Package 'labelr'

September 8, 2024

Title Label Data Frames, Variables, and Values

Version 0.1.9

Maintainer Robert Hartman <rohartman@gmail.com>

Description Create and use data frame labels for data frame objects (frame labels), their columns (name labels), and individual values of a column (value labels). Value labels include one-to-one and many-to-one labels for nominal and ordinal variables, as well as numerical range-based value labels for continuous variables. Convert value-labeled variables so each value is replaced by its corresponding value label. Add values-converted-to-labels columns to a value-labeled data frame while preserving parent columns. Filter and subset a value-labeled data frame using labels, while returning results in terms of values. Overlay labels in place of values in common R commands to increase interpretability. Generate tables of value frequencies, with categories expressed as raw values or as labels. Access data frames that show value-to-label mappings for easy reference.

License GPL (>= 3)

URL https://github.com/rhartmano/labelr

Encoding UTF-8

RoxygenNote 7.3.0

Depends R (>= 4.0.0)

Imports stats, utils

Suggests knitr, rmarkdown, car, nycflights13, collapse, tibble, haven, dplyr, modelr, ggplot2

VignetteBuilder knitr

NeedsCompilation no

Author Robert Hartman [aut, cre], The MITRE Corporation [cph]

Repository CRAN

Date/Publication 2024-09-08 20:20:06 UTC

Contents

add1m1 *Apply One Label to Multiple Values for a Single Variable*

Description

Apply a single variable value label to multiple values of a variable ("m1" is shorthand for "many values get one label").

Usage

```
add1m1(data, var, vals, lab, max.unique.vals = 10, init = FALSE)
```

Arguments

data	a data.frame.
var	the unquoted name of the variable (column) to which value labels will be added.
vals	a vector of distinct values of the actual variable, each of which is to be associated with the label supplied to the lab argument. Note: NA and other "irregular" (e.g., NaN, Inf) values all are automatically assigned the label "NA", and this cannot be overridden. Note that you do not need to specify all unique vals of var, and you can supply value labels incrementally, one (or a few, or all) unique vals of var at a time. However, if you do this, do not re-use a value label or repeat a value-label assignment you have already made: Once you've added the value label, it is bound to those values until you drop the label (see drop_val_labs) or some other action (intentional or otherwise) strips the value label attributes from your data.frame (see, e.g. strip_labs).
lab	a single distinct label that will be associated with all values specified in your vals argument. Note: NA and other "irregular" (e.g., NaN, Inf) values are automatically assigned the label "NA" and may not be assigned another label.
max.unique.vals	add1m1() will not assign value labels to non- integer (i.e., decimal-having) numeric variables. The max.unique.vals argument further constrains the variables that may receive value labels to those whose total unique values do not exceed the integer value supplied to this argument.
init	assign placeholder labels for variables that lack decimals and meet the max.unique.vals threshold.

Details

Note 1: add1m1 is a variant of add_m1_lab that allows you to specify only one var to label but allows you to pass its name without quoting it (compare add1m1(mtcars, am, ...) to add_m1_lab(mtcars, "carb", ...).

Note 2: add1m1 (and add_m1_lab) allows the user to assign the same value label to multiple distinct values of a variable ("m1" is short for "many-to-one"). This is in contrast to add_val1 (and add_val_labs), which requires a strict one-to-one mapping of distinct variable values and distinct value labels.

Note 3: This command is intended exclusively for interactive use. In particular, the var argument must be the literal name of a single variable (column) found in the supplied data.frame and may NOT be, e.g., the name of a character vector that contains the variable (column name) of interest. If you wish to supply a character vector with the names of variables (columns) of interest, use `add_m1_lab()`.

Value

A data.frame, with new variable value labels added (call `get_val_labs` to see them), other provisional/default labelr label information added, and previous user-added labelr label information preserved.

Examples

```
df <- mtcars

df <- add1m1(df,
  var = carb,
  vals = 1:3,
  lab = "<=3",
  max.unique.vals = 10
)

df <- add1m1(df,
  var = carb,
  vals = c(4, 6, 8),
  lab = ">=4",
  max.unique.vals = 10
)

head(use_val_labs(df), 8) # they're there
```

`add_factor_info` *Add Factor-specific Attributes to a Data Frame*

Description

`add_factor_info` searches a data.frame for labelr-specific factor meta-data, which it records and retains for future use. It is used by other labelr functions and need not be used directly by labelr end users.

Usage

```
add_factor_info(data)
```

Arguments

`data`	the data.frame to which you wish to add labelr-specific factor variable meta-data attributes (if any factors are present).

Value

a data.frame.

Examples

```
# this function does not strictly require prior or other use of labelr
ir2 <- add_factor_info(iris)
mt2 <- add_factor_info(mtcars)
get_factor_info(mtcars) # none
get_factor_info(iris) # none
get_factor_info(mt2) # none
get_factor_info(ir2) # some!
```

---

add_frame_lab *Add or Modify a Data Frame "Frame Label"*

Description

Add a 500-or-fewer-characters high-level descriptive label for your data.frame as whole (e.g., nature, originator, population / sample, year created, general contents, article citation).

Usage

```
add_frame_lab(data, frame.lab = NULL)

afl(data, frame.lab = NULL)
```

Arguments

data	a data.frame.
frame.lab	quoted text of the descriptive data.frame label that you wish to add to the data.frame.

Details

add_frame_lab assigns an overall descriptive "frame label" for a data.frame, which can be retrieved using get_frame_lab.

Note: afl is a compact alias for add_frame_lab: they do the same thing, and the former is easier to type

Value

A data.frame, with a frame.lab attribute added to the attributes meta-data

Examples

```
# add frame.lab to mtcars and assign to new data.frame mt2
mt2 <- add_frame_lab(mtcars, frame.lab = "Data extracted from the 1974 Motor
                    Trend US magazine, comprising fuel consumption and 10
                    aspects of automobile design and performance for 32
                    automobiles (1973-74 models). Source: Henderson and
                    Velleman (1981), Building multiple regression models
                    interactively. Biometrics, 37, 391-411.")

attr(mt2, "frame.lab") # check for attribute

get_frame_lab(mt2) # return frame.lab alongside data.frame name as a data.frame
```

---

add_lab_atts *Add labelr Attributes from a list to a Data Frame*

Description

`add_lab_atts` allows one to apply a list of labelr label attribute meta-data (created by `get_all_lab_atts`) to a data.frame.

Usage

```
add_lab_atts(
  data,
  lab.atts.list,
  strip.first = FALSE,
  num.convert = FALSE,
  clean = TRUE
)
```

Arguments

`data`	a data.frame object.
`lab.atts.list`	a list previously created using `get_all_lab_atts`.
`strip.first`	FALSE if you do not wish to strip the data.frame of all label attribute information it may already have (this information may still be overwritten, depending on what is in the lab.atts.list list).
`num.convert`	attempt to convert to numeric any data.frame variables where this can be done without producing new NA values.
`clean`	after adding label attributes, put them into a neat, logical order and drop any attributes that describe variables (columns) not present in the data.frame to which they have been added.

Details

See get_all_lab_atts.

add_lab_atts allows one to add or restore label attributes from a free- standing list (created by get_all_lab_atts) to a data.frame.

Value

a data.frame object with label attribute information (re-) attached (if it exists in the specified lab.atts.list).

Examples

```
# make toy demographic (gender, raceth, etc.) data set
set.seed(555)
df <- make_demo_data(n = 1000) # another labelr:: function

# let's add variable VALUE labels for variable "raceth"
df <- add_val_labs(df,
  vars = "raceth", vals = c(1:7),
  labs = c("White", "Black", "Hispanic", "Asian", "AIAN", "Multi", "Other"),
  max.unique.vals = 50
)

get_val_labs(df, "raceth") # it's here

zlab.df <- get_all_lab_atts(df) # back up labelr attributes for df

df <- strip_labs(df) # this removes labs from df

get_val_labs(df, "raceth") # it's gone

check_any_lab_atts(df) # FALSE (means "no labs here")

df <- add_lab_atts(df, zlab.df) # restore them
```

---

add_lab_col1 *Create a Value Labels Column for a Single Variable and Add to the Data Frame*

Description

For a single value-labeled column of a data.frame, create a copy of that column that replaces all of its values with the corresponding value labels and added that copy to the supplied data.frame.

Usage

```
add_lab_col1(data, var, suffix = "_lab")

alc1(data, var, suffix = "_lab")
```

Arguments

data	a data.frame.
var	the unquoted name of the column (variable) whose values you wish to replace with the corresponding value labels.
suffix	a suffix that will be appended to the name of the labels-on column that is added to the data.frame (e.g., if suffix = "_lab," the labels-on version of "x1" will be "x1_lab").

Details

Note 1: add_lab_col1 is a variant of add_lab_cols that allows you to specify only one variable at a time but that allows you to pass its name without quoting it (compare add_lab_col1(mtcars, am) to add_lab_cols(mtcars, "am")).

Note 2: alc1 is a compact alias for add_lab_col1: they do the same thing, and the former is easier to type.

Note 3: This command is intended exclusively for interactive use. In particular, the var argument must be the literal name of a single variable (column) found in the supplied data.frame and may NOT be, e.g., the name of a character vector that contains the variable (column name) of interest. If you wish to supply a character vector with the names of variables (columns) of interest, use add_lab_cols().

add_lab_col1 creates a "labels-on" version of a value-labeled column and adds that new column to the supplied data.frame. Here, "labels-on" means that the column's original values are replaced with the corresponding value labels. Note that this column does not replace but is added to its parent/source columns in the returned data.frame. The resulting "labels-on" column is a simple, self-contained character column that cannot itself be converted or reverted to the original ("labels-off") values of its parent/source column. See add_lab_cols for a list of other functions that may be useful in working with value labels.

Value

A data.frame consisting of the originally supplied data.frame, along with the labels-on column added to it.

Examples

```
# add "labels-on" version of "am" to copy of mtcars
df <- mtcars # copy of mtcars

# now, add value labels
df <- add_val1(
  data = df,
  var = am,
  vals = c(0, 1),
  labs = c("automatic", "manual")
)

# add value labels-on version of "am" to df, assign to df_plus
df_plus <- add_lab_col1(df, am)
head(df_plus[c("am", "am_lab")])
```

add_lab_cols *Add Variable Value Label Columns to a Data Frame*

Description

Add copies of value-labeled columns to a data.frame, where the new columns' values are replaced with the corresponding value labels.

Usage

```
add_lab_cols(data, vars = NULL, suffix = "_lab")

alc(data, vars = NULL, suffix = "_lab")
```

Arguments

data	a data.frame.
vars	the names of the columns (variables) for which "labels-on" (values replaced with value labels) versions of the variable will be added to the returned data.frame.
suffix	a suffix that will be added to the names of all labels-on variables added to the data.frame (the non-suffix portion of the variable name will be identical to the original variable, e.g., the labels-on version of "x1" will be "x1_lab" (or whatever alternative suffix you supply)).

Details

Note: alc is a compact alias for add_lab_cols: they do the same thing, and the former is easier to type.

add_lab_cols adds one or more "labels-on" columns to a data.frame, where "labels-on" means that the column's original values are replaced with the corresponding value labels. Note that these columns do not replace but are added to their parent/source columns in the returned data.frame. The resulting "labels-on" columns are simple, self-contained character columns that cannot themselves be converted or reverted to the original ("labels-off") values of their parent/source columns.

For other ways of accessing or leveraging value labels, see, e.g., use_val_labs, val_labs_vec, add_lab_dummies, lab_int_to_factor, flab, slab, get_val_labs, with_val_labs, headl, taill, somel, and tabl. In particular, see use_val_labs if, rather than adding a "labels-on" column to a data.frame, you wish to replace a column's values with the corresponding value labels. See val_labs_vec if you wish to convert a single, value-labeled column's values to labels and return the result as a stand-alone vector.

