eHDPrep: an R package for Electronic Health Data Quality Control and Semantic Enrichment

Tom Toner, Ian Overton

Contents

Quick Start

For quality control of health datasets, there are several high-level functions in eHDPrep:

1. `import_dataset()` - import the dataset in .csv, .tsv, or .xlsx format.
2. `assess_quality()` - assess data quality including diagnostics.
3. `apply_quality_ctrl()` - apply quality control measures to the dataset.
4. `review_quality_ctrl()` - review changes made during quality control.
5. `export_dataset()` - export dataset to .csv or .tsv format.

eHDPrep also provides functionality for semantic enrichment with `semantic_enrichment()`.

Introduction

Data preparation is a key foundation for reliable analysis of health data, eHDPrep has been developed for this purpose. The functionality is broadly divided between two themes:

- Quality Control (QC)
- Semantic Enrichment (SE)

Additionally, two "levels" of functions are provided:

- "High-level" functions wrap several "low-level" functions, allowing the user to perform fast, general quality control and SE. This is appropriate for inexperienced R users or those who require rapid data preparation.
- "Low-level" functions require more user interaction, but can be parameterised more extensively to accommodate specific aspects of a dataset. Some of the "low-level" functions are not provided in the "high-level" functions because they require additional user guidance; for example the merging function: `merge_cols()` (see Merge variables).

Finally, this package is built using several packages from the tidyverse and follows its structures and grammars (Wickham et al. 2019). Therefore, the `data` object can typically be piped through eHDPrep's functions which will be modified as described in the function's documentation and returned. We recommend that users have experience with `magrittr`'s pipe operator and core tidyverse packages before using eHDPrep.

Data

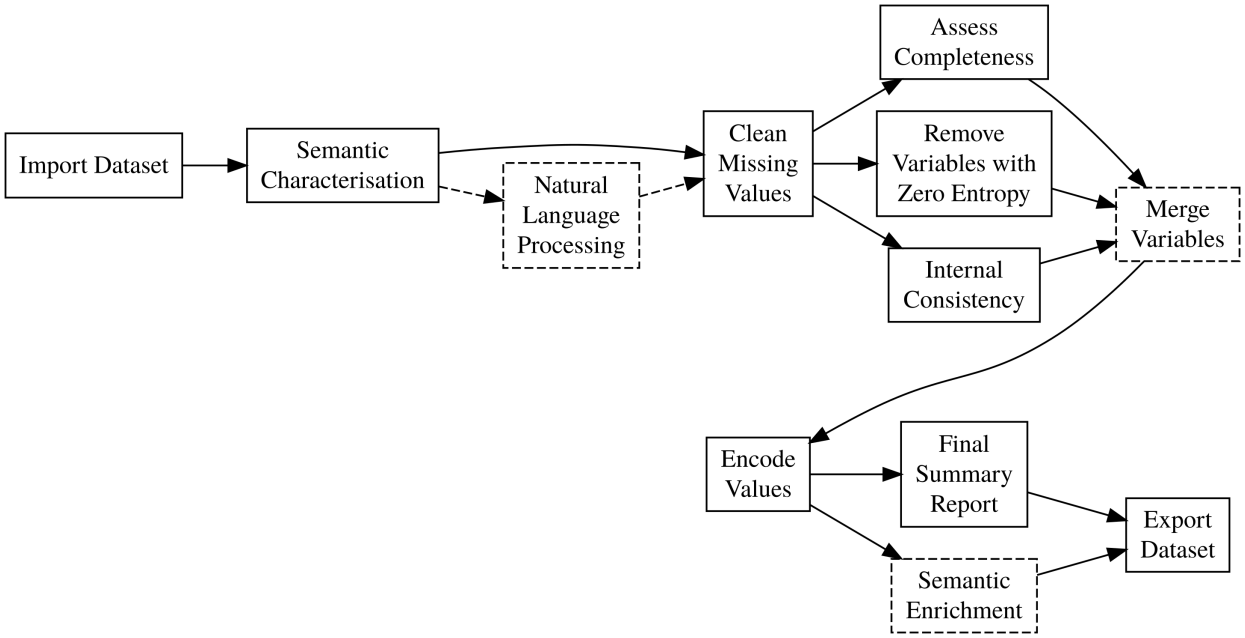
We have created a small synthetic health dataset (a tibble named `example_data`) to demonstrate the functionality of this package. It contains 10 variables and 1000 observations and is documented in `?example_data`.

```
data(example_data)
tibble::glimpse(example_data)
#> Rows: 1,000
#> Columns: 12
#> $ patient_id     <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 1~
#> $ tumoursize     <dbl> 61.71058, 64.18932, 47.81393, 40.93006, 62.11775, 13.64088, ~
#> $ t_stage        <chr> "T3a", "T3b", "T1", "T3a", "T4", "T1", "T1", "T3b", "T2", "T~
#> $ n_stage        <chr> "N2", "N1", "N2", "N0", "N1", "N2", "N2", "N1", "N1", "N0", ~
#> $ diabetes       <chr> "Yes", "Yes", "No", "No", "Yes", "No", "No", "No", "No", "No~
#> $ diabetes_type  <chr> "Type I", "Type II", "Type I", NA, "Type I", NA, NA, NA, NA,~
```

```
#> $ hypertension   <chr> "Yes", "Yes", "Yes", "Yes", "No", "Yes", "No", "No", "Yes", ~
#> $ rural_urban    <chr> "rural", "urban", "rural", "rural", "urban", "urban", "urban~
#> $ marital_status <chr> "married", "divorced", "divorced", "single", "single", "sing~
#> $ SNP_a          <chr> "cc", "cc", "cc", "c", "gg", "g", "c", "gg", "g", "g", "gg",~
#> $ SNP_b          <chr> "tt", "ta", "t", "tt", "t", "t", "t", "tt", "t", "a", "a", "~
#> $ free_text      <chr> "We need grain to keep our mules healthy.", "The gold ring f~
```

Quality control

The quality control functions aim to assess, improve, and compare the quality of a dataset along multiple quality dimensions: completeness, validity, accuracy, consistency, and uniqueness (Roebuck 2011). A suggested workflow for quality control is shown below. The order of these steps is defined by dependency, where later steps benefit from earlier steps. Dashed lines and boxes represent optional steps.



High level functions

Quality control can be performed with little code using the high-level functions. It is suggested that the functions are applied in the order that they appear in this section.

Data Import

eHDPrep provides methods to import a dataset into R from several file types where functionality from `readxl` and `readr` is wrapped into the function `import_dataset()`:

```
# Not run, just examples:
#excel
data <- import_dataset(file = "./dataset.xlsx", format = "excel")
#csv
data <- import_dataset(file = "./dataset.csv", format = "csv")
```

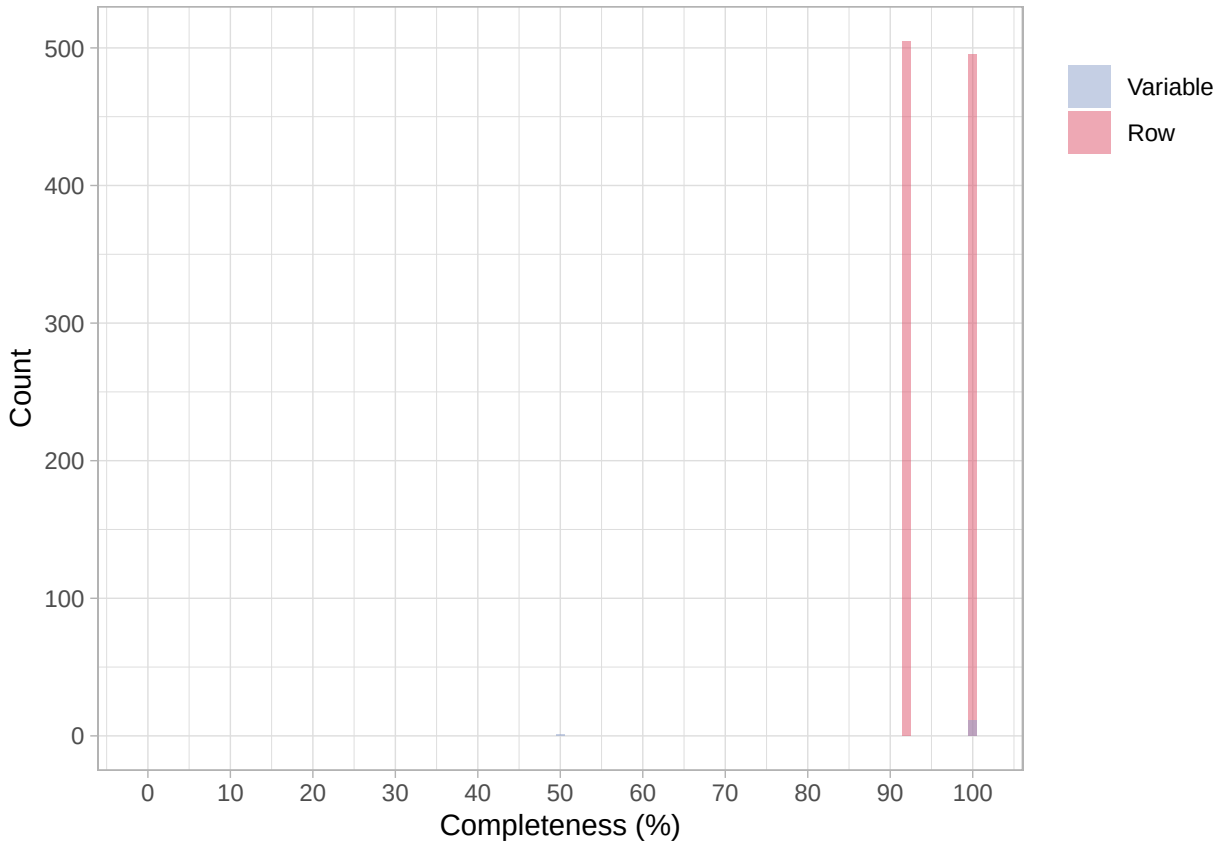
```
#tsv
data <- import_dataset(file = "./dataset.tsv", format = "tsv")
```

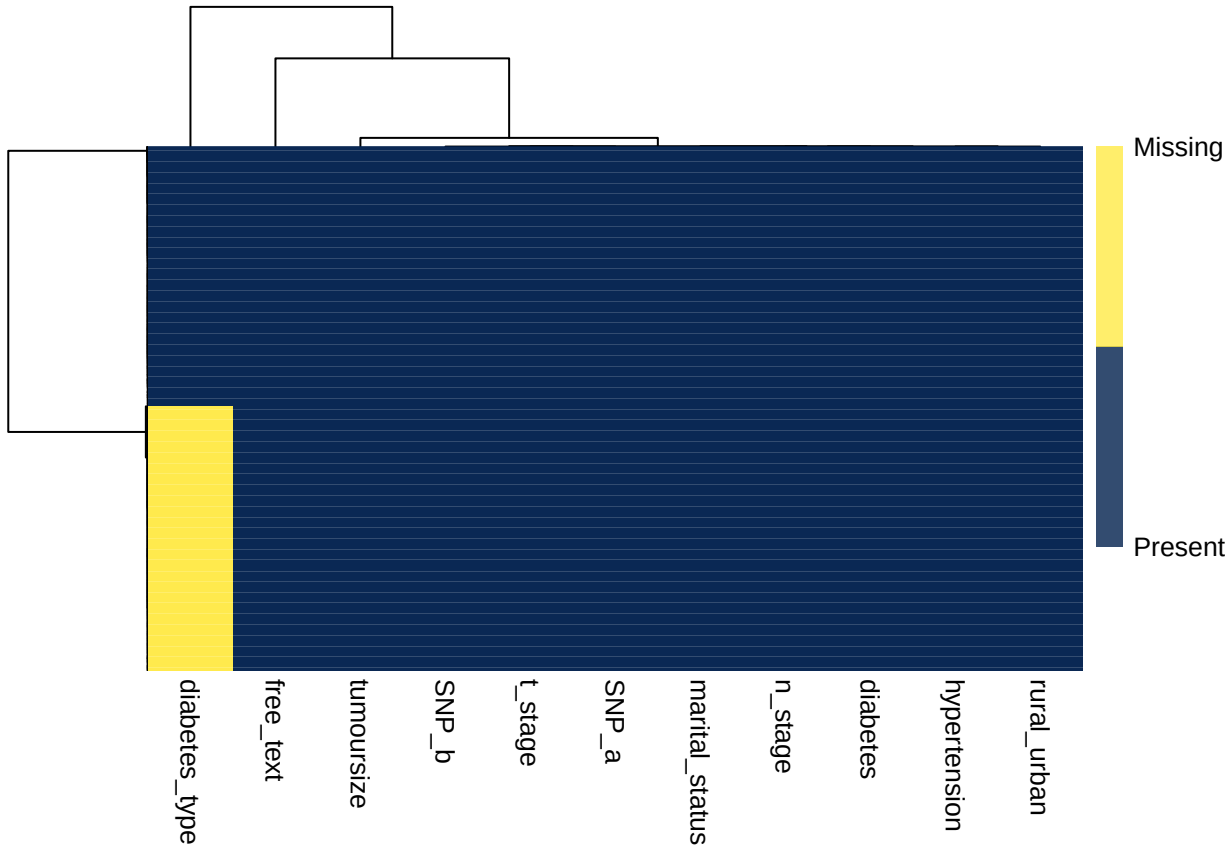
Assess input data quality

An initial assessment of a dataset's quality provides a good basis for its semantic characterisation in understanding variables which require particular attention during quality control. `assess_quality()` will return a list with three top-level elements.

1. A list of completeness measures:
    i. A tibble describing row completeness
    ii. A tibble describing variable (column) completeness
    iii. A bar plot showing row and variable completeness
    iv. A heatmap of completeness, clustered on both axes
    v. A function to ensure completeness heatmap is plotted on a blank canvas
2. A report of internal inconsistencies (requires `consis_tbl` to be provided; see Internal Consistency for more information).
3. A character vector of variables with no entropy (contains only one unique value; see Shannon (1948)).

