util_correct_variable_use
allow_na	logical default = FALSE. allow NAs in the variable names argument given in arg_name
allow_more_than_one	logical default = FALSE. allow more than one variable names in arg_name
allow_null	logical default = FALSE. allow an empty variable name vector in the argument arg_name
allow_all_obs_na	logical default = TRUE. check observations for not being all NA
allow_any_obs_na	logical default = TRUE. check observations for being complete without any NA
min_distinct_values	integer Minimum number of distinct observed values of a study variable
need_type	character if not NA, variables must be of data type need_type according to the meta data, can be a pipe (|) separated list of allowed data types. Use ! to exclude a type. See DATA_TYPES for the predefined variable types of the dataquieR concept.
role	character variable-argument role. Set different defaults for all allow-arguments and need_type of this util_correct_variable_use.. If given, it defines the intended use of the verified argument. For typical arguments and typical use cases, roles are predefined in .variable_arg_roles. The role's defaults can be overwritten by the arguments. If role is "" (default), the standards are allow_na = FALSE, allow_more_than_one = FALSE, allow_null = FALSE,

allow_all_obs_na = TRUE, allow_any_obs_na = TRUE, and need_type = NA. Use util_correct_variable_use2 for using the arg_name as default for role. See .variable_arg_roles for currently available variable-argument roles.

overwrite logical overwrite vector of variable names to match the labels given in label_col.

do_not_stop logical do not throw an error, if one of the variables violates allow_all_obs_na, allow_any_obs_na or min_distinct_values. Instead, the variable will be removed from arg_name in the parent environment with a warning. This is helpful for functions which work with multiple variables.

remove_not_found
TODO: Not yet implemented

Details

util_correct_variable_use and util_correct_variable_use2 differ only in the default of the argument role.

util_correct_variable_use and util_correct_variable_use2 put strong effort on producing compressible error messages to the caller's caller (who is typically an end user of a dataquieR function).

The function ensures, that a specified argument of its caller that refers variable names (one or more as character vector) matches some expectations.

This function accesses the caller's environment!

See Also

.variable_arg_roles

---

util_count_expected_observations
Count Expected Observations

Description

Count participants, if an observation was expected, given the PART_VARS from item-level metadata

Usage

```
util_count_expected_observations(
  resp_vars,
  study_data,
  meta_data,
  label_col = LABEL,
  expected_observations = c("HIERARCHY", "ALL", "SEGMENT")
)
```

Arguments

resp_vars	character the response variables, for that a value may be expected
study_data	study_data
meta_data	meta_data
label_col	character mapping attribute colnames(study_data) vs. meta_data[label_col]
expected_observations	enum HIERARCHY | ALL | SEGMENT. How should PART_VARS be handled: - ALL: Ignore, all observations are expected - SEGMENT: if PART_VAR is 1, an observation is expected - HIERARCHY: the default, if the PART_VAR is 1 for this variable and also for all PART_VARS of PART_VARS up in the hierarchy, an observation is expected.

Value

a vector with the number of expected observations for each resp_vars.

---

util_count_NA *Support function to count number of* NA*s*

Description

Counts the number of NAs in x.

Usage

```
util_count_NA(x)
```

Arguments

x	object to count NAs in

Value

number of NAs

`util_create_page_file` *Create an HTML file for the dq_report2*

Description

Create an HTML file for the dq_report2

Usage

```
util_create_page_file(
  page,
  pages,
  rendered_pages,
  dir,
  template_file,
  report,
  logo,
  loading,
  packageName,
  deps
)
```

Arguments

`page`	the name of the page-file being created
`pages`	list with all page-contents named by their desired file names
`rendered_pages`	list with all rendered (`htmltools::renderTags`) page-contents named by their desired file names
`dir`	target directory
`template_file`	the report template file to use
`report`	the output of dq_report2
`logo`	logo `PNG` file
`loading`	loading animation div
`packageName`	the name of the current package
`deps`	dependencies, as pre-processed by `htmltools::copyDependencyToDir` and `htmltools::renderDepend`

Value

`invisible(file_name)`

util_deparse1 *Expression De-Parsing*

Description

Turn unevaluated expressions into character strings.

Arguments

expr	any R expression.
collapse	a string, passed to paste()
width.cutoff	integer in [20, 500] determining the cutoff (in bytes) at which line-breaking is tried.
...	further arguments passed to deparse().

Details

This is a simple utility function for R < 4.0.0 to ensure a string result (character vector of length one), typically used in name construction, as util_deparse1(substitute(.)).

This avoids a dependency on backports and on R >= 4.0.0.

Value

the deparsed expression

util_deprecate_soft *Deprecate functions and arguments*

Description

if available, it calls lifecycle::deprecate_soft. otherwise, it just shows a warning.

Usage

```
util_deprecate_soft(
  when,
  what,
  with = NULL,
  details = NULL,
  id = NULL,
  env = rlang::caller_env(),
  user_env = rlang::caller_env(2)
)
```

Arguments

when	A string giving the version when the behavior was deprecated.
what	A string describing what is deprecated
with	An optional string giving a recommended replacement for the deprecated behavior. This takes the same form as what.
details	only used, if lifecycle::deprecate_soft is available
id	only used, if lifecycle::deprecate_soft is available
env	only used, if lifecycle::deprecate_soft is available
user_env	only used, if lifecycle::deprecate_soft is available

Value

NULL, invisibly.

---

util_detect_cores *Detect cores*

Description

See parallel::detectCores for further details.

Usage

```
util_detect_cores()
```

Value

number of available CPU cores.

---

util_dichotomize *utility function to dichotomize variables*

Description

use the meta data attribute RECODE (="recode") to dichotomize the data

Usage

```
util_dichotomize(study_data, meta_data, label_col = VAR_NAMES)
```

Arguments

study_data	Study data including jump/missing codes as specified in the code conventions
meta_data	Meta data as specified in the code conventions
label_col	variable attribute the name of the column in the metadata with labels of variables

util_dist_selection *Utility function distribution-selection*

Description

This function differentiates the type of measurement variables.

Usage

```
util_dist_selection(measurements, meta_data)
```

Arguments

measurements	study data
meta_data	meta data, not yet used

Value

data frame with one column for each variable in study data giving IsInteger, IsMultCat and IsNCategory

IsInteger contains a guess, if the variable contains integer values or is a factor

IsMultCat contains a guess, if the variable has more than two categories, if it is categorical or ordinal

NCategory contains the number of distinct values detected for the variable

util_ds1_eval_env *Create an environment with several alias names for the study data variables*

Description

generates an environment similar to as.environment(ds1), but makes variables available by their VAR_NAME, LABEL, and label_col - names.

Usage

```
util_ds1_eval_env(study_data, meta_data = "item_level", label_col = LABEL)
```

Arguments

study_data	data.frame the data frame that contains the measurements
meta_data	data.frame the data frame that contains metadata attributes of study data
label_col	variable attribute the name of the column in the metadata with labels of variables. If study_data has already been mapped, i.e., util_ds1_eval_env(ds1, ...) is called, this will work too

util_empty *Test, if values of x are empty, i.e. NA or whitespace characters*

Description

Test, if values of x are empty, i.e. NA or whitespace characters

Usage

```
util_empty(x)
```

Arguments

x	the vector to test

Value

a logical vector, same length as x; TRUE, if resp. element in x is "empty"

util_ensure_character *convert a value to character*

Description

convert a value to character

Usage

```
util_ensure_character(x, error = FALSE, error_msg, ...)
```

Arguments

x	the value
error	logical if TRUE, an error is thrown, a warning otherwise in case of a conversion error
error_msg	error message to be displayed, if conversion was not possible
...	additional arguments passed to util_error or util_warning respectively in case of an error, and if an error_msg has been passed

Value

as.character(x)

`util_ensure_data_type` *Ensure matching data types*

Description

Utility function to convert selected variables in the study data to match the data types given in the metadata. If such a conversion is not possible, the study data remains unchanged.

Usage

```
util_ensure_data_type(variables, study_data, meta_data, label_col)
```

Arguments

`variables`	variable list the names of the variables
`study_data`	data.frame the data frame that contains the measurements
`meta_data`	data.frame the data frame that contains metadata attributes of study data
`label_col`	variable attribute the name of the column in the metadata with labels of variables

Value

the transformed (if necessary and possible) study data

Examples

```
## Not run:
load(system.file("extdata/meta_data.RData", package = "dataquieR"), envir =
  environment())
load(system.file("extdata/study_data.RData", package = "dataquieR"), envir =
  environment())
study_data$v00000 <- as.character(study_data$v00000)
sd1 <-
  util_ensure_data_type(
    c("CENTER_0", "AGE_0", "v00000", "v003"),
    study_data = study_data,
    meta_data = meta_data,
    label_col = "LABEL"
  )
all.equal(study_data, sd1)
study_data$v00013 <- as.numeric(study_data$v00013)
Sys.setenv(TZ = 'CET')
sd2 <-
  util_ensure_data_type(
    c("CENTER_0", "AGE_0", "v00000", "EXAM_DT_0"),
    study_data = study_data,
    meta_data = meta_data,
    label_col = "LABEL"
   )
```

```
all.equal(study_data, sd2)
all.equal(sd1$v00013, sd2$v00013)

## End(Not run)
```

`util_ensure_in` *similar to match.arg*

Description

will only warn and return a cleaned `x`.

Usage

```
util_ensure_in(x, set, err_msg, error = FALSE, applicability_problem = NA)
```

Arguments

`x`	character vector of needles
`set`	character vector representing the haystack
`err_msg`	character optional error message. Use %s twice, once for the missing elements and once for proposals
`error`	logical if `TRUE`, the execution will stop with an error, if not all `x` are elements of `set`, otherwise, it will throw a warning and "clean" the vector `x` from unexpected elements.
`applicability_problem`	logical error indicates unsuitable resp_vars

Value

character invisible(intersect(x, set))

`util_ensure_suggested` *Support function to stop, if an optional package is not installed*

Description

This function stops, if a package is not installed but needed for using an optional feature of `dataquieR`.

