Package 'EDCimport'

November 14, 2024

Version 0.5.2

Title Import Data from EDC Software

Description A convenient toolbox to import data exported from Electronic Data Capture (EDC) software 'TrialMaster'.

License GPL-3

URL https://github.com/DanChaltiel/EDCimport, https://danchaltiel.github.io/EDCimport/

BugReports https://github.com/DanChaltiel/EDCimport/issues

Depends R (>= 3.1.0)

Imports cli, dplyr, forcats, fs, glue, ggplot2, haven, purrr, readr, rlang, scales, stringr, tibble, tidyr, tidyselect, utils, lifecycle

Suggests callr, crosstable, gtools, htmlwidgets, janitor, knitr, openxlsx, patchwork, plotly, quarto, rmarkdown, rstudioapi, testthat (>= 3.1.8), usethis, vdiffr, withr

Encoding UTF-8

RoxygenNote 7.3.2

Config/testthat/edition 3

Config/testthat/parallel true

Config/testthat/start-first local, trialmaster, utils

Collate '7zip.R' 'EDCimport-package.R' 'assertions.R' 'burgled.R' 'crf_status.R' 'data.R' 'helpers.R' 'utils.R' 'lookup.R' 'options.R' 'population_plot.R' 'read_all_csv.R' 'read_all_sas.R' 'read_all_xpt.R' 'read_trialmaster.R' 'sanity_checks.R' 'sas_format.R' 'save_to_excel.R' 'search_for_newer_data.R' 'split_mixed.R' 'swimmerplot.R' 'utils_read.R'

NeedsCompilation no

Author Dan Chaltiel [aut, cre] (<https://orcid.org/0000-0003-3488-779X>)

Maintainer Dan Chaltiel <dan.chaltiel@gmail.com>

Repository CRAN

Date/Publication 2024-11-14 16:10:11 UTC

Contents

`assert_no_duplicate` *Assert that a dataframe has one row per patient*

Description

Check that there is no duplicate on the column holding patient ID in a pipeable style.
Mostly useful after joining two datasets.

Usage

```
assert_no_duplicate(df, by = NULL, id_col = get_subjid_cols())
```

Arguments

`df`	a dataframe
`by`	*(optional)* grouping columns
`id_col`	the name of the columns holding patient ID

Value

the `df` dataset, unchanged

Examples

```
## Not run:
#without duplicate => no error, continue the pipeline
tibble(subjid=c(1:10)) %>% assert_no_duplicate() %>% nrow()

#with duplicate => throws an error
tibble(subjid=c(1:10, 1:2)) %>% assert_no_duplicate() %>% nrow()

#By groups
df = tibble(subjid=rep(1:10, 4), visit=rep(c("V1", "V2"), 2, each=10),
            group=rep(c("A", "B"), each=20))
df %>% assert_no_duplicate() #error
df %>% assert_no_duplicate(by=c(visit, group)) #no error

## End(Not run)
```

build_lookup *Generate a lookup table*

Description

Generate a lookup table

Usage

```
build_lookup(data_list)
```

Arguments

data_list	a list containing at least 1 dataframe

Value

a dataframe summarizing column names and labels

See Also

extend_lookup(), edc_lookup()

Examples

```
x = edc_example()
x$.lookup=NULL
lk = build_lookup(x)
lk
lk %>% tidyr::unnest(c(names, labels))
```

crf_status_plot *Show the current CRF status distribution*

Description

Generate a barplot showing the distribution of CRF status (Complete, Incomplete, ...) for each dataset of the database.

Usage

```
crf_status_plot(
  crfstat_col = "CRFSTAT",
  ...,
  details = FALSE,
  pal = edc_pal_crf(),
  crfstat_lvls = names(pal),
  x_label = "{dataset}",
  treat_as_worst = NULL
)

edc_pal_crf()
```

Arguments

crfstat_col	the column name of the CRF status
...	unused
details	whether to show all the CRF status levels. When FALSE (default), recode the status into "Complete", "Incomplete", or "No Data".
pal	the palette, defaulting to the helper EDCimport:::edc_pal_crf()
crfstat_lvls	the CRF status levels, from "best" to "worst". The plot is ordered by the "worst" level.
x_label	a glue pattern determining the tick label in the x axis. Available variables are c("nrow", "ncol", "n_id", "rows_per_id", "crfname"), taken from edc_lookup().
treat_as_worst	a regex for levels that should be treated as worst in the ordering

Value

a ggplot

Source

ggsci:::ggsci_db$lancet[["lanonc"]] %>% dput()

Examples

```
## Not run:
#import a TM database and use load_list(), then:
crf_status_plot() + ggtitle(date_extraction)
crf_status_plot(pal=rev(edc_pal_crf()))
crf_status_plot(details=TRUE, treat_as_worst="No Data")
crf_status_plot(x_label="{crfname} (N={n_id}, n={nrow})")

p = crf_status_plot(details=TRUE)
p$data$crfstat %>% unique()
#> [1] "Incomplete"        "No Data Locked"    "No Data"           "Signed"
#> [5] "Partial Monitored" "Monitored"         "Complete Locked"   "Complete"

## End(Not run)
```

data_example *Example databases*

Description

List of tables used in EDCimport examples:

- `edc_example()` can be used as the result of `read_trialmaster()`
- `edc_example_plot()` can be used to test `edc_swimmerplot()`
- `edc_example_mixed()` can be used to test `split_mixed_datasets()`

Usage

```
edc_example_mixed(N = 100, seed = 42)

edc_example(N = 50, seed = 42)

edc_example_plot(N = 50, seed = 42)

edc_example_ae(N = 50, seed = 42)
```

Arguments

N	the number of patients
seed	the random seed

Value

a list of tables

edc_data_warn *Standardized warning system*

Description

When checking your data, filter your dataset to get only problematic rows.
Then, use either:

- `edc_data_warn()` to generate a standardized warning that can be forwarded to the data-manager
- `edc_data_warn()` to abort the script if the problem is too serious

Database issues should be traced in a separate file, each with an identifying row number, and the file should be shared with the data-manager.
Use `edc_data_warnings()` to generate the table for such a file.

Usage

```
edc_data_warn(
  df,
  message,
  ...,
  issue_n = "xx",
  max_subjid = 5,
  csv_path = FALSE,
  col_subjid = get_subjid_cols()
)

edc_data_stop(df, message, ..., issue_n, max_subjid, csv_path, col_subjid)

edc_data_warnings()
```

Arguments

df	the filtered dataframe
message	the message. Can use cli formats. df can be accessed using the .data special keyword (see example)
...	unused
issue_n	identifying row number
max_subjid	max number of subject ID to show in the message
csv_path	a path to save df in a csv file that can be shared with the DM for more details.
col_subjid	column name for subject ID. Set to NULL to ignore.

Value

df invisibly

Examples

```
library(dplyr)
tm = edc_example()
load_list(tm)
db0 %>%
  filter(age>70) %>%
  edc_data_warn("Age should not be >70", issue_n=1)

db0 %>%
  filter(age<25) %>%
  edc_data_warn("Age should not be <25", issue_n=2)

db1 %>%
  filter(n()>1, .by=SUBJID) %>%
  edc_data_warn("There are duplicated patients in `db1` ({nrow(.data)} rows)", issue_n=3)

db0 %>%
```

```
  filter(age<25) %>%
  edc_data_warn("Age should not be <25", issue_n=NULL)

edc_data_warnings()

## Not run:
db0 %>%
  filter(age<25) %>%
  edc_data_warn("Age should not be <25", csv_path="check/check_age_25.csv")

db0 %>%
  filter(age<25) %>%
  edc_data_stop("Age should *never* be <25")

## End(Not run)
```

`edc_db_to_excel` *Save the database as an Excel file*

Description

Because RStudio is not very good at showing data, it can be more convenient to browse the database using MS Excel. This function turns the whole TM export (or any named list of datasets) into an Excel workbook, with one tab for each dataset.
Use `edc_db_to_excel()` to create the file and `edc_browse_excel()` to open it.

Usage

```
edc_db_to_excel(
  filename = tempfile(fileext = ".xlsx"),
  ...,
  datasets = get_datasets(),
  overwrite = FALSE,
  open = FALSE
)

edc_browse_excel()
```

Arguments

`filename`	the path to the Excel output file. Default to a temporary file. Use the special value `TRUE` to save in "data/database_{date_extraction}.xlsx".
`...`	unused
`datasets`	a named list of dataframes. Default to the TM export.
`overwrite`	whether to overwrite any existing file. Default to `FALSE`.
`open`	whether to open the Excel file afterward. Default to `FALSE`.

Value

nothing

Examples

```
## Not run:
  tm = edc_example()
  load_list(tm)
  edc_db_to_excel() #default arguments are usually OK
  edc_db_to_excel(filename=TRUE)

## End(Not run)
```

---

edc_inform_code *Shows how many code you wrote*

Description

Shows how many code you wrote

Usage

```
edc_inform_code(main = "main.R", Rdir = "R/")
```

Arguments

main	the main R file, which sources the other ones
Rdir	the R directory, where sourced R files are located

Value

Nothing

---

edc_lookup *Retrieve the lookup table from options*

Description

Retrieve the lookup table from options

Usage

```
edc_lookup(..., check_null = TRUE)
```

Arguments

...	passed on to dplyr::arrange()
check_null	whether to stop if lookup is NULL

Value

the lookup dataframe summarizing the database import

See Also

build_lookup(), extend_lookup()

Examples

```
tm = edc_example()
load_list(tm)
edc_lookup()
edc_lookup(dataset)
```

---

edc_options *Set global options for* EDCimport

Description

Use this function to manage your EDCimport parameters globally while taking advantage of auto-completion.
Use edc_peek_options() to see which option is currently set and edc_reset_options() to set all options back to default.

