Package 'eHDPrep'

September 22, 2026

Type Package

Title Quality Control and Semantic Enrichment of Datasets

Version 2.0.0

Maintainer Ian Overton <I.Overton@qub.ac.uk>

Description A tool for the preparation and enrichment of health datasets for analysis (Toner et al. (2023) <doi:10.1093/gigascience/giad030>). Provides functionality for assessing data quality and for improving the reliability and machine interpretability of a dataset. 'eHDPrep' also enables semantic enrichment of a dataset where metavariables are discovered from the relationships between input variables determined from user-provided ontologies.

License GPL-3

URL https://github.com/overton-group/eHDPrep

BugReports https://github.com/overton-group/eHDPrep/issues

Encoding UTF-8

LazyData true

Config/build/compact-vignettes both

Imports ggplot2 (>= 3.3.3), dplyr (>= 1.1.0), forcats (>= 0.5.0), stringr (>= 1.4.0), purrr (>= 0.3.4), tidyr (>= 1.1.2), kableExtra (>= 1.3.1), magrittr (>= 2.0.1), tibble (>= 3.0.5), scales (>= 1.1.1), rlang (>= 0.4.10), quanteda (>= 2.1.2), tm (>= 0.7-8), pheatmap (>= 1.0.12), igraph (>= 1.2.6), tidygraph (>= 1.2.0), readr (>= 1.4.0), readxl (>= 1.3.1), knitr (>= 1.31), tidyselect (>= 1.1.0)

BuildVignettes true

VignetteBuilder knitr

Depends R (>= 3.6.0)

Suggests testthat (>= 2.1.0), ggraph (>= 2.0.5), rmarkdown (>= 2.3), VIM (>= 6.0.0)

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Tom Toner [aut] (ORCID: <https://orcid.org/0000-0001-8059-5822>),
Ian Overton [aut, cre] (ORCID: <https://orcid.org/0000-0003-1158-8527>)

Repository CRAN

Date/Publication 2026-09-22 20:10:07 UTC

Contents

apply_quality_ctrl . . . 3
assess_completeness . . . 5
assess_quality . . . 7
assume_var_classes . . . 9
cellspec_lgl . . . 10
coerce_dates . . . 11
coerce_numeric_vars . . . 12
compare_completeness . . . 13
compare_info_content . . . 14
compare_info_content_plt . . . 15
completeness_heatmap . . . 16
count_compare . . . 17
discrete.mi . . . 19
distant_neg_val . . . 20
edge_tbl_to_graph . . . 20
encode_as_num_mat . . . 21
encode_binary_cats . . . 22
encode_bin_cat_vec . . . 23
encode_cats . . . 24
encode_genotypes . . . 25
encode_genotype_vec . . . 26
encode_ordinals . . . 27
entropy . . . 28
exact.kde . . . 28
example_data . . . 29
example_edge_tbl . . . 30
example_mapping_file . . . 31
example_ontology . . . 31
export_dataset . . . 32
extract_freetext . . . 32
identify_inconsistency . . . 34
import_dataset . . . 36
import_var_classes . . . 37
impute_missing_values . . . 38
information_content_contin . . . 40
information_content_discrete . . . 41
join_vars_to_ontol . . . 41
max_catchNAs . . . 42
mean_catchNAs . . . 43
merge_cols . . . 43

metavariable_agg . . . 44
metavariable_info . . . 46
metavariable_variable_descendants . . . 48
min_catchNAs . . . 49
mi_content_discrete . . . 49
mod_track . . . 50
node_IC_zhou . . . 51
normalize . . . 52
nums_to_NA . . . 52
onehot_vec . . . 53
ordinal_label_levels . . . 53
plot_completeness . . . 54
prod_catchNAs . . . 55
report_var_mods . . . 56
review_quality_ctrl . . . 56
row_completeness . . . 58
semantic_enrichment . . . 59
skipgram_append . . . 61
skipgram_freq . . . 62
skipgram_identify . . . 63
strings_to_NA . . . 65
sum_catchNAs . . . 66
validate_consistency_tbl . . . 66
validate_mapping_tbl . . . 68
validate_ontol_nw . . . 69
variable.bw.kde . . . 69
variable_completeness . . . 70
variable_entropy . . . 71
warn_missing_dots . . . 71
zero_entropy_variables . . . 72

Index **73**

`apply_quality_ctrl` *Apply quality control measures to a dataset*

Description

The primary high level function for quality control. Applies several quality control functions in sequence to input data frame (see Details for individual functions).

Usage

```
apply_quality_ctrl(
  data,
  id_var,
  class_tbl,
  bin_cats = NULL,
```

```
  min_freq = 1,
  to_numeric_matrix = FALSE,
  impute = FALSE,
  impute_method = "auto"
)
```

Arguments

data	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from dbplyr or dtplyr).
id_var	An unquoted expression which corresponds to a variable (column) in data which identifies each row (sample).
class_tbl	data frame such as the output tibble from assume_var_classes followed by import_var_classes.
bin_cats	Optional named vector of user-defined values for binary values using binary_label_1 = binary_label_2 syntax (e.g. c("No" = "Yes") would assign level 1 to "No" and 2 to "Yes"). See encode_binary_cats for defaults. Applied to variables (columns) labelled "binary", "character", "factor", or "logical" in class_tbl. Note: "character" and "factor" variables with exactly two unique non-missing values are automatically treated as binary.
min_freq	Minimum frequency of occurrence extract_freetext will use to extract groups of proximal words in free-text from variables (columns) labelled "freetext" in class_tbl.
to_numeric_matrix	Should QC'ed data be converted to a numeric matrix? Default: FALSE.
impute	Should missing values be imputed? Default: FALSE. When TRUE, impute_missing_values is applied after missing values are standardised (strings_to_NA) but before any encoding, so that imputation operates on the raw variable values (e.g. mode imputation fills genuine categories). All variables (columns) except id_var and any free text variables are imputed.
impute_method	Imputation method passed to impute_missing_values when impute = TRUE. One of "auto" (default), "median", "mean", "mode", "constant", or "knn".

Details

The wrapped functions are applied in the following order:

1. Standardise missing values (strings_to_NA)
2. Impute missing values (optional, impute_missing_values)
3. Encode binary categorical variables (columns) (encode_binary_cats)
4. Encode (specific) ordinal variables (columns)(encode_ordinals)
5. Encode genotype variables (encode_genotypes)
6. Extract information from free text variables (columns) (extract_freetext)
7. Encode non-binary categorical variables (columns) (encode_cats)
8. Encode output as numeric matrix (optional, encode_as_num_mat)

class_tbl is used to apply the above functions to the appropriate variables (columns).

Value

data with several QC measures applied.

See Also

Other high level functionality: assess_quality(), impute_missing_values(), review_quality_ctrl(), semantic_enrichment()

Examples

```
data(example_data)
require(tibble)

# create an example class_tbl object
# note that diabetes_type is classes as ordinal and is not modified as its
# levels are not pre-coded
tibble::tribble(~"var", ~"datatype",
"patient_id", "id",
"tumoursize", "numeric",
"t_stage", "ordinal_tstage",
"n_stage", "ordinal_nstage",
"diabetes", "factor",
"diabetes_type", "ordinal",
"hypertension", "factor",
"rural_urban", "factor",
"marital_status", "factor",
"SNP_a", "genotype",
"SNP_b", "genotype",
"free_text", "freetext") -> data_types

data_QC <- apply_quality_ctrl(example_data, patient_id, data_types,
   bin_cats =c("No" = "Yes", "rural" = "urban"),  min_freq = 0.6)
```

---

assess_completeness *Assess completeness of a dataset*

Description

Assesses and visualises completeness of the input data across both rows (samples) and columns (variables).

Usage

```
assess_completeness(data, id_var, plot = TRUE)
```

Arguments

data	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from dbplyr or dtplyr).
id_var	An unquoted expression which corresponds to a variable (column) in data which identifies each row (sample).
plot	Should plots be rendered when function is run? (Default: TRUE)

Details

Returns a list of completeness assessments:

variable_completeness A tibble detailing completeness of variables (columns) (via variable_completeness).

row_completeness A tibble detailing completeness of rows (via row_completeness).

completeness_plot A plot of row and variable (column) completeness (via plot_completeness).

completeness_heatmap A clustered heatmap of cell completeness (via completeness_heatmap).

plot_completeness_heatmap A function which creates a clean canvas before plotting the completeness heatmap.

Value

list of completeness tibbles and plots

See Also

Other measures of completeness: compare_completeness(), completeness_heatmap(), plot_completeness(), row_completeness(), variable_completeness()

Examples

```
data(example_data)
res <- assess_completeness(example_data, patient_id)

# variable completeness table
res$variable_completeness

# row completeness table
res$row_completeness

# show completeness of rows and variables as a bar plot
res$completeness_plot

# show dataset completeness in a clustered heatmap
# (this is similar to res$completeness_heatmap but ensures a blank canvas is first created)
res$plot_completeness_heatmap(res)
```

`assess_quality` *Assess quality of a dataset*

Description

Provides information on the quality of a dataset. Assesses dataset's completeness, internal consistency, and entropy.

Usage

```
assess_quality(data, id_var, consis_tbl, plot = TRUE)
```

Arguments

`data`	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from dbplyr or dtplyr).
`id_var`	An unquoted expression which corresponds to a variable (column) in `data` which identifies each row (sample).
`consis_tbl`	Optional consistency table.
`plot`	Logical. Should completeness plots be printed? Defaults to TRUE. Plots are only displayed when a graphics device is active.

Details

Wraps several quality assessment functions from `eHDPrep` and returns a nested list with the following structure:

completeness - A list of completeness assessments:

1. Tibble of variable (column) completeness (via `variable_completeness`)
2. Tibble of row (sample) completeness (via `row_completeness`)
3. Plot of row and variable completeness (via `plot_completeness`)
4. Completeness heatmap (via `completeness_heatmap`)
5. A function which creates a clean canvas before plotting the completeness heatmap.

internal_inconsistency - Tibble of internal inconsistencies, if any are present and if a consistency table is supplied (via `identify_inconsistency`).

vars_with_zero_entropy - Names of variables (columns) with zero entropy (via `zero_entropy_variables`)

Value

Nested list of quality measurements

Consistency Table Requirements

Table must have exactly five character columns. The columns should be ordered according to the list below which describes the values of each column:

1. First column name of data values that will be subject to consistency checking. String. Required.
2. Second column name of data values that will be subject to consistency checking. String. Required for a two-variable check; NA for a single-variable check (see below).
3. Logical test to compare columns one and two. One of: ">",">=", "<","<=","==", "!=". String. Optional if columns 4 and 5 have non-NA values.
4. Either a single character string or a colon-separated range of numbers which should only appear in column A. Optional if column 3 has a non-NA value.
5. Either a single character string or a colon-separated range of numbers which should only appear in column B given the value/range specified in column 4. Optional if column 3 has a non-NA value.

Each row should detail one test to make. Therefore, either column 3 or columns 4 and 5 must contain non-NA values.

Single-variable checks. A rule may also constrain one variable against fixed values, rather than comparing two variables. Such a rule has NA in column 2, the boundary in column 4, and NA in column 5. Two forms are available:

- *Comparison against a value*: column 3 holds the logical test and column 4 the value to compare against. For example `"age", NA, ">=", "70", NA` requires every value of age to be at least 70.
- *Permitted values*: column 3 is NA and column 4 holds either a colon-separated numeric range or a single category. For example `"age", NA, NA, "0:120", NA` requires every value of age to lie between 0 and 120, and `"sex", NA, NA, "Male", NA` requires every value of sex to be `"Male"`.

Unlike two-variable rules, single-variable rules may use columns 3 and 4 together. Missing (NA) values are never reported as inconsistent by a single-variable check, as they cannot be assessed against a boundary.

