Package 'NBDCtools'

November 10, 2026

Title National Institutes of Health Brain Development Cohorts Data Hub Tools

Description A suite of functions to work with data from the National Institutes of Health Brain Development Cohorts Data Hub. The package provides tools to create, clean, process, and filter datasets and associated metadata. These utilities are intended to simplify reproducible data-preparation for future research.

URL https://software.nbdc-datahub.org/NBDCtools/

Version 1.1.2

Additional_repositories https://nbdc-datahub.r-universe.dev

Depends R (>= 4.3.0)

Imports arrow, chk, cli, dplyr, glue, magrittr, purrr, readr, stringr, sjlabelled, jsonlite, hms, tidyr, rlang, utils, tibble, stats, sjmisc, haven, crayon

Suggests testthat (>= 3.0.0), usethis, knitr, rmarkdown, naniar, NBDCtoolsData

License GPL (>= 3)

Encoding UTF-8

Config/testthat/edition 3

Config/Needs/website rmarkdown

LazyData true

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Le Zhang [aut, cre] (ORCID: <https://orcid.org/0009-0008-0205-2150>), Janosch Linkersdoerfer [aut] (ORCID: <https://orcid.org/0000-0002-1577-1233>)

Maintainer Le Zhang <dairc.service@gmail.com>

Repository CRAN

Date/Publication 2026-11-10 20:00:07 UTC

Contents

add_custom_metadata . . . 2
convert_names_data . . . 3
convert_names_file . . . 5
create_bids_sidecar_data . . . 7
create_bids_sidecar_metadata . . . 8
create_dataset . . . 9
filter_empty_cols . . . 13
filter_empty_rows . . . 14
filter_events_abcd . . . 15
filter_id_events . . . 16
get_dd . . . 18
get_id_cols . . . 19
get_levels . . . 20
get_metadata . . . 21
get_releases . . . 22
get_sessions . . . 22
join_tabulated . . . 24
read_dsv_formatted . . . 27
shadow_bind_data . . . 28
shadow_replace_binding_missing . . . 29
transf_factor . . . 30
transf_label . . . 31
transf_time_to_hms . . . 32
transf_value_to_label . . . 33
transf_value_to_na . . . 34

Index **36**

`add_custom_metadata` *Add custom metadata*

Description

This function allows users to add custom metadata (data dictionary, levels table, sessions table) to the package environment. This can be useful for users who want to use their own metadata instead of the ones provided by the package, or for testing and development purposes.

Usage

```
add_custom_metadata(dd = NULL, levels = NULL, sessions = NULL)
```

Arguments

`dd`	data frame. Custom data dictionary. Should have the same structure as the data dictionary provided by the package.
`levels`	data frame. Custom levels table.
`sessions`	data frame. Custom sessions table.

Details

The custom metadata will be stored in the package environment can be accessed with any function that contains the "release" argument with `release = "custom"`. For example, `get_dd(study = "abcd", release = "custom")`.

The default value for `dd`, `levels`, and `sessions` is `NULL`. If any of them is not `NULL`, it will be added to the package environment. If the argument is `NULL`, the corresponding metadata will not be added.

Value

invisible `TRUE`.

Examples

```
add_custom_metadata(
  dd = tibble::tibble(
    name = "var1",
    table_name = "table1",
    identifier_columns = "participant_id"
  )
)
get_dd(study = "abcd", release = "custom")
```

---

convert_names_data *Convert column names in a data frame*

Description

This function renames columns in a data frame to another type of column name specified in the data dictionary.

For example, this can be used to convert the ABCD column names introduced in the 6.0 release to the previously used column names. If you instead want to convert the column names in a file, use `convert_names_file()`.

Note: Please use this function with caution and make sure that the data in the converted column is equivalent to the data in the original column. Also, please make sure that the names can be mapped one-to-one. Some variables in the ABCD data dictionary have been collapsed from previous releases and thus might have multiple names in the `name_to` column that map to a single name (see `skip_sep_check` argument below).

Usage

```
convert_names_data(
  data,
  dd,
  name_from = "name",
  name_to,
  ignore_cols = union(get_id_cols_abcd(), get_id_cols_hbcd()),
  skip_sep_check = FALSE
)
```

Arguments

data	tibble. The input data frame with columns to be renamed.
dd	tibble. The data dictionary table. One can use get_dd() family of functions to get the data dictionary for a given study and release or provide a custom data dictionary.
name_from	character. The column name type in the data dictionary that the columns in data currently use (Default: "name"). This column must exist in the data dictionary.
name_to	character. The column name type in the data dictionary that the columns in data should be renamed to. This column must exist in the data dictionary.
ignore_cols	character vector. The columns to ignore (Default: identifier columns used in ABCD and HBCD).
skip_sep_check	logical. Whether to skip the check for name_to column's separators validation. In our official data dictionaries, some columns have multiple names separated by a "|" in the same cell. For columns with multiple names, it the recommended to use functions like tidyr::separate_rows() to split the names into multiple rows and decide which name to use for the renaming by filtering the rows, so that the name_from and name_to columns are one-to-one mapping. If skip_sep_check = FALSE (default), the function will check if the name_from or name_to columns have the "|" separator and will throw an error if the separator is found. If skip_sep_check = TRUE, it means you understand that character strings with "|" inside will be used for rename mapping, and the function will not check for the separator.

Value

tibble. The data with renamed column names.

Examples

```
## Not run:
# rename columns to previous ABCD names used by NDA
convert_names_data(
  data,
  dd = get_dd("abcd"),
  name_from = "name",
  name_to = "name_nda"
)

# rename columns to Stata names
convert_names_data(
  data,
  dd = get_dd("abcd"),
  name_from = "name",
  name_to = "name_stata"
)

## End(Not run)
```

convert_names_file *Convert column names in a file*

Description

This function replaces all matched column names in a file with another type of column name specified in the data dictionary.

For example, this function can be used to convert script files that specified previously used column names to the the ABCD column names introduced in the 6.0 release. If you instead want to convert the column names in a data frame, use `convert_names_data()`.

Note: Please use this function with caution and make sure that the data in the converted column is equivalent to the data in the original column. Also, please make sure that the names can be mapped one-to-one. Some variables in the ABCD data dictionary have been collapsed from previous releases and thus might have multiple names in the `name_from` column that map to a single name (see `skip_sep_check` argument below).