Usage

```
edc_options(
  ...,
  trialmaster_pw,
  path_7zip,
  edc_lookup,
  edc_subjid_ref,
  edc_plotly,
  edc_fct_yesno,
  edc_cols_subjid,
  edc_cols_meta,
  edc_cols_id,
  edc_cols_crfname,
  edc_meta_cols_pct,
  edc_warn_max_subjid,
  edc_read_verbose,
  edc_correction_verbose,
```

```
  edc_get_key_cols_verbose,
  edc_lookup_overwrite_warn,
  .local = FALSE
)
```

Arguments

`...`	unused
`trialmaster_pw`	the password of the trialmaster zip archive. For instance, you can use `edc_options(trialmaster_pw="my` in the console once per session, so that you don't have to write the password in clear in your R code
`path_7zip`	the path to the 7zip executable. Default to `"C:/Program Files/7-Zip/"`.
`edc_lookup`	**(Internal)** a reference to the lookup table (usually `.lookup`). Should usually not be changed manually.
`edc_subjid_ref`	**used in edc_warn_patient_diffs** the vector of the reference subject IDs. You should usually write `edc_options(edc_subjid_ref=enrolres$subjid)`.
`edc_plotly`	**used in edc_swimmerplot** whether to use plotly to visualize the plot.
`edc_fct_yesno`	**used in fct_yesno** list of values to be considered as Yes/No values. Defaults to `get_yesno_lvl()`.
`edc_cols_subjid, edc_cols_meta`	**used in get_key_cols** the name of the columns holding the subject id (default to `c("ptno", "subjid")`) and the CRF form name (default to `c("crfname")`). It is case-insensitive.
`edc_cols_id, edc_cols_crfname`	deprecated
`edc_meta_cols_pct`	The minimal proportion of datasets a column has to reach to be considered "meta"
`edc_warn_max_subjid`	The max number of subject IDs to show in edc_data_warn
`edc_read_verbose, edc_correction_verbose, edc_get_key_cols_verbose`	the verbosity of the output of functions read_trialmaster and read_tm_all_xpt, manual_correction, and get_key_cols. For example, set `edc_options(edc_read_verbose=0)` to silence the first 2.
`edc_lookup_overwrite_warn`	default to TRUE. Whether there should be warning when overwriting `.lookup` (like when reading 2 databases successively)
`.local`	if TRUE, the effect will only apply to the local frame (internally using `rlang::local_options()`)

Value

Nothing, called for its side effects

edc_peek_options *See which* EDCimport *option is currently set.*

Description

See which EDCimport option is currently set.

Usage

```
edc_peek_options(keep_null = FALSE)
```

Arguments

keep_null	set to TRUE to get a list

Value

A named list of EDCimport options

edc_population_plot *Plot the populations*

Description

In a RCT, you usually have several populations of analysis, and this function allow to show which patient is in which population graphically.

Usage

```
edc_population_plot(x, id_per_row = 50, ref = "first")
```

Arguments

x	a named list of subject ID.
id_per_row	number of patients per rows.
ref	the whole population. Default to the first member of x.

Value

a ggplot

Examples

```
#in real word code, use filter and pull to get these vectors
pop_total = c(1:180) %>% setdiff(55) #screen failure, no patient 55
pop_itt = pop_total %>% setdiff(10) #patient 10 has had the wrong treatment
pop_safety = pop_total %>% setdiff(c(40,160)) #patients 40 and 160 didn't receive any treatment
pop_m_itt = pop_total %>% setdiff(c(40,160,80)) #patient 80 had a wrong inclusion criterion
pop_evaluable = pop_total %>% setdiff(c(40,160,101,147,186)) #patients with no recist evaluation

l = list(
  "Total population"=pop_total,
  "ITT population"=pop_itt,
  "Safety population"=pop_safety,
  "mITT population"=pop_m_itt,
  "Evaluable population"=pop_evaluable
)
edc_population_plot(l)
edc_population_plot(l[-1], ref=pop_total)
edc_population_plot(l, ref=1:200)
edc_population_plot(l, id_per_row=60)
```

---

edc_reset_options *Reset all* EDCimport *options.*

Description

Reset all EDCimport options.

Usage

```
edc_reset_options(
  except = c("edc_lookup", "trialmaster_pw", "path_7zip"),
  quiet = FALSE
)
```

Arguments

except	options that are not reset by default
quiet	set to TRUE to remove the message.

Value

Nothing, called for its side effects

`edc_swimmerplot` *Swimmer plot of all dates columns*

Description

Join all tables from `.lookup$dataset` on `id`

Usage

```
edc_swimmerplot(
  .lookup = edc_lookup(),
  ...,
  id = get_subjid_cols(),
  group = NULL,
  origin = NULL,
  id_lim = NULL,
  exclude = NULL,
  time_unit = c("days", "weeks", "months", "years"),
  aes_color = c("variable", "label"),
  plotly = getOption("edc_plotly", FALSE)
)
```

Arguments

`.lookup`	the lookup table, default to `edc_lookup()`
`...`	not used
`id`	the patient identifier. Will be coerced as numeric.
`group`	a grouping variable, given as "dataset$column"
`origin`	a variable to consider as time 0, given as "dataset$column"
`id_lim`	a numeric vector of length 2 providing the minimum and maximum `id` to subset on.
`exclude`	a character vector of variables to exclude, in the form `dataset$column`. Can be a regex, but $ symbols don't count. Case-insensitive.
`time_unit`	if `origin!=NULL`, the unit to measure time. One of `c("days", "weeks", "months", "years")`.
`aes_color`	either `variable` ("{dataset} - {column}") or `label` (the column label)
`plotly`	whether to use `{plotly}` to get an interactive plot