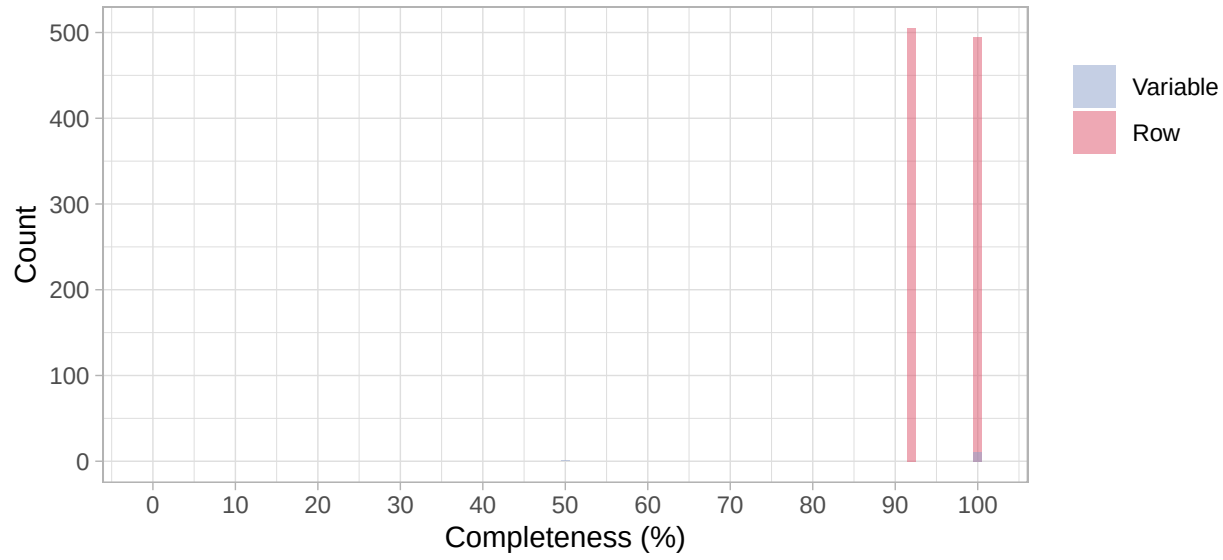
```
data(example_data)

# create a consistency table containing consistency rules
# below states: if a patient has a type of diabetes, they should have diabetes
ct <- tibble::tribble(~varA, ~varB, ~lgl_test, ~varA_boundaries, ~varB_boundaries,
                      "diabetes_type", "diabetes", NA, "Type I", "Yes",
                      "diabetes_type", "diabetes", NA, "Type II", "Yes")

res <- assess_quality(data = example_data, id_var = patient_id, consis_tbl = ct)
```


Count
500
400
300
200
100
0
0
10
20
30
40
50
60
70
80
90
100
Completeness (%)
Variable
Row

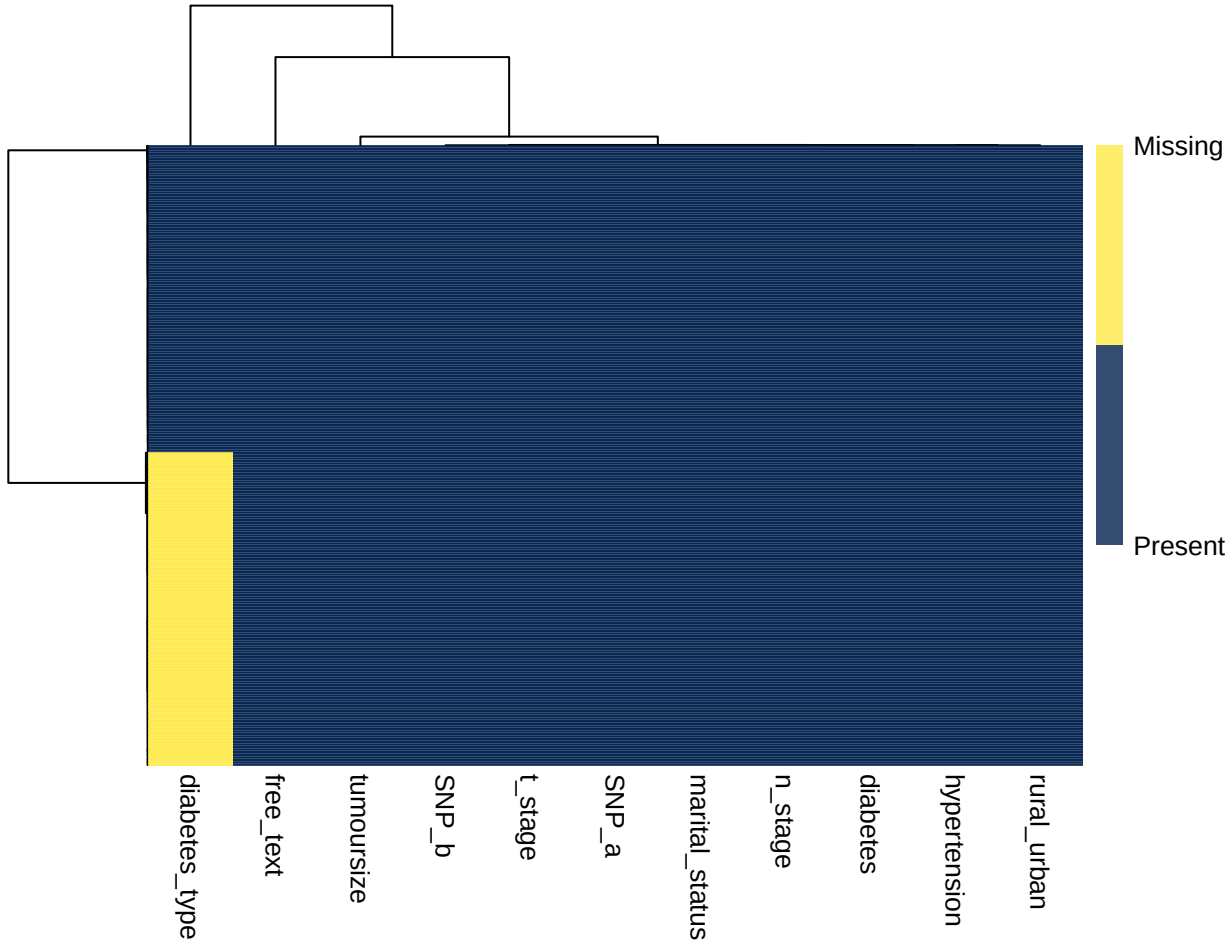

Missing
Present
diabetes_type
free_text
tumoursize
SNP_b
t_stage
SNP_a
marital_status
n_stage
diabetes
hypertension
rural_urban

```
res$completeness$variable_completeness
#> # A tibble: 12 x 4
#>    Variable          NAs NAs_percent Completeness
#>    <chr>           <int>       <dbl>        <dbl>
#>  1 diabetes_type     505        50.         5.0e1
#>  2 patient_id          0         0          1  e2
#>  3 tumoursize          0         0          1  e2
#>  4 t_stage             0         0          1  e2
#>  5 n_stage             0         0          1  e2
#>  6 diabetes            0         0          1  e2
#>  7 hypertension        0         0          1  e2
#>  8 rural_urban         0         0          1  e2
#>  9 marital_status      0         0          1  e2
#> 10 SNP_a               0         0          1  e2
#> 11 SNP_b               0         0          1  e2
#> 12 free_text           0         0          1  e2
res$completeness$row_completeness
#> # A tibble: 1,000 x 4
#>    patient_id   NAs NAs_percent Completeness
#>    <chr>      <int>       <dbl>        <dbl>
#>  1 4              1         8.3         92.
#>  2 6              1         8.3         92.
#>  3 7              1         8.3         92.
#>  4 8              1         8.3         92.
#>  5 9              1         8.3         92.
#>  6 10             1         8.3         92.
#>  7 12             1         8.3         92.
#>  8 15             1         8.3         92.
#>  9 17             1         8.3         92.
#> 10 18             1         8.3         92.
#> # ... with 990 more rows
#> # i Use `print(n = ...)` to see more rows
```

```
res$completeness$completeness_plot
```



```
plot.new()
res$completeness$completeness_heatmap
```




```
res$internal_inconsistency
#> # A tibble: 4 x 8
#>   var_a         var_b    lgl_test var_a_range var_b_range   row values_a
#>   <chr>         <chr>    <lgl>    <chr>       <chr>       <int> <chr>
#> 1 diabetes_type diabetes NA       Type I      Yes             3 Type I
#> 2 diabetes_type diabetes NA       Type I      Yes           190 Type I
#> 3 diabetes_type diabetes NA       Type I      Yes           873 Type I
#> 4 diabetes_type diabetes NA       Type II     Yes           715 Type II
#>   values_b
#>   <chr>
#> 1 No
#> 2 missing
#> 3 missing
#> 4 missing
```

```
res$vars_with_zero_entropy
#> character(0)
```

Apply quality control

To apply quality control to a dataset in one function, as `apply_quality_ctrl()` does, it is important to ensure variables are processed appropriately according to their data type. R and eHDPrep suggest data types with `assume_var_classes()` which writes the results to a .csv file. The user can amend this externally and import back into R with `import_var_classes()`.

```
assume_var_classes(data = example_data, out_file = "./datatypes.csv")

# (user makes manual edits externally)

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

The permitted datatypes are: `"id"`, `"numeric"`, `"double"`, `"integer"`, `"character"`, `"factor"`, `"ordinal"`, `"ordinal_tstage`, `"ordinal_nstage"`, `"genotype"`, `"freetext"`, `"logical"`. Note that `ordinal` variables are not modified by `apply_quality_ctrl()` as the ordinal classes would need to be specified for each variable. eHDPrep provides two special ordinal data types `"ordinal_tstage"` and `"ordinal_nstage"` for two common cancer staging measures where the orders are precoded.

The data types for `example_data` are shown below:

```
data_types
#> # A tibble: 12 x 2
#>    var            datatype
#>    <chr>          <chr>
#>  1 patient_id     id
#>  2 tumoursize     numeric
#>  3 t_stage        ordinal_tstage
#>  4 n_stage        ordinal_nstage
#>  5 diabetes       factor
#>  6 diabetes_type  factor
#>  7 hypertension   factor
#>  8 rural_urban    factor
#>  9 marital_status factor
#> 10 SNP_a          genotype
#> 11 SNP_b          genotype
#> 12 free_text      freetext
```

Data types are modified as follows:

Data type	Modification Summary
`id`	Ignored
`numeric`	Ignored
`double`	Ignored
`integer`	Ignored
`ordinal`	Ignored
`logical`	Ignored
`ordinal_tstage`	Converted to ordered factor with predetermined levels
`ordinal_nstage`	Converted to ordered factor with predetermined levels
`factor`; `character`	*If >2 categories:* converted to multiple variables using one-hot encoding (see Encoding categorical data). *If 2 categories, specified in* `bin_cats` *parameter:* converted to ordered factor with two levels (see `?encode_binary_cats`)

Data type	Modification Summary
`genotype`	Converted to ordered factors using SNP allele frequency in the variable (see Encoding genotype (SNP) data)
`freetext`	Groups of words which appear within two words each other in the variable with a minimum frequency of occurrence set by `min_freq` are converted to logical variables describing each group (see Extract information from free text variables)

Quality Control with the function `apply_quality_control()` is performed upon the `example_data` with the following parameters (please see below):

- `data`: The dataset to be quality controlled.
- `id_var`: The variable which identifies each row. Note it is not surrounded by quotes.
- `class_tbl`: The object shown above describing variables' data types.
- `bin_cats:` A character vector showing how variables with two options should be encoded with the syntax `negative_finding = positive_finding`. If positivity/negativity is not associated with the binary categories of a variable (e.g. `rural_urban` in `example_data`) then the ordering can be arbitrarily decided.
- `min_freq`: The minimum frequency of occurrence for groups of proximal words in free-text variables. Those which meet this threshold are added as logical variables (see `?extract_freetext` and `?skipgram_append`). This is ignored if there are no free-text variables specified in `class_tbl`.

```
apply_quality_ctrl(data = example_data,
                   id_var = patient_id,
                   class_tbl = data_types,
                   bin_cats =c("No" = "Yes", "rural" = "urban"),
                   min_freq = 0.6)
#> # A tibble: 1,000 x 18
#>    patient_id tumoursize t_stage n_stage diabetes diabetes_type hypert~1 rural~2
#>         <dbl>      <dbl> <ord>   <ord>   <fct>    <chr>         <fct>    <fct>
#>  1          1         62. T3a     N2      Yes      Type I        Yes      rural
#>  2          2         64. T3b     N1      Yes      Type II       Yes      urban
#>  3          3         48. T1      N2      No       Type I        Yes      rural
#>  4          4         41. T3a     N0      No       <NA>          Yes      rural
#>  5          5         62. T4      N1      Yes      Type I        No       urban
#>  6          6         14. T1      N2      No       <NA>          Yes      urban
#>  7          7         63. T1      N2      No       <NA>          No       urban
#>  8          8         44. T3b     N1      No       <NA>          No       rural
#>  9          9         44. T2      N1      No       <NA>          Yes      rural
#> 10         10         32. T1      N0      No       <NA>          No       rural
#>    SNP_a SNP_b board_w~3 leas_~4 sixte~5 white~6 marit~7 marit~8 marit~9 marit~*
#>    <ord> <ord>     <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
#>  1 C/C   T/T           0       0       0       0       0       1       0       0
#>  2 C/C   A/T           0       0       0       0       1       0       0       0
#>  3 C/C   T/T           0       0       0       0       1       0       0       0
#>  4 C/C   T/T           0       0       0       0       0       0       1       0
#>  5 G/G   T/T           0       0       0       0       0       0       1       0
#>  6 G/G   T/T           0       0       0       0       0       0       1       0
#>  7 C/C   T/T           0       0       0       0       0       1       0       0
```

```
#>  8 G/G   T/T           0       0       0       0       0       0       1       0
#>  9 G/G   T/T           0       0       0       0       0       0       1       0
#> 10 G/G   A/A           0       0       0       0       0       1       0       0
#> # ... with 990 more rows, and abbreviated variable names 1: hypertension,
#> #   2: rural_urban, 3: board_will, 4: leas_ran, 5: sixteen_week, 6: white_back,
#> #   7: marital_status_divorced, 8: marital_status_married, 9: marital_status_single,
#> #   *: marital_status_NA
#> # i Use `print(n = ...)` to see more rows
```

The variables `diabetes` and `diabetes_type` demonstrate some of the limitations of using the high-level functions which do not support variable merging due to required additional user configuration. For this dataset, we can first merge the two diabetes variables using a low-level function (see `merge_cols()` in Merge Variables) before `apply_quality_ctrl()` to produce a dataset with higher uniqueness. Note the data types need to be updated to include the new merged variable:

Updated `class_tbl`:

```
data_types_diabetes_m
#> # A tibble: 11 x 2
#>    var             datatype
#>    <chr>           <chr>
#>  1 patient_id      id
#>  2 tumoursize      numeric
#>  3 t_stage         ordinal_tstage
#>  4 n_stage         ordinal_nstage
#>  5 diabetes_merged factor
#>  6 hypertension    factor
#>  7 rural_urban     factor
#>  8 marital_status  factor
#>  9 SNP_a           genotype
#> 10 SNP_b           genotype
#> 11 free_text       freetext
```

Quality control using low-level function, `merge_cols()`, to merge variables (see Merge Variables):

```
require(magrittr) # for pipe: %>%
#> Loading required package: magrittr
example_data %>%
  # first merge diabetes variables
  merge_cols(primary_var = diabetes_type,
             secondary_var = diabetes,
             merge_var_name = "diabetes_merged",
             rm_in_vars = TRUE) %>%
  # pass data with diabetes_merged to high-level QC function
  apply_quality_ctrl(id_var = patient_id, class_tbl = data_types_diabetes_m,
                     bin_cats =c("No" = "Yes", "rural" = "urban")) ->
  post_QC_example_data
#> New names:
#> * `diabetes_merged_Type I` -> `diabetes_merged_Type.I`
#> * `diabetes_merged_Type II` -> `diabetes_merged_Type.II`

  post_QC_example_data
#> # A tibble: 1,000 x 16
```

```
#>    patient_id tumoursize t_stage n_stage hypertension rural_urban SNP_a SNP_b
#>         <dbl>      <dbl> <ord>   <ord>   <fct>        <fct>       <ord> <ord>
#>  1          1        62. T3a     N2      Yes          rural       C/C   T/T
#>  2          2        64. T3b     N1      Yes          urban       C/C   A/T
#>  3          3        48. T1      N2      Yes          rural       C/C   T/T
#>  4          4        41. T3a     N0      Yes          rural       C/C   T/T
#>  5          5        62. T4      N1      No           urban       G/G   T/T
#>  6          6        14. T1      N2      Yes          urban       G/G   T/T
#>  7          7        63. T1      N2      No           urban       C/C   T/T
#>  8          8        44. T3b     N1      No           rural       G/G   T/T
#>  9          9        44. T2      N1      Yes          rural       G/G   T/T
#> 10         10        32. T1      N0      No           rural       G/G   A/A
#>    diabetes_merged_No diabetes~1 diabe~2 diabe~3 marit~4 marit~5 marit~6 marit~7
#>                 <dbl>      <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
#>  1                  0          1       0       0       0       1       0       0
#>  2                  0          0       1       0       1       0       0       0
#>  3                  0          1       0       0       1       0       0       0
#>  4                  1          0       0       0       0       0       1       0
#>  5                  0          1       0       0       0       0       1       0
#>  6                  1          0       0       0       0       0       1       0
#>  7                  1          0       0       0       0       1       0       0
#>  8                  1          0       0       0       0       0       1       0
#>  9                  1          0       0       0       0       0       1       0
#> 10                  1          0       0       0       0       1       0       0
#> # ... with 990 more rows, and abbreviated variable names 1: diabetes_merged_Type.I,
#> #   2: diabetes_merged_Type.II, 3: diabetes_merged_NA, 4: marital_status_divorced,
#> #   5: marital_status_married, 6: marital_status_single, 7: marital_status_NA
#> # i Use `print(n = ...)` to see more rows
```

The function `merge_cols()` may be run with the parameter `to_numeric_matrix = TRUE`, which automatically converts the dataset to numeric values, facilitated by prior encoding of any categorical, ordinal, or genotype data:

```
example_data %>%
  # first merge diabetes variables
    merge_cols(primary_var = diabetes_type,
               secondary_var = diabetes,
               merge_var_name = "diabetes_merged",
               rm_in_vars = TRUE) %>%
  # pass data with diabetes_merged to high-level QC function
  apply_quality_ctrl(id_var = patient_id, class_tbl = data_types_diabetes_m,
                     bin_cats =c("No" = "Yes", "rural" = "urban"),
                     # Relevant line:
                     to_numeric_matrix = TRUE) ->
  post_QC_example_data_m
#> New names:
#> * `diabetes_merged_Type I` -> `diabetes_merged_Type.I`
#> * `diabetes_merged_Type II` -> `diabetes_merged_Type.II`

  # concise summary of output:
  tibble::glimpse(post_QC_example_data_m)
#> Rows: 1,000
#> Columns: 15
```

```
#> $ tumoursize              <dbl> 61.71058, 64.18932, 47.81393, 40.93006, 62.11775, 1~
#> $ t_stage                 <dbl> 3, 4, 1, 3, 5, 1, 1, 4, 2, 1, 5, 4, 3, 2, 5, 3, 3, ~
#> $ n_stage                 <dbl> 3, 2, 3, 1, 2, 3, 3, 2, 2, 1, 3, 1, 3, 1, 2, 2, 1, ~
#> $ hypertension            <dbl> 2, 2, 2, 2, 1, 2, 1, 1, 2, 1, 2, 1, 1, 1, 1, 2, 1, ~
#> $ rural_urban             <dbl> 1, 2, 1, 1, 2, 2, 2, 1, 1, 1, 2, 2, 2, 2, 2, 1, 1, ~
#> $ SNP_a                   <dbl> 1, 1, 1, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2, 1, 2, 2, 2, ~
#> $ SNP_b                   <dbl> 2, 3, 2, 2, 2, 2, 2, 2, 2, 1, 1, 3, 1, 3, 2, 1, 2, ~
#> $ diabetes_merged_No      <dbl> 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, ~
#> $ diabetes_merged_Type.I  <dbl> 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, ~
#> $ diabetes_merged_Type.II <dbl> 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, ~
#> $ diabetes_merged_NA      <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
#> $ marital_status_divorced <dbl> 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, ~
#> $ marital_status_married  <dbl> 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, ~
#> $ marital_status_single   <dbl> 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, ~
#> $ marital_status_NA       <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
```

This is useful for preparation for later analysis, including machine learning applications, or for further preparation with Semantic Enrichment.

Review of all quality control

eHDPrep provides functionality for users to review the results of quality control operations that have been applied to the input data. `review_quality_ctrl()` provides information of quality control modifications at multiple levels of detail.

```
qc_review <- review_quality_ctrl(before_tbl = example_data,
                                 after_tbl = post_QC_example_data,
                                 id_var = patient_id)
```

The `variable_level_changes` list element is a tibble with variable names in the first column. The second column can contain up to three unique values describing the presence of the variable in the post-quality control dataset (Added, Removed, Preserved):

```
qc_review$variable_level_changes
#> # A tibble: 20 x 2
#>    variable                presence
#>    <chr>                   <chr>
#>  1 patient_id              Preserved
#>  2 tumoursize              Preserved
#>  3 t_stage                 Preserved
#>  4 n_stage                 Preserved
#>  5 diabetes                Removed
#>  6 diabetes_type           Removed
#>  7 hypertension            Preserved
#>  8 rural_urban             Preserved
#>  9 marital_status          Removed
#> 10 SNP_a                   Preserved
#> 11 SNP_b                   Preserved
#> 12 free_text               Removed
#> 13 diabetes_merged_No      Added
#> 14 diabetes_merged_Type.I  Added
#> 15 diabetes_merged_Type.II Added
```

```
#> 16 diabetes_merged_NA      Added
#> 17 marital_status_divorced Added
#> 18 marital_status_married  Added
#> 19 marital_status_single   Added
#> 20 marital_status_NA       Added
```

The `value_level_changes` element is a tibble which shows changes made during quality control where each row records a value modification:

```
qc_review$value_level_changes
#> # A tibble: 2,019 x 6
#>    patient_id new_var old_var old_value new_value mod_type
#>    <chr>      <chr>   <chr>   <chr>     <chr>     <chr>
#>  1 31         t_stage t_stage equivocal <NA>      Removal
#>  2 34         t_stage t_stage equivocal <NA>      Removal
#>  3 44         t_stage t_stage equivocal <NA>      Removal
#>  4 48         t_stage t_stage equivocal <NA>      Removal
#>  5 261        t_stage t_stage equivocal <NA>      Removal
#>  6 263        t_stage t_stage equivocal <NA>      Removal
#>  7 348        t_stage t_stage equivocal <NA>      Removal
#>  8 454        t_stage t_stage equivocal <NA>      Removal
#>  9 468        t_stage t_stage equivocal <NA>      Removal
#> 10 569        t_stage t_stage equivocal <NA>      Removal
#> # ... with 2,009 more rows
#> # i Use `print(n = ...)` to see more rows

# summary of above
qc_review$value_level_changes %>%
  dplyr::distinct(across(!patient_id))
#> # A tibble: 13 x 5
#>    new_var old_var old_value new_value mod_type
#>    <chr>   <chr>   <chr>     <chr>     <chr>
#>  1 t_stage t_stage equivocal <NA>      Removal
#>  2 SNP_a   SNP_a   cc        C/C       Substitution
#>  3 SNP_a   SNP_a   c         C/C       Substitution
#>  4 SNP_a   SNP_a   gg        G/G       Substitution
#>  5 SNP_a   SNP_a   g         G/G       Substitution
#>  6 SNP_a   SNP_a   gc        C/G       Substitution
#>  7 SNP_a   SNP_a   cg        C/G       Substitution
#>  8 SNP_b   SNP_b   tt        T/T       Substitution
#>  9 SNP_b   SNP_b   ta        A/T       Substitution
#> 10 SNP_b   SNP_b   t         T/T       Substitution
#> 11 SNP_b   SNP_b   a         A/A       Substitution
#> 12 SNP_b   SNP_b   at        A/T       Substitution
#> 13 SNP_b   SNP_b   aa        A/A       Substitution
```

Note in the above that "gc" has been encoded, via encode_genotypes(), as "C/G" to create a standard representation of this SNP allele. Positional information for SNP allele should be recorded elsewhere, if required.

The `value_level_changes_plt` element visualises the content of the `value_level_changes` element. It is a bar plot with rows on the x-axis and the proportion of each row's values which were modified (removed, substituted, or added) on the y-axis:

```
qc_review$value_level_changes_plt
```

This can be useful when many changes have occurred and the source table contains a large amount of data.

Data export

Modified data can be exported with `export_dataset()` as either .csv or .tsv:

```
#csv
export_dataset(x = post_QC_example_data,
               file = "./post_QC_example_data.csv",
               format = "csv")
#tsv
export_dataset(x = post_QC_example_data,
               file = "./post_QC_example_data.csv",
               format = "tsv")
```

Low level functions

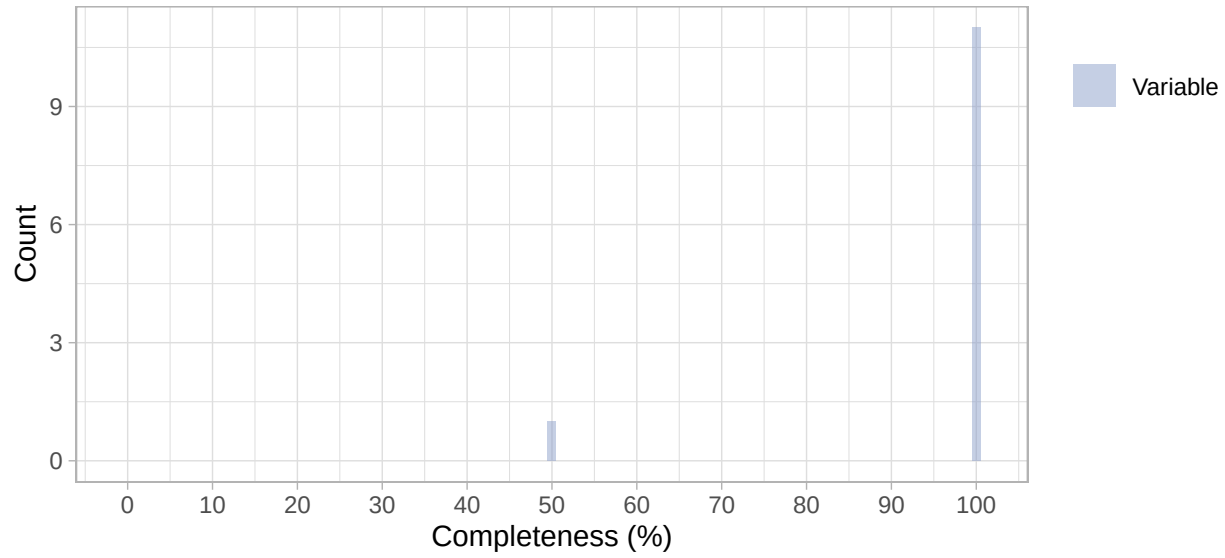
This section describes functions that can provide more granular access to the eHDPrep quality control operations. While the functionality is largely available within 'high-level' functions, directly calling low-level functions provides greater scope to adjust individual parameter values and allows for finer-grained assessment of each step in the quality control process. Operations that may only be performed using low-level functions are: `merge_cols()`, `compare_info_content()`, `compare_info_content_plt()` (see Merge variables).

Measure completeness

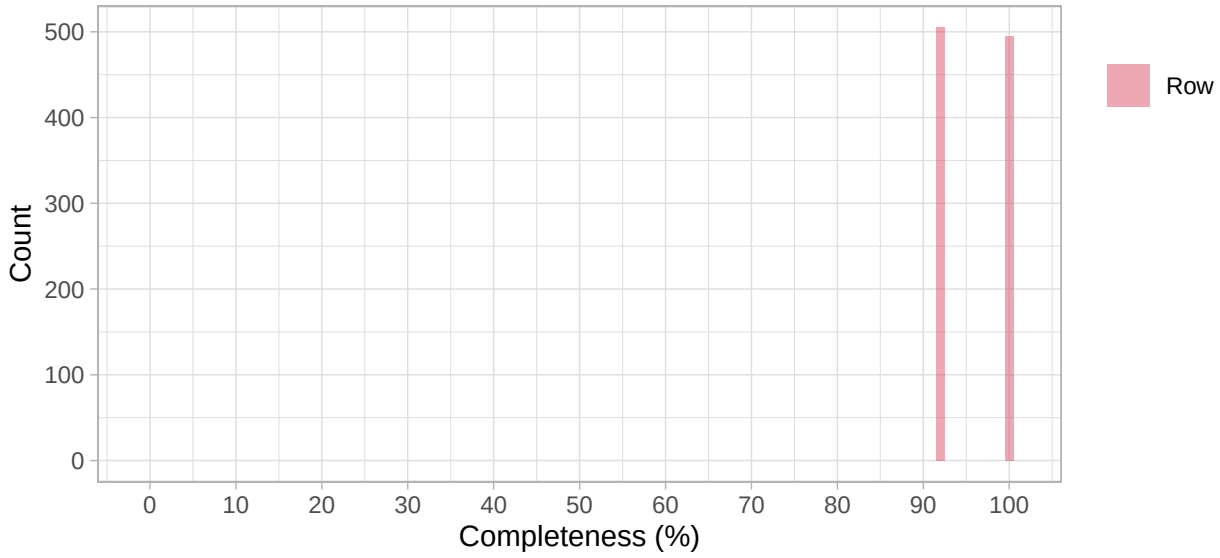
Completeness in variables and rows can be calculated in tibbles and visualised in a bar plot as shown below:

```
variable_completeness(example_data)
#> # A tibble: 12 x 4
#>    Variable         NAs NAs_percent Completeness
```

```
#>    <chr>          <int>       <dbl>        <dbl>
#>  1 diabetes_type    505        50.         5.0e1
#>  2 patient_id         0         0          1  e2
#>  3 tumoursize         0         0          1  e2
#>  4 t_stage            0         0          1  e2
#>  5 n_stage            0         0          1  e2
#>  6 diabetes           0         0          1  e2
#>  7 hypertension       0         0          1  e2
#>  8 rural_urban        0         0          1  e2
#>  9 marital_status     0         0          1  e2
#> 10 SNP_a              0         0          1  e2
#> 11 SNP_b              0         0          1  e2
#> 12 free_text          0         0          1  e2

row_completeness(data = example_data, id_var = patient_id)
#> # A tibble: 1,000 x 4
#>    patient_id   NAs NAs_percent Completeness
#>    <chr>      <int>       <dbl>        <dbl>
#>  1 4              1         8.3          92.
#>  2 6              1         8.3          92.
#>  3 7              1         8.3          92.
#>  4 8              1         8.3          92.
#>  5 9              1         8.3          92.
#>  6 10             1         8.3          92.
#>  7 12             1         8.3          92.
#>  8 15             1         8.3          92.
#>  9 17             1         8.3          92.
#> 10 18             1         8.3          92.
#> # ... with 990 more rows
#> # i Use `print(n = ...)` to see more rows

plot_completeness(data = example_data, id_var = patient_id, plot = "variables")
```



```
plot_completeness(data = example_data, id_var = patient_id, plot = "rows")
```



An overview of the dataset completeness is generated by `completeness_heatmap()` which utilises `pheatmap` (Kolde 2019). Additional parameters are passed to `pheatmap()` through `...` (see `?pheatmap` for all options). Additional parameters are supplied in creating the heatmap below where the row names are hidden because they clutter the plot. The completeness
heatmap may be useful for identifying structural patterns in the missingness, for example
indicative of non-random missingness.There are three underlying methods which are used to encode the data so that non-numeric data can be visualised:

1. (Default). Missing values are numerically encoded with a highly negative number, numerically distant from all values in data. Non-missing 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).
2. Same as 1 but cells are coloured by the values input to the clustering algorithm (instead of missing or present).
3. Boolean values are used for clustering (present values = 1; missing values = 0). Cells are coloured by presence (yellow = missing; blue = present).