Usage

```
util_ensure_suggested(
  pkg,
  goal = ifelse(is.null(rlang::caller_call()), "work", paste("call",
    sQuote(rlang::call_name(rlang::caller_call())))),
  err = TRUE,
  and_import = c()
)
```

Arguments

pkg	needed package
goal	feature description for error message.
err	logical Should the function throw an error (default) or a warning?
and_import	import the listed function to the caller's environment

Value

TRUE if all packages in pkg are available, FALSE if at least one of the packages is missing.

Examples

```
## Not run:  # internal use, only
f <- function() {
  util_ensure_suggested <- get("util_ensure_suggested",
    asNamespace("dataquieR"))
  util_ensure_suggested("ggplot2", "Test",
      and_import = "(ggplot|geom_.*|aes)")
  print(ggplot(cars, aes(x = speed)) + geom_histogram())
}
f()

## End(Not run)
```

---

util_error *Produce an error message with a useful short stack trace. Then it stops the execution.*

Description

Produce an error message with a useful short stack trace. Then it stops the execution.

Usage

```
util_error(
  m,
  ...,
  applicability_problem = NA,
  integrity_indicator = "none",
  level = 0
)
```

Arguments

m	error message or a condition
...	arguments for sprintf on m, if m is a character
applicability_problem	logical error indicates unsuitable resp_vars
integrity_indicator	character the message is an integrity problem, here is the indicator abbreviation..
level	integer level of the error message (defaults to 0). Higher levels are more severe.

Value

nothing, its purpose is to stop.

---

util_evaluate_calls *Generate a full DQ report, v2*

Description

Generate a full DQ report, v2

Usage

```
util_evaluate_calls(
  all_calls,
  study_data,
  meta_data,
  label_col,
  meta_data_segment,
  meta_data_dataframe,
  meta_data_cross_item,
  resp_vars,
  filter_result_slots,
  cores,
  debug_parallel
)
```

Arguments

all_calls	list a list of calls
study_data	data.frame the data frame that contains the measurements
meta_data	data.frame the data frame that contains metadata attributes of study data
label_col	variable attribute the name of the column in the metadata with labels of variables
meta_data_segment	data.frame – optional: Segment level metadata

meta_data_dataframe
: data.frame – optional: Data frame level metadata

meta_data_cross_item
: data.frame – optional: cross-item level metadata

resp_vars
: variable list the name of the measurement variables for the report.

filter_result_slots
: character regular expressions, only if an indicator function's result's name matches one of these, it'll be used for the report. If of length zero, no filtering is performed.

cores
: integer number of cpu cores to use or a named list with arguments for parallelMap::parallelStart or NULL, if parallel has already been started by the caller.

debug_parallel
: logical print blocks currently evaluated in parallel

Value

a dataquieR_resultset2. Can be printed creating a RMarkdown-report.

---

util_eval_rule *Evaluate a parsed redcap rule for given study data*

Description

also allows to use VAR_NAMES in the rules, if other labels have been selected

Usage

```
util_eval_rule(rule, ds1, meta_data = "item_level", use_value_labels)
```

Arguments

rule
: the redcap rule (parsed, already)

ds1
: the study data as prepared by prep_prepare_dataframes

meta_data
: the metadata

use_value_labels
: map columns with VALUE_LABELS as factor variables

Value

the result of the parsed rule

util_eval_to_dataquieR_result

Evaluate an expression and create a `dataquieR_result` *object from it's evaluated value*

Description

if an error occurs, the function will return a corresponding object representing that error. all conditions will be recorded and replayed, whenever the result is printed by print.dataquieR_result.

Usage

```
util_eval_to_dataquieR_result(
  expression,
  env = parent.frame(),
  filter_result_slots
)
```

Arguments

`expression`	the expression
`env`	the environment to evaluate the expression in
`filter_result_slots`	character regular expressions, only if an indicator function's result's name matches one of these, it'll be used for the report. If of length zero, no filtering is performed.

Value

a `dataquieR_result` object

util_expect_data_frame

Verify, that argument is a data frame

Description

stops with an error, if not. will add the columns, and return the resulting extended data frame, and also updating the original data frame in the calling environment, if #' `x` is empty (data frames easily break to 0-columns in R, if they have not rows, e.g. using some `split`/`rbind` pattern)

Usage

```
util_expect_data_frame(x, col_names, convert_if_possible, dont_assign)
```

Arguments

x	an object that is verified to be a data.frame.
col_names	column names x must contain or named list of predicates to check the columns (e.g., list(AGE=is.numeric, SEX=is.character))
convert_if_possible	if given, for each column, a lambda can be given similar to col_names check functions. This lambda would be used to try a conversion. If a conversion fails (returns NA, where the input was not 'util_empty'), an error is still thrown, the data is converted, otherwise
dont_assign	set TRUE to keep x in the caller environment untouched

Value

invisible data frame

---

util_expect_scalar *check, if a scalar/vector function argument matches expectations*

Description

check, if a scalar/vector function argument matches expectations

Usage

```
util_expect_scalar(
  arg_name,
  allow_more_than_one = FALSE,
  allow_null = FALSE,
  allow_na = FALSE,
  min_length = -Inf,
  max_length = Inf,
  check_type,
  convert_if_possible,
  dont_assign = FALSE
)
```

Arguments

arg_name	the argument
allow_more_than_one	allow vectors
allow_null	allow NULL
allow_na	allow NAs
min_length	minimum length of the argument's value
max_length	maximum length of the argument's value

check_type	a predicate function, that must return TRUE on the argument's value.
convert_if_possible	if given, a lambda can be given similar to check_type This lambda would be used to try a conversion. If a conversion fails (returns NA, where the input was not 'util_empty'), an error is still thrown, the data is converted, otherwise
dont_assign	set TRUE to keep x in the caller environment untouched

Value

the value of arg_name – but this is updated in the calling frame anyway.

Examples

```
## Not run:
f <- function(x) {
  util_expect_scalar(x, check_type = is.integer)
}
f(42L)
try(f(42))
g <- function(x) {
  util_expect_scalar(x, check_type = is.integer, convert_if_possible =
          as.integer)
}
g(42L)
g(42)

## End(Not run)
```

---

util_extract_matches *return all matches of an expression*

Description

return all matches of an expression

Usage

```
util_extract_matches(data, pattern)
```

Arguments

data	TODO
pattern	TODO

Value

TODO

Author(s)

Josh O'Brien

See Also

Stack Overflow

Examples

```
## Not run:  # not exported, so not tested
dat0 <- list("a sentence with citation (Ref. 12), (Ref. 13), and then (Ref. 14)",
  "another sentence without reference")
pat <- "Ref. (\\d+)"
util_extract_matches(dat0, pat)

## End(Not run)
```

---

util_filter_missing_list_table_for_rv

Filter a MISSING_LIST_TABLE *for rows matching the variable* rv

---

Description

In MISSING_LIST_TABLE, a column resp_vars may be specified. If so, and if, for a row, this column is not empty, then that row only affects the one variable specified in that cell

Usage

```
util_filter_missing_list_table_for_rv(table, rv, rv2 = rv)
```

Arguments

table	cause_label_df a data frame with missing codes and optionally resp_vars. It also comprises labels and optionally an interpretation column with AAPOR codes. Must already cover the variable rv, i.e., item level metadata is not checked to find the suitable missing table for rv.
rv	variable the response variable to filter the missing list for specified by a label.
rv2	variable the response variable to filter the missing list for specified by a VAR_NAMES-name.

Value

data.frame the row-wise bound data frames as one data frame

util_filter_names_by_regexps

Filter collection based on its `names()` *using regular expressions*

Description

Filter collection based on its `names()` using regular expressions

Usage

```
util_filter_names_by_regexps(collection, regexps)
```

Arguments

`collection`	a named collection (list, vector, ...)
`regexps`	character a vector of regular expressions

Value

`collection` reduced to entries, that's names match at least any expression from `regexps`

Examples

```
## Not run:  # internal function
util_filter_names_by_regexps(iris, c("epa", "eta"))

## End(Not run)
```

util_find_external_functions_in_stacktrace

Find externally called function in the stack trace

Description

intended use: error messages for the user

Usage

```
util_find_external_functions_in_stacktrace(
  sfs = rev(sys.frames()),
  cls = rev(sys.calls())
)
```

Arguments

`sfs`	reverse sys.frames to search in
`cls`	reverse sys.calls to search in

Value

vector of logicals stating for each index, if it had been called externally

---

`util_find_first_externally_called_functions_in_stacktrace` *Find first externally called function in the stack trace*

Description

intended use: error messages for the user

Usage

```
util_find_first_externally_called_functions_in_stacktrace(
  sfs = rev(sys.frames()),
  cls = rev(sys.calls())
)
```

Arguments

`sfs`	reverse sys.frames to search in
`cls`	reverse sys.calls to search in

Value

reverse sys.frames index of first non-dataquieR function in this stack

---

`util_find_var_by_meta` *Try hard, to map a variable*

Description

does not warn on ambiguities nor if not found (but in the latter case, it returns `ifnotfound`)

Usage

```
util_find_var_by_meta(
  resp_vars,
  meta_data = "item_level",
  label_col = LABEL,
  allowed_sources = c(VAR_NAMES, label_col, LABEL, LONG_LABEL),
  target = VAR_NAMES,
  ifnotfound = NA_character_
)
```

Arguments

resp_vars	variables to map from
meta_data	metadata
label_col	label-col to map from, if not allowed_sources should be entirely passed
allowed_sources	allowed names to map from (as metadata columns)
target	metadata attribute to map to
ifnotfound	list A list of values to be used if the item is not found: it will be coerced to a list if necessary.