Value

A data.frame consisting of the originally supplied data.frame, along with (all or the select) labels-on variable versions added to it.

Examples

```
# one variable at a time, mtcars
df <- mtcars
# now, add value labels
df <- add_val_labs(
  data = df,
  vars = "am",
  vals = c(0, 1),
  labs = c("automatic", "manual")
)

df <- add_val_labs(
  data = df,
  vars = "carb",
  vals = c(1, 2, 3, 4, 6, 8),
  labs = c(
    "1-carb", "2-carbs",
    "3-carbs", "4-carbs",
    "6-carbs", "8-carbs"
  )
)

# var arg can be unquoted if using add_val1()
# note that this is not add_val_labs(); add_val1() has "var" (not "vars) arg
df <- add_val1(
  data = df,
  var = cyl, # note, "var," not "vars" arg
  vals = c(4, 6, 8),
  labs = c(
    "four-cyl",
    "six-cyl",
    "eight-cyl"
  )
)

df <- add_val_labs(
  data = df,
  vars = "gear",
  vals = c(3, 4),
  labs = c(
    "3-speed",
    "4-speed"
  )
)

# Oops, we forgot 5-speeds; let's finish the job.
df <- add_val_labs(
  data = df,
  vars = "gear",
  vals = 5,
  labs = "5-speed"
)
```

```
# add value labels-on versions of the foregoing to df and return as "df_plus"
df_plus <- add_lab_cols(df)
head(df_plus)
head(df_plus[c("am", "am_lab")])
```

---

add_lab_dumm1 *Add A Dummy Variable for Each Value Label of a Single Variable*

Description

For a single value-labeled data.frame column, create a dummy (aka indicator) variable for each of that column's unique value labels.

Usage

```
add_lab_dumm1(
  data,
  var,
  simple.names = TRUE,
  sep = "_",
  prefix.length = 4,
  suffix.length = 7
)

ald1(
  data,
  var,
  simple.names = TRUE,
  sep = "_",
  prefix.length = 4,
  suffix.length = 7
)
```

Arguments

data	a data.frame with at least one value-labeled variable (column).
var	the unquoted name of the value-labeled variable (column) from which dummy variable columns will be generated.
simple.names	if TRUE (the default), dummy variable names will be the parent variable's name, followed by the sep separator (see above), followed by an automatically generated numerical id suffix. For example two dummy variable columns created from value-labeled column "tacos" using the sep argument of "." would be given the respective names "tacos.1" and "tacos.2").
sep	the separator character to use in constructing dummy variable column names (appears between the dummy variable name prefix and suffix).

`prefix.length` (NOTE: This argument is ignored if simple.names = TRUE). A 1L integer indicating the number of leading characters of the parent column's name to use in constructing dummy variable column names. For example, if simple.names = FALSE, if prefix.length = 2, and for a parent column named "tacos", each derivative dummy variable column name will begin with the prefix string "ta," (corresponding to the first two characters of "tacos"), followed by the sep separator character (see sep param, above), followed by the suffix string (see suffix.length param, below).

`suffix.length` (NOTE: This argument is ignored if simple.names = TRUE). A 1L integer indicating the number of leading characters of each variable value label to use use in constructing dummy variable column names. For example, consider the following setup: parent column name is "tacos"; prefix.length = 3; sep = "_", and suffix.length = 2. In this case, if simple.names = FALSE, then a dummy variable column named "tac_so" would be created to represent those values of the tacos" column that have the value label "soft" (because "tac" are the first three letters of the parent column name, the separator is "_", and "so" are the first two characters in "soft").

Details

Note 1: `add_lab_dumm1` is a variant of `add_lab_dummies` that allows you to specify only one var to label at a time but that allows you to pass its name without quoting.

Note 2: `ald1` is a compact alias for `add_lab_dumm1`: they do the same thing, and the former is easier to type

Note 3: If the default of simple.names is used, dummy variable column names will be the "parent" variable column name, followed by a separator character (by default, "_"), followed by a number, to differentiate each dummy variable from the others in the set. If one of the automatically generated dummy column names is already "taken" by a pre-existing data.frame column, an error to this effect will be thrown. If simple.names = FALSE, then prefix.length and suffix.length arguments will be used to construct dummy variable column names using the leading characters of the parent column name, followed by a separator character, followed by the leading characters of the value label. (white spaces in the value label will be replaced with the separator character).

Note 4: This command is intended exclusively for interactive use. In particular, the var argument must be the literal name of a single variable (column) found in the supplied data.frame and may NOT be, e.g., the name of a character vector that contains the variable (column name) of interest. If you wish to supply a character vector with the names of variables (columns) of interest, use `add_lab_dummies()`.

Value

A data.frame with dummy variables added for all value labels of the value-labeled column supplied to the var argument.

Examples

```
# one variable at a time, mtcars
df <- mtcars
```

```
# now, add 1-to-1 value labels
df <- add_val_labs(
  data = df,
  vars = "am",
  vals = c(0, 1),
  labs = c("automatic", "manual")
)

df <- add_val_labs(
  data = df,
  vars = "carb",
  vals = c(1, 2, 3, 4, 6, 8),
  labs = c(
    "1-carb", "2-carbs",
    "3-carbs", "4-carbs",
    "6-carbs", "8-carbs"
  )
)

# var arg can be unquoted if using add_val1()
# note that this is not add_val_labs(); add_val1() has "var" (not "vars) arg
df <- add_val1(
  data = df,
  var = cyl, # note, "var," not "vars" arg
  vals = c(4, 6, 8),
  labs = c(
    "four-cyl",
    "six-cyl",
    "eight-cyl"
  )
)

# add many-to-1 value labels
df <- add_m1_lab(
  data = df,
  vars = "gear",
  vals = 4:5,
  lab = "4+"
)

# add quartile-based numerical range value labels
df <- add_quant_labs(
  data = df,
  vars = "disp",
  qtiles = 4
)

# add "pretty" cut-based numerical range value labels
(mpg_bins <- pretty(range(df$mpg, na.rm = TRUE)))

df <- add_quant_labs(data = df, vars = "mpg", vals = mpg_bins)

# add dummy variables for value labels of column "mpg"
```

```
df1 <- add_lab_dumm1(df,
  var = mpg,
  simple.names = TRUE
) # simple.names = TRUE is default
df1

# add dummy variables for value labels of column "am"
df2 <- add_lab_dumm1(df, am,
  sep = ".", simple.names = FALSE,
  prefix.length = 2, suffix.length = 6
)
df2
```

---

add_lab_dummies *Add A Dummy Variable for Each Value Label*

Description

For one or more value-labeled data.frame columns, create a dummy (aka indicator) variable for each unique value label.

Usage

```
add_lab_dummies(
  data,
  vars,
  simple.names = TRUE,
  sep = "_",
  prefix.length = 4,
  suffix.length = 7
)

ald(
  data,
  vars,
  simple.names = TRUE,
  sep = "_",
  prefix.length = 4,
  suffix.length = 7
)
```

Arguments

data	a data.frame with at least one value-labeled variable (column).
vars	the value-labeled variable or variables from which dummy variable columns will be generated (variable names must be quoted).

`simple.names` if TRUE (the default), dummy variable names will be the parent variable's name, followed by the sep separator (see above), followed by an automatically generated numerical id suffix. For example two dummy variable columns created from value-labeled column "tacos" using the sep argument of "." would be given the respective names "tacos.1" and "tacos.2").

`sep` the separator character to use in constructing dummy variable column names (appears between the dummy variable name prefix and suffix).

`prefix.length` (NOTE: This argument is ignored if simple.names = TRUE). A 1L integer indicating the number of leading characters of the parent column's name to use in constructing dummy variable column names. For example, if simple.names = FALSE, if prefix.length = 2, and for a parent column named "tacos", each derivative dummy variable column name will begin with the prefix string "ta," (corresponding to the first two characters of "tacos"), followed by the sep separator character (see sep param, above), followed by the suffix string (see suffix.length param, below).

`suffix.length` (NOTE: This argument is ignored if simple.names = TRUE). A 1L integer indicating the number of leading characters of each variable value label to use in constructing dummy variable column names. For example, consider the following setup: parent column name is "tacos"; prefix.length = 3; sep = "*", and suffix.length = 2. In this case, if simple.names = FALSE, then a dummy variable column named "tac_so" would be created to represent those values of the tacos" column that have the value label "soft" (because "tac" are the first three letters of the parent column name, the separator is "*", and "so" are the first two characters in "soft").

Details

If the default of simple.names is used, dummy variable column names will be the "parent" variable column name, followed by a separator character (by default, "_"), followed by a number, to differentiate each dummy variable from the others in the set. If one of the automatically generated dummy column names is already "taken" by a pre-existing data.frame column, an error to this effect will be thrown. If simple.names = FALSE, then prefix.length and suffix.length arguments will be used to construct dummy variable column names using the leading characters of the parent column name, followed by a separator character, followed by the leading characters of the value label. (white spaces in the value label will be replaced with the separator character).

Note: `ald()` is an alias function that behaves identically to `add_lab_dummies`.

Value

A data.frame with dummy variables added for all value labels of the value-labeled columns supplied to the vars argument.

Examples

```
# one variable at a time, mtcars
df <- mtcars

# now, add 1-to-1 value labels
```

```
df <- add_val_labs(
  data = df,
  vars = "am",
  vals = c(0, 1),
  labs = c("automatic", "manual")
)

df <- add_val_labs(
  data = df,
  vars = "carb",
  vals = c(1, 2, 3, 4, 6, 8),
  labs = c(
    "1-carb", "2-carbs",
    "3-carbs", "4-carbs",
    "6-carbs", "8-carbs"
  )
)

# var arg can be unquoted if using add_val1()
# note that this is not add_val_labs(); add_val1() has "var" (not "vars) arg
df <- add_val1(
  data = df,
  var = cyl, # note, "var," not "vars" arg
  vals = c(4, 6, 8),
  labs = c(
    "four-cyl",
    "six-cyl",
    "eight-cyl"
  )
)

# add many-to-1 value labels
df <- add_m1_lab(
  data = df,
  vars = "gear",
  vals = 4:5,
  lab = "4+"
)

# add quartile-based numerical range value labels
df <- add_quant_labs(
  data = df,
  vars = "disp",
  qtiles = 4
)

# add "pretty" cut-based numerical range value labels
(mpg_bins <- pretty(range(df$mpg, na.rm = TRUE)))

df <- add_quant_labs(data = df, vars = "mpg", vals = mpg_bins)

# add dummy variables for the labels of column "am"
df2 <- add_lab_dummies(df, "am",
```

```
  sep = ".", simple.names = FALSE,
  prefix.length = 2, suffix.length = 6
)
df2

# add dummy variables for the labels of columns "mpg", "gear", and "cyl
df3 <- add_lab_dummies(df, c("mpg", "gear", "cyl"), simple.names = TRUE) # default
df3
```

---

add_m1_lab *Apply One Label to Multiple Values*

Description

Apply a single variable value label to multiple values of a variable ("m1" is shorthand for "many values get one label").