Usage

```
convert_names_file(
  file_in,
  file_out = NULL,
  dd,
  name_from,
  name_to,
  skip_sep_check = FALSE
)
```

Arguments

`file_in`	character. The input file path.
`file_out`	character. The output file path. If not provided, defaults to the input file path with a "_converted" suffix.
`dd`	tibble. The data dictionary table. One can use `get_dd()` family of functions to get the data dictionary for a given study and release or provide a custom data dictionary.
`name_from`	character. The column name type in the data dictionary that the columns in `data` currently use (Default: `"name"`). This column must exist in the data dictionary.
`name_to`	character. The column name type in the data dictionary that the columns in `data` should be renamed to. This column must exist in the data dictionary.
`skip_sep_check`	logical. Whether to skip the check for `name_from` and `name_to` columns' separators validation. In our official data dictionaries, some columns have multiple names separated by a `"|"` in the same cell. For columns with multiple names, it the recommended to use functions like `tidyr::separate_rows()` to split the names into multiple rows and decide

which name to use for the renaming by filtering the rows, so that the `name_from` and `name_to` columns are one-to-one mapping.

If `skip_sep_check = FALSE` (default), the function will check if the `name_from` or `name_to` columns have the `"|"` separator and will throw an error if the separator is found. If `skip_sep_check = TRUE`, it means you understand that character strings with `"|"` inside will be used for rename mapping, and the function will not check for the separator.

Details

Word matching:

The function uses word boundaries to match the names in the file. It Uses regex word boundaries (\\b) to ensure exact word matches. This prevents partial matches within larger words. For example, matching "age" will not match "cage" or "page".

Speed:

The data dictionary is big from `get_dd()`, so the function would loop through all the names in the data dictionary. If there are only a few names to replace, it is the best to trim the data dictionary to only those names before using this function.

Value

character. The path to the output file with converted names, invisible.

Examples

```
## Not run:
convert_names_file(
  file_in = "analysis_script.R",
  dd = get_dd("abcd"),
  name_from = "name_nda",
  name_to = "name"
)

# Specify custom output file
convert_names_file(
  file_in = "analysis_script.py",
  file_out = "analysis_script_new.py",
  dd = get_dd("abcd"),
  name_from = "name_nda",
  name_to = "name"
)

## End(Not run)
```

create_bids_sidecar_data *Create BIDS sidecar from labelled data*

Description

Creates a Brain Imaging Data Structure (BIDS) JSON sidecar file from the metadata (data dictionary and levels table). Returns the JSON object or writes it to a file.

Usage

```
create_bids_sidecar_data(
  data,
  study,
  release = "latest",
  var_coding = "values",
  metadata_description = "Dataset exported using NBDCtools",
  path_out = NULL,
  pretty = TRUE
)
```

Arguments

data	tibble. The raw data or data with labels, see transf_label(). If the data is not labelled, the function will try to label the data first.
study	character. NBDC study (One of "abcd" or "hbcd")
release	character. Release version (Default: "latest").
var_coding	character. the variable coding, one of "values", "labels". If the data is processed with transf_value_to_label(), please use "labels".
metadata_description	string, the description of the metadata
path_out	character. the path to the output file. If NULL, the function will return the json object.
pretty	logical. Whether to pretty print the json.

Details

- If you have a labelled dataset, and want to create data specific BIDS sidecar with variable levels from the data, please use create_bids_sidecar_data().
- If you want to create a BIDS sidecar without the underlying data, please use create_bids_sidecar_metadata().

Value

the json object or the path to the json file

See Also

create_bids_sidecar_metadata()

Examples

```
## Not run:
data |> create_bids_sidecar_data()
data |> create_bids_sidecar_data(path_out = "data.json")

## End(Not run)
```

---

create_bids_sidecar_metadata

Create BIDS sidecar from metadata

Description

Generates a Brain Imaging Data Structure (BIDS) JSON sidecar using metadata tables (data dictionary and levels) without requiring the underlying data.

Usage

```
create_bids_sidecar_metadata(
  dd,
  levels,
  vars = NULL,
  tables = NULL,
  metadata_description = "Dataset exported using NBDCtools",
  path_out = NULL,
  pretty = TRUE
)
```

Arguments

dd	tibble, Data dictionary metadata, see get_dd() or provide a custom data dictionary. The custom data dictionary must have same structure as the official data dictionaries.
levels	tibble, Levels metadata corresponding to dd, see get_levels(), or provide a custom levels table. The custom levels table must have same structure as the official levels tables.
vars	character vector, variable names to include. vars and tables cannot be both NULL.
tables	character vector, table names to include.
metadata_description	string, the description of the metadata
path_out	character. the path to the output file. If NULL, the function will return the json object.
pretty	logical. Whether to pretty print the json.

Details

- If you have a labelled dataset, and want to create data specific BIDS sidecar with variable levels from the data, please use `create_bids_sidecar_data()`.
- If you want to create a BIDS sidecar without the underlying data, please use `create_bids_sidecar_metadata()`.

Value

Either a JSON string (when `path_out` is `NULL`) or the output file path (invisibly) after writing the JSON sidecar to disk.

See Also

`create_bids_sidecar_data()`

Examples

```
create_bids_sidecar_metadata(
  dd = get_dd("abcd"),
  levels = get_levels("abcd"),
  tables = c("ph_y_mctq")
)
```

`create_dataset` *Create a dataset*

Description

This high-level function simplifies the process of creating a dataset from the ABCD or HBCD Study data by allowing users to create an analysis-ready dataset in a single step. It executes the lower-level functions provided in the `NBDCtools` package in sequence to load, join, and transform the data.

The function expects study data to be stored as one `.parquet` or `.tsv` file per database table within a specified directory, provided as `dir_data`. Variables specified in `vars` and `tables` will be full-joined together, while variables specified in `vars_add` and `tables_add` will be left-joined to these variables. For more details, see `join_tabulated()`.

In addition to the main `create_dataset()` function, there are two study-specific variations:

- `create_dataset_abcd()`: for the ABCD study.
- `create_dataset_hbcd()`: for the HBCD study.

They have the same arguments as the `create_dataset()` function, except that the `study` argument is set to the respective study by default, and should not be set by the user.