Value

either a `plotly` or a `ggplot`

Examples

```
#tm = read_trialmaster("filename.zip", pw="xx")
tm = edc_example_plot()
load_list(tm)
p = edc_swimmerplot(.lookup, id_lim=c(5,45))
p2 = edc_swimmerplot(.lookup, origin="db0$date_naissance", time_unit="weeks",
                     exclude=c("DB1$DATE2", "db3$.*"))
p3 = edc_swimmerplot(.lookup, group="db0$group", aes_color="label")
## Not run:
#save the plotly plot as HTML to share it
save_plotly(p, "edc_swimmerplot.html")

## End(Not run)
```

`edc_warn_extraction_date` *Warn if extraction is too old*

Description

Warn if extraction is too old

Usage

```
edc_warn_extraction_date(max_days = 30)
```

Arguments

`max_days`	the max acceptable age of the data

Value

nothing

Examples

```
tm = edc_example()
load_list(tm)
edc_warn_extraction_date()
```

`edc_warn_patient_diffs` *Check the validity of the subject ID column*

Description

Compare a subject ID vector to the study's reference subject ID (usually something like `enrolres$subjid`), and warn if any patient is missing or extra.
`check_subjid()` is the old, deprecated name.

Usage

```
edc_warn_patient_diffs(
  x,
  ref = getOption("edc_subjid_ref"),
  issue_n = "xx",
  data_name = NULL,
  col_subjid = get_subjid_cols()
)
```

Arguments

`x`	the subject ID vector to check, or a dataframe which ID column will be guessed
`ref`	the reference for subject ID. Should usually be set through `edc_options(edc_subjid_ref=xxx)`. See example.
`issue_n`	identifying row number
`data_name`	the name of the data (for the warning message)
`col_subjid`	name of the subject ID column if `x` is a dataframe.

Value

nothing, called for errors/warnings

Examples

```
tm = edc_example()
load_list(tm)
options(edc_subjid_ref=db0$SUBJID)
#usually, you set something like:
#options(edc_subjid_ref=enrolres$subjid)
edc_warn_patient_diffs(db1)
db1 %>% dplyr::filter(SUBJID>1) %>% edc_warn_patient_diffs()
edc_warn_patient_diffs(c(db1$SUBJID, 99, 999))
```

`extend_lookup` *Extend the lookup table*

Description

This utility extends the lookup table to include:

- `n_id` the number of patients present in the dataset
- `rows_per_id` the mean number of row per patient
- `crfname` the actual name of the dataset

Usage

```
extend_lookup(
  lookup,
  ...,
  id_cols = get_subjid_cols(lookup),
  crf_cols = get_crfname_cols(lookup),
  datasets = get_datasets(lookup, envir = parent.frame())
)
```

Arguments

`lookup`	[`data.frame(1)`] the lookup table
`...`	unused
`id_cols, crf_cols`	[`character(n)`] for experts only
`datasets`	[`data.frame(n)`] for experts only

Value

the lookup, extended

See Also

`build_lookup()`, `edc_lookup()`

Examples

```
#tm = read_trialmaster("filename.zip", pw="xx")
tm = edc_example_mixed()
load_list(tm)
.lookup
.lookup = extend_lookup(.lookup)
.lookup
```

`fct_yesno` *Format factor levels as Yes/No*

Description

Format factor levels as arbitrary values of Yes/No (with Yes always first) while **leaving untouched** all vectors that contain other information.

Usage

```
fct_yesno(
  x,
  input = list(yes = c("Yes", "Oui"), no = c("No", "Non")),
  output = c("Yes", "No"),
  strict = FALSE,
  mutate_character = TRUE,
  fail = TRUE
)
```

Arguments

`x`	a vector of any type/class.
`input`	list of values to be considered as "yes" and "no".
`output`	the output factor levels.
`strict`	whether to match the input strictly or use stringr::str_detect to find them.
`mutate_character`	whether to turn characters into factor.
`fail`	whether to fail if some levels cannot be recoded to yes/no.

Value

a factor, or `x` untouched.

Examples

```
fct_yesno(c("No", "Yes")) #levels are in order

set.seed(42)
N=6
x = tibble(
  a=sample(c("Yes", "No"), size=N, replace=TRUE),
  b=sample(c("Oui", "Non"), size=N, replace=TRUE),
  c=sample(0:1, size=N, replace=TRUE),
  d=sample(c(TRUE, FALSE), size=N, replace=TRUE),
  e=sample(c("1-Yes", "0-No"), size=N, replace=TRUE),

  y=sample(c("aaa", "bbb", "ccc"), size=N, replace=TRUE),
```

```
  z=1:N,
)

x
#y and z are left untouched (or throw an error if fail=TRUE)
sapply(x, fct_yesno, fail=FALSE)

# as "1-Yes" is not in `input`, x$e is untouched/fails if strict=TRUE
fct_yesno(x$e)
fct_yesno(x$e, strict=TRUE, fail=FALSE)
fct_yesno(x$e, output=c("Ja", "Nein"))
```

`find_keyword` *Find a keyword in the whole database*

Description

Find a keyword in all names and labels of a list of datasets.

