Package: datadaptor (via r-universe)

February 17, 2025

Title Modify Labelled Data Sets With Excel Files

Version 0.0.0.9000

Description An R package to modify labelled data sets with commands in Excel files. The commands in this package allow to create new variables, and modify the labels of the variables, as well as the variables themselves. The goal is to provide an easy & concise syntax, and to allow for fast systematic data entry using Excel for advanced users. The commands work on the variables inside the data.frame environment (like e.g. inside dplyr verbs), thus providing an approach that might ease the use for people without in-depth programming experience.

License AGPL (>= 3) + file LICENSE

Suggests testthat, gt, pillar, gtExtras, knitr, htmltools, rmarkdown, markdown, here, spelling, vctrs

Config/testthat/edition 3

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Depends R (>= 4.1.0)

Imports dplyr (>= 1.0.8), haven, magrittr, readxl, tidyr, tibble, purrr, rlang, stringr, fs, labelled, R6, writexl, lifecycle, utils, withr, powerjoin, digest, qs, openxlsx2

VignetteBuilder knitr

Language en-US

URL https://urswilke.gitlab.io/datadaptor, https://gitlab.com/urswilke/datadaptor, https://urswilke.r-universe.dev/datadaptor

Config/pak/sysreqs make libicu-dev libx11-dev zlib1g-dev

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

RemoteUrl https://gitlab.com/urswilke/datadaptor

RemoteRef HEAD

RemoteSha a43647a491c9cfac1e52a5cf8776863901571635

Contents

`apply_command` *Apply a command block to the data*

Description

see `vignette("command_blocks")`

Usage

```
apply_command(cdb, mapping, ...)

## S3 method for class 'cmd_recna_xcpt'
apply_command(cdb, mapping, xs, v, vallab, ...)

## S3 method for class 'cmd_r'
apply_command(cdb, mapping, exs, ...)

## S3 method for class 'cmd_rfun'
apply_command(cdb, mapping, filepath, ex_fun, ...)

## S3 method for class 'cmd_kg'
```

```
apply_command(cdb, mapping, x, y, ...)

## S3 method for class 'cmd_drop'
apply_command(cdb, mapping, xs, ...)

## S3 method for class 'cmd_select'
apply_command(cdb, mapping, exs, ...)

## S3 method for class 'cmd_across'
apply_command(cdb, mapping, exs, ex_fun, exs_fns_names, ex_names, ...)

## S3 method for class 'cmd_filter'
apply_command(cdb, mapping, exs, ...)

## S3 method for class 'cmd_verbatim'
apply_command(
  cdb,
  mapping,
  x,
  v,
  varlab,
  vs,
  vallabs,
  id_list,
  v0,
  ex_further_cond,
  id = mapping$opts$da$id_var,
  ...
)

## S3 method for class 'cmd_verbatim_custom'
apply_command(
  cdb,
  mapping,
  x,
  varlab,
  vs,
  vallabs,
  id_list,
  v0,
  ex_further_cond,
  ex_assign,
  id = mapping$opts$da$id_var,
  ...
)

## S3 method for class 'cmd_merge'
apply_command(cdb, mapping, xs, filepath, id, coal, ...)
```

```
## S3 method for class 'cmd_addfile'
apply_command(cdb, mapping, filepath, ...)

## S3 method for class 'cmd_rename_varsheet'
apply_command(cdb, mapping, xs, ys, ...)

## S3 method for class 'cmd_rename'
apply_command(cdb, mapping, xs, ys, ...)

## S3 method for class 'cmd_if'
apply_command(cdb, mapping, x, ex_cond, ex, ...)

## S3 method for class 'cmd_comp'
apply_command(cdb, mapping, x, ex, ...)

## S3 method for class 'cmd_debug'
apply_command(cdb, mapping, ...)

## S3 method for class 'cmd_set_lab'
apply_command(cdb, mapping, x, varlab, ...)

## S3 method for class 'cmd_newlab'
apply_command(cdb, mapping, x, varlab, ...)

## S3 method for class 'cmd_rmval'
apply_command(cdb, mapping, x, y, vs, varlab, ...)

## S3 method for class 'cmd_set_labs'
apply_command(cdb, mapping, x, varlab, vs, vallabs, ...)

## S3 method for class 'cmd_add_labs'
apply_command(cdb, mapping, x, varlab = NULL, vs, vallabs, ...)

## S3 method for class 'cmd_newvall'
apply_command(cdb, mapping, x, varlab = NULL, vs, vallabs, ...)

## S3 method for class 'cmd_rec'
apply_command(cdb, mapping, x, y, varlab, vs0, vs, vs2, vallabs, ...)

## S3 method for class 'cmd_sumvar'
apply_command(cdb, mapping, x, y, varlab, vs0, vs, vallabs, ...)

## S3 method for class 'cmd_dic'
apply_command(cdb, mapping, x, y, ...)

## S3 method for class 'cmd_autorec'
apply_command(cdb, mapping, x, ...)
```

```
## S3 method for class 'cmd_str_to_num'
apply_command(cdb, mapping, x, ...)
```

