Package ‘dataquieR’

March 29, 2024

Title Data Quality in Epidemiological Research

Version 2.1.0

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.qihs.uni-greifswald.de/Annotation_of_Metadata.html>.

License BSD_2_clause + file LICENSE

URL https://dataquality.qihs.uni-greifswald.de/

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

Depends R (>= 3.6.0)

Imports dplyr (>= 1.0.2), emmeans, ggplot2 (>= 3.5.0), lme4,
lubridate, MASS, MultinomialCI, parallelMap, patchwork,
R.devices, rlang, robustbase, qmrparser, utils, rio, readr,
scales, withr, lifecycle, units

Suggests GGally, grDevices, cli, whoami, anytime, cowplot (>= 0.9.4),
digest, DT (>= 0.23), flexdashboard, flexsiteboard, htmltools,
knitr, markdown, parallel, parallelly, rJava, rmarkdown,
rstudioapi, testthat (>= 3.1.9), tibble, vdiffr, pkgload,
Rdpack, callr, colorspace, plotly, ggvenn, htmlwidgets, future,
processx, R6, shiny, xml2, mgcv, rvest, textutils

VignetteBuilder knitr

Encoding UTF-8

KeepSource TRUE

Language en-US

RoxygenNote 7.3.1

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>),
Elena Salogni [aut] (<https://orcid.org/0009-0007-3767-7145>),
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 2024-03-29 10:30:02 UTC

R topics documented:

`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.

Indicator

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 `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

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.

Indicator

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.

Indicator

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 `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_only` *Plots and checks for distributions – only*

Description

Descriptor

Usage

```
acc_distributions_only(
  resp_vars = NULL,
  study_data,
  meta_data,
  label_col,
  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

`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_only_ecdf

Plots and checks for distributions – only, but with ecdf

Description

Descriptor

Usage

```
acc_distributions_only_ecdf(
  resp_vars = NULL,
  study_data,
  group_vars = NULL,
  meta_data,
  label_col,
  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
`group_vars`	variable list the name of the observer, device or reader 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
`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.

Indicator

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_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)).

Indicator

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:

- `SummaryTable`: 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 location" (see here. 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.

Descriptor

Usage

```
acc_loess(
  resp_vars,
  group_vars = NULL,
  time_vars,
  co_vars = NULL,
  study_data,
  meta_data,
  label_col = NULL,
  min_obs_in_subgroup = 30,
  resolution = 80,
 comparison_lines = list(type = c("mean/sd", "quartiles"), color = "grey30", linetype =
    2, sd_factor = 0.5),
  mark_time_points = getOption("dataquieR.acc_loess.mark_time_points", FALSE),
  plot_observations = getOption("dataquieR.acc_loess.plot_observations", TRUE),
  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 the name of the variable giving the time of measurement

`co_vars` variable list a vector of covariables for adjustment, for example age and sex. Can be NULL (default) for no adjustment.

`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

`min_obs_in_subgroup`

integer (optional argument) If `group_vars` is specified, this argument can be used to specify the minimum number of observations required for each of the subgroups. Subgroups with fewer observations are excluded. The default number is `30`.

resolution numeric the maximum number of time points used for plotting the trend lines

comparison_lines
list type and style of lines with which trend lines are to be compared. Can be mean +/- 0.5 standard deviation (the factor can be specified differently in sd_factor) or quartiles (Q1, Q2, and Q3). Arguments color and linetype are passed to ggplot2::geom_line().

mark_time_points
logical mark time points with observations (caution, there may be many marks)

plot_observations
logical show observations as scatter plot in the background. If there are co_vars specified, the values of the observations in the plot will also be adjusted for the specified covariables.

plot_format enum AUTO | COMBINED | FACETS | BOTH. Return the plot as one combined plot for all groups or as facet plots (one figure per group). BOTH will return both variants, AUTO will decide based on the number of observers.

Details

If mark_time_points or plot_observations is selected, but would result in plotting more than 400 points, only a sample of the data will be displayed.

Limitations

The application of LOESS 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, high variability of measurements combined with a low number of observations in one level of the group_vars may distort the fit. 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 of 0.75, which provides in most cases reasonable fits. The function acc_loess adapts the span for each level of the group_vars (with at least as many observations as specified in min_obs_in_subgroup and with at least three time points) based on the respective number of observations. LOESS consumes a lot of memory for larger datasets. That is why acc_loess switches to a generalized additive model with integrated smoothness estimation (gam by mgcv) if there are 1000 observations or more for at least one level of the group_vars (similar to geom_smooth from ggplot2).

Value

a list with:

- SummaryPlotList: list with two plots if plot_format = "BOTH", otherwise one of the two figures described below:
  - Loess_fits_facets: The plot contains LOESS-smoothed curves for each level of the group_vars in a separate panel. Added trend lines represent mean and standard deviation or quartiles (specified in comparison_lines) for moving windows over the whole data.
  - Loess_fits_combined: This plot combines all curves into one panel. Given a low number of levels in the group_vars, this plot eases comparisons. However, if the 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.

Indicator

Usage

```
acc_margins(
  resp_vars = NULL,
  group_vars = NULL,
  co_vars = NULL,
  threshold_type = NULL,
  threshold_value,
  min_obs_in_subgroup = 5,
  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"))
acc_margins(resp_vars = "DBP_0",
            study_data = study_data,
            meta_data = meta_data,
            group_vars = "USR_BP_0",
            label_col = LABEL,
            co_vars = "AGE_0")

## 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 3SD 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.

Indicator

Usage

```
acc_multivariate_outlier(
  variable_group = NULL,
  id_vars = NULL,
  label_col,
  n_rules = 4,
  max_non_outliers_plot = 10000,
  criteria = c("tukey", "3sd", "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 | 3SD | 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`, `3SD`, `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 $MD^2_i = (x_i - \mu)^T \Sigma^{-1} (x_i - \mu)$
- 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 3SD 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.

Indicator

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", "3sd", "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. 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 | 3SD | 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)`, `3SD (N)`, `Hubert (N)`, `Sigma-gap (N)`, `NUM_acc_ud_outlu`, `Outliers, low (N)`, `Outliers, high (N)` `Grading`
  - `SummaryData`: `data.frame` with the columns `Variables`, `Mean`, `SD`, `Median`, `Skewness`, `Tukey (N)`, `3SD (N)`, `Hubert (N)`, `Sigma-gap (N)`, `Outliers (N)`, `Outliers, low (N)`, `Outliers, 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 outliers according to the approaches of Tukey (Tukey 1977), 3SD (Saleem et al. 2021), Hubert (Hubert and Vandervieren 2008), and SigmaGap (heuristic)
- An output data frame is generated which indicates the no. possible outliers, the direction of deviations (Outliers, low; Outliers, 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. 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).

Indicator

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 `FLG_acc_ud_shape`

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 3SD 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.

Indicator

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", "3sd", "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. 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 | 3SD | 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)`, `3SD (N)`, `Hubert (N)`, `Sigma-gap (N)`, `NUM_acc_ud_outlu`, `Outliers, low (N)`, `Outliers, high (N)` `Grading`
  - `SummaryData`: `data.frame` with the columns `Variables`, `Mean`, `SD`, `Median`, `Skewness`, `Tukey (N)`, `3SD (N)`, `Hubert (N)`, `Sigma-gap (N)`, `Outliers (N)`, `Outliers, low (N)`, `Outliers, 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 outliers according to the approaches of Tukey (Tukey 1977), 3SD (Saleem et al. 2021), Hubert (Hubert and Vandervieren 2008), and SigmaGap (heuristic)
- An output data frame is generated which indicates the no. possible outliers, the direction of deviations (Outliers, low; Outliers, 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. 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.

Indicator

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`
- `SummaryData`: 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

Deprecated

Usage

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

Arguments

`x`	Deprecated
`...`	Deprecated

Value

Deprecated

`as.list.dataquieR_resultset`

Convert a full `dataquieR` *report to a* `list`

Description

Deprecated

Usage

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

Arguments

`x`	Deprecated
`...`	Deprecated

Value

Deprecated

ASSOCIATION_DIRECTION *Cross-item level metadata attribute name*

Description

The allowable direction of an association. The input is a string that can be either "positive" or "negative".

Usage

