Package: REDCapCAST (via r-universe)

October 29, 2026

Title REDCap Metadata Casting and Castellated Data Handling

Version 26.10.1

Description Casting metadata for REDCap database creation and handling of castellated data using repeated instruments and longitudinal projects in 'REDCap'. Keeps a focused data export approach, by allowing to only export required data from the database. Also for casting new REDCap databases based on datasets from other sources. Originally forked from the R part of 'REDCapRITS' by Paul Egeler. See <https://github.com/pegeler/REDCapRITS>. 'REDCap' (Research Electronic Data Capture) is a secure, web-based software platform designed to support data capture for research studies, providing 1) an intuitive interface for validated data capture; 2) audit trails for tracking data manipulation and export procedures; 3) automated export procedures for seamless data downloads to common statistical packages; and 4) procedures for data integration and interoperability with external sources (Harris et al (2009) <doi:10.1016/j.jbi.2008.08.010>; Harris et al (2019) <doi:10.1016/j.jbi.2019.103208>).

Depends R (>= 4.1.0)

Suggests Hmisc, knitr, rmarkdown, devtools, roxygen2, spelling, jsonlite, testthat

License GPL (>= 3)

Encoding UTF-8

LazyData true

URL https://github.com/agdamsbo/REDCapCAST, https://agdamsbo.github.io/REDCapCAST/

BugReports https://github.com/agdamsbo/REDCapCAST/issues

Imports dplyr, REDCapR, tidyr, tidyselect, keyring, purrr, readr, stats, zip, assertthat, forcats, vctrs, gt, bslib, here, glue, gtsummary, shiny, haven, openxlsx2, readODS

Language en-US

VignetteBuilder knitr

Collate 'REDCapCAST-package.R' 'utils.r' 'process_user_input.r'
'REDCap_split.r' 'as_factor.R' 'as_logical.R' 'doc2dd.R'
'ds2dd_detailed.R' 'easy_redcap.R' 'export_redcap_instrument.R'
'fct_drop.R' 'html_styling.R' 'mtcars_redcap.R'
'read_redcap_instrument.R' 'read_redcap_tables.R'
'redcap_wider.R' 'redcapcast_data.R' 'redcapcast_meta.R'
'shiny_cast.R' 'shiny_descr.R' 'topological_sort.R'

Config/roxygen2/version 8.1.0

Repository https://agdamsbo.r-universe.dev

Date/Publication 2026-10-29 11:24:27 UTC

RemoteUrl https://github.com/agdamsbo/redcapcast

RemoteRef HEAD

RemoteSha 1f9a2b75bc8c5a3e6ad7d1188719d14416378948

Contents

all_na . . . 4
apply_factor_labels . . . 4
apply_field_label . . . 5
as_factor . . . 5
as_logical . . . 7
case_match_regex_list . . . 8
cast_data_overview . . . 9
cast_meta_overview . . . 9
char2choice . . . 10
char2cond . . . 10
clean_field_label . . . 11
clean_redcap_name . . . 12
compact_vec . . . 12
create_html_table . . . 13
create_instrument_meta . . . 14
cut_string_length . . . 15
d2w . . . 15
doc2dd . . . 16
ds2dd . . . 17
ds2dd_detailed . . . 19
easy_redcap . . . 21
export_redcap_instrument . . . 22
fct_drop . . . 23
fct2num . . . 23
file_extension . . . 24
focused_metadata . . . 25
format_redcap_factor . . . 25
format_subheader . . . 26
get_api_key . . . 26

get_attr . . . 27
get_id_name . . . 28
guess_time_only . . . 28
guess_time_only_filter . . . 29
haven_all_levels . . . 30
hms2character . . . 30
html_tag_wrap . . . 31
is.labelled . . . 31
is_missing . . . 32
is_repeated_longitudinal . . . 32
mark_complete . . . 33
match_fields_to_form . . . 34
mtcars_redcap . . . 34
named_levels . . . 35
nav_bar_page . . . 36
numchar2fct . . . 36
parse_data . . . 37
possibly_numeric . . . 38
possibly_roman . . . 38
process_user_input . . . 39
process_user_input.character . . . 39
process_user_input.data.frame . . . 40
process_user_input.default . . . 40
process_user_input.response . . . 41
read_input . . . 41
read_redcap_instrument . . . 42
read_redcap_tables . . . 42
REDCap_split . . . 44
redcap_wider . . . 46
redcapcast_data . . . 48
redcapcast_meta . . . 49
replace_curly_quote . . . 50
safe_topological_sort . . . 50
sanitize_split . . . 51
set_attr . . . 52
shiny_cast . . . 53
split_non_repeating_forms . . . 53
strsplitx . . . 54
suffix2label . . . 55
time_only_correction . . . 56
topological_sort . . . 56
var2fct . . . 58
vec2choice . . . 59

Index **60**

all_na *Check if vector is all NA*

Description

Check if vector is all NA

Usage

```
all_na(data)
```

Arguments

data	vector of data.frame

Value

logical

Examples

```
rep(NA, 4) |> all_na()
```

apply_factor_labels *Preserve all factor levels from REDCap data dictionary in data export*

Description

Preserve all factor levels from REDCap data dictionary in data export

Usage

```
apply_factor_labels(data, meta = NULL)
```

Arguments

data	REDCap exported data set
meta	REDCap data dictionary

Value

data.frame

apply_field_label *Apply REDCap filed labels to data frame*

Description

Apply REDCap filed labels to data frame

Usage

```
apply_field_label(data, meta)
```

Arguments

data	REDCap exported data set
meta	REDCap data dictionary

Value

data.frame

as_factor *Convert labelled vectors to factors while preserving attributes*

Description

This extends as_factor as well as as_factor, by appending original attributes except for "class" after converting to factor to avoid ta loss in case of rich formatted and labelled data.

Usage

```
as_factor(x, ...)

## S3 method for class 'factor'
as_factor(x, ...)

## S3 method for class 'logical'
as_factor(x, ...)

## S3 method for class 'numeric'
as_factor(x, ...)

## S3 method for class 'character'
as_factor(x, ...)

## S3 method for class 'haven_labelled'
```

```
as_factor(
  x,
  levels = c("default", "labels", "values", "both"),
  ordered = FALSE,
  ...
)

## S3 method for class 'labelled'
as_factor(
  x,
  levels = c("default", "labels", "values", "both"),
  ordered = FALSE,
  ...
)

## S3 method for class 'data.frame'
as_factor(x, ..., only_labelled = TRUE)
```

Arguments

x	Object to coerce to a factor.
...	Other arguments passed down to method.
levels	How to create the levels of the generated factor: * "default": uses labels where available, otherwise the values. Labels are sorted by value. * "both": like "default", but pastes together the level and value * "label": use only the labels; unlabelled values become ‘NA‘ * "values": use only the values
ordered	If ‘TRUE‘ create an ordered (ordinal) factor, if ‘FALSE‘ (the default) create a regular (nominal) factor.
only_labelled	Only apply to labelled columns?

