Package ‘dataquieR’

September 3, 2021

Title  Data Quality in Epidemiological Research
Version  1.0.9
License  BSD_2_clause + file LICENSE
URL  https://dataquality.ship-med.uni-greifswald.de/
BugReports  https://gitlab.com/libreumg/dataquier/-/issues
Depends  R (>= 3.6.0)
Imports  patchwork, dplyr (>= 1.0.2), emmeans, ggplot2 (>= 2.1.0), ggpubr, lme4, lubridate, MASS, MultinomialCI, parallelMap, R.devices, reshape, rlang, robustbase, utils
Suggests  cowplot (>= 0.9.4), anytime, digest, DT (>= 0.15), flexdashboard, htmltools, knitr, rmarkdown, rstudioapi, testthat (>= 2.3.2), tibble, markdown, vdiffr, parallel
VignetteBuilder  knitr
Encoding  UTF-8
KeepSource  TRUE
Language  en-US
RoxygenNote  7.1.1
NeedsCompilation  no
Author  University Medicine Greifswald [cph], Adrian Richter [aut], Carsten Oliver Schmidt [aut], Stephan Struckmann [aut, cre]
Maintainer  Stephan Struckmann <stephan.struckmann@uni-greifswald.de>
Repository  CRAN
Date/Publication  2021-09-03 12:10:09 UTC
### R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>.variable_arg_roles</td>
<td>4</td>
</tr>
<tr>
<td>acc_distributions</td>
<td>5</td>
</tr>
<tr>
<td>acc_end_digits</td>
<td>6</td>
</tr>
<tr>
<td>acc_loess</td>
<td>7</td>
</tr>
<tr>
<td>accMargins</td>
<td>8</td>
</tr>
<tr>
<td>acc_multivariate_outlier</td>
<td>11</td>
</tr>
<tr>
<td>acc_robust_univariate_outlier</td>
<td>12</td>
</tr>
<tr>
<td>acc_shape_or_scale</td>
<td>14</td>
</tr>
<tr>
<td>acc_univariate_outlier</td>
<td>16</td>
</tr>
<tr>
<td>acc_varcomp</td>
<td>18</td>
</tr>
<tr>
<td>as.data_frame.dataquieR_resultset</td>
<td>20</td>
</tr>
<tr>
<td>as.list.dataquieR_resultset</td>
<td>20</td>
</tr>
<tr>
<td>com_item_missingness</td>
<td>21</td>
</tr>
<tr>
<td>com_segment_missingness</td>
<td>22</td>
</tr>
<tr>
<td>com_unit_missingness</td>
<td>24</td>
</tr>
<tr>
<td>contradiction_functions</td>
<td>25</td>
</tr>
<tr>
<td>contradiction_functions_descriptions</td>
<td>26</td>
</tr>
<tr>
<td>con_contradictions</td>
<td>27</td>
</tr>
<tr>
<td>con_detection_limits</td>
<td>29</td>
</tr>
<tr>
<td>con_inadmissible_categorical</td>
<td>31</td>
</tr>
<tr>
<td>con_limit_deviations</td>
<td>33</td>
</tr>
<tr>
<td>dataquieR</td>
<td>35</td>
</tr>
<tr>
<td>dataquieR_resultset</td>
<td>35</td>
</tr>
<tr>
<td>dataquieR_resultset_verify</td>
<td>36</td>
</tr>
<tr>
<td>DATA_TYPES</td>
<td>36</td>
</tr>
<tr>
<td>DATA_TYPES_OF_R_TYPE</td>
<td>37</td>
</tr>
<tr>
<td>dimensions</td>
<td>37</td>
</tr>
<tr>
<td>DISTRIBUTIONS</td>
<td>38</td>
</tr>
<tr>
<td>dq_report</td>
<td>38</td>
</tr>
<tr>
<td>dq_report_by</td>
<td>41</td>
</tr>
<tr>
<td>html_dependency_vert_dt</td>
<td>42</td>
</tr>
<tr>
<td>int_datatype_matrix</td>
<td>42</td>
</tr>
<tr>
<td>pipeline_recursive_result</td>
<td>44</td>
</tr>
<tr>
<td>pipeline_vectorized</td>
<td>45</td>
</tr>
<tr>
<td>prep_add_to_meta</td>
<td>48</td>
</tr>
<tr>
<td>prep_check_meta_names</td>
<td>49</td>
</tr>
<tr>
<td>prep_clean_labels</td>
<td>50</td>
</tr>
<tr>
<td>prep_create_meta</td>
<td>52</td>
</tr>
<tr>
<td>prep_datatype_from_data</td>
<td>53</td>
</tr>
<tr>
<td>prep_map_labels</td>
<td>53</td>
</tr>
<tr>
<td>prep_min_obs_level</td>
<td>54</td>
</tr>
<tr>
<td>prep_pmap</td>
<td>55</td>
</tr>
<tr>
<td>prep_prepare_dataframes</td>
<td>56</td>
</tr>
<tr>
<td>prep_study2meta</td>
<td>58</td>
</tr>
<tr>
<td>prep_value_labels_from_data</td>
<td>59</td>
</tr>
<tr>
<td>print.dataquieR_result</td>
<td>59</td>
</tr>
</tbody>
</table>
R topics documented:

- print.dataquieR_resultset ................................................. 60
- print.ReportSummaryTable .............................................. 61
- pro_applicability_matrix .............................................. 62
- rbind.ReportSummaryTable .............................................. 64
- SPLIT_CHAR ................................................................. 64
- summary.dataquieR_resultset .......................................... 65
- util_anytime_installed .................................................. 66
- util_app_cd ................................................................. 66
- util_app_dc ................................................................. 67
- util_app_dl ................................................................. 67
- util_app_ed ................................................................. 68
- util_app_iac ............................................................... 69
- util_app_iav ............................................................... 69
- util_app_im ................................................................. 70
- util_app_loess ............................................................. 71
- util_app_mar ............................................................... 71
- util_app_mol ............................................................... 72
- util_app_ol ................................................................. 73
- util_app_sm ................................................................. 73
- util_app_sos ............................................................... 74
- util_app_vc ................................................................. 75
- util_assign_levlabs ...................................................... 75
- util_as_numeric ............................................................ 76
- util_backtickQuote ....................................................... 77
- util_check_data_type .................................................... 77
- util_check_one_unique_value ......................................... 78
- util_compare_meta_with_study ....................................... 78
- util_correct_variable_use .......................................... 79
- util_count_codes ......................................................... 81
- util_count_code_classes ............................................... 81
- util_count_NA ............................................................... 82
- util_detect_cores ......................................................... 82
- util_dichotomize .......................................................... 83
- util_dist_selection ........................................................ 83
- util_empty ................................................................. 84
- utilEnsure_suggested ................................................... 84
- util_error ................................................................. 85
- util_find_external_functions_in_stacktrace ...................... 85
- util_find_first_externally_called_functions_in_stacktrace .... 86
- util_fix_rstudio_bugs ................................................... 86
- util_get_code_list ...................................................... 87
- util_get_var_att_names_of_level .................................... 87
- util_heatmap_1th ......................................................... 88
- util_hubert ............................................................... 89
- util_interpret_limits .................................................... 89
- util_is_integer ............................................................ 90
- utilLooks_like_missing ................................................ 90
- util_make_function ...................................................... 91
### Variable-argument roles

A Variable-argument role is the intended use of an argument of a indicator function – an argument that refers variables. In general for the table `.variable_arg_roles`, the suffix `_var` means one variable allowed, while `_vars` means more than one. The default sets of arguments for `util_correct_variable_use/`util_correct_variable_use2` are defined from the point of usage, e.g. if it could be, that NAs are in the list of variable names, the function should be able to remove certain response variables from the output and not disallow them by setting `allow_na` to `FALSE`.

### Usage

`.variable_arg_roles`

### Format

An object of class `tbl_df` (inherits from `tbl, data.frame`) with 14 rows and 7 columns.

### See Also

- `util_correct_variable_use()`
- `util_correct_variable_use2()`
acc_distributions

Function to plot histograms added by empirical cumulative distributions for subgroups

Description

Function to identify inadmissible measurements according to hard limits (multiple variables)

Usage

acc_distributions(
  resp_vars = NULL,
  label_col,
  group_vars = NULL,
  study_data,
  meta_data
)

Arguments

resp_vars variable list the names of the measurement variables
label_col variable attribute the name of the column in the metadata with labels of 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

Value

A list with:

• SummaryPlots: list of ggplots for each response variable in resp_vars.

ALGORITHM OF THIS IMPLEMENTATION:

• 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 limits defined in the metadata
• Plot histograms
  – If group_vars is specified by the user, distributions within group-wise ecdf are presented.

See Also

Online Documentation
acc_end_digits  

Extension of acc_shape_or_scale to examine uniform distributions of end digits

Description

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

Usage

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

Arguments

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

Value

a list with:

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

ALGORITHM OF THIS IMPLEMENTATION:

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

See Also

Online Documentation
Description

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

Usage

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

Arguments

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

Details

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

Limitations

The application of LOESS usually requires model fitting, i.e. the smoothness of a model is subject to a smoothing parameter (span). Particularly in the presence of interval-based missing data (USR\_181), high variability of measurements combined with a low number of observations in one level of the group\_vars the fit to the data may be distorted. Since our approach handles data without knowledge of such underlying characteristics, finding the best fit is complicated if computational costs should be minimal. The default of LOESS in R uses a span 0.75 which provides in most cases reasonable fits. The function above increases the fit to the data automatically if the minimum of observations in one level of the group\_vars is higher than \texttt{n=30}.