Usage

```
create_dataset(
  dir_data,
  study,
  vars = NULL,
  tables = NULL,
  vars_add = NULL,
  tables_add = NULL,
  release = "latest",
  format = "parquet",
  bypass_ram_check = FALSE,
  ignore_version_mismatch = FALSE,
  categ_to_factor = TRUE,
  add_labels = TRUE,
  value_to_label = FALSE,
  value_to_na = FALSE,
  time_to_hms = FALSE,
  bind_shadow = FALSE,
  finer_add = c("error", "first", "all"),
  ...
)

create_dataset_abcd(...)

create_dataset_hbcd(...)
```

Arguments

dir_data	character. Path to the directory with the data files in .parquet or .tsv format.
study	character. NBDC study (One of "abcd" or "hbcd").
vars	character (vector). Name(s) of variable(s) to be joined. (Default: NULL, i.e., no variables are selected; one of tables or vars has to be provided).
tables	character (vector). Name(s) of table(s) to be joined (Default: NULL, i.e., no tables are selected; one of tables or vars has to be provided).
vars_add	character (vector). Name(s) of additional variable(s) to be left-joined to the variables selected in vars and tables (Default: NULL, i.e., no additional variables are selected)
tables_add	character (vector). Name(s) of additional table(s) to be left-joined to the variables selected in vars and tables (Default: NULL, i.e., no additional tables are selected)
release	character. Release version (Default: "latest")
format	character. Data format (One of "parquet" or "tsv"; default: "parquet").
bypass_ram_check	logical. If TRUE, the function will not abort if the number of variables exceeds 10000 and current available RAM is less than 75% of the estimated RAM usage. This can prevent the long loading time of the data, but failing in the middle due

to insufficient RAM. For large datasets, it is recommended to save 2 times or more of estimated RAM before running this function.

This argument is only used for the ABCD study, as the HBCD data is small enough to be loaded without RAM issues with most personal computers. As HBCD data grows in the future, this may change.

`ignore_version_mismatch`
: logical. Whether to ignore version mismatch between data files and metadata (dd, levels, etc) and proceed with joining anyway (default: `FALSE`).

The function performs a version check by reading a specific file from the `dir_data` directory to determine the latest session in the data files. This session is then compared to the expected latest session for the specified `release`. If there is a mismatch, a warning or error is raised depending on the value of this parameter. If set to `FALSE`, the function will abort on version mismatch. If set to `TRUE`, a warning will be issued, and the function will proceed with joining the data.

`categ_to_factor`
: logical. Whether to convert categorical variables to factors class, see `transf_factor()` (Default: `TRUE`).

`add_labels`
: logical. Whether to adds variable and value labels to the variables, see `transf_label()` (Default: `TRUE`).

`value_to_label`
: logical. Whether to convert the categorical variables' numeric values to labels, see `transf_value_to_label()` (Default: `FALSE`). To run this process, `categ_to_factor` and `add_labels` must be `TRUE`.

`value_to_na`
: logical. Whether to convert categorical missingness/non-response codes to `NA`, see `transf_value_to_na()` (Default: `FALSE`). To run this process, `categ_to_factor` and `add_labels` must be `TRUE`.

`time_to_hms`
: logical. Whether to convert time variables to `hms` class, see `transf_time_to_hms()` (Default: `FALSE`).

`bind_shadow`
: logical. Whether to bind the shadow matrix to the dataset (Default: `FALSE`). See more in details.

`finer_add`
: character. How to treat an additional table that is more finely grained than everything selected in `vars`/`tables` (one of `"error"`, `"first"` or `"all"`; default: `"error"`). Passed to `join_tabulated()`, where the options are described. The same value is used for the data and for the shadow matrix, so that the two keep the same rows.

`...`
: additional arguments passed to downstream functions after the `join_tabulated()` step. See examples for details.

Details

Order:

This high-level function executes the different steps in the following order:

1. Read the data/shadow matrix using `join_tabulated()`.
2. Convert categorical variables to factors using `transf_factor()`.
3. Add labels to the variables and values using `transf_label()`.
4. Convert categorical variables' numeric values to labels using `transf_value_to_label()`.

5. Convert categorical missingness/non-response codes to NA using transf_value_to_na().
6. Convert time variables to hms class using transf_time_to_hms().
7. If bind_shadow and the study is "HBCD", replace the missing values in the shadow due to joining multiple datasets using shadow_replace_binding_missing().
8. Bind the shadow matrix to the data using shadow_bind_data().

Not all steps are executed by default. The above order represents the maximal order of execution.

bind_shadow**:**

If bind_shadow is TRUE, the shadow matrix will be added to the data using shadow_bind_data().

- **HBCD study:** For the HBCD study, this function uses the shadow matrix from the dir_data directory by default (the HBCD Study releases a _shadow.parquet/_shadow.tsv file per table that accompanies the data). Alternatively, one can set naniar_shadow = TRUE as part of the ... arguments to use naniar::as_shadow() to create a shadow matrix from the data.
- **ABCD study:** The ABCD Study does not currently release shadow matrices. If bind_shadow is set to TRUE, the function will create the shadow matrix from the data using naniar::as_shadow(); no extra naniar_shadow = TRUE argument is needed.

Value

A tibble with the analysis-ready dataset.

Examples

```
## Not run:
# most common use case
create_dataset(
  dir_data = "6_0/data",
  study = "abcd",
  vars = c("var1", "var2", "var3")
)

# to handle with tagged missingness
create_dataset(
  dir_data = "1_0/data",
  study = "hbcd",
  vars = c("var1", "var2", "var3"),
  value_to_na = TRUE
)

# to bind shadow matrices to the data
create_dataset(
  dir_data = "1_0/data/",
  study = "hbcd",
  vars = c("var1", "var2", "var3"),
  bind_shadow = TRUE
)

# to use the additional arguments
# for example in `value_to_na` option, the underlying function
# `transf_value_to_na()` has 2 more arguments,
```

```
# which can be passed to the `create_dataset()` function
create_dataset(
  dir_data = "6_0/data",
  study = "abcd",
  vars = c("var1", "var2", "var3"),
  value_to_na = TRUE,
  missing_codes = c("999", "888", "777", "666", "555", "444", "333", "222"),
  ignore_col_pattern = "__dk$|__dk__1$"
)

# use study specific functions
create_dataset_abcd(
  dir_data = "6_0/data",
  vars = c("var1", "var2", "var3")
)

## End(Not run)
```

`filter_empty_cols` *Filter empty columns*

Description

This function filters out columns that are empty.

Usage

```
filter_empty_cols(
  data,
  id_cols = union(get_id_cols_abcd(), get_id_cols_hbcd())
)
```

Arguments

`data`	tibble. The data to be filtered.
`id_cols`	character (vector). The names of the ID columns to be excluded from the filtering (Default: identifier columns used in ABCD and HBCD).