Details

Please refer to parent functions for extended documentation. To avoid redundancy calls and errors, functions are copy-pasted here

Empty variables with empty levels attribute are interpreted as logicals

Examples

```
# will preserve all attributes
c(1, 4, 3, "A", 7, 8, 1) |> as_factor()
structure(c(1, 2, 3, 2, 10, 9),
  labels = c(Unknown = 9, Refused = 10)
) |>
  as_factor() |>
  dput()

structure(c(1, 2, 3, 2, 10, 9),
```

```
  labels = c(Unknown = 9, Refused = 10),
  class = "haven_labelled"
) |>
  as_factor() |> class()
structure(rep(NA,10),
  class = c("labelled")
) |>
  as_factor() |> summary()

rep(NA,10) |> as_factor()
```

`as_logical` *Interpret specific binary values as logicals*

Description

Interpret specific binary values as logicals

Usage

```
as_logical(
  x,
  values = list(c("TRUE", "FALSE"), c("Yes", "No"), c(1, 0), c(1, 2)),
  ...
)

## S3 method for class 'data.frame'
as_logical(
  x,
  values = list(c("TRUE", "FALSE"), c("Yes", "No"), c(1, 0), c(1, 2)),
  ...
)

## Default S3 method:
as_logical(
  x,
  values = list(c("TRUE", "FALSE"), c("Yes", "No"), c(1, 0), c(1, 2)),
  ...
)
```

Arguments

`x`	vector or data.frame
`values`	list of values to interpret as logicals. First value is
`...`	ignored interpreted as TRUE.

Value

vector

Examples

```
c(sample(c("TRUE", "FALSE"), 20, TRUE), NA) |>
  as_logical() |>
  class()
ds <- dplyr::tibble(
  B = factor(sample(c(1, 2), 20, TRUE)),
  A = factor(sample(c("TRUE", "FALSE"), 20, TRUE)),
  C = sample(c(3, 4), 20, TRUE),
  D = factor(sample(c("In", "Out"), 20, TRUE))
)
ds |>
  as_logical() |>
  sapply(class)
ds$A |> class()
sample(c("TRUE",NA), 20, TRUE) |>
  as_logical()
as_logical(0)
```

---

case_match_regex_list *List-base regex case_when*

Description

Mimics case_when for list of regex patterns and values. Used for date/time validation generation from name vector. Like case_when, the matches are in order of priority. Primarily used in REDCapCAST to do data type coding from systematic variable naming.

Usage

```
case_match_regex_list(data, match.list, .default = NA)
```

Arguments

data	vector
match.list	list of case matches
.default	Default value for non-matches. Default is NA.

Value

vector

Examples

```
case_match_regex_list(
  c("test_date", "test_time", "test_tida", "test_tid"),
  list(date_dmy = "_dat[eo]$", time_hh_mm_ss = "_ti[md]e?$")
)
```

`cast_data_overview` *Overview of REDCapCAST data for shiny*

Description

Overview of REDCapCAST data for shiny

Usage

```
cast_data_overview(data)
```

Arguments

`data`	list with class 'REDCapCAST'

Value

gt object

`cast_meta_overview` *Overview of REDCapCAST meta data for shiny*

Description

Overview of REDCapCAST meta data for shiny

Usage

```
cast_meta_overview(data)
```

Arguments

`data`	list with class 'REDCapCAST'

Value

gt object

char2choice *Simple function to generate REDCap choices from character vector*

Description

Simple function to generate REDCap choices from character vector

Usage

```
char2choice(data, char.split = "/", raw = NULL, .default = NA)
```

Arguments

data	vector
char.split	splitting character(s)
raw	specific values. Can be used for options of same length.
.default	default value for missing. Default is NA.

Value

vector

Examples

```
char2choice(c("yes/no","  yep. / nope  ","",NA,"what"),.default=NA)
```

char2cond *Simple function to generate REDCap branching logic from character vector*

Description

Simple function to generate REDCap branching logic from character vector

Usage

```
char2cond(
  data,
  minor.split = ",",
  major.split = ";",
  major.sep = " or ",
  .default = NA
)
```

Arguments

data	vector
minor.split	minor split
major.split	major split
major.sep	argument separation. Default is " or ".
.default	default value for missing. Default is NA.

Value

vector

Examples

```
#data <- dd_inst$betingelse
#c("Extubation_novent, 2; Pacu_delay, 1") |> char2cond()
```

---

clean_field_label *Very simple function to remove rich text formatting from field label and save the first paragraph (' ... ').*

Description

Very simple function to remove rich text formatting from field label and save the first paragraph (' ... ').

Usage

```
clean_field_label(data)
```

Arguments

data	field label

Value

character vector

Examples

```
clean_field_label("<div class=\"rich-text-field-label\"><p>Fazekas score</p></div>")
```

clean_redcap_name *clean_redcap_name*

Description

Stepwise removal on non-alphanumeric characters, trailing white space, substitutes spaces for underscores and converts to lower case. Trying to make up for different naming conventions.

Usage

```
clean_redcap_name(x)
```

Arguments

x	vector or data frame for cleaning

Value

vector or data frame, same format as input

Examples

```
"Research!, ne:ws? and c;l-.ls" |> clean_redcap_name()
"8_new_TEST_" |> clean_redcap_name()
```

compact_vec *Compacting a vector of any length with or without names*

Description

Compacting a vector of any length with or without names

Usage

```
compact_vec(data, nm.sep = ": ", val.sep = "; ")
```

Arguments

data	vector, optionally named
nm.sep	string separating name from value if any
val.sep	string separating values

Value

character string

Examples

```
sample(seq_len(4), 20, TRUE) |>
  as_factor() |>
  named_levels() |>
  sort() |>
  compact_vec()
1:6 |> compact_vec()
"test" |> compact_vec()
sample(letters[1:9], 20, TRUE) |> compact_vec()
```

---

create_html_table *Create two-column HTML table for data piping in REDCap instruments*

Description

Create two-column HTML table for data piping in REDCap instruments

Usage

```
create_html_table(text, variable)
```

Arguments

text	descriptive text
variable	variable to pipe

Value

character vector

Examples

```
create_html_table(text = "Patient ID", variable = c("[cpr]"))
create_html_table(text = paste("assessor", 1:2, sep = "_"), variable = c("[cpr]"))
# create_html_table(text = c("CPR nummer","Word"), variable = c("[cpr][1]", "[cpr][2]", "[test]"))
```

`create_instrument_meta` *DEPRECATED Create zips file with necessary content based on data set*

Description

Use export_redcap_instrument() Metadata can be added by editing the data dictionary of a project in the initial design phase. If you want to later add new instruments, this function can be used to create (an) instrument(s) to add to a project in production.

Usage

```
create_instrument_meta(data, dir = here::here(""), record.id = TRUE)
```

Arguments

`data`	metadata for the relevant instrument. Could be from 'ds2dd_detailed()'
`dir`	destination dir for the instrument zip. Default is the current WD.
`record.id`	flag to omit the first row of the data dictionary assuming this is the record_id field which should not be included in the instrument. Default is TRUE.

Value

list

Examples

```
## Not run:
data <- iris |>
  ds2dd_detailed(
    add.auto.id = TRUE,
    form.name = sample(c("b", "c"),
      size = 6,
      replace = TRUE, prob = rep(.5, 2)
    )
  ) |>
  purrr::pluck("meta")
# data |> create_instrument_meta()

data <- iris |>
  ds2dd_detailed(add.auto.id = FALSE) |>
  purrr::pluck("data")
iris |>
  setNames(glue::glue("{sample(x = c('a','b'),size = length(ncol(iris)),
replace=TRUE,prob = rep(x=.5,2))}__{names(iris)}")) |>
  ds2dd_detailed(form.sep = "__")
data |>
```

```
  purrr::pluck("meta") |>
  create_instrument_meta(record.id = FALSE)

## End(Not run)
```

`cut_string_length` *Cut string to desired length*

Description

Cut string to desired length

Usage

```
cut_string_length(data, l = 100)
```

Arguments

`data`	data
`l`	length

Value

character string of length l

Examples

```
"length" |> cut_string_length(l = 3)
```

`d2w` *Convert single digits to words*

Description

Convert single digits to words

Usage

```
d2w(x, lang = "en", neutrum = FALSE, everything = FALSE)
```

Arguments

`x`	data. Handle vectors, data.frames and lists
`lang`	language. Danish (da) and English (en), Default is "en"
`neutrum`	for numbers depending on counted word
`everything`	flag to also split numbers >9 to single digits

Value

returns characters in same format as input

Examples

```
d2w(c(2:8, 21))
d2w(data.frame(2:7, 3:8, 1), lang = "da", neutrum = TRUE)

## If everything=T, also larger numbers are reduced.
## Elements in the list are same length as input
d2w(list(2:8, c(2, 6, 4, 23), 2), everything = TRUE)
```

---

doc2dd *Doc table to data dictionary - EARLY, DOCS MISSING*

Description

Works well with 'project.aid::docx2list()'. Allows defining a database in a text document (see provided template) for an easier to use data base creation. This approach allows easier collaboration when defining the database. The generic case is a data frame with variable names as values in a column. This is a format like the REDCap data dictionary, but gives a few options for formatting.