```
ASSOCIATION_DIRECTION
```

Format

An object of class `character` of length 1.

See Also

meta_data_cross

Other meta_data_cross: `ASSOCIATION_FORM`, `ASSOCIATION_METRIC`, `ASSOCIATION_RANGE`, `CHECK_ID`, `CHECK_LABEL`, `CONTRADICTION_TERM`, `CONTRADICTION_TYPE`, `DATA_PREPARATION`, `GOLDSTANDARD`, `MULTIVARIATE_OUTLIER_CHECKTYPE`, `N_RULES`, `REL_VAL`, `VARIABLE_LIST`, `util_normalize_cross_item()`

ASSOCIATION_FORM *Cross-item level metadata attribute name*

Description

The allowable form of association. The string specifies the form based on a selected list.

Usage

```
ASSOCIATION_FORM
```

Format

An object of class `character` of length 1.

See Also

meta_data_cross

Other meta_data_cross: `ASSOCIATION_DIRECTION`, `ASSOCIATION_METRIC`, `ASSOCIATION_RANGE`, `CHECK_ID`, `CHECK_LABEL`, `CONTRADICTION_TERM`, `CONTRADICTION_TYPE`, `DATA_PREPARATION`, `GOLDSTANDARD`, `MULTIVARIATE_OUTLIER_CHECKTYPE`, `N_RULES`, `REL_VAL`, `VARIABLE_LIST`, `util_normalize_cross_item()`

ASSOCIATION_METRIC *Cross-item level metadata attribute name*

Description

The metric underlying the association in ASSOCIATION_RANGE. The input is a string that specifies the analysis algorithm to be used.

Usage

```
ASSOCIATION_METRIC
```

Format

An object of class `character` of length 1.

See Also

meta_data_cross

Other meta_data_cross: `ASSOCIATION_DIRECTION`, `ASSOCIATION_FORM`, `ASSOCIATION_RANGE`, `CHECK_ID`, `CHECK_LABEL`, `CONTRADICTION_TERM`, `CONTRADICTION_TYPE`, `DATA_PREPARATION`, `GOLDSTANDARD`, `MULTIVARIATE_OUTLIER_CHECKTYPE`, `N_RULES`, `REL_VAL`, `VARIABLE_LIST`, `util_normalize_cross_item()`

ASSOCIATION_RANGE *Cross-item level metadata attribute name*

Description

Specifies the allowable range of an association. The inclusion of the endpoints follows standard mathematical notation using round brackets for open intervals and square brackets for closed intervals. Values must be separated by a semicolon.

Usage

```
ASSOCIATION_RANGE
```

Format

An object of class `character` of length 1.

See Also

meta_data_cross

Other meta_data_cross: `ASSOCIATION_DIRECTION`, `ASSOCIATION_FORM`, `ASSOCIATION_METRIC`, `CHECK_ID`, `CHECK_LABEL`, `CONTRADICTION_TERM`, `CONTRADICTION_TYPE`, `DATA_PREPARATION`, `GOLDSTANDARD`, `MULTIVARIATE_OUTLIER_CHECKTYPE`, `N_RULES`, `REL_VAL`, `VARIABLE_LIST`, `util_normalize_cross_item()`

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

Description

data.frame with the following columns:

- `CODE_VALUE`: numeric | DATETIME Missing code (the number or date representing a missing)
- `CODE_LABEL`: character a label for the missing code
- `CODE_CLASS`: enum JUMP | MISSING. Class of the missing code.
- `CODE_INTERPRET` enum I | P | PL | R | BO | NC | O | UH | UO | NE. Class of the missing code according to `AAPOR`.
- `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.

See Also

Online

CHECK_ID *Cross-item level metadata attribute name*

Description

Specifies the unique IDs for cross-item level metadata records

Usage

```
CHECK_ID
```

Format

An object of class `character` of length 1.

Details

if missing, `dataquieR` will create such IDs

See Also

meta_data_cross

Other meta_data_cross: `ASSOCIATION_DIRECTION`, `ASSOCIATION_FORM`, `ASSOCIATION_METRIC`, `ASSOCIATION_RANGE`, `CHECK_LABEL`, `CONTRADICTION_TERM`, `CONTRADICTION_TYPE`, `DATA_PREPARATION`, `GOLDSTANDARD`, `MULTIVARIATE_OUTLIER_CHECKTYPE`, `N_RULES`, `REL_VAL`, `VARIABLE_LIST`, `util_normalize_cross_item()`

CHECK_LABEL *Cross-item level metadata attribute name*

Description

Specifies the unique labels for cross-item level metadata records

Usage

```
CHECK_LABEL
```

Format

An object of class `character` of length 1.

Details

if missing, `dataquieR` will create such labels

See Also

meta_data_cross

Other meta_data_cross: `ASSOCIATION_DIRECTION`, `ASSOCIATION_FORM`, `ASSOCIATION_METRIC`, `ASSOCIATION_RANGE`, `CHECK_ID`, `CONTRADICTION_TERM`, `CONTRADICTION_TYPE`, `DATA_PREPARATION`, `GOLDSTANDARD`, `MULTIVARIATE_OUTLIER_CH`
`N_RULES`, `REL_VAL`, `VARIABLE_LIST`, `util_normalize_cross_item()`

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 is described here.

See Also

meta_data_cross

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.

Indicator

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 = lifecycle::deprecated()
)
```

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 deprecated. If you want to have a human readable output, use `SummaryData` instead of `SummaryTable`

Value

a list with:

- `SummaryTable`: data frame about item missingness per response variable
- `SummaryData`: data frame about item missingness per response variable formatted for user
- `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

Indicator

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 != "I", "CODE_VALUE",
    drop =
    TRUE] # sample only replacement codes on item level. I 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

Indicator

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:

`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 metadata. 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.

Descriptor

Usage

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

`meta_data_segment`
data.frame Segment level metadata. Optional.

`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.

Descriptor

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_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.

CONTRADICTION_TERM *Cross-item level metadata attribute name*

Description

Specifies a contradiction rule. Use REDCap like syntax, see online vignette

Usage

```
CONTRADICTION_TERM
```

Format

An object of class character of length 1.