See Also

Other high level functionality: `apply_quality_ctrl()`, `impute_missing_values()`, `review_quality_ctrl()`, `semantic_enrichment()`

Examples

```
# general example
data(example_data)
res <- assess_quality(example_data, patient_id)

# example of internal consistency checks on more simple dataset
# describing bean counts
require(tibble)
```

```
# creating `data`:
beans <- tibble::tibble(red_beans = 1:15,
blue_beans = 1:15,
total_beans = 1:15*2,
red_bean_summary = c(rep("few_beans",9), rep("many_beans",6)))

# creating `consis_tbl`
bean_rules <- tibble::tribble(~varA, ~varB, ~lgl_test, ~varA_boundaries, ~varB_boundaries,
"red_beans", "blue_beans", "==", NA, NA,
"red_beans", "total_beans", "<=", NA,NA,
"red_beans", "red_bean_summary", NA, "1:9", "few_beans",
"red_beans", "red_bean_summary", NA, "10:15", "many_beans")

# add some inconsistencies
beans[1, "red_bean_summary"] <- "many_beans"
beans[1, "red_beans"] <- 10

res <- assess_quality(beans, consis_tbl = bean_rules)

# variable completeness table
res$completeness$variable_completeness

# row completeness table
res$completeness$row_completeness

# show completeness of rows and variables as a bar plot
res$completeness$completeness_plot

# show dataset completeness in a clustered heatmap
res$completeness$plot_completeness_heatmap(res$completeness)

# show any internal inconsistencies
res$internal_inconsistency

# show any variables with zero entropy
res$vars_with_zero_entropy
```

`assume_var_classes` *Assume variable classes in data*

Description

Classes/data types of data variables are assumed with this function and exported to a .csv file for amendment. Any incorrect classes can then be corrected and imported using `import_var_classes`. Character and factor variables with exactly two unique non-missing values are labelled "binary". Character variables with more than two but no more than `factor_threshold` unique non-missing values are labelled "factor".

Usage

```
assume_var_classes(data, out_file = NULL, factor_threshold = 5L)
```

Arguments

data	data frame
out_file	file where variables and their assumed classes are stored for user verification.
factor_threshold	integer. Maximum number of unique non-missing values for a character variable to be labelled "factor". Variables with exactly 2 unique values are labelled "binary" regardless of this threshold. Set to 0 or NULL to disable factor detection. Default: 5.

Value

Writes a .csv file containing the variables and their assumed data types / classes.

See Also

import_var_classes

Examples

```
# example below assumes incorrectly for several variables
tmp = tempfile(fileext = ".csv")
data(example_data)
assume_var_classes(example_data, tmp)
```

---

cellspec_lgl *Kable logical data highlighting*

Description

Adds colour highlighting to cell values if they are encoded as logical values. Output should then be passed to knitr's kable function.

Usage

```
cellspec_lgl(.data, rg = FALSE)
```

Arguments

.data	Table to be highlighted.
rg	Should red and green be used for TRUE and FALSE values, respectively? If FALSE (default), colour-blind-friendly colours are applied.

Details

This is useful for identifying the encoding used in a value (e.g. the difference between the string "TRUE" and truth value of logic TRUE). This highlighting can also be useful when visually assessing cell values in a table. The colour naming format (HTML or LaTeX) is automatically detected. There are four cell types considered:

1. non-logical cells are coloured black)
2. TRUE cells are coloured red (default) or green if rg is TRUE
3. FALSE cells are coloured cyan (default) or red if rg is TRUE
4. NA cells are coloured gray

Note: When passed to kable(), the escape parameter should be FALSE for colours to be rendered correctly.

Value

Table with cell colours specified.

See Also

kable cell_spec

---

coerce_dates *Coerce columns to dates, replacing unparseable values with NA*

Description

For each specified column, attempts to coerce the values to Date. Values that cannot be parsed (e.g. "not recorded" appearing in an otherwise date-valued column) are replaced with NA. A message lists the dropped values per column so the user is aware of what was discarded. Use suppressMessages() to silence.

Usage

```
coerce_dates(
  data,
  ...,
  formats = c("%Y-%m-%d", "%Y/%m/%d", "%d-%m-%Y", "%d/%m/%Y")
)
```

Arguments

data	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from dbplyr or dtplyr).
...	<dplyr_tidy_select> Columns to coerce. If none are supplied, data is returned unchanged.
formats	character vector of date formats to try, in order, as accepted by strptime. Default: c("%Y-%m-%d", "%Y/%m/%d", "%d-%m-%Y", "%d/%m/%Y").

Details

Columns which are already `Date` objects are left untouched. Date-time (`POSIXct`) columns are converted to `Date`, discarding the time of day; a message reports this, as it is a loss of information.

Dates are parsed with the formats given in `formats`, tried in order. The default covers the ISO 8601 form (`"2020-01-15"`) and the common unambiguous variants. Note that formats are not guessed per value: a column is parsed with the first format which succeeds for all of its non-missing values, so mixed-format columns will report the values that did not parse.

Value

`data` with the specified columns coerced to `Date`.

See Also

`coerce_numeric_vars`

Examples

```
df <- data.frame(x = c("2020-01-15", "2020-02-30", "not recorded"),
                 y = c("a", "b", "c"))
# "2020-02-30" is not a real date and "not recorded" is not a date at all,
# so both become NA and are reported:
coerce_dates(df, x)
```

---

coerce_numeric_vars *Coerce columns to numeric, replacing non-numeric values with NA*

Description

For each specified column, attempts to coerce the values to numeric. Values that cannot be coerced (e.g. character strings such as `"hemolysed"` appearing in an otherwise numeric column) are replaced with `NA`. A message lists the dropped values per column so the user is aware of what was discarded. Use `suppressMessages()` to silence.

Usage

```
coerce_numeric_vars(data, ...)
```

Arguments

`data`	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from dbplyr or dtplyr).
`...`	<`dplyr_tidy_select`> Columns to coerce. If none are supplied, `data` is returned unchanged.

Details

Columns that are already numeric are left untouched.

Value

`data` with the specified columns coerced to numeric.

Examples

```
df <- data.frame(x = c("1", "2", "hemolysed", "3"),
                 y = c("a", "b", "c", "d"))
coerce_numeric_vars(df, x)
```

---

compare_completeness *Compare Completeness between Datasets*

Description

Produces a density plot comparing the completeness of two datasets (`tbl_a` and `tbl_b`) for variables (if `dim` == 2, default) or row (if `dim` == 1). The label used to identify the dataset's density curve can be specified using `tbl_a_lab` and `tbl_b_lab`.

Usage

```
compare_completeness(tbl_a, tbl_b, dim = 2, tbl_a_lab = NULL, tbl_b_lab = NULL)
```

Arguments

`tbl_a`	Data frame of the first data frame to compare.
`tbl_b`	Data frame of the second data frame to compare.
`dim`	Integer. Dimension to measure completeness on. 2 (Default) measures completeness by variable. 1 measures completeness by row.
`tbl_a_lab`	String to be used to label `tbl_a` on the output plot.
`tbl_b_lab`	String to be used to label `tbl_b` on the output plot.

Value

Plot showing densities of completeness across both datasets.

See Also

Other measures of completeness: `assess_completeness()`, `completeness_heatmap()`, `plot_completeness()`, `row_completeness()`, `variable_completeness()`

Examples

```
data(example_data)
compare_completeness(example_data, strings_to_NA(example_data), dim = 2,
                     "raw", "cleaned")
```

compare_info_content *Information Content Comparison Table*

Description

Used to quantify the amount of information loss, if any, which has occurred in a merging procedure between two discrete variables.

Usage

```
compare_info_content(input1, input2, composite)
```

Arguments

input1	Character vector. First variable to compare
input2	Character vector. Second variable to compare
composite	Character vector. Composite variable, resultant of merging input1 and input2.

Details

The function requires the two discrete variables which have been merged (input1 and input2) and the composite variable (output). For each input, information content is calculated using information_content_discrete along with each input's mutual information content with the composite variable using mi_content_discrete. The function returns a table describing these measures.

If the mutual information content between an input variable and the composite variable is equal to the information content of the input variable, it is confirmed that all information in the input variable has been incorporated into the composite variable. However, if one or both input variables' information content is not equal to their mutual information with the composite variables, information loss has occurred.

Value

Table containing information content for input1 and input2 and their mutual information content with composite.

See Also

compare_info_content_plt

Examples

```
data(example_data)
require(dplyr)
require(magrittr)
example_data %>%
   mutate(diabetes_merged = coalesce(diabetes_type, diabetes)) %>%
```

```
   select(starts_with("diabetes")) ->
   merged_data

compare_info_content(merged_data$diabetes,
                     merged_data$diabetes_type,
                     merged_data$diabetes_merged)
```

`compare_info_content_plt` *Information Content Comparison Plot*

Description

This function requires the output from `compare_info_content`. It is used to visualise the amount of information loss, if any, which has occurred in a merging procedure between two discrete variables.

Usage

```
compare_info_content_plt(compare_info_content_res)
```

Arguments

`compare_info_content_res`
: Output from `compare_info_content`.

Details

If the mutual information content between an input variable and the composite variable is equal to the information content of the input variable, it is confirmed that all information in the input variable has been incorporated into the composite variable.

Value

Plot of measures calculated in `compare_info_content`.

See Also

`compare_info_content`

Examples

```
data(example_data)
require(dplyr)
require(magrittr)
example_data %>%
   mutate(diabetes_merged = coalesce(diabetes_type, diabetes)) %>%
   select(starts_with("diabetes")) ->
   merged_data
```

```
compare_info_content(merged_data$diabetes,
                     merged_data$diabetes_type,
                     merged_data$diabetes_merged) %>%
                     compare_info_content_plt()
```

`completeness_heatmap` *Completeness Heatmap*

Description

Produces a heatmap visualising completeness across a dataset.

Usage

```
completeness_heatmap(
  data,
  id_var,
  annotation_tbl = NULL,
  method = 1,
  show_rownames = FALSE,
  ...
)
```

Arguments

`data`	Data frame to be analysed.
`id_var`	Character constant of row identifier variable name.
`annotation_tbl`	Data frame containing variable annotation data. Column 1 should contain variable names, column 2 should contain an annotation label.
`method`	Integer between 1 and 3. Default: 1. See Details for more information.
`show_rownames`	Boolean. Should rownames be shown. Default: False.
`...`	Parameters to be passed to `pheatmap`.

Details

- Method 1: Missing values are numerically encoded with a highly negative number, numerically distant from all values in `data`, using `distant_neg_val`. Values in categorical variables are replaced with the number of unique values in the variable. Clustering uses these values. Cells are coloured by presence (yellow = missing; blue = present).
- Method 2: Same as Method 1 but cells are coloured by values used to cluster.
- Method 3: Values in `data` are encoded as Boolean values for clustering (present values = 1; missing values = 0). Cells are coloured by presence (yellow = missing; blue = present).

Value

completeness heatmap

Note

See examples of how to plot using plot.new(). This is ensure a new plot is created for the heatmap

References

Kolde R (2019). _pheatmap: Pretty Heatmaps_. R package version 1.0.12, <https://CRAN.R-project.org/package=pheatmap>.

See Also

pheatmap

Other measures of completeness: assess_completeness(), compare_completeness(), plot_completeness(), row_completeness(), variable_completeness()

Examples

```
data(example_data)

# heatmap without variable category annotations:
hm <- completeness_heatmap(example_data,patient_id)
plot.new() # ensure new plot is created
hm


# heatmap with variable category annotations:
## create a dataframe containing variable annotations
tibble::tribble(~"var", ~"datatype",
"patient_id", "id",
"tumoursize", "numeric",
"t_stage", "ordinal_tstage",
"n_stage", "ordinal_nstage",
"diabetes", "factor",
"diabetes_type", "ordinal",
"hypertension", "factor",
"rural_urban", "factor",
"marital_status", "factor",
"SNP_a", "genotype",
"SNP_b", "genotype",
"free_text", "freetext") -> data_types

hm <- completeness_heatmap(example_data,patient_id, annotation_tbl = data_types)
plot.new() # ensure new plot is created
hm
```

---

count_compare *Compare unique values before and after data modification*

Description

Performs comparison of variables before and after a change has been applied in order to allow manual inspection and review of modifications made during the dataset preparation process.

Usage

```
count_compare(
  cols2compare,
  before_tbl = NULL,
  after_tbl = NULL,
  only_diff = FALSE,
  kableout = TRUE,
  caption = NULL,
  latex_wrap = FALSE
)
```

Arguments

`cols2compare`	Variables to compare between tables.
`before_tbl`	Data frame from before modification was made.
`after_tbl`	Data frame from after modification was made.
`only_diff`	Keep only rows which differ between the tables (good for variables with many unique values, such as numeric variables).
`kableout`	Should output be a `kable` from `knitr`? If not, returns a `tibble`. (Default: TRUE)
`caption`	Caption for `kable`'s `caption` parameter.
`latex_wrap`	Should tables be aligned vertically rather than horizontally? Useful for wide table which would otherwise run off a page in LaTeX format.

