Package ‘pewmethods’

February 21, 2020

Title Pew Research Center Methods Miscellaneous Functions

Version 1.0

Description Some tools for working with survey data that we've developed over the last couple of years, including functions that create quick weighted crosstabs, facilitate survey weighting, ease recoding variables, and extract elements from files originating from SPSS.

Depends R (>= 3.5)

Imports dplyr,
tibble,
tidyr,
purrr,
forcats,
stringr,
haven,
survey,
rlang,
mice,
ranger,
openxlsx

Suggests knitr,
rmarkdown

VignetteBuilder knitr

License MIT

Encoding UTF-8

LazyData true

RoxygenNote 7.0.2

R topics documented:

`acs_2017_excerpt` *American Community Survey 2017 Public Use Microdata Excerpt*

Description

Contains a weight for each combination of sex, 6-category age, and 3-category education proportional to its share in the non-institutionalized U.S. population, aged 18 and above.

Usage

```
data(acs_2017_excerpt)
```

Format

A data.frame with 36 rows and 3 variables:

sex Male/Female

agecat6 Age, divided into 6 buckets

educ3 HS or less, some college, college graduate+

weight A weight for each row proportional to the share of that row's variables in the population

---

`calculate_deff` *Calculate design effect*

Description

Takes a vector of weights and calculates the design effect, or the proportional increase in variance due to weighting. Calculated using the Kish (1965) approximation.

Usage

```
calculate_deff(weight, include_zeroes = FALSE)
```

Arguments

`weight`	A vector of weights. Non-numeric weights will be coerced to numeric.
`include_zeroes`	Should zero weights be included when calculating `n`, `sd_wt`, and `deff`? Defaults to `FALSE`.

Value

A tibble containing the following columns:

`n` The sample size

`sd_wt` The standard deviation of the weights

`deff` The Kish approximation of an overall survey design effect

`ess` The effective sample size

`moe` The design-adjusted margin of error at 95

Examples

```
calculate_deff(dec13_excerpt$weight)
```

`create_raking_targets` *Create raking targets*

Description

Given a dataset, creates a list of tibbles, each summarizing the marginal distribution of categorical variables from that dataset. These can be used as raking targets by passing them to the `pop_margins` argument in `rake_survey`. Each element in the list will have two columns: the name of the raking target, which will by default have the prefix "rk_" appended to indicate being a raking target, and the percentage of each category in that variable.

Usage

```
create_raking_targets(bm_data, vars, prefix = "rk_", new_sep = "_", wt = NULL)
```

Arguments

`bm_data`	The name of the dataset to be used for calculating marginal distributions.
`vars`	A character vector containing the names of all the variables that will be used for raking targets. Interactions between variables can be specified using the convention `"variable1:variable2"`.
`prefix`	A string containing the prefix to be prepended to the name of the first column of each raking target. "rk_" by default.
`new_sep`	The character separating interaction variables, if applicable. `"_"` by default, does not do anything if no interaction variables present.
`wt`	The weight to be used in calculating the targets. For unweighted targets, use wt = 1.

Details

Datasets used to create raking targets generally come from microdata describing your population, which is taken to be the ground truth. For example, one frequently used dataset for obtaining demographic raking targets for the population of U.S. adults is the American Community Survey. If the dataset used is itself a survey, it may come with ts own survey weights needed for the raking targets to accurately describe the target population, in which case those weights need to be passed to the `wt` argument. To prevent errors, a value must be supplied for `wt`. Use `wt = 1` if targets are to be based on unweighted data.

It is good practice to separate out variables used for raking from the raw variables, because raking variables may be processed via recoding and imputation, among other things. The `prefix` argument enforces this practice by adding a prefix to the names of the raking variables. If the output of this function is subsequently passed to the `pop_margins` argument of `rake_survey`, the code will search the sample data file for variables with the same names. This is meant to ensure consistency.

This function allows you to pass interactions between variables into the `vars` argument by inserting a `:` between two variable names. When an interaction is specified, the variable names will be concatenated using `new_sep`.

Value

A list, with each element being a tibble returned by `get_totals` for each raking target.