Usage

```
doc2dd(
  data,
  instrument.name,
  col.variables = 1,
 list.datetime.format = list(date_dmy = "_dat[eo]$", time_hh_mm_ss = "_ti[md]e?$"),
  col.description = NULL,
  col.condition = NULL,
  col.subheader = NULL,
  subheader.tag = "h2",
  condition.minor.sep = ",",
  condition.major.sep = ";",
  col.calculation = NULL,
  col.choices = NULL,
  choices.char.sep = "/",
  missing.default = NA
)
```

Arguments

data	tibble or data.frame with all variable names in one column
instrument.name	character vector length one. Instrument name.

`col.variables` variable names column (default = 1), allows dplyr subsetting

`list.datetime.format`
formatting for date/time detection. See ‘case_match_regex_list()‘

`col.description`
descriptions column, allows dplyr subsetting. If empty, variable names will be used.

`col.condition` conditions for branching column, allows dplyr subsetting. See ‘char2cond()‘.

`col.subheader` sub-header column, allows dplyr subsetting. See ‘format_subheader()‘.

`subheader.tag` formatting tag. Default is "h2"

`condition.minor.sep`
condition split minor. See ‘char2cond()‘. Default is ",".

`condition.major.sep`
condition split major. See ‘char2cond()‘. Default is ";".

`col.calculation`
calculations column. Has to be written exact. Character vector.

`col.choices` choices column. See ‘char2choice()‘.

`choices.char.sep`
choices split. See ‘char2choice()‘. Default is "/".

`missing.default`
value for missing fields. Default is NA.

Value

tibble or data.frame (same as data)

Examples

```
# data <- dd_inst
# data |> doc2dd(instrument.name = "evt",
# col.description = 3,
# col.condition = 4,
# col.subheader = 2,
# col.calculation = 5,
# col.choices = 6)
```

`ds2dd` *(DEPRECATED) Data set to data dictionary function*

Description

Creates a very basic data dictionary skeleton. Please see ‘ds2dd_detailed()‘ for a more advanced function.

Usage

```
ds2dd(
  ds,
  record.id = "record_id",
  form.name = "basis",
  field.type = "text",
  field.label = NULL,
  include.column.names = FALSE,
  metadata = names(REDCapCAST::redcapcast_meta)
)
```

Arguments

ds	data set
record.id	name or column number of id variable, moved to first row of data dictionary, character of integer. Default is "record_id".
form.name	vector of form names, character string, length 1 or length equal to number of variables. Default is "basis".
field.type	vector of field types, character string, length 1 or length equal to number of variables. Default is "text.
field.label	vector of form names, character string, length 1 or length equal to number of variables. Default is NULL and is then identical to field names.
include.column.names	Flag to give detailed output including new column names for original data set for upload.
metadata	Metadata column names. Default is the included names(REDCapCAST::redcapcast_meta).

Details

Migrated from stRoke ds2dd(). Fits better with the functionality of 'REDCapCAST'.

Value

data.frame or list of data.frame and vector

Examples

```
redcapcast_data$record_id <- seq_len(nrow(redcapcast_data))
ds2dd(redcapcast_data, include.column.names = TRUE)
```

ds2dd_detailed *Extract data from stata file for data dictionary*

Description

Extract data from stata file for data dictionary

Usage

```
ds2dd_detailed(
  data,
  add.auto.id = FALSE,
  date.format = "dmy",
  form.name = NULL,
  form.sep = NULL,
  form.prefix = TRUE,
  field.type = NULL,
  field.label = NULL,
  field.label.attr = "label",
  field.validation = NULL,
  metadata = names(REDCapCAST::redcapcast_meta),
  convert.logicals = TRUE
)
```

Arguments

data	data frame
add.auto.id	flag to add id column
date.format	date format, character string. ymd/dmy/mdy. dafault is dmy.
form.name	manually specify form name(s). Vector of length 1 or ncol(data). Default is NULL and "data" is used.
form.sep	If supplied dataset has form names as suffix or prefix to the column/variable names, the seperator can be specified. If supplied, the form.name is ignored. Default is NULL.
form.prefix	Flag to set if form is prefix (TRUE) or suffix (FALSE) to the column names. Assumes all columns have pre- or suffix if specified.
field.type	manually specify field type(s). Vector of length 1 or ncol(data). Default is NULL and "text" is used for everything but factors, which wil get "radio".
field.label	manually specify field label(s). Vector of length 1 or ncol(data). Default is NULL and colnames(data) is used or attribute ‘field.label.attr‘ for haven_labelled data set (imported .dta file with ‘haven::read_dta()‘).
field.label.attr	attribute name for named labels for haven_labelled data set (imported .dta file with ‘haven::read_dta()‘. Default is "label".

`field.validation`
: manually specify field validation(s). Vector of length 1 or ncol(data). Default is NULL and ‘levels()‘ are used for factors or attribute ‘factor.labels.attr‘ for haven_labelled data set (imported .dta file with ‘haven::read_dta()‘).

`metadata`
: redcap metadata headings. Default is names(REDCapCAST::redcapcast_meta).

`convert.logicals`
: convert logicals to factor. Default is TRUE.

Details

This function is a natural development of the ds2dd() function. It assumes that the first column is the ID-column. No checks. Please, do always inspect the data dictionary before upload.

Ensure, that the data set is formatted with as much information as possible.

‘field.type‘ can be supplied

Value

list of length 2

Examples

```
## Basic parsing with default options
requireNamespace("REDCapCAST")
redcapcast_data |>
  dplyr::select(-dplyr::starts_with("redcap_")) |>
  ds2dd_detailed()

## Adding a record_id field
iris |> ds2dd_detailed(add.auto.id = TRUE)

## Passing form name information to function
iris |>
  ds2dd_detailed(
    add.auto.id = TRUE,
    form.name = sample(c("b", "c"), size = 6, replace = TRUE, prob = rep(.5, 2))
  ) |>
  purrr::pluck("meta")
mtcars |>
  dplyr::mutate(unknown = NA) |>
  numchar2fct() |>
  ds2dd_detailed(add.auto.id = TRUE)

## Using column name suffix to carry form name
data <- iris |>
  ds2dd_detailed(add.auto.id = TRUE) |>
  purrr::pluck("data")
names(data) <- glue::glue("{sample(x = c('a','b'),size = length(names(data)),
replace=TRUE,prob = rep(x=.5,2))}__{names(data)}")
data |> ds2dd_detailed(form.sep = "__")
```

easy_redcap *Secure API key storage and data acquisition in one*

Description

Secure API key storage and data acquisition in one

Usage

```
easy_redcap(
  project.name,
  uri,
  raw_or_label = "both",
  data_format = c("wide", "list", "redcap", "long"),
  widen.data = NULL,
  ...
)
```

Arguments

project.name	The name of the current project (for key storage with key_set, using the default keyring)
uri	REDCap database API uri
raw_or_label	argument passed on to read_redcap_tables. Default is "both" to get labelled data.
data_format	Choose the data
widen.data	argument to widen the exported data. [DEPRECATED], use ‘data_format‘instead
...	arguments passed on to read_redcap_tables.