Details

The purpose of this function is to summarise individual alterations in a dataset and works best with categorical variables. The output contains two tables derived from the parameters `before_tbl` and `after_tbl`. Each table shows the unique combinations of values in variables specified in the parameter `cols2compare` if the variable is present. The tables are presented as two sub-tables and therefore share a single table caption. This caption is automatically generated describing the content of the two sub-tables when the parameter `caption` is not specified. The default output is a `kable` containing two sub-kables however if the parameter `kableout` is `FALSE`, a list containing the two `tibbles` are returned. This may preferable for further analysis on the tables' contents.

Value

Returns list of two tibbles or a kable (see `kableout` argument), each tallying unique values in specified columns in each input table.

Examples

```
# merge data as the example modification
example_data_merged <- merge_cols(example_data, diabetes_type, diabetes,
"diabetes_merged", rm_in_vars = TRUE)

# review the differences between the input and output of the variable merging step above:
count_compare(before_tbl = example_data,
              after_tbl = example_data_merged,
                        cols2compare = c("diabetes", "diabetes_type", "diabetes_merged"),
                             kableout = FALSE)
```

`discrete.mi` *Calculate mutual information of a matrix of discrete values*

Description

Compute mutual information between all rows of a matrix containing discrete outcomes.

Usage

```
discrete.mi(mat, progress.bar = FALSE)
```

Arguments

`mat`	A matrix of discrete values
`progress.bar`	Outputs status to terminal when set to 'text', or no status updates are output when set to `FALSE`.

Details

Note that only the lower triangle of the matrix is populated for speed, as the result is symmetric. Takes a matrix as input.

Value

A lower triangular matrix where element [i,j] contains the mutual information in bits between row i and row j of the input matrix

Author(s)

Alexander Lyulph Robert Lubbock, Ian Overton

distant_neg_val *Find highly distant value for data frame*

Description

Returns a numeric value which is distant from the values in `data` using the following equation:

$$output = -2 * (max(data) - min(data)))$$

Usage

```
distant_neg_val(data)
```

Arguments

`data`	data frame.

Value

Numeric vector of length 1

edge_tbl_to_graph *Convert edge table to tidygraph graph*

Description

A edge table, as a data frame, is converted to a directed tidygraph `tidygraph`. Column 1 of the edge table is interpreted as a "from" column, Column 2 is interpreted as a "to" column, and any further columns are interpreted as attributes of the entity/node recorded in column 1. Incomplete cases are removed from the edge table (rows) to avoid redundancy

Usage

```
edge_tbl_to_graph(edge_tbl)
```

Arguments

`edge_tbl`	data frame containing 'from' nodes in column 1 and 'to' nodes in column 2 so that all nodes go 'towards' the root node

Value

`tidygraph` representation of the edge table

Examples

```
# basic edge table
edge_tbl <- tibble::tribble(~from, ~to,
"Nstage", "TNM",
"Tstage", "TNM",
"Tumoursize", "property_of_tumour",
"Tstage", "property_of_tumour",
"property_of_tumour", "property_of_cancer",
"TNM", "property_of_cancer",
"property_of_cancer", "disease",
"disease", "root",
"root", NA)

graph <- edge_tbl_to_graph(edge_tbl)

graph

plot(graph)


# edge table with node attributes
## note that root node is included in final row to include its label
edge_tbl <- tibble::tribble(~from, ~to, ~label,
"Nstage", "TNM", "N stage",
"Tstage", "TNM", "T stage",
"Tumoursize", "property_of_tumour", "Tumour size",
"Tstage", "property_of_tumour", "T stage",
"property_of_tumour", "property_of_cancer", "Property of tumour",
"TNM", "property_of_cancer", "TNM",
"property_of_cancer", "disease", "Property of cancer",
"disease", "root", "Disease",
"root", NA, "Ontology Root")
graph <- edge_tbl_to_graph(edge_tbl)

graph

plot(graph)
```

---

encode_as_num_mat *Convert data frame to numeric matrix*

Description

Converts all columns to numeric and uses the row identifier column (`id_var`) as row names.

Usage

```
encode_as_num_mat(data, id_var)
```

Arguments

data	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from dbplyr or dtplyr).
id_var	An unquoted expression which corresponds to a variable in data which identifies each row.

Value

Numeric matrix with id_var values as row names

Examples

```
require(dplyr)
require(magrittr)
mtcars %>%
  dplyr::as_tibble(rownames = "id") %>%
  encode_as_num_mat(id)
```

---

encode_binary_cats *Encode categorical variables as binary factors*

Description

In a data frame, converts binary categories to factors. Ordering of levels is standardised to: negative_finding, positive_finding. This embeds a standardised numeric relationship between the binary categories while preserving value labels.

Usage

```
encode_binary_cats(data, ..., values = NULL)
```

Arguments

data	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from dbplyr or dtplyr).
...	<tidy-select> One or more unquoted expressions separated by commas. Variable names can be used as if they were positions in the data frame, so expressions like x:y can be used to select a range of variables.
values	Optional named vector of user-defined values for binary values using binary_label_1 = binary_label_2 syntax (e.g. c("No" = "Yes") would assign level 1 to "No" and 2 to "Yes").

Details

Binary categories to convert can be specified with a named character vector, specified in values. The syntax of the named vector is: negative_finding = positive_finding. If values is not provided, the default list will be used: "No"="Yes", "No/unknown"="Yes", "no/unknown"="Yes", "Non-user"="User", "Never"="Ever", "WT"="MT", "Male"="Female", "male"="female", "M"="F", "m"="f", "FALSE"="TRUE", "false"="true", "0"="1", "rural"="urban", "Absent"="Present", "absent"="present", "Negative"="Positive", "negative"="positive".

Value

dataset with specified binary categories converted to factors.

Examples

```
# use built-in values. Note: rural_urban is not modified
# Note: diabetes is not modified because "missing" is interpreted as a third category.
# strings_to_NA() should be applied first
encode_binary_cats(example_data, hypertension, rural_urban)

# use custom values. Note: rural_urban is now modified as well.
encoded_data <- encode_binary_cats(example_data, hypertension, rural_urban,
                   values = c("No"= "Yes", "rural" = "urban"))

# to demonstrate the new numeric encoding:
dplyr::mutate(encoded_data, hypertension_num = as.numeric(hypertension), .keep = "used")
```

---

encode_bin_cat_vec *Encode a categorical vector with binary categories*

Description

In a character vector, converts binary categories to factor levels.

Usage

```
encode_bin_cat_vec(x, values = NULL, numeric_out = FALSE)
```

Arguments

x	non-numeric input vector
values	Optional named vector of user-defined values for binary values using binary_label_1 = binary_label_2 syntax (e.g. c("No" = "Yes") would assign level 1 to "No" and 2 to "Yes").
numeric_out	If true, numeric vector is returned. If false, factor is returned.

Details

Before pair matching, mixed boolean encodings are normalised automatically. If all unique non-missing values are boolean synonyms (e.g. a column containing `"false"` and `"true"` recorded inconsistently as `"False"` or `"No"`), they are collapsed to canonical `"false"`/`"true"` so that the `"false"="true"` default pair matches. The synonyms recognised are: `"false"`, `"FALSE"`, `"False"`, `"no"`, `"No"`, `"NO"`, `"n"`, `"N"` (negative); and `"true"`, `"TRUE"`, `"True"`, `"yes"`, `"Yes"`, `"YES"`, `"y"`, `"Y"` (positive). Note: numeric `"0"`/`"1"` are intentionally excluded as they are ambiguous (e.g. `"0"` may represent a missing value); handle these with `strings_to_NA` beforehand.

Binary categories to convert can be specified with a named character vector, specified in `values`. The syntax of the named vector is: `negative_finding = positive_finding`. If `values` is not provided, the default list will be used: `"No"="Yes"`, `"No/unknown"="Yes"`, `"no/unknown"="Yes"`, `"Non-user"="User"`, `"Never"="Ever"`, `"WT"="MT"`, `"Male"="Female"`, `"male"="female"`, `"M"="F"`, `"m"="f"`, `"FALSE"="TRUE"`, `"false"="true"`, `"0"="1"`, `"rural"="urban"`, `"Absent"="Present"`, `"absent"="present"`, `"Negative"="Positive"`, `"negative"="positive"`.

Value

Factor with false finding encoded as 1 and true finding encoded as 2. Alternatively, numeric vector if `numeric_out` parameter is `TRUE`.

`encode_cats` *Encode categorical variables using one-hot encoding.*

Description

Variables specified in `...` are replaced with new variables describing the presence of each unique category. Generated variable names have space characters replaced with "_" and commas are removed.

Usage

```
encode_cats(data, ...)
```

Arguments

`data`	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from dbplyr or dtplyr).
`...`	<`tidy-select`> One or more unquoted expressions separated by commas. Variable names can be used as if they were positions in the data frame, so expressions like `x:y` can be used to select a range of variables.

Value

Tibble with converted variables.

Examples

```
require(magrittr)
require(dplyr)

data(example_data)

# encode one variable
encode_cats(example_data, marital_status) %>%
select(starts_with("marital_status"))

# encode multiple variables
encoded <- encode_cats(example_data, diabetes, marital_status)

select(encoded, starts_with("marital_status"))
# diabetes_type included below but was not modified:
select(encoded, starts_with("diabetes"))
```

`encode_genotypes` *Encode genotype/SNP variables in data frame*

Description

Standardises homozygous SNPs (e.g. recorded as "A") to two character form (e.g. "A/A") and orders heterozygous SNPs alphabetically (e.g. "GA" becomes "A/G"). The SNP values are then converted from a character vector to an ordered factor, ordered by observed allele frequency (in the supplied cohort). The most frequent allele is assigned level 1, the second most frequent value is assigned level 2, and the least frequent values is assigned level 3). This method embeds the numeric relationship between the allele frequencies while preserving value labels.

Usage

```
encode_genotypes(data, ...)
```

Arguments

`data`	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from dbplyr or dtplyr).
`...`	<tidy-select> One or more unquoted expressions separated by commas. Variable names can be used as if they were positions in the data frame, so expressions like `x:y` can be used to select a range of variables.

Value

'data' with variables (`...`) encoded as standardised genotypes

Examples

```
data(example_data)
require(dplyr)
require(magrittr)

# one variable
encode_genotypes(example_data, SNP_a) %>%
select(SNP_a)

# multiple variables
encode_genotypes(example_data, SNP_a, SNP_b) %>%
select(SNP_a, SNP_b)

# using tidyselect helpers
encode_genotypes(example_data, dplyr::starts_with("SNP")) %>%
select(starts_with("SNP"))
```

`encode_genotype_vec` *Encode a genotype/SNP vector*

Description

Standardises homozygous SNP alleles (e.g. recorded as 'A') to two character form (e.g. 'A/A') and orders heterozygous SNP alleles alphabetically (e.g. "GA" becomes "A/G"). The SNP values are then converted from a character vector to an ordered factor, ordered by SNP allele frequency (e.g. most frequent SNP allele is 1, second most frequent value is 2, and least frequent values is 3). This method embeds the numeric relationship between the SNP allele frequencies while preserving value labels.

Usage

```
encode_genotype_vec(x)
```

Arguments

`x`	input vector containing genotype data

Value

Ordered factor, ordered by allele frequency in variable

encode_ordinals *Encode ordinal variables*

Description

Converts character or factor variables in the input data frame to ordered factors embedding numeric relationship between values while preserving value labels.

Usage

```
encode_ordinals(data, ord_levels, ..., strict_levels = TRUE)
```

Arguments

data	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from dbplyr or dtplyr).
ord_levels	character vector containing values in desired order (lowest to highest).
...	<tidy-select> One or more unquoted expressions separated by commas. Variable names can be used as if they were positions in the data frame, so expressions like x:y can be used to select a range of variables.
strict_levels	logical constant. If TRUE, variables in ... which contain values other than ord_levels (including NA) are not modified and a warning is given. If FALSE, values not in ord_levels are converted to NA.

Value

dataframe with specified variables encoded as ordered factors.

Examples

```
data(example_data)
require(dplyr)
require(magrittr)
encode_ordinals(example_data, ord_levels = c("N0","N1","N2"), n_stage)

# Note: "unequivocal" is present in  t_stage but not in `ord_levels`.
# with `strict_levels` TRUE, t_stage is unmodified and a warning message is given:

encode_ordinals(example_data,
   ord_levels = c("T1","T2","T3a", "T3b", "T4"), strict_levels = TRUE, t_stage) %>%
   select(t_stage)

# with `strict_levels` FALSE, it is replaced with NA:

encode_ordinals(example_data,
   ord_levels = c("T1","T2","T3a", "T3b", "T4"), strict_levels = FALSE, t_stage) %>%
   select(t_stage)
```

entropy *Calculate Entropy of a Vector*

Description

Calculates Shannon Entropy of a vector in bits (default) or natural units. Missing values are omitted from the calculation.