Value

vector of mapped target names of resp_vars

---

util_fix_rstudio_bugs *RStudio crashes on parallel calls in some versions on Darwin based operating systems with R 4*

Description

RStudio crashes on parallel calls in some versions on Darwin based operating systems with R 4

Usage

```
util_fix_rstudio_bugs()
```

Value

invisible null

---

util_float_index_menu *return a single page navigation menu floating on the right*

Description

if displayed in a dq_report2

Usage

```
util_float_index_menu(index_menu_table, object)
```

Arguments

index_menu_table	data.frame columns: links, hovers, texts
object	htmltools tag list, used, instead of index_menu_table, if passed

Examples

```
## Not run:
util_float_index_menu(tibble::tribble(
   ~ links, ~ hovers, ~ texts,
   "http://www.google.de/#xxx", "This is Google", "to Google",
   "http://www.uni-giessen.de/#xxx", "This is Gießen", "cruising on the A45"
))

## End(Not run)
```

---

util_generate_anchor_link

Generate a link to a specific result

Description

for dq_report2

Usage

```
util_generate_anchor_link(
  varname,
  callname,
  order_context = c("variable", "indicator"),
  name,
  title
)
```

Arguments

varname	variable to create a link to
callname	function call to create a link to
order_context	link created to variable overview or indicator overview page
name	replaces varname and callname, must contain the . separator, then
title	optional, replaces auto-generated link title

Value

the htmltools tag

util_generate_anchor_tag

Generate a tag for a specific result

Description

for dq_report2

Usage

```
util_generate_anchor_tag(
  varname,
  callname,
  order_context = c("variable", "indicator"),
  name
)
```

Arguments

varname	variable to create an anchor for
callname	function call to create an anchor for
order_context	anchor created on variable overview or indicator overview page
name	replaces varname and callname, must contain the . separator, then

Value

the htmltools tag

util_generate_calls

Generate an execution/calling plan for computing a report from the metadata

Description

Generate an execution/calling plan for computing a report from the metadata

Usage

```
util_generate_calls(
  dimensions,
  meta_data,
  label_col,
  meta_data_segment,
  meta_data_dataframe,
  meta_data_cross_item,
```

```
  specific_args,
  arg_overrides,
  resp_vars,
  filter_indicator_functions
)
```

Arguments

dimensions	dimensions Vector of dimensions to address in the report. Allowed values in the vector are Completeness, Consistency, and Accuracy. The generated report will only cover the listed data quality dimensions. Accuracy is computational expensive, so this dimension is not enabled by default. Completeness should be included, if Consistency is included, and Consistency should be included, if Accuracy is included to avoid misleading detections of e.g. missing codes as outliers, please refer to the data quality concept for more details. Integrity is always included.
meta_data	data.frame the data frame that contains metadata attributes of study data
label_col	variable attribute the name of the column in the metadata with labels of variables
meta_data_segment	data.frame – optional: Segment level metadata
meta_data_dataframe	data.frame – optional: Data frame level metadata
meta_data_cross_item	data.frame – optional: Cross-item level metadata
specific_args	list named list of arguments specifically for one of the called functions, the of the list elements correspond to the indicator functions whose calls should be modified. The elements are lists of arguments.
arg_overrides	list arguments to be passed to all called indicator functions if applicable.
resp_vars	variables to be respected, NULL means to use all.
filter_indicator_functions	character regular expressions, only if an indicator function's name matches one of these, it'll be used for the report. If of length zero, no filtering is performed.

Value

a list of calls

`util_generate_calls_for_function` *Generate function calls for a given indicator function*

Description

new reporting pipeline v2.0

Usage

```
util_generate_calls_for_function(
  fkt,
  meta_data,
  label_col,
  meta_data_segment,
  meta_data_dataframe,
  meta_data_cross_item,
  specific_args,
  arg_overrides,
  resp_vars
)
```

Arguments

`fkt`	the indicator function's name
`meta_data`	the item level metadata data frame
`label_col`	the label column
`meta_data_segment`	segment level metadata
`meta_data_dataframe`	data frame level metadata
`meta_data_cross_item`	cross-item level metadata
`specific_args`	argument overrides for specific functions
`arg_overrides`	general argument overrides
`resp_vars`	variables to be respected

Value

function calls for the given function

`util_generate_pages_from_report` *Convert a dataquieR report v2 to a named list of web pages*

Description

Convert a dataquieR report v2 to a named list of web pages

Usage

```
util_generate_pages_from_report(
  report,
  template,
  disable_plotly,
  progress = progress,
  progress_msg = progress_msg
)
```

Arguments

report	dataquieR report v2.
template	character template to use, only the name, not the path
disable_plotly	logical do not use plotly, even if installed
progress	function lambda for progress in percent – 1-100
progress_msg	function lambda for progress messages

Value

named list, each entry becomes a file with the name of the entry. the contents are HTML objects as used by htmltools.

Examples

```
## Not run:
devtools::load_all()
prep_load_workbook_like_file("meta_data_v2")
report <- dq_report2("study_data", dimensions = NULL, label_col = "LABEL");
save(report, file = "report_v2.RData")
report <- dq_report2("study_data", label_col = "LABEL");
save(report, file = "report_v2_short.RData")

## End(Not run)
```

---

util_get_code_list *Fetch a missing code list from the metadata*

Description

get missing codes from metadata (e.g. MISSING_LIST or JUMP_LIST)

Usage

```
util_get_code_list(
  x,
  code_name,
  split_char = SPLIT_CHAR,
  mdf,
  label_col = VAR_NAMES,
  warning_if_no_list = TRUE
)
```

Arguments

x	variable the name of the variable to retrieve code lists for. only one variable at a time is supported, *not* vectorized!!
code_name	variable attribute JUMP_LIST or MISSING_LIST: Which codes to retrieve.
split_char	character len = 1. Character(s) used to separate different codes in the metadata, usually |, as in 99999|99998|99997.
mdf	data.frame the data frame that contains metadata attributes of study data
label_col	variable attribute the name of the column in the metadata with labels of variables
warning_if_no_list	logical len = 1. If TRUE, a warning is displayed, if not missing codes are available for a variable.

Value

numeric vector of missing codes.

---

util_get_color_for_result

Return the color for a result

Description

messages do not cause any coloring, warnings are yellow, errors are red

Usage

```
util_get_color_for_result(
  result,
  aspect = c("applicability", "error", "issue"),
  ...
)
```

Arguments

result	a dataquieR_resultset2 result
aspect	an aspect/problem category of results (error, applicability error or data quality issue)
...	not used

Value

a color as a grading

---

util_get_concept_info *Read additional concept tables*

Description

Read additional concept tables

Usage

```
util_get_concept_info(filename, ...)
```

Arguments

filename	RDS-file name without extension to read from
...	passed to subset

Value

a data frame

---

util_get_html_cell_for_result *Return the html summary table cell for a result*

Description

Return the html summary table cell for a result

Usage

```
util_get_html_cell_for_result(
  result,
  aspect = c("applicability", "error", "issue"),
  rn,
  cn,
  ...
)
```

Arguments

result	a dataquieR_resultset2 result
aspect	an aspect/problem category of results
rn	row name for links inside a dq_report2 report
cn	column name for links inside a dq_report2 report
...	not used

Value

character(1) html result for a DT cell

---

util_get_message_for_result

Return messages/warnings/notes/error messages for a result

Description

Return messages/warnings/notes/error messages for a result

Usage

```
util_get_message_for_result(
  result,
  aspect = c("applicability", "error", "issue"),
  collapse = "\n<br />\n",
  ...
)
```

Arguments

result	a dataquieR_resultset2 result
aspect	an aspect/problem category of results
collapse	either a lambda function or a separator for combining multiple messages for the same result
...	not used

Value

hover texts for results with data quality issues, run-time errors, warnings or notes (aka messages)

`util_get_redcap_rule_env`
an environment with functions available for REDcap *rules*

Description

an environment with functions available for REDcap rules

Usage

```
util_get_redcap_rule_env()
```

Value

environment

`util_get_vars_in_segment`
Return all variables in the segment segment

Description

Return all variables in the segment segment

Usage

```
util_get_vars_in_segment(segment, meta_data = "item_level", label_col = LABEL)
```

Arguments

segment	character the segment as specified in STUDY_SEGMENT
meta_data	data.frame the metadata
label_col	character the meta data attribute used for naming the variables

Value

vector of variable names

util_get_var_att_names_of_level *Get variable attributes of a certain provision level*

Description

This function returns all variable attribute names of a certain meta data provision level or of more than one level.

Usage

```
util_get_var_att_names_of_level(level, cumulative = TRUE)
```

Arguments

level	level(s) of requirement
cumulative	include all names from more basic levels

Value

all matching variable attribute names

util_gg_var_label *Add labels to* ggplot

Description

EXPERIMENTAL

Usage

```
util_gg_var_label(
  ...,
  meta_data = get("meta_data", parent.frame()),
  label_col = get("label_col", parent.frame())
)
```

Arguments

...	EXPERIMENTAL
meta_data	the metadata
label_col	the label columns

Value

a modified ggplot

Examples

```
## Not run:
load(system.file("extdata", "study_data.RData", package = "dataquieR"))
load(system.file("extdata", "meta_data.RData", package = "dataquieR"))
ggplot(study_data, aes(x = v00013, y = v00004)) + geom_point() +
  util_gg_var_label()
m <- acc_margins(study_data = study_data, meta_data = meta_data,
  resp_vars = "v00004",
  group_vars = "v00012", label_col = VAR_NAMES)
m$SummaryPlot + util_gg_var_label()
l <- acc_loess(study_data = study_data, meta_data = meta_data,
  time_vars = "v00013",
  resp_vars = "v00004",
  group_vars = "v00012", label_col = VAR_NAMES)
l$SummaryPlotList$v00004 + util_gg_var_label()

d <- acc_distributions("v00004", study_data = study_data,
  meta_data = meta_data,
  label_col = VAR_NAMES)

d$SummaryPlotList$v00004 + util_gg_var_label()

## End(Not run)
```

---

util_heatmap_1th *Utility Function Heatmap with 1 Threshold*

Description

Function to create heatmap-like plot given one threshold – works for percentages for now.