See Also

meta_data_cross

Other meta_data_cross: ASSOCIATION_DIRECTION, ASSOCIATION_FORM, ASSOCIATION_METRIC, ASSOCIATION_RANGE, CHECK_ID, CHECK_LABEL, CONTRADICTION_TYPE, DATA_PREPARATION, GOLDSTANDARD, MULTIVARIATE_OUTLIER_CHECKTYPE N_RULES, REL_VAL, VARIABLE_LIST, util_normalize_cross_item()

CONTRADICTION_TYPE *Cross-item level metadata attribute name*

Description

Specifies the type of a contradiction. According to the data quality concept, there are logical and empirical contradictions, see online vignette

Usage

```
CONTRADICTION_TYPE
```

Format

An object of class character of length 1.

See Also

meta_data_cross

Other meta_data_cross: ASSOCIATION_DIRECTION, ASSOCIATION_FORM, ASSOCIATION_METRIC, ASSOCIATION_RANGE, CHECK_ID, CHECK_LABEL, CONTRADICTION_TERM, DATA_PREPARATION, GOLDSTANDARD, MULTIVARIATE_OUTLIER_CHECKTYPE N_RULES, REL_VAL, VARIABLE_LIST, util_normalize_cross_item()

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.

Descriptor

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 contradictions. 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.

Indicator

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 contradictions. See details for its required structure.

`use_value_labels` logical Deprecated in favor of DATA_PREPARATION. 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. TODO: Not yet controllable by metadata.

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_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.

Indicator

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).

Indicator

Usage

```
con_limit_deviations(
  resp_vars = NULL,
  label_col,
  study_data,
  meta_data,
  limits = NULL,
  flip_mode = "noflip",
  return_flagged_study_data = FALSE
)
```

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`.

`return_flagged_study_data`
logical return `FlaggedStudyData` in the result

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

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. Optional, see `return_flagged_study_data`.
- `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).
- `ReportSummaryTable`: heatmap-like data frame about limit violations

See Also

- Online Documentation

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_resultset_verify *Verify an object of class dataquieR_resultset*

Description

Deprecated

Usage

```
dataquieR_resultset_verify(...)
```

Arguments

...	Deprecated

Value

Deprecated

`DATA_PREPARATION` *Cross-item level metadata attribute name*

Description

For contradiction rules, the required pre-processing steps that can be given. TODO JM: MISSING_LABEL will not work for non-factor variables

Usage

```
DATA_PREPARATION
```

Format

An object of class `character` of length 1.

Details

LABEL MISSING LIMITS MISSING_LABEL MISSING_INTERPRET

See Also

meta_data_cross

Other meta_data_cross: `ASSOCIATION_DIRECTION`, `ASSOCIATION_FORM`, `ASSOCIATION_METRIC`, `ASSOCIATION_RANGE`, `CHECK_ID`, `CHECK_LABEL`, `CONTRADICTION_TERM`, `CONTRADICTION_TYPE`, `GOLDSTANDARD`, `MULTIVARIATE_OUTLIER_CHECKTY` `N_RULES`, `REL_VAL`, `VARIABLE_LIST`, `util_normalize_cross_item()`

`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 metadata attribute VAR_NAMES or, if label_col is set, given in the metadata 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 metadata attribute VAR_NAMES or, if label_col is set, given in the metadata 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

des_scatterplot_matrix *Compute Pairwise Correlations*

Description

works on variable groups (`cross-item_level`), which are expected to show a Pearson correlation

Usage

```
des_scatterplot_matrix(
  study_data,
  meta_data,
  label_col = LABEL,
  meta_data_cross_item = "cross-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
`label_col`	variable attribute the name of the column in the metadata with labels of variables
`meta_data_cross_item`	meta_data_cross

Details

Descriptor # TODO: This can be an indicator

Value

a `list` with the slots:

- `SummaryPlotList`: for each variable group a ggplot object with pairwise correlation plots
- `SummaryData`: table with columns `VARIABLE_LIST`, `cors`, `max_cor`, `min_cor`
- `SummaryTable`: like `SummaryData`, but machine readable and with stable column names.

Examples

```
## Not run:
devtools::load_all()
prep_load_workbook_like_file("meta_data_v2")
des_scatterplot_matrix("study_data")

## End(Not run)
```

`des_summary` *Compute Descriptive Statistics*

Description

generates a descriptive overview on the variables ins `resp_vars`.

Descriptor

Usage

```
des_summary(
  study_data,
  resp_vars = NULL,
  meta_data = "item_level",
  label_col = LABEL
)
```

Arguments

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

`resp_vars` variable 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

Details

TODO

Value

a list with:

- SummaryTable: data frame
- SummaryData: data frame

See Also

Online Documentation

Examples

```
## Not run:
prep_load_workbook_like_file("meta_data_v2")
xx <- des_summary("study_data", meta_data =
                              prep_get_data_frame("item_level"))
util_html_table(xx$SummaryData)
util_html_table(des_summary(study_data = prep_get_data_frame("study_data"),
                    meta_data = prep_get_data_frame("item_level"))$SummaryData)

## End(Not run)
```

`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

dims *Dimension Titles for Prefixes*

Description

order does matter, because it defines the order in the dq_report2.

Usage

```
dims
```

Format

An object of class character of length 5.

See Also

util_html_for_var()

util_html_for_dims()

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

Deprecated

Usage

```
dq_report(...)
```

Arguments

`...`	Deprecated

Value

Deprecated

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",
  meta_data_v2,
  ...,
  dimensions = c("Completeness", "Consistency"),
  cores = list(mode = "socket", logging = FALSE, cpus = util_detect_cores(),
    load.balancing = TRUE),
  specific_args = list(),
  advanced_options = list(),
  author = prep_get_user_name(),
  title = "Data quality report",
  subtitle = as.character(Sys.Date()),
  user_info = NULL,
  debug_parallel = FALSE,
  resp_vars = character(0),
  filter_indicator_functions = character(0),
  filter_result_slots = c("^Summary", "^Segment", "^DataTypePlotList",
    "^ReportSummaryTable", "^Dataframe", "^Result", "^VariableGroup"),
  mode = c("default", "futures", "queue", "parallel"),
  mode_args = list(),
  notes_from_wrapper = list()
)
```

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

`meta_data_v2`
: character path to workbook like metadata file, see `prep_load_workbook_like_file` for details. **ALL LOADED DATAFRAMES WILL BE PURGED**, using `prep_purge_data_frame_cache`, if you specify `meta_data_v2`.

`...`
: 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. Can also be a cluster.

`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.

`advanced_options`
: list options to set during report computation, see `options()`

`author`
: character author for the report documents.

`title`
: character optional argument to specify the title for the data quality report

`subtitle`
: character optional argument to specify a subtitle for the data quality report

`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.

`mode`
: character work mode for parallel execution. default is "default", the values mean: - default: use `queue` except `cores` has been set explicitly - futures: use the `future` package - queue: use a queue as described in the examples from the `callr` package by Csárdi and Chang and start sub-processes as workers that evaluate the queue. - parallel: use the cluster from `cores` to evaluate all calls of indicator functions using the classic R `parallel` back-ends

`mode_args`
: list of arguments for the selected `mode`. As of writing this manual, only for the mode `queue` the argument `step` is supported, which gives the number of

function calls that are run by one worker at a time. the default is 15, which gives on most of the tested systems a good balance between synchronization overhead and idling workers.