Usage

```
entropy(x, unit = c("bits"))
```

Arguments

x	Input vector
unit	Unit to measure entropy. Either "bits" (default) or "nats".

Value

Entropy of input variable

References

Shannon, C. E. A mathematical theory of communication. The Bell System Technical Journal 27, 379–423 (1948).

Examples

```
# no entropy:
vec <- c(1,1,1,1,1,1)
entropy(vec)

# entropy
vec <- c(1,2,3,4,5,6)
entropy(vec)
```

exact.kde *Exact kernel density estimation*

Description

Calculates KDE for a set of points exactly, rather than an approximation as per the density() core function.

Usage

```
exact.kde(x, bw, output.domain = x, na.rm = FALSE)
```

Arguments

x	A numeric vector of values
bw	The bandwidth to use - either a single value, or a vector of values the same length as x if using adaptive bandwidth estimation (with each value giving the bandwidth at the corresponding data point).
output.domain	The domain of values over which to estimate the density. Defaults to x. To use the same domain of x values as R's density, set to NULL.
na.rm	Remove missing values if TRUE

Details

Only tractable for around 10,000 data points or less - otherwise consider using the density() core function for a close approximation.

The density() core function approximation is normally a very good approximation, but some small values close to zero may become zero rather than just very small. This makes it less suitable for mutual information estimation.

Value

The exact kernel density estimate as a density object, compatible with R's density function.

Author(s)

Alexander Lyulph Robert Lubbock, Ian Overton

---

example_data *Example data for eHDPrep*

Description

A dataset containing synthetic example values to demonstrate functionality of 'eHDprep'

Usage

```
example_data
```

Format

A data frame with 1,000 rows and 10 variables:

patient_id 1 to 1000, effictively row numbers

tumoursize double. random values with a mean of 50 and SD of 20

t_stage character. T stage random values

n_stage character. N stage random values

diabetes character. Patient diabetes category

diabetes_type character. Patient diabetes type category

hypertension character. Patient hypertension category

rural_urban character. Patient domestic address category

marital_status character. Patient marital status category

SNP_a character. Single Nucleotide Polymorphism (SNP) of the patient

SNP_b character. Another SNP of the patient

free_text character. sentences from the 'stringr' package as an example of short free text variables

Source

synthetic

`example_edge_tbl` *Example ontology as an edge table for semantic enrichment*

Description

A data frame describing semantic links (edges) between entities in 'example_ontology'. Used to demonstrate semantic enrichment.

Usage

```
example_edge_tbl
```

Format

A data frame:

from character. Names of semantic concepts which have a directed relationship to concepts in 'to' column.

to character. Names of semantic concepts which have a directed relationship to concepts in 'from' column.

Details

Used in documentation and creation of 'example_ontology' in 'eHDPrep'.

Source

synthetic

example_mapping_file *Example mapping file for semantic enrichment*

Description

A data frame containing mappings between variables in ‘example_data‘ and ‘example_onto‘. Used to demonstrate semantic enrichment.

Usage

```
example_mapping_file
```

Format

A data frame:

variable character. names of variables in post-QC ‘example_data‘.

onto_entity character. names of mapped entities in ‘example_ontology‘.

Details

Maps variables in ‘example_data‘ to ‘example_ontology‘ in ’eHDPrep’.

Source

synthetic

example_ontology *Example ontology as a network graph for semantic enrichment*

Description

A small custom network graph to demonstrate semantic enrichment.

Usage

```
example_ontology
```

Format

tidygraph graph

Details

Contains semantic links of variables in ’eHDPrep”s ‘example_data‘ following quality control.

Source

synthetic

export_dataset *Export data to delimited file*

Description

Save dataset in .csv or .tsv format. A wrapper function for readr's write_csv and write_tsv.

Usage

```
export_dataset(x, file, format = "csv", ...)
```

Arguments

x	A data frame or tibble to write to disk.
file	File or connection to write to.
format	Character constant. "csv" (default) or "tsv"
...	parameters to pass to write_csv or write_tsv.

Value

x saved to file in selected format

See Also

write_csv and write_tsv

Other import to/export from 'R' functions: import_dataset()

Examples

```
data(example_data)
tmp = tempfile(fileext = ".csv")
export_dataset(example_data, tmp)
```

extract_freetext *Extract information from free text*

Description

Extracts information from specified free text variables (...) which occur in a minimum amount of rows (min_freq) and appends new variables to data.

Usage

```
extract_freetext(data, id_var, min_freq = 1, ...)
```

Arguments

data	Data frame to append skipgram variables to.
id_var	An unquoted expression which corresponds to a variable in data which identifies each row.
min_freq	Minimum percentage frequency of skipgram occurrence to return. Default = 1.
...	Unquoted expressions of free text variable names from which to extract information.

Details

New variables report the presence of skipgrams (proximal words in the text) with a minimum frequency (min_freq, default = 1%)).

Value

data with additional Boolean variables describing skipgrams in ...

References

Guthrie, D., Allison, B., Liu, W., Guthrie, L. & Wilks, Y. A Closer Look at Skip-gram Modelling. in Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06) (European Language Resources Association (ELRA), 2006).

Benoit K, Watanabe K, Wang H, Nulty P, Obeng A, Müller S, Matsuo A (2018). "quanteda: An R package for the quantitative analysis of textual data." _Journal of Open Source Software_, *3*(30), 774. doi:10.21105/joss.00774 <https://doi.org/10.21105/joss.00774>, <https://quanteda.io>.

Feinerer I, Hornik K (2020). _tm: Text Mining Package_. R package version 0.7-8, <https://CRAN.R-project.org/package=tm>.

Ingo Feinerer, Kurt Hornik, and David Meyer (2008). Text Mining Infrastructure in R. Journal of Statistical Software 25(5): 1-54. URL: https://www.jstatsoft.org/v25/i05/.

See Also

Principle underlying function: tokens_ngrams

Other free text functions: skipgram_append(), skipgram_freq(), skipgram_identify()

Examples

```
data(example_data)
extract_freetext(example_data, patient_id, min_freq = 0.6, free_text)
```

`identify_inconsistency` *Identify inconsistencies in a dataset*

Description

Tests variables for consistency between their values according to a table of rules or 'consistency table'.

Usage

```
identify_inconsistency(data = NULL, consis_tbl = NULL, id_var = NULL)
```

Arguments

`data`	data frame which will be checked for internal consistency
`consis_tbl`	data frame or tibble containing information on internal consistency rules (see "Consistency Table Requirements" section)
`id_var`	An unquoted expression which corresponds to a variable in `data` which identifies each row.

Details

Multiple types of checks for inconsistency are supported:

1. Comparing by logical operators (<, <=, ==, !=, >=, >)
2. Comparing permitted categories (e.g. cat1 in varA only if cat2 in varB)
3. Comparing permitted numeric ranges (e.g. 20-25 in varC only if 10-20 in varD)
4. Mixtures of 2 and 3 (e.g. cat1 in varA only if 20-25 in varC)
5. Constraining a single variable against fixed values (e.g. varA must be >= 70, or must lie in 0-120), by leaving the second column `NA`

The consistency tests rely on such rules being specified in a separate data frame (`consis_tbl`; see section "Consistency Table Requirements").

Variable A is given higher priority than Variable B when A is a category. If A (as char) is not equal to the value in col 4, the check is not made. This is to account for one way dependencies (i.e. VarA is fruit, VarB is apple)

Value

tibble detailing any identified internal inconsistencies in `data`, if any are found. If no inconsistencies are found, `data` is returned invisibly.

Consistency Table Requirements

Table must have exactly five character columns. The columns should be ordered according to the list below which describes the values of each column:

1. First column name of data values that will be subject to consistency checking. String. Required.
2. Second column name of data values that will be subject to consistency checking. String. Required for a two-variable check; NA for a single-variable check (see below).
3. Logical test to compare columns one and two. One of: ">",">=", "<","<=","==", "!=". String. Optional if columns 4 and 5 have non-NA values.
4. Either a single character string or a colon-separated range of numbers which should only appear in column A. Optional if column 3 has a non-NA value.
5. Either a single character string or a colon-separated range of numbers which should only appear in column B given the value/range specified in column 4. Optional if column 3 has a non-NA value.

Each row should detail one test to make. Therefore, either column 3 or columns 4 and 5 must contain non-NA values.

Single-variable checks. A rule may also constrain one variable against fixed values, rather than comparing two variables. Such a rule has NA in column 2, the boundary in column 4, and NA in column 5. Two forms are available:

- *Comparison against a value*: column 3 holds the logical test and column 4 the value to compare against. For example `"age", NA, ">=", "70", NA` requires every value of age to be at least 70.
- *Permitted values*: column 3 is NA and column 4 holds either a colon-separated numeric range or a single category. For example `"age", NA, NA, "0:120", NA` requires every value of age to lie between 0 and 120, and `"sex", NA, NA, "Male", NA` requires every value of sex to be `"Male"`.

Unlike two-variable rules, single-variable rules may use columns 3 and 4 together. Missing (NA) values are never reported as inconsistent by a single-variable check, as they cannot be assessed against a boundary.

See Also

Other internal consistency functions: validate_consistency_tbl()

Examples

```
require(tibble)
# example with synthetic dataset on number of bean counts
# there is a lot going on in the function so a simple dataset aids this example
#
# creating `data`:
beans <- tibble::tibble(red_beans = 1:15,
blue_beans = 1:15,
total_beans = 1:15*2,
red_bean_summary = c(rep("few_beans",9), rep("many_beans",6)))
```

```
#
# creating `consis_tbl`
bean_rules <- tibble::tribble(~varA, ~varB, ~lgl_test, ~varA_boundaries, ~varB_boundaries,
"red_beans", "blue_beans", "==", NA, NA,
"red_beans", "total_beans", "<=", NA,NA,
"red_beans", "red_bean_summary", NA, "1:9", "few_beans",
"red_beans", "red_bean_summary", NA, "10:15", "many_beans")

identify_inconsistency(beans, bean_rules)

# creating some inconsistencies as examples
beans[1, "red_bean_summary"] <- "many_beans"
beans[1, "red_beans"] <- 10

identify_inconsistency(beans, bean_rules)

# single-variable rules: the second column is NA and the boundary is in the
# fourth column. Here no more than 10 red beans are permitted, so the rows
# holding 11 to 15 are reported:
bean_limits <- tibble::tribble(~varA, ~varB, ~lgl_test, ~varA_boundaries, ~varB_boundaries,
"red_beans", NA, "<=", "10", NA)

identify_inconsistency(beans, bean_limits)
```

`import_dataset` *Import data into 'R'*

Description

Imports a rectangular single table into R from a .xls, .xlsx, .csv, or .tsv file.

Usage

```
import_dataset(file, format = "excel", ...)
```

Arguments

`file`	Character constant. Path to file.
`format`	Character constant. "excel" (default, for .xls or.xlsx files), csv", or "tsv".
`...`	Parameters to pass to `read_excel`, `read_csv` or `read_tsv`

Details

First row is interpreted as column headers by default. For more details see `read_excel` (.xlsx/.xls), `read_csv` (.csv), or `read_tsv` (.tsv).

Value

data as a `tibble`

See Also

read_excel for additional parameters for importing .xls or .xlsx files, read_csv for .csv files, read_tsv for .tsv files

Other import to/export from 'R' functions: export_dataset()

Examples

```
## Not run:
   # This code will not run as it requires an xlsx file
   # ./dataset.xlsx should be replaced with path to user's dataset

   # excel
   import_dataset(file = "./dataset.xlsx", format = "excel")
   #csv
   import_dataset(file = "./dataset.csv", format = "csv")
   #tsv
   import_dataset(file = "./dataset.tsv", format = "tsv")

## End(Not run)
```

---

import_var_classes *Import corrected variable classes*

Description

Reads in output of assume_var_classes, ensures all specified datatypes are one of ("id", "numeric", "double", "integer", "character", "factor", "binary", "ordinal", "ordinal_tstage", "ordinal_nstage", "genotype", "freetext", "logical", "date") as required for high level 'eHDPrep' functions.

Usage

```
import_var_classes(file = "./datatypes.csv")
```

Arguments

file	character string. Path to output of assume_var_classes which should be manually verified outside of R and corrected where any data type is incorrect.