Arguments

cdb	command_block object
mapping	mapping object
...	Arguments passed to method
v, v0, vs, vs0, vs2	Numeric value(s)
vallab, vallabs	Value label(s)
filepath	Character string containing valid file path
x, xs, y, ys	character string (vector) of variable names in mapping$dat_mod
varlab	Character string containing a variable label
id_list	Vector of id values in mapping$dat_mod[id].
id	Character string of the variable name of the id variable in mapping$dat.
coal	Character string containing either "xy" or "yx". This determines if powerjoin::coalesce_xy() or powerjoin::coalesce_yx() is used when merging data with variables that already exist.
ex, exs, ex_cond, ex_fun, ex_further_cond, ex_assign, exs_fns_names, ex_names	Character strings containing valid R expressions. They will be evaluated in mapping$opts$da$expr_eval_env (see get_mapping_options()), except exs which contains a list of expressions evaluated in the global environment.

Methods (by class)

- apply_command(cmd_recna_xcpt): Replace missing values of the labelled variables in mapping$dat_mod (except those specified in xs) with the value v, labelled vallab.
- apply_command(cmd_r): Execute R code
- apply_command(cmd_rfun): Execute the function named ex_fun and defined in the R script named filepath.
- apply_command(cmd_kg): Split variable
- apply_command(cmd_drop):
- apply_command(cmd_select):
- apply_command(cmd_across):
- apply_command(cmd_filter):
- apply_command(cmd_verbatim):
- apply_command(cmd_verbatim_custom):
- apply_command(cmd_merge):
- apply_command(cmd_addfile):

- apply_command(cmd_rename_varsheet):
- apply_command(cmd_rename):
- apply_command(cmd_if):
- apply_command(cmd_comp):
- apply_command(cmd_debug):
- apply_command(cmd_set_lab):
- apply_command(cmd_newlab):
- apply_command(cmd_rmval):
- apply_command(cmd_set_labs):
- apply_command(cmd_add_labs):
- apply_command(cmd_newvall):
- apply_command(cmd_rec):
- apply_command(cmd_sumvar):
- apply_command(cmd_dic):
- apply_command(cmd_autorec):
- apply_command(cmd_str_to_num):

Examples

```
# see vignette("command_blocks")
```

`command_block_classes` `command_block` *overview*

Description

A dataset containing the list of `keywords` that can be used in the Excel mapping file to generate command_block objects.

Usage

```
command_block_classes
```

Format

A data frame of 26 `keywords` and their corresponding command_block classes:

keyword Excel mapping file keyword

command_block String denoting the name of the command_block subclass

sheet The sheet(s) in the Excel mapping file from where this command can be called

Examples

```
# print all rows of tibble:
print(command_block_classes, n = 111)
```

create_mapping *Create an Excel mapping file based on a labelled dataframe*

Description

The mapping file consists of the sheets "Variables", "Label", "Verbatims" & "Free". Each of these controls different aspects of data manipulations you can apply to a labelled dataset. You can add as much of those sheets (sheets starting with one of these 4 prefixes) as you want to the file. The commands entered in the mapping file can then be executed on the data set with the `Mapping` class. The sequence of commands is executed in the same order as the sequence of sheets in the mapping file.

Usage

```
create_mapping(df_raw, mapping_file, mapping_type = "excel")
```

Arguments

`df_raw`	dataframe with labelled variables, e.g. resulting from haven::read_sav
`mapping_file`	name of the Excel file to be created
`mapping_type`	String specifying the mapping type. Either "excel" or "list". Defaults to "excel".