Usage

```
add_m1_lab(
  data,
  vars,
  vals,
  lab,
  partial = FALSE,
  not.vars = NULL,
  max.unique.vals = 10,
  init = FALSE
)

am1l(
  data,
  vars,
  vals,
  lab,
  partial = FALSE,
  not.vars = NULL,
  max.unique.vals = 10,
  init = FALSE
)
```

Arguments

data	a data.frame.
vars	a character vector that corresponds to the name(s) of one or more variables to which value labels will be added.

vals — a vector of distinct values of the actual variable, each of which is to be associated with the single label supplied to the lab argument. Note: NA and other "irregular" (e.g., NaN, Inf) values all are automatically assigned the label "NA", and this cannot be overridden.

lab — a single distinct label that will be associated with all values specified in your vals argument. Note: NA and other "irregular" (e.g., NaN, Inf) values are automatically assigned the label "NA" and may not be assigned another label.

partial — To apply the same value labeling scheme to many variables at once, you can provide those variable names explicitly (e.g., vars = c("x1","x2", "x3") or vars = paste0("x", 1:3), or you can provide a substring only and set partial = TRUE (default is FALSE). For example, to apply the same labeling scheme to vars "x1", "x2" ... sequentially through "x10", you could use vars = c("x"), along with partial = TRUE. Be careful with this, as it also will attempt to apply the scheme to "sex" or "tax.bracket", etc.

not.vars — use of the partial argument can result in situations where you inadvertently attempt to value-label a variable. For example, if vars="x" and partial=TRUE, then add_m1_lab will attempt to label not only "x1", "x2","x3", and "x4", but also "sex", "tax.bracket.", and other "x"-containing variable names. Use of not.vars allows you to indicate variables that match your vars argument that you do not wish to attempt to value-label. Note that not.vars gets priority: setting vars="x", partial=TRUE, and not.vars="x" is tantamount to telling add_m1_lab() that you actually do not wish to label any of the variables that you specified in vars, resulting in no variables receiving value labels.

max.unique.vals — add_m1_lab() will not assign value labels to non- integer (i.e., decimal-having) numeric variables. The max.unique.vals argument further constrains the variables that may receive value labels to those whose total unique values do not exceed the integer value supplied to this argument. Note that labelr sets a hard ceiling of 5000 on the total number of unique value labels that any variable is permitted to have under any circumstance, as labelr is primarily intended for interactive use with moderately-sized (<=~1M-row) data.frames.

init — assign placeholder labels for variables that lack decimals and meet the max.unique.vals threshold.

Details

' add_m1_lab(andadd1m1) allows the user to assign the same value label to multiple distinct values of a v
which require a strict one-to-one mapping of distinct variable values and distinct value labels.

Note 1: Each call to add_m1_lab accepts only one value label, which may be applied to multiple distinct values of the specified column(s). Additional labels can be applied to other values of the same column(s) by making additional calls to add_m1_lab (see the example).

Note 2: am1l is a compact alias for add_m1_lab: they do the same thing, and the former is easier to type

Value

A data.frame, with new variable value labels added (call get_val_labs to see them), other provisional/default labelr label information added, and previous user-added labelr label information

preserved.

Examples

```
df <- mtcars

df <- add_m1_lab(df,
  vars = "carb",
  vals = 1:3,
  lab = "<=3",
  max.unique.vals = 10
)

df <- add_m1_lab(df,
  vars = "carb",
  vals = c(4, 6, 8),
  lab = ">=4",
  max.unique.vals = 10
)

get_val_lab1(df, carb)

head(use_val_labs(df), 8) # they're there
```

---

add_name_labs *Add or Modify Data Frame Variable Name Labels*

Description

Add descriptive variable name labels (up to one per column) to the columns of a data.frame.

Usage

```
add_name_labs(
  data,
  name.labs = NULL,
  vars = NULL,
  labs = NULL,
  init.max = NULL
)

anl(data, name.labs = NULL, vars = NULL, labs = NULL, init.max = NULL)
```

Arguments

data	the data.frame you wish to begin labeling.
name.labs	a named character vector, where names are current data.frame column (variable) names, and where values are the proposed labels. If this is NULL, vars and labs arguments may not be NULL. If latter are not NULL, this (name.labs) argument must be NULL.

vars	the names of the columns (variables) to which name labels will be applied. If NULL, labs arg must also be NULL and name.labs cannot be NULL.
labs	the proposed variable name labels to applied to the columns (variables). If non-NULL, vars arg must also be non-NULL.
init.max	If non-NULL, this must be a 1L integer, indicating the maximum number of unique values that a variable may have for it to receive placeholder value labels, which will consist of the variable's actual values coerced to character values. If NULL, or if the variable is numeric with decimal values, the variable will not be given initialized variable value labels.

Details

Note: `anl` is a compact alias for `add_name_labs`: they do the same thing, and the former is easier to type.

`add_name_labs` works with `get_name_labs`, `use_name_labs`, and `use_var_names` to facilitate the creation, accessing, and substitution of variable name labels for variable names.

Each variable (column) of a data.frame can receive one and only one "name label," which typically is a noun phrase that expounds the meaning of contents of the variable's name (e.g., "Weight in ounces at birth" might be a name label for a column called "wgt"). `add_name_labs` takes a data.frame and either a named character vector (names are current variable names, values are proposed name labels) supplied to the name.labs arg or two separate character vectors (one each for current variable names and proposed variable name labels, respectively) supplied to vars and labs args, respectively. If using the second approach, the order of each entry matters (e.g., the first variable name entry to the vars argument will be given the label of the first name label entry to the labs argument, and so on).

Note that any non-name-labeled columns will receive their own names as default name labels (e.g., if var "mpg" of mtcars is not assigned a name label, it will be given the default name label of "mpg"). Note also that other labelr functions (e.g., `add_val_labs`) will initialize name labels and other labelr attribute meta-data in this same fashion. Name labels can be removed with `drop_name_labs`.

Value

A data.frame, with new name labels added (call `get_name_labs` to see them), other provisional/default labelr label information added, and previous user-added labelr label information preserved.

Examples

```
# create a data set
df <- mtcars

# variable names and their labels
names_labs_vec <- c(
  "mpg" = "Miles/(US) gallon",
  "cyl" = "Number of cylinders",
  "disp" = "Displacement (cu.in.)",
  "hp" = "Gross horsepower",
  "drat" = "Rear axle ratio",
  "wt" = "Weight (1000 lbs)",
  "qsec" = "1/4 mile time",
```

```
  "vs" = "Engine (0 = V-shaped, 1 = straight)",
  "am" = "Transmission (0 = automatic, 1 = manual)",
  "gear" = "Number of forward gears",
  "carb" = "Number of carburetors"
)

# assign variable labels
df <- add_name_labs(df,
  vars = names(names_labs_vec),
  labs = names_labs_vec
)

# see what we have
get_name_labs(df)

# use these
df_labs_as_names <- use_name_labs(df)
head(df_labs_as_names)[1:3] # these are verbose, so, only show first three
head(df)[1:3]

# now revert back
df_names_as_before <- use_var_names(df_labs_as_names)
head(df_names_as_before)[1:3] # indeed, they are as before
identical(head(df), head(df_names_as_before))

# strip name label meta-data information from df
# NOT same as use_var_names(), which preserves the info but "turns it off"
# this strips the name labels meta-data from df altogether
df <- drop_name_labs(df)

# see what we have
get_name_labs(df) # they're gone

# alternative syntax (if you have a named vector like names_labs_vec)
# assign variable name labels
df <- add_name_labs(df,
  name.labs = c(
    "mpg" = "Miles/(US) gallon",
    "cyl" = "Number of cylinders",
    "disp" = "Displacement (cu.in.)",
    "hp" = "Gross horsepower",
    "drat" = "Rear axle ratio",
    "wt" = "Weight (1000 lbs)",
    "qsec" = "1/4 mile time",
    "vs" = "Engine (0 = V-shaped, 1 = straight)",
    "am" = "Transmission (0 = automatic, 1 = manual)",
    "gear" = "Number of forward gears",
    "carb" = "Number of carburetors"
  )
)

# replace two variable name labels, keeping the others
df <- add_name_labs(df,
```

```
  name.labs = c(
    "disp" = toupper("displacement"),
    "mpg" = toupper("miles per gallon")
  )
)

attributes(df) # show all attributes
get_name_labs(df) # show only the variable name labels
get_name_labs(df, var = c("disp", "mpg"))

# again, strip name label meta-data information from df
# NOT same as use_var_names(), which preserves the info but "turns it off"
df <- drop_name_labs(df)

# see what we have
get_name_labs(df) # they're gone

# alternative syntax to add name labels
df <- add_name_labs(df,
  vars = c("carb", "am"),
  labs = c("how many carburetors?", "automatic or stick?")
)

# see what we have
get_name_labs(df) # they're back! (and placeholders for others)

# add another
df <- add_name_labs(df,
  vars = c("mpg"),
  labs = c("miles per gallon, of course")
)

# see what we have
get_name_labs(df) # it's been added, and others preserved

head(use_name_labs(df)[c(1, 9, 11)]) # verbose, but they're there
```

---

add_quant1 *Associate Numerical Threshold-based Value Labels with a Single Numerical Variable*

Description

Add variable-specific value labels based on threshold cuts of a single numerical variable.

Usage

```
add_quant1(data, var, qtiles = NULL, vals = NULL, labs = NULL)

aql1(data, var, qtiles = NULL, vals = NULL, labs = NULL)
```

Arguments

data	a data.frame.
var	the unquoted name of the variable to which value labels will be added.
qtiles	the number of quantile categories to employ (e.g., 4 would indicate quartiles, 5 would indicate quintiles, 10 for deciles, etc.). If NULL, vals must be non-NULL.
vals	one more values of var that will define range cutpoints, such that all values at or below a given number and above the preceding val will be treated as part of the same numerical range for labeling purposes. If NULL, qtiles must be non-NULL.
labs	a character vector of distinct labels to identify the quantiles. If left NULL, convention "q" + quantile (e.g., "q10") will be used for qtile-based labels (i.e., if qtiles arg is non-NULL), and convention "<=" + val will be used for vals argument-based labels (i.e., if vals arg is non-NULL). Note that the labels "NA" and "Other" are (non-case-sensitively) reserved and may not be user-supplied.

Details

add_quant1 is a variant of add_quant_labs that allows you to specify only one var to label but allows you to pass its name without quoting it (compare add_quant1(mtcars, mpg) to add_quant_labs(mtcars, "mpg").

Numerical variables that feature decimals or large numbers of distinct values are not eligible to receive conventional value labels. add_quant1 allows one to label such variables according to user-supplied value thresholds (i.e., cutpoints) OR quantile membership, Thus, unlike value labels added with add_val_labs (and add_val1), add_quant1 (and add_quant_labs) will apply the same value label to all values that fall within the numerical value range defined by each threshold (cutpoint). For still another value-labeling approach, see add_m1_lab (and add1m1).

Note 1: Quantity labels cannot be added incrementally through repeated calls to add_quant1: each new call will overwrite all value labels applied to the specified vars in any previous add_quant1 calls. This is in contrast to add_val_labs (which allows for incremental value-labeling) and add_m1_lab (which requires incremental value-labeling).

Note 2: aql1 is a compact alias for add_quant1: they do the same thing, and the former is easier to type

Note 3: This command is intended exclusively for interactive use. In particular, the var argument must be the literal name of a single variable (column) found in the supplied data.frame and may NOT be, e.g., the name of a character vector that contains the variable (column name) of interest. If you wish to supply a character vector with the names of variables (columns) of interest, use add_quant_labs().

Value

A data.frame, with new variable value labels added (call get_val_labs to see them), other provisional/default labelr label information added, and previous user-added labelr label information preserved.

Examples

```
# mtcars demo
df <- mtcars
# now, add value labels
df <- add_val_labs(
  data = df,
  vars = "am",
  vals = c(0, 1),
  labs = c("automatic", "manual")
)

# label variable "mpg" in terms of 5 quintiles
df <- add_quant1(data = df, mpg, qtiles = 5)

# label variable "disp" in terms of "pretty" cutpoints
vals2use <- pretty(c(min(df$disp), max(df$disp)))[-1] # establish cutpoints
df <- add_quant1(data = df, disp, vals = vals2use)
df_labson <- use_val_labs(df)
head(df_labson)
```

---

add_quant_labs *Associate Numerical Threshold-based Value Labels with Select Numerical Variables*

Description

Add variable-specific value labels based on threshold cuts of a numerical variable.