Usage

```
find_keyword(keyword, data = edc_lookup(), ignore_case = TRUE)
```

Arguments

`keyword`	the keyword to search for. Can handle regular expressions (see examples).
`data`	the lookup dataframe where to search the keyword. Can be set using `edc_options(edc_lookup=my_data` which is done automatically when calling `read_trialmaster()`.
`ignore_case`	should case differences be ignored in the match? Default to `TRUE`.

Value

a tibble

Examples

```
## Not run:
path = system.file("extdata/Example_Export_SAS_XPORT_2022_08_25_15_16.zip",
                   package="EDCimport", mustWork=TRUE)
w = read_trialmaster(path, verbose=FALSE)

find_keyword("patient")

#with regex
find_keyword("patient$")
find_keyword("\\d")
find_keyword("(Trial|Form) Name")
find_keyword("\\(") #you need to escape special characters

## End(Not run)
```

get_common_cols *Get columns that are common to multiple datasets*

Description

[Experimental] Attempt to list all columns in the database and group the ones that are common to some datasets. Useful to find keys to pivot or summarise data.

Usage

```
get_common_cols(lookup = edc_lookup(), min_datasets = 3)

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

Arguments

lookup	the lookup table, default to edc_lookup()
min_datasets	the minimal number of datasets to be considered
object	an object of class "common_cols"
...	unused

Value

a tibble of class "common_cols"

Examples

```
tm = edc_example()
load_list(tm)
x = get_common_cols(min_datasets=1)
x
summary(x)
```

get_datasets *Retrieve the datasets as a list of data.frames*

Description

Get the datasets from the lookup table as a list of data.frames.

Usage

```
get_datasets(lookup = edc_lookup(), envir = parent.frame())
```

Arguments

lookup	the lookup table
envir	(internal use)

Value

a list of all datasets

---

get_key_cols *Important column names*

Description

Retrieve names of patient_id (usually "SUBJID" and "PATNO") and crfname (usually "CRFNAME") from the actual names of the datasets

Usage

```
get_key_cols(lookup = edc_lookup())
```

Arguments

lookup	the lookup table

Value

a list(2) of characters with names patient_id and crfname

---

get_meta_cols *Get columns shared by most datasets*

Description

In most trialmaster exports, many datasets share a certain amount of columns containing meta-data that are often irrelevant to the point. This function identifies the columns that are present in at least 95% of datasets (by default)

Usage

```
get_meta_cols(min_pct = getOption("edc_meta_cols_pct", 0.95))
```

Arguments

min_pct	Default=0.95. The minimal proportion of datasets a column has to reach. Subject ID is always excluded.

Value

a character vector

Examples

```
tm = edc_example_mixed()
load_list(tm)
meta_cols = get_meta_cols()
long_mixed %>% dplyr::select(-dplyr::any_of(meta_cols))
```

---

get_subjid_cols *Get key column names*

Description

Retrieve names of patient ID and CRF name from the actual names of the datasets, without respect of the case. Default values should be set through options.

Usage

```
get_subjid_cols(lookup = edc_lookup())

get_crfname_cols(lookup = edc_lookup())
```

Arguments

lookup	the lookup table

Value

a character vector

options

Use `edc_options()` to set default values:

- `edc_cols_subjid` defaults to `c("PTNO", "SUBJID")`
- `edc_cols_crfname` defaults to `c("CRFNAME")`

Examples

```
get_subjid_cols()
get_crfname_cols()
```

harmonize_subjid *Harmonize the subject ID of the database*

Description

Turns the subject ID columns of all datasets into a factor containing levels for all the subjects of the database. Avoid problems when joining tables, and some checks can be performed on the levels.

Usage

```
harmonize_subjid(datalist, preprocess = NULL, col_subjid = get_subjid_cols())
```

Arguments

datalist	a list of dataframes
preprocess	an optional function to modify the subject ID column, for example `as.numeric()`. See examples.
col_subjid	the names of the columns holding the subject ID (as character)

Value

datalist, with subject id modified

Examples

```
db = edc_example()
db$db0 = head(db$db0, 10)
db$db0$SUBJID %>% head()
db = harmonize_subjid(db)
db$db0$SUBJID %>% head()
db = harmonize_subjid(db, preprocess=function(x) paste0("#", x))
db$db0$SUBJID %>% head()
```

lastnews_table *Get a table with the latest date for each patient*

Description

This function search for date columns in every tables and returns the latest date for each patient with the variable it comes from. Useful in survival analysis to get the right censoring time.

Usage

```
lastnews_table(
  except = NULL,
  with_ties = FALSE,
  numeric_id = TRUE,
  prefer = NULL,
  warn_if_future = TRUE
)
```

Arguments

except	the datasets/columns that should not be searched. Example: a scheduled visit for which the patient may have died before attending should not be considered.
with_ties	in case of tie, whether to return the first `origin` (FALSE) or all the origins that share this tie (TRUE).
numeric_id	set to FALSE if the patient ID column is not numeric
prefer	preferred origins in the event of a tie. Usually the followup table.
warn_if_future	whether to show a warning about dates that are after the extraction date

Value

a dataframe

Examples