Value

A tibble with the filtered data.

Examples

```
data <- tibble::tibble(
  participant_id = c("sub-001", "sub-002", "sub-003"),
  session_id = c("ses-001", "ses-001", "ses-002"),
  var1 = c(NA, NA, NA),
  var2 = c(NA, NA, 2),
```

```
  var3 = c(NA, NA, 3)
)
filter_empty_cols(data)
```

`filter_empty_rows` *Filter empty rows*

Description

This function filters out rows that are empty

Usage

```
filter_empty_rows(
  data,
  id_cols = union(get_id_cols_abcd(), get_id_cols_hbcd())
)
```

Arguments

`data`	tibble. The data to be filtered.
`id_cols`	character (vector). The names of the ID columns to be excluded from the filtering (Default: identifier columns used in ABCD and HBCD).

Value

A tibble with the filtered data.

Examples

```
data <- tibble::tibble(
  participant_id = c("sub-001", "sub-002", "sub-003"),
  session_id = c("ses-001", "ses-001", "ses-002"),
  var1 = c(NA, NA, 1),
  var2 = c(NA, NA, 2),
  var3 = c(NA, NA, 3)
)
filter_empty_rows(data)
```

`filter_events_abcd` *Filter ABCD events*

Description

Given a (set of) condition(s), filters the events included in an ABCD dataset. Conditions can be specified as a vector of strings, where each string can be one of the following conditions:

- `"core"`: events for the ABCD core study
- `"annual"`: annual events for the ABCD core study
- `"mid_year"`: mid-year events for the ABCD core study
- `"substudy"`: events for ABCD substudies
- `"covid"`: events for the COVID substudy
- `"sdev"`: events for the Social Development substudy
- `"even"`: even-numbered events
- `"odd"`: odd-numbered events
- numerical expressions like `>2` or `<=5` to filter events by number
- any other string to be used as filter for the `session_id` column

The conditions can be combined with logical `"and"` or `"or"`.

Usage

```
filter_events_abcd(data, conditions, connect = "and")
```

Arguments

`data`	tibble. The data to be filtered.
`conditions`	character (vector). The events to keep.
`connect`	character. Whether to connect the conditions with `"and"` (an event is retained if *all* conditions are met; the default) or `"or"` (an event is retained if *any* condition is met).

Value

A tibble with the filtered data.

Examples

```
data <- tibble::tribble(
  ~session_id,     ~study,     ~type,
  "ses-00S",       "core",     "screener",
  "ses-00M",       "core",     "mid-year",
  "ses-00A",       "core",     "even",
  "ses-01M",       "core",     "mid-year",
```

```
  "ses-01A",       "core",     "odd",
  "ses-02M",       "core",     "mid-year",
  "ses-02A",       "core",     "even",
  "ses-03M",       "core",     "mid-year",
  "ses-03A",       "core",     "odd",
  "ses-04M",       "core",     "mid-year",
  "ses-04A",       "core",     "even",
  "ses-05M",       "core",     "mid-year",
  "ses-05A",       "core",     "odd",
  "ses-06M",       "core",     "mid-year",
  "ses-06A",       "core",     "even",
  "ses-C01",       "substudy", "covid",
  "ses-C02",       "substudy", "covid",
  "ses-C03",       "substudy", "covid",
  "ses-C04",       "substudy", "covid",
  "ses-C05",       "substudy", "covid",
  "ses-C06",       "substudy", "covid",
  "ses-C07",       "substudy", "covid",
  "ses-S01",       "substudy", "sdev",
  "ses-S02",       "substudy", "sdev",
  "ses-S03",       "substudy", "sdev",
  "ses-S04",       "substudy", "sdev",
  "ses-S05",       "substudy", "sdev"
)

# ABCD core study events
filter_events_abcd(data, c("core"))

# COVID substudy events
filter_events_abcd(data, c("covid"))

# imaging events
filter_events_abcd(data, c("annual", "even"))

# mid-years before year 5
filter_events_abcd(data, c("mid_year", "<5"))

# COVID or Social Development substudy events
filter_events_abcd(data, c("covid", "sdev"), connect = "or")
```

`filter_id_events` *Filter ID/events*

Description

Given a vector of ID/events (concatenated like "`{participant_id}_{session_id}`"), or a dataframe with `participant_id` and `session_id` columns, this function filters the data to keep or alternatively remove the rows for the given *ID/events*.

Usage

```
filter_id_events(data, id_events, revert = FALSE)
```

Arguments

`data`	tibble. The data to be filtered.
`id_events`	character (vector) or dataframe. *(Vector of) ID/event(s) or a dataframe with* `participant_id` *and* `session_id` *columns*.
`revert`	logical. Whether to revert the filter, i.e., to keep only rows NOT matching the `id_events` (Default: `FALSE`, i.e., keep only the rows matching the `id_events`).

Value

A tibble with the filtered data.

Examples

```
data <- tibble::tribble(
  ~participant_id, ~session_id,
  "sub-001",       "ses-001",
  "sub-001",       "ses-002",
  "sub-002",       "ses-001",
  "sub-002",       "ses-002",
  "sub-003",       "ses-001",
  "sub-003",       "ses-002"
)

# filter using a vector of ID/events
filter_id_events(
  data,
  id_events = c("sub-001_ses-001", "sub-003_ses-002")
)

# filter using a dataframe with participant_id and session_id
data_filter <- tibble::tibble(
  participant_id = c("sub-001", "sub-003"),
  session_id = c("ses-001", "ses-002")
)
filter_id_events(
  data,
  id_events = data_filter
)

# revert filter
filter_id_events(
  data,
  id_events = c("sub-001_ses-001", "sub-003_ses-002"),
  revert = TRUE
)
```

get_dd *Get data dictionary*

Description

Retrieves data dictionary for a given study and release version. Allows for filtering by variables and tables. Wrapper around `get_metadata()`.

In addition to the main `get_dd()` function, there are two study-specific variations:

- `get_dd_abcd()`: for the ABCD study.
- `get_dd_hbcd()`: for the HBCD study.

They have the same arguments as the `get_dd()` function, except that the `study` argument is set to the respective study by default, and should not be set by the user.

