Package ‘codebook’

January 8, 2025

Title Automatic Codebooks from Metadata Encoded in Dataset Attributes

Description Easily automate the following tasks to describe data frames:
Summarise the distributions, and labelled missings of variables graphically and using descriptive statistics.
For surveys, compute and summarise reliabilities (internal consistencies, retest, multilevel) for psychological scales.
Combine this information with metadata (such as item labels and labelled values) that is derived from R attributes.
To do so, the package relies on 'rmarkdown' partials, so you can generate HTML, PDF, and Word documents.
Codebooks are also available as tables (CSV, Excel, etc.) and in JSON-LD, so that search engines can find your data and index the metadata.
The metadata are also available at your fingertips via RStudio Addins.

Version 0.9.6

Depends R (>= 3.2.0)

Language en-GB

URL https://rubenarslan.github.io/codebook/,
https://github.com/rubenarslan/codebook

BugReports https://github.com/rubenarslan/codebook/issues

License MIT + file LICENSE

Imports stats, methods, graphics, utils, rmdpartials, forcats (>= 0.4.0), vctrs (>= 0.3.0), ggplot2 (>= 2.0.0), stringr, rlang, dplyr (>= 1.0.0), tidyr, jsonlite, haven (>= 2.3.0), purrr, tibble, glue, likert, knitr, skimr (>= 2.1.0), htmltools, labeling, labelled, future

Suggests testthat, DT, shinytest, lme4, rmarkdown, rstudioapi (>= 0.5), shiny (>= 0.13), miniUI (>= 0.1.1), roxygen2, rio, psych, ufs, rosetta, GGally

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

VignetteBuilder knitr

NeedsCompilation no

Author Ruben Arslan [aut, cre]

Maintainer Ruben Arslan <ruben.arslan@gmail.com>

Repository CRAN

Date/Publication 2025-01-08 08:10:08 UTC

Contents

add_R . . . 3
aggregate_and_document_scale . . . 3
bfi . . . 4
codebook . . . 4
codebook_browser . . . 6
codebook_component_scale . . . 6
codebook_component_single_item . . . 7
codebook_data_info . . . 8
codebook_items . . . 8
codebook_missingness . . . 9
codebook_survey_overview . . . 10
codebook_table . . . 10
compact_codebook . . . 11
compute_reliabilities . . . 11
data_description_default . . . 12
detect_missing . . . 13
detect_scales . . . 14
ended . . . 15
expired . . . 16
get_skimmers.haven_labelled . . . 16
get_skimmers.haven_labelled_spss . . . 17
has_label . . . 17
has_labels . . . 18
knit_print.alpha . . . 18
knit_print.htest . . . 19
knit_print.multilevel . . . 19
label_browser . . . 20
label_browser_static . . . 20
likert_from_items . . . 21
list_to_dict . . . 21
load_data_and_render_codebook . . . 22
md_pattern . . . 22
metadata . . . 23
metadata_jsonld . . . 24
metadata_list . . . 24
modified . . . 25
new_codebook_rmd . . . 25

plot_labelled 26
rescue_attributes 26
reverse_labelled_values 27
skim_codebook 28
to_factor 28
zap_attributes 29
zap_labelled 30

Index **31**

---

add_R   *Append R to string, if it doesn't end in R already.*

Description

Use this function to conveniently rename reverse-coded variables, so that they end in R.

Usage

```
add_R(x)
```

Arguments

x	a string

Examples

```
data('bfi')
bfi %>% dplyr::select(BFIK_open_2,BFIK_agree_2) %>% dplyr::rename_at(1, add_R) %>% head()
```

---

aggregate_and_document_scale
Aggregate variables and remember which variables this were

Description

The resulting variables will have the attribute `scale_item_names` containing the basis for aggregation. Its `label` attribute will refer to the common stem of the aggregated variable names (if any), the number of variables, and the aggregation function.

Usage

```
aggregate_and_document_scale(items, fun = rowMeans, stem = NULL)
```

Arguments

`items`	data.frame of the items that should be aggregated
`fun`	aggregation function, defaults to rowMeans with na.rm = FALSE
`stem`	common stem for the variables, specify if it should not be auto-detected as the longest common stem of the variable names

Examples

```
testdf <- data.frame(bfi_neuro_1 = rnorm(20), bfi_neuro_2 = rnorm(20),
                    bfi_neuro_3R = rnorm(20), age = rpois(20, 30))
item_names <- c('bfi_neuro_1', 'bfi_neuro_2', 'bfi_neuro_3R')
testdf$bfi_neuro <- aggregate_and_document_scale(testdf[, item_names])
testdf$bfi_neuro
```

`bfi` *Mock BFI data*

Description

a small mock BFI dataset with realistic values, exported from formr. The dataset is self-documenting via its attributes.