`notes_from_wrapper`

list a list containing notes about changed labels by `dq_report_by` (otherwise NULL)

Details

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

Value

a dataquieR_resultset2 that can be printed creating a `HTML`-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 = "item_level",
  meta_data_segment = "segment_level",
  meta_data_dataframe = "dataframe_level",
  meta_data_cross_item = "cross-item_level",
  label_col,
  meta_data_v2,
  meta_data_split = STUDY_SEGMENT,
  study_data_split,
  ...,
  output_dir = NULL,
  also_print = FALSE,
  disable_plotly = 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

`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

`label_col` variable attribute the name of the column in the metadata with labels of variables

`meta_data_v2` character path to workbook like metadata file, see `prep_load_workbook_like_file` for details. **ALL LOADED DATAFRAMES WILL BE PURGED**, using `prep_purge_data_frame_cache`, if you specify `meta_data_v2`.

`meta_data_split` variable attribute name of a metadata 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.

`...` arguments to be passed through to dq_report or dq_report2

`output_dir` character if given, the output is not returned but

`also_print` logical if `output_dir` is not `NULL`, also create `HTML` output for each segment using `print.dataquieR_resultset2()`. written to the path `output_dir`

`disable_plotly` logical do not use `plotly`, even if installed

Value

named list of named lists of dq_report2 reports or, if `output_dir` has been specified, `invisible(NULL)`

See Also

dq_report

Examples

```
## Not run:  # really long-running example.
prep_load_workbook_like_file("meta_data_v2")
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
)
unlink("/tmp/testRep/", force = TRUE, recursive = TRUE)
dq_report_by("study_data",
  label_col = LABEL, study_data_split = "CENTER_0",
  meta_data_split = STUDY_SEGMENT, output_dir = "/tmp/testRep"
)
unlink("/tmp/testRep/", force = TRUE, recursive = TRUE)
dq_report_by("study_data",
  label_col = LABEL, study_data_split = "CENTER_0",
  meta_data_split = NULL, output_dir = "/tmp/testRep"
)
dq_report_by("study_data",
  label_col = LABEL,
  meta_data_split = STUDY_SEGMENT, output_dir = "/tmp/testRep"
)
dq_report_by("study_data",
  label_col = LABEL,
  meta_data_split = STUDY_SEGMENT, output_dir = "/tmp/testRep",
  also_print = TRUE
)
dq_report_by(study_data = "study_data", meta_data_v2 = "meta_data_v2",
  advanced_options = list(dataquieR.study_data_cache_max = 0,
  dataquieR.study_data_cache_metrics = TRUE,
  dataquieR.study_data_cache_metrics_env = environment()),
  cores = NULL, dimensions = "int")
dq_report_by(study_data = "study_data", meta_data_v2 = "meta_data_v2",
  advanced_options = list(dataquieR.study_data_cache_max = 0),
  cores = NULL, dimensions = "int")

## End(Not run)
```

GOLDSTANDARD *Cross-item level metadata attribute name*

Description

Defines the measurement variable to be used as a known gold standard. Only one variable can be defined as the gold standard.

Usage

```
GOLDSTANDARD
```

Format

An object of class `character` of length 1.

See Also

meta_data_cross

Other meta_data_cross: `ASSOCIATION_DIRECTION`, `ASSOCIATION_FORM`, `ASSOCIATION_METRIC`, `ASSOCIATION_RANGE`, `CHECK_ID`, `CHECK_LABEL`, `CONTRADICTION_TERM`, `CONTRADICTION_TYPE`, `DATA_PREPARATION`, `MULTIVARIATE_OUTLIER_CHE` `N_RULES`, `REL_VAL`, `VARIABLE_LIST`, `util_normalize_cross_item()`

`html_dependency_clipboard` *HTML Dependency for report headers in* `clipboard`

Description

HTML Dependency for report headers in `clipboard`

Usage

```
html_dependency_clipboard()
```

Value

the dependency

`html_dependency_dataquieR` *HTML Dependency for* `dataquieR`

Description

generate all dependencies used in static `dataquieR` reports

Usage

```
html_dependency_dataquieR(iframe = FALSE)
```

Arguments

`iframe`	logical`(1)` if `TRUE`, create the dependency used in figure `iframes`.

Value

the dependency

`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_tippy` *HTML Dependency for* `tippy`

Description

HTML Dependency for `tippy`

Usage

```
html_dependency_tippy()
```

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.

Indicator

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.ELEMENT_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.ELEMENT_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.

Indicator

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

Indicator

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())
study_data$v00000 <- as.character(study_data$v00000)
study_data$v00002 <- as.character(study_data$v00002)
study_data$v00002[3] <- ""
appmatrix <- int_datatype_matrix(study_data = study_data,
                                 meta_data = meta_data,
                                 label_col = LABEL)
study_data$v00002[5] <- "X"
appmatrix <- int_datatype_matrix(study_data = study_data,
                                 meta_data = meta_data,
                                 label_col = LABEL)
appmatrix$ReportSummaryTable

## 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.

Indicator

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.

Indicator

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)

Details

Descriptor

Value

empty list, so far – the function only warns.

int_sts_element_dataframe

Determine missing and/or superfluous data elements

Description

Depends on `dataquieR.ELEMENT_MISSMATCH_CHECKTYPE` option, see there – # TODO: Rind 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

Details

Indicator

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

Depends on `dataquieR.ELEMENT_MISSMATCH_CHECKTYPE` option, see there – # TODO: Rind out, how to document and link it here using `Roxygen`.

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.

Details

Indicator

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.ELEMENT_MISSMATCH_CHECKTYPE = "none")
int_sts_element_segment(study_data, meta_data)
options(dataquieR.ELEMENT_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.ELEMENT_MISSMATCH_CHECKTYPE = "none")
int_sts_element_segment(study_data, meta_data)
options(dataquieR.ELEMENT_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.ELEMENT_MISSMATCH_CHECKTYPE = "none")
int_sts_element_segment(study_data, meta_data)
options(dataquieR.ELEMENT_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.

Indicator

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.

Indicator

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.

Indicator

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.

Indicator

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)
```

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_cross *Well known columns on the* `meta_data_cross-item` *sheet*

Description

Metadata describing groups of variables, e.g., for their multivariate distribution or for defining contradiction rules.

See Also

check_table

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

Description

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

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

Description

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

MULTIVARIATE_OUTLIER_CHECKTYPE *Cross-item level metadata attribute name*

Description

Select, which outlier criteria to compute, see acc_multivariate_outlier.

Usage

```
MULTIVARIATE_OUTLIER_CHECKTYPE
```

Format

An object of class character of length 1.

Details

You can leave the cell empty, then, all checks will apply. If you enter a set of methods, the maximum for N_RULES changes. See also UNIVARIATE_OUTLIER_CHECKTYPE.

See Also

meta_data_cross

Other meta_data_cross: ASSOCIATION_DIRECTION, ASSOCIATION_FORM, ASSOCIATION_METRIC, ASSOCIATION_RANGE, CHECK_ID, CHECK_LABEL, CONTRADICTION_TERM, CONTRADICTION_TYPE, DATA_PREPARATION, GOLDSTANDARD, N_RULES, REL_VAL, VARIABLE_LIST, util_normalize_cross_item()

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

N_RULES *Cross-item and item level metadata attribute name*

Description

Select, how many violated outlier criteria make an observation an outlier, see acc_multivariate_outlier.

Usage