Value

a list with:

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

See Also

Online Documentation

---

\texttt{accMargins}  \textit{Function to estimate marginal means, see \texttt{emmeans::emmeans}}
acc_margins

Description

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

CAT: Unexpected distribution w.r.t. location

Marginal means

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

Usage

acc_margins(
  resp_vars = NULL,
  group_vars = NULL,
  co_vars = NULL,
  threshold_type = NULL,
  threshold_value,
  min_obs_in_subgroup,
  study_data,
  meta_data,
  label_col
)

Arguments

resp_vars variable the name of the continuous measurement variable

group_vars variable list len=1-1. the name of the observer, device or reader variable

co_vars variable list a vector of covariables, e.g. age and sex for adjustment

threshold_type enum empirical | user | none. In case empirical is chosen a multiplier of the scale measure is used, in case of user a value of the mean or probability (binary data) has to be defined see Implementation and use of thresholds. In case of none, no thresholds are displayed and no flagging of unusual group levels is applied.

threshold_value numeric a multiplier or absolute value see Implementation and use of thresholds

min_obs_in_subgroup integer from=0. optional argument if a "group_var" is used. This argument specifies the minimum no. of observations that is required to include a subgroup (level) of the "group_var" in the analysis. Subgroups with less observations are excluded. The default is 5.

study_data data.frame the data frame that contains the measurements

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

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

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

## End(Not run)
```
Function to 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: $1.5 \times IQR$ from the 1st ($Q_{25}$) or 3rd ($Q_{75}$) quartile.
- the $6 \times \sigma$ approach, i.e. any measurement of the Mahalanobis distance not in the interval of $ar{x} \pm 3 \times \sigma$ is considered an outlier.
- the approach from Hubert for skewed distributions which is embedded in the R package robustbase
- a completely heuristic approach named $\sigma$-gap.

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

Usage

```r
acc_multivariate_outlier(
  resp_vars,
  id_vars = NULL,
  label_col,
  n_rules = 4,
  study_data,
  meta_data
)
```

Arguments

- `resp_vars` variable list the name of the continuous measurement variables
- `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
- `study_data` data.frame the data frame that contains the measurements
- `meta_data` data.frame the data frame that contains metadata attributes of study data
Value

a list with:

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

ALGORITHM OF THIS IMPLEMENTATION:

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

See Also

Online Documentation

Description

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

A somewhat more conservative approach in terms of symmetric and/or normal distributions is the $6 \sigma$ approach, i.e. any measurement not in the interval of $\mu \pm 3 \sigma$ 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 $1 \times \sigma$ has been been chosen.

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

**Usage**

```r
acc_robust_univariate_outlier(
  resp_vars = NULL,
  label_col,
  study_data,
  meta_data,
  exclude_roles,
  n_rules = 4,
  max_non_outliers_plot = 10000
)
```

**Arguments**

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

**Details**

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

a list with:

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

ALGORITHM OF THIS IMPLEMENTATION:

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

See Also

`acc_univariate_outlier`

Description

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

Usage

```r
acc_shape_or_scale(
  resp_vars,
  dist_col,
  guess,
  par1,
  par2,
  end_digits,
  label_col,
  study_data,
  meta_data
)
```
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

Value

a list with:

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

ALGORITHM OF THIS IMPLEMENTATION:

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

See Also

Online Documentation
Function to identify univariate outliers by four different approaches

Description

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

A somewhat more conservative approach in terms of symmetric and/or normal distributions is the $6 \times \sigma$ approach, i.e. any measurement not in the interval of $\text{mean}(x) \pm 3 \times \sigma$ 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 $1 \times \sigma$ has been chosen.

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

Usage

```r
acc_univariate_outlier(
  resp_vars = NULL,
  label_col,
  study_data,
  meta_data,
  exclude_roles,
  n_rules = 4,
  max_non_outliers_plot = 10000
)
```
Arguments

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

Details

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

Value

A list with:

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

**ALGORITHM OF THIS IMPLEMENTATION:**

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

See Also

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

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

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

### Usage

```r
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 less 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 less subgroups it is not used for analysis. The default is 5.

- **label_col**  
  variable attribute the name of the column in the metadata with labels of variables
threshold_value numeric from=0 to=1. a numerical value ranging from 0-1
study_data data.frame the data frame that contains the measurements
meta_data data.frame the data frame that contains metadata attributes of study data

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

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

See Also
Online Documentation

Examples
```r
## 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 = KEY_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
)
```
## as.data.frame.dataquieR_resultset

Convert a full dataquieR report to a data.frame

### Description

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

### Usage

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

### Arguments

- **x**: dataquieR report
- **...**: not used

### Value

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

## as.list.dataquieR_resultset

Convert a full dataquieR report to a list

### Description

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

### Usage

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

### Arguments

- **x**: dataquieR report
- **...**: arguments passed to `pipeline_recursive_result`
com_item_missingness

Value

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

---

com_item_missingness  Summarize missingness columnwise (in variable)

Description

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

Usage

com_item_missingness(
    study_data,
    meta_data,
    resp_vars = NULL,
    label_col,
    show_causes = TRUE,
    cause_label_df,
    include_sysmiss = NULL,
    threshold_value,
    suppressWarnings = 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
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 must be specified here
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 mixed missing and jump code lists
Value

a list with:

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

ALGORITHM OF THIS IMPLEMENTATION:

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

See Also

Online Documentation

---

**com_segment_missingness**

*Summarizes missingness for individuals in specific segments*

Description

**This implementation can be applied in two use cases:**

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

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

Use case (2) assumes a more complex structure of study data and meta data. The study data comprise so-called intro-variables (either TRUE/FALSE or codes for non-participation). The column KEY_STUDY_SEGMENT 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.
com_segment_missingness

Usage

com_segment_missingness(
  study_data,
  meta_data,
  group_vars = NULL,
  strata_vars = NULL,
  label_col,
  threshold_value,
  direction,
  exclude_roles = "process"
)

Arguments

study_data  data.frame  the data frame that contains the measurements
meta_data   data.frame  the data frame that contains metadata attributes of study data
group_vars  variable    the name of a variable used for grouping, defaults to NULL for not
             grouping output
strata_vars variable    the name of a variable used for stratification, defaults to NULL for not
             grouping output
label_col   variable    attribute the name of the column in the metadata with labels of variables
threshold_value numeric  from=0 to=100. a numerical value ranging from 0-100
direction   enum        low | high. "high" or "low", i.e. are deviations above/below the threshold
critical
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 high
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 low than all values above the threshold_value are normal (displayed in dark blue,
GRADING = 0).

Hint:
This function does not support a resp_vars argument but exclude_roles to specify variables
not relevant for detecting a missing segment.
List function.
Value

a list with:

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

See Also

Online Documentation

---

com_unit_missingness  Counts all individuals with no measurements at all

Description

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

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

Usage

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

Arguments

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

This implementation 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 from 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 otherwise 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

Description

Detect abnormalities help functions

Usage

```r
contradiction_functions
```

Format

An object of class `list` of length 11.
Details

2 variables:

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

contradiction_functions_descriptions
description of the contradiction functions

Description

description of the contradiction functions

Usage

contradiction_functions_descriptions

Format

An object of class list of length 11.
Description

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

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

Usage

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

Arguments

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

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

```r
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"
```
Description

**APPROACH:**

Inadmissible numerical values can be of type integer or float. This implementation requires the definition of intervals in the metadata to examine the admissibility of numerical study data. This helps identify inadmissible measurements according to hard limits (for multiple variables).

Usage

```r
con_detection_limits

con_detection_limits

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_detection_limits

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

Details

**ALGORITHM OF THIS IMPLEMENTATION::**

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

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

Value

a list with:

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

See Also

- `con_limit_deviations`
- Online Documentation
Examples

```r
load(system.file("extdata", "study_data.RData", package = "dataquieR"))
load(system.file("extdata", "meta_data.RData", package = "dataquieR"))

# make things a bit more complicated for the function, giving datetimes
# as numeric
study_data[, vapply(study_data, inherits, "POSIXct", FUN.VALUE = logical(1))] <-
lapply(study_data[, vapply(study_data, inherits, "POSIXct", FUN.VALUE = logical(1))], as.numeric)

MyValueLimits <- con_limit_deviations(
  resp_vars = NULL,
  label_col = "LABEL",
  study_data = study_data,
  meta_data = meta_data,
  limits = "HARD_LIMITS"
)

names(MyValueLimits$SummaryPlotList)

MyValueLimits <- con_limit_deviations(
  resp_vars = c("QUEST_DT_0"),
  label_col = "LABEL",
  study_data = study_data,
  meta_data = meta_data,
  limits = "HARD_LIMITS"
)

MyValueLimits$SummaryPlotList$QUEST_DT_0
```

---

**con_inadmissible_categorical**

_Detects variable levels not specified in metadata_

**Description**

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

**Usage**