Usage

```
bfi
```

Format

A data frame with 28 rows and 29 variables:

`codebook` *Generate rmarkdown codebook*

Description

Pass a data frame to this function to make a codebook for that dataset. If the dataset has metadata (attributes) set on its variables, these will be used to make the codebook more informative. Examples are item, value, and missing labels. Data frames imported via `haven::read_dta()`, `haven::read_sav()`, or from formr.org will have these attributes in the right format. By calling this function inside a knitr code chunk, the codebook will become part of the document you are generating.

Usage

```
codebook(
  results,
  reliabilities = NULL,
  survey_repetition = c("auto", "single", "repeated_once", "repeated_many"),
  detailed_variables = TRUE,
  detailed_scales = TRUE,
  survey_overview = TRUE,
  missingness_report = TRUE,
  metadata_table = TRUE,
  metadata_json = TRUE,
  indent = "#"
)
```

Arguments

`results`	a data frame, ideally with attributes set on variables
`reliabilities`	a named list with one entry per scale and one or several printable reliability computations for this scale. if NULL, computed on-the-fly using compute_reliabilities
`survey_repetition`	defaults to "auto" which is to try to determine the level of repetition from the "session" and "created" variables. Other values are: single, repeated_once, repeated_many
`detailed_variables`	whether to print a graph and summary for each variable
`detailed_scales`	whether to print a graph and summary for each scale
`survey_overview`	whether to print an overview of survey entries, durations (depends on presence of columns session, created, modified, ended, expired)
`missingness_report`	whether to print a missingness report. Turn off if this gets too complicated and you need a custom solution (e.g. in case of random missings).
`metadata_table`	whether to print a metadata table/tabular codebook.
`metadata_json`	whether to include machine-readable metadata as JSON-LD (not visible)
`indent`	add # to this to make the headings in the components lower-level. defaults to beginning at h2

Examples

```
# will generate figures in a temporary directory
## Not run:
data("bfi")
bfi <- bfi[, c("BFIK_open_1", "BFIK_open_1")]
md <- codebook(bfi, survey_repetition = "single", metadata_table = FALSE)

## End(Not run)
```

codebook_browser *Browse and search codebook*

Description

Usable as an Addin in RStudio. You can select it from a menu at the top, when this package is installed. If you're currently selecting the name of a data frame in your source code, this will be the dataset shown by default. If you don't select text, you can pick a dataset from a dropdown. You can add a keyboard shortcut for this command by following the instructions by RStudio. How about Cmd+Ctrl+C?

Usage

```
codebook_browser(
  data = NULL,
  labels_only = FALSE,
  title = "Codebook metadata",
  viewer = rstudioapi::viewer
)
```

Arguments

`data`	the dataset to display. If left empty will try to use selected text in RStudio or offer a dropdown
`labels_only`	defaults to false called with TRUE from `label_browser()`
`title`	title of the gadget
`viewer`	defaults to displaying in the RStudio viewer

codebook_component_scale *Codebook component for scales*

Description

Codebook component for scales

Usage

```
codebook_component_scale(
  scale,
  scale_name = deparse(substitute(scale)),
  items,
  reliabilities = list(),
  indent = "##"
)
```

Arguments

`scale`	a scale with attributes set
`scale_name`	the variable name of this scale
`items`	a data.frame with the items constituting the scale
`reliabilities`	a list with one or several results from calls to psych package functions for computing reliability
`indent`	add # to this to make the headings in the components lower-level. defaults to beginning at h2

Examples

```
# will generate figures in a temporary directory
## Not run:
data("bfi")
bfi <- bfi[,c("BFIK_open", paste0("BFIK_open_", 1:4))]
codebook_component_scale(bfi[,1], "BFIK_open", bfi[,-1],
   reliabilities = list(BFIK_open = psych::alpha(bfi[,-1])))

## End(Not run)
```

`codebook_component_single_item` *Codebook component for single items*

Description

Codebook component for single items

Usage

```
codebook_component_single_item(
  item,
  item_name = deparse(substitute(item)),
  indent = "##"
)
```