```
# show_rownames is pased to pheatmap() through ...
completeness_heatmap(data = strings_to_NA(example_data),
                     id_var = patient_id,
                     show_rownames = F)
```

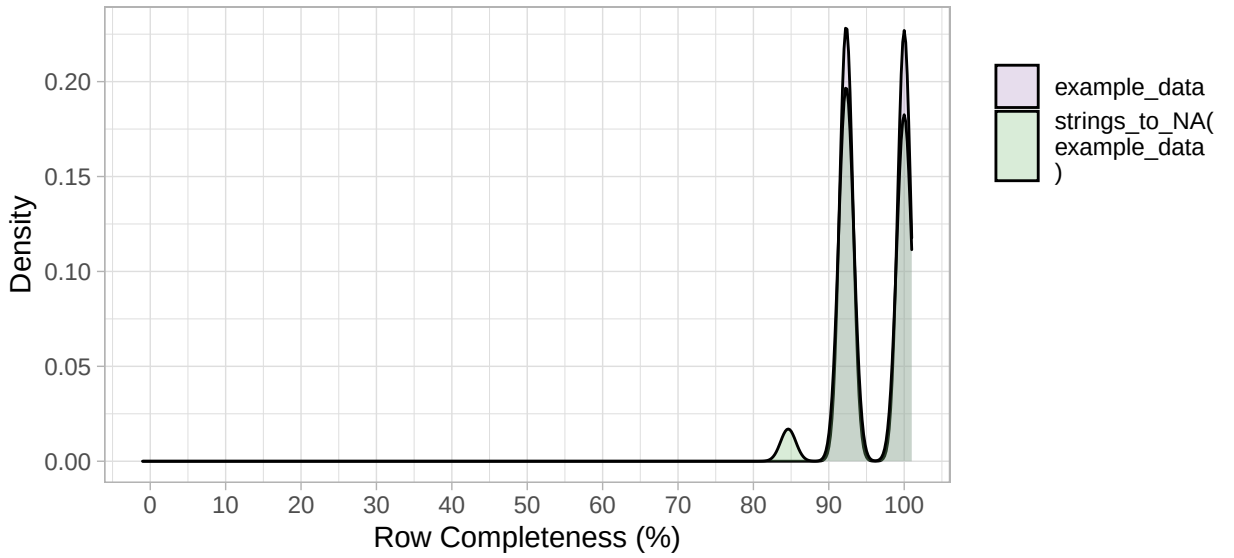
Variable-level annotations can be provided with the `annotation_tbl` parameter to further characterise completeness patterns. Below, the `data_types` tibble is used:

```
completeness_heatmap(data = strings_to_NA(example_data),
                     id_var = patient_id,
                     show_rownames = FALSE,
                     annotation_tbl = data_types)
```

Comparison of dataset completeness before and after quality control is available using the `compare_completeness()` function. The plot below reveals a decrease in reported

completeness following quality control because the input `example_data` encoded missingness as strings (for example 'not recorded') which were converted to `NA` values during quality control.

```
compare_completeness(tbl_a = example_data, tbl_b = strings_to_NA(example_data),
                     dim = 1, tbl_a_lab = "example_data",
                     tbl_b_lab = "strings_to_NA(\nexample_data\n)")
```



Internal consistency

Relationships between the values of different, but related, variables for a given patient may be used to define rules that identify internal inconsistencies. For example the number_of_lymph_nodes examined should be greater than or equal to the number_of_positive_lymph_nodes. `identify_inconsistency()` can test pairs of variables in multiple ways:

1. 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)

The internal consistency tests rely on such rules being specified in a separate data frame (argument: `consis_tbl`). An example of this type of table is shown below. Column headers are not important but column order is important. See `?validate_consistency_tbl` for all requirements.

```
example_incon_rules <-  tibble::tribble(~varA, ~varB, ~lgl_test, ~varA_boundaries, ~varB_boundaries,
                                        "diabetes_type", "diabetes", NA, "Type I", "Yes",
                                        "diabetes_type", "diabetes", NA, "Type II", "Yes"
                                        )
example_incon_rules
#> # A tibble: 2 x 5
#>   varA          varB     lgl_test varA_boundaries varB_boundaries
```

```
#>   <chr>         <chr>    <lgl>    <chr>           <chr>
#> 1 diabetes_type diabetes NA       Type I          Yes
#> 2 diabetes_type diabetes NA       Type II         Yes
```

These rules are interpreted as:

- in rows where `diabetes_type` equals `Type I`, `diabetes` should equal `Yes`.
- in rows where `diabetes_type` equals `Type II`, `diabetes` should equal `Yes`.

Note: The order of variables in each row is important here, switching `diabetes` and `diabetes_type` in the first row of `example_incon_rules` would be interpreted as where `diabetes` equals `Yes`, `diabetes_type` should always equal `Type I`.

This format of a user-defined consistency table should be validated as shown below:

```
# validate the consistency rule table
validate_consistency_tbl(data = example_data, consis_tbl = example_incon_rules)
#> Consistency table is valid.
```

The tests are run against the data and all instances (rows). When no inconsistencies are found, a confirmatory message is returned along with the data (invisibly). However, when inconsistencies are found, a warning is thrown and a table detailing the inconsistencies is returned:

```
identify_inconsistency(data = example_data, consis_tbl = example_incon_rules)
#> Warning: One or more inconsistencies were identified. They are shown in the
#> returned tibble.
#> # A tibble: 4 x 8
#>   var_a         var_b    lgl_test var_a_range var_b_range   row values_a
#>   <chr>         <chr>    <lgl>    <chr>       <chr>       <int> <chr>
#> 1 diabetes_type diabetes NA       Type I      Yes             3 Type I
#> 2 diabetes_type diabetes NA       Type I      Yes           190 Type I
#> 3 diabetes_type diabetes NA       Type I      Yes           873 Type I
#> 4 diabetes_type diabetes NA       Type II     Yes           715 Type II
#>   values_b
#>   <chr>
#> 1 No
#> 2 missing
#> 3 missing
#> 4 missing
```

The first five columns represent the rules set in `consis_tbl`. The additional columns describe:

6. The inconsistent row(s)
7. The value in the variable reported in column 1
8. The value in the variable reported in column 2 (inconsistent with the corresponding value in the variable in column 1, given the rules in `consis_tbl`).

Merge variables

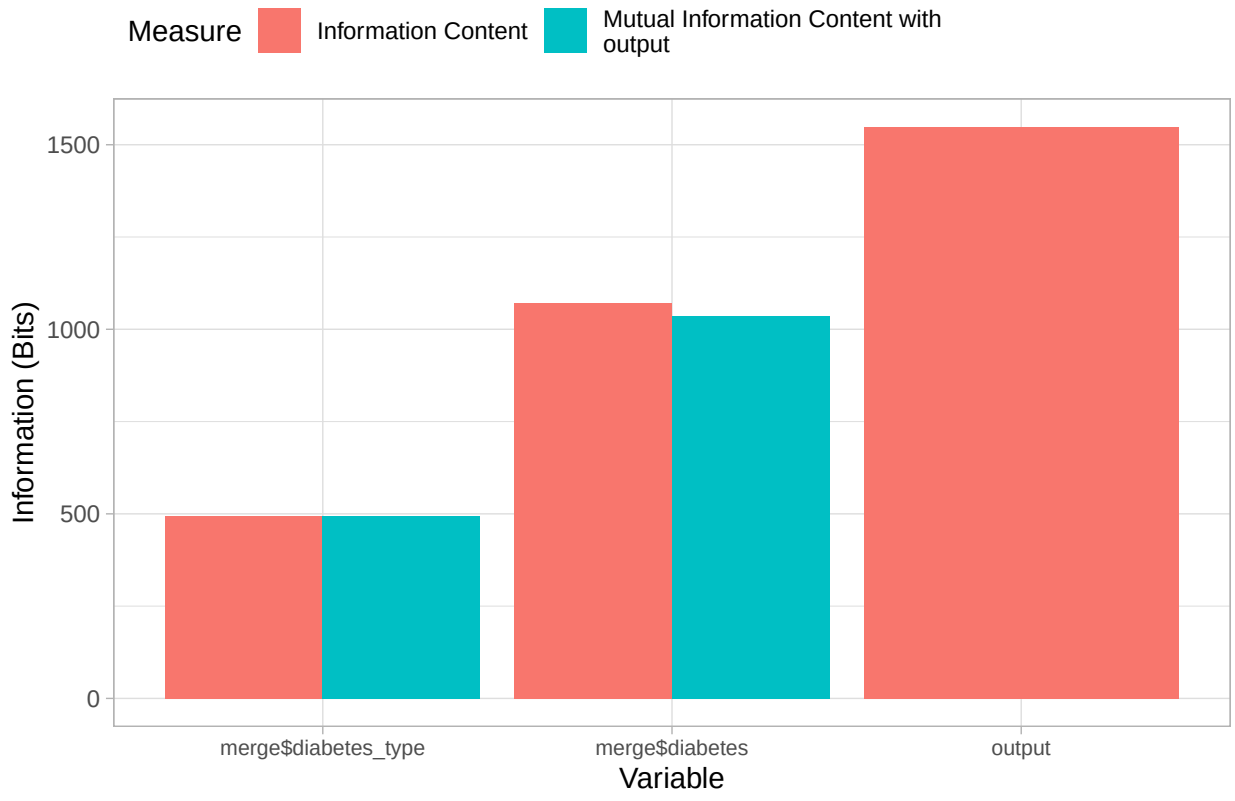
Merging variables can improve the uniqueness and completeness of the dataset, also reducing its dimensionality. In `example_data`, `diabetes` and `diabetes_type` record observations of the same disease at different levels and are merged below:

```
merge <- merge_cols(data = example_data,
                    primary_var = diabetes_type,
                    secondary_var = diabetes,
                    merge_var_name = "diabetes_merged")
```

By default, the input variables (e.g. `diabetes` and `diabetes_type`) are preserved. They can be removed with the parameter `rm_in_vars = TRUE`.

`compare_info_content` and `compare_info_content_plt` can support merging strategies by identifying when a pairwise merge operation results in loss of information:

```
merge_IC <- compare_info_content(input1 = merge$diabetes,
                                 input2 = merge$diabetes_type,
                                 composite = merge$diabetes_merged)

merge_IC
#> # A tibble: 5 x 3
#>   Information      Variable            Measure
#>   <chr>            <chr>               <chr>
#> 1 1070.74650282141 merge$diabetes      "Information Content"
#> 2 494.9635676875   merge$diabetes_type "Information Content"
#> 3 1548.12947181663 output              "Information Content"
#> 4 1035.17075316701 merge$diabetes      "Mutual Information Content with\noutput"
#> 5 494.9635676875   merge$diabetes_type "Mutual Information Content with\noutput"

compare_info_content_plt(compare_info_content_res = merge_IC)
```



In the above bar chart, the variable `diabetes` has higher information content than its mutual information

with the output variable; shown in the bars for `merge$diabetes`. Therefore information loss has occurred in the merge operation, due to two features of the input variables (`diabetes_type` and `diabetes`). Firstly missing values in `diabetes` are represented as "missing". Secondly there is an internal inconsistency where `diabetes` is recorded as "No" but `diabetes_type` is recorded as "Type 1". This example reveals that additional preprocessing is required before the variable merge can be successfully achieved.

Encoding missing values

Missing values can be recorded in several ways (e.g. "unknown", "missing"). R uses `NA` as a standard representation of missing values which allows for the user and packages to process them appropriately (e.g. `mean(x, na.rm = T)`).

eHDPrep can convert values representing missingness to NA with two functions:

- `strings_to_NA()` will encode a series of predefined strings which represent missingness or specific strings specified in the argument `strings_to_replace` as `NA`.
  - predefined strings: "Undetermined", "unknown", "missing", "fail", "fail / unknown", "equivocal", "equivocal / unknown", "*"
- `nums_to_NA()` will replace (only) numbers specified in `nums_to_replace` with `NA` in numeric variables.

```
# default values
example_data_NAs1 <- strings_to_NA(data = example_data)

# predefined value "equivocal" is removed
unique(example_data_NAs1$t_stage)
#> [1] "T3a" "T3b" "T1"  "T4"  "T2"  NA

# custom values (T1 does not represent missingness, just used as an example)
example_data_NAs2 <-strings_to_NA(data = example_data,
                                  strings_to_replace = "T1")

# custom value "T1" is removed
unique(example_data_NAs2$t_stage)
#> [1] "T3a"       "T3b"       NA          "T4"        "T2"        "equivocal"

# numeric value is removed in patient_id
nums_to_NA(data = example_data, patient_id, nums_to_replace = c(1,3))
#> # A tibble: 1,000 x 12
#>    patient_id tumoursize t_stage n_stage diabetes diabetes_type hypertension
#>         <int>      <dbl> <chr>   <chr>   <chr>    <chr>         <chr>
#>  1         NA         62. T3a     N2      Yes      Type I        Yes
#>  2          2         64. T3b     N1      Yes      Type II       Yes
#>  3         NA         48. T1      N2      No       Type I        Yes
#>  4          4         41. T3a     N0      No       <NA>          Yes
#>  5          5         62. T4      N1      Yes      Type I        No
#>  6          6         14. T1      N2      No       <NA>          Yes
#>  7          7         63. T1      N2      No       <NA>          No
#>  8          8         44. T3b     N1      No       <NA>          No
#>  9          9         44. T2      N1      No       <NA>          Yes
#> 10         10         32. T1      N0      No       <NA>          No
#>    rural_urban marital_status SNP_a SNP_b free_text
#>    <chr>       <chr>          <chr> <chr> <chr>
```

```
#>  1 rural       married       cc    tt    We need grain to keep our mules healt~
#>  2 urban       divorced      cc    ta    The gold ring fits only a pierced ear.
#>  3 rural       divorced      cc    t     The vamp of the shoe had a gold buckl~
#>  4 rural       single        c     tt    Wipe the grease off his dirty face.
#>  5 urban       single        gg    t     Look in the corner to find the tan sh~
#>  6 urban       single        g     t     Float the soap on top of the bath wat~
#>  7 urban       married       c     t     Feel the heat of the weak dying flame.
#>  8 rural       single        gg    tt    A stuffed chair slipped from the movi~
#>  9 rural       single        g     t     The beam dropped down on the workmen'~
#> 10 rural       married       g     a     Screen the porch with woven straw mat~
#> # ... with 990 more rows
#> # i Use `print(n = ...)` to see more rows
```

Encoding categorical data

Categorical (nominal) data can present problems when analysed; either resulting in an error or improper analysis; for example, treating the relationships between categories as if they were ordinal. To combat this, `encode_cats()` utilises one hot encoding and creates a new variable for each unique value in the input categorical variable. The values in each new variable describe the presence of the unique value where 1 means present and 0 means not present. This is demonstrated below with `marital_status`.