```
N_RULES
```

Format

An object of class `character` of length 1.

Details

You can leave the cell empty, then, all applied checks must deem an observation an outlier to have it flagged. See UNIVARIATE_OUTLIER_CHECKTYPE and MULTIVARIATE_OUTLIER_CHECKTYPE for the selected outlier criteria.

See Also

meta_data_cross

meta_data

Other meta_data_cross: `ASSOCIATION_DIRECTION`, `ASSOCIATION_FORM`, `ASSOCIATION_METRIC`, `ASSOCIATION_RANGE`, `CHECK_ID`, `CHECK_LABEL`, `CONTRADICTION_TERM`, `CONTRADICTION_TYPE`, `DATA_PREPARATION`, `GOLDSTANDARD`, `MULTIVARIATE_OUTLIER_CHECKTYPE`, `REL_VAL`, `VARIABLE_LIST`, `util_normalize_cross_item()`

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

Description

Deprecated

Usage

```
pipeline_recursive_result(...)
```

Arguments

`...`	Deprecated

Value

Deprecated

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

Description

Deprecated

Usage

```
pipeline_vectorized(...)
```

Arguments

`...`	Deprecated

Value

Deprecated

plot.dataquieR_summary

Plot a dataquieR summary

Description

Plot a dataquieR summary

Usage

```
## S3 method for class 'dataquieR_summary'
plot(x, y, ..., filter, dont_plot = FALSE, stratify_by)
```

Arguments

x	the dataquieR summary, see summary() and dq_report2()
y	not yet used
...	not yet used
filter	if given, this filters the summary, e.g., filter = call_names == "com_qualified_item_missingness"
dont_plot	suppress the actual plotting, just return a printable object derived from x
stratify_by	column to stratify the summary, may be one string.

Value

invisible html object

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

Other data-frame-cache: `prep_get_data_frame()`, `prep_list_dataframes()`, `prep_load_folder_with_metadata()`, `prep_load_workbook_like_file()`, `prep_purge_data_frame_cache()`

`prep_add_missing_codes` *Insert missing codes for* `NA`*s based on rules*

Description

Insert missing codes for `NA`s 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 metadata.

---

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_for_dataquieR_updates

Check for package updates

Description

Check for package updates

Usage

```
prep_check_for_dataquieR_updates(beta = FALSE, deps = TRUE)
```

Arguments

beta	logical check for beta version too
deps	logical check for missing (optional) dependencies

Value

`invisible(NULL)`

prep_check_meta_data_dataframe

Verify and normalize metadata 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 metadata 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 metadata w.r.t. the provided column names*

Description

This function verifies, if a data frame complies to metadata 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 metadata, 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",
    END_DIGIT_CHECK = "END_DIGIT_CHECK",
    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 metadata data frame: If `label_col` is a label attribute to adjust, this is the metadata 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_combine_report_summaries` *Combine two report summaries*

Description

Combine two report summaries

Combine two report summaries

Usage

```
prep_combine_report_summaries(..., summaries_list, amend_segment_names = FALSE)

prep_combine_report_summaries(..., summaries_list, amend_segment_names = FALSE)
```

Arguments

`...` objects returned by prep_extract_summary

`summaries_list` if given, list of objects returned by prep_extract_summary

`amend_segment_names` logical use names of the `summaries_list` and argument names as segment prefixes

Value

combined summaries

combined summaries

See Also

Other summary_functions: `prep_extract_classes_by_functions()`, `prep_extract_summary()`, `prep_extract_summary.dataquieR_result()`, `prep_extract_summary.dataquieR_resultset2()`, `prep_render_pie_chart_from_summaryclasses_ggplot2()`, `prep_render_pie_chart_from_summaryclasses_plotl` `prep_summary_to_classes()`, `util_as_cat()`, `util_extract_indicator_metrics()`, `util_get_category_for_resu` `util_get_colors()`, `util_get_html_cell_for_result()`, `util_get_labels_grading_class()`, `util_get_message_for_result()`, `util_get_rule_sets()`, `util_get_ruleset_formats()`, `util_get_thresholds()`, `util_html_table()`, `util_melt_summary()`, `util_sort_by_order()`

Other summary_functions: `prep_extract_classes_by_functions()`, `prep_extract_summary()`, `prep_extract_summary.dataquieR_result()`, `prep_extract_summary.dataquieR_resultset2()`, `prep_render_pie_chart_from_summaryclasses_ggplot2()`, `prep_render_pie_chart_from_summaryclasses_plotl` `prep_summary_to_classes()`, `util_as_cat()`, `util_extract_indicator_metrics()`, `util_get_category_for_resu` `util_get_colors()`, `util_get_html_cell_for_result()`, `util_get_labels_grading_class()`, `util_get_message_for_result()`, `util_get_rule_sets()`, `util_get_ruleset_formats()`, `util_get_thresholds()`, `util_html_table()`, `util_melt_summary()`, `util_sort_by_order()`

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

Description

Create a metadata 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:

- metadata attribute names mapped and
- metadata 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_create_meta_data_file

Instantiate a new metadata file

Description

Instantiate a new metadata file

Usage

```
prep_create_meta_data_file(
  file_name,
  study_data,
  open = TRUE,
  overwrite = FALSE
)
```

Arguments

`file_name`	character file path to write to
`study_data`	data.frame optional, study data to guess metadata from
`open`	logical open the file after creation
`overwrite`	logical overwrite `file`, if exists

Value

`invisible(NULL)`

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,
  .dont_cast_off_cols = FALSE
)
```

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.

`.dont_cast_off_cols` logical internal use, only

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,
  mode = c("numeric_codes", "string_codes")
)
```

Arguments

codes	codes, numeric or dates (as default, but string codes can be enabled using the option 'mode', see below)
labels	character labels, same length as codes
split_char	character split character character to split code assignments
mode	character one of two options to insist on numeric or datetime codes (default) or to allow for string codes

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 metadata 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_extract_classes_by_functions

Extract old function based summary from data quality results

Description

Extract old function based summary from data quality results

Usage

```
prep_extract_classes_by_functions(r)
```

Arguments

r	dq_report2

Value

data.frame long format, compatible with `prep_summary_to_classes()`

See Also

Other summary_functions: `prep_combine_report_summaries()`, `prep_extract_summary()`, `prep_extract_summary.da`
`prep_extract_summary.dataquieR_resultset2()`, `prep_render_pie_chart_from_summaryclasses_ggplot2()`, `prep_render_pie_chart_from_summaryclasses_plotly()`, `prep_summary_to_classes()`, `util_as_cat()`, `util_extract_indicator_metrics()`, `util_get_category_for_result()`, `util_get_colors()`, `util_get_html_cell_for_result()`, `util_get_labels_grading_class()`, `util_get_message_for_result()`, `util_get_rule_sets()`, `util_get_ruleset_formats()`, `util_get_thresholds()`, `util_html_table()`, `util_melt_summary()`, `util_sort_by_order()`

`prep_extract_summary` *Extract summary from data quality results*

Description

Generic function, currently supports dq_report2 and dataquieR_result

Usage