Usage

```
util_heatmap_1th(
  df,
  cat_vars,
  values,
  threshold,
  right_intv,
  invert,
  cols,
  strata
)
```

Arguments

df	data.frame with data to display as a heatmap.
cat_vars	variable list len=1-2. Variables to group by. Up to 2 group levels supported.

values	variable the name of the percentage variable
threshold	numeric lowest acceptable value
right_intv	logical len=1. If FALSE (default), intervals used to define color ranges in the heatmap are closed on the left side, if TRUE on the right side, respectively.
invert	logical len=1. If TRUE, high values are better, warning colors are used for low values. FALSE works vice versa.
cols	deprecated, ignored.
strata	variable optional, the name of a variable used for stratification inheritParams acc_distributions

Value

a list with:

- SummaryPlot: ggplot object with the heatmap

---

util_html_attr_quote_escape *escape "*

Description

escape "

Usage

```
util_html_attr_quote_escape(s)
```

Arguments

s	haystack

Value

s with " replaced by "

`util_html_table` *The jack of all trades device for tables*

Description

The jack of all trades device for tables

Usage

```
util_html_table(
  tb,
  filter = "top",
  columnDefs = NULL,
  autoWidth = FALSE,
  hideCols = character(0),
  rowCallback = DT::JS("function(r,d) {$(r).attr('height', '2em')}"),
 copy_row_names_to_column = length(rownames(tb)) == nrow(tb) && !is.integer(attr(tb,
    "row.names")) && !all(seq_len(nrow(tb)) == rownames(tb)),
  link_variables = TRUE,
  tb_rownames = FALSE,
  meta_data,
  rotate_headers = FALSE,
  fillContainer = TRUE,
  ...,
  colnames,
  options = list(),
  is_matrix_table = FALSE,
  colnames_aliases2acronyms = is_matrix_table,
  label_col = LABEL,
  output_format = c("RMD", "HTML"),
  dl_fn = "*"
)
```

Arguments

`tb`	the table as data.frame
`filter`	passed to `DT::datatable`
`columnDefs`	column specifications for the `datatables` JavaScript object
`autoWidth`	passed to the `datatables` JavaScript library
`hideCols`	columns to hide (by name)
`rowCallback`	passed to the `datatables` JavaScript library (with default)
`copy_row_names_to_column`	add a column 0 with `rownames`
`link_variables`	considering row names being variables, convert row names to links to the variable specific reports

tb_rownames	number of columns from the left considered as row-names
meta_data	the data dictionary for labels and similar stuff
rotate_headers	rotate headers by 90 degrees
fillContainer	see DT::datatable
...	passed to DT::datatable
colnames	column names for the table (defaults to colnames(tb))
options	individually overwrites defaults in options passed to DT::datatable
is_matrix_table	create a heat map like table without padding
colnames_aliases2acronyms	abbreviate column names considering being analysis matrix columns by their acronyms defined in square.
label_col	label col used for mapping labels in case of link_variables is used (that argument set to TRUE and Variables or VAR_NAMES in meta_data)
output_format	target format RMD or HTML, for RMD, markdown will be used in the output, for HTML, only HTML code is being generated
dl_fn	file name for downloaded table – see https://datatables.net/reference/button/excel)

Value

the table to be added to an rmd/´html file as htmlwidgets::htmlwidgets

---

`util_hubert` *utility function for the outliers rule of Huber et al.*

Description

function to calculate outliers according to the rule of Huber et al. This function requires the package robustbase

Usage

```
util_hubert(x)
```

Arguments

x	numeric data to check for outliers

Value

binary vector

util_interpret_limits *Utility function to interpret mathematical interval notation*

Description

Utility function to split limit definitions into interpretable elements

Usage

```
util_interpret_limits(mdata)
```

Arguments

mdata	data.frame the data frame that contains metadata attributes of study data

Value

augments metadata by interpretable limit columns

util_interpret_range *Utility function to interpret mathematical interval notation for numeric ranges*

Description

Utility function to split range definitions into interpretable elements

Usage

```
util_interpret_range(mdata)
```

Arguments

mdata	data.frame the data frame that contains metadata attributes of study data

Value

augments metadata by interpretable limit columns

util_int_duplicate_content_dataframe

Check for duplicated content

Description

This function tests for duplicates entries in the data set. It is possible to check duplicated entries by study segments or to consider only selected segments.

Usage

```
util_int_duplicate_content_dataframe(
  level = c("dataframe"),
  identifier_name_list
)
```

Arguments

`level`	character a character vector indicating whether the assessment should be conducted at the study level (level = "dataframe") or at the segment level (level = "segment").
`identifier_name_list`	vector the vector that contains the name of the identifier to be used in the assessment. For the study level, corresponds to the names of the different data frames. For the segment level, indicates the name of the segments.

Value

a list with

- `SegmentData`: data frame with the results of the quality check for duplicated entries
- `SegmentTable`: data frame with selected duplicated entries check results, used for the data quality report.
- `Duplicates`: vector with row indices of duplicated entries, if any, otherwise NULL.

util_int_duplicate_content_segment

Check for duplicated content

Description

This function tests for duplicates entries in the data set. It is possible to check duplicated entries by study segments or to consider only selected segments.

Usage

```
util_int_duplicate_content_segment(
  level = c("segment"),
  study_segment,
  study_data,
  meta_data
)
```

Arguments

level	character a character vector indicating whether the assessment should be conducted at the study level (level = "dataframe") or at the segment level (level = "segment").
study_segment	vector the vector that contains the name of the identifier to be used in the assessment. For the study level, corresponds to the names of the different data frames. For the segment level, indicates the name of the segments.
study_data	data.frame the data frame that contains the measurements, mandatory.
meta_data	data.frame the data frame that contains metadata attributes of the study data, mandatory.

Value

a list with

- SegmentData: data frame with the results of the quality check for duplicated entries
- SegmentTable: data frame with selected duplicated entries check results, used for the data quality report.
- Duplicates: vector with row indices of duplicated entries, if any, otherwise NULL.

Examples

```
## Not run:
study_data <- readRDS(system.file("extdata", "ship.RDS", package = "dataquieR"))
meta_data <- readRDS(system.file("extdata", "ship_meta.RDS", package = "dataquieR"))

# Segment level
int_duplicate_content(
  level = "segment",
  study_segment = c("INTRO", "INTERVIEW"),
  study_data = study_data,
  meta_data = meta_data
)

# Studies or data frame level
study_tables <- list(
  "sd1" = readRDS(system.file("extdata", "ship.RDS", package = "dataquieR")),
  "sd2" = readRDS(system.file("extdata", "ship.RDS", package = "dataquieR"))
)
```

```
int_duplicate_content(
  level = "dataframe",
  study_segment = c("sd1", "sd2"),
  study_data = study_tables,
  meta_data = meta_data
)

## End(Not run)
```

`util_int_duplicate_ids_dataframe` *Check for duplicated IDs*

Description

This function tests for duplicates entries in identifiers. It is possible to check duplicated identifiers by study segments or to consider only selected segments.

Usage

```
util_int_duplicate_ids_dataframe(
  level = c("dataframe"),
  id_vars_list,
  identifier_name_list,
  repetitions,
  meta_data = NULL
)
```

Arguments

`level`	character a character vector indicating whether the assessment should be conducted at the study level (level = "dataframe") or at the segment level (level = "segment").
`id_vars_list`	list id variable names for each segment or data frame
`identifier_name_list`	vector the segments or data frame names being assessed
`repetitions`	vector an integer vector indicating the number of allowed repetitions in the id_vars. Currently, no repetitions are supported. # TODO
`meta_data`	data.frame the data frame that contains metadata attributes of the study data, mandatory.

Value

a list with

- `DataframeData`: data frame with the results of the quality check for duplicated identifiers

- DataframeTable: data frame with selected duplicated identifiers check results, used for the data quality report.
- Duplicates: vector with row indices of duplicated identifiers, if any, otherwise NULL.

`util_int_duplicate_ids_segment` *Check for duplicated IDs*

Description

This function tests for duplicates entries in identifiers. It is possible to check duplicated identifiers by study segments or to consider only selected segments.

Usage

```
util_int_duplicate_ids_segment(
  level = c("segment"),
  id_vars_list,
  study_segment,
  repetitions,
  study_data,
  meta_data
)
```

Arguments

level	character a character vector indicating whether the assessment should be conducted at the study level (level = "dataframe") or at the segment level (level = "segment").
id_vars_list	list id variable names for each segment or data frame
study_segment	vector the segments or data frame names being assessed
repetitions	vector an integer vector indicating the number of allowed repetitions in the id_vars. Currently, no repetitions are supported. # TODO
study_data	data.frame the data frame that contains the measurements, mandatory.
meta_data	data.frame the data frame that contains metadata attributes of the study data, mandatory.

Value

a list with

- SegmentData: data frame with the results of the quality check for duplicated identifiers
- SegmentTable: data frame with selected duplicated identifiers check results, used for the data quality report.
- Duplicates: vector with row indices of duplicated identifiers, if any, otherwise NULL.

`util_int_unexp_records_set_dataframe` *Check for unexpected data record set*

Description

This function tests that the identifiers match a provided record set. It is possible to check for unexpected data record sets by study segments or to consider only selected segments.

Usage

```
util_int_unexp_records_set_dataframe(
  level = c("dataframe"),
  id_vars_list,
  identifier_name_list,
  valid_id_table_list,
  meta_data_record_check_list
)
```

Arguments

`level`	character a character vector indicating whether the assessment should be conducted at the study level (level = "dataframe") or at the segment level (level = "segment").
`id_vars_list`	list the list containing the identifier variables names to be used in the assessment.
`identifier_name_list`	list the list that contains the name of the identifier to be used in the assessment. For the study level, corresponds to the names of the different data frames. For the segment level, indicates the name of the segments.
`valid_id_table_list`	list the reference list with the identifier variable values.
`meta_data_record_check_list`	character a character vector indicating the type of check to conduct, either "subset" or "exact".

Value

a list with

- `SegmentData`: data frame with the results of the quality check for unexpected data elements
- `SegmentTable`: data frame with selected unexpected data elements check results, used for the data quality report.
- `UnexpectedRecords`: vector with row indices of duplicated records, if any, otherwise NULL.

`util_int_unexp_records_set_segment` *Check for unexpected data record set*

Description

This function tests that the identifiers match a provided record set. It is possible to check for unexpected data record sets by study segments or to consider only selected segments.