```
encode_cats(data = example_data, marital_status) %>%
  dplyr::select(dplyr::starts_with("marital_status"))
#> # A tibble: 1,000 x 4
#>    marital_status_divorced marital_status_married marital_status_single
#>                      <dbl>                  <dbl>                 <dbl>
#>  1                       0                      1                     0
#>  2                       1                      0                     0
#>  3                       1                      0                     0
#>  4                       0                      0                     1
#>  5                       0                      0                     1
#>  6                       0                      0                     1
#>  7                       0                      1                     0
#>  8                       0                      0                     1
#>  9                       0                      0                     1
#> 10                       0                      1                     0
#>    marital_status_unknown
#>                     <dbl>
#>  1                      0
#>  2                      0
#>  3                      0
#>  4                      0
#>  5                      0
#>  6                      0
#>  7                      0
#>  8                      0
#>  9                      0
#> 10                      0
#> # ... with 990 more rows
#> # i Use `print(n = ...)` to see more rows
```

Encoding ordinal data

The relationships in ordinal variables can be encoded numerically while preserving the labels in R with 'ordered factors' using `encode_ordinals()`. The numeric relations can later be extracted if fully numeric variables are required. The `ord_levels` parameter should describe the order of categories in ascending order:

```
example_data %>%
  encode_ordinals(ord_levels = c("N0","N1","N2"), n_stage) %>%
  dplyr::select(n_stage)
#> # A tibble: 1,000 x 1
#>    n_stage
#>    <ord>
#>  1 N2
#>  2 N1
#>  3 N2
#>  4 N0
#>  5 N1
#>  6 N2
#>  7 N2
#>  8 N1
#>  9 N1
#> 10 N0
#> # ... with 990 more rows
#> # i Use `print(n = ...)` to see more rows

# demonstrating how ordered factors can be converted to numeric vectors
example_data %>%
  encode_ordinals(ord_levels = c("N0","N1","N2"), n_stage) %>%
  dplyr::select(n_stage) %>%
  dplyr::mutate(dplyr::across(n_stage, as.numeric))
#> # A tibble: 1,000 x 1
#>    n_stage
#>      <dbl>
#>  1       3
#>  2       2
#>  3       3
#>  4       1
#>  5       2
#>  6       3
#>  7       3
#>  8       2
#>  9       2
#> 10       1
#> # ... with 990 more rows
#> # i Use `print(n = ...)` to see more rows
```

Encoding genotype (SNP) data

In `encode_genotypes()`, variables which record single nucleotide polymorphism (SNP) information are standardised to a "A/B" syntax. Homozygous SNPs (e.g. recorded as "A") are encoded in two character form (e.g. "A/A") while heterozygous SNPs are ordered alphabetically (e.g. "GA" becomes "A/G"). Alleles are encoded as ordinal factors, 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.

```
encode_genotypes(data = example_data, SNP_a, SNP_b) %>%
  dplyr::select(dplyr::starts_with("SNP"))
#> # A tibble: 1,000 x 2
#>    SNP_a SNP_b
#>    <ord> <ord>
#>  1 C/C   T/T
#>  2 C/C   A/T
#>  3 C/C   T/T
#>  4 C/C   T/T
#>  5 G/G   T/T
#>  6 G/G   T/T
#>  7 C/C   T/T
#>  8 G/G   T/T
#>  9 G/G   T/T
#> 10 G/G   A/A
#> # ... with 990 more rows
#> # i Use `print(n = ...)` to see more rows
```

Extract information from free text variables

Medical notes and other free text variables can contain additional information but require Natural Language Processing (NLP). Information on the presence of words, phrases, or groups of proximal words can be extracted with the functionality below; utilising the Quanteda package (Benoit et al. 2018). A knowledge of NLP terminology can be beneficial however the crucial term for this functionality is 'skipgram' which, in this context, is a series of words in a string which can have interrupting words ('skips') between them (see examples in `?quanteda::tokens_skipgrams`). The high-level function `extract_freetext()` can be applied to extract skipgrams in free text variables by their frequency.

There are three underlying stages of extracting skipgrams:

1. Identify skipgrams in a character variable (`skipgram_identify()`). The variable is also preprocessed here where:
    - Punctuation, numbers, symbols, stop-words (see `?tm::stopwords`) are removed.
    - Text is standardised to lower case
    - Words are stemmed (see `?quanteda::tokens_wordstem`).
2. Measure skipgram frequency across the variable (`skipgram_freq()`)
3. Append specified skipgrams to dataset as logical variables (`skipgram_append()`)

In medical notes, a clear signal may appear where certain skipgrams provide information suitable for analysis. The variable `free_text` in `example_data` comprises sample sentences from `stringr::sentences`. While these are not medical notes they are established examples of short pieces of text. As `free_text` does not contain true signals, we set generous parameters below: The number of interrupting words is set to five in the example below (`max_interrupt_words = 5`) however values of one or two are more likely to be useful in real-world applications. Additionally, the minimum frequency of skipgrams across the cohort to consider (`min_freq = 0.5`) is used below although 5 or 10 may be more suitable with real data. Ultimately, user evaluation and tuning is required.

```
# Identify skipgrams in example_data$free_text
skipgrams <- skipgram_identify(x = example_data$free_text,
                  ids = example_data$patient_id,
                  num_of_words = 2,
                  max_interrupt_words = 5)
skipgrams
#> # A tibble: 1,000 x 1,335
#>    doc_id need_grain mule_heal~1 gold_~2 gold_~3 ring_~4 pierc~5 gold_~6 dirti~7
#>    <chr>       <dbl>       <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
#>  1 1               1           1       0       0       0       0       0       0
#>  2 2               0           0       1       1       1       1       0       0
#>  3 3               0           0       0       0       0       0       1       0
#>  4 4               0           0       0       0       0       0       0       1
#>  5 5               0           0       0       0       0       0       0       0
#>  6 6               0           0       0       0       0       0       0       0
#>  7 7               0           0       0       0       0       0       0       0
#>  8 8               0           0       0       0       0       0       0       0
#>  9 9               0           0       0       0       0       0       0       0
#> 10 10              0           0       0       0       0       0       0       0
#>    tan_shirt bath_water weak_die weak_~8 die_f~9 stuf_~* stuf_~* chair~* move_~*
#>        <dbl>      <dbl>    <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
#>  1         0          0        0       0       0       0       0       0       0
#>  2         0          0        0       0       0       0       0       0       0
#>  3         0          0        0       0       0       0       0       0       0
#>  4         0          0        0       0       0       0       0       0       0
#>  5         1          0        0       0       0       0       0       0       0
#>  6         0          1        0       0       0       0       0       0       0
#>  7         0          0        1       1       1       0       0       0       0
#>  8         0          0        0       0       0       1       1       1       1
#>  9         0          0        0       0       0       0       0       0       0
#> 10         0          0        0       0       0       0       0       0       0
#> # ... with 990 more rows, 1,317 more variables: beam_drop <dbl>, workmen_head <dbl>,
#> #   woven_straw <dbl>, woven_mat <dbl>, straw_mat <dbl>, worn_floor <dbl>,
#> #   fish_twist <dbl>, bent_hook <dbl>, quick_snip <dbl>, abrupt_start <dbl>,
#> #   clan_gather <dbl>, dull_night <dbl>, trust_fund <dbl>, bank_earli <dbl>,
#> #   dens_crowd <dbl>, two_distinct <dbl>, two_way <dbl>, distinct_way <dbl>,
#> #   empti_flask <dbl>, empti_stood <dbl>, flask_stood <dbl>, tin_tray <dbl>,
#> #   fig_tree <dbl>, cool_green <dbl>, cool_grass <dbl>, green_grass <dbl>, ...
#> # i Use `print(n = ...)` to see more rows, and `colnames()` to see all variable names

# Summarise frequency of skipgrams to consider which should be added to the
# data.
skipgram_freq(skipgram_tokens = skipgrams, min_freq = 0.5)
#> # A tibble: 41 x 3
#>    skipgram     count percentage
#>    <chr>        <dbl>      <dbl>
#>  1 board_will       6        0.6
#>  2 leas_ran         6        0.6
#>  3 sixteen_week     6        0.6
#>  4 white_back       6        0.6
#>  5 alway_show       5        0.5
#>  6 bad_strain       5        0.5
#>  7 catch_pink       5        0.5
```

```
#>  8 catch_salmon     5        0.5
#>  9 cone_cent        5        0.5
#> 10 cone_cost        5        0.5
#> # ... with 31 more rows
#> # i Use `print(n = ...)` to see more rows

# Append chosen skipgrams to example_data
## a) by minimum frequency
skipgram_append(skipgram_tokens = skipgrams,
                id_var = patient_id,
                min_freq = 0.6,
                data = example_data)
#> `skipgrams2append` not provided. Searching for skipgrams with a `min_freq` of 0.6%
#> 4 skipgrams have been appended the data.
#> # A tibble: 1,000 x 16
#>    patient_id tumoursize t_stage n_stage diabetes diabetes_type hypertension
#>         <dbl>      <dbl> <chr>   <chr>   <chr>    <chr>         <chr>
#>  1          1         62. T3a     N2      Yes      Type I        Yes
#>  2          2         64. T3b     N1      Yes      Type II       Yes
#>  3          3         48. T1      N2      No       Type I        Yes
#>  4          4         41. T3a     N0      No       <NA>          Yes
#>  5          5         62. T4      N1      Yes      Type I        No
#>  6          6         14. T1      N2      No       <NA>          Yes
#>  7          7         63. T1      N2      No       <NA>          No
#>  8          8         44. T3b     N1      No       <NA>          No
#>  9          9         44. T2      N1      No       <NA>          Yes
#> 10         10         32. T1      N0      No       <NA>          No
#>    rural_urban marital_sta~1 SNP_a SNP_b free_~2 board~3 leas_~4 sixte~5 white~6
#>    <chr>       <chr>         <chr> <chr> <chr>     <dbl>   <dbl>   <dbl>   <dbl>
#>  1 rural       married       cc    tt    We nee~       0       0       0       0
#>  2 urban       divorced      cc    ta    The go~       0       0       0       0
#>  3 rural       divorced      cc    t     The va~       0       0       0       0
#>  4 rural       single        c     tt    Wipe t~       0       0       0       0
#>  5 urban       single        gg    t     Look i~       0       0       0       0
#>  6 urban       single        g     t     Float ~       0       0       0       0
#>  7 urban       married       c     t     Feel t~       0       0       0       0
#>  8 rural       single        gg    tt    A stuf~       0       0       0       0
#>  9 rural       single        g     t     The be~       0       0       0       0
#> 10 rural       married       g     a     Screen~       0       0       0       0
#> # ... with 990 more rows, and abbreviated variable names 1: marital_status,
#> #   2: free_text, 3: board_will, 4: leas_ran, 5: sixteen_week, 6: white_back
#> # i Use `print(n = ...)` to see more rows

## b) by specific skipgram(s)
skipgram_append(skipgram_tokens = skipgrams,
                id_var = patient_id,
                skipgrams2append = c("sixteen_week", "bad_strain"),
                data = example_data)
#> 2 skipgrams have been appended the data.
#> # A tibble: 1,000 x 14
#>    patient_id tumoursize t_stage n_stage diabetes diabetes_type hypertension
#>         <dbl>      <dbl> <chr>   <chr>   <chr>    <chr>         <chr>
#>  1          1         62. T3a     N2      Yes      Type I        Yes
```

```
#>  2          2        64. T3b     N1      Yes       Type II       Yes
#>  3          3        48. T1      N2      No        Type I        Yes
#>  4          4        41. T3a     N0      No        <NA>          Yes
#>  5          5        62. T4      N1      Yes       Type I        No
#>  6          6        14. T1      N2      No        <NA>          Yes
#>  7          7        63. T1      N2      No        <NA>          No
#>  8          8        44. T3b     N1      No        <NA>          No
#>  9          9        44. T2      N1      No        <NA>          Yes
#> 10         10        32. T1      N0      No        <NA>          No
#>    rural_urban marital_status SNP_a SNP_b free_text              sixte~1 bad_s~2
#>    <chr>       <chr>          <chr> <chr> <chr>                    <dbl>   <dbl>
#>  1 rural       married        cc    tt    We need grain to keep~       0       0
#>  2 urban       divorced       cc    ta    The gold ring fits on~       0       0
#>  3 rural       divorced       cc    t     The vamp of the shoe ~       0       0
#>  4 rural       single         c     tt    Wipe the grease off h~       0       0
#>  5 urban       single         gg    t     Look in the corner to~       0       0
#>  6 urban       single         g     t     Float the soap on top~       0       0
#>  7 urban       married        c     t     Feel the heat of the ~       0       0
#>  8 rural       single         gg    tt    A stuffed chair slipp~       0       0
#>  9 rural       single         g     t     The beam dropped down~       0       0
#> 10 rural       married        g     a     Screen the porch with~       0       0
#> # ... with 990 more rows, and abbreviated variable names 1: sixteen_week,
#> #   2: bad_strain
#> # i Use `print(n = ...)` to see more rows
```

The high-level function `extract_freetext()` is a wrapper for the low-level functions in the above example. However use of `extract_freetext()` is limited to appending skipgrams by minimum frequency and selection of skipgrams by name to append is not possible because they are not defined at the point the `extract_freetext()` function is called.:

```
extract_freetext(data = example_data,
                 id_var = patient_id,
                 min_freq = 0.6, free_text)
#> `skipgrams2append` not provided. Searching for skipgrams with a `min_freq` of 0.6%
#> 4 skipgrams have been appended the data.
#> # A tibble: 1,000 x 15
#>    patient_id tumoursize t_stage n_stage diabetes diabetes_type hypertension
#>         <dbl>      <dbl> <chr>   <chr>   <chr>    <chr>         <chr>
#>  1          1        62. T3a     N2      Yes      Type I        Yes
#>  2          2        64. T3b     N1      Yes      Type II       Yes
#>  3          3        48. T1      N2      No       Type I        Yes
#>  4          4        41. T3a     N0      No       <NA>          Yes
#>  5          5        62. T4      N1      Yes      Type I        No
#>  6          6        14. T1      N2      No       <NA>          Yes
#>  7          7        63. T1      N2      No       <NA>          No
#>  8          8        44. T3b     N1      No       <NA>          No
#>  9          9        44. T2      N1      No       <NA>          Yes
#> 10         10        32. T1      N0      No       <NA>          No
#>    rural_urban marital_status SNP_a SNP_b board_will leas_ran sixteen_~1 white~2
#>    <chr>       <chr>          <chr> <chr>      <dbl>    <dbl>      <dbl>   <dbl>
#>  1 rural       married        cc    tt             0        0          0       0
#>  2 urban       divorced       cc    ta             0        0          0       0
#>  3 rural       divorced       cc    t              0        0          0       0
```

```
#>  4 rural       single         c     tt             0        0          0       0
#>  5 urban       single         gg    t              0        0          0       0
#>  6 urban       single         g     t              0        0          0       0
#>  7 urban       married        c     t              0        0          0       0
#>  8 rural       single         gg    tt             0        0          0       0
#>  9 rural       single         g     t              0        0          0       0
#> 10 rural       married        g     a              0        0          0       0
#> # ... with 990 more rows, and abbreviated variable names 1: sixteen_week,
#> #   2: white_back
#> # i Use `print(n = ...)` to see more rows
```

Review quality control

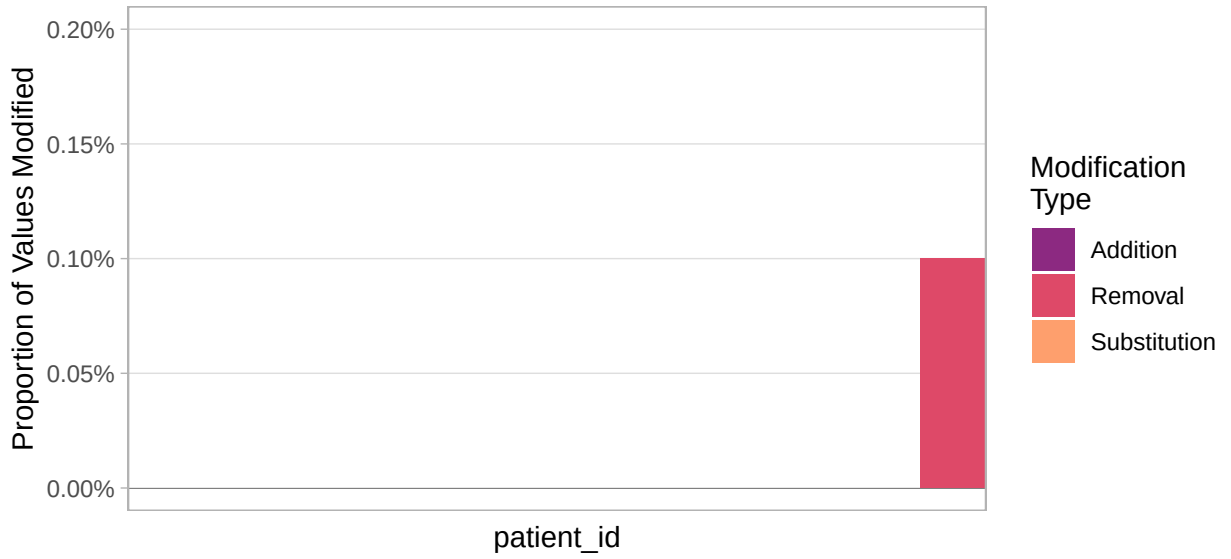
Quality control modifications may have unintended effects on the data which could remain undetected until later stages of analysis. `count_compare()` can avoid this situation by reporting changes at each step and reporting a tally of values in relevant variables. The code below uses the earlier variable merging operation (Merge Variables) as an example:

```
# merge data
example_data_merged <- merge_cols(data = example_data,
                                  primary_var = diabetes_type,
                                  secondary_var = diabetes,
                                  merge_var_name = "diabetes_merged",
                                  rm_in_vars = T)

# review this step's effects on the involved variables:
count_compare(before_tbl = example_data,
          after_tbl = example_data_merged,
          cols2compare = c("diabetes", "diabetes_type", "diabetes_merged"),
          kableout = F)
#> $before_tbl
#> # A tibble: 7 x 3
#>   diabetes diabetes_type     n
#>   <chr>    <chr>         <int>
#> 1 No       <NA>            498
#> 2 Yes      Type I          247
#> 3 Yes      Type II         244
#> 4 missing  <NA>              7
#> 5 missing  Type I            2
#> 6 missing  Type II           1
#> 7 No       Type I            1
#>
#> $after_tbl
#> # A tibble: 4 x 2
#>   diabetes_merged     n
#>   <chr>           <int>
#> 1 No                498
#> 2 Type I            250
#> 3 Type II           245
#> 4 missing             7
```

Documentation of quality control modifications is important for writing methodology and summarising changes. The remaining quality control review functions are intended for review once all quality control has

been implemented, as in Review quality control, but can be used at any point; as below with `report_var_mods()` and `mod_plot()` comparing the merging operation example and a `strings_to_NA()` example with the original data:

```
#variable level modifications
report_var_mods(before_tbl = example_data,
                after_tbl = example_data_merged)
#> # A tibble: 13 x 2
#>    variable        presence
#>    <chr>           <chr>
#>  1 patient_id      Preserved
#>  2 tumoursize      Preserved
#>  3 t_stage         Preserved
#>  4 n_stage         Preserved
#>  5 diabetes        Removed
#>  6 diabetes_type   Removed
#>  7 hypertension    Preserved
#>  8 rural_urban     Preserved
#>  9 marital_status  Preserved
#> 10 SNP_a           Preserved
#> 11 SNP_b           Preserved
#> 12 free_text       Preserved
#> 13 diabetes_merged Added

# value level modifications showing which exact missingness values
# were removed
mod_track(before_tbl = example_data,
          after_tbl = strings_to_NA(example_data),
          id_var = patient_id)
#> `vars2compare` not supplied. Attempting to compare all variables...
#> # A tibble: 78 x 6
#>    patient_id new_var old_var old_value new_value mod_type
#>    <chr>      <chr>   <chr>   <chr>     <chr>     <chr>
#>  1 31         t_stage t_stage equivocal <NA>      Removal
#>  2 34         t_stage t_stage equivocal <NA>      Removal
#>  3 44         t_stage t_stage equivocal <NA>      Removal
#>  4 48         t_stage t_stage equivocal <NA>      Removal
#>  5 261        t_stage t_stage equivocal <NA>      Removal
#>  6 263        t_stage t_stage equivocal <NA>      Removal
#>  7 348        t_stage t_stage equivocal <NA>      Removal
#>  8 454        t_stage t_stage equivocal <NA>      Removal
#>  9 468        t_stage t_stage equivocal <NA>      Removal
#> 10 569        t_stage t_stage equivocal <NA>      Removal
#> # ... with 68 more rows
#> # i Use `print(n = ...)` to see more rows

# plot value level modifications
mod_track(before_tbl = example_data,
          after_tbl = strings_to_NA(example_data),
          id_var = patient_id, plot = T)
#> `vars2compare` not supplied. Attempting to compare all variables...
```



`mod_track()` with `plot = TRUE` can visualise the extent and any disparity of value modification within the dataset.

Encoding data as numeric matrix

As a late or final quality control step, the dataset may be converted to a numeric matrix for future analysis. This will require many of the earlier steps, such as encoding categorical variables (see Encoding categorical data), to be completed. `encode_as_num_mat()` will convert all columns to numeric and use the row identifier column (`id_var`) as row names:

```
# example of data which has been quality controlled.
example_data %>%
  merge_cols(primary_var = diabetes_type,
             secondary_var = diabetes,
             merge_var_name = "diabetes_merged",
             rm_in_vars = TRUE) %>%
  apply_quality_ctrl(id_var = patient_id,
                     class_tbl = data_types_diabetes_m,
                     bin_cats =c("No" = "Yes", "rural" = "urban"),
                     min_freq = 0.6) ->
  post_qc_data
#> New names:
#> * `diabetes_merged_Type I` -> `diabetes_merged_Type.I`
#> * `diabetes_merged_Type II` -> `diabetes_merged_Type.II`

post_qc_data %>%
  encode_as_num_mat(id_var = patient_id) %>%
  tibble::glimpse()
#> Rows: 1,000
#> Columns: 19
#> $ tumoursize              <dbl> 61.71058, 64.18932, 47.81393, 40.93006, 62.11775, 1~
#> $ t_stage                 <dbl> 3, 4, 1, 3, 5, 1, 1, 4, 2, 1, 5, 4, 3, 2, 5, 3, 3, ~
#> $ n_stage                 <dbl> 3, 2, 3, 1, 2, 3, 3, 2, 2, 1, 3, 1, 3, 1, 2, 2, 1, ~
#> $ hypertension            <dbl> 2, 2, 2, 2, 1, 2, 1, 1, 2, 1, 2, 1, 1, 1, 1, 2, 1, ~
#> $ rural_urban             <dbl> 1, 2, 1, 1, 2, 2, 2, 1, 1, 1, 2, 2, 2, 2, 2, 1, 1, ~
```

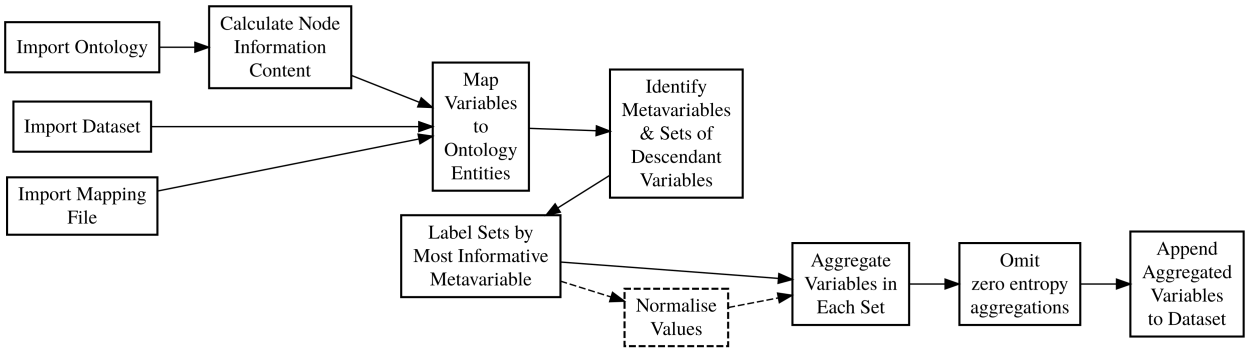
```
#> $ SNP_a                   <dbl> 1, 1, 1, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2, 1, 2, 2, 2, ~
#> $ SNP_b                   <dbl> 2, 3, 2, 2, 2, 2, 2, 2, 2, 1, 1, 3, 1, 3, 2, 1, 2, ~
#> $ board_will              <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
#> $ leas_ran                <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
#> $ sixteen_week            <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
#> $ white_back              <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
#> $ diabetes_merged_No      <dbl> 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, ~
#> $ diabetes_merged_Type.I  <dbl> 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, ~
#> $ diabetes_merged_Type.II <dbl> 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, ~
#> $ diabetes_merged_NA      <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
#> $ marital_status_divorced <dbl> 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, ~
#> $ marital_status_married  <dbl> 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, ~
#> $ marital_status_single   <dbl> 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, ~
#> $ marital_status_NA       <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
```

Note that the text labels in ordinal variables will be removed in the above conversion to a numeric matrix. The mapping between the text labels and the numerical levels can be extracted to another data frame for future reference using `ordinal_label_levels()`, as below:

```
post_qc_data %>%
  ordinal_label_levels()
#> # A tibble: 15 x 3
#>    variable label level
#>    <chr>    <chr> <dbl>
#>  1 n_stage  N0        1
#>  2 n_stage  N1        2
#>  3 n_stage  N2        3
#>  4 SNP_a    C/C       1
#>  5 SNP_a    G/G       2
#>  6 SNP_a    C/G       3
#>  7 SNP_b    A/A       1
#>  8 SNP_b    T/T       2
#>  9 SNP_b    A/T       3
#> 10 t_stage  T1        1
#> 11 t_stage  T2        2
#> 12 t_stage  T3a       3
#> 13 t_stage  T3b       4
#> 14 t_stage  T4        5
#> 15 t_stage  <NA>     NA
```

Semantic enrichment

Data frames are semantically disorganised because no information on the semantic relationships between variables is present. Biomedical ontologies contain extensive semantic information between concepts across medical domains. The semantic commonalities of a dataset's variables can be incorporated with semantic enrichment (SE). The added information may improve performance of later analysis. An overview of the workflow for SE is shown below where the "Normalise Values" box is dashed as it is an optional step:



Required inputs

SE requires three input objects:

1. A numeric dataset (data frame or matrix).
    - All variables must be numeric because SE attempts to aggregate values.
2. An ontology in R as a `igraph` or `tidygraph` object.
    - `example_ontology` is a synthetic ontology we have created to demonstrate the semantic commonalities in `example_data`.
    - At present, users must supply an ontology themselves. There are several potential ontologies for health data including SNOMED CT, the Gene Ontology, the Disease Ontology, and the Human Phenotype Ontology (Millar 2016; Gene Ontology Consortium 2019; Schriml et al. 2019; Köhler et al. 2021).
3. A mapping file (csv or data frame) which links variables in the data with entities in the ontology.
    - `example_mapping_file` is used to demonstrate SE here.
    - The variable name must not be identical to the ontological entity to which it is mapped (e.g. variable `hypertension` cannot be mapped to a ontological entity `hypertension`). This is not typically a problem as most ontologies use a numeric naming system unlikely to be used for variable names.

Example data

Examples of the three required inputs, described above, are provided with this package.

1. Because SE requires a numeric dataset, quality control is applied to `example_data`:

```
example_data %>%
  # first merge diabetes variables
  merge_cols(primary_var = diabetes_type,
             secondary_var = diabetes,
             merge_var_name = "diabetes_merged",
             rm_in_vars = TRUE) %>%
  # pass data with diabetes_merged to high-level QC function
  apply_quality_ctrl(id_var = patient_id,
                     class_tbl = data_types_diabetes_m,
                     bin_cats =c("No" = "Yes", "rural" = "urban"),
```

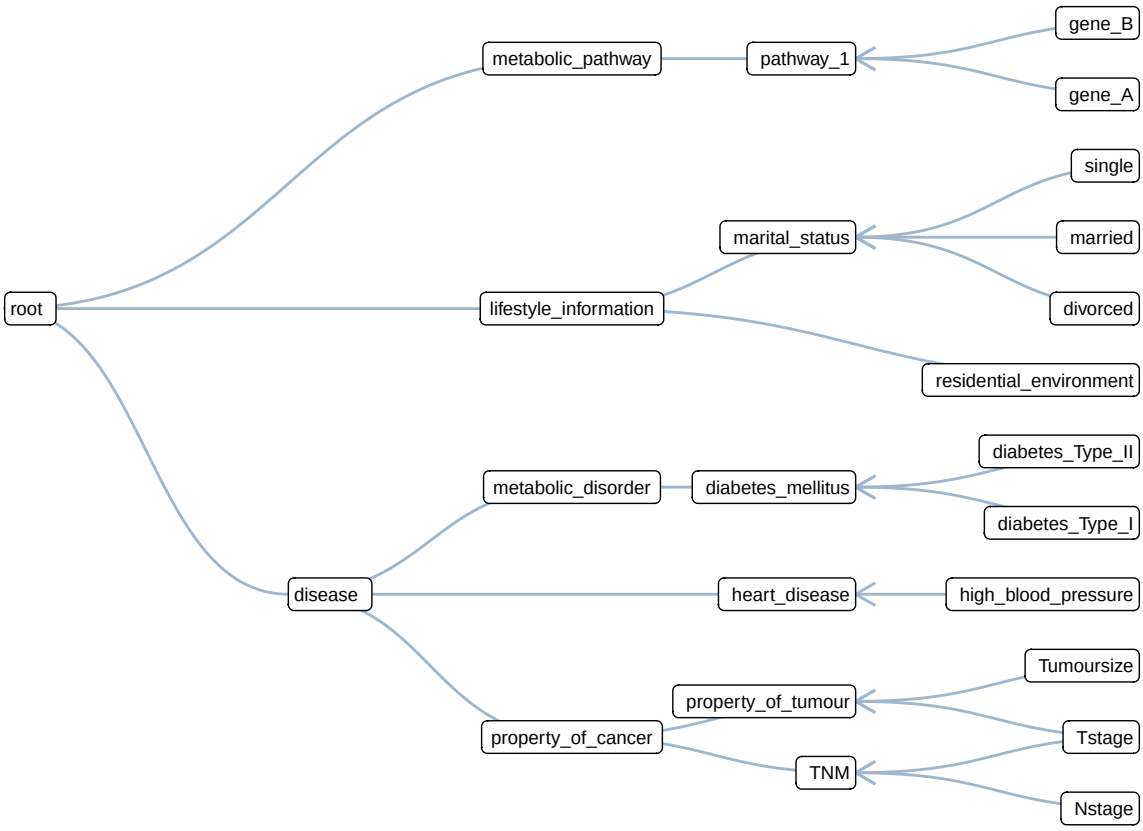
```
                      to_numeric_matrix = TRUE) ->
  post_qc_data
#> New names:
#> * `diabetes_merged_Type I` -> `diabetes_merged_Type.I`
#> * `diabetes_merged_Type II` -> `diabetes_merged_Type.II`
```

2. The example ontology containing the semantic information of variables in `example_data` is stored in `example_ontology` as a tidygraph `tbl_graph` object:

```
data(example_ontology)
example_ontology
#> # A tbl_graph: 24 nodes and 24 edges
#> #
#> # A directed acyclic simple graph with 1 component
#> #
#> # Node Data: 24 x 1 (active)
#>   name
#>   <chr>
#> 1 Nstage
#> 2 Tstage
#> 3 Tumoursize
#> 4 property_of_tumour
#> 5 TNM
#> 6 property_of_cancer
#> # ... with 18 more rows
#> #
#> # Edge Data: 24 x 2
#>    from    to
#>   <int> <int>
#> 1     1     5
#> 2     2     5
#> 3     3     4
#> # ... with 21 more rows
```

`example_ontology` is visualised in the network graph below:

```
require(ggplot2)
#> Loading required package: ggplot2
ggraph::ggraph(example_ontology, layout = "sugiyama") +
    ggraph::geom_edge_diagonal(arrow = arrow(length = unit(3, 'mm')),
                       colour = "slategray3") +
    ggraph::geom_node_label(aes(label = name),
                            size = 2.5, repel = FALSE, hjust="inward") +
    theme_void() +
    theme(legend.position = "none") +
    coord_flip()
```



3. The example mapping file, as a data frame, is as follows:

```
data(example_mapping_file)
example_mapping_file
#> # A tibble: 12 x 2
#>    variable                onto_entity
#>    <chr>                   <chr>
#>  1 tumoursize              Tumoursize
#>  2 t_stage                 Tstage
#>  3 n_stage                 Nstage
#>  4 hypertension            high_blood_pressure
#>  5 rural_urban             residential_environment
#>  6 SNP_a                   gene_A
#>  7 SNP_b                   gene_B
#>  8 diabetes_merged_Type.I  diabetes_Type_I
#>  9 diabetes_merged_Type.II diabetes_Type_II
#> 10 marital_status_divorced divorced
#> 11 marital_status_married  married
#> 12 marital_status_single   single
```

High level functionality

With the three inputs (`data`, `ontology`, and `mapping_file`) supplied, the semantic enrichment of `post_qc_data` can completed with `semantic_enrichment()`:

```
qc_se_data <- semantic_enrichment(data = post_qc_data,
                                  ontology = example_ontology,
                                  mapping_file = example_mapping_file,
                                  mode = "in",
                                  root = "root")
#> Aggregating variables by semantic commonalities and appending to `data`...
#> Identifying semantic commonalities through metavariables...
#> Complete. Duration: 1.04 secs.
#> 9 semantic commonalities found (via most informative common ancestors).
#> Complete. Duration: 26.53 secs.
#> The dataset has been enriched with 35 new variables
#> (10 new variables were not appended as they had zero entropy).
```

Below is an overview of the enriched dataset with semantic aggregations:

```
tibble::glimpse(qc_se_data)
#> Rows: 1,000
#> Columns: 50
#> $ tumoursize                  <dbl> 61.71058, 64.18932, 47.81393, 40.93006, 62.117~
#> $ t_stage                     <dbl> 3, 4, 1, 3, 5, 1, 1, 4, 2, 1, 5, 4, 3, 2, 5, 3~
#> $ n_stage                     <dbl> 3, 2, 3, 1, 2, 3, 3, 2, 2, 1, 3, 1, 3, 1, 2, 2~
#> $ hypertension                <dbl> 2, 2, 2, 2, 1, 2, 1, 1, 2, 1, 2, 1, 1, 1, 1, 2~
#> $ rural_urban                 <dbl> 1, 2, 1, 1, 2, 2, 2, 1, 1, 1, 2, 2, 2, 2, 2, 1~
#> $ SNP_a                       <dbl> 1, 1, 1, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2, 1, 2, 2~
#> $ SNP_b                       <dbl> 2, 3, 2, 2, 2, 2, 2, 2, 2, 1, 1, 3, 1, 3, 2, 1~
#> $ diabetes_merged_No          <dbl> 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0~
#> $ diabetes_merged_Type.I      <dbl> 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1~
#> $ diabetes_merged_Type.II     <dbl> 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0~
#> $ diabetes_merged_NA          <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
#> $ marital_status_divorced     <dbl> 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1~
#> $ marital_status_married      <dbl> 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0~
#> $ marital_status_single       <dbl> 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0~
#> $ marital_status_NA           <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
#> $ MV_property_of_tumour_SUM   <dbl> 1.0506338, 1.3209213, 0.4368958, 0.8805543, 1.~
#> $ MV_property_of_tumour_AVG   <dbl> 0.7004225, 0.8806142, 0.2912639, 0.5870362, 1.~
#> $ MV_property_of_tumour_MAX   <dbl> 1.0506338, 1.3209213, 0.4368958, 0.8805543, 1.~
#> $ MV_property_of_tumour_MIN   <dbl> 0.5000000, 0.5709213, 0.0000000, 0.3805543, 0.~
#> $ MV_property_of_tumour_MUL   <dbl> 0.106430411, 0.375623897, 0.000000000, 0.03295~
#> $ MV_TNM_SUM                  <dbl> 1.50, 1.25, 1.00, 0.50, 1.50, 1.00, 1.00, 1.25~
#> $ MV_TNM_AVG                  <dbl> 1.0000000, 0.8333333, 0.6666667, 0.3333333, 1.~
#> $ MV_TNM_MAX                  <dbl> 1.50, 1.25, 1.00, 0.50, 1.50, 1.00, 1.00, 1.25~
#> $ MV_TNM_MIN                  <dbl> 0.50, 0.50, 0.00, 0.00, 0.50, 0.00, 0.00, 0.50~
#> $ MV_TNM_MUL                  <dbl> 0.562500000, 0.244140625, 0.000000000, 0.00000~
#> $ MV_property_of_cancer_SUM   <dbl> 2.0506338, 1.8209213, 1.4368958, 0.8805543, 2.~
#> $ MV_property_of_cancer_AVG   <dbl> 1.0253169, 0.9104606, 0.7184479, 0.4402771, 1.~
#> $ MV_property_of_cancer_MAX   <dbl> 2.0506338, 1.8209213, 1.4368958, 0.8805543, 2.~
#> $ MV_property_of_cancer_MIN   <dbl> 0.5000000, 0.5000000, 0.0000000, 0.0000000, 0.~
#> $ MV_property_of_cancer_MUL   <dbl> 0.593522555, 0.323162519, 0.000000000, 0.00000~
#> $ MV_disease_SUM              <dbl> 4.0506338, 3.8209213, 3.4368958, 1.8805543, 3.~
#> $ MV_disease_AVG              <dbl> 1.15732395, 1.09169179, 0.98197023, 0.53730122~
#> $ MV_disease_MAX              <dbl> 4.0506338, 3.8209213, 3.4368958, 1.8805543, 3.~
#> $ MV_diabetes_mellitus_SUM    <dbl> 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1~
#> $ MV_diabetes_mellitus_AVG    <dbl> 0.6666667, 0.6666667, 0.6666667, 0.0000000, 0.~
```

```
#> $ MV_diabetes_mellitus_MAX     <dbl> 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1~
#> $ MV_marital_status_SUM        <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
#> $ MV_marital_status_AVG        <dbl> 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0~
#> $ MV_marital_status_MAX        <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
#> $ MV_lifestyle_information_SUM <dbl> 1, 2, 1, 1, 2, 2, 2, 1, 1, 1, 2, 2, 2, 2, 2, 1~
#> $ MV_lifestyle_information_AVG <dbl> 0.4, 0.8, 0.4, 0.4, 0.8, 0.8, 0.8, 0.4, 0.4, 0~
#> $ MV_lifestyle_information_MAX <dbl> 1, 2, 1, 1, 2, 2, 2, 1, 1, 1, 2, 2, 2, 2, 2, 1~
#> $ MV_pathway_1_SUM             <dbl> 0.5, 1.0, 0.5, 0.5, 1.0, 1.0, 0.5, 1.0, 1.0, 0~
#> $ MV_pathway_1_AVG             <dbl> 0.3333333, 0.6666667, 0.3333333, 0.3333333, 0.~
#> $ MV_pathway_1_MAX             <dbl> 0.5, 1.0, 0.5, 0.5, 1.0, 1.0, 0.5, 1.0, 1.0, 0~
#> $ MV_pathway_1_MIN             <dbl> 0.0, 0.0, 0.0, 0.0, 0.5, 0.5, 0.0, 0.5, 0.5, 0~
#> $ MV_pathway_1_MUL             <dbl> 0.00000000, 0.00000000, 0.00000000, 0.00000000~
#> $ MV_root_SUM                  <dbl> 5.550634, 6.820921, 4.936896, 3.380554, 6.0539~
#> $ MV_root_AVG                  <dbl> 0.8539437, 1.0493725, 0.7595224, 0.5200853, 0.~
#> $ MV_root_MAX                  <dbl> 5.550634, 6.820921, 4.936896, 3.380554, 6.0539~
```

Below is an example of how the variables `tumoursize`, `t_stage,` and `n_stage`, which all relate to cancer, have this relationship recognised through their semantic commonality of `property_of_cancer` in `example_ontology`:

```
qc_se_data %>%
  dplyr::select(tumoursize, t_stage, n_stage,
                dplyr::starts_with("MV_property_of_cancer")) %>%
                tibble::glimpse()
#> Rows: 1,000
#> Columns: 8
#> $ tumoursize                <dbl> 61.71058, 64.18932, 47.81393, 40.93006, 62.11775,~
#> $ t_stage                   <dbl> 3, 4, 1, 3, 5, 1, 1, 4, 2, 1, 5, 4, 3, 2, 5, 3, 3~
#> $ n_stage                   <dbl> 3, 2, 3, 1, 2, 3, 3, 2, 2, 1, 3, 1, 3, 1, 2, 2, 1~
#> $ MV_property_of_cancer_SUM <dbl> 2.0506338, 1.8209213, 1.4368958, 0.8805543, 2.053~
#> $ MV_property_of_cancer_AVG <dbl> 1.0253169, 0.9104606, 0.7184479, 0.4402771, 1.026~
#> $ MV_property_of_cancer_MAX <dbl> 2.0506338, 1.8209213, 1.4368958, 0.8805543, 2.053~
#> $ MV_property_of_cancer_MIN <dbl> 0.5000000, 0.5000000, 0.0000000, 0.0000000, 0.500~
#> $ MV_property_of_cancer_MUL <dbl> 0.593522555, 0.323162519, 0.000000000, 0.00000000~
```

Note:

- The normalisation of values prevents `tumoursize`'s large magnitude (relative to the other variables) having a disproportional effect on the aggregations.
- The prefix "MV_" stands for "meta-variable".

In summary, the SE process added 35 aggregation variables to the dataset from 9 meta-variables.

Low level functionality

There are some exported lower level functions which users may find useful to see the intermediate steps of SE. The should be carried out in the order shown below:

Join ontology and data variable names

Nodes representing variable names in the dataset can be joined to the ontology to which they have been mapped with `join_vars_to_ontol()`. Prior to joining, this function calculates the information content of each ontological entity using the equation below, developed by Zhou, Wang, and Gu (2008):

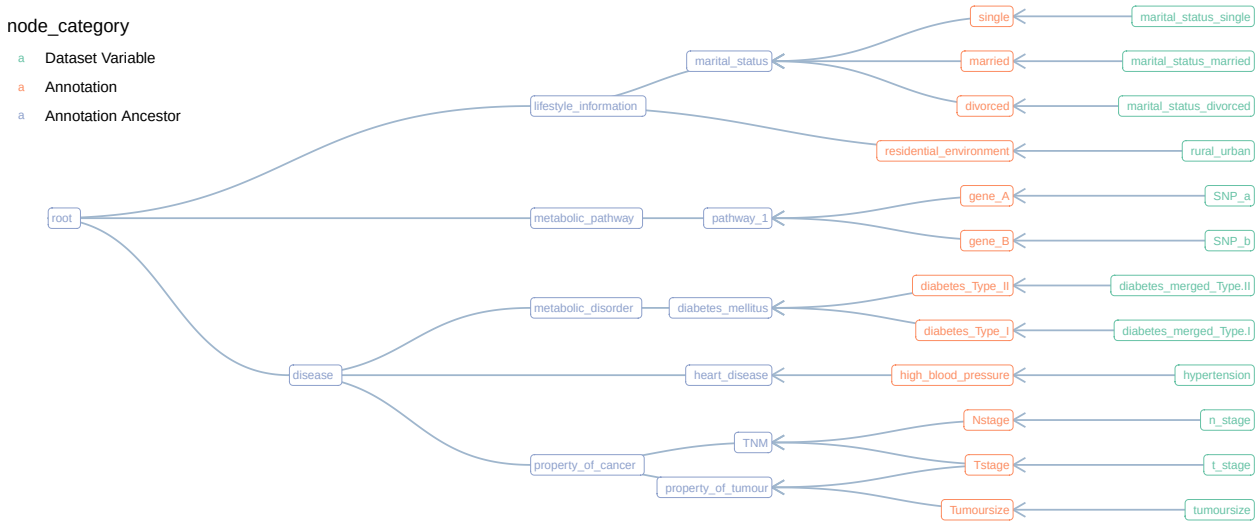
$$IC(c) = k(1 - \frac{\log(hypo(c) + 1)}{\log(node_{\max})}) + (1 - k)(\frac{\log(deep(c)}{log(deep_{\max})})$$

Where c is an ontological entity in `ontol_graph`, $hypo(c)$ is the number of hyponyms (descendants) of c, $node_{\max}$ is the total number of ontological entities in `ontol_graph`, $deep(c)$ is the depth of c (distance from `root`), $deep_{\max}$ is the maximum depth in `ontol_graph`, and k is an adjustable factor to adjust the weight of the two terms (default = 0.5); a higher k value will reduce the importance/impact of c's relative depth in the ontology.

```
joined_nw <- join_vars_to_ontol(ontol_graph = example_ontology,
                                var2entity_tbl = example_mapping_file,
                                root = "root", k = 0.5)
#> Warning in dfs(graph = graph, root = root, neimode = mode, unreachable =
#> unreachable, : Argument `neimode' is deprecated; use `mode' instead
```

This network, with the variable names added as nodes, can be visualised as below. The `node_category` node attribute can be used to colour the nodes:

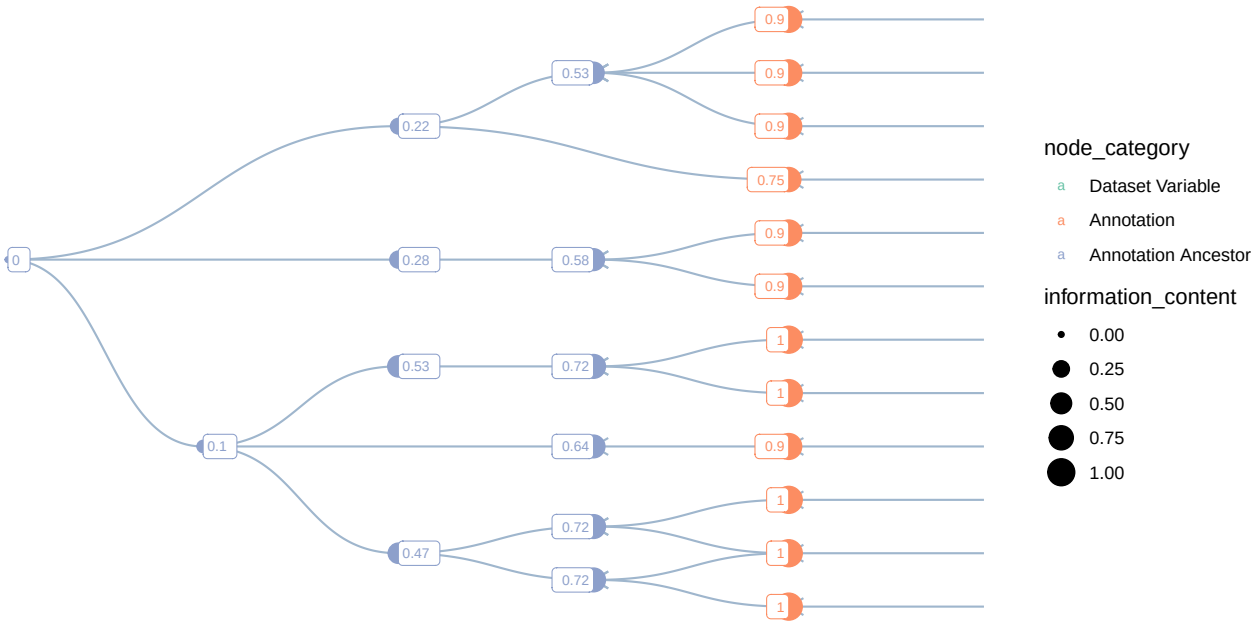
```
ggraph::ggraph(joined_nw, layout = "sugiyama") +
    ggraph::geom_edge_diagonal(arrow = arrow(length = unit(3, 'mm')),
                       colour = "slategray3") +
    ggraph::geom_node_label(aes(
      label = name, color = node_category), size = 2.5,
      repel = F, hjust="inward") +
    theme_void() +
    scale_color_brewer(palette = "Set2") +
    coord_flip() +
    theme(legend.position = c(0.08, 0.85))
```



Node information content can be visualised, as below. Information content is not calculated for dataset variables because they are not part of the original ontology, therefore their node size is 0. This visualisation helps demonstrate how nodes/concepts further down the ontology are more informative than those higher up.

This calculation benefits SE as the common ancestor of a set of variable nodes with the highest information content is chosen to label the group. In the middle branch from the root node, there are two annotation ancestor nodes which are multiple common ancestors of two variables (`SNP_a` and `SNP_b`). The node with the higher information content, `pathway_1`, is chosen to label the semantic commonality between these variables over the less informative node, `metabolic_pathway`.