Examples

```
spss_file <- system.file(
  "extdata",
  "mtcars_labelled.sav",
  package = "datadaptor"
)
df <- haven::read_sav(spss_file)
## Not run:
create_mapping(df, "mapping.xlsx")

## End(Not run)
```

create_mapping_workbook *Create a mapping openxlsx2 workbook object*

Description

Create a mapping openxlsx2 workbook object

Usage

```
create_mapping_workbook(df_raw)
```

Arguments

df_raw	dataframe with labelled variables, e.g. resulting from haven::read_sav

Value

openxlsx2 workbook object

See Also

create_mapping() to directly save the mapping to a file.

Examples

```
spss_file <- system.file(
  "extdata",
  "mtcars_labelled.sav",
  package = "datadaptor"
)
df <- haven::read_sav(spss_file)
## Not run:
create_mapping_workbook(df)

## End(Not run)
```

---

curlychop *Multiply repetitive parts of command blocks using curly braces*

Description

This function turns the first line of command blocks of the "Free" sheets into multiple by replacing the curly braces by each of the parts inside (separated by spaces). This can help to save yourself from repetitive writing without diving into something like regular expressions.

Usage

```
curlychop(df_free_raw)
```

Arguments

df_free_raw	code blocks read in by mapp_free_sheet_cmd_table_raw()

Value

Dataframe containing multiple code blocks. The number of returned code blocks corresponds to the number of space separated parts in the curly brackets. The part embraced by the curly braces of the initial code block is replaced by each of the space separated parts.

Examples

```
# Minimal example:
df_curly <- data.frame(
  X1 = "#IF",
  X2 = "q{2 3} == 1",
  X3 = "kq{5 6} = {7 8}",
  X4 = NA_character_,
  row = "1"
)
df_curly
curlychop(df_curly)
```

`diff_data` *Diff to labelled data frames*

Description

Diff to labelled data frames

Usage

```
diff_data(df1, df2, id_var = "DC_ID", n_max = 20, warn = TRUE)
```

Arguments

`df1`	data frame 1
`df2`	data frame 2
`id_var`	name of the id variable (string)
`n_max`	Maximum number of values/value labels in variables. Variables containing more than `n_max` won't be printed.
`warn`	whether to emit a warning if `df1` and `df2` don't contain the same ids.

Value

data frame of diff results: For every variable `var`in the data.frames, the counts `n` are shown for all the values (one column per value type), variable and value labels, well as their type (column prefixes). The column suffixes `"_old"` and `"new"` indicate `df` and `df2`, respectively. If the type column is empty, the variable doesn't exist in the respective data.frame.

Examples

```
mapping_file <- system.file("extdata", "mapping.xlsx", package = "datadaptor")
mapping <- Mapping$new(mtcars_labelled, mapping_file)
mapping$modify_data()
diff_data(mapping$dat, mapping$dat_mod, "id")
```

format_sheet_data *Format the dataframe returned by reading an excel sheet with* `openxlsx2::wb_read()`

Description

Turns the dataframe into a tibble with character columns and trims the leading/trailing spaces of the strings.

Usage

```
format_sheet_data(df, cols = dplyr::everything())
```

Arguments

df	dataframe
cols	tidy-select expression to specify which columns to trim; defaults to `dplyr::everything()`

Value

formatted dataframe

Examples

```
data.frame(a = " a ", b = 1) |> format_sheet_data()
```

fruit_survey *Toy data of a fictional survey about fruits*

Description

This dataset contains the made-up answers of a fictional survey about fruits. The same data is also included in the package in SPSS format. See in the examples section how to load the SPSS version to R.

Usage

```
fruit_survey
```

Format

A data frame with 100 observations on 12 variables:

id respondent id

q1 answers to Q1

q2_1 answers to Q2 - 1st item

q2_2 answers to Q2 - 2nd item

q2_3 answers to Q2 - 3rd item

q2_97 answers to Q2 - 4th item

q3_1 answers to Q3 - 1st item

q3_2 answers to Q3 - 2nd item

q3_3 answers to Q3 - 3rd item

q3_97 answers to Q3 - 4th item

q4 answers to Q4

q5 answers to Q5

Examples

```
datadaptor::fruit_survey
path <- system.file("extdata", "fruit_survey.sav", package = "datadaptor")
df <- haven::read_sav(path)
df
labelled::generate_dictionary(fruit_survey)
```

---

gen_data_table *Generate data counts table*

Description

Generate data counts table

Usage

```
gen_data_table(df, values_drop_na = FALSE)
```

Arguments

df	dataframe
values_drop_na	remove missing values? (passed to `tidyr::pivot_longer()`.)