Arguments

`item`	an item with attributes set
`item_name`	the item name
`indent`	add # to this to make the headings in the components lower-level. defaults to beginning at h2

Examples

```
## Not run:
data("bfi")
codebook_component_single_item(bfi$BFIK_open_1, "BFIK_open_1")

## End(Not run)
```

---

codebook_data_info *Codebook data info*

Description

A readout of the metadata for this dataset, with some defaults set

Usage

```
codebook_data_info(results, indent = "##")
```

Arguments

results	a data frame which has the following columns: session, created, modified, expired, ended
indent	add # to this to make the headings in the components lower-level. defaults to beginning at h2

Examples

```
# will generate figures in a figure/ subdirectory
data("bfi")
metadata(bfi)$name <- "MOCK Big Five Inventory dataset (German metadata demo)"
metadata(bfi)$description <- "a small mock Big Five Inventory dataset"
metadata(bfi)$citation <- "doi:10.5281/zenodo.1326520"
metadata(bfi)$url <-
   "https://rubenarslan.github.io/codebook/articles/codebook.html"
codebook_data_info(bfi)
```

---

codebook_items *Tabular codebook*

Description

Renders a tabular codebook including attributes and data summaries. The table is generated using `DT::datatable()` and can be exported to CSV, Excel, etc.

Usage

```
codebook_items(results, indent = "##")
```

Arguments

`results`	a data frame, ideally with attributes set on variables
`indent`	add # to this to make the headings in the components lower-level. defaults to beginning at h2

Examples

```
data("bfi")
## Not run:
# doesn't show interactively, because a html widget needs to be registered
codebook_items(bfi)

## End(Not run)
```

`codebook_missingness` *Codebook missingness*

Description

An overview table of missingness patterns generated using `md_pattern()`.

Usage

```
codebook_missingness(results, indent = "##")
```

Arguments

`results`	a data frame
`indent`	add # to this to make the headings in the components lower-level. defaults to beginning at h2

Examples

```
data("bfi")
codebook_missingness(bfi)
```

codebook_survey_overview *Codebook survey overview*

Description

An overview of the number of rows and groups, and of the durations participants needed to respond (if those data are available).

Usage

```
codebook_survey_overview(results, survey_repetition = "single", indent = "##")
```

Arguments

`results`	a data frame which has all the following columns: session, created, modified, expired, ended
`survey_repetition`	defaults to single (other values: repeated_once, repeated_many). controls whether internal consistency, retest reliability or multilevel reliability is computed
`indent`	add # to this to make the headings in the components lower-level. defaults to beginning at h2

Examples

```
## Not run:
data("bfi")
codebook_survey_overview(bfi)

## End(Not run)
```

codebook_table *Codebook metadata table*

Description

will generate a table combining metadata from variable attributes with data summaries generated using `skimr::skim()`

Usage

```
codebook_table(results)
```

Arguments

`results`	a data frame, ideally with attributes set on variables

Examples

```
data("bfi")
codebook_table(bfi)
```

`compact_codebook` *Compact Codebook*

Description

Generate only the tabular codebook and the machine-readable JSON-LD metadata.

Usage

```
compact_codebook(results)
```

Arguments

`results`	the data frame

Examples

```
# will generate figures in a figure/ subdirectory
## Not run:
data("bfi")
bfi <- bfi[, c("BFIK_open_1", "BFIK_open_2")]
compact_codebook(bfi)

## End(Not run)
```

`compute_reliabilities` *Compute reliabilities*

Description

If you pass the object resulting from a call to formr_results to this function, it will compute reliabilities for each scale. Internally, each reliability computation is passed to a `future::future()`. If you are calculating multilevel reliabilities, it may be worthwhile to parallelise this operation using `future::plan()`. If you don't plan on any complicated parallelisation, you probably do not need to call this function directly, but can rely on it being automatically called during codebook generation. If you do plan to do that, you can pass the results of this operation to the codebook function.

Usage

```
compute_reliabilities(results, survey_repetition = "single", use_psych = TRUE)
```

Arguments

`results`	a formr results table with attributes set on items and scales
`survey_repetition`	defaults to "single". Can also be "repeated_once" or "repeated_many"
`use_psych`	compute reliabilities using the psych package, defaults to TRUE. if false, will use rosetta (computationally more expensive, more dependencies)

Examples

```
data("bfi", package = "codebook")
bfi <- bfi %>% dplyr::select(dplyr::starts_with("BFIK_agree"))
reliabilities <- compute_reliabilities(bfi)
```

`data_description_default` *Data description default*

Description

If you do not define a dataset description yourself, this will be the automatically generated default.