Usage

```
util_int_unexp_records_set_segment(
  level = c("segment"),
  id_vars_list,
  identifier_name_list,
  valid_id_table_list,
  meta_data_record_check_list,
  study_data,
  meta_data
)
```

Arguments

`level`	character a character vector indicating whether the assessment should be conducted at the study level (level = "dataframe") or at the segment level (level = "segment").
`id_vars_list`	list the list containing the identifier variables names to be used in the assessment.
`identifier_name_list`	list the list that contains the name of the identifier to be used in the assessment. For the study level, corresponds to the names of the different data frames. For the segment level, indicates the name of the segments.
`valid_id_table_list`	list the reference list with the identifier variable values.
`meta_data_record_check_list`	character a character vector indicating the type of check to conduct, either "subset" or "exact".
`study_data`	data.frame the data frame that contains the measurements, mandatory.
`meta_data`	data.frame the data frame that contains metadata attributes of the study data, mandatory.

Value

a list with

- `SegmentData`: data frame with the results of the quality check for unexpected data elements

- SegmentTable: data frame with selected unexpected data elements check results, used for the data quality report.
- UnexpectedRecords: vector with row indices of duplicated records, if any, otherwise NULL.

Examples

```
## Not run:
study_data <- readRDS(system.file("extdata", "ship.RDS",
  package = "dataquieR"
))
meta_data <- readRDS(system.file("extdata", "ship_meta.RDS",
  package = "dataquieR"
))
md1_segment <- readRDS(system.file("extdata", "meta_data_segment.RDS",
  package = "dataquieR"
))
ids_segment <- readRDS(system.file("extdata", "meta_data_ids_segment.RDS",
  package = "dataquieR"
))

# TODO: update examples
int_unexp_records_set(
  level = "segment",
  identifier_name_list = c("INTERVIEW", "LABORATORY"),
  valid_id_table_list = ids_segment,
  meta_data_record_check = md1_segment[,
    c("STUDY_SEGMENT", "SEGMENT_RECORD_CHECK")],
  study_data = study_data,
  meta_data = meta_data
)

## End(Not run)
```

---

util_is_integer *Check for integer values*

Description

This function checks if a variable is integer.

Usage

```
util_is_integer(x, tol = .Machine$double.eps^0.5)
```

Arguments

x	the object to test
tol	precision of the detection. Values deviating more than tol from their closest integer value will not be deemed integer.

Value

TRUE or FALSE

See Also

is.integer

Copied from the documentation of is.integer

is.integer detects, if the storage mode of an R-object is integer. Usually, users want to know, if the values are integer. As suggested by is.integer's documentation, `is.wholenumber` does so.

---

util_is_na_0_empty_or_false *Detect falsish values*

Description

Detect falsish values

Usage

```
util_is_na_0_empty_or_false(x)
```

Arguments

x	a value/vector of values

Value

vector of logical values: TRUE, wherever x is somehow empty

---

util_is_numeric_in *Create a predicate function to check for certain numeric properties*

Description

useful, e.g., for util_expect_data_frame and util_expect_scalar.

Usage

```
util_is_numeric_in(min = -Inf, max = +Inf, whole_num = FALSE, set = NULL)
```

Arguments

min	if given, minimum for numeric values
max	if given, maximum for numeric values
whole_num	if TRUE, expect a whole number
set	if given, a set, the value must be in (see util_match_arg)

Value

a function that checks an x for the properties.

Examples

```
## Not run:
util_is_numeric_in(min = 0)(42)
util_is_numeric_in(min = 43)(42)
util_is_numeric_in(max = 3)(42)
util_is_numeric_in(whole_num = TRUE)(42)
util_is_numeric_in(whole_num = TRUE)(42.1)
util_is_numeric_in(set = c(1, 3, 5))(1)
util_is_numeric_in(set = c(1, 3, 5))(2)

## End(Not run)
```

---

util_load_manual *being called by the active binding function for .manual*

Description

being called by the active binding function for .manual

Usage

```
util_load_manual()
```

---

util_looks_like_missing *Check for repetitive values using the digits 8 or 9 only*

Description

Values not being finite (see is.finite) are also reported as missing codes.

Usage

```
util_looks_like_missing(x, n_rules = 1)
```

Arguments

x	numeric vector to test
n_rules	numeric Only outlying values can be missing codes; at least n_rules rules in acc_univariate_outlier match

Value

logical indicates for each value in x, if it looks like a missing code

See Also

acc_univariate_outlier

---

util_make_data_slot_from_table_slot

Rename columns of a SummaryTable *(or Segment, ...) to look nice*

Description

Rename columns of a SummaryTable (or Segment, ...) to look nice

Usage

```
util_make_data_slot_from_table_slot(Table)
```

Arguments

Table	data.frame, a table

Value

renamed table

---

util_make_function

Make a function capturing errors and other conditions for parallelization

---

Description

Make a function capturing errors and other conditions for parallelization

Usage

```
util_make_function(fct, caller.)
```

Arguments

fct	function to prepare
caller.	call for error messages, default is the caller of util_make_function

Value

decorated function

---

util_map_all *Maps label column meta data on study data variable names*

Description

Maps a certain label column from the meta data to the study data frame.

Usage

```
util_map_all(label_col = VAR_NAMES, study_data, meta_data)
```

Arguments

label_col	the variable of the metadata that contains the variable names of the study data
study_data	the name of the data frame that contains the measurements
meta_data	the name of the data frame that contains metadata attributes of study data

Value

list with slot df with a study data frame with mapped column names

---

util_map_by_largest_prefix *Map based on largest common prefix*

Description

Map based on largest common prefix

Usage

```
util_map_by_largest_prefix(needle, haystack, split_char = "_")
```

Arguments

needle	character(1) item to search
haystack	character items to find the entry sharing the largest prefix with needle
split_char	character(1) to split entries to atomic words (like letters, if "" or snake_elements, if "_")

Value

character(1) with the fitting function name or NA_character_

Examples

```
## Not run:  # internal function
util_map_by_largest_prefix(
  "acc_distributions_loc_ecdf_observer_time",
  names(dataquieR:::.manual$titles)
)
util_map_by_largest_prefix(
  "acc_distributions_loc_observer_time",
  names(dataquieR:::.manual$titles)
)
util_map_by_largest_prefix(
  "acc_distributions_loc_ecdf",
  names(dataquieR:::.manual$titles)
)
util_map_by_largest_prefix(
  "acc_distributions_loc",
  names(dataquieR:::.manual$titles)
)

## End(Not run)
```

---

util_map_labels *Support function to allocate labels to variables*

Description

Map variables to certain attributes, e.g. by default their labels.

Usage

```
util_map_labels(
  x,
  meta_data = "item_level",
  to = LABEL,
  from = VAR_NAMES,
  ifnotfound,
  warn_ambiguous = FALSE
)
```

Arguments

x	character variable names, character vector, see parameter from
meta_data	data.frame meta data frame, if, as a dataquieR developer, you do not have **item-level-metadata**, you should use util_map_labels instead to avoid consistency checks on for item-level meta_data.
to	character variable attribute to map to
from	character variable identifier to map from
ifnotfound	list A list of values to be used if the item is not found: it will be coerced to a list if necessary.
warn_ambiguous	logical print a warning if mapping variables from from to to produces ambiguous identifiers.

Details

This function basically calls colnames(study_data) <- meta_data$LABEL, ensuring correct merging/joining of study data columns to the corresponding meta data rows, even if the orders differ. If a variable/study_data-column name is not found in meta_data[[from]] (default from = VAR_NAMES), either stop is called or, if ifnotfound has been assigned a value, that value is returned. See mget, which is internally used by this function.

The function not only maps to the LABEL column, but to can be any metadata variable attribute, so the function can also be used, to get, e.g. all HARD_LIMITS from the metadata.

Value

a character vector with:

- mapped values

Examples

```
## Not run:
meta_data <- prep_create_meta(
  VAR_NAMES = c("ID", "SEX", "AGE", "DOE"),
  LABEL = c("Pseudo-ID", "Gender", "Age", "Examination Date"),
  DATA_TYPE = c(DATA_TYPES$INTEGER, DATA_TYPES$INTEGER, DATA_TYPES$INTEGER,
                DATA_TYPES$DATETIME),
  MISSING_LIST = ""
)
stopifnot(all(prep_map_labels(c("AGE", "DOE"), meta_data) == c("Age",
                                                    "Examination Date")))

## End(Not run)
```

util_match_arg dataquieR *version of match.arg*

Description

does not support partial matching, but will display the most likely match as a warning/error.

Usage

```
util_match_arg(arg, choices, several_ok = FALSE, error = TRUE)
```

Arguments

arg	the argument
choices	the choices
several_ok	allow more than one entry in arg
error	stop(), if arg is not in choices (warns and cleans arg, otherwise)

Value

"cleaned" arg

util_merge_data_frame_list *Combine data frames by merging*

Description

This is an extension of merge working for a list of data frames.

Usage

```
util_merge_data_frame_list(data_frames, id_vars)
```

Arguments

data_frames	list of data.frames
id_vars	character the variable(s) to merge the data frames by. each of them must exist in all data frames.

Value

data.frame combination of data frames

util_message *Produce a condition message with a useful short stack trace.*

Description

Produce a condition message with a useful short stack trace.

Usage

```
util_message(
  m,
  ...,
  applicability_problem = NA,
  integrity_indicator = "none",
  level = 0
)
```

Arguments

m	a message or a condition
...	arguments for sprintf on m, if m is a character
applicability_problem	logical message indicates unsuitable resp_vars
integrity_indicator	character the message is an integrity problem, here is the indicator abbreviation..
level	integer level of the message (defaults to 0). Higher levels are more severe.

Value

condition the condition object, if the execution is not stopped

util_no_value_labels *Select really numeric variables*

Description

Reduce `resp_vars` to those, which are either `float` or `integer` without VALUE_LABELS, i.e. likely numeric but not a factor

Usage

```
util_no_value_labels(resp_vars, meta_data, label_col, warn = TRUE, stop = TRUE)
```

Arguments

resp_vars	variable list len=1-2. the name of the continuous measurement variable
meta_data	data.frame the data frame that contains metadata attributes of study data
label_col	variable attribute the name of the column in the metadata with labels of variables
warn	logical warn about removed variable names
stop	logical stop on no matching resp_var

Value

character vector of matching resp_vars.