```r
con_inadmissible_categorical(
  resp_vars = NULL,
  study_data,
  meta_data,
  label_col,
  threshold = NULL
)
```
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**: numeric from=0 to=100. a numerical value ranging from 0-100. Not yet implemented.

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:

- **SummaryTable**: 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
  - GRADING: indicator TRUE/FALSE if inadmissible categorical values were observed
- **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**

**APPROACH:**
Inadmissible numerical values can be of type integer or float. This implementation requires the definition of intervals in the metadata to examine the admissibility of numerical study data. This helps identify inadmissible measurements according to hard limits (for multiple variables).

**Usage**

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

**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

**Details**

**ALGORITHM OF THIS IMPLEMENTATION::**

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

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

a list with:

• FlaggedStudyData data.frame related to the study data by a 1:1 relationship, i.e. for each observation is checked whether the value is below or above the limits.

• SummaryTable data.frame summarizes limit deviations for each variable.

• SummaryPlotList list of ggplots The plots for each variable are either a histogram (continuous) or a barplot (discrete).

• ModifiedStudyData data.frame If the function identifies limit deviations, the respective values are removed in ModifiedStudyData.

• ReportSummaryTable: heatmap-like data frame about limit violations

See Also

• con_detection_limits

• Online Documentation

Examples

load(system.file("extdata", "study_data.RData", package = "dataquieR"))
load(system.file("extdata", "meta_data.RData", package = "dataquieR"))

# make things a bit more complicated for the function, giving datetimes as numeric
study_data[, vapply(study_data, inherits, "POSIXct", FUN.VALUE = logical(1))] <- lapply(study_data[, vapply(study_data, inherits, "POSIXct", FUN.VALUE = logical(1))], as.numeric)

MyValueLimits <- con_limit_deviations(
    resp_vars = NULL,
    label_col = "LABEL",
    study_data = study_data,
    meta_data = meta_data,
    limits = "HARD_LIMITS"
)

names(MyValueLimits$SummaryPlotList)

MyValueLimits <- con_limit_deviations(
    resp_vars = c("QUEST_DT_0"),
    label_col = "LABEL",
    study_data = study_data,
    meta_data = meta_data,
    limits = "HARD_LIMITS"
)

MyValueLimits$SummaryPlotList$QUEST_DT_0
The dataquieR package about Data Quality in Epidemiological Research

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

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

stops on errors

Usage

dataquieR_resultset_verify(list_to_verify)

Arguments

list_to_verify object to be checked

Value

invisible(TRUE) – stops on errors.

DATA_TYPES

Data Types

Description

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

Usage

DATA_TYPES

Format

An object of class list of length 4.

Details

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

Data Types of Function Arguments:
As function arguments, dataquieR uses additional type specifications:
• numeric is a numerical value (float or integer), but it is not an allowed data type in the metadata. However, some functions may accept float or integer for specific function arguments. This is, where we use the term numeric.
• enum allows one element out of a set of allowed options similar to match.arg
• 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 meta data attribute given in that argument. Labels can easily be translated using prep_map_labels
• variable list Function arguments of this type expect a character vector that specifies variables using the variable identifiers given in the meta data attribute VAR_NAMES or, if label_col is set, given in the meta data attribute given in that argument. Labels can easily be translated using prep_map_labels

See Also

integer string

---

**DATA_TYPES_OF_R_TYPE**

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

**Description**

All available data types, mapped from their respective R types

**Usage**

DATA_TYPES_OF_R_TYPE

**Format**

An object of class list of length 13.

<table>
<thead>
<tr>
<th>dimensions</th>
<th>Names of DQ dimensions</th>
</tr>
</thead>
</table>

**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

---

**DISTRIBUTIONS**

All available probability distributions for `acc_shape_or_scale`

---

**Description**

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

**Usage**

DISTRIBUTIONS

**Format**

An object of class `list` of length 3.

---

**dq_report**

Generate a full DQ report

---

**Description**

Generate a full DQ report
dq_report

Usage
dq_report(
  study_data,
  meta_data,
  label_col = NULL,
  ..., 
  dimensions = c("Completeness", "Consistency"),
  strata_attribute,
  strata_vars,
  cores = list(mode = "socket", logging = FALSE, cpus = util_detect_cores(),
                load.balancing = TRUE),
  specific_args = 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
...: 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.
strata_attribute: character variable of a variable attribute coding study segments. Values other than leaving this empty or passing KEY_STUDY_SEGMENT are not yet supported. Stratification is not yet fully supported, please use dq_report_by.
strata_vars: character name of variables to stratify the report on, such as "study_center". Not yet supported, please use dq_report_by.
cores: integer number of cpu cores to use or a named list with arguments for parallelMap::parallelStart or NULL, if parallel has already been started by the caller.
specific_args: list named list of arguments specifically for one of the called functions, the of the list elements correspond to the indicator functions whose calls should be modified. The elements are lists of arguments.

Details

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

Value

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

- `as.data.frame.dataquieR_resultset`
- `as.list.dataquieR_resultset`
- `print.dataquieR_resultset`
- `summary.dataquieR_resultset`
- `dq_report_by`

Examples

```r
## Not run: # really long-running example.
load(system.file("extdata", "study.data.RData", package = "dataquieR"))
load(system.file("extdata", "meta.data.RData", package = "dataquieR"))
report <- dq_report(study_data, meta_data, label_col = LABEL) # most easy use
report <- dq_report(study_data, meta_data,
  label_col = LABEL, dimensions =
  c("Completeness", "Consistency", "Accuracy"),
  check_table = read.csv(system.file("extdata",
    "contradiction_checks.csv", package = "dataquieR"
  ), header = TRUE, sep = "#"),
  show_causes = TRUE,
  cause_label_df = read.csv(
    system.file("extdata", "Missing-Codes-2020.csv", package = "dataquieR"),
    header = TRUE, sep = ";"
  )
)
save(report, file = "report.RData") # careful, this contains the study_data
report <- dq_report(study_data, meta_data,
  label_col = LABEL,
  check_table = read.csv(system.file("extdata",
    "contradiction_checks.csv", package = "dataquieR"
  ), header = TRUE, sep = "#"),
  specific_args = list(acc_univariate_outlier = list(resp_vars = "K")),
  resp_vars = "SBP_0"
)
report <- dq_report(study_data, meta_data,
  label_col = LABEL,
  check_table = read.csv(system.file("extdata",
    "contradiction_checks.csv", package = "dataquieR"
  ), header = TRUE, sep = "#"),
  specific_args = list(acc_univariate_outlier = list(resp_vars = "DBP_0")),
  resp_vars = "SBP_0"
)
report <- dq_report(study_data, meta_data,
  label_col = LABEL,
  check_table = read.csv(system.file("extdata",
    "contradiction_checks.csv", package = "dataquieR"
  ), header = TRUE, sep = "#"),
```
dq_report_by

), header = TRUE, sep = "#"),
specific_args = list(acc_univariate_outlier = list(resp_vars = "DBP_0")),
resp_vars = "SBP_0", cores = NULL
)

## End(Not run)

dq_report_by Generate a stratified full DQ report

Description

Generate a stratified full DQ report

Usage

dq_report_by(
  study_data,
  meta_data,
  label_col,
  meta_data_split = KEY_STUDY_SEGMENT,
  study_data_split,
  ...
)

Arguments

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

See Also

dq_report
Examples

```r
## Not run: # really long-running example.
load(system.file("extdata", "study_data.RData", package = "dataquieR"))
load(system.file("extdata", "meta_data.RData", package = "dataquieR"))
rep <- dq_report_by(study_data, meta_data, label_col =
  LABEL, study_data_split = "CENTER_0")
rep <- dq_report_by(study_data, meta_data,
  label_col = LABEL, study_data_split = "CENTER_0",
  meta_data_split = NULL
)

## End(Not run)
```

---

**html_dependency_vert_dt**

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

**Description**

HTML Dependency for vertical headers in DT::datatable

**Usage**

```r
html_dependency_vert_dt()
```

**Value**

the dependency

---

**int_datatype_matrix**

*Function to check declared data types of metadata in study data*

**Description**

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

**Usage**

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

**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.

**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**

```r
load(system.file("extdata/meta_data.RData", package = "dataquieR"), envir = environment())
load(system.file("extdata/study_data.RData", package = "dataquieR"), envir = environment())
appmatrix <- int_datatype_matrix(study_data = study_data, meta_data = meta_data, label_col = LABEL)
```
pipeline_recursive_result

Function to convert a pipeline result data frame to named encapsulated lists

Description

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

Usage

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

Arguments

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

Details

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

Value

a list with:

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

Examples

call_plan_with_results <- structure(list(
  resp_vars =
  c("SBP_0", "DBP_0", "VO2_CAPCAT_0", "BSG_0"),
  group_vars = c("USR_BP_0", "USR_BP_0", "USR_VO2_0", "USR_BP_0"),
  co_vars = list("SEX_0", "SEX_0", "SEX_0", "SEX_0")
),
pipeline_vectorized

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

Description

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

Usage

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

Arguments

fct function function to call
resp_vars variable list the name of the measurement variables, if NULL (default), all variables are used.
study_data data.frame the data frame that contains the measurements
meta_data data.frame the data frame that contains metadata attributes of study data
label_col variable attribute the name of the column in the metadata with labels of variables
... additional arguments for the function

key_var_names character character vector named by arguments to be filled by meta data KEY_-
entries as follows: c(group_vars = KEY_OBSERVER) – may be missing, then all possible combinations will be analyzed. Cannot contain resp_vars.

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

variable_roles list restrict each function argument (referred to by its name matching a name in names(variable_roles)) to variables of the role given here.

result_groups character columns to group results to encapsulated lists or NULL receive a data frame with all call arguments and their respective results in a column 'result' – see pipeline_recursive_result

use_cache logical set to FALSE to omit re-using already distributed study- and metadata on a parallel cluster

compute_plan_only logical set to TRUE to omit computations and return only the compute plan filled with planned evaluations. used in pipelines.

Details

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

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

Value

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

Examples