Examples

```
# Here we will use the acs_2017_excerpt dataset included wih the package

# Notice that the names in the output are by default called rk_sex, rk_recage, rk_receduc,
# and rk_sex_receduc
create_raking_targets(acs_2017_excerpt,
                      vars = c("sex", "recage", "receduc", "sex:receduc"),
                      wt = "weight")
```

`dec13_excerpt` *Pew Research Center December 2013 Political Survey Excerpt*

Description

Selected attitudinal and demographic variables from a survey conducted by the Pew Research Center in December 2013, to help illustrate `pewmethods` functions.

Usage

```
data(dec13_excerpt)
```

Format

A tibble with 2001 observations and 14 variables:

psraid Respondent ID

cregion Census region

q1 Obama approval

q2 Strong/not strong Obama approval

q45 Affordable Care Act approval

sex Respondent sex

recage Respondent age brackets

receduc Respondent 3-category education

racethn2 Respondent race/ethnicity

party Party ID

partyln Party lean for independents, etc.

weight Survey weight

llweight Weight for landline RDD sample only

cellweight Weight for cellphone RDD sample only

`df_list_to_xlsx` *Write a list of data frames to Excel*

Description

Takes a list of data frames - for example, obtained by mapping get_totals() over multiple variables - and writes them vertically into an Excel spreadsheet.

Usage

```
df_list_to_xlsx(
  df_list,
  sheet_name,
  outfile,
  overwrite = TRUE,
  label_list = NULL,
  title = NULL,
  borders = "surrounding"
)
```

Arguments

`df_list`	A list of data frames, or tables, or tibbles, or anything rectangular-shaped. For writing to multiple sheets, a list of lists, with each top-level list corresponding to one sheet.
`sheet_name`	The name/s of the sheet/s within the Excel file.
`outfile`	The filename of the Excel file itself. Include the full directory path.
`overwrite`	TRUE by default. If a file of the same name already exists, setting `overwrite` to TRUE will overwrite that file with the output of this function. Setting `overwrite` to FALSE will halt function execution if the file already exists.
`label_list`	Optional list of labels that correspond to each of the elements of `df_list`. These are printed above the corresponding tables in the output. For writing to multiple sheets, a list of lists, with each top-level list corresponding to one sheet.
`title`	Optional title to be printed on the first row of the Excel sheet. For writing to multiple sheets, there must be one title per sheet.
`borders`	Optional borders to be drawn around each table. Will create a border around the outside by default. See `openxlsx::writeData` for other options.

Value

This function does not return an object within R. Instead, it will write an Excel file to the filename specified in `outfile`.

Examples

```
library(dplyr)
# Identify a list of variables I want to run crosstabs on
vars <- c("q1", "q45", "party")

# Run weighted crosstabs by gender on each of the variables and shove them into a list
# get_totals() is a weighted crosstab function, while map() allows us to do get_totals()
# over and over again on everything in the vector called vars.
dec13_tabs <- purrr::map(vars, ~get_totals(.x, dec13_excerpt, by = "sex", wt = "weight"))

# Get a corresponding list of labels for vars
dec13_labs <- get_spss_label(dec13_excerpt, vars, unlist = FALSE)

# Write the weighted crosstabs to an Excel spreadsheet
# Note that if you run this example code on your own computer, then you will end up
# with an Excel spreadsheet written to your current working directory.
# Remove comment to run
# df_list_to_xlsx(df_list = dec13_tabs, sheet_name = "Dec 2013 crosstabs",
#                outfile = "df_list_to_xlsx_example_output.xlsx", label_list = dec13_labs,
#                   title = "Dec 2013 crosstab example")

# What if we want crosstabs across multiple sheets?
# Let's do one sheet for each education category.
# Recode the receduc variable to change DK/Ref to NA
dec13_excerpt <- dec13_excerpt %>% mutate(receduc = dk_to_na(receduc))

# Convert the dataset into a list of three datasets, one per education category
dec13_list <- dec13_excerpt %>% split(.$receduc)

# For each of the three datasets, get weighted crosstabs
# This will give us a list of three (one per education category),
# each with its own list of three (one per crosstab)
dec13_tabs <- purrr::map(dec13_list, function(df) {
  purrr::map(vars, ~get_totals(.x, df, wt = "weight"))
  })

# Associate a label with each crosstab. The split() function removes labels,
# so we're going to call get_spss_label over the original dataset.
dec13_labs <- purrr::map(dec13_list, function(df) {
  get_spss_label(dec13_excerpt, vars, unlist = FALSE)
  })

# Write the weighted crosstabs to an Excel spreadsheet with multiple sheets
# Notice that a warning about unsupported characters is printed
# Remove comment to run
# df_list_to_xlsx(dec13_tabs, sheet_name = names(dec13_tabs),
#              outfile = "df_list_to_xlsx_example_output_2.xlsx", label_list = dec13_labs)
```

---

dk_to_na *DK to NA*

Description

Takes a factor and converts "Don’t know/Refused" responses to `NA`. Matches a variety of common patterns such as "Don’t know", "Refused"s, "DK/Ref", etc.