Usage

```
data_description_default(data)
```

Arguments

`data`	the data frame

Examples

```
data('bfi')
data_description_default(bfi)
```

`detect_missing` *Detect missing values*

Description

SPSS users frequently label their missing values, but don't set them as missing. This function will rectify that for negative values and for the values 99 and 999 (only if they're 5*MAD away from the median). Using different settings, you can also easily tag other missing values.

SPSS users frequently label their missing values, but don't set them as missing. This function will rectify that for negative values and for the values 99 and 999 (only if they're 5*MAD away from the median). Using different settings, you can also easily tag other missing values.

Usage

```
detect_missing(
  data,
  only_labelled = TRUE,
  negative_values_are_missing = TRUE,
  ninety_nine_problems = TRUE,
  learn_from_labels = TRUE,
  missing = c(),
  non_missing = c(),
  vars = names(data),
  use_labelled_spss = FALSE,
  coerce_integer_to_double = FALSE,
  verbose = FALSE
)

detect_missings(data, only_labelled_missings = TRUE, ...)
```

```
detect_missing(
  data,
  only_labelled = TRUE,
  negative_values_are_missing = TRUE,
  ninety_nine_problems = TRUE,
  learn_from_labels = TRUE,
  missing = c(),
  non_missing = c(),
  vars = names(data),
  use_labelled_spss = FALSE,
  coerce_integer_to_double = FALSE,
  verbose = FALSE
)
```

Arguments

`data`	the data frame with labelled missing values

`only_labelled`	don't set values to missing if there's no label for them
`negative_values_are_missing`	by default we label negative values as missing
`ninety_nine_problems`	SPSS users often store values as 99/999, should we do this for values with 5*MAD of the median
`learn_from_labels`	if there are labels for missing values of the form `[-1] no answer`, set -1 in the data to the corresponding tagged missing
`missing`	also set these values to missing (or enforce for 99/999 within 5*MAD)
`non_missing`	don't set these values to missing
`vars`	only edit these variables
`use_labelled_spss`	the labelled_spss class has a few drawbacks. Since R can't store missing values like -1 and 99, we're replacing them with letters unless this option is enabled. If you prefer to keep your -1 etc, turn this on.
`coerce_integer_to_double`	By default, missing values in the columns of integers are not labelled, because it's not technically possible. Let this parameter be `TRUE` if you want to automatically coerce integer columns into double to be able to label the missing values.
`verbose`	defaults to FALSE, if set to true, the function lets you know where and how it found potential missing values
`only_labelled_missings`	passed to `detect_missing()`
`...`	passed to `detect_missing()`

Functions

- `detect_missings()`: Deprecated version

`detect_scales` *Detect item scales*

Description

Did you create aggregates of items like this `scale <- scale_1 + scale_2R + scale_3R`? If you run this function on a dataset, it will detect these relationships and set the appropriate attributes. Once they are set, the codebook package can perform reliability computations for you.

Did you create aggregates of items like this `scale <- scale_1 + scale_2R + scale_3R`? If you run this function on a dataset, it will detect these relationships and set the appropriate attributes. Once they are set, the codebook package can perform reliability computations for you.

Usage

```
detect_scales(data, quiet = FALSE)

detect_scales(data, quiet = FALSE)
```

Arguments

`data`	the data frame
`quiet`	defaults to false. Suppresses messages about found items.

Examples

```
bfi <- data.frame(matrix(data = rnorm(300), ncol = 3))
names(bfi) <- c("bfi_e1", "bfi_e2R", "bfi_e3")
bfi$bfi_e <- rowMeans(bfi[, c("bfi_e1", "bfi_e2R", "bfi_e3")])
bfi <- detect_scales(bfi)
bfi$bfi_e
bfi <- data.frame(matrix(data = rnorm(300), ncol = 3))
names(bfi) <- c("bfi_e1", "bfi_e2R", "bfi_e3")
bfi$bfi_e <- rowMeans(bfi[, c("bfi_e1", "bfi_e2R", "bfi_e3")])
bfi <- detect_scales(bfi)
bfi$bfi_e
```

`ended` *How many surveys were ended?*

Description

Just a simple to check how many times a survey (e.g. diary) was finished. It defaults to checking the "ended" variable for this.