```r
## Not run: # really long-running example
load(system.file("extdata/study_data.RData", package = "dataquieR"))
load(system.file("extdata/meta_data.RData", package = "dataquieR"))
a <- pipeline_vectorized(
```

\begin{verbatim}

pipeline_vectorized

    fct = accMargins, study_data = study_data,
    meta_data = meta_data, label_col = LABEL,
    key_var_names = c(group_vars = KEY_OBSERVER)
)
b <- pipeline_vectorized(
    fct = accMargins, study_data = study_data,
    meta_data = meta_data, label_col = LABEL
)

b_adj <-
    pipeline_vectorized(
        fct = accMargins, study_data = study_data,
        meta_data = meta_data, label_col = LABEL, co_vars = c("SEX_1", "AGE_1")
    )
c <- pipeline_vectorized(
    fct = acc_loess, study_data = study_data,
    meta_data = meta_data, label_col = LABEL,
    variable_roles = list(
        resp_vars = list(VARIABLE_ROLES$PRIMARY),
        group_vars = VARIABLE_ROLES$PROCESS
    )
)

fct = acc_loess, study_data = study_data,
meta_data = meta_data, label_col = LABEL,
variable_roles = list(
    resp_vars = list(VARIABLE_ROLES$PRIMARY, VARIABLE_ROLES$SECONDARY),
    group_vars = VARIABLE_ROLES$PROCESS
)
)

f <- pipeline_vectorized(
    fct = accMargins, study_data = study_data,
    meta_data = meta_data, label_col = LABEL,
    key_var_names = c(group_vars = KEY_OBSERVER, co_vars = "SEX_0"
)
)

f <- pipeline_vectorized(
    fct = accMargins, study_data = study_data,
    meta_data = meta_data, label_col = LABEL,
    key_var_names = c(group_vars = KEY_OBSERVER), co_vars = "SEX_0",
    result_groups = NULL
)

pipeline_recursive_result(f)
g <- pipeline_vectorized(
    fct = accMargins, study_data = study_data,
    meta_data = meta_data, label_col = LABEL,
    key_var_names = c(group_vars = KEY_OBSERVER), co_vars = "SEX_0",
    result_groups = c("co_vars")
)
g1 <- pipeline_vectorized(
    fct = accMargins, study_data = study_data,
    meta_data = meta_data, label_col = LABEL,
    key_var_names = c(group_vars = KEY_OBSERVER), co_vars = "SEX_0",
    result_groups = c("group_vars")
)

\end{verbatim}
g2 <- pipeline_vectorized(
  fct = acc_margins, study_data = study_data,
  meta_data = meta_data, label_col = LABEL,
  key_var_names = c(group_vars = KEY_OBSERVER), co_vars = "SEX_0",
  result_groups = c("group_vars", "co_vars")
)
g3 <- pipeline_vectorized(
  fct = acc_margins, study_data = study_data,
  meta_data = meta_data, label_col = LABEL,
  key_var_names = c(group_vars = KEY_OBSERVER), co_vars = "SEX_0",
  result_groups = c("co_vars", "group_vars")
)
g4 <- pipeline_vectorized(
  fct = acc_margins, study_data = study_data,
  meta_data = meta_data, label_col = LABEL,
  co_vars = "SEX_0", result_groups = c("co_vars")
)
meta_datax <- meta_data
meta_datax[9, "KEY_DEVICE"] <- "v00011"
g5 <- pipeline_vectorized(
  fct = acc_margins, study_data = study_data,
  meta_data = meta_datax, label_col = LABEL,
  co_vars = "SEX_0", result_groups = c("co_vars")
)
g6 <- pipeline_vectorized(
  fct = acc_margins, study_data = study_data,
  meta_data = meta_datax, label_col = LABEL,
  co_vars = "SEX_0", result_groups = c("co_vars", "group_vars")
)
## End(Not run)

---

prep_add_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, ...)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAR_NAMES</td>
<td>character</td>
<td>Names of the Variables to add</td>
</tr>
<tr>
<td>DATA_TYPE</td>
<td>character</td>
<td>Data type for the added variables</td>
</tr>
<tr>
<td>LABEL</td>
<td>character</td>
<td>Labels for these variables</td>
</tr>
</tbody>
</table>
prep_check_meta_names

VALUE_LABELS character Value labels for the values of the variables as usually pipe separated and assigned with =: 1 = male | 2 = female
meta_data data.frame the metadata to extend
Further defined variable attributes, see prep_create_meta

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

Value
a data frame with amended meta data.

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

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

Usage
prep_check_meta_names(meta_data, 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)
character.only logical a logical indicating whether level can be assumed to be character strings.

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

Value
a logical with:
• invisible(TRUE). In case of problems with the meta data, a condition is raised (stop()).
Examples

```r
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",
  KEY_OBSERVER = "KEY_OBSERVER", KEY_DEVICE = "KEY_DEVICE",
  KEY_DATE_TIME = "KEY_DATE_TIME",
  KEY_STUDY_SEGMENT = "KEY_STUDY_SEGMENT"
),
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",
  KEY_OBSERVER = "KEY_OBSERVER", KEYDEVICE = "KEY_DEVICE",
  KEY_DATE_TIME = "KEY_DATE_TIME", KEY_STUDY_SEGMENT =
  "KEY_STUDY_SEGMENT",
  DETECTION_LIMITS = "DETECTION_LIMITS", SOFT_LIMITS = "SOFT_LIMITS",
  CONTRADICTIONS = "CONTRADICTIONS", DISTRIBUTION = "DISTRIBUTION",
  DECIMALS = "DECIMALS", VARIABLE_ROLE = "VARIABLE_ROLE",
  DATA_ENTRY_TYPE = "DATA_ENTRY_TYPE",
  VARIABLE_ORDER = "VARIABLE_ORDER", LONG_LABEL =
  "LONG_LABEL", recode = "recode"
),
OPTIONAL)
)

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

---

**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**

```r
prep_clean_labels(label_col, meta_data, no_dups = FALSE)
```
Arguments

- **label_col**
  - character: label attribute to adjust or character vector to adjust, depending on `meta_data` argument is given or missing.

- **meta_data**
  - data.frame: meta data frame: If `label_col` is a label attribute to adjust, this is the meta data table to process on. If missing, `label_col` must be a character vector with values to adjust.

- **no_dups**
  - logical: disallow duplicates in input or output vectors of the function, then, `prep_clean_labels` would call `stop()` on duplicated labels.

Details

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

A warning is emitted, if modifications were necessary.

Value

A data.frame with:

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

Examples

```r
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)
```
prep_create_meta  Support function to create data.frames of metadata

Description

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

Usage

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

Arguments

... named column vectors, names will be mapped using WELL_KNOWN_META_VARIABLE_NAMES, if included in WELL_KNOWN_META_VARIABLE_NAMES can also be a data frame, then its column names will be mapped using WELL_KNOWN_META_VARIABLE_NAMES

stringsAsFactors

logical if the argument is a list of vectors, a data frame will be created. In this case, stringsAsFactors controls, whether characters will be auto-converted to Factors, which defaults here always to false independent from the default.stringsAsFactors.

level

enum level of requirement (see also VARATT_REQUIRE_LEVELS) set to NULL, if not a complete metadata frame is created.

character.only  logical a logical indicating whether level can be assumed to be character strings.

Details

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

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

Value

a data frame with:

• meta data attribute names mapped and

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

See Also

WELL_KNOWN_META_VARIABLE_NAMES
**prep_datatype_from_data**

*Get data types from data*

**Description**

Get data types from data

**Usage**

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

**Arguments**

- **resp_vars** variable names of the variables to fetch the data type from the data
- **study_data** data.frame the data frame that contains the measurements

**Value**

vector of data types

**Examples**

```
dataquieR::prep_datatype_from_data(cars)
```

---

**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 = NULL, to = LABEL, from = VAR_NAMES, ifnotfound)
```

**Arguments**

- **x** character variable names, character vector, see parameter from
- **meta_data** data.frame meta data frame
- **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.
Details

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

The function not only maps to the `LABEL` column, but 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

```r
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")))
```

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

**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 function 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

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_missings)

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_missings**: replace missing codes, defaults to TRUE

Details

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

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

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

Value

ds1 the study data with mapped column names

See Also

accMargins
Examples

```r
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))

prep_study2meta

---

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

**Description**

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

**Usage**

```r
prep_study2meta(
  study_data,
  level = c(VARATT_REQUIRE_LEVELS$REQUIRED, VARATT_REQUIRE_LEVELS$OPTIONAL),
  convert_factors = FALSE
)
```

**Arguments**

- **study_data**  
  *data.frame* the data frame that contains the measurements
- **level**  
  *enum* level to provide (see also `VARATT_REQUIRE_LEVELS`)
- **convert_factors**  
  *logical* convert the

**Details**

```r
dataquieR::util_get_var_att_names_of_level(VARATT_REQUIRE_LEVELS$REQUIRED)
```

```r
## VAR_NAMES DATA_TYPE MISSING_LIST
## "VAR_NAMES" "DATA_TYPE" "MISSING_LIST"
```

The function also tries to detect missing codes.

**Value**

*a meta_data data frame*
### prep_valuelabels_from_data

*Get value labels from data*

**Description**

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

**Usage**

```r
df <- data.frame(Var1 = c(1, 2, 3), Var2 = c('a', 'b', 'c'))
pred_valuelabels_from_data(resp_vars = colnames(df), df)
```

**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**

```r
dataquieR::prep_datatype_from_data(iris)
```

### print.dataquieR_result

*Print a dataquieR result returned by pipeline_vectorized*

**Description**

Print a dataquieR result returned by pipeline_vectorized

**Usage**

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

**Arguments**

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

see print

Examples

load(system.file("extdata", "study_data.RData", package = "dataquieR"))
load(system.file("extdata", "meta_data.RData", package = "dataquieR"))
result <- pipeline_vectorized(acc_margins, cores = list(mode = "local"),
    resp_vars = "SBP_0", group_vars = "USR_BP_0",
    study_data = study_data, meta_data = meta_data, label_col = LABEL
)
single_result <- result$'group_vars' = USR_BP_0'$'resp_vars' = SBP_0'
print(single_result, slot = "SummaryPlot")

print.dataquieR_resultset

Generate a RMarkdown-based report from a dataquieR report

Description

Generate a RMarkdown-based report from a dataquieR report

Usage

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

Arguments

x  dataquieR report.