Value

data frame containing the data type values of variables, as described in file

See Also

assume_var_classes

Examples

```
tmp = tempfile(fileext = ".csv")
data(example_data)
assume_var_classes(example_data, tmp)
import_var_classes(tmp)
```

`impute_missing_values` *Impute missing values in a dataset*

Description

Replaces missing (`NA`) values in a dataset using either simple summary statistics or k-nearest-neighbours (kNN) imputation.

Usage

```
impute_missing_values(
  data,
  ...,
  method = "auto",
  constant = NULL,
  k = 5,
  ignore = NULL
)
```

Arguments

`data`	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from dbplyr or dtplyr).
`...`	<dplyr_tidy_select> Variables (columns) to impute. If none are supplied, all variables are imputed.
`method`	Imputation method. One of `"auto"` (default), `"median"`, `"mean"`, `"mode"`, `"constant"`, or `"knn"`. See Details.
`constant`	Value used to replace missing values when `method = "constant"`. Must be of the same type as the variables being imputed so that their class is preserved (a whole number for integer variables); an incompatible value is an error. For factors, a value which is not an existing level is added as one.
`k`	Number of nearest neighbours to use when `method = "knn"` (Default: 5).
`ignore`	<dplyr_tidy_select> Variables (columns) to exclude from being used as predictors when `method = "knn"` (e.g. a row identifier). Ignored for the simple methods.

Details

Two families of imputation are provided:

Simple imputation Each selected variable (column) is imputed independently using a summary statistic of its own non-missing values.

- `"auto"` (default): numeric variables are imputed with their median and non-numeric variables with their mode (most frequent value).
- `"median"` / `"mean"`: numeric variables are imputed with the chosen statistic. Non-numeric variables fall back to the mode (a message is emitted) as means and medians are undefined for them.
- `"mode"`: all selected variables are imputed with their mode.
- `"constant"`: all selected variables have missing values replaced with `constant`.

kNN imputation (`method = "knn"`) Missing values are imputed from the `k` most similar rows (samples), using all other variables as predictors. Similarity is measured with Gower distance, which handles mixed numeric and categorical data. This method uses `kNN` from the **VIM** package, which is only required (`Suggests`) when this method is used. Unlike the simple methods, imputed values depend on each row's own profile, better preserving relationships between variables.

Only variables (columns) selected via `...` are imputed; if none are supplied all variables are eligible. For `method = "knn"`, all variables (except any supplied to `ignore`) are still used to compute distances, even if they are not themselves imputed.

Imputation never changes the class of a variable:

- Integer variables are imputed with a rounded integer, as the median or mean of a set of integers is frequently fractional. The rounding is reported via a message.
- Factors gain the imputed value as an additional level if it is not already one.
- Any other type mismatch (e.g. a character `constant` for a numeric variable) is an error rather than a silent conversion of the whole variable.

Variables which contain no non-missing values are skipped (there is no statistic to impute from) and reported as such, except under `method = "constant"` where `constant` does not depend on the observed values.

Imputation is deterministic: the same input always yields the same output. The number of values imputed per variable is reported via a message (silence with `suppressMessages`). Value-level changes can be reviewed with `review_quality_ctrl`.

Value

`data` with missing values in the selected variables imputed.

References

Kowarik, A. & Templ, M. Imputation with the R Package VIM. *Journal of Statistical Software* **74**, 1-16 (2016). doi:10.18637/jss.v074.i07

Gower, J. C. A General Coefficient of Similarity and Some of Its Properties. *Biometrics* **27**, 857-871 (1971). doi:10.2307/2528823

Little, R. J. A. & Rubin, D. B. *Statistical Analysis with Missing Data*. (John Wiley & Sons, 2019).

See Also

Other high level functionality: apply_quality_ctrl(), assess_quality(), review_quality_ctrl(), semantic_enrichment()

Examples

```
data(example_data)

# median (numeric) / mode (categorical) imputation of all variables:
res <- impute_missing_values(example_data)

# impute only specific variables:
res <- impute_missing_values(example_data, tumoursize, diabetes,
                             method = "median")

# k-nearest-neighbours imputation (requires the 'VIM' package),
# excluding the identifier from the distance calculation:

if (requireNamespace("VIM", quietly = TRUE)) {
  res <- impute_missing_values(example_data, method = "knn",
                               ignore = patient_id)
}
```

---

information_content_contin

Calculate Information Content (Continuous Variable)

Description

Calculates information content of a continuous (numeric) vector in bits (default) or natural units. Missing values are omitted from the calculation.

Usage

```
information_content_contin(x, unit = c("bits"))
```

Arguments

x	Input vector
unit	Unit to measure entropy. Either "bits" (default) of "nats".

Value

Information content of input variable

Examples

```
data(example_data)
information_content_contin(example_data$tumoursize)
```

information_content_discrete *Calculate Information Content (Discrete Variable)*

Description

Calculates information content of a discrete (categorical or ordinal) vector in bits (default) or natural units. Missing values are omitted from the calculation.

Usage

```
information_content_discrete(x, unit = c("bits"))
```

Arguments

x	Input vector
unit	Unit to measure entropy. Either "bits" (default) or "nats".

Value

Information content of input variable

Examples

```
data(example_data)
information_content_discrete(example_data$marital_status)
```

join_vars_to_ontol *Join Mapping Table to Ontology Network Graph*

Description

This function creates new nodes representing dataset variables and joins them to an input ontology network using a mapping file. Prior to joining, the information content of all nodes is calculated using node_IC_zhou.

Usage

```
join_vars_to_ontol(ontol_graph, var2entity_tbl, mode = "in", root, k = 0.5)
```

Arguments

ontol_graph	Graph containing the chosen ontology. Must be in tidygraph format or coercible to this format.
var2entity_tbl	Edge table containing dataset variable names in first column and entities in ontologies to which they are mapped in the second column.
mode	Character constant specifying the directionality of the edges. One of "in" or "out".
root	name of root node identifier in column 1 to calculate node depth from.
k	numeric value to adjust the weight of the two items of information content equation (relative number of hyponyms/descendants and relative node depth). Default = 0.5

Details

- The user-defined mappings between variables in a dataset and entities/terms in an ontology are provided in an edge table (var2entity_tbl).
- A node attribute column, node_category is generated to describe if a node is one of "Dataset Variable", "Annotation", or "Annotation Ancestor".

Value

A tidygraph resulting from the joining of var2entity_tbl and ontol_graph.

See Also

node_IC_zhou

Other semantic enrichment functions: metavariable_agg(), metavariable_info(), metavariable_variable_descenda

Examples

```
data(example_ontology)
join_vars_to_ontol(example_ontology, example_mapping_file, root = "root", mode = "in")
```

---

max_catchNAs *Find maximum of vector safely*

Description

This low-level function is deployed as part of the semantic enrichment process.Calculates maximum of values in numeric vector (ignoring NAs). If all values in input vector are NA, returns NA (rather than -Inf),

Usage

```
max_catchNAs(x)
```

Arguments

x	numeric vector

Value

maximum value of x

`mean_catchNAs` *Find mean of vector safely*

Description

This low-level function is deployed as part of the semantic enrichment process. Averages values in numeric vector (ignoring NAs). If all values in numeric vector are NA, returns NA (rather than NaN),

Usage

```
mean_catchNAs(x)
```

Arguments

x	numeric vector

Value

mean of x

`merge_cols` *Merge columns in data frame*

Description

Merges two columns in a single data frame. The merging draws on the functionality of 'dplyr''s coalesce where missing values from one vector are replaced by corresponding values in a second variable. The name of the merged variable is specified in merge_var_name. primary_var and secondary_var can be removed with rm_in_vars. Variables must be combinable (i.e. not a combination of numeric and character).

Usage

```
merge_cols(
  data,
  primary_var,
  secondary_var,
  merge_var_name = NULL,
  rm_in_vars = FALSE
)
```

Arguments

data	data frame containing primary_var and secondary_var.
primary_var	Data variable which contains the best quality / most detailed information. Missing values will be supplied by values in corresponding rows from secondary_var.
secondary_var	Data variable which will be used to fill missing values in primary_var.
merge_var_name	character constant. Name for merged variable. Default: [primary_var]_[secondary_var]_merged
rm_in_vars	logical constant. Should primary_var and secondary_var be removed? Default = FALSE.

Value

data frame with coalesced primary_var and secondary_var

See Also

coalesce

Examples

```
data(example_data)

# preserve input variables (default)
res <- merge_cols(example_data, diabetes_type, diabetes)
dplyr::select(res, dplyr::starts_with("diabetes"))

# remove input variables
res <- merge_cols(example_data, diabetes_type, diabetes, rm_in_vars = TRUE)
dplyr::select(res, dplyr::starts_with("diabetes"))
```

---

metavariable_agg *Aggregate Data by Metavariable*

Description

Variables in a numeric data frame are aggregated into metavariables via their most informative common ancestors identified in an ontological graph object (see metavariable_info). Metavariables are appended to the data frame.

Usage

```
metavariable_agg(graph, data, label_attr = "name", normalize_vals = TRUE)
```

Arguments

graph	Graph containing ontological and dataset nodes. Must be in tidygraph format or coercible to this format. Must have been processed using metavariable_info.
data	Numeric data frame or matrix containing variables which are also in graph.
label_attr	Node attribute containing labels used for column names when creating metavariable aggregations. Default: "name"
normalize_vals	Should values be normalized before aggregation? Default: TRUE

Details

Metavariables are created from the aggregation of data variables via their most informative common ancestor (expected to have been calculated in metavariable_info). Metavariables are labelled using the syntax: MV_[label_attr]_[Aggregation function]. The data variables are aggregated row-wise by their maximum, minimum, mean, sum, and product. Metavariables with zero entropy (no information) are not appended to the data. See examples for where this function should be applied in the semantic enrichment workflow.

Value

data with semantic aggregations derived from common ontological ancestry (metavariables) appended as new columns, each prefixed with "MV_" and suffixed by their aggregation function (e.g. "_SUM").

Note

A warning may be shown regarding the '.add' argument being deprecated, this is believed to be an issue with tidygraph which may be resolved in a future release: <https://github.com/thomasp85/tidygraph/issues/131>. Another warning may be shown regarding the 'neimode' argument being deprecated, this is believed to be an issue with tidygraph which may be resolved in a future release: <https://github.com/thomasp85/tidygraph/issues/15(These warning messages are not believed to have an effect on the functionality of 'eHDPrep'.

See Also

Other semantic enrichment functions: join_vars_to_ontol(), metavariable_info(), metavariable_variable_descen

Examples

```
require(magrittr)
require(dplyr)
data(example_ontology)
data(example_mapping_file)
data(example_data)

#' # define datatypes
tibble::tribble(~"var", ~"datatype",
"patient_id", "id",
"tumoursize", "numeric",
"t_stage", "ordinal_tstage",
"n_stage", "ordinal_nstage",
```

```
"diabetes_merged", "character",
"hypertension", "factor",
"rural_urban", "factor",
"marital_status", "factor",
"SNP_a", "genotype",
"SNP_b", "genotype",
"free_text", "freetext") -> data_types

# create post-QC data
example_data %>%
  merge_cols(diabetes_type, diabetes, "diabetes_merged", rm_in_vars = TRUE) %>%
  apply_quality_ctrl(patient_id, data_types,
                     bin_cats =c("No" = "Yes", "rural" = "urban"),
                     to_numeric_matrix = TRUE) %>%
                     suppressMessages() ->
                     post_qc_data

# minimal example on first four coloums of example data:
dplyr::slice(example_ontology, 1:7,24) %>%
   join_vars_to_ontol(example_mapping_file[1:3,], root = "root") %>%
   metavariable_info() %>%
   metavariable_agg(post_qc_data[1:10,1:4]) -> res
# see Note section of documentation for information on possible warnings.

# summary of result:
tibble::glimpse(res)


# full example:
example_ontology %>%
   join_vars_to_ontol(example_mapping_file, root = "root") %>%
   metavariable_info() %>%
   metavariable_agg(post_qc_data) -> res
 # see Note section of documentation for information on possible warnings.

# summary of result:
tibble::glimpse(res)
```

`metavariable_info` *Compute Metavariable Information*

Description

Calculates attributes for each node in a graph object pertaining to their suitability and rank as metavariables; primarily if they are the most informative common ancestor (see `node_IC_zhou`) of a set of nodes representing a dataset variable.

Usage

```
metavariable_info(graph, mode = "in", IC_threshold = 0)
```

Arguments

graph	Graph containing ontological and dataset nodes. Must be in tidygraph format or coercible to this format.
mode	Character constant specifying the directionality of the edges. One of: "in" or "out".
IC_threshold	Metavariables with IC less than this value will be omitted from output. Default = 0 (no omission).