Usage

```
ended(survey, variable = "ended")
```

Arguments

`survey`	which survey are you asking about?
`variable`	which variable should be filled out, defaults to "ended"

Examples

```
survey <- data.frame(ended = c("2016-05-28 10:11:00", NA, "2016-05-30 11:18:28"))
ended(survey = survey)
```

`expired` *How many surveys were expired?*

Description

Just a simple to check how many times a survey (e.g. diary) has expired (i.e. user missed it). It defaults to checking the "expired" variable for this.

Usage

```
expired(survey, variable = "expired")
```

Arguments

`survey`	which survey are you asking about?
`variable`	which variable should be filled out, defaults to "expired"

Examples

```
survey <- data.frame(expired = c(NA, "2016-05-29 10:11:00", NA))
expired(survey = survey)
```

`get_skimmers.haven_labelled` *Define skimmers for haven_labelled variables*

Description

Variables labelled using the haven_labelled class are special because the underlying data can be numeric or character. This skimmers summarises both and leaves non-pertinent columns missings.

Usage

```
get_skimmers.haven_labelled(column)
```

Arguments

`column`	the column to skim

`get_skimmers.haven_labelled_spss` *Define skimmers for haven_labelled_spss variables*

Description

Variables labelled using the haven_labelled_spss class are special because the underlying data can be numeric or character. This skimmers summarises both and leaves non-pertinent columns missings.

Usage

```
get_skimmers.haven_labelled_spss(column)
```

Arguments

`column`	the column to skim

`has_label` *Has label*

Description

Has label

Usage

```
has_label(x)
```

Arguments

`x`	a vector

Examples

```
example("labelled", "haven")
has_label(x)
```

`has_labels` *Has labels*

Description

Has labels

Usage

```
has_labels(x)
```

Arguments

`x`	a vector

Examples

```
example("labelled", "haven")
has_labels(x)
```

`knit_print.alpha` *Pretty-print a Cronbach's alpha object*

Description

Turn a `psych::alpha()` object into HTML tables.

Usage

```
knit_print.alpha(x, indent = "######", ...)
```

Arguments

`x`	a psych alpha object
`indent`	add # to this to make the headings in the components lower-level. defaults to beginning at h5
`...`	ignored

Examples

```
example("alpha", "psych")
knitr::knit_print(a4)
```

`knit_print.htest` *Print a `stats::cor.test()` object for knitr*

Description

Just prints the normal output of `stats::cor.test()`.

Usage

```
knit_print.htest(x, ...)
```

Arguments

`x`	a psych alpha object
`...`	ignored

Examples

```
knitr::knit_print(cor.test(rnorm(100), rnorm(100)))
```

`knit_print.multilevel` *Print a `psych::multilevel.reliability()` object for knitr*

Description

Just prints the normal output of `psych::multilevel.reliability()`.

Usage

```
knit_print.multilevel(x, ...)
```

Arguments

`x`	a psych alpha object
`...`	ignored

Examples

```
example("mlr", "psych")
knitr::knit_print(mg)
```

label_browser *Browse and search variable and value labels*

Description

Same as the `codebook_browser()`, but doesn't show data summaries and additional attributes.

Usage

```
label_browser(data = NULL, viewer = rstudioapi::viewer)
```

Arguments

`data`	the dataset to display. If left empty will try to use selected text in RStudio or offer a dropdown
`viewer`	defaults to displaying in the RStudio viewer

label_browser_static *Browse and search variable and value labels*

Description

Same as the `codebook_browser()`, but doesn't show data summaries and additional attributes. This yields a static table, so you can continue to edit code while viewing the labels, but you cannot switch the dataset via a dropdown menu.

Usage

```
label_browser_static(data = NULL, viewer = rstudioapi::viewer)
```

Arguments

`data`	data frame. if left empty, will use the text you currently select in RStudio as the label or the first data frame in your environment
`viewer`	where to show. defaults to viewer tab

Examples

```
label_browser_static(bfi)
```

likert_from_items *Derive a likert object from items*

Description

Pass a data.frame containing several items composing one scale, get a `likert::likert()` object, which you can plot. Intelligently makes use of labels and value labels if present.