Usage

```
get_dd(study, release = "latest", vars = NULL, tables = NULL)

get_dd_abcd(...)

get_dd_hbcd(...)
```

Arguments

`study`	character. The study name. One of "abcd" or "hbcd".
`release`	character. Release version (Default: `"latest"`).
`vars`	character (vector). Vector with the names of variables to be included.
`tables`	character (vector). Vector with the names of tables to be included.
`...`	Additional arguments passed to the underlying `get_dd()` function.

Value

Data frame with the data dictionary.

Examples

```
get_dd("abcd")

get_dd("hbcd", release = "1.0")

get_dd("abcd", vars = c("ab_g_dyn__visit_dtt", "ab_g_dyn__visit_age"))

get_dd("abcd", tables = "ab_g_dyn")

get_dd_abcd()
```

```
get_dd_hbcd(release = "1.0")
```

`get_id_cols` *Get identifier columns*

Description

Retrieves the identifier columns for a given study and release version.

In addition to the main `get_id_cols()` function, there are two study-specific variations:

- `get_id_cols_abcd()`: for the ABCD study.
- `get_id_cols_hbcd()`: for the HBCD study.

They have the same arguments as the `get_id_cols()` function, except that the `study` argument is set to the respective study by default, and should not be set by the user.

Usage

```
get_id_cols(study, release = "latest")

get_id_cols_abcd(...)

get_id_cols_hbcd(...)
```

Arguments

`study`	character. The study name. One of "abcd" or "hbcd".
`release`	character. Release version (Default: `"latest"`).
`...`	Additional arguments passed to the underlying `get_id_cols()` function.

Value

character vector with the identifier columns.

Examples

```
get_id_cols("abcd")

get_id_cols("hbcd")

get_id_cols_abcd(release = "6.0")

get_id_cols_hbcd(release = "1.0")
```

`get_levels` *Get levels table*

Description

Retrieves levels table for a given study and release version. Allows for filtering by variables and tables. Wrapper around `get_metadata()`.

In addition to the main `get_levels()` function, there are two study-specific variations:

- `get_levels_abcd()`: for the ABCD study.
- `get_levels_hbcd()`: for the HBCD study.

They have the same arguments as the `get_levels()` function, except that the `study` argument is set to the respective study by default, and should not be set by the user.

Usage

```
get_levels(study, release = "latest", vars = NULL, tables = NULL)

get_levels_abcd(...)

get_levels_hbcd(...)
```

Arguments

`study`	character. The study name. One of "abcd" or "hbcd".
`release`	character. Release version (Default: `"latest"`).
`vars`	character (vector). Vector with the names of variables to be included.
`tables`	character (vector). Vector with the names of tables to be included.
`...`	Additional arguments passed to the underlying `get_levels()` function.

Value

Data frame with the levels table.

Examples

```
get_levels("abcd")

get_levels("hbcd", release = "1.0")

get_levels("abcd", vars = c("ab_g_dyn__visit_type"))

get_levels("abcd", tables = "ab_g_dyn")

get_levels_abcd(release = "6.0")

get_levels_hbcd()
```

`get_metadata` *Get metadata*

Description

Retrieves metadata (data dictionary, levels table, event map) for a given study and release version. Allows for filtering by variables and tables.

Usage

```
get_metadata(
  study,
  release = "latest",
  vars = NULL,
  tables = NULL,
  type = "dd"
)
```

Arguments

`study`	character. The study name. One of "abcd" or "hbcd".
`release`	character. Release version (Default: `"latest"`).
`vars`	character (vector). Vector with the names of variables to be included.
`tables`	character (vector). Vector with the names of tables to be included.
`type`	character. Type of metadata to retrieve. One of `"dd"`, `"levels"`, `"sessions"` (Default: `"dd"`).

Value

Data frame with the metadata.

Examples

```
get_metadata("abcd", type = "levels")

get_metadata("hbcd", release = "1.0")

get_metadata("abcd", vars = c("ab_g_dyn__visit_dtt", "ab_g_dyn__visit_age"))

get_metadata("abcd", tables = "ab_g_dyn")

get_metadata("abcd", tables = "ab_g_dyn")

get_metadata("abcd", type = "sessions")
```

get_releases *Get release version number(s)*

Description

These functions retrieve all release version number(s) for a given study, or the latest release version number.

Usage

```
get_releases(study)

get_releases_abcd()

get_releases_hbcd()

get_latest_release(study)

get_latest_release_abcd()

get_latest_release_hbcd()
```

Arguments

study	character. The study name. One of "abcd" or "hbcd".

Value

character. The latest release version number(s) of the specified study.

Examples

```
get_releases("abcd")
get_releases("hbcd")
get_latest_release("abcd")
get_latest_release("hbcd")
```

get_sessions *Get sessions table*

Description

Retrieves the sessions table for a given study and release version. Wrapper around `get_metadata()`.

In addition to the main `get_sessions()` function, there are two study-specific variations:

- `get_sessions_abcd()`: for the ABCD study.
- `get_sessions_hbcd()`: for the HBCD study.

They have the same arguments as the `get_sessions()` function, except that the `study` argument is set to the respective study by default, and should not be set by the user.

Usage

```
get_sessions(study, release = "latest")

get_sessions_abcd(...)

get_sessions_hbcd(...)
```

Arguments

`study`	character. The study name. One of "abcd" or "hbcd".
`release`	character. Release version (Default: `"latest"`).
`...`	Additional arguments passed to the underlying `get_sessions()` function.

Value

Data frame with the sessions table.

Examples

```
get_sessions("abcd")

get_sessions("hbcd")

get_sessions_abcd(release = "6.0")

get_sessions_hbcd(release = "1.0")
```

`join_tabulated` *Join tabulated data*

Description

Joins selected variables and/or whole tables from the tabulated data/shadow files into a single data frame. Expects the data files to be stored in one directory in `.parquet` or `.tsv` format, with one file per table following the naming convention of the respective NBDC dataset (from the ABCD or HBCD studies). Typically, this will be the `rawdata/phenotype/` directory within a BIDS dataset downloaded from the NBDC Data Hub.

Variables specified in `vars` and `tables` will be full-joined together, i.e., all rows will be kept, even if they do not have a value for all columns. Variables specified in `vars_add` will be left-joined to the variables selected in `vars` and `tables`, i.e., only the values for already existing rows will be added and no new rows will be created. This is useful for adding variables to the dataset that are important for a given analysis but are not the main variables of interest (e.g., design/nesting or demographic information). By left-joining these variables, one avoids creating new rows that contain only missing values for the main variables of interest selected using `vars` and `tables`. If the same variables are specified in `vars`/`tables` and `vars_add`/`tables_add`, the variables in `vars_add`/`tables_add` will be ignored.