Usage

```
dk_to_na(
  f,

   pattern = "^[Dd]on['’]t [Kk]|^[Dd][Kk]|^[Rr]efuse|^[Dd][Kk]/[Rr]ef|\\(VOL\\.\\)$"
)
```

Arguments

`f`	A factor variable.
`pattern`	A regular expression that matches factor levels to convert to NA.

Details

Make sure to tabulate the variable after running this function and check that levels weren’t unintentionally removed.

Value

The factor variable, with DK/Ref values set to NA.

Examples

```
demonstration <- factor(c(rep("Yes", 57), rep("No", 45), rep("Don't know", 4),
                          rep("Refused", 2), rep("DK/Ref", 10), rep("Not sure (VOL.)", 5)))
tablena(demonstration)
demonstration <- dk_to_na(demonstration)
tablena(demonstration) # success!
```

`dummify_factors` *Convert all categorical variables into dummy/indicator variables*

Description

Takes a data frame and separates all columns containing categorical variables into dummy variables (one column per category). Unlike model.matrix(), does not exclude any categories from being converted into columns, and retains missing values.

Usage

```
dummify_factors(df, dummify_characters = TRUE, max_levels = 52, sep = "__")
```

Arguments

`df`	A data frame containing the factors to be converted to dummies.
`dummify_characters`	Should `character` variables be converted to factors and dummified? Defaults to `TRUE`.
`max_levels`	The maximum number of levels that a categorical variable can have in order to be converted to dummies. This is to prevent converting variables such as respondent IDs or open-ended responses to dummy variables. Defaults to 52, which is slightly more than the number of U.S. states plus Washington, D.C.
`sep`	A character string that will go in between the original variable name and the corresponding factor level in the output, e.g. VARIABLE__Category. Defaults to `"__"`.

Value

A copy of the original data frame where all factor variables have been replaced with dummies. Formatted as a tibble.

Examples

```
test <- dummify_factors(dec13_excerpt)
head(test)
```

`fct_case_when` *Factor case_when*

Description

Wrapper around `dplyr::case_when` that converts the output to a factor and preserves the order in which value labels were passed into the function.

Usage

```
fct_case_when(...)
```

Arguments

`...`	A sequence of two-sided formulas consistent with `dplyr::case_when`.

Details

Unlike case_when, fct_case_when allows factors to be passed in as right-hand-side arguments - they are treated internally as characters, but the resulting vector will preserve the order of the original factor levels.

Value

The output of `dplyr::case_when`, as class `"factor"` and ordered however you wanted it.

Examples

```
library(dplyr)
partysum <- with(dec13_excerpt, fct_case_when(party == "Republican" ~ "Rep/Lean Rep",
                          party == "Democrat"  ~ "Dem/Lean Dem",
                          partyln == "Republican" ~ "Rep/Lean Rep",
                          partyln == "Democrat" ~ "Dem/Lean Dem",
                          TRUE ~ partyln)
                          )

# Compare to normal case_when() and then factor(), which will arrange the levels in
# unwanted alphabetical order

partysum <- with(dec13_excerpt, factor(case_when(party == "Republican" ~ "Rep/Lean Rep",
                             party == "Democrat"  ~ "Dem/Lean Dem",
                             partyln == "Republican" ~ "Rep/Lean Rep",
                             partyln == "Democrat" ~ "Dem/Lean Dem",
                             TRUE ~ as.character(partyln))))
```

`get_spss_label` *Get SPSS label*

Description

Returns variable labels from an SPSS dataset read in with either the `foreign` or `haven` packages.