Value

data.frame or list depending on widen.data

Examples

```
## Not run:
easy_redcap("My_new_project", fields = c("record_id", "age", "hypertension"))

## End(Not run)
```

`export_redcap_instrument` *Creates zip-file with necessary content to manually add instrument to database*

Description

Metadata can be added by editing the data dictionary of a project in the initial design phase. If you want to later add new instruments, this function can be used to create (an) instrument(s) to add to a project in production.

Usage

```
export_redcap_instrument(
  data,
  file = here::here(paste0("instrument_", Sys.Date(), ".zip")),
  force = FALSE,
  record.id = "record_id"
)
```

Arguments

`data`	metadata for the relevant instrument. Could be from ‘ds2dd_detailed()‘
`file`	destination file name.
`force`	force instrument creation and ignore different form names by just using the first.
`record.id`	record id variable name. Default is ’record_id’.

Value

exports zip-file

Examples

```
# iris |>
#  ds2dd_detailed(
#    add.auto.id = TRUE,
#    form.name = sample(c("b", "c"), size = 6, replace = TRUE, prob = rep(.5, 2))
#  ) |>
#  purrr::pluck("meta") |>
#  (\(.x){
#  split(.x, .x$form_name)
#  })() |>
#  purrr::imap(function(.x, .i){
#  export_redcap_instrument(.x,file=here::here(paste0(.i,Sys.Date(),".zip")))
#  })

# iris |>
#  ds2dd_detailed(
```

```
#    add.auto.id = TRUE
#  ) |>
#  purrr::pluck("meta") |>
#  export_redcap_instrument(file=here::here(paste0("instrument",Sys.Date(),".zip")))
```

`fct_drop` *Drop unused levels preserving label data*

Description

This extends [forcats::fct_drop()] to natively work across a data.frame and replaces [base::droplevels()].

Usage

```
fct_drop(x, ...)

## S3 method for class 'data.frame'
fct_drop(x, ...)

## S3 method for class 'factor'
fct_drop(x, ...)
```

Arguments

`x`	Factor to drop unused levels
`...`	Other arguments passed down to method.

Examples

```
mtcars |>
  numchar2fct() |>
  fct_drop()
mtcars |>
  numchar2fct() |>
  dplyr::mutate(vs = fct_drop(vs))
```

`fct2num` *Allows conversion of factor to numeric values preserving original levels*

Description

Allows conversion of factor to numeric values preserving original levels

Usage

```
fct2num(data)
```

Arguments

data	vector

Value

numeric vector

Examples

```
c(1, 4, 3, "A", 7, 8, 1) |>
  as_factor() |>
  fct2num()

structure(c(1, 2, 3, 2, 10, 9),
  labels = c(Unknown = 9, Refused = 10),
  class = "haven_labelled"
) |>
  as_factor() |>
  fct2num()

structure(c(1, 2, 3, 2, 10, 9),
  labels = c(Unknown = 9, Refused = 10),
  class = "labelled"
) |>
  as_factor() |>
  fct2num()

structure(c(1, 2, 3, 2, 10, 9),
  labels = c(Unknown = 9, Refused = 10)
) |>
  as_factor() |>
  fct2num()
```

---

file_extension *DEPRECATED Helper to import files correctly*

Description

DEPRECATED Helper to import files correctly

Usage

```
file_extension(filenames)
```

Arguments

filenames	file names

Value

character vector

Examples

```
file_extension(list.files(here::here(""))[[2]])[[1]]
file_extension(c("file.cd..ks", "file"))
```

---

focused_metadata *focused_metadata*

Description

Extracts limited metadata for variables in a dataset

Usage

```
focused_metadata(metadata, vars_in_data)
```

Arguments

metadata	A dataframe containing metadata
vars_in_data	Vector of variable names in the dataset

Value

A dataframe containing metadata for the variables in the dataset

---

format_redcap_factor *Converts REDCap choices to factor levels and stores in labels attribute*

Description

Applying as_factor to the data.frame or variable, will coerce to a factor.

Usage

```
format_redcap_factor(data, meta)
```

Arguments

data	vector
meta	vector of REDCap choices

Value

vector of class "labelled" with a "labels" attribute

Examples

```
format_redcap_factor(sample(1:3, 20, TRUE), "1, First. | 2, second | 3, THIRD")
```

`format_subheader` *Sub-header formatting wrapper*

Description

Sub-header formatting wrapper

Usage

```
format_subheader(data, tag = "h2")
```

Arguments

`data`	character vector
`tag`	character vector length 1

Value

character vector

Examples

```
"Instrument header" |> format_subheader()
```

`get_api_key` *Retrieve project API key if stored, if not, set and retrieve*

Description

Attempting to make secure API key storage so simple, that no other way makes sense. Wrapping key_get and key_set using the key_list to check if key is in storage already.

Usage

```
get_api_key(key.name, ...)
```

Arguments

key.name	character vector of key name
...	passed to key_set

Value

character vector

---

get_attr *Extract attribute. Returns NA if none*

Description

Extract attribute. Returns NA if none

Usage

```
get_attr(data, attr = NULL)
```

Arguments

data	vector
attr	attribute name

Value

character vector

Examples

```
attr(mtcars$mpg, "label") <- "testing"
do.call(c, sapply(mtcars, get_attr))
## Not run:
mtcars |>
  numchar2fct(numeric.threshold = 6) |>
  ds2dd_detailed()

## End(Not run)
```

get_id_name *Get the id name*

Description

Get the id name

Usage

```
get_id_name(data)
```

Arguments

data	data frame or list

Value

character vector

guess_time_only *Guess time variables based on naming pattern*

Description

This is for repairing data with time variables with appended "1970-01-01"

Usage

```
guess_time_only(
  data,
  validate.time = FALSE,
  time.var.sel.pos = "[Tt]i[d(me)]",
  time.var.sel.neg = "[Dd]at[eo]"
)
```

Arguments

data	data.frame or tibble
validate.time	Flag to validate guessed time columns
time.var.sel.pos	Positive selection regex string passed to ‘gues_time_only_filter()‘ as sel.pos.
time.var.sel.neg	Negative selection regex string passed to ‘gues_time_only_filter()‘ as sel.neg.

Value

data.frame or tibble

Examples

```
redcapcast_data |> guess_time_only(validate.time = TRUE)
```

---

guess_time_only_filter

Try at determining which are true time only variables

Description

This is just a try at guessing data type based on data class and column names hoping for a tiny bit of naming consistency. R does not include a time-only data format natively, so the "hms" class from ‘readr‘ is used. This has to be converted to character class before REDCap upload.

Usage

```
guess_time_only_filter(
  data,
  validate = FALSE,
  sel.pos = "[Tt]i[d(me)]",
  sel.neg = "[Dd]at[eo]"
)
```

Arguments

data	data set
validate	flag to output validation data. Will output list.
sel.pos	Positive selection regex string
sel.neg	Negative selection regex string

Value

character vector or list depending on ‘validate‘ flag.