```
tm = edc_example_plot()
load_list(tm)
lastnews_table()
lastnews_table(except="db3")
lastnews_table(except="db3$date9")
lastnews_table(prefer="db2")
```

---

load_as_list *Load a* `.RData` *file as a list*

Description

Instead of loading a `.RData` file in the global environment, extract every object into a list.

Usage

```
load_as_list(filename)
```

Arguments

filename	the filename, with the `.RData` extension.

Value

a list

Examples

```
x = list(a=1, b=mtcars)
save_list(x, "test.RData")
y = load_as_list("test.RData")
print(y$a)
```

`load_list` *Load a list in an environment*

Description

Load a list in an environment

Usage

```
load_list(x, env = parent.frame(), remove = TRUE)
```

Arguments

`x`	a list
`env`	the environment onto which the list should be loaded
`remove`	if `TRUE`, `x` will be removed from the environment afterward

Value

nothing, called for its side-effect

Examples

```
x=list(a=1, b=mtcars)
load_list(x, remove=FALSE)
print(a)
print(nrow(b))
```

`manual_correction` *Manual correction*

Description

When finding wrong or unexpected values in an exported dataset, it can be useful to temporarily correct them by hard-coding a value. However, this manual correction should be undone as soon as the central database is updated with the correction.

- `manual_correction()` applies a correction in a specific dataset column location and throws an error if the correction is already in place. This check applies only once per R session so you can source your script without errors.
- `reset_manual_correction()` resets all checks. For instance, it is called by `read_trialmaster()`.

Usage

```
manual_correction(
  data,
  col,
  rows,
  wrong,
  correct,
  verbose = getOption("edc_correction_verbose", TRUE)
)

reset_manual_correction()
```

Arguments

`data`, `col`, `rows`	the rows of a column of a dataframe where the error lies
`wrong`	the actual wrong value
`correct`	the temporary correction value
`verbose`	whether to print informations (once)

Value

Nothing, used for side effects

Examples

```
library(dplyr)
x = iris %>% mutate(id=row_number(), .before=1) %>% as_tibble()
x$Sepal.Length[c(1,3,5)]

#1st correction is silent
manual_correction(x, Sepal.Length, rows=c(1,3,5),
                  wrong=c(5.1, 4.7, 5.0), correct=c(5, 4, 3))
```

```
x$Sepal.Length[c(1,3,5)]

#further correction is silent
manual_correction(x, Sepal.Length, rows=c(1,3,5),
                  wrong=c(5.1, 4.7, 5.0), correct=c(5, 4, 3))

#if the database is corrected, an error is thrown
## Not run:
reset_manual_correction()
x$Sepal.Length[c(1,3,5)] = c(5, 4, 3) #mimics db correction
manual_correction(x, Sepal.Length, rows=c(1,3,5),
                  wrong=c(5.1, 4.7, 5.0), correct=c(5, 4, 3))

## End(Not run)
```

`read_all_csv` *Read all* `.csv` *files in a directory*

Description

Read all `.csv` files in a directory, with labels if specified.

Usage

```
read_all_csv(
  path,
  ...,
  labels_from = NULL,
  clean_names_fun = NULL,
  read_fun = "guess",
  datetime_extraction = "guess",
  verbose = getOption("edc_read_verbose", 1)
)
```

Arguments

`path`	[`character(1)`] path to the directory containing `.csv` files.
`...`	unused
`labels_from`	[`misc`] list of path to file containing the labels.
`clean_names_fun`	[`function`] a function to clean column names, e.g. tolower, `janitor::clean_names()`,...
`read_fun`	[`function`] a function to read the files in path, e.g. `read.csv()`, `read.csv2()`,...

datetime_extraction
: [dateish(1)]
the datetime of database extraction (database lock). If "guess", the datetime will be inferred from the files modification time.

verbose
: [numeric(1)]
the level of verbosity

Value

a list containing one dataframe for each .csv file in the folder, the extraction date (datetime_extraction), and a summary of all imported tables (.lookup).

Labels file

labels_from should contain the information about column labels. It should be a data file (.csv) containing 2 columns: one for the column name and the other for its associated label. Use options(edc_col_name="xxx", edc_col_label="xxx") to specify the names of the columns.

---

read_all_sas *Read all .sas7bdat files in a directory*

Description

Read all .sas7bdat files in a directory. Formats can be applied from a procformat.sas SAS file, from a .

Usage