```
prep_extract_summary(r, ...)
```

Arguments

`r`	dq_report2 or dataquieR_result object
`...`	further arguments, maybe needed for some implementations

Value

list with two slots `Data` and `Table` with data.frames featuring all metrics columns from the report or result in `x`, the STUDY_SEGMENT and the VAR_NAMES. In case of `Data`, the columns are formatted nicely but still with the standardized column names – use `util_translate_indicator_metrics()` to rename them nicely. In case of `Table`, just as they are.

See Also

Other summary_functions: `prep_combine_report_summaries()`, `prep_extract_classes_by_functions()`, `prep_extract_summary.dataquieR_result()`, `prep_extract_summary.dataquieR_resultset2()`, `prep_render_pie_chart_from_summaryclasses_ggplot2()`, `prep_render_pie_chart_from_summaryclasses_plotl`
`prep_summary_to_classes()`, `util_as_cat()`, `util_extract_indicator_metrics()`, `util_get_category_for_resu`
`util_get_colors()`, `util_get_html_cell_for_result()`, `util_get_labels_grading_class()`, `util_get_message_for_result()`, `util_get_rule_sets()`, `util_get_ruleset_formats()`, `util_get_thresholds()`, `util_html_table()`, `util_melt_summary()`, `util_sort_by_order()`

prep_extract_summary.dataquieR_result

Extract report summary from reports

Description

Extract report summary from reports

Usage

```
## S3 method for class 'dataquieR_result'
prep_extract_summary(r, ...)
```

Arguments

r	dataquieR_result a result from adq_report2 report
...	not used

Value

list with two slots Data and Table with data.frames featuring all metrics columns from the report r, the STUDY_SEGMENT and the VAR_NAMES. In case of Data, the columns are formatted nicely but still with the standardized column names – use util_translate_indicator_metrics() to rename them nicely. In case of Table, just as they are.

See Also

prep_combine_report_summaries()

Other summary_functions: prep_combine_report_summaries(), prep_extract_classes_by_functions(), prep_extract_summary(), prep_extract_summary.dataquieR_resultset2(), prep_render_pie_chart_from_summar
prep_render_pie_chart_from_summaryclasses_plotly(), prep_summary_to_classes(), util_as_cat(), util_extract_indicator_metrics(), util_get_category_for_result(), util_get_colors(), util_get_html_cell_for_result(), util_get_labels_grading_class(), util_get_message_for_result(), util_get_rule_sets(), util_get_ruleset_formats(), util_get_thresholds(), util_html_table(), util_melt_summary(), util_sort_by_order()

prep_extract_summary.dataquieR_resultset2

Extract report summary from reports

Description

Extract report summary from reports

Usage

```
## S3 method for class 'dataquieR_resultset2'
prep_extract_summary(r, ...)
```

Arguments

`r`	dq_report2 a dq_report2 report
`...`	not used

Value

list with two slots `Data` and `Table` with data.frames featuring all metrics columns from the report `r`, the STUDY_SEGMENT and the VAR_NAMES. In case of `Data`, the columns are formatted nicely but still with the standardized column names – use `util_translate_indicator_metrics()` to rename them nicely. In case of `Table`, just as they are.

See Also

`prep_combine_report_summaries()`

Other summary_functions: `prep_combine_report_summaries()`, `prep_extract_classes_by_functions()`, `prep_extract_summary()`, `prep_extract_summary.dataquieR_result()`, `prep_render_pie_chart_from_summarycla`
`prep_render_pie_chart_from_summaryclasses_plotly()`, `prep_summary_to_classes()`, `util_as_cat()`, `util_extract_indicator_metrics()`, `util_get_category_for_result()`, `util_get_colors()`, `util_get_html_cell_for_result()`, `util_get_labels_grading_class()`, `util_get_message_for_result()`, `util_get_rule_sets()`, `util_get_ruleset_formats()`, `util_get_thresholds()`, `util_html_table()`, `util_melt_summary()`, `util_sort_by_order()`

`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,
  keep_types = FALSE
)
```

Arguments

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

`.data_frame_list`
: environment cache for loaded data frames

`keep_types`
: logical keep types as possibly defined in a file, if the data frame is loaded from one. set `TRUE` for study data.

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

Other data-frame-cache: `prep_add_data_frames()`, `prep_list_dataframes()`, `prep_load_folder_with_metadata()`, `prep_load_workbook_like_file()`, `prep_purge_data_frame_cache()`

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_labels *Fetch a label for a variable based on its purpose*

Description

Fetch a label for a variable based on its purpose

Usage

```
prep_get_labels(
  resp_vars,
  meta_data = "item_level",
  label_col,
  max_len = MAX_LABEL_LEN,
  label_class = c("SHORT", "LONG"),
  label_lang = "",
  resp_vars_are_var_names_only = FALSE
)
```

Arguments

`resp_vars`	variable list the variable names to fetch for
`meta_data`	meta_data the metadata, item-level
`label_col`	variable attribute the name of the column in the metadata with labels of variables
`max_len`	integer the maximum label length to return, if not possible w/o causing ambiguous labels, the labels may still be longer
`label_class`	enum SHORT | LONG. which sort of label according to the metadata model should be returned
`label_lang`	character optional language suffix, if available in the metadata
`resp_vars_are_var_names_only`	logical If `TRUE`, do not use other labels than VAR_NAMES for finding `resp_vars` in `meta_data`

Value

character suitable labels for each `resp_vars`, names of this vector are VAR_NAMES

Examples

```
## Not run:
prep_load_workbook_like_file("meta_data_v2")
prep_get_labels("SEX_0", label_class = "SHORT", max_len = 2)

## 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

See Also

Other data-frame-cache: `prep_add_data_frames()`, `prep_get_data_frame()`, `prep_load_folder_with_metadata()`, `prep_load_workbook_like_file()`, `prep_purge_data_frame_cache()`

`prep_load_folder_with_metadata` *Pre-load a folder 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_folder_with_metadata(folder, keep_types = FALSE, ...)
```

Arguments

`folder`	the folder name to load.
`keep_types`	logical keep types as possibly defined in the file. set `TRUE` for study data.
`...`	arguments passed to []

Details

Note, that this function in contrast to prep_get_data_frame does neither support selecting specific sheets/columns from a file.

Value

`invisible(the cache environment)`

See Also

prep_add_data_frames

prep_get_data_frame

Other data-frame-cache: `prep_add_data_frames()`, `prep_get_data_frame()`, `prep_list_dataframes()`, `prep_load_workbook_like_file()`, `prep_purge_data_frame_cache()`

Examples

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

## End(Not run)
```

`prep_load_report` *Load a* `dq_report2`

Description

Load a `dq_report2`

Usage

```
prep_load_report(file)
```

Arguments

`file`	character the file name to load from

Value

dataquieR_resultset2 the report

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, keep_types = FALSE)
```

Arguments

file	the file name to load.
keep_types	logical keep types as possibly defined in the file. set TRUE for study data.

Details

Note, that this function in contrast to prep_get_data_frame does neither support selecting specific sheets/columns from a file.

Value

invisible(the cache environment)

See Also

prep_add_data_frames

prep_get_data_frame

Other data-frame-cache: prep_add_data_frames(), prep_get_data_frame(), prep_list_dataframes(), prep_load_folder_with_metadata(), prep_purge_data_frame_cache()

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 metadata 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 metadata 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, defaults to `TRUE`, except if called in a dq_report2 pipeline.