Examples

```
data <- redcapcast_data
data |> guess_time_only_filter()
data |>
  guess_time_only_filter(validate = TRUE) |>
  lapply(head)
```

haven_all_levels *Finish incomplete haven attributes substituting missings with values*

Description

Finish incomplete haven attributes substituting missings with values

Usage

```
haven_all_levels(data)
```

Arguments

data	haven labelled variable

Value

named vector

Examples

```
ds <- structure(c(1, 2, 3, 2, 10, 9),
  labels = c(Unknown = 9, Refused = 10),
  class = "haven_labelled"
)
haven::is.labelled(ds)
attributes(ds)
ds |> haven_all_levels()
```

hms2character *Change "hms" to "character" for REDCap upload.*

Description

Change "hms" to "character" for REDCap upload.

Usage

```
hms2character(data)
```

Arguments

data	data set

Value

data.frame or tibble

Examples

```
data <- redcapcast_data
## data |> time_only_correction() |> hms2character()
```

---

html_tag_wrap *Simple html tag wrapping for REDCap text formatting*

Description

Simple html tag wrapping for REDCap text formatting

Usage

```
html_tag_wrap(data, tag = "h2", extra = NULL)
```

Arguments

data	character vector
tag	character vector length 1
extra	character vector

Value

character vector

Examples

```
html_tag_wrap("Titel", tag = "div", extra = 'class="rich-text-field-label"')
html_tag_wrap("Titel", tag = "h2")
```

---

is.labelled *Tests for multiple label classes*

Description

Tests for multiple label classes

Usage

```
is.labelled(x, classes = c("haven_labelled", "labelled"))
```

Arguments

x	data
classes	classes to test

Value

logical

Examples

```
structure(c(1, 2, 3, 2, 10, 9),
  labels = c(Unknown = 9, Refused = 10),
  class = "haven_labelled"
) |> is.labelled()
```

`is_missing` *Multi missing check*

Description

Multi missing check

Usage

```
is_missing(data, nas = c("", "NA"))
```

Arguments

`data`	character vector
`nas`	character vector of strings considered as NA

Value

logical vector

`is_repeated_longitudinal` *Test if repeatable or longitudinal*

Description

Test if repeatable or longitudinal

Usage

```
is_repeated_longitudinal(
  data,
 generics = c("redcap_event_name", "redcap_repeat_instrument", "redcap_repeat_instance")
)
```

Arguments

data	data set
generics	default is "redcap_event_name", "redcap_repeat_instrument" and "redcap_repeat_instance"

Value

logical

Examples

```
is_repeated_longitudinal(c("record_id", "age", "record_id", "gender"))
is_repeated_longitudinal(redcapcast_data)
is_repeated_longitudinal(list(redcapcast_data))
```

---

mark_complete *Completion marking based on completed upload*

Description

Completion marking based on completed upload

Usage

```
mark_complete(upload, ls)
```

Arguments

upload	output list from ‘REDCapR::redcap_write()‘
ls	output list from ‘ds2dd_detailed()‘

Value

list with ‘REDCapR::redcap_write()‘ results

match_fields_to_form *Match fields to forms*

Description

Match fields to forms

Usage

```
match_fields_to_form(metadata, vars_in_data)
```

Arguments

metadata	A data frame containing field names and form names
vars_in_data	A character vector of variable names

Value

A data frame containing field names and form names

mtcars_redcap *mtcars dataset slightly modified to use for Shiny app upload demonstration*

Description

mtcars dataset slightly modified to use for Shiny app upload demonstration

Usage

```
data(mtcars_redcap)
```

Format

A data frame with 13 variables:

record_id ID, numeric

mpg ID, numeric

cyl ID, numeric

disp ID, numeric

hp ID, numeric

drat ID, numeric

wt ID, numeric

qsec ID, numeric

vs ID, numeric

am ID, numeric

gear ID, numeric

carb ID, numeric

name original rownames, charater

`named_levels` *Get named vector of factor levels and values*

Description

Get named vector of factor levels and values

Usage

```
named_levels(
  data,
  label = "labels",
  na.label = NULL,
  na.value = 99,
  sort.numeric = TRUE
)
```

Arguments

`data`	factor
`label`	character string of attribute with named vector of factor labels
`na.label`	character string to refactor NA values. Default is NULL.
`na.value`	new value for NA strings. Ignored if na.label is NULL. Default is 99.
`sort.numeric`	sort factor levels if levels are numeric. Default is TRUE

Value

named vector

Examples

```
structure(c(1, 2, 3, 2, 10, 9),
  labels = c(Unknown = 9, Refused = 10),
  class = "haven_labelled"
) |>
  as_factor() |>
  named_levels()
structure(c(1, 2, 3, 2, 10, 9),
  labels = c(Unknown = 9, Refused = 10),
  class = "labelled"
```

```
) |>
  as_factor() |>
  named_levels()
```

---

nav_bar_page *Nav_bar defining function for shiny ui*

Description

Nav_bar defining function for shiny ui

Usage

```
nav_bar_page()
```

Value

shiny object

---

numchar2fct *Applying var2fct across data set*

Description

Individual thresholds for character and numeric columns

Usage

```
numchar2fct(data, numeric.threshold = 6, character.throshold = 6)
```

Arguments

data	dataset. data.frame or tibble
numeric.threshold	threshold for var2fct for numeric columns. Default is 6.
character.throshold	threshold for var2fct for character columns. Default is 6.

Value

data.frame or tibble

Examples

```
mtcars |> str()
## Not run:
mtcars |>
  numchar2fct(numeric.threshold = 6) |>
  str()

## End(Not run)
```

`parse_data` *Helper to auto-parse un-formatted data with haven and readr*

Description

Helper to auto-parse un-formatted data with haven and readr

Usage

```
parse_data(
  data,
  guess_type = TRUE,
  col_types = NULL,
  locale = readr::default_locale(),
  ignore.vars = "cpr",
  ...
)
```

Arguments

`data`	data.frame or tibble
`guess_type`	logical to guess type with readr
`col_types`	specify col_types using readr semantics. Ignored if guess_type is TRUE
`locale`	option to specify locale. Defaults to readr::default_locale().
`ignore.vars`	specify column names of columns to ignore when parsing
`...`	ignored

Value

data.frame or tibble

Examples

```
mtcars |>
  parse_data() |>
  str()
```

possibly_numeric *Tests if vector can be interpreted as numeric without introducing NAs by coercion*

Description

Tests if vector can be interpreted as numeric without introducing NAs by coercion

Usage

```
possibly_numeric(data)
```

Arguments

data	vector

Value

logical

Examples

```
c("1","5") |> possibly_numeric()
c("1","5","e") |> possibly_numeric()
```

possibly_roman *Test if vector can be interpreted as roman numerals*

Description

Test if vector can be interpreted as roman numerals

Usage

```
possibly_roman(data)
```

Arguments

data	character vector

Value

logical

Examples

```
sample(1:100, 10) |>
  as.roman() |>
  possibly_roman()
sample(c(TRUE, FALSE), 10, TRUE) |> possibly_roman()
rep(NA, 10) |> possibly_roman()
```

`process_user_input` *User input processing*

Description

User input processing

Usage

```
process_user_input(x)
```

Arguments

x	input

Value

processed input

`process_user_input.character` *User input processing character*

Description

User input processing character

Usage

```
## S3 method for class 'character'
process_user_input(x, ...)
```

Arguments

x	input
...	ignored

Value

processed input

process_user_input.data.frame *User input processing data.frame*

Description

User input processing data.frame

Usage

```
## S3 method for class 'data.frame'
process_user_input(x, ...)
```

Arguments

x	input
...	ignored

Value

processed input

process_user_input.default *User input processing default*

Description

User input processing default

Usage

```
## Default S3 method:
process_user_input(x, ...)
```

Arguments

x	input
...	ignored

Value

processed input

process_user_input.response *User input processing response*

Description

User input processing response

Usage

```
## S3 method for class 'response'
process_user_input(x, ...)
```

Arguments

x	input
...	ignored

Value

processed input

read_input *Flexible file import based on extension*

Description

Flexible file import based on extension

Usage

```
read_input(file, consider.na = c("NA", "\"\"", ""), sheet = 1, ...)
```

Arguments

file	file name
consider.na	character vector of strings to consider as NAs
sheet	sheet number
...	magic dots to allow overflow. Ignored for now.