... additional arguments:

• template: Report template to use, not yet supported.
• chunk_error: display error messages in report
• chunk_warning: display warnings in report
• output_format: output format to use, see rmarkdown::render – currently, html based formats are supported by the default template.
• chunk_echo: display R code in report
• chunk_message: display message outputs in report

Value

file name of the generated report
print.ReportSummaryTable

print implementation for the class ReportSummaryTable

Description

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

Usage

```r
## S3 method for class 'ReportSummaryTable'
print(
  x,
  relative,
  dt = FALSE,
  fillContainer = FALSE,
  displayValues = FALSE,
  ...
)
```

Arguments

- `x` an object used to select a method.
- `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
- `...` further arguments passed to or from other methods.

Details

The default method, `print.default` has its own help page. Use `methods("print")` to get all the methods for the print generic.

`print.factor` allows some customization and is used for printing ordered factors as well.

`print.table` for printing tables allows other customization. As of R 3.0.0, it only prints a description in case of a table with 0-extents (this can happen if a classifier has no valid data).

See `noquote` as an example of a class whose main purpose is a specific print method.

Value

the printed object

References

See Also

The default method `print.default`, and help for the methods above; further options, `noquote`. For more customizable (but cumbersome) printing, see `cat`, `format` or also `write`. For a simple prototypical print method, see `.print.via.format` in package `tools`.

Examples

```r
require(stats)

ts(1:20)  #-- print is the "Default function" --> print.ts(.) is called
for(i in 1:3) print(1:i)

## Printing of factors
attenu$station ## 117 levels -> 'max.levels' depending on width

## ordered factors: levels "l1 < l2 < .."
esoph$agegp[1:12]
esoph$alcgp[1:12]

## Printing of sparse (contingency) tables
set.seed(521)
t1 <- round(abs(rt(200, df = 1.8)))
t2 <- round(abs(rt(200, df = 1.4)))
table(t1, t2)  # simple
print(table(t1, t2), zero.print = ".")  # nicer to read

## same for non-integer "table":
T <- table(t2,t1)
T <- T * (1+round(rlnorm(length(T)))/4)
print(T, zero.print = ".")  # quite nicer,
print.table(T[,2:8] * 1e9, digits=3, zero.print = ".")

## Corner cases with empty extents:
table(1, NA)  # < table of extent 1 x 0 >
```

---

**pro_applicability_matrix**

*Function to check applicability of DQ functions on study data*

**Description**

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

**Usage**

```r
pro_applicability_matrix(
    study_data,
    meta_data,
```
split_segments = FALSE,
label_col,
max_vars_per_plot = 20
)

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.

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

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)
**Description**

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

**Usage**

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

**Arguments**

`...` ReportSummaryTable objects to combine.

**See Also**

`base::rbind.data.frame`

---

**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.
summary.dataquieR_resultset

Summarize a dataquieR report

Description

Summarizes a dataquieR report extracting all GRADING results.

Usage

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

Arguments

- `object`  
  dataquieR report.

- `...`  
  not used yet.

Value

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

Examples

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

## End(Not run)
```
util_anytime_installed

Test, if package anytime is installed

Description
Test, if package anytime is installed

Usage
util_anytime_installed()

Value
TRUE if anytime is installed.

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

util_app_cd
utility function for the applicability of contradiction checks

Description
Test for applicability of contradiction checks

Usage
util_app_cd(x, dta)

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

Value
factor 0-3 for each variable in metadata
- 0 data type mismatch and not applicable
- 1 data type mismatches but applicable
- 2 data type matches but not applicable
- 3 data type matches and applicable
util_app_dc  

utility function for the applicability of distribution plots

Description

Test for applicability of distribution plots

Usage

util_app_dc(x, dta)

Arguments

x  
data.frame metadata

dta  
logical vector, 1=matching data type, 0 = non-matching data type

Value

factor 0-3 for each variable in metadata

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

util_app_dl  

utility function to test for applicability of detection limits checks

Description

Test for applicability of detection limits checks

Usage

util_app_dl(x, dta)

Arguments

x  
data.frame metadata

dta  
logical vector, 1=matching data type, 0 = non-matching data type
Value

factor 0-3 for each variable in metadata

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

util_app_ed

utility function for the applicability of end digits preferences checks

Description

Test for applicability of end digits preferences checks

Usage

util_app_ed(x, dta)

Arguments

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

Value

factor 0-3 for each variable in metadata

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

utility function for the applicability of categorical admissibility

Description
Test for applicability of categorical admissibility

Usage
util_app_iac(x, dta)

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

Value
factor 0-3 for each variable in metadata
- 0 data type mismatch and not applicable
- 1 data type mismatches but applicable
- 2 data type matches but not applicable
- 3 data type matches and applicable

util_app_iav
utility function for the applicability of numeric admissibility

Description
Test for applicability of numeric admissibility

Usage
util_app_iav(x, dta)

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

factor 0-3 for each variable in metadata

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

Description

Test for applicability of item missingness

Usage

util_app_im(x, dta)

Arguments

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

Value

factor 0-3 for each variable in metadata

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

utility function for applicability of LOESS smoothed time course plots

Description
Test for applicability of LOESS smoothed time course plots

Usage
util_app_loess(x, dta)

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

Value
factor 0-3 for each variable in metadata
  • 0 data type mismatch and not applicable
  • 1 data type mismatches but applicable
  • 2 data type matches but not applicable
  • 3 data type matches and applicable

util_app_mar

utility function to test for applicability of marginal means plots

Description
Test for applicability of detection limits checks

Usage
util_app_mar(x, dta)

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

Value

factor 0-3 for each variable in metadata

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

util_app_mol  utility function applicability of multivariate outlier detection

Description

Test for applicability of multivariate outlier detection

Usage

util_app_mol(x, dta)

Arguments

x          data.frame metadata

dta        logical vector, 1=matching data type, 0 = non-matching data type

Value

factor 0-3 for each variable in metadata

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

*utility function for the applicability of outlier detection*

**Description**

Test for applicability of univariate outlier detection

**Usage**

```r
test_app_ol(x, dta)
```

**Arguments**

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

**Value**

*factor* 0-3 for each variable in metadata

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

**util_app_sm**

*utility function applicability of segment missingness*

**Description**

Test for applicability of segment missingness

**Usage**

```r
test_app_sm(x, dta)
```

**Arguments**

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

factor 0-3 for each variable in metadata

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

Description

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

Usage

util_app_sos(x, dta)

Arguments

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

Value

factor 0-3 for each variable in metadata

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

utility applicability variance components

Description

Test for applicability of ICC

Usage

util_app_vc(x, dta)

Arguments

x  
data.frame metadata

dta  
logical vector, 1=matching data type, 0 = non-matching data type

Value

factor 0-3 for each variable in metadata

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

util_assign_levlabs

utility function to assign labels to levels

Description

function to assign labels to levels of a variable

Usage

util_assign_levlabs(
    variable,
    string_of_levlabs,
    splitchar,
    assignchar,
    ordered = TRUE
)
convert factors to label-corresponding numeric values

util_as_numeric

Description

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

Usage

util_as_numeric(v, warn)

Arguments

v the vector

warn if not missing: character with error message stating conversion error

Value

the converted vector
util_backtickQuote

utility function to set string in backticks

Description
Quote a set of variable names with backticks

Usage
util_backtickQuote(x)

Arguments
x variable names

Value
quoted variable names

util_check_data_type Support function to verify the data type of a value

Description
Function to verify the data type of a value.

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

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

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

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

util_check_one_unique_value

Description
utility function to identify variables with one value only.

Usage
util_check_one_unique_value(x)

Arguments
x vector with values

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

Arguments

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

Value

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

util_correct_variable_use

Check referred variables

Description

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

Usage

util_correct_variable_use(
  arg_name,
  allow_na,
  allow_more_than_one,
  allow_null,
  allow_all_obs_na,
  allow_any_obs_na,
  need_type,
  role = ""
)

util_correct_variable_use2(
  arg_name,
  allow_na,
util_correct_variable_use