Value

Counts and labels data frame

Examples

```
gen_data_table(mtcars_labelled)
```

get_mapping_options *Mapping parameters*

Description

`get_mapping_options()` is a helper function called by `Mapping$new(...)` or `Mapping$set_options(...)` to generate the parameters in the `opts$da` field of a Mapping object.

Usage

```
get_mapping_options(
  id_var = NULL,
  error_out = "unsafe",
  debug = FALSE,
  save_path = tempdir(),
  write_mapping_to_txt = FALSE,
  expr_eval_env = new.env(parent = baseenv()),
  lab_before_var_sheet = "yes",
  miss_rec_lab = "FILTER",
  miss_rec_val = -2,
  na_to_filter = TRUE,
  not_miss_to_filter_vars = NA_character_,
  verbose = FALSE,
  ...
)
```

Arguments

`id_var`	character string of the id variable name in the dataset.
`error_out`	character string. Either "safe", "quiet" or "unsafe" (the default). Whether to continue executing when a command block fails ("safe" or "quiet"), or to error out ("unsafe"). Adds a column "error" to the mapping's command table `mapping$cmd_tbl`. The difference between "safe" & "quiet" is whether to print errors & warnings to the command line while running `Mapping$modify_data()`.
`debug`	whether to enter in debug mode when an error occurs. Automatically sets `error_out = "safe"`.
`save_path`	filepath where to save files.
`write_mapping_to_txt`	Whether to write the `Mapping`'s data to text files (for instance, in order to allow for git version control during the course of a project that evolves). Defaults to FALSE. Will probably be deprecated in the future.
`expr_eval_env`	The environment where expressions are evaluated. See `?safer_env`.
`lab_before_var_sheet`	Whether to apply the "Label" sheet before the "Variables" sheet. Defaults to `TRUE`.

miss_rec_lab	Label given if na_to_filter = TRUE.
miss_rec_val	Replace value if na_to_filter = TRUE.
na_to_filter	if TRUE (the default), NA values ("missing" in SPSS) are transformed with. apply_command.cmd_recna_xcpt() in the first command block.
not_miss_to_filter_vars	Space separated character string of variable names spared out for apply_command.cmd_recna_xcpt().
verbose	Defaults to FALSE; If TRUE will be more chatty about what's happening (Very preliminary! at the moment, only used in crosstabser).
...	used to pass arguments from Mapping$new(...)

Value

list object (see examples)

Examples

```
get_mapping_options()
```

---

label_tables *Generate tables with the variables' labels*

Description

gen_var_table() generates the "Variables" sheet table with the variable labels in the data.

gen_label_table() generates the "Label" sheet table with the value labels in the data.

Usage

```
gen_var_table(dat)

gen_label_table(dat)
```

Arguments

dat	The dataset containing variables of type haven::labelled .

Value

For gen_var_table() a dataframe containing the table of the "Variables" sheet.

For gen_label_table() a dataframe containing the table for the "Label" sheet.

Examples

```
spss_file <- system.file(
  "extdata",
  "mtcars_labelled.sav",
  package = "datadaptor"
)
dat <- spss_file |>
  haven::read_sav()
gen_var_table(dat)
gen_label_table(dat)
```

`Mapping` *Mapping class*

Description

The `R6::R6Class()` `Mapping` can be used to apply the changes specified in the command blocks of an Excel mapping file to a (labelled) dataframe.

The information of the Excel mapping file results in the `cmd_tbl` dataframe field of the mapping object. This dataframe has a column `command_blocks` which is applied to the data in the `dat` field by the method `modify_data()` and then results in the `dat_mod` field.

Public fields

`dat` (filepath to pass to `haven::read_sav()` to read in the) labelled dataframe to apply the mapping on.

`mapping_file` Mapping file document (see `mapping_type`). The class of this string will be set to "mapping_type".

`mapping_type` String specifying the mapping type. Either "excel" or "list". If not specified, when initializing it is auto-determined:

- "list": If `mapping_file` is a list object.
- "excel": If the `mapping_file` path ends on "xlsm" or "xlsx".

`cmd_tbl` Dataframe with the command block information

`cmd` R list structure containing the processed command block information of the Excel mapping file. **[Experimental]**

`dat_mod` modified dataframe

`opts` Parameter list object (in `opts$da`)

`wb` For an excel mapping, the openxlsx2 workbook object, otherwise `NULL`.