Usage

```
add_quant_labs(
  data,
  vars,
  qtiles = NULL,
  vals = NULL,
  labs = NULL,
  partial = FALSE,
  not.vars = NULL
)

aql(
  data,
  vars,
  qtiles = NULL,
  vals = NULL,
  labs = NULL,
  partial = FALSE,
  not.vars = NULL
)
```

Arguments

data	a data.frame.
vars	a character vector that corresponds to the name(s) of one or more variables to which value threshold-based labels will be added.
qtiles	the number of quantile categories to employ (e.g., 4 would indicate quartiles, 5 would indicate quintiles, 10 for deciles, etc.). If NULL, vals must be non-NULL.
vals	one more values of vars that will define range cutpoints, such that all values at or below a given number and above the preceding val will be treated as part of the same numerical range for labeling purposes. If NULL, qtiles must be non-NULL.
labs	a character vector of distinct labels to identify the quantiles. If left NULL, convention "q" + quantile (e.g., "q10") will be used for qtile-based labels (i.e., if qtiles arg is non-NULL), and convention "<=" + val will be used for vals argument-based labels (i.e., if vals arg is non-NULL). Note that the labels "NA" and "Other" are (non-case-sensitively) reserved and may not be user-supplied.
partial	To apply the same numerical value labeling scheme to many variables at once, you can provide those variable names explicitly (e.g., vars = c("x1","x2", "x3") or vars = paste0("x", 1:3), or you can provide a substring only and set partial = TRUE (default is FALSE). For example, to apply the same labeling scheme to vars "x1", "x2" ... sequentially through "x10", you could use vars = c("x"), along with partial = TRUE. Be careful with this, as it also will attempt to apply the scheme to "sex" or "tax.bracket", etc. (See not.vars argument for a way to mitigate this.)
not.vars	use of the partial argument can result in situations where you inadvertently attempt to value-label a variable. For example, if vars="x" and partial=TRUE, then add_quant_labs will attempt to label not only "x1", "x2","x3", and "x4", but also "sex", "tax.bracket.", and other "x"-containing variable names. Use of not.vars allows you to indicate variables that match your vars argument that you do not wish to attempt to value-label. Note that not.vars gets priority: setting vars="x", partial=TRUE, and not.vars="x" is tantamount to telling add_val_labs() that you actually do not wish to label any of the variables that you specified in vars, resulting in no variables receiving value labels.

Details

Note: aql is a compact alias for add_quant_labs: they do the same thing, and the former is easier to type.

Numerical variables that feature decimals or large numbers of distinct values are not eligible to receive conventional value labels. add_quant_labs allows one to label such variables according to user-supplied value thresholds (i.e., cutpoints) OR quantile membership, Thus, unlike value labels added with add_val_labs (and add_val1), add_quant_labs (and add_quant1) will apply the same value label to all values that fall within the numerical value range defined by each threshold (cutpoint). For still another value-labeling approach, see add_m1_lab (and add1m1).

Note: Quantity labels cannot be added incrementally through repeated calls to add_quant_labs: each new call will overwrite all value labels applied to the specified vars in any previous add_quant_labs calls. This is in contrast to add_val_labs (which allows for incremental value-labeling) and add_m1_lab (which requires incremental value-labeling).

Value

A data.frame, with new variable value labels added (call `get_val_labs` to see them), other provisional/default labelr label information added, and previous user-added labelr label information preserved.

Examples

```
# mtcars demo
df <- mtcars
# now, add value labels
df <- add_val_labs(
  data = df,
  vars = "am",
  vals = c(0, 1),
  labs = c("automatic", "manual")
)

# label variable "mpg" in terms of 5 quintiles
df <- add_quant_labs(data = df, vars = "mpg", qtiles = 5)

# label variable "disp" in terms of "pretty" cutpoints
vals2use <- pretty(c(min(df$disp), max(df$disp)))[-1] # establish cutpoints
df <- add_quant_labs(data = df, vars = "disp", vals = vals2use)
df_labson <- use_val_labs(df)
head(df_labson)
```

---

add_val1 *Add or Modify a Single Variable's Value Labels*

Description

Add variable value-specific, descriptive value labels to a data.frame.

Usage

```
add_val1(data, var, vals, labs, max.unique.vals = 10, init = FALSE)

avl1(data, var, vals, labs, max.unique.vals = 10, init = FALSE)
```

Arguments

`data`	a data.frame.
`var`	the unquoted name of the variable (column) to which value labels will be added.
`vals`	a vector of distinct values of the actual variable, each of which is to be associated with a label supplied to the labs argument in the same positional order (e.g., vals = c(1,0), labs = c("manual", "automatic") will associate lab "manual" with val 1 and lab "automatic" with val 0.). Note: NA and other "irregular" (e.g., NaN, Inf) values all are automatically assigned the label "NA", and this cannot be

overridden. Note that you do not need to specify all unique vals of var, and you can supply value labels incrementally, one (or a few, or all) unique vals of var at a time. Once you've added the value label, it is bound to that value until you drop it (see `drop_val_labs`) or some other action (intentional or otherwise) strips or overwrites it.

`labs` a character vector of distinct label values, each of which is to be associated with exactly one corresponding distinct value (vals argument element) of the variable identified in the var argument. The order of labs argument must match that of vals argument entries (e.g., if a three-element vector of values is supplied to vals, then a three- element vector of proposed labels must be supplied to labs, and the first value of vals will get the first label of labs, the second value of vals will get the second label of labs, etc.). Note: NA and other "irregular" (e.g., NaN, Inf) values are automatically assigned the label "NA" and may not be assigned another label.

`max.unique.vals`
`add_val1()` will not assign value labels to non- integer (i.e., decimal-having) numeric variables. The max.unique.vals argument further constrains the variables that may receive value labels to those whose total unique values do not exceed the integer value supplied to this argument. Note that labelr sets a hard ceiling of 5000 on the total number of unique value labels that any variable is permitted to have under any circumstance, as labelr is primarily intended for interactive use with moderately-sized (<=~1M-row) data.frames.

`init` assign placeholder labels for variables that lack decimals and meet the max.unique.vals threshold.

Details

`add_val1` is intended for associating value labels with binary, nominal, or ordinal (e.g., integer) variables, where each of a limited number of distinct values is to be associated one-to-one with a distinct value label. To assign labels to ranges of numerical variables, see `add_quant_labs` (or `add_quant1`). To apply the same label to multiple distinct values of a variable, see `add_m1_lab` or `add1m1`.

`add_val1` works with other labelr functions (e.g., `add_val_labs`, `drop_val_labs`, `get_val_labs`, `use_val_labs`, `add_lab_cols`) to facilitate the creation, accessing, modification, use, or deletion of variable value labels.

Note 1: `add_val1` is a variant of `add_val_labs` that allows you to specify only one var to label at a time but that allows you to pass its name without quoting it (compare add_val1(mtcars, am) to add_val_labs(mtcars, "am").

Note 2: `avl1` is a compact alias for `add_val1`: they do the same thing, and the former is easier to type

Note 3: This command is intended exclusively for interactive use. In particular, the var argument must be the literal name of a single variable (column) found in the supplied data.frame and may NOT be, e.g., the name of a character vector that contains the variable (column name) of interest. If you wish to supply a character vector with the names of variables (columns) of interest, use `add_val_labs()`.

Value

A data.frame, with new name labels added (call `get_val_labs` to see them), other provisional/default labelr label information added, and previous user-added labelr label information preserved.

Examples

```
# one variable at a time, mtcars
df <- mtcars
# add value labels
# first, using add_val_labs() -- add_val1() example is below
df <- add_val_labs(
  data = df,
  vars = "carb", # note, vars arg; add_val1() takes var arg
  vals = c(1, 2, 3, 4, 6, 8),
  labs = c(
    "1-carb", "2-carbs",
    "3-carbs", "4-carbs",
    "6-carbs", "8-carbs"
  )
)

# now, using add_val1(), where single var arg can be unquoted (cyl, not "cyl")
# note that this is not add_val_labs();
df <- add_val1(
  data = df,
  var = cyl, # note, var arg, not vars arg
  vals = c(4, 6, 8),
  labs = c(
    "four-cyl",
    "six-cyl",
    "eight-cyl"
  )
)
```

`add_val_labs` *Add or Modify a Variable's Value Labels*

Description

Add variable value-specific, descriptive value labels to a data.frame.

Usage

```
add_val_labs(
  data,
  vars,
  vals,
  labs,
```

```
  partial = FALSE,
  not.vars = NULL,
  max.unique.vals = 10,
  init = FALSE
)

avl(
  data,
  vars,
  vals,
  labs,
  partial = FALSE,
  not.vars = NULL,
  max.unique.vals = 10,
  init = FALSE
)
```

Arguments

data	a data.frame.
vars	a character vector that corresponds to the name(s) of one or more variables to which value labels will be added.
vals	a vector of distinct values of the actual variable, each of which is to be associated with a label supplied to the labs argument in the same positional order (e.g., vals = c(1,0), labs = c("manual", "automatic") will associate lab "manual" with val 1 and lab "automatic" with val 0.). Note: NA and other "irregular" (e.g., NaN, Inf) values all are automatically assigned the label "NA", and this cannot be overridden. Note that you do not need to specify all unique vals of var, and you can supply value labels incrementally, one (or a few, or all) unique vals of var at a time. Once you've added the value label, it is bound to that value until you drop it (see `drop_val_labs`) or some other action (intentional or otherwise) strips or overwrites it.
labs	a character vector of distinct label values, each of which is to be associated with exactly one corresponding distinct value (vals argument element) of the variable(s) identified in the vars argument. The order of labs argument must match that of vals argument entries (e.g., if a three-element vector of values is supplied to vals, then a three-element vector of proposed labels must be supplied to labs, and the first value of vals will get the first label of labs, the second value of vals will get the second label of labs, etc.). Note: NA and other "irregular" (e.g., NaN, Inf) values are automatically assigned the label "NA" and may not be assigned another label.
partial	To apply the same value labeling scheme to many variables at once, you can provide those variable names explicitly (e.g., vars = c("x1","x2", "x3") or vars = paste0("x", 1:3), or you can provide a substring only and set partial = TRUE (default is FALSE). For example, to apply the same labeling scheme to vars "x1", "x2" ... sequentially through "x10", you could use vars = c("x"), along with partial = TRUE. Be careful with this, as it also will attempt to apply the scheme to "sex" or "tax.bracket", etc.

not.vars — use of the partial argument can result in situations where you inadvertently attempt to value-label a variable. For example, if vars="x" and partial=TRUE, then add_val_labs will attempt to label not only "x1", "x2","x3", and "x4", but also "sex", "tax.bracket.", and other "x"-containing variable names. Use of not.vars allows you to indicate variables that match your vars argument that you do not wish to attempt to value-label. Note that not.vars gets priority: setting vars="x", partial=TRUE, and not.vars="x" is tantamount to telling add_val_labs() that you actually do not wish to label any of the variables that you specified in vars, resulting in no variables receiving value labels.

max.unique.vals — add_val_labs() will not assign value labels to non- integer (i.e., decimal-having) numeric variables. The max.unique.vals argument further constrains the variables that may receive value labels to those whose total unique values do not exceed the integer value supplied to this argument. Note that labelr sets a hard ceiling of 5000 on the total number of unique value labels that any variable is permitted to have under any circumstance, as labelr is primarily intended for interactive use with moderately-sized (<=~1M-row) data.frames.

init — assign placeholder labels for variables that lack decimals and meet the max.unique.vals threshold.