Value

tibble

Examples

```
read_input("https://raw.githubusercontent.com/agdamsbo/cognitive.index.lookup/main/data/sample.csv")
```

`read_redcap_instrument` *Convenience function to download complete instrument, using token storage in keyring.*

Description

Convenience function to download complete instrument, using token storage in keyring.

Usage

```
read_redcap_instrument(
  key,
  uri,
  instrument,
  raw_or_label = "raw",
  id_name = "record_id",
  records = NULL
)
```

Arguments

`key`	key name in standard keyring for token retrieval.
`uri`	REDCap database API uri
`instrument`	instrument name
`raw_or_label`	raw or label passed to 'REDCapR::redcap_read()'
`id_name`	id variable name. Default is "record_id".
`records`	specify the records to download. Index numbers. Numeric vector.

Value

data.frame

`read_redcap_tables` *Download REDCap data*

Description

Implementation of passed on to REDCap_split with a focused data acquisition approach using passed on to redcap_read and only downloading specified fields, forms and/or events using the built-in focused_metadata including some clean-up. Works with classical and longitudinal projects with or without repeating instruments. Will preserve metadata in the data.frames as labels.

Usage

```
read_redcap_tables(
  uri,
  token,
  records = NULL,
  fields = NULL,
  events = NULL,
  forms = NULL,
  raw_or_label = c("raw", "label", "both"),
  split_forms = c("all", "repeating", "none"),
  ...
)
```

Arguments

uri	REDCap database API uri
token	API token
records	records to download
fields	fields to download
events	events to download
forms	forms to download
raw_or_label	raw or label tags. Can be "raw", "label" or "both". * "raw": Standard redcap_read method to get raw values. * "label": Standard redcap_read method to get label values. * "both": Get raw values with REDCap labels applied as labels. Use as_factor to format factors with original labels and use the ‘gtsummary‘ package functions like tbl_summary to easily get beautiful tables with original labels from REDCap. Use fct_drop to drop empty levels.
split_forms	Whether to split "repeating" or "all" forms, default is all. Give "none" to export native semi-long REDCap format
...	passed on to redcap_read

Value

list of instruments

Examples

```
# Examples will be provided later
```

REDCap_split *Split REDCap repeating instruments table into multiple tables*

Description

This will take output from a REDCap export and split it into a base table and child tables for each repeating instrument. Metadata is used to determine which fields should be included in each resultant table.

Usage

```
REDCap_split(
  records,
  metadata,
  primary_table_name = "",
  forms = c("repeating", "all")
)
```

Arguments

records	Exported project records. May be a data.frame, response, or character vector containing JSON from an API call.
metadata	Project metadata (the data dictionary). May be a data.frame, response, or character vector containing JSON from an API call.
primary_table_name	Name given to the list element for the primary output table. Ignored if forms = 'all'.
forms	Indicate whether to create separate tables for repeating instruments only or for all forms.

Value

A list of "data.frame"s. The number of tables will differ depending on the forms option selected.

- 'repeating': one base table and one or more tables for each repeating instrument.
- 'all': a data.frame for each instrument, regardless of whether it is a repeating instrument or not.

Author(s)

Paul W. Egeler

Examples

```
## Not run:
# Using an API call ----------------------------------------------------

library(RCurl)

# Get the records
records <- postForm(
  uri = api_url, # Supply your site-specific URI
  token = api_token, # Supply your own API token
  content = "record",
  format = "json",
  returnFormat = "json"
)

# Get the metadata
metadata <- postForm(
  uri = api_url, # Supply your site-specific URI
  token = api_token, # Supply your own API token
  content = "metadata",
  format = "json"
)

# Convert exported JSON strings into a list of data.frames
REDCapCAST::REDCap_split(records, metadata)

# Using a raw data export ----------------------------------------------

# Get the records
records <- read.csv("/path/to/data/ExampleProject_DATA_2018-06-03_1700.csv")

# Get the metadata
metadata <- read.csv(
  "/path/to/data/ExampleProject_DataDictionary_2018-06-03.csv"
)

# Split the tables
REDCapCAST::REDCap_split(records, metadata)

# In conjunction with the R export script ------------------------------

# You must set the working directory first since the REDCap data export
# script contains relative file references.
old <- getwd()
setwd("/path/to/data/")

# Run the data export script supplied by REDCap.
# This will create a data.frame of your records called 'data'
source("ExampleProject_R_2018-06-03_1700.r")

# Get the metadatan
metadata <- read.csv("ExampleProject_DataDictionary_2018-06-03.csv")
```

```
# Split the tables
REDCapCAST::REDCap_split(data, metadata)
setwd(old)

## End(Not run)
```

---

redcap_wider *Transforms list of REDCap data.frames to a single wide data.frame*

Description

Converts a list of REDCap data.frames from long to wide format. In essence it is a wrapper for the pivot_wider function applied on a REDCap output (from read_redcap_tables) or manually split by REDCap_split.

Usage

```
redcap_wider(
  data,
  event.glue = "{.value}____{redcap_event_name}",
  inst.glue = "{.value}____{redcap_repeat_instance}"
)
```

Arguments

data	A list of data frames
event.glue	A glue string for repeated events naming
inst.glue	A glue string for repeated instruments naming

Value

data.frame in wide format

Examples

```
# Longitudinal
list1 <- list(
  data.frame(
    record_id = c(1, 2, 1, 2),
    redcap_event_name = c("baseline", "baseline", "followup", "followup"),
    age = c(25, 26, 27, 28)
  ),
  data.frame(
    record_id = c(1, 2),
    redcap_event_name = c("baseline", "baseline"),
    gender = c("male", "female")
  )
```

```
)
redcap_wider(list1)
# Simpel with two instruments
list2 <- list(
  data.frame(
    record_id = c(1, 2),
    age = c(25, 26)
  ),
  data.frame(
    record_id = c(1, 2),
    gender = c("male", "female")
  )
)
redcap_wider(list2)
# Simple with single instrument
list3 <- list(data.frame(
  record_id = c(1, 2),
  age = c(25, 26)
))
redcap_wider(list3)
# Longitudinal with repeatable instruments
list4 <- list(
  data.frame(
    record_id = c(1, 2, 1, 2),
    redcap_event_name = c("baseline", "baseline", "followup", "followup"),
    age = c(25, 26, 27, 28)
  ),
  data.frame(
    record_id = c(1, 1, 1, 1, 2, 2, 2, 2),
    redcap_event_name = c(
      "baseline", "baseline", "followup", "followup",
      "baseline", "baseline", "followup", "followup"
    ),
    redcap_repeat_instrument = "walk",
    redcap_repeat_instance = c(1, 2, 1, 2, 1, 2, 1, 2),
    dist = c(40, 32, 25, 33, 28, 24, 23, 36)
  ),
  data.frame(
    record_id = c(1, 2),
    redcap_event_name = c("baseline", "baseline"),
    gender = c("male", "female")
  )
)
redcap_wider(list4)

list5 <- list(
  data.frame(
    record_id = c(1, 2, 1, 2),
    redcap_event_name = c("baseline", "baseline", "followup", "followup")
  ),
  data.frame(
    record_id = c(1, 1, 1, 1, 2, 2, 2, 2),
    redcap_event_name = c(
```

```
      "baseline", "baseline", "followup", "followup",
      "baseline", "baseline", "followup", "followup"
    ),
    redcap_repeat_instrument = "walk",
    redcap_repeat_instance = c(1, 2, 1, 2, 1, 2, 1, 2),
    dist = c(40, 32, 25, 33, 28, 24, 23, 36)
  ),
  data.frame(
    record_id = c(1, 2),
    redcap_event_name = c("baseline", "baseline"),
    gender = c("male", "female")
  )
)
redcap_wider(list5)
```

`redcapcast_data` *Data set for demonstration*

Description

This is a small dataset from a REDCap database for demonstrational purposes. Contains only synthetic data.