```
read_all_sas(
  path,
  ...,
  format_file = "procformat.sas",
  clean_names_fun = NULL,
  datetime_extraction = "guess",
  verbose = getOption("edc_read_verbose", 1)
)
```

Arguments

path
: [character(1)]
the path to the directory containing all .sas7bdat files.

...
: unused

format_file
: [character(1)]
the path to the file that should be used to apply formats. See details. Use NULL to not apply formats.

clean_names_fun
: [function]
a function to clean column names, e.g. tolower, janitor::clean_names(),...

datetime_extraction
: [POSIXt(1)]
the datetime of the data extraction. Default to the most common date of last modification in directory.

verbose
: [logical(1)]
one of c(0, 1, 2). The higher, the more information will be printed.

Value

a list containing one dataframe for each .xpt file in the folder, the extraction date (datetime_extraction), and a summary of all imported tables (.lookup).

Format file

format_file should contain the information about SAS formats. It can be either

- a procformat.sas file, containing the whole PROC FORMAT
- a catalog file (.sas7bcat)
- or a data file (.csv or .sas7bdat) containing 3 columns: the SAS format name (repeated), each level, and its associated label. Use options(edc_var_format_name="xxx", edc_var_level="xxx", edc_var_label="xxx") to specify the names of the columns.

---

read_all_xpt *Read all .xpt files in a directory*

Description

Read all .xpt files in a directory (unzipped TrialMaster archive).
If 7zip is installed, you should probably rather use read_trialmaster() instead.
If a procformat.sas file exists in the directory, formats will be applied.

Usage

```
read_all_xpt(
  path,
  ...,
  format_file = "procformat.sas",
  clean_names_fun = NULL,
  split_mixed = FALSE,
  extend_lookup = TRUE,
  datetime_extraction = "guess",
  verbose = getOption("edc_read_verbose", 1),
  directory = "deprecated",
  key_columns = "deprecated"
)
```

Arguments

path	[character(1)] the path to the directory containing all .xpt files.
...	unused
format_file	[character(1)] the path to the file that should be used to apply formats. See details. Use NULL to not apply formats.
clean_names_fun	[function] a function to clean column names, e.g. tolower, janitor::clean_names(),...
split_mixed	[logical(1): FALSE] whether to split mixed datasets. See split_mixed_datasets.
extend_lookup	[character(1): FALSE] whether to enrich the lookup table. See extend_lookup.
datetime_extraction	[POSIXt(1)] the datetime of the data extraction. Default to the most common date of last modification in directory.
verbose	[logical(1)] one of c(0, 1, 2). The higher, the more information will be printed.
directory	deprecated
key_columns	deprecated

Value

a list containing one dataframe for each .xpt file in the folder, the extraction date (datetime_extraction), and a summary of all imported tables (.lookup).

Format file

format_file should contain the information about SAS formats. It can be either

- a procformat.sas file, containing the whole PROC FORMAT
- or a data file (.csv or .sas7bdat) containing 3 columns: the SAS format name (repeated), each level, and its associated label. Use options(edc_var_format_name="xxx", edc_var_level="xxx", edc_var_label="xxx") to specify the names of the columns.

`read_trialmaster` *Read the* `.zip` *archive of a TrialMaster export*

Description

Import the `.zip` archive of a TrialMaster trial export as a list of dataframes. The archive filename should be leaved untouched as it contains the project name and the date of extraction.
Generate a `.rds` cache file for future reads.
If `7zip` is not installed or available, use `read_tm_all_xpt()` instead.

Usage

```
read_trialmaster(
  archive,
  ...,
  use_cache = "write",
  clean_names_fun = NULL,
  split_mixed = FALSE,
  extend_lookup = TRUE,
  pw = getOption("trialmaster_pw"),
  verbose = getOption("edc_read_verbose", 1),
  key_columns = "deprecated"
)
```

Arguments

`archive`	`[character(1)]` the path to the archive
`...`	unused
`use_cache`	`[mixed(1): "write"]` controls the `.rds` cache. If `TRUE`, read the cache if any or extract the archive and create a cache. If `FALSE` extract the archive without creating a cache file. Can also be `"read"` or `"write"`.
`clean_names_fun`	`[function]` a function to clean column names, e.g. tolower, `janitor::clean_names()`,...
`split_mixed`	`[logical(1): FALSE]` whether to split mixed datasets. See split_mixed_datasets.
`extend_lookup`	`[character(1): FALSE]` whether to enrich the lookup table. See extend_lookup.
`pw`	`[character(1)]` The password if the archive is protected. To avoid writing passwords in plain text, it is probably better to use `options(trialmaster_pw="xxx")` instead though.

verbose	[logical(1)] one of c(0, 1, 2). The higher, the more information will be printed.
key_columns	deprecated

Value

a list containing one dataframe for each .xpt file in the folder, the extraction date (datetime_extraction), and a summary of all imported tables (.lookup).

---

save_list *Save a list as .RData file*

Description

Save a list as .RData file

Usage

```
save_list(x, filename)
```

Arguments

x	a list
filename	the filename, with the .RData extension.

Value

nothing, called for its side-effect

Examples

```
x=list(a=1, b=mtcars)
save_list(x, "test.RData")
load("test.RData")
file.remove("test.RData")
print(a)
print(nrow(b))
```

save_plotly *Save a plotly to an HTML file*

Description

Save a plotly to an HTML file

Usage

```
save_plotly(p, file, ...)
```

Arguments

p	a plot object (plotly or ggplot)
file	a file path to save the HTML file
...	passed on to htmlwidgets::saveWidget

Value

nothing, used for side effect

Examples

```
## Not run:
tm = edc_example_plot()
p = edc_swimmerplot(tm$.lookup, id_lim=c(5,45))
save_plotly(p, "graph/swimplots/edc_swimmerplot.html", title="My Swimmerplot")