Details

Note: avl is a compact alias for add_val_labs: they do the same thing, and the former is easier to type

add_val_labs is intended for associating value labels with binary, nominal, or ordinal (e.g., integer) variables, where each of a limited number of distinct values is to be associated one-to-one with a distinct value label. To assign labels to ranges of numerical variables, see add_quant_labs (or add_quant1). To apply the same label to multiple distinct values of a variable, see add_m1_lab or add1m1.

add_val_labs works with other labelr functions (e.g., add_val1, drop_val_labs, get_val_labs, use_val_labs, add_lab_cols) to facilitate the creation, accessing, modification, use, or deletion of variable value labels.

When using add_val_labs or add_val1, each distinct variable value can receive one and only one value label, and for any given variable, each unique label can be assigned to only one unique value (e.g., mtcars$gear==3 and mtcars$gear==4 cannot both share a single "3 or 4 gears" label: each of these two distinct values must have its own label). This latter constraint may be relaxed by using add_m1_lab.

If partial = TRUE, add_val_labs will apply the specified labeling scheme to all variables that contain a key variable name substring of interest (supplied to the vars argument), which may be one or more variables found in the data.frame (see Example #2).

Value

A data.frame, with new variable value labels added (call get_val_labs to see them), other provisional/default labelr label information added, and previous user-added labelr label information preserved.

Examples

```
# Example #1 - mtcars example, one variable at a time
# one variable at a time, mtcars
df <- mtcars
# now, add value labels
df <- add_val_labs(
  data = df,
  vars = "am",
  vals = c(0, 1),
  labs = c("automatic", "manual")
)

df <- add_val_labs(
  data = df,
  vars = "carb",
  vals = c(1, 2, 3, 4, 6, 8),
  labs = c(
    "1-carb", "2-carbs",
    "3-carbs", "4-carbs",
    "6-carbs", "8-carbs"
  )
)

# var arg can be unquoted if using add_val1()
# note that this is not add_val_labs(); add_val1() has "var" (not "vars" arg)
df <- add_val1(
  data = df,
  var = cyl, # note, "var," not "vars" arg
  vals = c(4, 6, 8),
  labs = c(
    "four-cyl",
    "six-cyl",
    "eight-cyl"
  )
)

df <- add_val_labs(
  data = df,
  vars = "gear",
  vals = c(3, 4),
  labs = c(
    "3-speed",
    "4-speed"
  )
)

# Oops, we forgot 5-speeds; let's finish the job.
df <- add_val_labs(
  data = df,
  vars = "gear",
  vals = 5,
  labs = "5-speed"
```

```
)

head(use_val_labs(df), 3) # they're there

# Example #2 - (Fake) Likert Data
# add val labs to multiple variables at once
# make a "Likert"-type fake data set to demo
# note, by default, add_val_labs() "vars" arg will do partial matching
# in this case, we catch all vars with "x" in their name
set.seed(272)
dflik <- make_likert_data(scale = 1:7)
vals2label <- 1:7
labs2use <- c(
  "VSD",
  "SD",
  "D",
  "N",
  "A",
  "SA",
  "VSA"
)

dflik <- add_val_labs(
  data = dflik, vars = c("x", "y3"), # note the vars args
  vals = vals2label,
  labs = labs2use,
  partial = TRUE
)

# note, all "x" vars get the labs, as does "y3"
# see vars = args above
lik1 <- use_val_labs(dflik)
head(lik1)
# keep a copy
dflik_conv <- use_val_labs(dflik)
head(dflik_conv, 3)
```

`all_quant_labs` *Add Quantile-based Value Labels to All Numeric Vars that Meet Specifications*

Description

Add variable-specific quantile-based value labels to all numeric variables of a data.frame that meet specified conditions.

Usage

```
all_quant_labs(data, qtiles = 5, not.vars = NULL, unique.vals.thresh = 10)

allq(data, qtiles = 5, not.vars = NULL, unique.vals.thresh = 10)
```

Arguments

data	a data.frame.
qtiles	the number of quantile categories to employ (e.g., 4 would indicate quartiles, 5 would indicate quintiles, 10 for deciles, etc.).
not.vars	used to specify any numeric variables that should be exempted from this operation.
unique.vals.thresh	an integer. Numeric variables with fewer than this many unique variables will be exempted from the operation (i.e., will NOT receive quantile value labels).

Details

Note: allq is a compact alias for all_quant_labs: they do the same thing, and the former is easier to type.

Numerical variables that feature decimals or large numbers of distinct values are not eligible to receive conventional add_val_labs()-style value labels. all_quant_labs allows one to label such variables based on quantile thresholds.

Value

A data.frame, with new variable value labels added.

Examples

```
# mtcars demo
df <- mtcars
get_val_labs(df) # none
# add quintile val labs for all numeric vars with >10 unique vals
df <- all_quant_labs(data = df, qtiles = 5, unique.vals.thresh = 10)
get_val_labs(df) # here now
headl(df) # show them; note this is labelr::headl(), not utils::head()
```

---

all_uniquev *Are All Values in a Free-standing Vector Unique?*

Description

For a given vector, does the length of (number of values in) the vector equal the number of unique values in the vector?

Note: all_univ is a compact alias for all_uniquev: they do the same thing, and the former is easier to type

Usage

```
all_uniquev(x, na.rm = TRUE)

all_univ(x, na.rm = TRUE)
```

Arguments

x	a vector.
na.rm	a logical evaluating to TRUE or FALSE indicating whether NA values should be stripped before the computation proceeds.

Value

a 1L logical.

Examples

```
all_uniquev(mtcars$am) # FALSE

set.seed(35994)
z <- runif(25)
all_univ(z) # TRUE; all_univ is an alias for all_uniquev()

z[c(1, 2)] <- NA # two NA values added
all_univ(z, na.rm = FALSE) # FALSE, because the two NA values are not unique
```

---

as_base_data_frame *Convert Augmented Data Frame to Base R Data Frame*

Description

as_base_data_frame noisily converts an augmented data.frame to a Base R data.frame.

Usage

```
as_base_data_frame(data, fact.to.char = FALSE, irreg.to.na = FALSE)

adf(data, fact.to.char = FALSE, irreg.to.na = FALSE)
```

Arguments

data	a data.frame object.
fact.to.char	coerce all factor variables to character variables.
irreg.to.na	convert all irregular values (see irregular2v()) to NA.

Details

Note: adf is a compact alias for as_base_data_frame: they do the same thing, and the former is easier to type

To minimize dependencies and complexities, labelr label-assigning functions are designed to work exclusively with Base R data.frames, not alternative data structures like matrices or augmented data.frames, such as data.tables or tibbles. The suggested labeling workflow is to first assign and work with labels using a Base R data.frame and then convert the resulting object to an augmented

data.frame as desired and without any assumption that labelr labels or functions will smoothly interoperate with the augmented data.frame construct or functions that depend on it.

as_base_data_frame determines whether data argument is a conventional Base R data.frame, some kind of augmented data.frame (e.g., data.table, tibble), or not a data.frame at all (e.g., matrix). If the object has multiple classes, one of which is a data.frame, the object is coerced to be a conventional Base R data.frame, and a message to that effect is issued. If the supplied object is not any kind of data.frame (i.e., a matrix is not any kind of data.frame, while a data.table is a kind of data.frame), an error is thrown. If the supplied object already is a Base R data.frame with no additional classes (i.e., not an augmented data.frame), that supplied object is returned with no changes made and no messages.

Value

a data.frame object with any additional classes removed.

Examples

```
x1 <- runif(10)
x2 <- as.character(sample(c(1:20), 10, replace = TRUE))
x3 <- sample(letters, size = 10, replace = TRUE)
df <- data.frame(x1, x2, x3)
dft <- tibble::as_tibble(df)
class(dft)
df_vanilla <- as_base_data_frame(dft)
class(df_vanilla)
```

---

as_base_data_frame2 *Convert Augmented Data Frame to Base R Data Frame with Alternate Defaults*

Description

as_base_data_frame2 noisily converts an augmented data.frame to a Base R data.frame, with any factors converted to character vectors, and any irregular values (see irregular2()) converted to NA.

Usage

```
as_base_data_frame2(data, fact.to.char = TRUE, irreg.to.na = TRUE)

adf2(data, fact.to.char = TRUE, irreg.to.na = TRUE)
```

Arguments

data	a data.frame object.
fact.to.char	coerce all factor variables to character variables.
irreg.to.na	convert all irregular values (see irregular2v()) to NA.

Details

Note: adf2 is a compact alias for as_base_data_frame: they do the same thing, and the former is easier to type

as_base_data_frame2 is a variant of as_base_data_frame with different default values for fact.to.char and irreg.to.na. Whereas both of these default to FALSE in as_base_data_frame, they both default to TRUE in as_base_data_frame2. This is the only difference between the two functions. As such, as_base_data_frame2 is intended as a simple shortcut to save typing if one prefers to reverse these default logical argument values.

Value

a data.frame object with any additional classes removed.

Examples

```
iris_tib <- tibble::as_tibble(iris)
class(iris_tib)
iris_tib$Sepal.Length[1] <- Inf
head(iris_tib, 1)
iris_df <- as_base_data_frame2(iris_tib)
class(iris_df)
sapply(iris_df, class)
head(iris_df, 1)
```

---

as_labeled_data_frame *Assign Class labeled.data.frame to a Data Frame Object*

Description

as_labeled_data_frame quietly assigns the class labeled.data.frame to a data.frame and discards other augmented data.frame classes, returning a data.frame with classes labeled.data.frame and data.frame.

Usage

```
as_labeled_data_frame(data, fact.to.char = FALSE, irreg.to.na = FALSE)

aldf(data, fact.to.char = FALSE, irreg.to.na = FALSE)
```

Arguments

data	a data.frame object.
fact.to.char	coerce all factor variables to character variables.
irreg.to.na	convert all irregular values (see irregular2v()) to NA.

Details

Note 1: `aldf` is a compact alias for `as_labeled_data_frame`: they do the same thing, and the former is easier to type

Note 2: `as_labeled_data_frame` is used internally by other labelr commands and is not intended for interactive use.

To minimize dependencies and complexities, labelr label-assigning functions are designed to work exclusively with Base R data.frames, not alternative data structures like matrices or augmented data.frames, such as data.tables or tibbles.

`as_labeled_data_frame` determines whether data argument is a conventional Base R data.frame, some kind of augmented data.frame (e.g., data.table, tibble), or not a data.frame at all (e.g., matrix). If the submitted object is a type of data.frame, the object will be returned with the class labeled.data.frame applied, the class data.frame retained, and any other previous class attributes discarded. If the supplied object is not any kind of data.frame (i.e., a matrix is not any kind of data.frame, while a data.table is a kind of data.frame), an error is thrown.

Value

an object of classes labeled.data.frame and data.frame, with any additional classes removed.

Examples

```
x1 <- runif(10)
x2 <- as.character(sample(c(1:20), 10, replace = TRUE))
x3 <- sample(letters, size = 10, replace = TRUE)
df <- data.frame(x1, x2, x3)
dft <- tibble::as_tibble(df)
class(dft)
dfl <- as_labeled_data_frame(dft)
class(dfl)
```

---

as_num *Convert all Suitable Character Variables to Numeric*

Description

`as_num` identifies the character variables of a data.frame that can be coerced to numeric without generating new NA values and, for those variables where this can be done, it makes those conversions (similar to Stata's destring command).

Usage

```
as_num(data, nan2na = TRUE, inf2na = TRUE)
```

Arguments

data	a data.frame object.
nan2na	a logical argument. TRUE if the non-case-sensitive string "nan" should be converted to NA.
inf2na	a logical argument. TRUE if the non-case-sensitive strings "inf" or "-inf should be converted to NA.

Details

Core labelr functions coerce integers to characters and back, which as_num facilitates. Note that character values of "NA" (including "na", "Na", and "nA") will be converted to NA and, by default, so will other "irregular" values (in the sense of check_irregular).