```
ggraph::ggraph(joined_nw, layout = "sugiyama") +
    ggraph::geom_edge_diagonal(arrow = arrow(length = unit(3, 'mm')),
                         colour = "slategray3") +
    ggraph::geom_node_point(aes(
      color = node_category, size = information_content)) +
    ggraph::geom_node_label(aes(
      label = round(information_content,2),
      color = node_category),  size = 2.5, hjust="inward") +
    scale_color_brewer(palette = "Set2") +
    theme_void() +
    coord_flip()
```



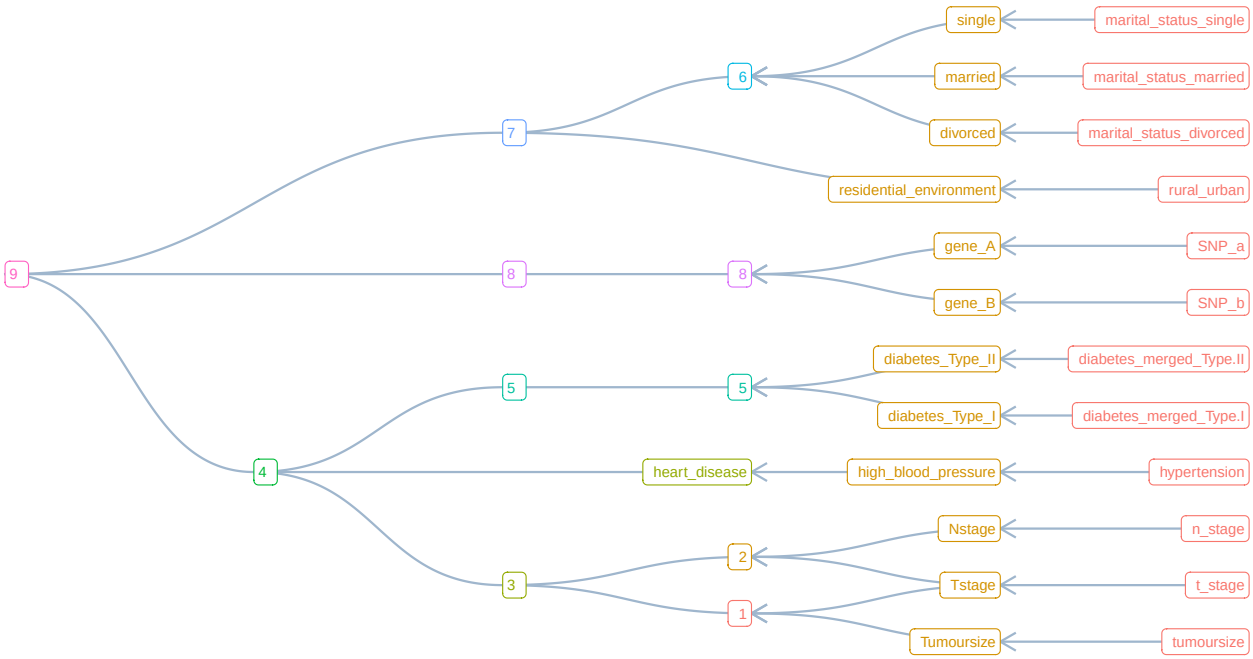
Compute meta-variable information

"Meta-variable" is defined here as an ontological entity which is the most informative common ancestor of two or more variables in the joined network. Meta-variables are identified in `metavariable_info()` by first determining the unique sets of variable nodes which are descendants of ontological nodes. The information content (IC, calculated above) of nodes which share the same set of variable descendants is compared and the node with the highest IC is used to label the set.

`example_ontology` links variable nodes in `joined_nw` to identify nine sets, shown in the graph below:

```
example_ontology %>%
  join_vars_to_ontol(var2entity_tbl = example_mapping_file, root = "root") %>%
  metavariable_info() ->
  metavariables_nw
#> Identifying semantic commonalities through metavariables...
#> Warning in dfs(graph = graph, root = root, neimode = mode, unreachable =
#> unreachable, : Argument `neimode' is deprecated; use `mode' instead
#> Complete. Duration: 0.95 secs.
#> 9 semantic commonalities found (via most informative common ancestors).
```

```
metavariables_nw %>%
  # annotations are also considered a set. This isn't helpful for this visualisation
  # Therefore, the sets of non-meta-variables are removed below
  tidygraph::mutate(variable_set = ifelse(!is_metavariable, NA, variable_set)) %>%
  tidygraph::mutate(variable_set = as.factor(variable_set)) %>%
  ggraph::ggraph(layout = "sugiyama") +
    ggraph::geom_edge_diagonal(arrow = arrow(length = unit(3, 'mm')),
                         colour = "slategray3") +
    ggraph::geom_node_label(aes(label = ifelse(is_metavariable,
                                       as.factor(as.numeric(variable_set)),
                                       name),
                          color = ifelse(is_metavariable,
                                       as.character(as.numeric(variable_set)),
                                       node_category)),
                      repel = F, size = 2.5, hjust="inward") +
    theme_void() +
    theme(legend.position = "none") +
    coord_flip()
```



Note how variable sets 5 and 8 each have two nodes which share the same set of variables. The information

content, calculated during `join_vars_to_ontol()`, of these nodes is compared and the node with the highest information content is used to label the set.

Generate semantic aggregations

With the sets of variables identified and the meta-variable used to label each set confirmed, the next step is to perform the aggregations. This functionality requires the previously described low-level SE functions and is overlaps with the `semantic_enrichment()` function, but does not carry out some of the checks.

```
example_ontology %>%
    join_vars_to_ontol(var2entity_tbl = example_mapping_file, root = "root") %>%
    metavariable_info() %>%
    metavariable_agg(data = post_qc_data) ->
    qc_se_data
#> Aggregating variables by semantic commonalities and appending to `data`...
#> Identifying semantic commonalities through metavariables...
#> Warning in dfs(graph = graph, root = root, neimode = mode, unreachable =
#> unreachable, : Argument `neimode' is deprecated; use `mode' instead
#> Complete. Duration: 0.98 secs.
#> 9 semantic commonalities found (via most informative common ancestors).
#> Complete. Duration: 26.59 secs.
#> The dataset has been enriched with 35 new variables
#> (10 new variables were not appended as they had zero entropy).

## summary of output
tibble::glimpse(qc_se_data)
#> Rows: 1,000
#> Columns: 50
#> $ tumoursize                  <dbl> 61.71058, 64.18932, 47.81393, 40.93006, 62.117~
#> $ t_stage                     <dbl> 3, 4, 1, 3, 5, 1, 1, 4, 2, 1, 5, 4, 3, 2, 5, 3~
#> $ n_stage                     <dbl> 3, 2, 3, 1, 2, 3, 3, 2, 2, 1, 3, 1, 3, 1, 2, 2~
#> $ hypertension                <dbl> 2, 2, 2, 2, 1, 2, 1, 1, 2, 1, 2, 1, 1, 1, 1, 2~
#> $ rural_urban                 <dbl> 1, 2, 1, 1, 2, 2, 2, 1, 1, 1, 2, 2, 2, 2, 2, 1~
#> $ SNP_a                       <dbl> 1, 1, 1, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2, 1, 2, 2~
#> $ SNP_b                       <dbl> 2, 3, 2, 2, 2, 2, 2, 2, 2, 1, 1, 3, 1, 3, 2, 1~
#> $ diabetes_merged_No          <dbl> 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0~
#> $ diabetes_merged_Type.I      <dbl> 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1~
#> $ diabetes_merged_Type.II     <dbl> 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0~
#> $ diabetes_merged_NA          <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
#> $ marital_status_divorced     <dbl> 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1~
#> $ marital_status_married      <dbl> 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0~
#> $ marital_status_single       <dbl> 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0~
#> $ marital_status_NA           <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
#> $ MV_property_of_tumour_SUM   <dbl> 1.0506338, 1.3209213, 0.4368958, 0.8805543, 1.~
#> $ MV_property_of_tumour_AVG   <dbl> 0.7004225, 0.8806142, 0.2912639, 0.5870362, 1.~
#> $ MV_property_of_tumour_MAX   <dbl> 1.0506338, 1.3209213, 0.4368958, 0.8805543, 1.~
#> $ MV_property_of_tumour_MIN   <dbl> 0.5000000, 0.5709213, 0.0000000, 0.3805543, 0.~
#> $ MV_property_of_tumour_MUL   <dbl> 0.106430411, 0.375623897, 0.000000000, 0.03295~
#> $ MV_TNM_SUM                  <dbl> 1.50, 1.25, 1.00, 0.50, 1.50, 1.00, 1.00, 1.25~
#> $ MV_TNM_AVG                  <dbl> 1.0000000, 0.8333333, 0.6666667, 0.3333333, 1.~
#> $ MV_TNM_MAX                  <dbl> 1.50, 1.25, 1.00, 0.50, 1.50, 1.00, 1.00, 1.25~
#> $ MV_TNM_MIN                  <dbl> 0.50, 0.50, 0.00, 0.00, 0.50, 0.00, 0.00, 0.50~
#> $ MV_TNM_MUL                  <dbl> 0.562500000, 0.244140625, 0.000000000, 0.00000~
```

```
#> $ MV_property_of_cancer_SUM    <dbl> 2.0506338, 1.8209213, 1.4368958, 0.8805543, 2.~
#> $ MV_property_of_cancer_AVG    <dbl> 1.0253169, 0.9104606, 0.7184479, 0.4402771, 1.~
#> $ MV_property_of_cancer_MAX    <dbl> 2.0506338, 1.8209213, 1.4368958, 0.8805543, 2.~
#> $ MV_property_of_cancer_MIN    <dbl> 0.5000000, 0.5000000, 0.0000000, 0.0000000, 0.~
#> $ MV_property_of_cancer_MUL    <dbl> 0.593522555, 0.323162519, 0.000000000, 0.00000~
#> $ MV_disease_SUM               <dbl> 4.0506338, 3.8209213, 3.4368958, 1.8805543, 3.~
#> $ MV_disease_AVG               <dbl> 1.15732395, 1.09169179, 0.98197023, 0.53730122~
#> $ MV_disease_MAX               <dbl> 4.0506338, 3.8209213, 3.4368958, 1.8805543, 3.~
#> $ MV_diabetes_mellitus_SUM     <dbl> 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1~
#> $ MV_diabetes_mellitus_AVG     <dbl> 0.6666667, 0.6666667, 0.6666667, 0.0000000, 0.~
#> $ MV_diabetes_mellitus_MAX     <dbl> 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1~
#> $ MV_marital_status_SUM        <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
#> $ MV_marital_status_AVG        <dbl> 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0~
#> $ MV_marital_status_MAX        <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
#> $ MV_lifestyle_information_SUM <dbl> 1, 2, 1, 1, 2, 2, 2, 1, 1, 1, 2, 2, 2, 2, 2, 1~
#> $ MV_lifestyle_information_AVG <dbl> 0.4, 0.8, 0.4, 0.4, 0.8, 0.8, 0.8, 0.4, 0.4, 0~
#> $ MV_lifestyle_information_MAX <dbl> 1, 2, 1, 1, 2, 2, 2, 1, 1, 1, 2, 2, 2, 2, 2, 1~
#> $ MV_pathway_1_SUM             <dbl> 0.5, 1.0, 0.5, 0.5, 1.0, 1.0, 0.5, 1.0, 1.0, 0~
#> $ MV_pathway_1_AVG             <dbl> 0.3333333, 0.6666667, 0.3333333, 0.3333333, 0.~
#> $ MV_pathway_1_MAX             <dbl> 0.5, 1.0, 0.5, 0.5, 1.0, 1.0, 0.5, 1.0, 1.0, 0~
#> $ MV_pathway_1_MIN             <dbl> 0.0, 0.0, 0.0, 0.0, 0.5, 0.5, 0.0, 0.5, 0.5, 0~
#> $ MV_pathway_1_MUL             <dbl> 0.00000000, 0.00000000, 0.00000000, 0.00000000~
#> $ MV_root_SUM                  <dbl> 5.550634, 6.820921, 4.936896, 3.380554, 6.0539~
#> $ MV_root_AVG                  <dbl> 0.8539437, 1.0493725, 0.7595224, 0.5200853, 0.~
#> $ MV_root_MAX                  <dbl> 5.550634, 6.820921, 4.936896, 3.380554, 6.0539~
```

References

Benoit, Kenneth, Kohei Watanabe, Haiyan Wang, Paul Nulty, Adam Obeng, Stefan Müller, and Akitaka Matsuo. 2018. "Quanteda: An r Package for the Quantitative Analysis of Textual Data." *Journal of Open Source Software* 3 (30): 774. https://doi.org/10.21105/joss.00774.

Gene Ontology Consortium, The. 2019. "The Gene Ontology Resource: 20 Years and Still GOing Strong." *Nucleic Acids Research* 47 (D1): D330–38. https://doi.org/10.1093/nar/gky1055.

Köhler, Sebastian, Michael Gargano, Nicolas Matentzoglu, Leigh C. Carmody, David Lewis-Smith, Nicole A. Vasilevsky, Daniel Danis, et al. 2021. "The Human Phenotype Ontology in 2021." *Nucleic Acids Research* 49 (D1): D1207–17. https://doi.org/10.1093/nar/gkaa1043.

Kolde, Raivo. 2019. *Pheatmap: Pretty Heatmaps*. https://CRAN.R-project.org/package=pheatmap.

Millar, Jane. 2016. "The Need for a Global Language - SNOMED CT Introduction." *Studies in Health Technology and Informatics* 225: 683–85.

Roebuck, Kevin. 2011. *Data Quality: High-Impact Strategies - What You Need to Know: Definitions, Adoptions, Impact, Benefits, Maturity, Vendors*. Lightning Source Incorporated.

Schriml, Lynn M., Elvira Mitraka, James Munro, Becky Tauber, Mike Schor, Lance Nickle, Victor Felix, et al. 2019. "Human Disease Ontology 2018 update: classification, content and workflow expansion." *Nucleic Acids Research* 47 (D1): D955–62. https://doi.org/10.1093/nar/gky1032.

Shannon, C. E. 1948. "A Mathematical Theory of Communication." *The Bell System Technical Journal* 27 (3): 379–423. https://doi.org/10.1002/j.1538-7305.1948.tb01338.x.

Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemund, et al. 2019. "Welcome to the Tidyverse." *Journal of Open Source Software* 4 (43): 1686. https://doi.org/10.21105/joss.01686.

Zhou, Zili, Yanna Wang, and Junzhong Gu. 2008. "2008 Second International Conference on Future Generation Communication and Networking Symposia." In, 3:85–89. https://doi.org/10.1109/FGCNS.2008.16.