`ditw` This is the "dust in the wind" list object field that stores data that didn't make it into their own field. For developers only! For reproducible code you should NEVER rely on this field as it might be subject to change without any warning.

Methods

Public methods:

- Mapping$new()
- Mapping$process_sheet_commands()
- Mapping$modify_data()
- Mapping$save()
- Mapping$set_options()
- Mapping$read_data()
- Mapping$clone()

Method `new()`**:** Initialize a Mapping object

Usage:

```
Mapping$new(
  dat = NULL,
  mapping_file = NULL,
  mapping_type = NULL,
  process_sheets = TRUE,
  ...
)
```

Arguments:

`dat` Dataframe to apply the mapping on.

`mapping_file` Path to the Excel mapping file.

`mapping_type` String specifying the mapping type. Either "excel" or "list".

`process_sheets` (default TRUE) allows (process_sheets = FALSE) to postpone the execution of the commands in the Excel mapping file to the modify_data() method

`...` Arguments passed to the `Mapping$set_options()` method which will populate the `Mapping$opts$da` field of the object.

Returns: A new `Mapping` object.

Method `process_sheet_commands()`**:** Parse the sheet data of the mapping file and derive the command blocks included. Automatically run in the constructor if `process_sheets = TRUE` (the default). Automatically run by the `modify_data()` method if not done before.

Usage:

```
Mapping$process_sheet_commands()
```

Method `modify_data()`**:** Run all command blocks of the mapping file. The commands in the argument `command_blocks` (defaults to the Mapping's `cmd_tbl$command_blocks` field) successively are applied to the data in the field `"dat_mod"` according to their subclass methods of `apply_command()`.

Usage:

```
Mapping$modify_data(reset = TRUE, command_blocks = self$cmd_tbl$command_blocks)
```

Arguments:

`reset` whether to apply the modifications to the input data (field `dat`) or whether to keep previous modifications (only relevant when applying `modify_data()` multiple times).

command_blocks The "command_blocks" object results of the processing of the Excel mapping file.

Method `save()`**:** Save the modified data to a file

The data can be exported to the file formats of Stata & SPSS. The Excel export removes variable & value labels.

Usage:

```
Mapping$save(path = NULL, show = FALSE, name = "dat", filetype = "sav", ...)
```

Arguments:

path character() string or NULL. If NULL (the default) it will write the file to the path in self$opts$da$save_path with the file name & filetype.

show Whether to directly open the file (needs the according software installed and setup to open its filetype).

name character() string containing the file name to be written. Is overwritten, by path if not NULL.

filetype character() string containing the file type to be written. Is overwritten, by path if not NULL.

... Passed to methods.

Examples:

```
\dontrun{
# Create a Mapping object from the files provided by the package:
mapping_file <- system.file(
  "extdata",
  "mapping.xlsx",
  package = "datadaptor"
)
spss_file <- system.file(
  "extdata",
  "mtcars_labelled.sav",
  package = "datadaptor"
)
m <- Mapping$new(spss_file, mapping_file)

# The method applies the modifications specified in a command_blocks object
m$modify_data(command_blocks = m$cmd_tbl$command_blocks)
m$save("stata_data.dta", show = TRUE)
}
```

Method `set_options()`**:** Set / change options of the Mapping object

The dots (...) can be passed here to change settings, or already when initializing the object with `Mapping$new(...)`

Additionally to the dots you can also pass parameters from an Excel mapping file by using named regions starting with "R_", for instance, "R_id_var" will become "id_var". The complete set of arguments consists of he default values in `get_mapping_options()` overwritten by the above named regions of the Excel file, and all this can be overwritten by the dots.

The part of the arguments known to `get_mapping_options()` is written to the opts$da field, The rest is written to the opts$dev field.

Usage:

```
Mapping$set_options(...)
```

Arguments:

`...` arguments passed to `get_mapping_options()`

Method `read_data()`**:** Read in dataset

Usage:

```
Mapping$read_data(dat, ...)
```

Arguments:

`dat` Dataset indentifier (see `?read_data_` helper function).