Usage

```
likert_from_items(items)
```

Arguments

`items`	a data frame of items composing one scale

Examples

```
data("bfi", package = "codebook")
open_items <- paste0("BFIK_open_",1:4)
graphics::plot(likert_from_items(bfi[, open_items]))
```

list_to_dict *Go from a named list to a key-value data frame or data dictionary and back*

Description

Sometimes, you'll want to have variable labels in a data.frame, sometimes you'll have imported an existing data dictionary and will need to turn it into a list before setting `labelled::var_label()`.

Usage

```
list_to_dict(named_list)

dict_to_list(dict)
```

Arguments

`named_list`	a named list with one element each (names being variable names, elements being labels)
`dict`	a data frame with the variable names in the first and the labels in the second column. If they are named variable and label, they can also be in a different order.

Examples

```
data('bfi')
labels <- var_label(bfi)
head(labels, 2)
dict <- list_to_dict(labels)
head(dict, 2)
head(dict_to_list(list_to_dict(labels)), 2)
```

`load_data_and_render_codebook` *Submit a data file and an rmarkdown template as a file to generate a codebook. Used chiefly in the webapp.*

Description

Submit a data file and an rmarkdown template as a file to generate a codebook. Used chiefly in the webapp.

Usage

```
load_data_and_render_codebook(file, text, remove_file = FALSE, ...)
```

Arguments

`file`	path to a file to make codebook from (sav, rds, dta, por, xpt, csv, csv2, tsv, etc.)
`text`	codebook template
`remove_file`	whether to remove file after rendering
`...`	all other arguments passed to `rmarkdown::render()`

`md_pattern` *Missing data patterns*

Description

Generate missingness patterns using a function borrowed from mice, with options to reduce the complexity of the output.

Usage

```
md_pattern(data, omit_complete = TRUE, min_freq = 0.01)
```

Arguments

`data`	the dataset
`omit_complete`	defaults to TRUE, omitting variables without missing values
`min_freq`	minimum number of rows to have this missingness pattern

Examples

```
data("bfi", package = 'psych')
md_pattern(bfi)
md_pattern(bfi, omit_complete = FALSE, min_freq = 0.2)
```

`metadata` *Add metadata to a dataset*

Description

Use this function to describe a data frame in preparation for JSON-LD metadata generation using `codebook()` or `metadata_list()`.

Usage

```
metadata(data)

metadata(data) <- value
```

Arguments

`data`	the data frame
`value`	the metadata attribute

Examples

```
data('bfi')
metadata(bfi)$name <- "MOCK Big Five Inventory dataset (German metadata demo)"
metadata(bfi)$description <- "a small mock Big Five Inventory dataset"
metadata(bfi)$identifier <- "doi:10.5281/zenodo.1326520"
metadata(bfi)$datePublished <- "2016-06-01"
metadata(bfi)$creator <- list(
  "@type" = "Person",
  givenName = "Ruben", familyName = "Arslan",
  email = "ruben.arslan@gmail.com",
  affiliation = list("@type" = "Organization",
                     name = "MPI Human Development, Berlin"))
metadata(bfi)$citation <- "Arslan (2016). Mock BFI data."
metadata(bfi)$url <-
  "https://rubenarslan.github.io/codebook/articles/codebook.html"
metadata(bfi)$temporalCoverage <- "2016"
metadata(bfi)$spatialCoverage <- "Goettingen, Germany"
metadata(bfi)$keywords <- c("Personality", "Psychology")
metadata(bfi)
```

`metadata_jsonld` *Metadata as JSON-LD*

Description

Echo a list of a metadata, generated using `metadata_list()` as JSON-LD in a script tag.

Usage

```
metadata_jsonld(results)
```

Arguments

`results`	a data frame, ideally with attributes set on variables

Examples

```
data("bfi")
metadata_jsonld(bfi)
```

`metadata_list` *Metadata from dataframe*

Description

Returns a list containing variable metadata (attributes) and data summaries.

Usage

```
metadata_list(results, only_existing = TRUE)
```

Arguments

`results`	a data frame, ideally with attributes set on variables
`only_existing`	whether to drop helpful metadata to comply with the list of currently defined schema.org properties

Examples

```
data("bfi")
md_list <- metadata_list(bfi)
md_list$variableMeasured[[20]]
```

`modified` *How many surveys were modified?*

Description

Just a simple to check how many times a survey (e.g. diary) has expired (i.e. user missed it). It defaults to checking the "expired" variable for this.