Usage

```
get_spss_label(df, var, null_label = "No label found", unlist = TRUE)
```

Arguments

`df`	An SPSS dataset originally read into R via either `foreign` or `haven`.
`var`	`character` vector containing the names of variables whose labels should be returned.
`null_label`	`character` value to return if there is no label for a variable. Defaults to `"No label found"`.
`unlist`	Should the output be converted from a list to a vector? Defaults to `TRUE`. Set to `FALSE` to return a list instead.

Value

The SPSS label/s, as a vector or a list.

Examples

```
get_spss_label(dec13_excerpt, "q1")
get_spss_label(dec13_excerpt, c("q1", "party"))
get_spss_label(dec13_excerpt, c("q1", "sdfklsdf", "party"), null_label = "here be dragons")
get_spss_label(dec13_excerpt, c("q1", "sdfklsdf", "party"), unlist = FALSE)
```

`get_totals` *Get weighted percentages or totals*

Description

Takes a categorical variable and returns the weighted or unweighted percentage or total of each category. Can include a grouping variable. Can be used to compare weights versus one another.

Usage

```
get_totals(
  var,
  df,
  wt = NULL,
  by = NULL,
  by_total = FALSE,
  percent = TRUE,
  include_unw = FALSE,
  digits = NULL,
  complete = TRUE,
  na.rm = FALSE
)
```

Arguments

`var`	`character`, indicating the name of the variable to be tabulated. Specify interactions by separating variable names with a colon.
`df`	The `data.frame` containing the variables to be tabulated.
`wt`	A `character` vector containing the name(s) of the weight variable(s) to be used in tabulating results. If nothing is passed, results will be unweighted.
`by`	For creating crosstabulations, an optional `character` variable containing the name of the variable to be crossed with `var`. Can pass multiple variables in as a vector.
`by_total`	`logical` indicating whether a "total" column should be returned in addition to columns defined by variables passed to `by`. Defaults to `FALSE`.
`percent`	Should the results be scaled as percentages? Defaults to `TRUE`. If `FALSE`, weighted totals are returned.
`include_unw`	Include unweighted frequencies in the output along with weighted. Defaults to `FALSE`

`digits`	The number of decimal points displayed. Defaults to value specified in `options("digits")`.
`complete`	TRUE/FALSE: Should factor levels with no observations be included in the results? Defaults to `TRUE`.
`na.rm`	If `FALSE`, `NA` values in `var` are included in the results and included in the denominator for calculating percentages. If `TRUE`, they are excluded from any calculations. Defaults to `FALSE`.

Details

If no arguments are supplied to by, then the column names will be the weight names. If arguments are supplied to by, then the column names will be the categories of the grouping variable, and the output will have an additional column for the weight name.

Value

A `data.frame` with a column for the variable name, columns displaying percentages or totals, and additional columns as specified by arguments to this function.

Examples

```
library(dplyr)
# Basic unweighted crosstab
get_totals("q1", dec13_excerpt)

# Totals instead of percentages
get_totals("q1", dec13_excerpt, percent = FALSE)

# Weighted crosstab
get_totals("receduc", dec13_excerpt, wt = "weight")

# Weighted crosstab by grouping variable
get_totals("q1", dec13_excerpt, wt = "weight", by = "receduc")

# Compare weights, including unweighted
# Let's make a fake weight by combining the landline and cellphone weights
dec13_excerpt <- dec13_excerpt %>% mutate(fake_weight = coalesce(llweight, cellweight))
get_totals("q1", dec13_excerpt, wt = c("weight", "fake_weight"), include_unw = TRUE)

# Use dplyr::filter along with complete = FALSE to remove unwanted categories from the base
get_totals("q1", dec13_excerpt %>% filter(q1 != "Don't know/Refused (VOL.)"), wt = "weight",
           complete = FALSE)

# Alternatively, filter unwanted categories out beforehand with dk_to_na
# and then use na.rm = TRUE
dec13_excerpt <- dec13_excerpt %>% mutate(q1 = dk_to_na(q1))
get_totals("q1", dec13_excerpt, wt = "weight", na.rm = TRUE)
```

impute_vars *Quick impute*

Description

Given a dataset, singly imputes specified variables within that dataset. Meant for tasks that need data to be filled in as an intermediate step, such as filling in a small amount of missing values in variables that will be used for raking. Not meant to be used for applications for which measuring the uncertainty due to imputation is important.