Value

a data.frame object with all applicable character variables coerced to numeric.

Examples

```
set.seed(123)
x1 <- runif(10)
x2 <- as.character(sample(c(1:20), 10, replace = TRUE))
x3 <- sample(letters, size = 10, replace = TRUE)
df <- data.frame(x1, x2, x3)
head(df, 3)
sapply(df, class)
class(df$x2)

df <- as_num(df)
head(df,3)
sapply(df, class)
class(df$x2)
```

---

as_numv *Convert a Suitable Character Vector to Numeric*

Description

as_numv determines whether a character vector can be coerced to numeric without generating new NA values and, if so, it makes that conversion (similar to Stata's destring command).

Usage

```
as_numv(x, nan2na = TRUE, inf2na = TRUE)
```

Arguments

x	a character vector.
nan2na	a logical argument. TRUE if the non-case-sensitive string "nan" should be converted to NA.
inf2na	a logical argument. TRUE if the non-case-sensitive strings "inf" or "-inf should be converted to NA.

Details

Core labelr functions coerce integers to characters and back, which as_numv facilitates. Note that character values of "NA" (including "na", "Na", and "nA") will be converted to NA and, by default, so will other "irregular" values (in the sense of check_irregular).

Value

a vector, converted to numeric if feasible (else, the same character vector that was supplied).

Examples

```
set.seed(123)
x1 <- runif(10)
x2 <- as.character(sample(c(1:20), 10, replace = TRUE))
x2_num <- as_numv(x2)
class(x2)
class(x2_num)
head(x2)
```

---

axis_lab *Retrieve Variable's Name Label for Plot Labeling*

Description

axis_lab accepts a data.frame and single unquoted variable name and returns that variable's name label for use in axis labeling or plot labeling function options.

Usage

```
axis_lab(data, var)

alb(data, var)
```

Arguments

data	a data.frame.
var	the unquoted name of a variable that exists in the data.frame and is name-labeled (using add_name_labs()).

Details

Note 1: `alb` is a compact alias for `axis_lab`: they do the same thing, and the former is easier to type.

Note 2: This command is intended exclusively for interactive use. In particular, the var argument must be the literal name of a single variable (column) found in the supplied data.frame and may NOT be, e.g., the name of a character vector that contains the variable (column name) of interest.

Value

a 1L character vector with var's name label.

Examples

```
# copy mtcars to df
# create a data set
df <- mtcars

# variable names and their labels
names_labs_vec <- c(
  "mpg" = "Miles/(US) gallon",
  "cyl" = "Number of cylinders",
  "wt" = "Weight (1000 lbs)"
)

df <- add_name_labs(df, name.labs = names_labs_vec)

# ggplot example of axis_lab()
library(ggplot2)
p <- ggplot(df, aes(mpg, wt, color = cyl)) +
  geom_point()
p <- p +
  labs(color = axis_lab(df, cyl)) +
  xlab(axis_lab(df, mpg)) +
  ylab(axis_lab(df, wt))

# Base R plot example (using alb() alias)
with(df, plot(mpg, wt,
  xlab = alb(df, mpg),
  ylab = alb(df, wt)
))
```

---

check_any_lab_atts *Check Whether Data Frame Has Any labelr Attributes*

Description

`check_any_lab_atts` returns FALSE if your data.frame has no labelr-generated meta-data attributes (still) associated with it (at all or of a specific sub-type), and TRUE if it does.

Usage

```
check_any_lab_atts(data, labs = "any")
```

Arguments

data	the data.frame you are checking for the presence (or absence) of labelr meta-data.
labs	which label meta-data you are looking for. Default of "any" will look for types "frame.lab", "name.labs", "val.labs", and "factor." (period is part of the sub-string), which are the core labelr meta-data. To search more narrowly, you can try things like labs = "val.labs", labs="name.labs", etc.

Details

By default (labs = "any"), this function looks to see if your data.frame's attributes includes any attribute with a name containing the substring "frame.lab", "name.labs", "val.labs", and/or "factor." These are the core substrings of the label meta-data attributes that labelr creates and manipulates. If you wish to narrow your search for a specific labelr, attribute, you may supply this as a character (sub)string (e.g., check_any_lab_atts(df, "val.labs.cyl") to see if the variable "cyl" has variable value label meta-data). But make sure that your second argument is meaningful (e.g., check_any_lab_atts(iris, "row.") will return TRUE true based on the presence of a standard "row.names" attribute, which has nothing to do with labels.

Value

TRUE if any instance of the default or user-specified meta-data attribute is found, FALSE if not.

Examples

```
# make toy demographic (gender, raceth, etc.) data set
set.seed(555)
df <- make_demo_data(n = 1000) # another labelr:: function
# let's add variable VALUE labels for variable "raceth"
df <- add_val_labs(df,
  vars = "raceth", vals = c(1:7),
  labs = c("White", "Black", "Hispanic", "Asian", "AIAN", "Multi", "Other"),
  max.unique.vals = 50
)

check_any_lab_atts(df)
```

---

check_class *Determine If Vector Belongs to Any of Specified Classes*

Description

check_class determines whether a vector's class is among those specified.

Usage

```
check_class(
  x,
  classes = c("numeric", "integer", "logical", "character", "factor", "ordered"),
  strict = TRUE
)
```

Arguments

x	the vector to check against specified classes.
classes	a character vector of classes against which x is checked.
strict	If TRUE, all of x's classes must be among those specified in the classes argument. If FALSE, at least one but not necessarily all of x's classes must be among those specified in the classes argument.

Details

By default (strict = TRUE), if a vector is of multiple classes, all of its classes must be among those specified via the classes argument.

Value

a 1L logical vector indicating whether x's class is found among those passed to the classes argument.

Examples

```
check_class(mtcars$mpg) # TRUE
check_class(mtcars$mpg, classes = c("numeric", "factor")) # TRUE
check_class(iris$Species) # TRUE
check_class(iris$Species, classes = c("logical", "numeric")) # FALSE
check_class(mtcars$mpg, classes = c("logical", "character", "factor")) # FALSE
```

---

check_irregular *Check Vector for "Irregular" Values*

Description

Check a vector for the presence of "irregular" values, defined as NA values, other arbitrary values you specify, and (by default): NaN, Inf, -Inf, and character variants of same (i.e., upper, lower, or mixed-case variants of "NA","NAN","INF","-INF").

Usage

```
check_irregular(
  x,
  nan.include = TRUE,
  inf.include = TRUE,
  special = c("NA", "NAN", "INF", "-INF"),
  other = NULL,
  any = FALSE
)
```

Arguments

x	an atomic vector to checked for the presence of (any) NA values.
nan.include	treat NaN values as NA (i.e., return TRUE if present).
inf.include	treat Inf and -Inf values as NA (i.e., return TRUE if present).
special	a modifiable set of default character values that will be treated as equivalent to NA values (i.e., will return TRUE if present).
other	an argument for additional values of arbitrary but consistent class (e.g., all numeric, all character) that will be treated as equivalent to NA values (i.e., check_irregular will return TRUE where/if found).
any	if TRUE, return a 1L vector that is TRUE if any irregular/NA-esque value is found in the vector x, FALSE if no such value is found; if any=FALSE, function will return a logical value for every element of x (TRUE if that specific value meets the "irregular"-ity test).

Details

check_irregular is used by core labelr functions (e.g., add_val_labs) to ensure that NA and other irregular (e.g., Inf) values are handled in a simple and consistent – and, hence, rigid – fashion. It is not intended as a user- facing command as part of a labelr data-analytic workflow, though it may be useful in other applications where one wishes to test a vector against a focal and user-extensible class of NA-esque (or other) offending values.

Value

A logical vector (1L if any==TRUE; length of x if any==FALSE).

Examples

```
# below is FALSE, because there is nothing NA-like in this vector
check_irregular(1:10)

# below is TRUE, because we're treating 99 as "NA-esque"
check_irregular(1:100, other = 99)

# below is TRUE, because of NA val
check_irregular(c(1:100, NA))
```

```
# below is TRUE, because nan.include is on (by default)
check_irregular(c(1:100, NaN), nan.include = TRUE)

# below is TRUE, because inf.include is on (by default)
check_irregular(c(1:100, Inf), inf.include = TRUE)

# below is TRUE, because inf.include is on (by default)
check_irregular(c(1:100, -Inf), inf.include = TRUE)

# below is FALSE, it's just letters
check_irregular(letters)

# below is TRUE - see default vals for arg special (function not case-sens)
check_irregular(c(letters, "NA"))

# below is TRUE - see default vals for arg special (function not case-sens)
check_irregular(c(letters, "NAN"))

# below is TRUE - see default vals for arg special (function not case-sens)
check_irregular(c(letters, "-iNf"))

# below is FALSE, search for irregular vals is not substring/regex-based
check_irregular(c(letters, "nan-iNf"))
```

`check_labs_att` *Check Data Frame for Specified labelr Attribute*

Description

`check_labs_att` returns TRUE if your data.frame has the specific attribute indicated and FALSE if it does not.

Usage

```
check_labs_att(data, att = NULL)
```

Arguments

`data`	the data.frame you are checking for the presence (or absence) of labelr meta-data.
`att`	the specific label meta-data you are looking for. Default of NULL will return TRUE if any valid labelr meta-data item of types "frame.lab", "name.labs", "val.labs", or "factor." (period is part of the substring) is present.

Value

TRUE if any instance of the default or user-specified meta-data attribute is found, FALSE if not.

Examples

```
# make toy demographic (gender, raceth, etc.) data set
set.seed(555)
df <- make_demo_data(n = 1000) # another labelr:: function
# let's add variable VALUE labels for variable "race"
df <- add_val_labs(df,
  vars = "raceth", vals = c(1:7),
  labs = c("White", "Black", "Hispanic", "Asian", "AIAN", "Multi", "Other"),
  max.unique.vals = 50
)

check_labs_att(df) # is any valid labelr lab(el) attribute present?
check_labs_att(df, "val.labs.race") # "race" lab specifically TRUE
```

---

clean_data_atts *"Clean" Data Frame Attributes*

Description

Drops name.lab and val.lab attributes associated with columns that are not present in the data.frame (i.e., have been dropped) and re-arranges data.frame attributes so that they appear in a clean, logical order.

Usage

```
clean_data_atts(data)
```

Arguments

data	a data.frame.

Details

labelr meta-data exist as data.frame attributes, added through interactive use in a potentially haphazard order. This function, which is used inside other labelr functions, drops labels for variables that are not (no longer) present in the data.frame and re-arranges label and other data.frame attributes to put them in a more, logical, user-readable order when accessed via, e.g., `attributes()`.

Value

A data.frame, with attributes re-arranged.

Examples

```
# make toy demographic (age, gender, raceth) data set
set.seed(555)
df <- make_demo_data(n = 1000)

# let's add variable VALUE labels for variable "raceth"
```

```
df <- add_val_labs(df,
  vars = "raceth", vals = c(1:7),
  labs = c("White", "Black", "Latino", "Asian", "AIAN", "Multi", "Other"),
  max.unique.vals = 50
)

# let's add variable VALUE labels for variable "gender"
df <- add_val1(
  data = df, gender, vals = c(0, 1, 2),
  labs = c("M", "F", "O"), max.unique.vals = 50
)

# let's add variable NAME labels
df <- add_name_labs(df, name.labs = c(
  "age" = "Age in years",
  "raceth" = "raceth category",
  "gender" = "gender assigned at birth"
))


# let's add a frame label
df <- add_frame_lab(df, frame.lab = "This is a fictional data set that includes
                    demographic variables. It is generated by
                    labelr::make_demo_data")

# show attributes
attributes(df)

# re-arrange and show attributes
df2 <- clean_data_atts(df)
attributes(df2)

# confirm that attributes from df are all present in df2
all(attributes(df) %in% attributes(df2)) # TRUE
```

`convert_labs` *Convert from Haven-style to labelr Variable Value Labels*

Description

Convert a data.frame with Haven package-style labels to a data.frame with labelr name labels and `add_val_labs`-style one-to-one, value labels.