`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,
  .adjust_data_type = TRUE,
  .amend_scale_level = TRUE,
 .internal = rlang::env_inherits(rlang::caller_env(), parent.env(environment()))
)
```

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 `NA`s, 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

`.adjust_data_type`
ensure that the data type of variables in the study data corresponds to their data type specified in the metadata

`.amend_scale_level`
ensure that `SCALE_LEVEL` is available in the item-level `meta_data`. internally used to prevent recursion, if called from `prep_scalelevel_from_data_and_metadata()`.

`.internal` logical internally called, modify caller's environment.

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, if .internal is TRUE.

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

Clear data frame cache

Description

Clear data frame cache

Usage

```
prep_purge_data_frame_cache()
```

Value

nothing

See Also

Other data-frame-cache: prep_add_data_frames(), prep_get_data_frame(), prep_list_dataframes(), prep_load_folder_with_metadata(), prep_load_workbook_like_file()

prep_render_pie_chart_from_summaryclasses_ggplot2

Create a ggplot2 *pie chart*

Description

Create a ggplot2 pie chart

Usage

```
prep_render_pie_chart_from_summaryclasses_ggplot2(
  data,
  meta_data = "item_level"
)
```

Arguments

data	data as returned by prep_summary_to_classes but summarized by one column (currently, we support indicator_metric, STUDY_SEGMENT, and VAR_NAMES)
meta_data	meta_data

Value

a ggplot2 plot

See Also

Other summary_functions: prep_combine_report_summaries(), prep_extract_classes_by_functions(), prep_extract_summary(), prep_extract_summary.dataquieR_result(), prep_extract_summary.dataquieR_resul
prep_render_pie_chart_from_summaryclasses_plotly(), prep_summary_to_classes(), util_as_cat(), util_extract_indicator_metrics(), util_get_category_for_result(), util_get_colors(), util_get_html_cell_for_result(), util_get_labels_grading_class(), util_get_message_for_result(), util_get_rule_sets(), util_get_ruleset_formats(), util_get_thresholds(), util_html_table(), util_melt_summary(), util_sort_by_order()

`prep_render_pie_chart_from_summaryclasses_plotly` *Create a* `plotly` *pie chart*

Description

Create a `plotly` pie chart

Usage

```
prep_render_pie_chart_from_summaryclasses_plotly(
  data,
  meta_data = "item_level"
)
```

Arguments

`data` data as returned by `prep_summary_to_classes` but summarized by one column (currently, we support `indicator_metric`, `call_names`, `STUDY_SEGMENT`, and `VAR_NAMES`)

`meta_data` meta_data

Value

a `htmltools` compatible object

See Also

Other summary_functions: `prep_combine_report_summaries()`, `prep_extract_classes_by_functions()`, `prep_extract_summary()`, `prep_extract_summary.dataquieR_result()`, `prep_extract_summary.dataquieR_resul` `prep_render_pie_chart_from_summaryclasses_ggplot2()`, `prep_summary_to_classes()`, `util_as_cat()`, `util_extract_indicator_metrics()`, `util_get_category_for_result()`, `util_get_colors()`, `util_get_html_cell_for_result()`, `util_get_labels_grading_class()`, `util_get_message_for_result()`, `util_get_rule_sets()`, `util_get_ruleset_formats()`, `util_get_thresholds()`, `util_html_table()`, `util_melt_summary()`, `util_sort_by_order()`

`prep_save_report` *Save a* `dq_report2`

Description

Save a `dq_report2`

Usage

```
prep_save_report(report, file, compression_level = 3)
```

Arguments

`report`	dataquieR_resultset2 the report
`file`	character the file name to write to
`compression_level`	integer from=0 to=9. Compression level. 9 is very slow.

Value

`invisible(NULL)`

`prep_scalelevel_from_data_and_metadata`

Heuristics to amend a SCALE_LEVEL column and a UNIT column in the metadata

---

Description

...if missing

Usage

```
prep_scalelevel_from_data_and_metadata(
  resp_vars = NULL,
  study_data,
  meta_data = "item_level",
  label_col = LABEL
)
```

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

Value

data.frame modified metadata

Examples

```
## Not run:
  prep_load_workbook_like_file("meta_data_v2")
  prep_scalelevel_from_data_and_metadata(study_data = "study_data")

## End(Not run)
```

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

Description

Guess a minimum metadata 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 factor columns to coded integers. if selected, then also the study data will be updated and returned.

Details

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

The function also tries to detect missing codes.

Value

a meta_data data frame or a list with study data and metadata, if `convert_factors` == `TRUE`.

Examples

```
## Not run:
dataquieR::prep_study2meta(Orange, convert_factors = FALSE)

## End(Not run)
```

prep_summary_to_classes *Classify metrics from a report summary table*

Description

Classify metrics from a report summary table

Usage

```
prep_summary_to_classes(report_summary)
```

Arguments

`report_summary`	`list()` as returned by `prep_extract_summary()`

Value

data.frame classes for the report summary table, long format

See Also

Other summary_functions: `prep_combine_report_summaries()`, `prep_extract_classes_by_functions()`, `prep_extract_summary()`, `prep_extract_summary.dataquieR_result()`, `prep_extract_summary.dataquieR_resul`
`prep_render_pie_chart_from_summaryclasses_ggplot2()`, `prep_render_pie_chart_from_summaryclasses_plotl`
`util_as_cat()`, `util_extract_indicator_metrics()`, `util_get_category_for_result()`, `util_get_colors()`, `util_get_html_cell_for_result()`, `util_get_labels_grading_class()`, `util_get_message_for_result()`, `util_get_rule_sets()`, `util_get_ruleset_formats()`, `util_get_thresholds()`, `util_html_table()`, `util_melt_summary()`, `util_sort_by_order()`

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 dq_report2

Description

Print a dataquieR result returned by dq_report2

Usage

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

Arguments

x	list a dataquieR result from dq_report2 or util_eval_to_dataquieR_result
...	passed to print. Additionally, the argument slot may be passed to print only specific sub-results.

Value

see print

See Also

util_pretty_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(...)
```

Arguments

...	deprecated

Value

deprecated

print.dataquieR_resultset2

Generate a HTML-based report from a dataquieR report

Description

Generate a HTML-based report from a dataquieR report

Usage

```
## S3 method for class 'dataquieR_resultset2'
print(x, dir, view = TRUE, disable_plotly = FALSE, block_load_factor = 4, ...)
```

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
`block_load_factor`	numeric multiply size of parallel compute blocks by this factor.
`...`	additional arguments:

Value

file names of the generated report's HTML files

print.dataquieR_summary

Print a `dataquieR` *summary*

Description

Print a `dataquieR` summary

Usage

```
## S3 method for class 'dataquieR_summary'
print(
  x,
  ...,
  grouped_by = c("call_names", "indicator_metric"),
  dont_print = FALSE
)
```

Arguments

`x`	the `dataquieR` summary, see `summary()` and `dq_report2()`
`...`	not yet used
`grouped_by`	define the columns of the resulting matrix. It can be either "call_names", one column per function, or "indicator_metric", one column per indicator or both c("call_names", "indicator_metric"). The last combination is the default
`dont_print`	suppress the actual printing, just return a printable object derived from `x`

Value

invisible html object

`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

`REL_VAL` *Cross-item level metadata attribute name*

Description

Specifies the type of reliability or validity analysis. The string specifies the analysis algorithm to be used, and can be either "inter-class" or "intra-class".