Usage

```
data(redcapcast_data)
```

Format

A data frame with 22 variables:

record_id ID, numeric

redcap_event_name Event name, character

redcap_repeat_instrument Repeat instrument, character

redcap_repeat_instance Repeat instance, numeric

cpr CPR number, character

inclusion Inclusion date, Date

inclusion_time Inclusion time, hms

dob Date of birth, Date

age Age decimal, numeric

age_integer Age integer, numeric

sex Legal sex, character

cohabitation Cohabitation status, character

con_calc con_calc

con_mrs con_mrs

consensus_complete consensus_complete

hypertension Hypertension, character

diabetes diabetes, character

region region, character

baseline_data_start_complete Completed, character

mrs_assessed mRS Assessed, character

mrs_date Assesment date, Date

mrs_score Categorical score, numeric

mrs_complete Complete, numeric

event_datetime Event datetime, POSIXct

event_age Age at time of event, numeric

event_type Event type, character

new_event_complete Completed, character

`redcapcast_meta` *REDCap metadata from data base*

Description

This metadata dataset from a REDCap database is for demonstration purposes.

Usage

```
data(redcapcast_meta)
```

Format

A data frame with 22 variables:

field_name field_name, character

form_name form_name, character

section_header section_header, character

field_type field_type, character

field_label field_label, character

select_choices_or_calculations select_choices_or_calculations, character

field_note field_note, character

text_validation_type_or_show_slider_number text_validation_type_or_show_slider_number, character

text_validation_min text_validation_min, character

text_validation_max text_validation_max, character

identifier identifier, character

branching_logic branching_logic, character

required_field required_field, character

custom_alignment custom_alignment, character

question_number question_number, character

matrix_group_name matrix_group_name, character

matrix_ranking matrix_ranking, character

field_annotation field_annotation, character

`replace_curly_quote` *Replace curly apostrophes and quotes from word*

Description

Copied from textclean, which has not been updated since 2018 and is not on CRAN. Github:https://github.com/trinker/textclea

Usage

```
replace_curly_quote(x)
```

Arguments

x	character vector

Value

character vector

`safe_topological_sort` *Safe Wrapper for Topological Sort*

Description

Calls `topological_sort` and falls back to elements in order of first appearance if a cycle makes a valid ordering impossible. Guaranteed to always return a character vector of all unique elements.

Usage

```
safe_topological_sort(
  ordered_list,
  priority = c("none", "lexicographic", "least_constrained", "most_constrained")
)
```

Arguments

ordered_list	A list of character vectors, each specifying a partial ordering of elements. Consecutive elements within a vector imply a precedence constraint: earlier elements must appear before later ones in the output. Elements may appear across multiple vectors; constraints from all vectors are combined. Duplicate constraints are silently ignored.
priority	A string controlling how ties are broken when multiple elements are simultaneously eligible for placement. One of: "none" Default. Eligible elements are processed in the order they are first encountered in ordered_list. Fully deterministic with no resorting overhead. "lexicographic" Eligible elements are sorted alphabetically by name. Useful for reproducible output independent of input order. "least_constrained" Eligible elements with the fewest total edges (in-degree + out-degree) are placed first. Elements that participate in fewer ordering relationships surface earlier. "most_constrained" Eligible elements with the most total edges (in-degree + out-degree) are placed first. Elements that participate in more ordering relationships surface earlier.

Value

A character vector of all unique elements from ordered_list. If constraints are consistent, the order satisfies all precedence constraints as per topological_sort. If a cycle is detected, elements are returned in order of first appearance in ordered_list.

See Also

topological_sort

---

sanitize_split *Sanitize list of data frames*

Description

Removing empty rows

Usage

```
sanitize_split(
  l,
  generic.names = c("redcap_event_name", "redcap_repeat_instrument",
    "redcap_repeat_instance"),
  drop.complete = TRUE,
  drop.empty = TRUE
)
```

Arguments

l	A list of data frames.
generic.names	A vector of generic names to be excluded.
drop.complete	logical to remove generic REDCap variables indicating instrument completion. Default is TRUE.
drop.empty	logical to remove variables with only NAs Default is TRUE.

Value

A list of data frames with generic names excluded.

---

set_attr *Set attributes for named attribute. Appends if attr is NULL*

Description

Set attributes for named attribute. Appends if attr is NULL

Usage

```
set_attr(data, label, attr = NULL, overwrite = FALSE)
```

Arguments

data	vector
label	label
attr	attribute name
overwrite	overwrite existing attributes. Default is FALSE.

Value

vector with attribute

shiny_cast *Launch the included Shiny-app for database casting and upload*

Description

Wraps shiny::runApp()

Usage

```
shiny_cast(...)
```

Arguments

...	Arguments passed to shiny::runApp()

Value

shiny app

Examples

```
# shiny_cast()
```

split_non_repeating_forms *Split a data frame into separate tables for each form*

Description

Split a data frame into separate tables for each form

Usage

```
split_non_repeating_forms(table, universal_fields, fields)
```

Arguments

table	A data frame
universal_fields	A character vector of fields that should be included in every table
fields	A two-column matrix containing the names of fields that should be included in each form

Value

A list of data frames, one for each non-repeating form

Examples

```
# Create a table
table <- data.frame(
  id = c(1, 2, 3, 4, 5),
  form_a_name = c("John", "Alice", "Bob", "Eve", "Mallory"),
  form_a_age = c(25, 30, 25, 15, 20),
  form_b_name = c("John", "Alice", "Bob", "Eve", "Mallory"),
  form_b_gender = c("M", "F", "M", "F", "F")
)

# Create the universal fields
universal_fields <- c("id")

# Create the fields
fields <- matrix(
  c(
    "form_a_name", "form_a",
    "form_a_age", "form_a",
    "form_b_name", "form_b",
    "form_b_gender", "form_b"
  ),
  ncol = 2, byrow = TRUE
)

# Split the table
split_non_repeating_forms(table, universal_fields, fields)
```

---

strsplitx *Extended string splitting*

Description

Can be used as a substitute of the base function. Main claim to fame is easing the split around the defined delimiter, see example.