---

util_observations_in_subgroups

Utility function observations in subgroups

Description

This function uses !is.na to count the number of non-missing observations in subgroups of the data (list) and in a set of user defined response variables. In some applications it is required that the number of observations per e.g. factor level is higher than a user-defined minimum number.

Usage

```
util_observations_in_subgroups(x, rvs)
```

Arguments

x	data frame
rvs	variable names

Value

matrix of flags

util_observation_expected *Detect Expected Observations*

Description

For each participant, check, if an observation was expected, given the PART_VARS from item-level metadata

Usage

```
util_observation_expected(
  rv,
  study_data,
  meta_data,
  label_col = LABEL,
  expected_observations = c("HIERARCHY", "ALL", "SEGMENT")
)
```

Arguments

rv	character the response variable, for that a value may be expected
study_data	study_data
meta_data	meta_data
label_col	character mapping attribute colnames(study_data) vs. meta_data[label_col]
expected_observations	enum HIERARCHY | ALL | SEGMENT. How should PART_VARS be handled: - ALL: Ignore, all observations are expected - SEGMENT: if PART_VAR is 1, an observation is expected - HIERARCHY: the default, if the PART_VAR is 1 for this variable and also for all PART_VARS of PART_VARS up in the hierarchy, an observation is expected.

Value

a vector with TRUE or FALSE for each row of study_data, if for study_data[rv] a value is expected.

util_only_NAs *identify* NA*-only variables*

Description

This utility function identifies variables with NAs values only.

Usage

```
util_only_NAs(x)
```

Arguments

x	the variable to check a vector

Value

flagged binary vector

util_optimize_histogram_bins *Utility function to compute and optimize bin breaks for histograms*

Description

Utility function to compute and optimize bin breaks for histograms

Usage

```
util_optimize_histogram_bins(
  x,
  iqr_bw,
  n_bw,
  min_within = NULL,
  max_within = NULL,
  min_plot = NULL,
  max_plot = NULL,
  nbins_max = NULL
)
```

Arguments

x	a vector of data values (numeric or datetime)
iqr_bw	the interquartile range of values which should be included to calculate the Freedman-Diaconis bandwidth (e.g., for con_limit_deviations only values within limits)
n_bw	the number of values which should be included to calculate the Freedman-Diaconis bandwidth (e.g., for con_limit_deviations the number of values within limits)
min_within	the minimum value which is still within limits (needed for con_limit_deviations)
max_within	the maximum value which is still within limits (needed for con_limit_deviations)
min_plot	the minimum value which should be included in the plot
max_plot	the maximum value which should be included in the plot
nbins_max	the maximum number of bins for the histogram. Strong outliers can cause too many narrow bins, which might be even to narrow to be plotted. This also results in large files and rendering problems. So it is sensible to limit the number of bins. The function will produce a warning if it reduces the number of bins in such a case. Reasons could be unspecified missing value codes, or minimum or maximum values far away from most of the data values, or (for con_limit_deviations) no or few values within limits.

Value

a list with bin breaks below, within and above limits

---

util_order_by_order *Get the order of a vector with general order given in some other vector*

Description

Get the order of a vector with general order given in some other vector

Usage

```
util_order_by_order(x, order, ...)
```

Arguments

x	the vector
order	the "order vector
...	additional arguments passed to order

Examples

```
## Not run:
util_order_by_order(c("a", "b", "a", "c", "d"), letters)

## End(Not run)
```

util_parse_assignments *Utility function to parse assignments*

Description

This function parses labels & level assignments in the format `1 = male | 2 = female`. The function also handles `m = male | f = female`, but this would not match the metadata concept. The split-character can be given, if not the default from SPLIT_CHAR is to be used, but this would also violate the metadata concept.

Usage

```
util_parse_assignments(
  text,
  split_char = SPLIT_CHAR,
  multi_variate_text = FALSE
)
```

Arguments

`text`	Text to be parsed
`split_char`	Character separating assignments
`multi_variate_text`	don't paste text but parse element-wise

Value

the parsed assignments as a named list

util_parse_interval *Utility function to parse intervals*

Description

Utility function to parse intervals

Usage

```
util_parse_interval(int)
```

Arguments

`int`	an interval as string, e.g., "[0;Inf)"

Value

the parsed interval with elements `inc_l` (Is the lower limit included?), `low` (the value of the lower limit), `inc_u` (Is the upper limit included?), `upp` (the value of the upper limit)

`util_parse_redcap_rule`

Interpret a `REDcap`*-style rule and create an expression, that represents this rule*

Description

Interpret a `REDcap`-style rule and create an expression, that represents this rule

Usage

```
util_parse_redcap_rule(
  rule,
  debug = 0,
  entry_pred = "REDcapPred",
  must_eof = FALSE
)
```

Arguments

`rule`	character `REDcap` style rule
`debug`	integer debug level (0 = off, 1 = log, 2 = breakpoints)
`entry_pred`	character for debugging reasons: The production rule used entry point for the parser
`must_eof`	logical if `TRUE`, expect the input to be `eof`, when the parser succeeded, fail, if not.

Value

expression the interpreted rule

`REDcap` rules 1 `REDcap` rules 2 `REDcap` rules 3

For resolving left-recursive rules, StackOverflow helps understanding the grammar below, just in case, theoretical computer science is not right in your mind currently.

Examples

```
## Not run:
#  rules:
# pregnancies <- 9999 ~ SEX == 'm' |  is.na(SEX)
# pregnancies <- 9998 ~ AGE < 12 |  is.na(AGE)
# pregnancies = 9999 ~ dist > 2 |  speed == 0
```

```
data.frame(target = "SEX_0",
  rule = '[speed] > 5 and [dist] > 42 or 1 = "2"',
  CODE = 99999, LABEL = "PREGNANCIES_NOT_ASSESSED FOR MALES",
  class = "JUMP")
ModifyiedStudyData <- replace in SEX_0 where SEX_0 is empty, if rule fits
ModifyedMetaData <- add missing codes with labels and class here

subset(study_data, eval(pregnancies[[3]]))

rule <-
 paste0('[con_consentdt] <> "" and [sda_osd1dt] <> "" and',
 ' datediff([con_consentdt],[sda_osd1dt],"d",true) < 0')

x <- data.frame(con_consentdt = c(as.POSIXct("2020-01-01"),
                as.POSIXct("2020-10-20")),
                sda_osd1dt = c(as.POSIXct("2020-01-20"),
                as.POSIXct("2020-10-01")))
eval(util_parse_redcap_rule(paste0(
  '[con_consentdt] <> "" and [sda_osd1dt] <> "" and ',
  'datediff([con_consentdt],[sda_osd1dt],"d", "Y-M-D",true) < 10')),
  x, util_get_redcap_rule_env())

util_parse_redcap_rule("[a] = 12 or [b] = 13")
cars[eval(util_parse_redcap_rule(
  rule = '[speed] > 5 and [dist] > 42 or 1 = "2"'), cars,
  util_get_redcap_rule_env()), ]
cars[eval(util_parse_redcap_rule(
  rule = '[speed] > 5 and [dist] > 42 or 2 = "2"'), cars,
  util_get_redcap_rule_env()), ]
cars[eval(util_parse_redcap_rule(
  rule = '[speed] > 5 or [dist] > 42 and 1 = "2"'), cars,
  util_get_redcap_rule_env()), ]
cars[eval(util_parse_redcap_rule(
  rule = '[speed] > 5 or [dist] > 42 and 2 = "2"'), cars,
  util_get_redcap_rule_env()), ]
util_parse_redcap_rule(rule = '(1 = "2" or true) and (false)')
eval(util_parse_redcap_rule(rule =
  '[dist] > sum(1, +(2, [dist] + 5), [speed]) + 3 + [dist]'),
cars, util_get_redcap_rule_env())

## End(Not run)
```

`util_par_pmap` *Utility function parallel version of* `purrr::pmap`

Description

Parallel version of `purrr::pmap`.

Usage

```
util_par_pmap(
  .l,
  .f,
  ...,
  cores = list(mode = "socket", cpus = util_detect_cores(), logging = FALSE,
    load.balancing = TRUE),
  use_cache = FALSE
)
```

Arguments

.l	data.frame with one call per line and one function argument per column
.f	function to call with the arguments from .l
...	additional, static arguments for calling .f
cores	number of cpu cores to use or a (named) list with arguments for parallelMap::parallelStart or NULL, if parallel has already been started by the caller.
use_cache	logical set to FALSE to omit re-using already distributed study- and metadata on a parallel cluster

Value

list of results of the function calls

Author(s)

Aurèle

S Struckmann

See Also

purrr::pmap

Stack Overflow post

---

util_prepare_dataframes *util_prepare_dataframes*

Description

This function ensures, that a data frame ds1 with suitable variable names study_data and meta_data exist as base data.frames.

Usage

```
util_prepare_dataframes(
  .study_data,
  .meta_data,
  .label_col,
  .replace_hard_limits,
  .replace_missings,
  .sm_code = NULL,
  .allow_empty = FALSE
)
```

Arguments

.study_data	if provided, use this data set as study_data
.meta_data	if provided, use this data set as meta_data
.label_col	if provided, use this as label_col
.replace_hard_limits	replace HARD_LIMIT violations by NA, defaults to FALSE.
.replace_missings	replace missing codes, defaults to TRUE
.sm_code	missing code for NAs, if they have been re-coded by util_combine_missing_lists
.allow_empty	allow ds1 to be empty. i.e., 0 rows and/or 0 columns

Details

This function defines ds1 and modifies study_data and meta_data in the environment of its caller (see eval.parent). It also defines or modifies the object label_col in the calling environment. Almost all functions exported by dataquieR call this function initially, so that aspects common to all functions live here, e.g. testing, if an argument meta_data has been given and features really a data.frame. It verifies the existence of required metadata attributes (VARATT_REQUIRE_LEVELS). It can also replace missing codes by NAs, and calls prep_study2meta to generate a minimum set of metadata from the study data on the fly (should be amended, so on-the-fly-calling is not recommended for an instructive use of dataquieR).