Usage

```
impute_vars(.data, to_impute = NULL, method = "ranger", seed = NA, ...)
```

Arguments

`.data`	The `data.frame` to be imputed.
`to_impute`	The variables in the dataset for which missing data should be imputed. Can be a `character` vector of names, a `numeric` vector of column positions, or a list of columns generated by `dplyr::vars`. (See `help("select_at")` for details.) Must have at least two variables. Defaults to `NULL`, in which case this function will search for variables with the prefix "rk_", meant to fit in a workflow where missing data is imputed for variables to be used in weighting.
`method`	The imputation method, passed to `mice()`. The default method is random forest imputation via the `ranger()` package, which is a custom method that comes with the `pewmethods` package. Other methods built into the `mice` package will work.
`seed`	Ensures that the missing values will be filled in the exact same way when rerun. No seed is set by default.
`...`	Other arguments passed to `mice::mice`.

Details

This function is a wrapper around `mice::mice` that does only one imputation and does not output any diagnostics. The main use of this function is to quickly impute only some variables in a dataset. Quick imputation is useful for some limited purposes such as the need to fill in the generally small amounts of missing data in variables to be used in raking.

Note that the imputation model will only use data from the variables you pass to this function. If you pass only two variables, then only those two variables and nothing else will be used to fill in missing values. If there are other variables in your data that are strongly related to the variables to be imputed, they should be specified in `to_impute`, even if they have no missing data.

Value

The data frame with missing values filled in for only the raking variables, leaving the original ones as they were.

Examples

```
library(dplyr)
# We can use dk_to_na to create new versions of variables where certain factor labels
# are recoded as missing, then impute those variables. If the to_impute argument is not
# specified, the function will by default look for variables starting with "rk_".
dec13_excerpt_raking <- dec13_excerpt %>%
  mutate(rk_sex = sex,
         rk_recage = dk_to_na(recage, pattern = "DK/Ref"),
         rk_receduc = dk_to_na(receduc, pattern = "DK/Ref"),
         rk_racethn2 = dk_to_na(racethn2, pattern = "Ref")) %>%
  impute_vars(.)
# We can also pass specific variables to impute
# In this example, only q1 has missing data, but we want to fill in q1 based on values of
# age, education, gender and race/ethnicity, so we have to pass those variables in as well
dec13_excerpt_raking <- dec13_excerpt %>%
  mutate(q1 = dk_to_na(q1, pattern = "Refused")) %>%
  impute_vars(to_impute = c("q1", "recage", "receduc", "racethn2", "sex"))
```

`keyboardize_punctuation` *Keyboardize punctuation marks*

Description

Takes a dataset and, for all categories in factor variables, standardizes stylized typographic characters to ASCII versions that are easily typeable on a keyboard.

Usage

```
keyboardize_punctuation(df, convert_characters = FALSE)
```

Arguments

`df`	A `data.frame`.
`convert_characters`	Should variables of class `character` also be converted, in addition to variables of class "factor"? Defaults to `FALSE`.

Details

Text labels in datasets occasionally contain characters not easily typeable on a standard keyboard which are almost impossible to distinguish from keyboard characters. (This often happens when pasting text in from Microsoft Word). These include the curly apostrophe, the en-dash, the em-dash and the ellipsis. This function converts these typographic characters to characters that are on keyboards by default: the single straight apostrophe, the hyphen-minus, and three periods.

Value

A copy of the original `data.frame`, with nonstandard characters replaced.

Examples

```
library(dplyr)
# Create an unwanted factor label containing a stylized apostrophe ’ for illustration.
dec13_excerpt <- dec13_excerpt %>% mutate(receduc =
                                          forcats::fct_recode(receduc,
                                                     `Don’t know/Refused` = "DK/Ref"))
table(dec13_excerpt$receduc)
dec13_excerpt <- keyboardize_punctuation(dec13_excerpt)
table(dec13_excerpt$receduc)
```

`mice.impute.ranger` *Random forest imputation using the ranger package*

Description

Designed as an alternative to the the `"rf"` method in the `mice` package. Imputes univariate missing data using random forests. It uses the same algorithm as `"mice.impute.rf"` (included in the `mice` package, but is much faster because it uses the `ranger` package. This function should not be called directly. Instead, call this function through `mice()` by specifying `method = "ranger"`.