Tables are joined on their identifier columns, which differ by table: participant-level tables are identified by `participant_id` alone, session-level tables by `participant_id` and `session_id`, and run-level tables also by `run_id`. A variable from a coarser table is repeated on every finer row it applies to, so a participant-level variable such as birth data appears on every session and every run of that participant. The result does not depend on the order in which variables or tables are listed, and contains one row per distinct combination of identifier columns.

If an additional table is more finely grained than everything selected in `vars`/`tables` (for example a session-level `vars_add` on top of a participant-level selection), its values cannot all be attached without adding rows, which conflicts with the rule that `vars_add`/`tables_add` do not create rows. Neither way out is right for every analysis, so `finer_add` decides; see that argument for the three options. By default this is an error, since silently dropping values and silently adding rows are both surprising.

A third option is to not use `vars_add`/`tables_add` for those variables at all: selecting them via `vars`/`tables` keeps every value and makes the finer grain part of the row set, so no row is ambiguous.

In addition to the main `join_tabulated()` function, there are two study-specific variations:

- `join_tabulated_abcd()`: for the ABCD study.
- `join_tabulated_hbcd()`: for the HBCD study.

They have the same arguments as the `join_tabulated()` function, except that the `study` argument is set to the respective study by default, and should not be set by the user.

Usage

```
join_tabulated(
  dir_data,
```

```
  study,
  vars = NULL,
  tables = NULL,
  vars_add = NULL,
  tables_add = NULL,
  release = "latest",
  format = "parquet",
  shadow = FALSE,
  remove_empty_rows = TRUE,
  bypass_ram_check = FALSE,
  ignore_version_mismatch = FALSE,
  finer_add = c("error", "first", "all")
)

join_tabulated_abcd(...)

join_tabulated_hbcd(...)
```

Arguments

dir_data	character. Path to the directory with the data files in `.parquet` or `.tsv` format.
study	character. NBDC study (One of `"abcd"` or `"hbcd"`).
vars	character (vector). Name(s) of variable(s) to be joined. (Default: `NULL`, i.e., no variables are selected; one of `tables` or `vars` has to be provided).
tables	character (vector). Name(s) of table(s) to be joined (Default: `NULL`, i.e., no tables are selected; one of `tables` or `vars` has to be provided).
vars_add	character (vector). Name(s) of additional variable(s) to be left-joined to the variables selected in `vars` and `tables` (Default: `NULL`, i.e., no additional variables are selected)
tables_add	character (vector). Name(s) of additional table(s) to be left-joined to the variables selected in `vars` and `tables` (Default: `NULL`, i.e., no additional tables are selected)
release	character. Release version (Default: `"latest"`)
format	character. Data format (One of `"parquet"` or `"tsv"`; default: `"parquet"`).
shadow	logical. Whether to join the shadow matrix instead of the data table (default: `FALSE`).
remove_empty_rows	logical. Whether to filter out rows that have all values missing in the joined variables, except for the ID columns (default: `TRUE`).
bypass_ram_check	logical. If `TRUE`, the function will not abort if the number of variables exceeds 10000 and current available RAM is less than 75% of the estimated RAM usage. This can prevent the long loading time of the data, but failing in the middle due to insufficient RAM. For large datasets, it is recommended to save 2 times or more of estimated RAM before running this function.

This argument is only used for the ABCD study, as the HBCD data is small enough to be loaded without RAM issues with most personal computers. As HBCD data grows in the future, this may change.

`ignore_version_mismatch`
: logical. Whether to ignore version mismatch between data files and metadata (dd, levels, etc) and proceed with joining anyway (default: `FALSE`).

  The function performs a version check by reading a specific file from the `dir_data` directory to determine the latest session in the data files. This session is then compared to the expected latest session for the specified `release`. If there is a mismatch, a warning or error is raised depending on the value of this parameter. If set to `FALSE`, the function will abort on version mismatch. If set to `TRUE`, a warning will be issued, and the function will proceed with joining the data.

`finer_add`
: character. How to treat an additional table that is more finely grained than everything selected in `vars`/`tables` (one of `"error"`, `"first"` or `"all"`; default: `"error"`).

  - `"error"`: abort and ask for an explicit choice. This is the default because the alternatives either drop values or add rows, and which is appropriate depends on the analysis.
  - `"first"`: keep one value per row of the main selection and drop the rest, with a warning reporting how many were dropped. The value kept is the first by the additional table's own identifier columns, i.e. the earliest available session or the first run – not necessarily the baseline, since a participant need not have one. The whole row is taken, so all variables from that table come from the same session/run, and a missing value is not skipped over in favor of a later one. The finer identifier columns are not added to the result.
  - `"all"`: keep every value, which adds rows to the main selection. This was the behavior before version 1.1.2.

`...`
: Additional arguments passed to the underlying function `join_tabulated()`

  Note: Turning this parameter to `FALSE` is useful for shadow matrices processing. Some shadow-related functions expect the shadow matrix to have the same dimensions as the original data to proceed correctly. See `shadow_bind_data()` and `shadow_replace_binding_missing()`

Value

A tibble of data or shadow matrix with the joined variables.

Examples

```
## Not run:
join_tabulated(
  dir_data = "path/to/data/",
  vars     = c("var_1", "var_2", "var_3"),
  tables   = c("table_1", "table_2"),
  study    = "abcd",
  release  = "6.0"
)

## End(Not run)
```

`read_dsv_formatted` *Read delimiter (tab/comma) separated values file correctly formatted*

Description

Reads in a `.tsv` or `.csv` file with correctly formatted column types. Uses `readr::read_tsv()`/`readr::read_csv()` internally and specifies the column types explicitly using the `col_types` argument utilizing information from the data dictionary. Returns only the identifier columns and the columns specified in the data dictionary, i.e., all columns in the file that are not specified in the data dictionary are ignored.

Usage

```
read_dsv_formatted(file, dd, action = "warn")
```

Arguments

`file`	character. Path to the `.tsv` or `.csv` file.
`dd`	tibble. Data dictionary specifying the column types. Only columns specified in the data dictionary are read.
`action`	character. What to do if there are columns in the file that are not specified in the data dictionary (One of `"warn"`, `"error"`, or `"ignore"`; default: `"warn"`).