Usage

```
REL_VAL
```

Format

An object of class `character` of length 1.

See Also

meta_data_cross

Other meta_data_cross: `ASSOCIATION_DIRECTION`, `ASSOCIATION_FORM`, `ASSOCIATION_METRIC`, `ASSOCIATION_RANGE`, `CHECK_ID`, `CHECK_LABEL`, `CONTRADICTION_TERM`, `CONTRADICTION_TYPE`, `DATA_PREPARATION`, `GOLDSTANDARD`, `MULTIVARIATE_OUTLIER_CHECKTYPE`, `N_RULES`, `VARIABLE_LIST`, `util_normalize_cross_item()`

`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

SCALE_LEVELS

Scale Levels

Description

Scale Levels of Study Data according to `Stevens's` **Typology:**

In the metadata, the following entries are allowed for the variable attribute SCALE_LEVEL:

Usage

```
SCALE_LEVELS
```

Format

An object of class `list` of length 5.

Details

- `nominal` for categorical variables
- `ordinal` for ordinal variables (i.e., comparison of values is possible)
- `interval` for interval scales, i.e., distances are meaningful
- `ratio` for ratio scales, i.e., ratios are meaningful
- `na` for variables, that contain e.g. unstructured texts, `json`, `xml`, ... to distinguish them from variables, that still need to have the `SCALE_LEVEL` estimated by `prep_scalelevel_from_data_and_metadata()`

Examples:

- sex, eye color – `nominal`
- income group, education level – `ordinal`
- temperature in degree Celsius – `interval`
- body weight, temperature in Kelvin – `ratio`

See Also

Wikipedia

`SEGMENT_ID_REF_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_REF_TABLE
```

Format

An object of class `character` of length 1.

See Also

meta_data_segment

`SEGMENT_ID_TABLE` *Deprecated 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.

Details

Please use SEGMENT_ID_REF_TABLE

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_MISS *Segment level metadata attribute name*

Description

`true` or `false` to suppress crude segment missingness output (`Completeness/Misg. Segments` in the report). Defaults to compute the output, if more than one segment is available in the item-level metadata.

Usage

```
SEGMENT_MISS
```

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 metadata 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

Deprecated

Usage

```
## S3 method for class 'dataquieR_resultset'
summary(...)
```

Arguments

`...`	Deprecated

Value

Deprecated

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", "anamat", "indicator_or_descriptor"),
  FUN,
  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 dataquieR report

Examples

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

## End(Not run)
```

UNITS — *Valid unit symbols according to* units::valid_udunits()

Description

like m, g, N, ...

See Also

Other UNITS: UNIT_IS_COUNT, UNIT_PREFIXES, UNIT_SOURCES, WELL_KNOWN_META_VARIABLE_NAMES

UNIT_IS_COUNT — *Is a unit a count according to* units::valid_udunits()

Description

see column def, therein

Details

like %, ppt, ppm

See Also

Other UNITS: UNITS, UNIT_PREFIXES, UNIT_SOURCES, WELL_KNOWN_META_VARIABLE_NAMES

UNIT_PREFIXES — *Valid unit prefixes according to* units::valid_udunits_prefixes()

Description

like k, m, M, c, ...

See Also

Other UNITS: UNITS, UNIT_IS_COUNT, UNIT_SOURCES, WELL_KNOWN_META_VARIABLE_NAMES

UNIT_SOURCES *Maturity stage of a unit according to* units::valid_udunits()

Description

see column source_xml therein, i.e., base, derived, accepted, or common

See Also

Other UNITS: UNITS, UNIT_IS_COUNT, UNIT_PREFIXES, WELL_KNOWN_META_VARIABLE_NAMES

UNIVARIATE_OUTLIER_CHECKTYPE *Item level metadata attribute name*

Description

Select, which outlier criteria to compute, see acc_univariate_outlier.

Usage

```
UNIVARIATE_OUTLIER_CHECKTYPE
```

Format

An object of class character of length 1.

Details

You can leave the cell empty, then, all checks will apply. If you enter a set of methods, the maximum for N_RULES changes. See also MULTIVARIATE_OUTLIER_CHECKTYPE.

See Also

WELL_KNOWN_META_VARIABLE_NAMES

util_compute_kurtosis *Compute Kurtosis*

Description

Compute Kurtosis

Usage

```
util_compute_kurtosis(x)
```

Arguments

x	data

Value

the Kurtosis

util_compute_SE_skewness *Compute SE.Skewness*

Description

Compute SE.Skewness

Usage

```
util_compute_SE_skewness(x, skewness = util_compute_skewness(x))
```

Arguments

x	data
skewness	if already known

Value

the standard error of skewness

`util_compute_skewness` *Compute the Skewness*

Description

Compute the Skewness

Usage

```
util_compute_skewness(x)
```

Arguments

`x`	data

Value

the Skewness

`util_first_row_to_colnames` *Move the first row of a data frame to its column names*

Description

Move the first row of a data frame to its column names

Usage

```
util_first_row_to_colnames(dfr)
```

Arguments

`dfr`	data.frame

Value

data.frame with first row as column names

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_LIST *Cross-item level metadata attribute name*

Description

Specifies a group of variables for multivariate analyses. Separated by `|`, please use variable names from VAR_NAMES or a label as specified in `label_col`, usually LABEL or LONG_LABEL.

Usage

```
VARIABLE_LIST
```

Format

An object of class `character` of length 1.

Details

if missing, `dataquieR` will create such IDs from CONTRADICTION_TERM, if specified.

See Also

meta_data_cross

Other meta_data_cross: ASSOCIATION_DIRECTION, ASSOCIATION_FORM, ASSOCIATION_METRIC, ASSOCIATION_RANGE, CHECK_ID, CHECK_LABEL, CONTRADICTION_TERM, CONTRADICTION_TYPE, DATA_PREPARATION, GOLDSTANDARD, MULTIVARIATE_OUTLIER_CHECKTYPE, N_RULES, REL_VAL, util_normalize_cross_item()

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
- `suppress` a variable added on the fly computing sub-reports, i.e., by dq_report_by to have all referred variables available, even if they are not part of the currently processed segment. But they will only be fully assessed in their real segment's report.

Usage

```
VARIABLE_ROLES
```

Format

An object of class `list` of length 5.

WELL_KNOWN_META_VARIABLE_NAMES *Well-known metadata column names, names of metadata columns*

Description

names of the variable attributes in the metadata 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. For valid units see UNITS.

Usage

```
WELL_KNOWN_META_VARIABLE_NAMES
```

Format

An object of class list of length 53.

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, - SCALE_LEVEL, - UNIT, - VALUE_LABELS, - MISSING_LIST, - JUMP_LIST, - MISSING_LIST_TABLE, - HARD_LIMITS, - DETECTION_LIMITS, - SOFT_LIMITS, - CONTRADICTIONS, - DISTRIBUTION, - DECIMALS, - DATA_ENTRY_TYPE, - END_DIGIT_CHECK, - 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, - GRADING_RULESET

See Also

meta_data_segment for STUDY_SEGMENT

Other UNITS: UNITS, UNIT_IS_COUNT, UNIT_PREFIXES, UNIT_SOURCES

Examples

```
print(WELL_KNOWN_META_VARIABLE_NAMES$VAR_NAMES)
# print(VAR_NAMES) # should usually also work
```

[.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

Index