Usage

```
mice.impute.ranger(y, ry, x, ntree = 10, wy = NULL, type = NULL, ...)
```

Arguments

`y`	Vector to be imputed
`ry`	Logical vector of length `length(y)` marking observed values of y with TRUE and missing values of y with FALSE.
`x`	the matrix of predictors, without intercept
`ntree`	the number of trees for the random forest algorithm. Default is 10.
`wy`	Logical vector of length `length(y)`. A TRUE value indicates locations in y for which imputations are created.
`type`	Not used. Needed for compatibility with mice
`...`	Other arguments passed to `ranger()`

Value

A vector of imputed values.

pewmethods *pewmethods: Miscellaneous functions by the Pew Research Center Methods team*

Description

The pewmethods package contains functions that have been used internally by the Methods team at Pew Research Center to make everyday work with survey data easier. These include functions to quickly summarize weighted data, clean and recode variables, create survey weights (wrapping around the survey package), output crosstabs, and a number of other common tasks that arise in the analysis of survey data.

Details

For an overview of what's in pewmethods, browse our vignettes by calling `vignette("pewmethods_exploring")` and `vignette("pewmethods_weighting")`.

Author(s)

- **Maintainer:** Arnold Lau alau@pewresearch.org
- Andrew Mercer amercer@pewresearch.org
- Nick Hatley nhatley@pewresearch.org

rake_survey *Rake survey*

Description

Applies the raking algorithm to a survey using specified raking targets in order to obtain weights. Wrapper around the `calibrate` function from the `survey` package with argument `calfun = "raking"`.

Usage

```
rake_survey(
  .data,
  pop_margins,
  base_weight = 1,
  scale_to_n = TRUE,
  epsilon = 5e-06,
  maxit = 100,
  ...
)
```

Arguments

.data	A data.frame containing the survey data
pop_margins	A list of tibbles giving the population margins for raking variables. create_raking_targets outputs such a list in the correct format.
base_weight	The survey's base weight variable, if applicable. Can be ignored if the survey data doesn't come with a base weight.
scale_to_n	If TRUE, scales the resulting weights so so that they sum to the number of observations in the survey.
epsilon	A numeric variable indicating the tolerance for the raking algorithm. When the proportions of the raking variables are within tolerance, the raking algorithm will stop.
maxit	Maximum number of iterations before stopping if raking has not converged.
...	Other arguments passed to survey::calibrate().

Details

The variables in .data must exactly match the variables in pop_margins both in name in the factor values. Additionally, they should contain no missing data.

Value

A vector of survey weights.

Examples

```
library(dplyr)
# Prepare variables from the survey data for raking
dec13_excerpt_raking <- dec13_excerpt %>%
  mutate(rk_sex = sex,
         rk_recage = dk_to_na(recage, pattern = "DK/Ref"),
         rk_receduc = dk_to_na(receduc, pattern = "DK/Ref")) %>%
  impute_vars(.) %>%
  mutate(rk_sex_receduc = interaction(rk_sex, rk_receduc, sep = ":"))
# Prepare population marginal distributions for raking
# Here we will use the acs_2017_excerpt dataset included wih the package
targets <- create_raking_targets(acs_2017_excerpt,
                                 vars = c("sex", "recage", "receduc", "sex:receduc"),
                                 wt = "weight")

# Now that we have raking variables and population targets, we can create a raking weight
fake_weight <- rake_survey(dec13_excerpt_raking, targets)
```

`replace_if` *Replace if*

Description

Takes a vector and replaces elements in the vector according to some condition. (Equivalent to something like `df$var1[df$var2 == TRUE] <-4` but less clunky.) Unlike the base function `replace`, `replace_if` allows replacement of NAs with some other value and only accepts scalar values as replacements (preventing recycling value errors). As with `replace`, this function does not work with factors; use `forcats::fct_recode`.

Usage

```
replace_if(var, condition, replacement)
```

Arguments

`var`	A variable.
`condition`	Either a scalar, a logical vector of the same length as `var`, or NA (in which case the function will look for and replace all NA values).
`replacement`	A scalar.

Value

`var`, replaced.

Examples

```
x <- c(1, 2, 3, 4)
replace_if(var = x, condition = x > 2, replacement = 99)

x <- c(1, 2, NA, 4)
replace_if(var = x, condition = NA, replacement = 99)
```

`set_spss_label` *Set SPSS label*

Description

Writes a label to an SPSS dataset read into R with either the `foreign` or haven packages.