`...` Arguments passed to `read_data_()` helper function.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Mapping$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# Create a Mapping object from the files provided by the package:
mapping_file <- system.file(
  "extdata",
  "mapping.xlsx",
  package = "datadaptor"
)
spss_file <- system.file(
  "extdata",
  "mtcars_labelled.sav",
  package = "datadaptor"
)
mapping <- Mapping$new(spss_file, mapping_file)

# The spss_file path was read into a dataframe in the "dat" field of the
# mapping object:
mapping$dat

# The Excel mapping file is translated to a `command_blocks()` object.
# It contains the processed information in a list structure that has
# its own print method.
# You can access it with
## Not run:
mapping$cmd_tbl$command_blocks

## End(Not run)
# Apply the command blocks to the dataset:
mapping$modify_data()
```

```
# Access the modified dataframe:
mapping$dat_mod

# To write it back to an SPSS file, you could do:
# mapping$save("path/to/your/file.sav")
# or with haven (used under the hood by `save()`):
# haven::write_sav(mapping$dat_mod, "path/to/your/file.sav")

## ------------------------------------------------
## Method `Mapping$save`
## ------------------------------------------------

## Not run:
# Create a Mapping object from the files provided by the package:
mapping_file <- system.file(
  "extdata",
  "mapping.xlsx",
  package = "datadaptor"
)
spss_file <- system.file(
  "extdata",
  "mtcars_labelled.sav",
  package = "datadaptor"
)
m <- Mapping$new(spss_file, mapping_file)

# The method applies the modifications specified in a command_blocks object
m$modify_data(command_blocks = m$cmd_tbl$command_blocks)
m$save("stata_data.dta", show = TRUE)

## End(Not run)
```

`mtcars_labelled` *Labelled mtcars version*

Description

A labelled version of the `mtcars` dataset (see `?mtcars`). The data is stored in variables of the type haven::labelled. The same data is also included in the package in SPSS format. See in the examples section how to load the SPSS version to R.

Usage

```
mtcars_labelled
```

Format

A data frame with 32 observations on 13 variables:

id car id

model Name of the car - this information is stored in rownames in `mtcars`.

mpg see `?mtcars`

cyl see `?mtcars`

disp see `?mtcars`

hp see `?mtcars`

drat see `?mtcars`

wt see `?mtcars`

qsec see `?mtcars`

vs see `?mtcars`

am see `?mtcars`

gear see `?mtcars`

carb see `?mtcars`

Examples

```
datadaptor::mtcars_labelled
path <- system.file("extdata", "mtcars_labelled.sav", package = "datadaptor")
df <- haven::read_sav(path)
df
```

`read_data_` *Ingest data from data.frame or file path*

Description

Ingest data from data.frame or file path

Usage

```
read_data_(dat, ...)
```

Arguments

`dat`	String. Either a path to an SPSS file, a data.frame, or `NULL`.
`...`	Arguments passed to methods.

Value

Returns `dat` (unchanged) in case of a data.frame, in case of a character string returns the data.frame resulting of `haven::read_sav(dat)`/`haven::read_dta(dat)`/`qs::qread(dat)` or `openxlsx2::read_xls(x)` for excel files (depending on the file extension) or returns `NULL` in case of `NULL`.

safer_env *Execution environments*

Description

The default environment where expressions from the Excel mapping file are evaluated is baseenv(). (see argument `expr_eval_env` of `?get_mapping_options()`). For a safer option you can use `safer_env` which only contains a selection of base R functions (see example). Additionally to the functions in the used environment, the object `dat_mod` is added to the environment which represents the current state of the data in `Mapping$dat_mod`.

Usage

```
safer_env
```

Format

An object of class `environment` of length 74.

Examples

```
safer_env |>
  as.list() |>
  names()
# Apart from base R functions it also contains `dplyr::case_when()`:
safer_env$case_when
# To use it in a mapping, you can do:
## Not run:
mapping_file <- system.file(
  "extdata",
  "mapping.xlsx",
  package = "datadaptor"
)
spss_file <- system.file(
  "extdata",
  "mtcars_labelled.sav",
  package = "datadaptor"
)
m <- Mapping$new(spss_file, mapping_file, expr_eval_env = safer_env)

## End(Not run)
```

use_known_args *Apply function on the subset of arguments it knows*

Description

This function will execute the function `f` on the subset of the names of `l` which are in the formal arguments of `f`.

Usage

```
use_known_args(f, l)
```

Arguments

`f`	function
`l`	named list of arguments

Value

f applied on the subset of l

Examples

```
use_known_args(mean, list(x = 2, r = 3))
```

Index