Details

The added attributes are:

min_dist_to_var Integer. The minimum distance of an ontology node in the graph to a node representing a dataset variable.

is_metavariable Logical. If the node has at least two descendants in the graph which represent dataset variables.

variable_descendants List. The names of variables of which a node is an ancestor.

variable_set Integer. An identifier for the unique set of descendants in the graph which represent dataset variables. The assigned number corresponds to the order in which a unique set was identified when scanning through the node table.

highest_IC Logical. If the node possesses the highest information content of all other nodes which are common ancestors of the same variable set. Information content is expected to have been calculated in join_vars_to_ontol.

Value

A modified graph object with additional node attributes pertaining to their status as a metavariable.

See Also

node_IC_zhou

Other semantic enrichment functions: join_vars_to_ontol(), metavariable_agg(), metavariable_variable_descend

Examples

```
data(example_ontology)
require(magrittr)
example_ontology %>%
join_vars_to_ontol(example_mapping_file, root = "root") -> joined_ontol

metavariable_info(joined_ontol)
```

`metavariable_variable_descendants` *Extract metavariables' descendant variables*

Description

Formats the output of `metavariable_info` for easier interpretation of each metavariable's descendant variables

Usage

```
metavariable_variable_descendants(metavariable_info_output)
```

Arguments

`metavariable_info_output`	Output tibble of `metavariable_info`

Details

Not part of the standard semantic enrichment pipeline as this function just produces a simplified version of the output of `metavariable_info`.

The output of `metavariable_info` is converted to a tibble, filtered to only include metavariables with highest information content for the variable set. The tibble has three columns describing a metavariable, its information content, and its descendant variables.

Value

A tibble describing each metavariable, its information content, and its descendant variables

See Also

`node_IC_zhou`

Other semantic enrichment functions: `join_vars_to_ontol()`, `metavariable_agg()`, `metavariable_info()`

Examples

```
data(example_ontology)
require(magrittr)
example_ontology %>%
join_vars_to_ontol(example_mapping_file, root = "root") -> joined_ontol

mv_info <- metavariable_info(joined_ontol)
metavariable_variable_descendants(mv_info)
```

min_catchNAs *Find minimum of vector safely*

Description

This low-level function is deployed as part of the semantic enrichment process. Calculates minimum of values in numeric vector (ignoring NAs). If all values in numeric vector are NA, returns NA (rather than Inf),

Usage

```
min_catchNAs(x)
```

Arguments

x	numeric vector

Value

minimum value of x

mi_content_discrete *Calculate Mutual Information Content*

Description

Calculates mutual information content between two variables in bits. Missing values are omitted from the calculation.

Usage

```
mi_content_discrete(x, y)
```

Arguments

x	First variable
y	Second variable

Value

Mutual information content of x and y

Examples

```
data(example_data)
mi_content_discrete(example_data$diabetes, example_data$diabetes_type)
```

mod_track *Data modification tracking*

Description

This function produces a table where each row represents a value in a variable which is present in the cleaned dataset and which has been modified. The identifier, original and modified value, modification type, and variable names in the original and modified datasets are recorded.

Usage

```
mod_track(before_tbl, after_tbl, id_var, plot = FALSE, vars2compare)
```

Arguments

before_tbl	Data frame from before modifications were made.
after_tbl	Data frame from after modifications were made.
id_var	An unquoted expression which corresponds to a variable in both before_tbl and after_tbl which identifies each row. Required.
plot	Should a plot be returned instead of a table of results? Default: FALSE.
vars2compare	Character vectors of variable names to compare.

Value

Table containing row-level modification records or plot summarising modifications.

Examples

```
# merge data as the example modification

require(magrittr)

 # example with one modification type (removal)
 # return table
  mod_track(example_data, strings_to_NA(example_data), patient_id)

 # return plot
  mod_track(example_data, strings_to_NA(example_data), patient_id, plot = TRUE)

 # example with multiple modification types (removal, substitution and addition)
example_data %>%
   strings_to_NA() %>%
   merge_cols(diabetes_type, diabetes) ->
   modded_data

# return table
mod_track(example_data, modded_data, patient_id, vars2compare = c("t_stage",
"diabetes_type_diabetes_merged" = "diabetes", "diabetes_type_diabetes_merged"
```

```
= "diabetes_type"), plot = FALSE)

# return plot
mod_track(example_data, modded_data, patient_id, vars2compare = c("t_stage",
"diabetes_type_diabetes_merged" = "diabetes", "diabetes_type_diabetes_merged"
= "diabetes_type"), plot = TRUE)
```

---

node_IC_zhou *Calculate Node Information Content (Zhou et al 2008 method)*

Description

Computes the information content for each node in a directed graph according to the equation developed by Zhou *et al.* (2008).

Usage

```
node_IC_zhou(graph, mode = "in", root, k = 0.5)
```

Arguments

graph	tidygraph directed graph.
mode	Character constant specifying the directionality of the edges. One of "in" or "out".
root	name of root node identifier in column 1 to calculate node depth from.
k	numeric value to adjust the weight of the two items of information content equation (relative number of hyponyms/descendants and relative node depth). Default = 0.5

Value

tidygraph with additional node attribute "information_content"

Note

For use in semantic enrichment, this should be applied before joining an ontology with nodes representing data variables (i.e. before applying join_vars_to_ontol.

References

Zhou, Z., Wang, Y. & Gu, J. A New Model of Information Content for Semantic Similarity in WordNet. in 2008 Second International Conference on Future Generation Communication and Networking Symposia vol. 3 85–89 (2008).

Examples

```
data(example_ontology)
node_IC_zhou(example_ontology, mode = "in", root = "root")
```

normalize *Min max normalization*

Description

Normalizes values in x to be between 0 and 1 using min-max normalization.

Usage

```
normalize(x, na.rm = TRUE)
```

Arguments

x	numeric vector
na.rm	a logical indicating whether missing values should be removed. Default = TRUE.

Value

normalised x

nums_to_NA *Replace numeric values in numeric columns with NA*

Description

Replaces specified numbers in numeric columns with NA.

Usage

```
nums_to_NA(data, ..., nums_to_replace = NULL)
```

Arguments

data	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from dbplyr or dtplyr).
...	<tidy-select> One or more unquoted expressions separated by commas. Variable names can be used as if they were positions in the data frame, so expressions like x:y can be used to select a range of variables.
nums_to_replace	numeric vector of values to be replaced with NA. Case is ignored.

Details

Columns to process can be specified in ... or the function will be applied to all numeric columns.

Value

data with specified values replaced with NA

Examples

```
data(example_data)

# replace all 1,2, and 3 from tumoursize and patient_id with NA.
nums_to_NA(data = example_data, tumoursize, patient_id, nums_to_replace = c(1,2,3))
```

---

onehot_vec *One hot encode a vector*

Description

Uses one-hot encoding to convert nominal vectors to a tibble containing variables for each of the unique values in input vector.

Usage

```
onehot_vec(x, prefix)
```

Arguments

x	non-numeric vector
prefix	prefix to append to output variable names

Value

tibble

---

ordinal_label_levels *Extract labels and levels of ordinal variables in a dataset*

Description

This function enables preservation of the text labels for ordinal variables in a dataset in preparation for conversion to a numeric matrix. A table is produced which retains the mappings between the text labels and the numerical labels for future reference.

Usage

```
ordinal_label_levels(data, out_path = NULL)
```

Arguments

data	data frame with ordinal variables with labels and levels to be extracted.
out_path	Optional string. Path to write output to. If not supplied, R object will be returned.

Value

Tibble of text label and (numerical) level mappings

Examples

```
require(magrittr)  # for %>%

# create an example class_tbl object
# note that diabetes_type is classed as ordinal yet is not modified as its
# levels are not pre-coded. It should instead be encoded with encode_ordinals().
tibble::tribble(~"var", ~"datatype",
"patient_id", "id",
"tumoursize", "numeric",
"t_stage", "ordinal_tstage",
"n_stage", "ordinal_nstage",
"diabetes", "factor",
"diabetes_type", "ordinal",
"hypertension", "factor",
"rural_urban", "factor",
"marital_status", "factor",
"SNP_a", "genotype",
"SNP_b", "genotype",
"free_text", "freetext") -> data_types

# show unqiue values for t_stage in pre-QC example_data
unique(example_data$t_stage)

# apply quality control to example_data
apply_quality_ctrl(example_data, patient_id, data_types,
bin_cats =c("No" = "Yes", "rural" = "urban"),  min_freq = 0.6) %>%
ordinal_label_levels -> res

# examine the labels and levels of t_stage in post-QC example_data
dplyr::filter(res, variable == "t_stage")
```

---

plot_completeness *Plot Completeness of a Dataset*

Description

Generates a bar plot of percentage completeness for one or both data frame dimensions (rows/columns).

Usage

```
plot_completeness(data, id_var, plot = c("variables", "rows"))
```

Arguments

data	Data frame in tidy format (see https://tidyr.tidyverse.org/).
id_var	Row identifier variable name.
plot	Character vector containing one or both of variables and rows.

Value

Completeness bar plot.

See Also

Other measures of completeness: assess_completeness(), compare_completeness(), completeness_heatmap(), row_completeness(), variable_completeness()

Examples

```
data(example_data)
plot_completeness(example_data, patient_id, "variables")
```

---

prod_catchNAs *Find product of vector safely*

Description

This low-level function is deployed as part of the semantic enrichment process. Calculates product of values in numeric vector (ignoring NAs). If all values in numeric vector are NA, returns NA (rather than Inf),

Usage

```
prod_catchNAs(x)
```

Arguments

x	numeric vector

Value

product of x

report_var_mods *Track changes to dataset variables*

Description

Reports if variables have been added, removed, or are preserved between two data frames. Intended to be used to review quality control / data preparation.

Usage

```
report_var_mods(before_tbl = NULL, after_tbl = NULL)
```

Arguments

before_tbl	Data frame from before modifications were made.
after_tbl	Data frame from after modifications were made.

Value

Tibble containing two columns. ‘variable‘ contains name of each variable. ‘presence‘ contains the presence of the variable in after_tbl.

Examples

```
example_data_merged <- merge_cols(example_data, diabetes_type,
diabetes, "diabetes_merged", rm_in_vars = TRUE)

report_var_mods(example_data, example_data_merged)
```

review_quality_ctrl *Review Quality Control*

Description

Provides information on modifications made to a dataset at both variable (column) and value (sample) levels, designed for review of quality control measures.

Usage

```
review_quality_ctrl(before_tbl, after_tbl, id_var)
```

Arguments

before_tbl	Data frame from before modifications were made.
after_tbl	Data frame from after modifications were made.
id_var	An unquoted expression which corresponds to a variable in both before_tbl and after_tbl which identifies each row. Required.

Details

Modifications are identified by comparing the original and modified dataset.

QC review functions are applied in the following order:

1. Variable-level modifications (report_var_mods)
2. Value-level modifications (mod_track)
3. Value-level modifications (plot) (mod_track)

A list containing each of these functions' outputs is returned.

Value

List containing data for review of quality control

See Also

Other high level functionality: apply_quality_ctrl(), assess_quality(), impute_missing_values(), semantic_enrichment()

Examples

```
data(example_data)
require(tibble)

tibble::tribble(~"var", ~"datatype",
"patient_id", "id",
"tumoursize", "numeric",
"t_stage", "ordinal_tstage",
"n_stage", "ordinal_nstage",
"diabetes", "factor",
"diabetes_type", "ordinal",
"hypertension", "factor",
"rural_urban", "factor",
"marital_status", "factor",
"SNP_a", "genotype",
"SNP_b", "genotype",
"free_text", "freetext") -> data_types


# create QC'ed dataset
post_QC_example_data <- apply_quality_ctrl(example_data,
                                           patient_id,
                                           data_types,
                                           bin_cats =c("No" = "Yes",
                                                       "rural" = "urban"),
                                           min_freq = 0.6)

# review QC
QC_review <- review_quality_ctrl(before_tbl = example_data,
                     after_tbl = post_QC_example_data,
                     id_var = patient_id)
```

```
# view variable level changes
QC_review$variable_level_changes

# view value level changes
QC_review$value_level_changes

# view value level changes as a plot
QC_review$value_level_changes_plt
```

`row_completeness` *Calculate Row Completeness in a Data Frame*

Description

Calculates the completeness of each row/observation in a data frame.

Usage

```
row_completeness(data, id_var)
```

Arguments

`data`	Data frame.
`id_var`	Row identifier variable.