Usage

```
set_spss_label(df, var, label_text)
```

Arguments

`df`	An SPSS dataset read into R via either `foreign` or `haven`.
`var`	`character` vector containing the names of variables to be labeled.
`label_text`	`character` vector containing the label or labels to be assigned.

Value

A copy of `df` with variable labels updated.

Examples

```
dec13_excerpt <- set_spss_label(dec13_excerpt, "q1", "Obama approval")
dec13_excerpt <- set_spss_label(dec13_excerpt,
                                c("q1", "q45"),
                                c("Obama approval", "Obamacare approval"))

# Let's create a fake new variable for illustration
dec13_excerpt$newvar <- 1
dec13_excerpt <- set_spss_label(dec13_excerpt, "newvar", "New variable")
```

`standardize_weights` *Standardize weights*

Description

Divides each weight by the mean and converts missing weights to 0. The resulting weights will have a mean of 1 and sum to either the nominal sample size or the effective sample size, which is the nominal sample size divided by the Kish (1965) approximation of an overall survey design effect

Usage

```
standardize_weights(weight, to_ess = FALSE)
```

Arguments

`weight`	A vector of weights. Coerced to `numeric` if a different type.
`to_ess`	Should the weights sum to the effective sample size? Defaults to `FALSE`.

Value

A vector of standardized weights.

Examples

```
summary(dec13_excerpt$weight)
dec13_excerpt$weight <- standardize_weights(dec13_excerpt$weight)
summary(dec13_excerpt$weight)
```

syssamp *Systematic sample from a dataset*

Description

Takes a dataset, sorts it by one or more variables, and draws samples at regular intervals. A fractional skip interval is used so that each record still has an equal probability of selection when the number of records in the frame is not evenly divisible by the sample size.

Usage

```
syssamp(.data, size, stratvars, return_indices = TRUE, seed = NA)
```

Arguments

`.data`	A dataset, in data.frame or tibble format.
`size`	The size of the desired sample.
`stratvars`	Names of variables by which the sampling frame will be sorted
`return_indices`	Return a vector containing the indices of the selected records. Defaults to `TRUE`. If `FALSE`, a data.frame containing the sampled records will be returned.
`seed`	Optional. Ensures that the same sample will be created if the code is rerun.

Value

Either a vector of indices or a data.frame/tibble.

References

Buskirk, T.D. (2008) Sampling interval. In P.J. Lavrakas, *Encyclopedia of survey research methods*. Thousand Oaks, CA: Sage Publications.

Examples

```
dec13_sample_indices <- syssamp(dec13_excerpt, size = 10, stratvars = c("sex", "recage"))
dec13_sample <- syssamp(dec13_excerpt, size = 10, stratvars = c("sex", "recage"),
                        return_indices = FALSE)
```

`tablena` *Enhanced tables*

Description

Displays tables with NAs, dimension names, and dimension classes by default

Usage

```
tablena(..., show_class = TRUE)
```

Arguments

`...`	Arguments passed to `table()`
`show_class`	TRUE by default. Shows the class of the variable being tabled.

Value

A table

Examples

```
tablena(dec13_excerpt$q2, dec13_excerpt$receduc)
# Also works with the with() function and other such functions that construct environments from data
with(dec13_excerpt, tablena(q2, receduc))
```

`timefactory` *Create running timer functions*

Description

This is a factory function for creating running timers. A function created using `timefactory()` will return the number of seconds since it was first created. It is intended as a simple alternative to wrapping code in multiple calls to `proc.time()`.

Usage

```
timefactory()
```

Value

A function that will return the number of seconds since it was first created.

Examples

```
# Create an initial timer
timer1 = timefactory()
Sys.sleep(3)
timer1()

# Create a second timer that is independent of the first.
timer2 = timefactory()
Sys.sleep(2)

timer1()
timer2()
```

`transfer_spss_labels` *Transfer SPSS labels and attributes from* `haven`

Description

Takes SPSS attributes from a `haven_labelled` variable and attaches them to a recoded numeric variable, which preserves variable and value labels. This function only works with SPSS datasets read in with the haven package.