Usage

```
modified(survey, variable = "modified")
```

Arguments

`survey`	which survey are you asking about?
`variable`	which variable should be filled out, defaults to "modified"

Examples

```
survey <- data.frame(modified = c(NA, "2016-05-29 10:11:00", NA))
modified(survey = survey)
```

`new_codebook_rmd` *Create a codebook rmarkdown document*

Description

This function will create and open an .Rmd file in the current working directory. By default, the file is named codebook.Rmd. No files will be overwritten. The .Rmd file has some useful defaults set for creating codebooks.

Usage

```
new_codebook_rmd(filename = "codebook", template = "default")
```

Arguments

`filename`	under which file name do you want to create a template
`template`	only "default" exists for now

Examples

```
## Not run:
new_codebook_rmd()

## End(Not run)
```

`plot_labelled` *Plot labelled vector*

Description

Plot a labelled vector, making use of the variable name, label and value labels to make the plot more readable. This function also works for other vectors, but provides little benefit.

Usage

```
plot_labelled(
  item,
  item_name = NULL,
  wrap_at = 70,
  go_vertical = FALSE,
  trans = "identity",
  x_axis_label = "values"
)
```

Arguments

`item`	a vector
`item_name`	item name, defaults to name of first argument
`wrap_at`	the subtitle (the label) will be wrapped at this number of characters
`go_vertical`	defaults to FALSE. Whether to show choices on the Y axis instead.
`trans`	defaults to "identity" passed to `ggplot2::scale_x_continuous()`
`x_axis_label`	defaults to "values"

Examples

```
data("bfi", package = "codebook")
plot_labelled(bfi$BFIK_open_1)
```

`rescue_attributes` *Rescue lost attributes*

Description

You can use this function if some of your items have lost their attributes during wrangling Variables have to have the same name (Duh) and no attributes should be overwritten. But use with care. Similar to `labelled::copy_labels()`.

You can use this function if some of your items have lost their attributes during wrangling Variables have to have the same name (Duh) and no attributes should be overwritten. But use with care. Similar to `labelled::copy_labels()`.

Usage

```
rescue_attributes(df_no_attributes, df_with_attributes)

rescue_attributes(df_no_attributes, df_with_attributes)
```

Arguments

`df_no_attributes`	the data frame with missing attributes
`df_with_attributes`	the data frame from which you want to restore attributes

`reverse_labelled_values`

Reverse labelled values reverse the underlying values for a numeric `haven::labelled()` *vector while keeping the labels correct*

Description

Reverse labelled values reverse the underlying values for a numeric `haven::labelled()` vector while keeping the labels correct

Reverse labelled values reverse the underlying values for a numeric `haven::labelled()` vector while keeping the labels correct

Usage

```
reverse_labelled_values(x)

reverse_labelled_values(x)
```

Arguments

`x`	a labelled vector

Value

return the labelled vector with the underlying values having been reversed

return the labelled vector with the underlying values having been reversed

Examples

```
x <- haven::labelled(rep(1:3, each = 3), c(Bad = 1, Good = 5))
x
reverse_labelled_values(x)
x <- haven::labelled(rep(1:3, each = 3), c(Bad = 1, Good = 5))
x
reverse_labelled_values(x)
```

skim_codebook *Skim codebook*

Description

Implements the regular functionality of `skimr::skim()` but renames the columns p0, p50, and p100 to min, median, and max respectively for numeric types to keep things consistent across type (and produce a narrower wide table).

Usage

```
skim_codebook(data, ...)
```

Arguments

`data`	the dataset to skim
`...`	passed to `skimr::skim()`

Examples

```
skim_codebook(bfi)
```

to_factor *To factor*

Description

Convert a labelled vector to a factor, even if it doesn't have the proper class, as long as it has the "labels" attribute. You can have this attribute without, for example, the haven_labelled class, when importing data with `rio::import()` for example.