Details

Row completeness is measured by comparing the number of `NA` to non-`NA` values. Returns the count of `NA` as well as the percentage of `NA` values and the percentage completeness.

Value

Tibble detailing completeness statistics for each row in input data.

See Also

Other measures of completeness: `assess_completeness()`, `compare_completeness()`, `completeness_heatmap()`, `plot_completeness()`, `variable_completeness()`

Examples

```
data(example_data)
row_completeness(example_data, patient_id)
```

`semantic_enrichment` *Semantic enrichment*

Description

Enriches a dataset with additional (meta-)variables derived from the semantic commonalities between variables (columns).

Usage

```
semantic_enrichment(
  data,
  ontology,
  mapping_file,
  mode = "in",
  root,
  label_attr = "name",
  ...
)
```

Arguments

`data`	Required. Numeric data frame or matrix containing variables present in the mapping file.
`ontology`	Required. One of: • Path to ontology edge table in .csv format (String) • Edge table in data frame format • Graph containing the chosen ontology - must be in `tidygraph` format or coercible to this format. .
`mapping_file`	Required. Path to csv file or data frame containing mapping information. Should contain two columns only. The first column should contain column names, present in the data frame. The second column should contain the name of entities present in the ontology object.
`mode`	Character constant specifying the directionality of the edges. One of: "in" or "out".
`root`	Required. Name of root node identifier in column 1 to calculate node depth from.
`label_attr`	Node attribute containing labels used for column names when creating metavariable aggregations. Default: "name"
`...`	additional arguments to pass to `read_csv` when reading ‘mapping_file‘.

Details

Semantic enrichment generates meta-variables from the aggregation of data variables (columns) via their most informative common ancestor. Meta-variables are labelled using the syntax: `MV_[label_attr]_[Aggregation function]`. The data variables are aggregated row-wise by their maximum, minimum, mean, sum, and product. Meta-variables with zero entropy (no information) are not appended to the data. See the "Semantic Enrichment" section in the vignette of 'eHDPrep' for more information: `vignette("Introduction_to_eHDPrep", package = "eHDPrep")`

Value

Semantically enriched dataset

Note

A warning may be shown regarding the '.add' argument being deprecated, this is believed to be an issue with 'tidygraph' which may be resolved in a future release: <https://github.com/thomasp85/tidygraph/issues/131>. Another warning may be shown regarding the 'neimode' argument being deprecated, this is believed to be an issue with 'tidygraph' which may be resolved in a future release: <https://github.com/thomasp85/tidygraph/issues/15
These warning messages are not believed to have an effect on the functionality of 'eHDPrep'.

See Also

Other high level functionality: `apply_quality_ctrl()`, `assess_quality()`, `impute_missing_values()`, `review_quality_ctrl()`

Examples

```
require(magrittr)
require(dplyr)
data(example_ontology)
data(example_mapping_file)
data(example_data)

#' # define datatypes
tibble::tribble(~"var", ~"datatype",
"patient_id", "id",
"tumoursize", "numeric",
"t_stage", "ordinal_tstage",
"n_stage", "ordinal_nstage",
"diabetes_merged", "character",
"hypertension", "factor",
"rural_urban", "factor",
"marital_status", "factor",
"SNP_a", "genotype",
"SNP_b", "genotype",
"free_text", "freetext") -> data_types

# create post-QC data
example_data %>%
  merge_cols(diabetes_type, diabetes, "diabetes_merged", rm_in_vars = TRUE) %>%
  apply_quality_ctrl(patient_id, data_types,
```

```
                    bin_cats =c("No" = "Yes", "rural" = "urban"),
                    to_numeric_matrix = TRUE) %>%
                    suppressMessages() ->
                    post_qc_data

# minimal example on first four coloums of example data:
semantic_enrichment(post_qc_data[1:10,1:4],
                    dplyr::slice(example_ontology, 1:7,24),
                    example_mapping_file[1:3,], root = "root") -> res
# see Note section of documentation for information on possible warnings.

# summary of result:
tibble::glimpse(res)


# full example:
 res <- semantic_enrichment(post_qc_data, example_ontology,
 example_mapping_file, root = "root")
 # see Note section of documentation for information on possible warnings.
```

---

skipgram_append *Append Skipgram Presence Variables to Dataset*

Description

Adds new variables to `data` which report the presence of skipgrams (either those specified in `skipgrams2append` or, if not specified, skipgrams with a minimum frequency (`min_freq`, default = 1)).

Usage

```
skipgram_append(skipgram_tokens, skipgrams2append, data, id_var, min_freq = 1)
```

Arguments

`skipgram_tokens`	Output of `skipgram_identify`.
`skipgrams2append`	Which skipgrams in `skipgram_tokens` to append to dataset.
`data`	Data frame to append skipgram variables to.
`id_var`	An unquoted expression which corresponds to a variable in `data` which identifies each row.
`min_freq`	Minimum percentage frequency of skipgram occurrence to return. Default = 1.

Value

`data` with additional variables describing presence of skipgrams

References

Guthrie, D., Allison, B., Liu, W., Guthrie, L. & Wilks, Y. A Closer Look at Skip-gram Modelling. in Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06) (European Language Resources Association (ELRA), 2006).

Benoit K, Watanabe K, Wang H, Nulty P, Obeng A, Müller S, Matsuo A (2018). "quanteda: An R package for the quantitative analysis of textual data." _Journal of Open Source Software_, *3*(30), 774. doi:10.21105/joss.00774 <https://doi.org/10.21105/joss.00774>, <https://quanteda.io>.

Feinerer I, Hornik K (2020). _tm: Text Mining Package_. R package version 0.7-8, <https://CRAN.R-project.org/package=tm>.

Ingo Feinerer, Kurt Hornik, and David Meyer (2008). Text Mining Infrastructure in R. Journal of Statistical Software 25(5): 1-54. URL: https://www.jstatsoft.org/v25/i05/.

See Also

Principle underlying function: `tokens_ngrams`

Other free text functions: `extract_freetext()`, `skipgram_freq()`, `skipgram_identify()`

Examples

```
data(example_data)
# identify skipgrams
toks_m <- skipgram_identify(x = example_data$free_text,
                            ids = example_data$patient_id,
                            max_interrupt_words = 5)
# add skipgrams by minimum frequency
skipgram_append(toks_m,
                id_var = patient_id,
                min_freq = 0.6,
                data = example_data)
# add specific skipgrams
skipgram_append(toks_m,
                id_var = patient_id,
                skipgrams2append = c("sixteen_week", "bad_strain"),
                data = example_data)
```

---

skipgram_freq *Report Skipgram Frequency*

Description

Measures the frequency of skipgrams (non-contiguous words in free text), reported in a tibble. Frequency is reported as both counts and percentages.

Usage

```
skipgram_freq(skipgram_tokens, min_freq = 1)
```

Arguments

skipgram_tokens	Output of skipgram_identify.
min_freq	Minimum skipgram percentage frequency of occurrence to retain. Default = 1.

Value

Data frame containing frequency of skipgrams in absolute count and relative to the length of input variable.

References

Guthrie, D., Allison, B., Liu, W., Guthrie, L. & Wilks, Y. A Closer Look at Skip-gram Modelling. in Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06) (European Language Resources Association (ELRA), 2006).

Benoit K, Watanabe K, Wang H, Nulty P, Obeng A, Müller S, Matsuo A (2018). "quanteda: An R package for the quantitative analysis of textual data." _Journal of Open Source Software_, *3*(30), 774. doi:10.21105/joss.00774 <https://doi.org/10.21105/joss.00774>, <https://quanteda.io>.

Feinerer I, Hornik K (2020). _tm: Text Mining Package_. R package version 0.7-8, <https://CRAN.R-project.org/package=tm>.

Ingo Feinerer, Kurt Hornik, and David Meyer (2008). Text Mining Infrastructure in R. Journal of Statistical Software 25(5): 1-54. URL: https://www.jstatsoft.org/v25/i05/.

See Also

Principle underlying function: tokens_ngrams

Other free text functions: extract_freetext(), skipgram_append(), skipgram_identify()

Examples

```
data(example_data)
toks_m <- skipgram_identify(x = example_data$free_text,
                            ids = example_data$patient_id,
                            max_interrupt_words = 5)
skipgram_freq(toks_m, min_freq = 0.5)
```

---

skipgram_identify *Identify Neighbouring Words (Skipgrams) in a free-text vector*

Description

Identifies words which appear near each other in the free-text variable (var), referred to as "Skipgrams". Supported languages for stop words and stemming are danish, dutch, english, finnish, french, german, hungarian, italian, norwegian, portuguese, russian, spanish, and swedish.

Usage

```
skipgram_identify(
  x,
  ids,
  num_of_words = 2,
  max_interrupt_words = 2,
  words_to_rm = NULL,
  lan = "english"
)
```

Arguments

x	Free-text character vector to query.
ids	Character vector containing IDs for each element of var.
num_of_words	Number of words to consider for each returned skipgram. Default = 2.
max_interrupt_words	Maximum number of words which can interrupt proximal words. Default = 2.
words_to_rm	Character vector of words which should not be considered.
lan	Language of var. Default: english.

Value

Tibble containing skipgrams as variables and patient values as rows.

References

Guthrie, D., Allison, B., Liu, W., Guthrie, L. & Wilks, Y. A Closer Look at Skip-gram Modelling. in Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06) (European Language Resources Association (ELRA), 2006).

Benoit K, Watanabe K, Wang H, Nulty P, Obeng A, Müller S, Matsuo A (2018). "quanteda: An R package for the quantitative analysis of textual data." _Journal of Open Source Software_, *3*(30), 774. doi:10.21105/joss.00774 <https://doi.org/10.21105/joss.00774>, <https://quanteda.io>.

Feinerer I, Hornik K (2020). _tm: Text Mining Package_. R package version 0.7-8, <https://CRAN.R-project.org/package=tm>.

Ingo Feinerer, Kurt Hornik, and David Meyer (2008). Text Mining Infrastructure in R. Journal of Statistical Software 25(5): 1-54. URL: https://www.jstatsoft.org/v25/i05/.

See Also

Principle underlying function: tokens_ngrams

Other free text functions: extract_freetext(), skipgram_append(), skipgram_freq()

Examples

```
data(example_data)
skipgram_identify(x = example_data$free_text,
                  ids = example_data$patient_id,
                  max_interrupt_words = 5)
```

`strings_to_NA` *Replace values in non-numeric columns with NA*

Description

Replaces specified or pre-defined strings in non-numeric columns with NA.

Usage

```
strings_to_NA(data, ..., strings_to_replace = NULL)
```

Arguments

data	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from dbplyr or dtplyr).
...	<tidy-select> One or more unquoted expressions separated by commas. Variable names can be used as if they were positions in the data frame, so expressions like x:y can be used to select a range of variables.
strings_to_replace	character vector of values to be replaced with NA.

Details

Columns to process can be specified in custom arguments (...) or will be applied to all non-numeric columns. Default strings which will be replaced with NA are as follows: "Undetermined", "unknown", "missing", "fail", "fail / unknown", "equivocal", "equivocal / unknown", "*". String search is made using grepl and supports regex so metacharacters (. \ | ( ) [ ] { } ^ $ * + ? $) should be escaped with a "\" prefix. Matches are case sensitive by default but can ignore case with the parameter: ignore.case = TRUE in ...).

Value

data with specified values replaced with NA.

Examples

```
data(example_data)

# original unique values in diabetes column:
unique(example_data$diabetes)
# Using default values
res <- strings_to_NA(example_data)
```

```
unique(res$diabetes)


# original unique values in diabetes_type column:
unique(example_data$diabetes_type)
# Using custom values
res <- strings_to_NA(example_data, strings_to_replace = "Type I")
unique(res$diabetes_type)
```

`sum_catchNAs` *Sum vector safely for semantic enrichment*

Description

sums values in x (ignoring NAs). If all values in x are NA, returns NA (rather than 0),

Usage

```
sum_catchNAs(x)
```

Arguments

x	numeric vector

Value

sum of x

`validate_consistency_tbl` *Validate internal consistency table*

Description

Runs a series of checks on a table of internal consistency rules (see Consistency Table Requirements) in preparation for `identify_inconsistency`.

Usage

```
validate_consistency_tbl(data, consis_tbl)
```

Arguments

data	data frame which will be checked for internal consistency
consis_tbl	data frame or tibble containing information on internal consistency rules (see "Consistency Table Requirements" section)

Value

Error message or successful validation message is printed. The dataset is returned invisibly.