Usage

```
convert_labs(data, max.unique.vals = 50)
```

Arguments

data	the data.frame with Haven-style vector value label attributes.
max.unique.vals	constrains the variables that may receive value labels to those whose total unique values do not exceed the integer value supplied to this argument. Note that labelr sets a hard ceiling of 5000 on the total number of unique value labels that any variable is permitted to have under any circumstance, as labelr is primarily intended for interactive use with moderately-sized (<=~1M-row) data.frames.

Value

a data.frame.

Examples

```
# convert haven vector labels to labelr value labels
library(haven)
library(tibble)
x1 <- labelled(1:8, c(good = 1, bad = 5))
x2 <- labelled(1:8, c(good = 1, mediocre = 4, bad = 5, horrible = 8))

# make this a tibble
hdf <- tibble::tibble(x1, x2)
hdf # how it looks

# convert value labels to labelr label values
hdf1 <- convert_labs(hdf)

# show select values of hdf1
head(hdf1)

# show that labelr labels are there for the using
head(use_val_labs(hdf1))

# filter hdf1 using x1's "bad" labelr value label (with flab())
head(flab(hdf1, x1 == "bad"), 3)

# filter hdf1 using x1's "good" value label (with flab())
head(flab(hdf1, x1 == "good"), 3)

# return select rows and columns with slab()
slab(hdf1, x2 %in% c("good", 2), x2)
slab(hdf1, x2 %in% c("good", 2), x1)
```

---

copy_var *Copy a Data Frame Variable and its Value labels to Another Variable*

Description

Note: copy_var copies an existing variable and its value labels from a data.frame to another new or (if force = TRUE) existing variable of the data.frame.

Usage

```
copy_var(data, from.var, to.var, force = FALSE)
```

Arguments

data	a data.frame to which variable value labels will be added.
from.var	the unquoted name of the variable whose values and labels will be assigned to the to.var. This variable must presently exist in the data.frame.
to.var	the unquoted name of the variable to which the from.var's values and labels will be assigned. If force = FALSE, this must be a new variable name (one that does not refer to a variable that already exists in the data.frame).
force	if to.var already exists in the data.frame, allow it to be overwritten. If FALSE, this will not be allowed, and an error will be issued.

Details

Any non-labelr R operation that changes a variable's (column's) name or that copies its contents to another variable (column) with a different name will not associate the original variable's value labels with the new variable name. To mitigate this, copy_var allows one to copy both a variable (column) and its value labels and assign those to another variable.

Value

A data.frame.

Examples

```
# make toy demographic (gender, raceth, etc.) data set
set.seed(555)
df <- make_demo_data(n = 1000) # another labelr:: function
# let's add variable VALUE labels for variable "raceth"
df <- add_val_labs(df,
  vars = "raceth", vals = c(1:7),
  labs = c("White", "Black", "Hispanic", "Asian", "AIAN", "Multi", "Other"),
  max.unique.vals = 50
)

head(df, 4)
df <- copy_var(df, from.var = raceth, to.var = re_copy)
df <- copy_var(df, from.var = x1, to.var = var1)
head(df, 4)
get_val_labs(df)
```

drop_frame_lab *Remove Frame Label Attribute from a Data Frame*

Description

Remove the frame label attribute (see add_frame_lab) from a data.frame, if one is present.

Usage

```
drop_frame_lab(data)

dfl(data)
```

Arguments

data	the data.frame with a frame label that you wish to drop (and which was added using add_frame_lab).

Details

See add_frame_lab for more on this labeling construct.

Note: dfl is a compact alias for drop_frame_lab: they do the same thing, and the former is easier to type.

Value

a data.frame (with any previously applied frame.lab attribute removed).

Examples

```
# add frame.lab to mtcars and assign to new data.frame mt2
mt2 <- add_frame_lab(mtcars, frame.lab = "Data extracted from the 1974 Motor
                     Trend US magazine, comprising fuel consumption and 10
                     aspects of automobile design and performance for 32
                     automobiles (1973-74 models). Source: Henderson and
                     Velleman (1981), Building multiple regression models
                     interactively. Biometrics, 37, 391-411.")

get_frame_lab(mt2) # return frame.lab alongside data.frame name as a data.frame
drop_frame_lab(mt2) # remove this frame.lab
get_frame_lab(mt2) # the data.frame name now doubles as its frame label
is.null(attributes(data)[["frame.lab"]]) # the attribute is NULL
```

drop_name_labs *Remove Name Label Attributes from a Data Frame*

Description

Remove one or more descriptive variable name label attributes previously added to a data.frame using add_name_labs.

Note: dnl is a compact alias for drop_name_labs: they do the same thing, and the former is easier to type

Usage

```
drop_name_labs(data, vars = NULL)

dnl(data, vars = NULL)
```

Arguments

data	the data.frame with name labels that you wish to drop (and which were added using add_name_labs).
vars	the names of the columns/variables (not the name labels) whose name labels you wish to drop. If NULL, all variables' name labels will be dropped.

Details

drop_name_labs works with add_name_labs, get_name_labs and use_name_labs to facilitate creation, accessing, substitution, and removal of variable name labels for variable names. Each variable (column) of a data.frame can receive one and only one "name label," which typically is a noun phrase that expounds the meaning or contents of the variable's name (e.g., "Weight in ounces at birth" might be a viable name label for a column called "wgt"). drop_name_labs takes a data.frame and (optionally) a character vector of variables whose name labels should be discarded: If only a data.frame is provided, all variable name labels will be dropped. You can assign new name labels using new calls to add_name_labs (If you wish to change some or all name labels, you do not need to call drop_name_labs: you can simply pass the new name labels to add_name_labs, and they will overwrite the old ones (including any automatically generated provisional ones), while leaving in place any previously added name labels that you do not explicitly replace).

Value

The same data.frame you submitted, except that the selected name label attribute meta-data has been removed.

Examples

```
# create a data set
df <- mtcars
```

```
# variable names and their labels
names_labs_vec <- c(
  "mpg" = "Miles/(US) gallon",
  "cyl" = "Number of cylinders",
  "disp" = "Displacement (cu.in.)",
  "hp" = "Gross horsepower",
  "drat" = "Rear axle ratio",
  "wt" = "Weight (1000 lbs)",
  "qsec" = "1/4 mile time",
  "vs" = "Engine (0 = V-shaped, 1 = straight)",
  "am" = "Transmission (0 = automatic, 1 = manual)",
  "gear" = "Number of forward gears",
  "carb" = "Number of carburetors"
)

# assign variable labels
df <- add_name_labs(df,
  vars = names(names_labs_vec),
  labs = names_labs_vec
)

# see what we have
get_name_labs(df)

# drop the name label for var/col "am"
df <- drop_name_labs(df, "am")

# see what this did to the name label for "am"
get_name_labs(df)

# now, drop all of the name labels
df <- drop_name_labs(df)
get_name_labs(df) # they're gone
```

`drop_val1` *Drop a Single Variable's Value Labels*

Description

Drop all value labels previously applied to one or more variables using `add_val_labs`, `add_quant_labs`,`add_m1_lab`, and related functions (e.g., `add_val1`) or aliases (e.g., `avl`).

Note: `dvl1` is a compact alias for `drop_val1`: they do the same thing, and the former is easier to type

Usage

```
drop_val1(data, var)

dvl1(data, var)
```

Arguments

data	a data.frame.
var	the unquoted name of the variable whose value labels will be dropped.

Details

Note: drop_val1 is the drop_val_labs analogue to add_val1: just as add_val1 is a variant of add_val_labs that allows you to specify only one variable at a time unquoted for value labeling, drop_val1 allows you to pass one unquoted variable name at a time for value dropping. See those functions for further details regarding the conventions.

Value

A data.frame, with all value labels dropped from specified variable.

Examples

```
# make a "Likert"-type fake data set to demo
# note, by default, add_val_labs() "vars" arg will do partial matching
# in this case, we catch all vars with "x" in their name
set.seed(272)
dflik <- make_likert_data(scale = 1:7)
vals2label <- 1:7
labs2use <- c(
  "VSD",
  "SD",
  "D",
  "N",
  "A",
  "SA",
  "VSA"
)

dflik <- add_val1(
  data = dflik, var = x3,
  vals = vals2label,
  labs = labs2use
)

# see what this did
get_val_labs(dflik, "x3")

dfdrop <- drop_val1(dflik,
  var = x3
) # odd choice, but ok

# var x3's value labels are gone, like we asked for
get_val_labs(dfdrop, "x3")
```

drop_val_labs *Drop Value Labels from One or More Variables*

Description

Drop all value labels previously applied to one or more variables using add_val_labs, add_quant_labs,add_m1_lab, and related functions (e.g., add_val1) or aliases (e.g., avl).

Note: dvl is a compact alias for drop_val_labs: they do the same thing, and the former is easier to type

Usage

```
drop_val_labs(data, vars = NULL, partial = FALSE, not.vars = NULL)

dvl(data, vars = NULL, partial = FALSE, not.vars = NULL)
```

Arguments

data	a data.frame.
vars	a character vector that corresponds to the name(s) (or substring within the name(s), if partial = TRUE) of one or more variables form which value labels will be removed. If NULL, all value labels will be removed from all value-labeled variables.
partial	To drop labels for many, similarly named variables (e.g., "x1" through "x20"), you can provide a substring only and set partial = TRUE (default is FALSE). For example, to drop value labels for colnames "x1" through "x20", you could use vars = c("x"), along with partial = TRUE. Be careful with this, as it also will attempt to drop value labels for columns with colnames "sex" or "tax.bracket" (etc.), because they, too, contain an "x" in their names).
not.vars	use of the partial argument can result in situations where you inadvertently attempt to drop value labels for a variable. For example, if vars="x", and partial=TRUE, then drop_val_labs will attempt to drop labels for not only "x1", "x2","x3", and "x4", but also for "sex", "tax.bracket", and other "x"-containing variable names. Use of not.vars allows you to indicate variables that you wish to exempt from value label- dropping, even if their names contain the string found in vars. Note that not.vars gets priority: setting vars="x", partial=TRUE, and not.vars="x" is tantamount to telling drop_val_labs that you actually do not wish to drop value labels for any of the variables that you specified in vars, resulting in no value labels being dropped.

Details

drop_val_labs works with other labelr functions (e.g., add_val_labs, get_val_labs, use_val_labs, add_lab_cols) to facilitate the creation, accessing, modification, use, or deletion of variable value labels.

Value

A data.frame, with all value labels dropped from specified variables.

Examples

```
# make a "Likert"-type fake data set to demo
# note, by default, add_val_labs() "vars" arg will do partial matching
# in this case, we catch all vars with "x" in their name
set.seed(272)
dflik <- make_likert_data(scale = 1:7)
vals2label <- 1:7
labs2use <- c(
  "VSD",
  "SD",
  "D",
  "N",
  "A",
  "SA",
  "VSA"
)

dflik <- add_val_labs(
  data = dflik, vars = c("x", "y3"), # note the vars args
  vals = vals2label,
  labs = labs2use,
  partial = TRUE
)

dfdrop <- drop_val_labs(dflik,
  vars = c("x2", "y3"),
  partial = FALSE
)

# var x2's value labels are gone, like we asked for
get_val_labs(dfdrop, "x2")

# var x1's value labels are intact, b/c we didn't ask to drop them
get_val_labs(dfdrop, "x1")

dfxgone <- drop_val_labs(dflik,
  c("x"),
  partial = TRUE
)

# still a lot of value labels, but all are for "y" vars,
# ...none is left for "x" vars
get_val_labs(dfxgone)
```

---

fact2char *Convert All Factor Variables of a Data Frame to Character Variables*

Description

Convenience function to convert all factor variables to character.