Usage

```
to_factor(x, ...)
```

Arguments

`x`	a vector
`...`	passed to `haven::as_factor()`

Examples

```
example("labelled", "haven")
to_factor(x)
to_factor(zap_labelled(x))
to_factor(as_factor(x))
to_factor(1:4)
```

zap_attributes *Zap attributes*

Description

Modelled on haven::zap_labels(), but more encompassing. By default removes the following attributes: format.spss, format.sas, format.stata, label, labels, na_values, na_range, display_width

Modelled on haven::zap_labels(), but more encompassing. By default removes the following attributes: format.spss, format.sas, format.stata, label, labels, na_values, na_range, display_width

Usage

```
zap_attributes(
  x,
  attributes = c("format.spss", "format.sas", "format.stata", "label", "labels",
    "na_values", "na_range", "display_width")
)
```

```
zap_attributes(
  x,
  attributes = c("format.spss", "format.sas", "format.stata", "label", "labels",
    "na_values", "na_range", "display_width")
)
```

Arguments

`x`	the data frame or variable
`attributes`	character vector of attributes to zap. NULL if everything (including factor levels etc) should be zapped

Examples

```
bfi <- data.frame(matrix(data = rnorm(300), ncol = 3))
names(bfi) <- c("bfi_e1", "bfi_e2R", "bfi_e3")
attributes(bfi$bfi_e1)$label <- "I am outgoing."
attributes(bfi$bfi_e2R)$label <- "I prefer books to people."
attributes(bfi$bfi_e3)$label <- "I love to party."
bfi$bfi_e <- rowMeans(bfi[, c("bfi_e1", "bfi_e2R", "bfi_e3")])
bfi <- detect_scales(bfi, quiet = TRUE) # create attributes
str(zap_attributes(bfi, "label"))
zap_attributes(bfi$bfi_e)
bfi <- data.frame(matrix(data = rnorm(300), ncol = 3))
names(bfi) <- c("bfi_e1", "bfi_e2R", "bfi_e3")
attributes(bfi$bfi_e1)$label <- "I am outgoing."
attributes(bfi$bfi_e2R)$label <- "I prefer books to people."
attributes(bfi$bfi_e3)$label <- "I love to party."
bfi$bfi_e <- rowMeans(bfi[, c("bfi_e1", "bfi_e2R", "bfi_e3")])
bfi <- detect_scales(bfi, quiet = TRUE) # create attributes
```

```
str(zap_attributes(bfi, "label"))
zap_attributes(bfi$bfi_e)
```

`zap_labelled` *Zap labelled class*

Description

Modelled on `haven::zap_labels()`, zaps labelled class (not other attributes).

Modelled on `haven::zap_labels()`, zaps labelled class (not other attributes).

Usage

```
zap_labelled(x)
```

```
zap_labelled(x)
```

Arguments

`x`	the data frame or variable

Index

∗ **datasets**
    bfi, 4

add_R, 3
aggregate_and_document_scale, 3

bfi, 4

codebook, 4
codebook(), *23*
codebook_browser, 6
codebook_browser(), *20*
codebook_component_scale, 6
codebook_component_single_item, 7
codebook_data_info, 8
codebook_items, 8
codebook_missingness, 9
codebook_survey_overview, 10
codebook_table, 10
compact_codebook, 11
compute_reliabilities, 11

data_description_default, 12
detect_missing, 13
detect_missing(), *14*
detect_missings *(detect_missing)*, 13
detect_scales, 14
dict_to_list *(list_to_dict)*, 21
DT::datatable(), *8*

ended, 15
expired, 16

future::future(), *11*
future::plan(), *11*

get_skimmers.haven_labelled, 16
get_skimmers.haven_labelled_spss, 17
ggplot2::scale_x_continuous(), *26*

has_label, 17
has_labels, 18
haven::as_factor(), *28*
haven::labelled(), *27*
haven::read_dta(), *4*
haven::read_sav(), *4*
haven::zap_labels(), *29, 30*

knit_print.alpha, 18
knit_print.htest, 19
knit_print.multilevel, 19

label_browser, 20
label_browser(), *6*
label_browser_static, 20
labelled::copy_labels(), *26*
labelled::var_label(), *21*
likert::likert(), *21*
likert_from_items, 21
list_to_dict, 21
load_data_and_render_codebook, 22

md_pattern, 22
md_pattern(), *9*
metadata, 23
metadata<- *(metadata)*, 23
metadata_jsonld, 24
metadata_list, 24
metadata_list(), *23, 24*
modified, 25

new_codebook_rmd, 25

plot_labelled, 26
psych::alpha(), *18*
psych::multilevel.reliability(), *19*

rescue_attributes, 26
reverse_labelled_values, 27
rio::import(), *28*
rmarkdown::render(), *22*

skim_codebook, 28
skimr::skim(), *10*, *28*
stats::cor.test(), *19*

to_factor, 28

zap_attributes, 29
zap_labelled, 30