The function also detects tibbles, which are then converted to base-R data.frames, which are expected by dataquieR.

Different from the other utility function that work in the caller's environment, so it modifies objects in the calling function. It defines a new object ds1, it modifies study_data and/or meta_data and label_col.

Value

ds1 the study data with mapped column names

See Also

acc_margins

Examples

```
## Not run:
acc_test1 <- function(resp_variable, aux_variable,
                      time_variable, co_variables,
                      group_vars, study_data, meta_data) {
  prep_prepare_dataframes()
  invisible(ds1)
}
acc_test2 <- function(resp_variable, aux_variable,
                      time_variable, co_variables,
                      group_vars, study_data, meta_data, label_col) {
  ds1 <- prep_prepare_dataframes(study_data, meta_data)
  invisible(ds1)
}
environment(acc_test1) <- asNamespace("dataquieR")
# perform this inside the package (not needed for functions that have been
# integrated with the package already)

environment(acc_test2) <- asNamespace("dataquieR")
# perform this inside the package (not needed for functions that have been
# integrated with the package already)
acc_test3 <- function(resp_variable, aux_variable, time_variable,
                      co_variables, group_vars, study_data, meta_data,
                      label_col) {
  prep_prepare_dataframes()
  invisible(ds1)
}
acc_test4 <- function(resp_variable, aux_variable, time_variable,
                      co_variables, group_vars, study_data, meta_data,
                      label_col) {
  ds1 <- prep_prepare_dataframes(study_data, meta_data)
  invisible(ds1)
}
environment(acc_test3) <- asNamespace("dataquieR")
# perform this inside the package (not needed for functions that have been
# integrated with the package already)

environment(acc_test4) <- asNamespace("dataquieR")
# perform this inside the package (not needed for functions that have been
# integrated with the package already)
load(system.file("extdata/meta_data.RData", package = "dataquieR"))
load(system.file("extdata/study_data.RData", package = "dataquieR"))
try(acc_test1())
try(acc_test2())
acc_test1(study_data = study_data)
try(acc_test1(meta_data = meta_data))
try(acc_test2(study_data = 12, meta_data = meta_data))
print(head(acc_test1(study_data = study_data, meta_data = meta_data)))
print(head(acc_test2(study_data = study_data, meta_data = meta_data)))
print(head(acc_test3(study_data = study_data, meta_data = meta_data)))
print(head(acc_test3(study_data = study_data, meta_data = meta_data,
  label_col = LABEL)))
```

```
print(head(acc_test4(study_data = study_data, meta_data = meta_data)))
print(head(acc_test4(study_data = study_data, meta_data = meta_data,
  label_col = LABEL)))
try(acc_test2(study_data = NULL, meta_data = meta_data))

## End(Not run)
```

`util_prep_location_check`

Utility function to prepare the metadata for location checks

Description

Utility function to prepare the metadata for location checks

Usage

```
util_prep_location_check(
  resp_vars,
  meta_data,
  report_problems = c("error", "warning", "message")
)
```

Arguments

`resp_vars`	variable list the names of the measurement variables
`meta_data`	data.frame the data frame that contains metadata attributes of study data
`report_problems`	enum Should missing metadata information be reported as error, warning or message?

Value

a list with the location metric (mean or median) and expected range for the location check

---

`util_prep_proportion_check`

Utility function to prepare the metadata for proportion checks

---

Description

Utility function to prepare the metadata for proportion checks

Usage

```
util_prep_proportion_check(
  resp_vars,
  meta_data,
  study_data,
  report_problems = c("error", "warning", "message")
)
```

Arguments

resp_vars	variable list the names of the measurement variables
meta_data	data.frame the data frame that contains metadata attributes of study data
study_data	data.frame the data frame that contains the measurements (hint: missing value codes should be excluded, so the function should be called with ds1, if available)
report_problems	enum Should missing metadata information be reported as error, warning or message?

Value

a list with the expected range for the proportion check

---

util_pretty_vector_string

Prepare a vector four output

Description

Prepare a vector four output

Usage

```
util_pretty_vector_string(v, quote = dQuote)
```

Arguments

v	the vector
quote	function, used for quoting – sQuote or dQuote

Value

the "pretty" collapsed vector as a string.

util_rbind *Bind data frames row-based*

Description

if not all data frames share all columns, missing columns will be filled with NAs.

Usage

```
util_rbind(..., data_frames_list = list())
```

Arguments

...	data.frame none more more data frames
data_frames_list	list optional, a list of data frames

Value

data.frame all data frames appended

Examples

```
## Not run:
util_rbind(head(cars), head(iris))
util_rbind(head(cars), tail(cars))
util_rbind(head(cars)[, "dist", FALSE], tail(cars)[, "speed", FALSE])

## End(Not run)
```

util_recode *Map a vector of values based on an assignment table*

Description

Map a vector of values based on an assignment table

Usage

```
util_recode(values, mapping_table, from, to, default = NULL)
```

Arguments

values	vector the vector
mapping_table	data.frame a table with the mapping table
from	character the name of the column with the "old values"
to	character the name of the column with the "new values"
default	character either one character or on character per value, being used, if an entry from values is not in the from column in ‘mapping_table

Value

the mapped values

See Also

dplyr::recode

---

util_remove_empty_rows

removes empty rows from x

Description

removes empty rows from x

Usage

```
util_remove_empty_rows(x, id_vars = character(0))
```

Arguments

x	data.frame a data frame to be cleaned
id_vars	character column names, that will be treated as empty

Value

data.frame reduced x

`util_remove_na_records`

remove all records, that have at least one NA *in any of the given variables*

Description

remove all records, that have at least one NA in any of the given variables

Usage

```
util_remove_na_records(study_data, vars = colnames(study_data))
```

Arguments

`study_data`	the study data frame
`vars`	the variables being checked for NAs

Value

modified study_data data frame

Examples

```
## Not run:
dta <- iris
dim(util_remove_na_records(dta))
dta$Species[4:6] <- NA
dim(util_remove_na_records(dta))
dim(util_remove_na_records(dta, c("Sepal.Length", "Petal.Length")))

## End(Not run)
```

`util_replace_codes_by_NA`

Utility function to replace missing codes by NA*s*

Description

Substitute all missing codes in a data.frame by NA.

Usage

```
util_replace_codes_by_NA(
  study_data,
  meta_data = "item_level",
  split_char = SPLIT_CHAR,
  sm_code = NULL
)
```

Arguments

study_data	Study data including jump/missing codes as specified in the code conventions
meta_data	Metadata as specified in the code conventions
split_char	Character separating missing codes
sm_code	missing code for NAs, if they have been re-coded by util_combine_missing_lists Codes are expected to be numeric.

Value

a list with a modified data frame and some counts

---

util_replace_hard_limit_violations

Replace limit violations (HARD_LIMITS) by NAs

Description

Replace limit violations (HARD_LIMITS) by NAs

Usage

```
util_replace_hard_limit_violations(study_data, meta_data, label_col)
```

Arguments

study_data	study_data
meta_data	study_data
label_col	variable attribute the name of the column in the metadata with labels of variables

Value

modified study_data

util_setup_rstudio_job *Set up an RStudio job*

Description

Also defines a progress function and a `progress_msg` function in the caller's environment.

Usage

```
util_setup_rstudio_job(job_name = "Job")
```

Arguments

`job_name`	a name for the job

Value

the progress function

Examples

```
## Not run:
  test <- function() {
    util_setup_rstudio_job("xx")
    Sys.sleep(5)
    progress(50)
    progress_msg("halfway through")
    Sys.sleep(5)
    progress(100)
    Sys.sleep(1)
  }
  test()

## End(Not run)
```

util_set_dQuoteString *Utility function to put strings in quotes*

Description

This function generates usual double-quotes for each element of the character vector

Usage

```
util_set_dQuoteString(string)
```

Arguments

string	Character vector

Value

quoted string

---

util_set_size *Attaches attributes about the recommended minimum absolute sizes to the plot p*

Description

Attaches attributes about the recommended minimum absolute sizes to the plot p

Usage

```
util_set_size(p, width_em = NA_integer_, height_em = NA_integer_)
```

Arguments

p	ggplot the plot
width_em	numeric len=1. the minimum width hint in em
height_em	numeric len=1. the minimum height in em

Value

p the modified plot

---

util_set_sQuoteString *Utility function single quote string*

Description

This function generates usual single-quotes for each element of the character vector.

Usage

```
util_set_sQuoteString(string)
```

Arguments

string	Character vector

Value

quoted string

util_sigmagap *Utility function outliers according to the rule of Huber et al.*

Description

This function calculates outliers according to the rule of Huber et al.

Usage

```
util_sigmagap(x)
```

Arguments

x	numeric data to check for outliers

Value

binary vector

util_sixsigma *Utility function for six sigma deviations rule*

Description

This function calculates outliers according to the rule of six sigma deviations.

Usage

```
util_sixsigma(x)
```

Arguments

x	numeric data to check for outliers

Value

binary vector

util_sort_by_order *Sort a vector by order given in some other vector*

Description

Sort a vector by order given in some other vector

Usage

```
util_sort_by_order(x, order, ...)
```

Arguments

x	the vector
order	the "order vector
...	additional arguments passed to sort

Examples

```
## Not run:
util_sort_by_order(c("a", "b", "a", "c", "d"), letters)

## End(Not run)
```

util_stop_if_not *Verify assumptions made by the code, that must be TRUE*

Description

Verify assumptions made by the code, that must be TRUE

Usage

```
util_stop_if_not(..., label, label_only)
```

Arguments

...	see stopifnot
label	character a label for the assumptions, can be missing
label_only	logical if TRUE and label is given, the condition will not be displayed, if FALSE

Value

invisible(FALSE), if not stopped.