```r
allow_more_than_one,
allow_null,
allow_all_obs_na,
allow_any_obs_na,
need_type,
role = arg_name
```

**Arguments**

arg_name character Name of a function argument of the caller of `util_correct_variable_use`

allow_na logical default = FALSE. allow NAs in the variable names argument given in `arg_name`

allow_more_than_one logical default = FALSE. allow more than one variable names in `arg_name`

allow_null logical default = FALSE. allow an empty variable name vector in the argument `arg_name`

allow_all_obs_na logical default = TRUE. check observations for not being all NA

allow_any_obs_na logical default = TRUE. check observations for being complete without any NA

need_type character if not NA, variables must be of data type need_type according to the meta data, can be a pipe (|) separated list of allowed data types. Use `!` to exclude a type. See `DATA_TYPES` for the predefined variable types of the dataquieR concept.

role character variable-argument role. Set different defaults for all allow-arguments and need_type of this `util_correct_variable_use`. If given, it defines the intended use of the verified argument. For typical arguments and typical use cases, roles are predefined in `.variable_arg_roles`. The role’s defaults can be overwritten by the arguments. If role is "" (default), the standards are allow_na = FALSE, allow_more_than_one = FALSE, allow_null = FALSE, allow_all_obs_na = TRUE, allow_any_obs_na = TRUE, and need_type = NA.

Use `util_correct_variable_use2` for using the `arg_name` as default for role. See `.variable_arg_roles` for currently available variable-argument roles.

**Details**

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

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

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

This function accesses the caller’s environment!
util_count_codes

**See Also**

\.variable_arg_roles

---

**util_count_codes**

*count realizations of missing codes of any class*

**Description**

count total numbers of any sort of missing codes (MISSING or JUMP)

**Usage**

util_count_codes(sdf, mdf, variables, list, name, warn = TRUE)

**Arguments**

- **sdf** study data
- **mdf** meta data
- **variables** variables
- **list** variable attribute JUMP_LIST or MISSING_LIST: Count which categories.
- **name** variable attribute the name of the column in the metadata with labels of variables
- **warn** logical emit warnings on non-numeric missing codes

**Value**

a vector with the total number of missings of the class referred by list per variables.

---

**util_count_code_classes**

*count distinct realizations of missing codes of a specific class*

**Description**

count numbers of distinct codes of class missings or of class jump jumps

**Usage**

util_count_code_classes(sdf, mdf, variables, name, list, warn = FALSE)
### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sdf</td>
<td>study data</td>
</tr>
<tr>
<td>mdf</td>
<td>meta data</td>
</tr>
<tr>
<td>variables</td>
<td>meta data variables</td>
</tr>
<tr>
<td>name</td>
<td>the name of the column in the metadata with labels of variables</td>
</tr>
<tr>
<td>list</td>
<td>variable attribute JUMP_LIST or MISSING_LIST: Count which categories.</td>
</tr>
<tr>
<td>warn</td>
<td>logical emit warnings on non-numeric missing codes</td>
</tr>
</tbody>
</table>

### Value

A vector with the number of distinct realized missing codes of the missing class referred by list per variables.

---

### util_count_NA

**Support function to count number of NAs**

**Description**

Counts the number of NAs in x.

**Usage**

`util_count_NA(x)`

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>object to count NAs in</td>
</tr>
</tbody>
</table>

**Value**

Number of NAs

---

### util_detect_cores

**Detect cores**

**Description**

See `parallel::detectCores` for further details.

**Usage**

`util_detect_cores()`

**Value**

Number of available CPU cores.
util_dichotomize

utility function to dichotomize variables

Description
use the meta data attribute RECODE (=“recode”) to dichotomize the data

Usage
util_dichotomize(study_data, meta_data, label_col = VAR_NAMES)

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

util_dist_selection Utility function distribution-selection

Description
This function differentiates the type of measurement variables.

Usage
util_dist_selection(measurements, meta_data)

Arguments
measurements study data
meta_data meta data, not yet used

Value
data frame with one column for each variable in study data giving IsInteger, IsMultCat and IsNCategory
IsInteger contains a guess, if the variable contains integer values or is a factor
IsMultCat contains a guess, if the variable has more than two categories, if it is categorical or ordinal
NCategory contains the number of distinct values detected for the variable
**util_empty**

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

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

**Usage**
util_empty(x)

**Arguments**
- **x**: the vector to test

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

---

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

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

**Usage**
util_ensure_suggested(pkg, goal)

**Arguments**
- **pkg**: needed package
- **goal**: feature description for error message.
util_error

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

Description

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

Usage

util_error(m, ..., applicability_problem = NA)

Arguments

- **m**: error message or a simpleError
- **...**: arguments for sprintf on m, if m is a character
- **applicability_problem**: logical error indicates unsuitable resp_vars

Value

nothing, its purpose is to stop.

util_find_external_functions_in_stacktrace

Find externally called function in the stack trace

Description

intended use: error messages for the user

Usage

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

Arguments

- **sfs**: reverse sys.frames to search in
- **cls**: reverse sys.calls to search in

Value

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

*Find first externally called function in the stack trace*

**Description**

intended use: error messages for the user

**Usage**

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

**Arguments**

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

**Value**

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

---

**util_fix_rstudio_bugs**

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

**Description**

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

**Usage**

```r
util_fix_rstudio_bugs()
```

**Value**

invisible null
util_get_code_list

Fetch a missing code list from the metadata

Description

get missing codes from metadata (e.g. MISSING_LIST or JUMP_LIST)

Usage

util_get_code_list(
  x,
  code_name,
  split_char = SPLIT_CHAR,
  mdf,
  label_col = VAR_NAMES,
  warning_if_no_list = TRUE
)

Arguments

  x  variable the name of the variable to retrieve code lists for. only one variable at a
time is supported, not vectorized!!

  code_name  variable attribute JUMP_LIST or MISSING_LIST: Which codes to retrieve.

  split_char character len = 1. Character(s) used to separate different codes in the metadata,
usually |, as in 99999|99998|99997.

  mdf data.frame the data frame that contains metadata attributes of study data

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

  warning_if_no_list logical len = 1. If TRUE, a warning is displayed, if not missing codes are available
for a variable.

Value

  numeric vector of missing codes.

util_get_var_att_names_of_level

Get variable attributes of a certain provision level

Description

This function returns all variable attribute names of a certain meta data provision level or of more
than one level.
Usage

util_get_var_att_names_of_level(level)

Arguments

level level(s) of requirement

Value

all matching variable attribute names

util_heatmap_1th  Utility Function Heatmap with 1 Threshold

Description

Function to create heatmap-like plot given one threshold – works for percentages for now.

Usage

util_heatmap_1th(
    df,
    cat_vars,
    values,
    threshold,
    right_intv,
    invert,
    cols,
    strata
)

Arguments

df data.frame with data to display as a heatmap.
cat_vars variable list len=1-2. Variables to group by. Up to 2 group levels supported.
values variable the name of the percentage variable
threshold numeric lowest acceptable value
right_intv logical len=1. If FALSE (default), intervals used to define color ranges in the heatmap are closed on the left side, if TRUE on the right side, respectively.
invert logical len=1. If TRUE, high values are better, warning colors are used for low values. FALSE works vice versa.
cols deprecated, ignored.
strata variable optional, the name of a variable used for stratification
**util_hubert**

*utility function for the outliers rule of Huber et al.*

---

**Value**

- a list with:
  - `SummaryPlot`: `ggplot` object with the heatmap

---

**Description**

Function to calculate outliers according to the rule of Huber et al. This function requires the package `robustbase`

**Usage**

```r
util_hubert(x)
```

**Arguments**

- `x`: numeric data to check for outliers

**Value**

binary vector

---

**util_interpret_limits**

*Utility function to interpret mathematical interval notation*

---

**Description**

Utility function to split limit definitions into interpretable elements

**Usage**

```r
util_interpret_limits(mdata)
```

**Arguments**

- `mdata`: `data.frame` the data frame that contains metadata attributes of study data

**Value**

augments metadata by interpretable limit columns
util_is_integer  \hspace{1cm} Check for integer values

Description
This function checks if a variable is integer.

Usage
util_is_integer(x, tol = .Machine$double.eps^0.5)

Arguments
- \texttt{x} the object to test
- \texttt{tol} precision of the detection. Values deviating more than \texttt{tol} from their closest integer value will not be deemed integer.

Value
TRUE or FALSE

See Also
- \texttt{is.integer}
Copied from the documentation of \texttt{is.integer}
\texttt{is.integer} detects, if the storage mode of an R-object is integer. Usually, users want to know, if the values are integer. As suggested by \texttt{is.integer}'s documentation, \texttt{is.wholenumber} does so.

util_looks_like_missing  \hspace{1cm} Check for repetitive values using the digits 8 or 9 only

Description
Values not being finite (see \texttt{is.finite}) are also reported as missing codes.

Usage
util_looks_like_missing(x, n_rules = 1)

Arguments
- \texttt{x} numeric vector to test
- \texttt{n_rules} numeric Only outlying values can be missing codes; at least \texttt{n_rules} rules in \texttt{acc_univariate_outlier} match
**util_make_function**

Value

logical indicates for each value in x, if it looks like a missing code

See Also

acc_univariate_outlier

---

**util_make_function**

Make a function capturing errors and other conditions for parallelization

**Description**

Make a function capturing errors and other conditions for parallelization

**Usage**

util_make_function(fct)

**Arguments**

- **fct** function to prepare

**Value**

decorated function

---

**util_map_all**

Maps label column meta data on study data variable names

**Description**

Maps a certain label column from the meta data to the study data frame.

**Usage**

util_map_all(label_col = VAR_NAMES, study_data, meta_data)

**Arguments**

- **label_col** the variable of the metadata that contains the variable names of the study data
- **study_data** the name of the data frame that contains the measurements
- **meta_data** the name of the data frame that contains metadata attributes of study data

**Value**

list with slot df with a study data frame with mapped column names
util_map_labels  

Support function to allocate labels to variables

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

Usage
util_map_labels(x, meta_data = NULL, to = LABEL, from = VAR_NAMES, ifnotfound)

Arguments
x  character variable names, character vector, see parameter from
meta_data data.frame meta data frame
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.

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

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

Value
a character vector with:
• mapped values

Examples
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")))
**util_no_value_labels** Select really numeric variables

**Description**

Reduce `resp_vars` to those, which are either float or integer without `VALUE_LABELS`, i.e. likely numeric but not a factor

**Usage**

```r
util_no_value_labels(resp_vars, meta_data, label_col, warn = TRUE, stop = TRUE)
```

**Arguments**

- `resp_vars`: variable list len=1-2. the name of the continuous measurement variable
- `meta_data`: data.frame the data frame that contains metadata attributes of study data
- `label_col`: variable attribute the name of the column in the metadata with labels of variables
- `warn`: logical warn about removed variable names
- `stop`: logical stop on no matching `resp_var`

**Value**

character vector of matching `resp_vars`.

**util_observations_in_subgroups** Utility function observations in subgroups

**Description**

This function uses `!is.na` to count the number of non-missing observations in subgroups of the data (list) and in a set of user defined response variables. In some applications it is required that the number of observations per e.g. factor level is higher than a user-defined minimum number.

**Usage**