Consistency Table Requirements

Table must have exactly five character columns. The columns should be ordered according to the list below which describes the values of each column:

1. First column name of data values that will be subject to consistency checking. String. Required.
2. Second column name of data values that will be subject to consistency checking. String. Required for a two-variable check; NA for a single-variable check (see below).
3. Logical test to compare columns one and two. One of: ">","<=", "<","<=","==", "!=". String. Optional if columns 4 and 5 have non-NA values.
4. Either a single character string or a colon-separated range of numbers which should only appear in column A. Optional if column 3 has a non-NA value.
5. Either a single character string or a colon-separated range of numbers which should only appear in column B given the value/range specified in column 4. Optional if column 3 has a non-NA value.

Each row should detail one test to make. Therefore, either column 3 or columns 4 and 5 must contain non-NA values.

Single-variable checks. A rule may also constrain one variable against fixed values, rather than comparing two variables. Such a rule has NA in column 2, the boundary in column 4, and NA in column 5. Two forms are available:

- *Comparison against a value*: column 3 holds the logical test and column 4 the value to compare against. For example `"age", NA, ">=", "70", NA` requires every value of age to be at least 70.
- *Permitted values*: column 3 is NA and column 4 holds either a colon-separated numeric range or a single category. For example `"age", NA, NA, "0:120", NA` requires every value of age to lie between 0 and 120, and `"sex", NA, NA, "Male", NA` requires every value of sex to be `"Male"`.

Unlike two-variable rules, single-variable rules may use columns 3 and 4 together. Missing (NA) values are never reported as inconsistent by a single-variable check, as they cannot be assessed against a boundary.

See Also

Other internal consistency functions: identify_inconsistency()

Examples

```
require(tibble)
# example with synthetic dataset on number of bean counters
# there is a lot going on in the function so a simple dataset aids this example
#
```

```
# creating `data`:
beans <- tibble::tibble(red_beans = 1:15,
blue_beans = 1:15,
total_beans = 1:15*2,
red_bean_summary = c(rep("few_beans",9), rep("many_beans",6)))
#
# creating `consis_tbl`
bean_rules <- tibble::tribble(~varA, ~varB, ~lgl_test, ~varA_boundaries, ~varB_boundaries,
"red_beans", "blue_beans", "==", NA, NA,
"red_beans", "total_beans", "<=", NA,NA,
"red_beans", "red_bean_summary", NA, "1:9", "few_beans",
"red_beans", "red_bean_summary", NA, "10:15", "many_beans")

validate_consistency_tbl(beans, bean_rules)

# single-variable rules constrain one variable against fixed values.
# The second column is NA and the boundary is given in the fourth column:
bean_limits <- tibble::tribble(~varA, ~varB, ~lgl_test, ~varA_boundaries, ~varB_boundaries,
"red_beans", NA, ">=", "1", NA,
"total_beans", NA, NA, "0:30", NA,
"red_bean_summary", NA, NA, "few_beans", NA)

validate_consistency_tbl(beans, bean_limits)
```

`validate_mapping_tbl` *Validate mapping table for semantic enrichment*

Description

Applies tests to a mapping table to ensure it is valid for use with the data frame and ontological graph, in preparation for semantic enrichment.

Usage

```
validate_mapping_tbl(mapping_tbl, data, ontol_graph)
```

Arguments

`mapping_tbl`	data frame. Contains two columns. First column contains variable names of a primary dataset. Second column contains entities in an ontological graph to which the primary dataset's variable names are mapped.
`data`	data frame. Primary dataset which contains variable names referred to in first column of the mapping table
`ontol_graph`	ontological graph which contains entity names/IDs referred to in second column of the mapping table

Value

Any warnings and the mapping table returned invisibly

validate_ontol_nw *Validate ontology network for semantic enrichment*

Description

Performs tests on a graph object in preparation for semantic enrichment.

Usage

```
validate_ontol_nw(graph)
```

Arguments

graph	graph object to validate.

Details

The tests are:

1. Is graph coercible to tidygraph format?
2. Is graph directed?
3. Does graph contains one component (is one ontology)?

Value

input graph or validation errors

variable.bw.kde *Variable bandwidth Kernel Density Estimation*

Description

Calculates variable bandwidth KDE using Abramson's two stage estimator.

Usage

```
variable.bw.kde(x, output.domain = x, na.rm = FALSE, adjust.factor = 0.5)
```

Arguments

x	A numeric vector of values for estimating density
output.domain	The domain of values over which to estimate the density. Defaults to x. To use the same domain of x values as R's density, set to NULL.
na.rm	Remove missing values if TRUE
adjust.factor	A scaling factor (exponent) applied to the variable bandwidth calculation. Larger factors result in greater deviation from the fixed bandwidth (a value of 0 gives the fixed bandwidth case).

Details

Bandwidth is first calculated using Silverman's estimator, then refined in a second stage to allow local bandwidth variations in the data based on the initial estimate.

Value

The kernel density estimate as a `density` object, compatible with R's `density` function.

Author(s)

Alexander Lyulph Robert Lubbock, Ian Overton

References

Abramson, I. S. On Bandwidth Variation in Kernel Estimates-A Square Root Law. Ann. Statist. 10, 1217-1223 (1982).

`variable_completeness` *Calculate Variable Completeness in a Data Frame*

Description

Calculates the completeness of each variable in a data frame.

Usage

```
variable_completeness(data)
```

Arguments

`data`	Data frame.

Details

This is achieved by comparing the number of `NA` to non-`NA` values. Returns the count of `NA` as well as the percentage of `NA` values and the percentage completeness.

Value

`Tibble` detailing completeness statistics for each variable.

See Also

Other measures of completeness: `assess_completeness()`, `compare_completeness()`, `completeness_heatmap()`, `plot_completeness()`, `row_completeness()`

Examples

```
data(example_data)
variable_completeness(example_data)
```

variable_entropy *Calculate Entropy of Each Variable in Data Frame*

Description

Calculates Shannon entropy of all variables in a data frame in bits (default) or natural units. Missing values are omitted from the calculation.

Usage

```
variable_entropy(data, unit = "bits")
```

Arguments

data	Data Frame to compute on
unit	Unit to measure entropy. Either "bits" (default) or "nats".

Value

Named numeric vector containing entropy values

References

Shannon, C. E. A mathematical theory of communication. The Bell System Technical Journal 27, 379–423 (1948).

Examples

```
a <- matrix(c(c(1,1,1,1,1,1, 1,2,3,4,5,6)),ncol = 2, dimnames =
list(seq(1,6), c("no_entropy","entropy")))
variable_entropy(as.data.frame(a))
```

warn_missing_dots *Missing dots warning*

Description

Internal function. Warns if dots (...) argument have not been supplied

Usage

```
warn_missing_dots(test)
```

Arguments

test	expression to test.

Value

warning to user that no values were modified

`zero_entropy_variables` *Identify variables with zero entropy*

Description

Calculates Shannon entropy of variables in a data frame in bits (default) or natural units. Missing values are omitted from the calculation. Names of variables with zero entropy are returned.

Usage

```
zero_entropy_variables(data, unit = "bits")
```

Arguments

`data`	Data Frame to compute on
`unit`	Unit to measure entropy. Either "bits" (default) or "nats".

Value

Character vector of variable names with zero entropy

References

Shannon, C. E. A mathematical theory of communication. The Bell System Technical Journal 27, 379–423 (1948).

Examples

```
data(example_data)
zero_entropy_variables(example_data)
```

Index

∗ **datasets**
example_data, 29
example_edge_tbl, 30
example_mapping_file, 31
example_ontology, 31
∗ **free text functions**
extract_freetext, 32
skipgram_append, 61
skipgram_freq, 62
skipgram_identify, 63
∗ **high level functionality**
apply_quality_ctrl, 3
assess_quality, 7
impute_missing_values, 38
review_quality_ctrl, 56
semantic_enrichment, 59
∗ **import to/export from 'R' functions**
export_dataset, 32
import_dataset, 36
∗ **internal consistency functions**
identify_inconsistency, 34
validate_consistency_tbl, 66
∗ **measures of completeness**
assess_completeness, 5
compare_completeness, 13
completeness_heatmap, 16
plot_completeness, 54
row_completeness, 58
variable_completeness, 70
∗ **networks**
discrete.mi, 19
∗ **semantic enrichment functions**
join_vars_to_ontol, 41
metavariable_agg, 44
metavariable_info, 46
metavariable_variable_descendants, 48

apply_quality_ctrl, 3
apply_quality_ctrl(), *8*, *40*, *57*, *60*
assess_completeness, 5
assess_completeness(), *13*, *17*, *55*, *58*, *70*
assess_quality, 7
assess_quality(), *5*, *40*, *57*, *60*
assume_var_classes, *4*, 9, *37*

cell_spec, *11*
cellspec_lgl, 10
coalesce, *43, 44*
coerce_dates, 11
coerce_numeric_vars, *12*, 12
compare_completeness, 13
compare_completeness(), *6*, *17*, *55*, *58*, *70*
compare_info_content, 14, *15*
compare_info_content_plt, *14*, 15
completeness_heatmap, *6, 7*, 16
completeness_heatmap(), *6*, *13*, *55*, *58*, *70*
count_compare, 17

Date, *11*
discrete.mi, 19
distant_neg_val, *16*, 20
dplyr_tidy_select, *11, 12*, *38*

edge_tbl_to_graph, 20
encode_as_num_mat, *4*, 21
encode_bin_cat_vec, 23
encode_binary_cats, *4*, 22
encode_cats, *4*, 24
encode_genotype_vec, 26
encode_genotypes, *4*, 25
encode_ordinals, *4*, 27
entropy, 28
exact.kde, 28
example_data, 29
example_edge_tbl, 30
example_mapping_file, 31
example_ontology, 31
export_dataset, 32
export_dataset(), *37*

extract_freetext, *4*, 32
extract_freetext(), *62–64*

grepl, *65*

identify_inconsistency, *7*, 34, *66*
identify_inconsistency(), *67*
import_dataset, 36
import_dataset(), *32*
import_var_classes, *4*, *9, 10*, 37
impute_missing_values, *4*, 38
impute_missing_values(), *5*, *8*, *57*, *60*
information_content_contin, 40
information_content_discrete, *14*, 41

join_vars_to_ontol, 41, *47*, *51*
join_vars_to_ontol(), *45*, *47, 48*

kable, *11*
kNN, *39*

max_catchNAs, 42
mean_catchNAs, 43
merge_cols, 43
metavariable_agg, 44
metavariable_agg(), *42*, *47, 48*
metavariable_info, *44, 45*, 46, *48*
metavariable_info(), *42*, *45*, *48*
metavariable_variable_descendants, 48
metavariable_variable_descendants(), *42*, *45*, *47*
mi_content_discrete, *14*, 49
min_catchNAs, 49
mod_track, 50, *57*

node_IC_zhou, *41*, *46–48*, 51
normalize, 52
nums_to_NA, 52

onehot_vec, 53
ordinal_label_levels, 53

pheatmap, *16, 17*
plot_completeness, *6, 7*, 54
plot_completeness(), *6*, *13*, *17*, *58*, *70*
prod_catchNAs, 55

read_csv, *36, 37*, *59*
read_excel, *36, 37*
read_tsv, *36, 37*

regex, *65*
report_var_mods, 56, *57*
review_quality_ctrl, *39*, 56
review_quality_ctrl(), *5*, *8*, *40*, *60*
row_completeness, *6, 7*, 58
row_completeness(), *6*, *13*, *17*, *55*, *70*

semantic_enrichment, 59
semantic_enrichment(), *5*, *8*, *40*, *57*
skipgram_append, 61
skipgram_append(), *33*, *63, 64*
skipgram_freq, 62
skipgram_freq(), *33*, *62*, *64*
skipgram_identify, *61*, *63*, 63
skipgram_identify(), *33*, *62, 63*
strings_to_NA, *4*, *24*, 65
strptime, *11*
sum_catchNAs, 66
suppressMessages, *39*

tidygraph, *20*, *42*, *45*, *47*, *51*, *59*, *69*
tokens_ngrams, *33*, *62–64*

validate_consistency_tbl, 66
validate_consistency_tbl(), *35*
validate_mapping_tbl, 68
validate_ontol_nw, 69
variable.bw.kde, 69
variable_completeness, *6, 7*, 70
variable_completeness(), *6*, *13*, *17*, *55*, *58*
variable_entropy, 71

warn_missing_dots, 71
write_csv, *32*
write_tsv, *32*

zero_entropy_variables, *7*, 72