Details

WHY THIS IS IMPORTANT: `readr::read_tsv()`/`readr::read_csv()` (like other commands to load text files in R or other programming languages) by default infers the column types from the data. This doesn't always work perfectly. For example, it may interpret a column with only integers as a double, or a column with only dates as a character. Sometimes a column may even be read in completely empty because, by default, `readr::read_tsv()`/`readr::read_csv()` only considers the first 1000 rows when inferring the data type and interprets the column as an empty logical vector if those rows are all empty. The NBDC datasets store categorical data as integers formatted as character. By default, `readr::read_tsv()`/`readr::read_csv()` may interpret them as numeric. By specifying the column types explicitly based on what is defined in the data dictionary, we can avoid these issues.

GENERAL RECOMMENDATION: Other file formats like `.parquet` correctly store the column types and don't need to be handled explicitly. They also offer other advantages like faster reading speed and smaller file sizes. As such, these formats should generally be preferred over `.tsv`/`.csv` files. However, if you have to work with `.tsv`/`.csv` files, this function can help you avoid common pitfalls.

Value

A tibble with the data/shadow matrix read from the `.tsv` or `.csv` file.

Examples

```
## Not run:
dd <- NBDCtools::get_dd("abcd", "6.0")
read_tsv_formatted("path/to/file.tsv", dd)

## End(Not run)
```

`shadow_bind_data` *Bind the shadow matrix to the data*

Description

This function binds the shadow matrix to the data.

Usage

```
shadow_bind_data(
  data,
  shadow = NULL,
  naniar_shadow = FALSE,
  id_cols = union(get_id_cols_abcd(), get_id_cols_hbcd()),
  suffix = "_shadow"
)
```

Arguments

`data`	tibble. The data.
`shadow`	tibble. The shadow matrix. If `naniar_shadow` is `TRUE`, this argument is ignored.
`naniar_shadow`	logical. Whether to use `naniar::as_shadow()` to create the shadow matrix from data instead of providing it as an argument.
`id_cols`	character. The columns to join by (the identifier column(s)) in the data and shadow matrices (Default: identifier columns used in ABCD and HBCD). In `naniar_shadow = TRUE`, these columns are not included in the shadow matrix.
`suffix`	character. The suffix to add to the shadow columns. Default is `"_shadow"`. For example, if the column name is `"var1"` and the suffix is `"_shadow"`, the resulted column name will be `"var1_shadow"`. If `naniar_shadow = TRUE`, the suffix is `_NA`, as this suffix will have the most compatibility with other functions in the `naniar` package.

Details

Data requirements:

If `naniar_shadow = FASLE` and `shadow` is provided, the two dataframes must have the same columns, order of the columns does not matter, but ID columns must be the same in both dataframes. If there are extra rows in the shadow matrix, they will be ignored.

ABCD and HBCD data:

NBDC releases HBCD data with shadow matrices, which can be used for the `shadow` argument. To work with ABCD data, the option for now is to use `naniar_shadow = TRUE`, which will create a shadow matrix from the data using `naniar::as_shadow()`.

Value

a dataframe of the data matrix with shadow columns. It will be 2x the size of the original data matrix.

Examples

```
shadow <- tibble::tibble(
  participant_id = c("1", "2", "3"),
  session_id = c("1", "2", "3"),
  var1 = c("Unknown", NA, NA),
  var2 = c("Wish not to answer", NA, NA)
)
data <- tibble::tibble(
  participant_id = c("1", "2", "3"),
  session_id = c("1", "2", "3"),
  var1 = c(NA, NA, 1),
  var2 = c(NA, 2, NA)
)
shadow_bind_data(data, shadow)
if (requireNamespace("naniar", quietly = TRUE)) {
  shadow_bind_data(data, naniar_shadow = TRUE)
}
```

---

shadow_replace_binding_missing

Fix binding resulted missingness in shadow matrices

Description

This function replaces the missing values in the shadow matrices. This is done by checking if the values in shadow matrices are both NA. If they are, the value in the shadow matrix is replaced with `Missing due to joining`.

Usage

```
shadow_replace_binding_missing(
  data,
  shadow,
  id_cols = union(get_id_cols_abcd(), get_id_cols_hbcd()),
  replacement = "Missing due to joining"
)
```

Arguments

data	tibble. The data.
shadow	tibble. The shadow matrix.
id_cols	character (vector). The possible unique identifier columns. The data does not need to have all of these columns, but if they are present, they will be used to identify unique rows (Default: identifier columns used in ABCD and HBCD). For example, the ABCD data usually has only participant_id and session_id, so if run_id is provided, it will be ignored.
replacement	character. The value to replace the missing values with.

Details

Data and shadow requirements: The two dataframes must have the same columns and the same number of rows. They must have the same column names, but the order of the columns does not matter. It is recommended to use the same column order and the same row order (by ID columns) in both dataframes, which saves some processing time.

Value

A tibble of the shadow matrix with missing values replaced.

Examples

```
shadow <- tibble::tibble(
  participant_id = c("1", "2", "3"),
  session_id = c("1", "2", "3"),
  var1 = c("Unknown", NA, NA),
  var2 = c("Wish not to answer", NA, NA)
)
data <- tibble::tibble(
  participant_id = c("1", "2", "3"),
  session_id = c("1", "2", "3"),
  var1 = c(NA, NA, 1),
  var2 = c(NA, 2, NA)
)
shadow_replace_binding_missing(data, shadow)
```

---

transf_factor *Convert categorical columns to factor*

Description

Based on the specifications in the data dictionary, transforms all categorical columns to factor.

Usage

```
transf_factor(data, study, release = "latest")
```

Arguments

data	tibble. The data to be transformed. Columns are expected to be in the data dictionary. If not, they will be skipped.
study	character. NBDC study (One of "abcd" or "hbcd".
release	character. Release version (Default: "latest").

Value

A tibble with the transformed data.

Examples

```
## Not run:
transf_factor(data, study = "abcd")

## End(Not run)
```

---

transf_label *Add variable/value labels*

Description

This function can add variable labels and value labels to the data. The variable labels are descriptive information about the column, and the value labels are the levels of the factor variables.

Usage

```
transf_label(
  data,
  study,
  release = "latest",
  add_var_label = TRUE,
  add_value_label = TRUE,
  id_cols_labels = c(participant_id = "Participant identifier", session_id =
    "Event identifier", run_id = "Run identifier")
)
```

Arguments

data	tibble. The data to be transformed.
study	character. NBDC study (One of "abcd" or "hbcd".
release	character. Release version (Default: "latest").
add_var_label	logical. Whether to add variable labels (Default: TRUE).
add_value_label	logical. Whether to add value labels (Default: TRUE).
id_cols_labels	named character vector. A named vector of labels for the identifier columns, with the names being the column names and the values being the labels.