```r
util_observations_in_subgroups(x, rvs)
```

**Arguments**

- `x`: data frame
- `rvs`: variable names

**Value**

matrix of flags
util_only_NAs  identify NA-only variables

Description

This utility function identifies variables with NAs values only.

Usage

util_only_NAs(x)

Arguments

x  the variable to check a vector

Value

flagged binary vector

util_parse_assignments  Utility function to parse assignments

Description

This function parses labels & level assignments in the format 1 = male | 2 = female. The function also handles m = male | f = female, but this would not match the metadata concept. The split-character can be given, if not the default from SPLIT_CHAR is to be used, but this would also violate the metadata concept.

Usage

util_parse_assignments(text, split_char = SPLIT_CHAR)

Arguments

text  Text to be parsed
split_char  Character separating assignments

Value

the parsed assignments as a named list
util_par_pmap

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>util_par_pmap</td>
<td>Utility function parallel version of purrr::pmap</td>
</tr>
</tbody>
</table>

**Description**

Parallel version of purrr::pmap.

**Usage**

```r
util_par_pmap(
  .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 = list(mode = "socket", cpus = util_detect_cores(), logging = FALSE,
                load.balancing = TRUE),
  use_cache = FALSE
)
```

**Arguments**

- `.l`  
  *data.frame* with one call per line and one function argument per column
- `.f`  
  *function* to call with the arguments from `.l`
- `...`  
  Additional, static arguments for calling `.f`
- `cores`  
  Number of CPU cores to use or a (named) list with arguments for `parallelMap::parallelStart` or NULL, if parallel has already been started by the caller.
- `use_cache`  
  Logical set to FALSE to omit re-using already distributed study- and metadata on a parallel cluster

**Value**

List of results of the function calls

**Author(s)**

Aurèle
S Struckmann

**See Also**

purrr::pmap

Stack Overflow post
util_prepare_dataframes

Description

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

Usage

util_prepare_dataframes(.study_data, .meta_data, .label_col, .replace_missings)

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_missings replace missing codes, defaults to TRUE

Details

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

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

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

Value

ds1 the study data with mapped column names

See Also

accMargins
Examples

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))

util_replace_codes_by_NA

*Utility function to replace missing codes by NAs*

**Description**

Substitute all missing codes in a data.frame by NA.

**Usage**

`util_replace_codes_by_NA(study_data, meta_data, split_char = SPLIT_CHAR)`

**Arguments**

- **study_data**: Study data including jump/missing codes as specified in the code conventions
- **meta_data**: Metadata as specified in the code conventions
- **split_char**: Character separating missing codes

Codes are expected to be numeric.

util_set_dQuoteString

*Utility function to put strings in quotes*

**Description**

This function generates usual double-quotes for each element of the character vector.

**Usage**

`util_set_dQuoteString(string)`

**Arguments**

- **string**: Character vector

**Value**

quoted string
util_set_size

Attaches attributes about the recommended minimum absolute sizes to the plot p

### Description

Attaches attributes about the recommended minimum absolute sizes to the plot p

### Usage

```
util_set_size(p, width_em = NA_integer_, height_em = NA_integer_)
```

### Arguments

- **p** ggplot the plot
- **width_em** numeric len=1. the minimum width hint in em
- **height_em** numeric len=1. the minimum height in em

### Value

p the modified plot

util_set_sQuoteString Utility function single quote string

### Description

This function generates usual single-quotes for each element of the character vector.

### Usage

```
util_set_sQuoteString(string)
```

### Arguments

- **string** Character vector

### Value

quoted string
**util_sigmagap**  
*Utility function outliers according to the rule of Huber et al.*

**Description**

This function calculates outliers according to the rule of Huber et al.

**Usage**

```r
util_sigmagap(x)
```

**Arguments**

- `x`: numeric data to check for outliers

**Value**

binary vector

---

**util_sixsigma**  
*Utility function for six sigma deviations rule*

**Description**

This function calculates outliers according to the rule of six sigma deviations.

**Usage**

```r
util_sixsigma(x)
```

**Arguments**

- `x`: numeric data to check for outliers

**Value**

binary vector
**util_tukey**

Utility function Tukey outlier rule

### Description

This function calculates outliers according to the rule of Tukey.

### Usage

```r
util_tukey(x)
```

### Arguments

- **x** numeric data to check for outliers

### Value

binary vector

---

**util_validate_known_meta**

Utility function verifying syntax of known metadata columns

### Description

This function goes through metadata columns, dataquieR supports and verifies for these, that they follow its metadata conventions.

### Usage

```r
util_validate_known_meta(meta_data)
```

### Arguments

- **meta_data** data.frame the data frame that contains metadata attributes of study data

### Value

invisible(NULL)
**util_warning**  
*Produce a warning message with a useful short stack trace.*

**Description**

Produce a warning message with a useful short stack trace.

**Usage**

```r
util_warning(m, ..., applicability_problem = NA)
```

**Arguments**

- `m` : warning message or a `simpleWarning`
- `...` : arguments for `sprintf` on `m`, if `m` is a character
- `applicability_problem` : logical, warning indicates unsuitable `resp_vars`

**Value**

`invisible(NULL)`.

---

**util_warn_unordered**  
*Warn about a problem in varname, if x has no natural order*

**Description**

Also warns, if R does not have a comparison operator for `x`.

**Usage**