Usage

```
transfer_spss_labels(
  x,
  reference_df = NULL,
  varname = NULL,
  variable_label = NULL,
  value_labels = NULL,
  format.spss = NULL,
  display_width = NULL,
  use_reference = TRUE
)
```

Arguments

`x`	A vector, or an expression that will output a vector.
`reference_df`	Reference data.frame, that is to say, a data.frame read in using `haven::read_sav()` without any further alterations.
`varname`	`character`, indicating the name of the variable inside the reference data.frame whose attributes should be transferred.
`variable_label`	`character`, indicating the variable label (as displayed in the "Label" column in SPSS). Pass an argument here if you want the SPSS variable label to be different from what's in the original variable.

`value_labels`	Must be a named `numeric` or `integer` vector, indicating the value labels (as displayed in the "Values" column in SPSS). The numeric values of the vector indicate the numeric codes of a categorical variable (if applicable), while the names give the text labels.
`format.spss`	`haven` attribute that stores the variable format (as displayed in the "Width" and "Decimals" columns in SPSS. Pass an argument here if you want the SPSS variable format to be different from what's in the original variable.
`display_width`	`haven` attribute that stores the display width. (This is not what's displayed in the "Width" column.) Pass an argument here if you want the SPSS display width to be different from what's in the original variable.
`use_reference`	`logical` indicating whether a reference dataset and variable exists to transfer SPSS labels from. Defaults to `TRUE`. Set to `FALSE` in order to jump straight to passing in your own attributes.

Details

Variables in an SPSS dataset read in with `haven::read_sav()` will contain additional attributes affecting how they are displayed in SPSS, such as variable labels, value labels, and display width. Recoding variables in R will frequently cause loss of attributes, resulting in disappearing variable and value labels if the dataset with recoded variables is written back to SPSS. This function enables quick transfer of said attributes from a variable from a reference dataset to one that is being recoded. This function also allows you to overwrite selected attributes, for example as the result of a recode that collapses categories or creates new ones.

Value

The vector with SPSS attributes set.

Examples

```
# library(dplyr)
# # do not run
# # these examples are illustrative, replace them with your own datasets
# spss_df <- read_sav("spss_df.sav")
# # An expression like this would take a recoded variable such as with dplyr::case_when(),
# # carry over the value labels, format.spss and display_width attributes,
# # and replace the variable label
# spss_df_recoded <- spss_df %>%
#   mutate(new_variable = transfer_spss_labels(case_when(old_variable == 1 ~ 3,
#                                                        old_variable == 3 ~ 1),
#                                              spss_df,
#                                              "old_variable",
#                                             variable_label = "Old variable reversed"))
# # check using the attributes() function
# attributes(spss_df_recoded$new_variable)
# tablena(spss_df_recoded$new_variable)
```

trim_weights *Trim weights*

Description

Trim survey weights. Wrapper for the `trimWeights()` function in the `survey` package that eliminates the need to create a `svydesign` object.

Usage

```
trim_weights(
  weight,
  lower_quantile = 0.01,
  upper_quantile = 0.99,
  minval = NULL,
  maxval = NULL,
  strict = FALSE
)
```

Arguments

`weight`	A numeric vector of weights
`lower_quantile`	The quantile at which the weights should be trimmed on the lower end
`upper_quantile`	The quantile at which the weights should be trimmed on the upper end
`minval`	NULL by default. If the lower quantile is less than the minimum value, minval will be used for the trim.
`maxval`	NULL by default. If the upper quantile is greater than the maximum value, maxval will be used for the trim.
`strict`	When the trimmed weights are reapportioned, some values may exceed the specificed threshold value. If `strict = FALSE` this is permitted. Otherwise, weights are trimmed recursively until none exceed the limit.

Value

A numeric vector of trimmed weights

Examples

```
calculate_deff(dec13_excerpt$weight)
trimmed <- trim_weights(dec13_excerpt$weight, lower_quantile = 0.1, upper_quantile = 0.9)
# Trimming reduces the standard deviation of the weights, reducing reducing the design effect
# and increasing the effective sample size.
calculate_deff(trimmed)
```

`unify_margins` *Unify margins*

Description

This is an internal function called by `rake_survey()` that makes sure that the raking targets and the survey data match up.

Usage

```
unify_margins(pop_margins, .data)
```

Arguments

`pop_margins`	A list of tibbles giving the population margins for raking variables. `create_raking_targets()` outputs such a list.
`.data`	The survey data to be raked.

Details

Ensures that the factor levels for each variable in the targets and in the data are in the same order, and that the list of targets is not named.

Value

Cleaned raking targets.

Index