## End(Not run)
```

save_sessioninfo *Save* sessionInfo() *output*

Description

Save sessionInfo() output into a text file.

Usage

```
save_sessioninfo(path = "check/session_info.txt", with_date = TRUE)
```

Arguments

path	target path to write the file
with_date	whether to insert the date before the file extension

Value

nothing

Examples

```
## Not run:
   save_sessioninfo()

## End(Not run)
```

`search_for_newer_data` *Search for newer data*

Description

Search in some folders if a TrialMaster database more recent than the current extraction is present. By default, it will search the "data" folder and the OS usual "Downloads" folder. If a newer database is found, user will be asked if they want to move it to the "data" folder.

Usage

```
search_for_newer_data(
  archive,
  ...,
  source = path_home("Downloads"),
  target = "data",
  ask = TRUE,
  advice = TRUE
)
```

Arguments

`archive`	TM archive path, giving the project name and date
`...`	unused
`source`	the path vector to be searched, default to both "data" and the usual "Downloads" folder
`target`	the path where files should be copied
`ask`	whether to ask the user to move the file to "data"
`advice`	whether to advice how to move it instead, if `ask==FALSE`

Value

the path to the newer file, invisibly.

Examples

```
## Not run:
  archive = "data/MYPROJECT_ExportTemplate_xxx_SAS_XPORT_2024_06_01_12_00.zip"
  #tm = read_trialmaster(archive)
  search_for_newer_data(archive)

## End(Not run)
```

`select_distinct` *Select only distinct columns*

Description

Select all columns that has only one level for a given grouping scope. Useful when dealing with mixed datasets containing both long data and repeated short data.

Usage

```
select_distinct(df, .by)
```

Arguments

`df`	a dataframe
`.by`	optional grouping columns

Value

`df` with less columns

Examples

```
tm = edc_example_ae()
tm$ae %>% names
tm$ae %>% select_distinct() %>% names
tm$ae %>% select_distinct(.by=subjid) %>% names
```

split_mixed_datasets *Split mixed datasets*

Description

Split mixed tables, i.e. tables that hold both long data (N values per patient) and short data (one value per patient, duplicated on N lines), into one long table and one short table.

Usage

```
split_mixed_datasets(
  datasets = get_datasets(),
  id = get_subjid_cols(),
  ...,
  ignore_cols = get_meta_cols(0.95),
  output_code = FALSE,
  verbose = TRUE
)
```

Arguments

datasets	a dataframe or a list of dataframes to split. Default to all the datasets from .lookup.
id	the patient identifier, probably "SUBJID". Should be shared by all datasets. Case-insensitive.
...	not used
ignore_cols	columns to ignore when considering a table as long. Default to getOption("edc_cols_crfname", "CRFNAME"). Case-insensitive.
output_code	whether to print the code to explicitly write. Can also be a file path.
verbose	whether to print informations about the process.

Value

a list of the new long and short tables. Use load_list() to load them into the global environment.

Examples

```
#tm = read_trialmaster("filename.zip", pw="xx")
tm = edc_example_mixed()
names(tm)
#load_list(tm)
print(tm$long_mixed) #`val1` and `val2` are long but `val3` is short

mixed_data = split_mixed_datasets(tm, id="subjid", verbose=TRUE)
load_list(mixed_data)
print(long_mixed_short)
print(long_mixed_long)
```

```
#alternatively, get the code and only use the datasets you need
split_mixed_datasets(tm, id="SUBJID", output_code=TRUE)
filename = tempfile("mixed_code", fileext=".R")
split_mixed_datasets(tm, id="SUBJID", output_code=filename)
readLines(filename)
```

`table_format` *Identify if a dataframe has a long or a wide format*

Description

A dataset is either in the wide format or in the long format (link). This function identifies the format of a dataframe with respect to a subject ID. If a dataframe has some wide and long columns, it is considered "mixed".

Usage

```
table_format(
  df,
  id = get_subjid_cols(),
  ...,
  ignore_cols = get_meta_cols(0.95),
  na_rm = FALSE,
  warn = TRUE
)
```

Arguments

`df`	a dataframe
`id`	the identifying subject ID
`...`	not used
`ignore_cols`	columns to ignore. Usually meta columns (see get_meta_cols).
`na_rm`	whether to consider missing values
`warn`	whether to warn if ID is not found

Value

a string value in `c("wide", "long", "mixed)`

Examples

```
tm = edc_example_mixed()
sapply(tm, table_format, warn=FALSE)
```

unify *Unify a vector*

Description

Turn a vector of length N to a vector of length 1 after checking that there is only one unique value. Useful to safely flatten a duplicated table. This preserves the `label` attribute if set.

Usage

```
unify(x)
```

Arguments

x	a vector

Value

a vector of length 1

Examples

```
unify(c(1,1,1,1))
#unify(c(1,1,2,1)) #warning

library(dplyr)
x=tibble(id=rep(letters[1:5],10), value=rep(1:5,10))
x %>% group_by(id) %>% summarise(value=unify(value)) #safer than `value=value[1]`
x$value[2]=1
#x %>% group_by(id) %>% summarise(value=unify(value)) #warning about that non-unique value
```

Index