```r
util_warn_unordered(x, varname)
```

**Arguments**

- `x` : vector of data
- `varname` : character, len=1. Variable name for warning messages

**Value**

`invisible(NULL)`
VARATT_REQUIRE_LEVELS  

**Requirement levels of certain metadata columns**

**Description**

These levels are cumulatively used by the function `prep_create_meta` and related in the argument `level` therein.

**Usage**

VARATT_REQUIRE_LEVELS

**Format**

An object of class `list` of length 5.

**Details**

currently available:

- 'COMPATIBILITY' = "compatibility"
- 'REQUIRED' = "required"
- 'RECOMMENDED' = "recommended"
- 'OPTIONAL' = "optional"
- 'TECHNICAL' = "technical"

---

VARIABLE_ROLES  

**Variable roles can be one of the following:**

**Description**

- `intro` a variable holding consent-data
- `primary` a primary outcome variable
- `secondary` a secondary outcome variable
- `process` a variable describing the measurement process

**Usage**

VARIABLE_ROLES

**Format**

An object of class `list` of length 4.
WELL_KNOWN_META_VARIABLE_NAMES

Well-known metadata column names, names of metadata columns

Description

names of the variable attributes in the meta data frame holding the names of the respective observers, devices, lower limits for plausible values, upper limits for plausible values, lower limits for allowed values, upper limits for allowed values, the variable name (column name, e.g. v0020349) used in the study data, the variable name used for processing (readable name, e.g. RR_DIAST_1) and in parameters of the QA-Functions, the variable label, variable long label, variable short label, variable data type (see also DATA_TYPES), re-code for definition of lists of event categories, missing lists and jump lists as CSV strings.

Usage

WELL_KNOWN_META_VARIABLE_NAMES

Format

An object of class list of length 31.

Details

all entries of this list will be mapped to the package’s exported NAMESPACE environment directly, i.e. they are available directly by their names too:

- VAR_NAMES
- LABEL
- DATA_TYPE
- VALUE_LABELS
- MISSING_LIST
- JUMP_LIST
- HARD_LIMITS
- DETECTION_LIMITS
- SOFT_LIMITS
- CONTRADICTIONS
- DISTRIBUTION
- DECIMALS
- DATA_ENTRY_TYPE
- KEY_OBSERVER
- KEY_DEVICE
- KEY_DATETIME
- 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
- RECODE

Examples

print(VAR_NAMES)
Index

* accuracy
  accMargins, 8
* datasets
  .variable_arg_roles, 4
  contradiction_functions, 25
  contradiction_functions_descriptions, 26
  DATA_TYPES, 36
  DATA_TYPES_OF_R_TYPE, 37
  dimensions, 37
  DISTRIBUTIONS, 38
  SPLIT_CHAR, 64
  WELL_KNOWN_META_VARIABLE_NAMES, 104
  variable_arg_roles, 4, 80, 81
  acc_distributions, 5
  acc_end_digits, 6
  acc_loess, 7
  accMargins, 8
  acc_multivariate_outlier, 11
  acc_robust_univariate_outlier, 12, 17
  acc_shape_or_scale, 6, 14, 38
  acc_univariate_outlier, 14, 16, 90, 91
  acc_varcomp, 18
  as.data.frame.dataquieR_resultset, 20, 35, 40
  as.list.dataquieR_resultset, 20, 35, 40
  base::rbind.data.frame, 64
  cat, 62
  character, 39, 44, 46, 48, 49, 51, 53, 76, 80, 87, 92, 93, 102
  com_item_missingness, 21
  com_segment_missingness, 22
  com_unit_missingness, 24
  COMPATIBILITY (VARATT_REQUIRE_LEVELS), 103
  con_contradictions, 27
  con_detection_limits, 29, 30, 33, 34
  con_inadmissible_categorical, 31
  con_limit_deviations, 30, 33
  contradiction_functions, 25
  contradiction_functions_descriptions, 26
  CONTRADICTIONS, 104
  CONTRADICTIONS
    (WELL_KNOWN_META_VARIABLE_NAMES), 104
  DATA_ENTRY_TYPE, 104
  DATA_ENTRY_TYPE
    (WELL_KNOWN_META_VARIABLE_NAMES), 104
  DATA_TYPE, 36, 104
  DATA_TYPE
    (WELL_KNOWN_META_VARIABLE_NAMES), 104
  DATA_TYPES, 36, 80, 104
  DATA_TYPES_OF_R_TYPE, 37
  dataquieR, 13, 16, 35, 36, 59, 60, 65
  dataquieR_report, 20, 60, 65
  dataquieR-package (dataquieR), 35
  dataquieR_result
    (print.dataquieR_result), 59
  dataquieR_resultset, 35, 35, 39
  dataquieR_resultset_verify, 36
  DATETIME (DATA_TYPES), 36
  datetime (DATA_TYPES), 36
  DECIMALS, 104
  DECIMALS
    (WELL_KNOWN_META_VARIABLE_NAMES), 106
INDEX

104
default.stringsAsFactors, 52
DETECTION_LIMIT_LOW, 105
DETECTION_LIMIT_LOW
(WELL_KNOWN_META_VARIABLE_NAMES), 104
DETECTION_LIMIT_UP, 105
DETECTION_LIMIT_UP
(WELL_KNOWN_META_VARIABLE_NAMES), 104
DETECTION_LIMITS, 104
DETECTION_LIMITS
(WELL_KNOWN_META_VARIABLE_NAMES), 104
dimensions, 37, 39
DISTRIBUTION, 104
DISTRIBUTION
(WELL_KNOWN_META_VARIABLE_NAMES), 104
DISTRIBUTIONS, 38
dq_report, 35, 38, 41, 46
dq_report_by, 39, 40, 41
eEmmeans::emmeans, 8
enum, 8, 9, 23, 30, 33, 49, 52, 58
eNum/DATA_TYPES, 36
eval.parent, 56, 79, 96

factor, 66–76, 93
FLOAT (DATA_TYPES), 36
float, 37
float (DATA_TYPES), 36
format, 62
function, 45, 55, 91, 95

ggplot, 5, 14, 17, 28, 30, 34, 89, 99
ggplot2, 12, 15, 43, 63
ggplot2::geom_jitter, 13, 16
ggplot2::geom_line(), 8
glm, 50

HARD_LIMIT_LOW, 105
HARD_LIMIT_LOW
(WELL_KNOWN_META_VARIABLE_NAMES), 104
HARD_LIMIT_UP, 105
HARD_LIMIT_UP
(WELL_KNOWN_META_VARIABLE_NAMES), 104

HARD_LIMIتس, 104
HARD_LIMIتس
(WELL_KNOWN_META_VARIABLE_NAMES), 104
html_dependency_vert_dt, 42
INCL_HARD_LIMIT_LOW, 105
INCL_HARD_LIMIT_LOW
(WELL_KNOWN_META_VARIABLE_NAMES), 104
INCL_HARD_LIMIT_UP, 105
INCL_HARD_LIMIT_UP
(WELL_KNOWN_META_VARIABLE_NAMES), 104
INCL_SOFT_LIMIT_LOW, 105
INCL_SOFT_LIMIT_LOW
(WELL_KNOWN_META_VARIABLE_NAMES), 104
INCL_SOFT_LIMIT_UP, 105
INCL_SOFT_LIMIT_UP
(WELL_KNOWN_META_VARIABLE_NAMES), 104
int_datatype_matrix, 42
integer (DATA_TYPES), 36
integer, 7, 9, 13, 17, 18, 37, 39, 43, 46, 54, 63
is.finite, 90
is.integer, 90

JUMP_LIST, 87, 104
JUMP_LIST
(WELL_KNOWN_META_VARIABLE_NAMES), 104
KEY_DATETIME, 104
KEY_DATETIME
(WELL_KNOWN_META_VARIABLE_NAMES), 104
KEY_DEVICE, 104
KEY_DEVICE
(WELL_KNOWN_META_VARIABLE_NAMES), 104
KEY_OBSERVER, 104
KEY_OBSERVER
(WELL_KNOWN_META_VARIABLE_NAMES), 104
KEY_STUDY_SEGMENT, 41, 105
KEY_STUDY_SEGMENT
(WELL_KNOWN_META_VARIABLE_NAMES), 104
INDEX

LABEL, 104
LABEL (WELL_KNOWN_META_VARIABLE_NAMES), 104
list, 5, 6, 8, 20, 21, 28, 30, 34, 39, 43, 46, 53, 55, 59, 63, 89, 91, 92, 95
lme4::lmer, 50
logical, 8, 15, 21, 27, 43, 46, 49, 51, 52, 58, 61, 63, 66–75, 77–82, 85, 87, 88, 91, 93, 95, 102
LONG_LABEL, 105
LONG_LABEL
(WELL_KNOWN_META_VARIABLE_NAMES), 104
match.arg, 37
message, 60
methods, 61
mget, 54, 92
MISSING_LIST, 87, 104
MISSING_LIST
(WELL_KNOWN_META_VARIABLE_NAMES), 104
noquote, 61, 62
numeric, 7, 9, 11, 15, 19, 21, 23, 27, 32, 43, 77, 79, 87–90, 93, 99–101
numeric (DATA_TYPES), 36
OPTIONAL (VARATT_REQUIRE_LEVELS), 103
options, 62
ordered, 61, 76
parallelMap::parallelStart, 39, 46, 55, 95
pipeline_recursive_result, 20, 21, 44, 46
pipeline_vectorized, 44, 45, 46, 59
prep_add_to_meta, 48
prep_check_meta_names, 49, 52
prep_clean_labels, 50
prep_create_meta, 49, 52, 103
prep_datatype_from_data, 53
prep_map_labels, 37, 53
prep_min_obs_level, 54
prep_pmap, 55
prep_prepare_dataframes, 56
prep_study2meta, 56, 58, 96
prep_valuelabels_from_data, 59
print.dataquieR_result, 59
print.dataquieR_resultset, 35, 40, 60
print.default, 61, 62
print.ReportSummaryTable, 61
pro_applicability_matrix, 62
rbind_ReportSummaryTable, 64
RECODE, 105
RECODE
(WELL_KNOWN_META_VARIABLE_NAMES), 104
RECOMMENDED (VARATT_REQUIRE_LEVELS), 103
REQUIRED (VARATT_REQUIRE_LEVELS), 103
requireNamespace, 66
rmarkdown::render, 60
robustbase::mc, 13, 16
simpleError, 85
simpleWarning, 102
SOFTWARE_LIMIT_LOW, 105
SOFTWARELIMIT_LOW
(WELL_KNOWN_META_VARIABLE_NAMES), 104
SOFTWARE_LIMIT_UP, 105
SOFTWARELIMIT_UP
(WELL_KNOWN_META_VARIABLE_NAMES), 104
SOFTWARELIMITS, 104
SOFTWARELIMITS
(WELL_KNOWN_META_VARIABLE_NAMES), 104
SPLIT_CHAR, 64, 76, 94
sprintf, 85, 102
STRING (DATA_TYPES), 36
string, 37
string (DATA_TYPES), 36
summary.dataquieR_resultset, 35, 40, 65
sys.calls, 85, 86
sys.frames, 85, 86
table, 61
TECHNICAL (VARATT_REQUIRE_LEVELS), 103
UNKNOWN (VARATT_REQUIRE_LEVELS), 103
util_anytime_installed, 66
util_app_cd, 66
util_app_dc, 67
util_app_dl, 67
util_app_ed, 68
util_app_iac, 69
util_app_iav, 69
INDEX

util_app_im, 70
util_app_loess, 71
util_app_mar, 71
util_app_mol, 72
util_app_ol, 73
util_app_sm, 73
util_app_sos, 74
util_app_vc, 75
util_as_numeric, 76
util_assign_levlabs, 75
util_backtickQuote, 77
util_check_data_type, 77
util_check_one_unique_value, 78
util_correct_variable_use, 4, 79, 80
util_correct_variable_use(), 4
util_correct_variable_use2, 4, 80
util_correct_variable_use2
  (util_correct_variable_use), 79
util_correct_variable_use2(), 4
util_count_code_classes, 81
util_count_codes, 81
util_count_NA, 82
util_detectcores, 82
util_dichotomize, 83
util_dist_selection, 10, 83
util_empty, 84
util_ensure_suggested, 84
util_error, 85
util_find_external_functions_in_stacktrace, 85
util_find_first_externally_called_functions_in_stacktrace, 86
util_fix_rstudio_bugs, 86
util_get_code_list, 87
util_get_var_att_names_of_level, 87
util_heatmap_1th, 88
util_hubert, 89
util_interpret_limits, 89
util_is_integer, 90
utilLooks_like_missing, 90
util_make_function, 91
util_map_all, 91
util_map_labels, 92
util_no_value_labels, 93
util_observations_in_subgroups, 93
util_only_NAs, 94
util_par_pmap, 95
util_parse_assignments, 94
util_prepare_dataframes, 79, 96
util_replace_codes_by_NA, 98
util_set_dQuoteString, 98
util_set_size, 99
util_set_sQuoteString, 99
util_sigmagap, 100
util_sixsigma, 100
util_tukey, 101
util_validate_known_meta, 101
util_warn_unordered, 102
util_warning, 102

VALUE_LABELS, 93, 104
VALUE_LABELS
  (WELL_KNOWN_META_VARIABLE_NAMES), 104
VAR_NAMES, 104
VAR_NAMES
  (WELL_KNOWN_META_VARIABLE_NAMES), 104
VARATT_REQUIRE_LEVELS, 49, 52, 56, 58, 96, 103
variable, 6, 7, 9, 11, 15, 23, 24, 41, 43, 53, 59, 87, 88
variable (DATA_TYPES), 36
variable attribute, 5–7, 9, 11, 13, 15, 17, 18, 21, 23, 24, 27, 30, 32, 33, 36, 39, 41, 43, 45, 63, 79, 81–83, 87, 93
variable attribute
  (WELL_KNOWN_META_VARIABLE_NAMES), 104
variable list, 5, 7, 9, 11, 13, 17, 18, 21, 24, 27, 30, 32, 33, 45, 54, 88, 93
variable list (DATA_TYPES), 36
variable roles, 13, 17, 23
variable roles (VARIABLE_ROLES), 103
VARIABLE_ORDER, 105
VARIABLE_ORDER
  (WELL_KNOWN_META_VARIABLE_NAMES), 104
VARIABLE_ROLE, 105
VARIABLE_ROLE
  (WELL_KNOWN_META_VARIABLE_NAMES), 104
VARIABLE_ROLES, 103
vector, 76, 102
WELL_KNOWN_META_VARIABLE_NAMES, 52, 104
write, 62