See Also

stopifnot

`util_study_var2factor` *Convert a study variable to a factor*

Description

Convert a study variable to a factor

Usage

```
util_study_var2factor(
  resp_vars = NULL,
  study_data,
  meta_data = "item_level",
  label_col = LABEL,
  assume_consistent_codes = TRUE,
  have_cause_label_df = FALSE,
  code_name = c(JUMP_LIST, MISSING_LIST),
  include_sysmiss = TRUE
)
```

Arguments

`resp_vars`	variable list the name of the measurement variables
`study_data`	data.frame the data frame that contains the measurements
`meta_data`	data.frame the data frame that contains metadata attributes of study data
`label_col`	variable attribute the name of the column in the metadata with labels of variables
`assume_consistent_codes`	logical assume, that missing codes are consistent for all variables
`have_cause_label_df`	logical is a missing-code table available
`code_name`	character all lists from the meta_data to use for the coding.
`include_sysmiss`	logical add also a factor level for data values that were `NA` in the original study data (system missingness).

Value

`study_data` converted to factors using the coding provided in `code_name`

`util_sub_string_left_from_.`
Get sub-string left from first .

Description

Get sub-string left from first .

Usage

```
util_sub_string_left_from_.(x)
```

Arguments

x	the string with a least one .

Examples

```
## Not run:
util_sub_string_left_from_.(c("a.b", "asdf.xyz", "asdf.jkl.zuio"))

## End(Not run)
```

`util_sub_string_right_from_.`
Get sub-string right from first .

Description

Get sub-string right from first .

Usage

```
util_sub_string_right_from_.(x)
```

Arguments

x	the string with a least one .

Examples

```
## Not run:
util_sub_string_right_from_.(c("a.b", "asdf.xyz"))
util_sub_string_right_from_.(c("a.b", "asdf.xy.z"))
util_sub_string_right_from_.(c("ab", "asdxy.z"))

## End(Not run)
```

util_table_of_vct *Tabulate a vector*

Description

does the same as `as.data.frame(table(x))` but guarantees a data frame with two columns is returned

Usage

```
util_table_of_vct(Var1)
```

Arguments

`Var1`	vector to ta tabulate

Value

a data frame with columns `Var1` and `Freq`

util_tukey *Utility function Tukey outlier rule*

Description

This function calculates outliers according to the rule of Tukey.

Usage

```
util_tukey(x)
```

Arguments

`x`	numeric data to check for outliers

Value

binary vector

util_validate_known_meta

Utility function verifying syntax of known metadata columns

Description

This function goes through metadata columns, dataquieR supports and verifies for these, that they follow its metadata conventions.

Usage

```
util_validate_known_meta(meta_data)
```

Arguments

meta_data	data.frame the data frame that contains metadata attributes of study data

Value

invisible(NULL)

util_validate_missing_lists

Validate code lists for missing and/or jump codes

Description

will warn/stop on problems

Usage

```
util_validate_missing_lists(
  meta_data,
  cause_label_df,
  assume_consistent_codes = FALSE,
  expand_codes = FALSE,
  suppressWarnings = FALSE,
  label_col
)
```

Arguments

`meta_data`	data.frame the data frame that contains metadata attributes of study data
`cause_label_df`	data.frame missing code table. If missing codes have labels the respective data frame can be specified here, see cause_label_df
`assume_consistent_codes`	logical if TRUE and no labels are given and the same missing/jump code is used for more than one variable, the labels assigned for this code will be the same for all variables.
`expand_codes`	logical if TRUE, code labels are copied from other variables, if the code is the same and the label is set somewhere
`suppressWarnings`	logical warn about consistency issues with missing and jump lists
`label_col`	variable attribute the name of the column in the metadata with labels of variables

Value

list with entries:

- `cause_label_df` updated data frame with labels for missing codes

`util_variable_references` *Find all columns in item-level-metadata, that refer to some other variable*

Description

Find all columns in item-level-metadata, that refer to some other variable

Usage

```
util_variable_references(meta_data = "item_level")
```

Arguments

`meta_data`	data.frame the metadata

Value

character all column names referring to variables from item-level metadata

util_view_file *View a file in most suitable viewer*

Description

View a file in most suitable viewer

Usage

```
util_view_file(file)
```

Arguments

file	the file to view

Value

```
invisible(file)
```

util_warning *Produce a warning message with a useful short stack trace.*

Description

Produce a warning message with a useful short stack trace.

Usage

```
util_warning(
  m,
  ...,
  applicability_problem = NA,
  integrity_indicator = "none",
  level = 0
)
```

Arguments

m	warning message or a condition
...	arguments for sprintf on m, if m is a character
applicability_problem	logical warning indicates unsuitable resp_vars
integrity_indicator	character the warning is an integrity problem, here is the indicator abbreviation..
level	integer level of the warning message (defaults to 0). Higher levels are more severe.

Value

condition the condition object, if the execution is not stopped

---

util_warn_unordered *Warn about a problem in* varname*, if* x *has no natural order*

Description

Also warns, if R does not have a comparison operator for x.

Usage

```
util_warn_unordered(x, varname)
```

Arguments

x	vector of data
varname	character len=1. Variable name for warning messages

Value

invisible(NULL)

VARATT_REQUIRE_LEVELS *Requirement levels of certain metadata columns*

Description

These levels are cumulatively used by the function prep_create_meta and related in the argument level therein.

Usage

```
VARATT_REQUIRE_LEVELS
```

Format

An object of class list of length 5.

Details

currently available:

- 'COMPATIBILITY' = "compatibility"
- 'REQUIRED' = "required"
- 'RECOMMENDED' = "recommended"
- 'OPTIONAL' = "optional"
- 'TECHNICAL' = "technical"

VARIABLE_ROLES *Variable roles can be one of the following:*

Description

- `intro` a variable holding consent-data
- `primary` a primary outcome variable
- `secondary` a secondary outcome variable
- `process` a variable describing the measurement process

Usage

```
VARIABLE_ROLES
```

Format

An object of class `list` of length 4.

WELL_KNOWN_META_VARIABLE_NAMES *Well-known metadata column names, names of metadata columns*

Description

names of the variable attributes in the meta data frame holding the names of the respective observers, devices, lower limits for plausible values, upper limits for plausible values, lower limits for allowed values, upper limits for allowed values, the variable name (column name, e.g. v0020349) used in the study data, the variable name used for processing (readable name, e.g. RR_DIAST_1) and in parameters of the QA-Functions, the variable label, variable long label, variable short label, variable data type (see also DATA_TYPES), re-code for definition of lists of event categories, missing lists and jump lists as CSV strings.

Usage

```
WELL_KNOWN_META_VARIABLE_NAMES
```

Format

An object of class `list` of length 49.

Details

all entries of this list will be mapped to the package's exported NAMESPACE environment directly, i.e. they are available directly by their names too:

- VAR_NAMES, - LABEL, - DATA_TYPE, - VALUE_LABELS, - MISSING_LIST, - JUMP_LIST, - MISSING_LIST_TABLE, - HARD_LIMITS, - DETECTION_LIMITS, - SOFT_LIMITS, - CONTRADICTIONS, - DISTRIBUTION, - DECIMALS, - DATA_ENTRY_TYPE, - CO_VARS, - GROUP_VAR_OBSERVER, - GROUP_VAR_DEVICE, - KEY_OBSERVER, - KEY_DEVICE, - TIME_VAR, - KEY_DATETIME, - PART_VAR, - STUDY_SEGMENT, - KEY_STUDY_SEGMENT, - VARIABLE_ROLE, - VARIABLE_ORDER, - LONG_LABEL, - SOFT_LIMIT_LOW, - SOFT_LIMIT_UP, - HARD_LIMIT_LOW, - HARD_LIMIT_UP, - DETECTION_LIMIT_LOW, - DETECTION_LIMIT_UP, - INCL_SOFT_LIMIT_LOW, - INCL_SOFT_LIMIT_UP, - INCL_HARD_LIMIT_LOW - INCL_HARD_LIMIT_UP, - LOCATION_RANGE, - LOCATION_METRIC, - PROPORTION_RANGE, - LOCATION_LIMIT_LOW, - LOCATION_LIMIT_UP, - INCL_LOCATION_LIMIT_LOW, - INCL_LOCATION_LIMIT_UP, - PROPORTION_LIMIT_LOW, - PROPORTION_LIMIT_UP, - INCL_PROPORTION_LIMIT_LOW, - INCL_PROPORTION_LIMIT_UP, - RECODE

See Also

meta_data_segment for `STUDY_SEGMENT`

Examples

```
print(WELL_KNOWN_META_VARIABLE_NAMES$VAR_NAMES)
# print(VAR_NAMES) # should usually also work
```

`[.dataquieR_result` *Extract Parts of a* `dataquieR` *Result Object*

Description

Extract Parts of a `dataquieR` Result Object

Usage

```
## S3 method for class 'dataquieR_result'
x[...]
```

Arguments

`x`	the `dataquieR` result object
`...`	arguments passed to the implementation for lists.

Value

the sub-list of the `dataquieR` result object with all messages still attached

See Also

base::Extract

`[.dataquieR_resultset2` *Get a subset of a* `dataquieR` `dq_report2` *report*

Description

Get a subset of a `dataquieR` `dq_report2` report

Usage

```
## S3 method for class 'dataquieR_resultset2'
x[row, col, res, drop = FALSE]
```

Arguments

`x`	the report
`row`	the variable names, must be unique
`col`	the function-call-names, must be unique
`res`	the result slot, must be unique
`drop`	drop, if length is 1

Value

a list with results, depending on `drop` and the number of results, the list may contain all requested results in sub-lists. The order of the results follows the order of the row/column/result-names given

---

`[[.dataquieR_result` *Extract Elements of a* `dataquieR` *Result Object*

Description

Extract Elements of a `dataquieR` Result Object

Usage

```
## S3 method for class 'dataquieR_result'
x[[...]]
```

Arguments

`x`	the `dataquieR` result object
`...`	arguments passed to the implementation for lists.

Value

the element of the dataquieR result object with all messages still attached

See Also

base::Extract

`$.dataquieR_result` *Extract elements of a* `dataquieR` *Result Object*

Description

Extract elements of a dataquieR Result Object

Usage

```
## S3 method for class 'dataquieR_result'
x$...
```

Arguments

x	the dataquieR result object
...	arguments passed to the implementation for lists.

Value

the element of the dataquieR result object with all messages still attached

See Also

base::Extract

Index