Usage

```
strsplitx(x, split, type = "classic", perl = FALSE, ...)
```

Arguments

x	data
split	delimiter
type	Split type. Can be c("classic", "before", "after", "around")
perl	perl param from strsplit()
...	additional parameters are passed to base strsplit handling splits

Value

list

Examples

```
test <- c(
  "12 months follow-up", "3 steps", "mRS 6 weeks",
  "Counting to 231 now"
)
strsplitx(test, "[0-9]", type = "around")
```

`suffix2label` *Transfer variable name suffix to label in widened data*

Description

Transfer variable name suffix to label in widened data

Usage

```
suffix2label(
  data,
  suffix.sep = "____",
  attr = "label",
  glue.str = "{label} ({paste(suffixes,collapse=', ')})"
)
```

Arguments

`data`	data.frame
`suffix.sep`	string to split suffix(es). Passed to strsplit
`attr`	label attribute. Default is "label"
`glue.str`	glue string for new label. Available variables are "label" and "suffixes"

Value

data.frame

time_only_correction *Correction based on time_only_filter function*

Description

Correction based on time_only_filter function

Usage

```
time_only_correction(data, ...)
```

Arguments

data	data set
...	arguments passed on to ‘guess_time_only_filter()‘

Value

tibble

Examples

```
data <- redcapcast_data
## data |> time_only_correction()
```

topological_sort *Topological Sort of Ordered Element Lists*

Description

Derives a single consistent ordering of all elements from a list of partially ordered vectors. Each vector encodes pairwise precedence constraints between consecutive elements: if element A appears before element B in any input vector, A is guaranteed to precede B in the output. The overall ordering is resolved via Kahn's algorithm.

Usage

```
topological_sort(
  ordered_list,
  priority = c("none", "lexicographic", "least_constrained", "most_constrained")
)
```

Arguments

`ordered_list`	A list of character vectors, each specifying a partial ordering of elements. Consecutive elements within a vector imply a precedence constraint: earlier elements must appear before later ones in the output. Elements may appear across multiple vectors; constraints from all vectors are combined. Duplicate constraints are silently ignored.
`priority`	A string controlling how ties are broken when multiple elements are simultaneously eligible for placement. One of:

`"none"` Default. Eligible elements are processed in the order they are first encountered in `ordered_list`. Fully deterministic with no resorting overhead.

`"lexicographic"` Eligible elements are sorted alphabetically by name. Useful for reproducible output independent of input order.

`"least_constrained"` Eligible elements with the fewest total edges (in-degree + out-degree) are placed first. Elements that participate in fewer ordering relationships surface earlier.

`"most_constrained"` Eligible elements with the most total edges (in-degree + out-degree) are placed first. Elements that participate in more ordering relationships surface earlier.

Details

Precedence constraints are extracted from consecutive element pairs within each vector and deduplicated before processing. The algorithm runs in O(V + E) time for `priority = "none"`, where V is the number of unique elements and E the number of unique constraints. Priority modes incur an additional O(V log V) cost from sorting at each enqueue step.

Elements that appear in only one vector and have no constraints linking them to other elements except through a single predecessor or successor are considered loosely constrained. Their placement among other eligible elements is undefined beyond what `priority` dictates.

Value

A character vector containing all unique elements from `ordered_list`, in an order consistent with all pairwise precedence constraints. When constraints allow multiple valid orderings, the result among tied elements is determined by `priority`. The result is always deterministic for a given `ordered_list` and `priority`. Errors if the constraints form a cycle (e.g. A must precede B and B must precede A), as no valid ordering exists in that case.

See Also

`topo_sort` for graph-based topological sorting via the **igraph** package.

Examples

```
## Basic usage
#lst <- list(c("shoe", "ball"), c("shirt", "shoe", "car"), c("ball", "car"))
#topological_sort(lst)
## [1] "shirt" "shoe"  "ball"  "car"
```

```
#
## Element appearing in only one vector
#lst2 <- list(c("shoe", "ball"), c("shirt", "shoe", "car"),
#             c("ball", "car"),  c("stick", "car"))
#topological_sort(lst2)
## [1] "shirt" "stick" "shoe"  "ball"  "car"
#
#topological_sort(lst2, priority = "lexicographic")
## [1] "shirt" "stick" "shoe"  "ball"  "car"
#
#topological_sort(lst2, priority = "least_constrained")
## [1] "stick" "shirt" "shoe"  "ball"  "car"
#
#topological_sort(lst2, priority = "most_constrained")
## [1] "shirt" "shoe"  "ball"  "car"  "stick"

# Cycle detection
## Not run:
topological_sort(list(c("a", "b"), c("b", "a")))
# Error: Cycle detected - no valid ordering exists

## End(Not run)
```

`var2fct` *Convert vector to factor based on threshold of number of unique levels*

Description

This is a wrapper of forcats::as_factor, which sorts numeric vectors before factoring, but levels character vectors in order of appearance.

Usage

```
var2fct(data, unique.n)
```

Arguments

`data`	vector or data.frame column
`unique.n`	threshold to convert class to factor

Value

vector

Examples

```
sample(seq_len(4), 20, TRUE) |>
  var2fct(6) |>
  summary()
sample(letters, 20) |>
  var2fct(6) |>
  summary()
sample(letters[1:4], 20, TRUE) |> var2fct(6)
```

---

vec2choice *Named vector to REDCap choices ('wrapping compact_vec()')*

Description

Named vector to REDCap choices ('wrapping compact_vec()')

Usage

```
vec2choice(data)
```

Arguments

data	named vector

Value

character string

Examples

```
sample(seq_len(4), 20, TRUE) |>
  as_factor() |>
  named_levels() |>
  sort() |>
  vec2choice()
```

Index

∗ **datasets**
    mtcars_redcap, 34
    redcapcast_data, 48
    redcapcast_meta, 49

all_na, 4
apply_factor_labels, 4
apply_field_label, 5
as_factor, 5, *5*, *25*, *43*
as_logical, 7

case_match_regex_list, 8
cast_data_overview, 9
cast_meta_overview, 9
char2choice, 10
char2cond, 10
clean_field_label, 11
clean_redcap_name, 12
compact_vec, 12
create_html_table, 13
create_instrument_meta, 14
cut_string_length, 15

d2w, 15
doc2dd, 16
ds2dd, 17
ds2dd_detailed, 19

easy_redcap, 21
export_redcap_instrument, 22

fct2num, 23
fct_drop, 23, *43*
file_extension, 24
focused_metadata, 25
format_redcap_factor, 25
format_subheader, 26

get_api_key, 26
get_attr, 27
get_id_name, 28
glue, *46*
guess_time_only, 28
guess_time_only_filter, 29

haven_all_levels, 30
hms2character, 30
html_tag_wrap, 31

is.labelled, 31
is_missing, 32
is_repeated_longitudinal, 32

key_get, *26*
key_list, *26*
key_set, *21*, *26*, *27*

mark_complete, 33
match_fields_to_form, 34
mtcars_redcap, 34

named_levels, 35
nav_bar_page, 36
numchar2fct, 36

parse_data, 37
pivot_wider, *46*
possibly_numeric, 38
possibly_roman, 38
process_user_input, 39
process_user_input.character, 39
process_user_input.data.frame, 40
process_user_input.default, 40
process_user_input.response, 41

read_input, 41
read_redcap_instrument, 42
read_redcap_tables, *21*, 42, *46*
redcap_read, *42*, *43*
REDCap_split, *42*, 44, *46*
redcap_wider, 46
redcapcast_data, 48

redcapcast_meta, 49
replace_curly_quote, 50

safe_topological_sort, 50
sanitize_split, 51
set_attr, 52
shiny_cast, 53
split_non_repeating_forms, 53
strsplit, *55*
strsplitx, 54
suffix2label, 55

tbl_summary, *43*
time_only_correction, 56
topo_sort, *57*
topological_sort, *50, 51*, 56

var2fct, 58
vec2choice, 59