Details

Two types of labels:

At least one of `add_var_label` or `add_value_label` must be set to `TRUE`. If both are `FALSE`, an error will be raised.

Text columns:

The `transf_factor()` function has a `convert_text` argument, which will convert text columns to unordered factors. When one uses a type transformed data to add labels, the text-factor columns will not have labels at variable level.

Value

A tibble with the labelled data.

See Also

`transf_factor()` for transforming categorical columns to factors.

Examples

```
## Not run:
transf_label(data)

## End(Not run)
```

---

transf_time_to_hms *Convert time columns to* hms *format*

Description

This function converts time columns to `hms` format.

Usage

```
transf_time_to_hms(data, study, release = "latest")
```

Arguments

`data`	tibble. The data to be converted.
`study`	character. NBDC study (One of `"abcd"` or `"hbcd"`).
`release`	character. Release version (Default: `"latest"`).

Details

The input data with time columns are expected to have character format of `"HH:MM:SS"`. If it is not in this format, the function will return NA for that row.

Value

A tibble with time columns converted to hms format.

Examples

```
## Not run:
transf_time_to_hms(data)

## End(Not run)
```

`transf_value_to_label` *Convert values to labels for categorical variables*

Description

Converts the values of categorical/factor columns (e.g., "1", "2") to their labels (e.g., "Male", "Female"). The value labels will be set to the values.

Usage

```
transf_value_to_label(data, transf_sess_id = FALSE)
```

Arguments

data	tibble. The labelled dataset
transf_sess_id	logical. Whether to transform the session_id column

Details

Input requirements:
The data must be type transformed and labelled. See `transf_factor()` and `transf_label()` for details.

```
data <- data |>
  transf_factor() |>
  transf_label()
```

Value

A tibble with factor columns transformed to labels.

Examples

```
## Not run:
transf_value_to_label(data)
transf_value_to_label(data, value_to_na = TRUE)

## End(Not run)
```

`transf_value_to_na` *Convert categorical missingness/non-response codes to* `NA`

Description

This function converts the missing codes in the dataset to NA in all factor columns. Example of missing codes are `999`, `888`, `777`, etc.

Usage

```
transf_value_to_na(
  data,
  missing_codes = c("999", "888", "777", "666", "555", "444", "333", "222"),
  ignore_col_pattern = "__dk$|__dk__l$",
  id_cols = union(get_id_cols_abcd(), get_id_cols_hbcd())
)
```

Arguments

`data`	tibble. The labelled dataset and type converted data.
`missing_codes`	character vector. The missing codes to be converted to NA
`ignore_col_pattern`	character. A regex pattern to ignore columns that should not be converted to NA.
`id_cols`	character vector. The names of the ID columns to be excluded from the conversion (Default: identifier columns used in ABCD and HBCD).

Details

Use case:

This function works the best with `ABCD` data where the missing codes are strictly defined. For `HBCD` data, the missing codes are still under discussion. The function may work, but for some undecided future missing codes, the function may not work as expected.

In case of `HBCD` data or other aribitrary missing codes that one wishes to convert to NA, it is recommended to use the `sjmisc::set_na_if()` function instead.

Input requirements:

The data must be type transformed and labelled. See `transf_factor()` and `transf_label()` for details.

```
data <- data |>
  transf_factor() |>
  transf_label()
```

Value

A tibble of the dataset with missing codes converted to NA

Examples

```
## Not run:
data <- data |>
  transf_factor() |>
  transf_label()

transf_value_to_na(data)

## End(Not run)
```

Index

add_custom_metadata, 2

convert_names_data, 3
convert_names_data(), *5*
convert_names_file, 5
convert_names_file(), *3*
create_bids_sidecar_data, 7
create_bids_sidecar_data(), *7*, *9*
create_bids_sidecar_metadata, 8
create_bids_sidecar_metadata(), *7–9*
create_dataset, 9
create_dataset_abcd *(*create_dataset*)*, 9
create_dataset_hbcd *(*create_dataset*)*, 9

filter_empty_cols, 13
filter_empty_rows, 14
filter_events_abcd, 15
filter_id_events, 16

get_dd, 18
get_dd(), *4–6*, *8*, *18*
get_dd_abcd *(*get_dd*)*, 18
get_dd_hbcd *(*get_dd*)*, 18
get_id_cols, 19
get_id_cols(), *19*
get_id_cols_abcd *(*get_id_cols*)*, 19
get_id_cols_hbcd *(*get_id_cols*)*, 19
get_latest_release *(*get_releases*)*, 22
get_latest_release_abcd *(*get_releases*)*, 22
get_latest_release_hbcd *(*get_releases*)*, 22
get_levels, 20
get_levels(), *8*, *20*
get_levels_abcd *(*get_levels*)*, 20
get_levels_hbcd *(*get_levels*)*, 20
get_metadata, 21
get_metadata(), *18*, *20*, *23*
get_releases, 22
get_releases_abcd *(*get_releases*)*, 22
get_releases_hbcd *(*get_releases*)*, 22
get_sessions, 22
get_sessions(), *23*
get_sessions_abcd *(*get_sessions*)*, 22
get_sessions_hbcd *(*get_sessions*)*, 22

join_tabulated, 24
join_tabulated(), *9*, *11*, *26*
join_tabulated_abcd *(*join_tabulated*)*, 24
join_tabulated_hbcd *(*join_tabulated*)*, 24

read_dsv_formatted, 27
readr::read_csv(), *27*
readr::read_tsv(), *27*

shadow_bind_data, 28
shadow_bind_data(), *12*, *26*
shadow_replace_binding_missing, 29
shadow_replace_binding_missing(), *12*, *26*
sjmisc::set_na_if(), *34*

tidyr::separate_rows(), *4*, *5*
transf_factor, 30
transf_factor(), *11*, *32–34*
transf_label, 31
transf_label(), *7*, *11*, *33*, *34*
transf_time_to_hms, 32
transf_time_to_hms(), *11*, *12*
transf_value_to_label, 33
transf_value_to_label(), *7*, *11*
transf_value_to_na, 34
transf_value_to_na(), *11*, *12*