Usage

```
fact2char(data)

f2c(data)
```

Arguments

data	a data.frame object.

Value

a data.frame identical to data, with exception that any factors have been converted to character variables.

Examples

```
sapply(iris, class)
head(iris)

iris_ch <- fact2char(iris)

sapply(iris_ch, class)
head(iris_ch)
```

---

factor_to_lab_int *Convert a Factor Variable Column to Value-labeled Integer Variable Column*

Description

factor_to_lab_int converts a factor variable (column) of a data.frame to a value-labeled integer variable and converts the factor level labels to labelr value labels, returning the modified data.frame.

Usage

```
factor_to_lab_int(data, var)

f2int(data, var)
```

Arguments

data	a data.frame object.
var	the (unquoted) name of a factor variable found in data.

Details

Note 1: `f2int` is a compact alias for `factor_to_lab_int`: they do the same thing, and the former is easier to type.

Note 2: `factor_to_lab_int()` is NOT an "undo" for `lab_int_to_factor()`. `factor_to_lab_int()` will assign sequential integer values from 1 to k (the number of distinct factor levels) in factor level order, and this will not necessarily match the integer values of a variable previously subjected to a `lab_int_to_factor()` call. See extended second example below for demo.

Value

a data.frame.

Examples

```
class(iris[["Species"]])
iris_df <- factor_to_lab_int(iris, Species)
class(iris_df[["Species"]])
get_val_labs(iris_df, "Species")

# copy data.frame mtcars to mt2
carb_orig_int <- mtcars

# !NOTE! factor_to_lab_int() is NOT an "undo" for lab_int_to_factor()
# Integer values will be sequential integers in factor level order
# Demo this

# add value labels to mtcars$carb; and assign data.frame to carb_orig_int
carb_orig_int <- add_val_labs(
  data = mtcars,
  vars = "carb",
  vals = c(1, 2, 3, 4, 6, 8),
  labs = c(
    "1c", "2c", # a tad silly, but these value labels will demo the principle
    "3c", "4c",
    "6c", "8c"
  )
)

# carb as labeled numeric
class(carb_orig_int$carb) # numeric
levels(carb_orig_int$carb) # none, not a factor
head(carb_orig_int$carb, 3) # compare to carb_to_int (below)
mean(carb_orig_int$carb) # compare to carb_to_int (below)
lm(mpg ~ carb, data = carb_orig_int) # compare to carb_to_int (below)
(adj_r2_int <- summary(lm(mpg ~ carb, data = carb_orig_int))$adj.r.squared)
AIC(lm(mpg ~ carb, data = carb_orig_int)) # compare to carb_to_int (below)

# carb as factor
carb_fac <- carb_orig_int # copy carb_orig_int to new data.frame carb_fac
carb_fac <- lab_int_to_factor(carb_fac, carb) # alias int2f() also works
class(carb_fac$carb) # factor
```

```
levels(carb_fac$carb) # has levels
head(carb_fac$carb, 3)
lm(mpg ~ carb, data = carb_fac) # factor
(adj_r2_fac <- summary(lm(mpg ~ carb, data = carb_fac))$adj.r.squared)
AIC(lm(mpg ~ carb, data = carb_fac)) # compare to R2, AIC for carb_to_int

# ??back?? to integer? Not quite. Compare carb_to_int to carb_orig_int
carb_to_int <- carb_fac # copy carb_fac to carb_to_int
carb_to_int <- factor_to_lab_int(carb_to_int, carb) # alias int2f() also works
class(carb_to_int$carb) # Is an integer
levels(carb_to_int$carb) # NOT a factor
mean(carb_to_int$carb) # NOT the same as carb_orig_int
lm(mpg ~ carb, data = carb_to_int) # NOT the same as carb_orig_int
(adj_r2_fac <- summary(lm(mpg ~ carb, data = carb_to_int))$adj.r.squared)
AIC(lm(mpg ~ carb, data = carb_to_int)) # NOT the same as carb_orig_int
```

`flab` *Filter Data Frame Rows Using Variable Value Labels*

Description

`flab` ("filter using labels") allows one to filter-subset a data.frame based on variable-specific value label attributes.

Usage

```
flab(data, condition)
```

Arguments

`data`	the data.frame from which columns will be selected.
`condition`	row-filtering conditions along the lines of base::subset() and/or dplyr::filter(), which may involve a combination of value labels (for value-labeled variables only) and actual values (for non-value-labeled variables only).

Details

`flab` accepts a labelr value-labeled data.frame, followed by condition- based row-filtering instructions (akin to `base::subset` or `dplyr::filter`), expressed in terms of variable value labels that exist only as meta-data (i.e., not visible using `View()`, `head()`, etc.), and returns the filtered data.frame in terms of the values themselves. In other words, value labels are supplied to the `flab()` call to direct the filtering process, but those value labels are not displayed in the cells of the returned data.frame – the raw values themselves are. This functionality may be useful for interactively subsetting a data.frame, where character value labels may be more intuitive and easily recalled than the underlying variable values themselves (e.g., raceth=="White" & gender="F" may be more intuitive or readily recalled than raceth==3 & gender==2).

Note 1: When using `flab`, any conditional row-filtering syntax involving value-labeled variables must be expressed in terms of those variables' value labels, not the raw values themselves. Filtering

on non-value-labeled variables is also permitted, with those variables' filtering conditions being expressed in terms of raw values. Further, `flab()` calls may reference both types of columns (i.e., value-labeled variables and non-value-labeled variables), provided filtering conditions for the former are expressed in terms of value labels.

Note 2: `flab` (and labelr more broadly) is intended for moderate-sized (or smaller) data.frames, defined loosely as those with a few million or fewer rows. With a conventional (c. 2024) laptop, labelr operations on modest- sized (~100K rows) take seconds (or less); with larger (> a few million rows) data.frames, labelr may take several minutes (or run out of memory and fail altogether!), depending on the complexity of the call and the number and type of cells implicated in it.

See also `slab`, `use_val_labs`, `add_val_labs`, `add_val1`, `add_quant_labs`, `add_quant1`, `get_val_labs`, `drop_val_labs`. For label-preserving subsetting tools that subset in terms of raw values (not value labels), see `sfilter`, `sbrac`, `ssubset`, `sdrop`.

Value

a labelr label attribute-preserving data.frame consisting of the selected rows that meet the filtering condition(s).

Examples

```
# make toy demographic (gender, raceth, etc.) data set
set.seed(555)
df <- make_demo_data(n = 1000) # another labelr:: function
# let's add variable VALUE labels for variable "raceth"
df <- add_val_labs(df,
  vars = "raceth", vals = c(1:7),
  labs = c("White", "Black", "Hispanic", "Asian", "AIAN", "Multi", "Other"),
  max.unique.vals = 50
)

# let's add variable VALUE labels for variable "gender"
# note that, if we are labeling a single variable, we can use add_val1()
# distinction between add_val1() and add_val_labs() will become more meaningful
# when we get to our Likert example
df <- add_val1(
  data = df, gender, vals = c(0, 1, 2, 3, 4),
  labs = c("M", "F", "TR", "NB", "Diff-Term"), max.unique.vals = 50
)

# see what we did
# get_val_labs(df)
get_val_labs(df, "gender")
get_val_labs(df, "raceth")

# use --labels-- to filter w/ flab() ("*F*ilter *lab*el")
dflab <- flab(df, raceth == "Asian" & gender == "F")
head(dflab, 4)

# equivalently, use --values--- to filter w/ sfilter() ("*S*afe filter")
dfsf <- sfilter(df, raceth == 3 & gender == 1)
head(dfsf, 4)
```

get_all_factors *Put Data Frame Factor Level Information into a List*

Description

get_all_factors returns a list of character vectors, where each character vector is a given factor variable's unique levels, and where the vector is given the same name as the factor variable itself. If the data.frame contains no factors, an empty (length 0) list is returned.

Usage

```
get_all_factors(data)
```

Arguments

data	the data.frame you are checking for factor variables.

Value

A list of 0, 1, or more character variables.

Examples

```
class(get_all_factors(iris))
length(get_all_factors(iris))
zz <- iris
zz$u <- zz$Species # zz has two factor variables
class(get_all_factors(zz))
length(get_all_factors(zz))
get_all_factors(mtcars)
length(get_all_factors(mtcars))
```

get_all_lab_atts *Put all Data Frame label attributes into a List*

Description

get_all_lab_atts returns a list of labelr-generated meta-data attributes attached to a data.frame, all of which should have names beginning with one of these character strings: "frame.lab", "name.labs", "val.labs", "factor.".

Usage

```
get_all_lab_atts(data, atts = c("name.labs|val.labs|frame.lab|factor."))
```

Arguments

data	the data.frame you are checking for labelr meta-data attributes.
atts	default is to look for all/any, but you can specify a more narrow subset (or some other, altogether-irrelevant attribute, but do not do that).

Value

A free-standing list of labelr attributes.

Examples

```
# make toy demographic (gender, raceth, etc.) data set
set.seed(555)
df <- make_demo_data(n = 1000) # another labelr:: function
# let's add variable VALUE labels for variable "raceth"
df <- add_val_labs(df,
  vars = "raceth", vals = c(1:7),
  labs = c("White", "Black", "Hispanic", "Asian", "AIAN", "Multi", "Other"),
  max.unique.vals = 50
)

get_all_lab_atts(df) # returns all; is default
get_all_lab_atts(df, "val.labs") # returns only "val.labs" attributes
get_all_lab_atts(df, "class") # You can (but probably should not) use this way.
```

---

get_factor_atts *Get Factor Attributes from a Labeled Data Frame*

Description

get_factor_atts searches a labelr labeled data.frame for factors. If any are found, a list of character vectors of factor levels is returned, with each character vector being the set of unique levels for a factor variable, and with each character vector named according to the convention "factor." + variable name (e.g.,"factor.Species" for iris$Species). Used internally by other labelr functions to get information about factors in labeled data.frames.

Usage

```
get_factor_atts(data)
```

Arguments

data	the labelr labeled data.frame you are checking for factor variables.

Value

A list of character vectors, each named according to the convention "factor." + variable name (e.g.,"factor.Species" for iris$Species). If the data.frame lacks labelr attributes or lacks factors, an empty list will be returned.

Examples

```
ir2 <- iris
unique(ir2$Species)

ir2 <- add_val_labs(ir2,
  vars = "Species", vals = c(
    "setosa",
    "versicolor",
    "virginica"
  ),
  labs = c("se", "ve", "vi")
)
get_val_labs(ir2)
head(use_val_labs(ir2))
get_factor_atts(iris) # no such info: iris is not labelr labeled
get_factor_atts(ir2) # this one has info: it's labelr labeled
```

---

get_factor_info *Return Factor Attributes as a Data Frame*

Description

get_factor_info searches a labelr labeled data.frame for factors. If any are found, a data.frame is returned with the name and unique factor levels of each, along with a logical indicator of whether the factor is ordered, with one row per level per factor. If none are found, a one-row data.frame of NA values is returned.

Usage

```
get_factor_info(data, var = NULL)
```

Arguments

data	the labelr labeled data.frame you are checking for factor variables.
var	a 1L character vector with the name of a specific variable (column) of data, if you wish to restrict query to a single variable (else, keep NULL and will get info for any relevant variables).

Value

A data.frame with three columns: "factor.var" (the name of the factor variable in question), "levels" (the given level of that factor, expressed as a character), and "ordered" (TRUE if so, FALSE if not). If no factors are present in the supplied data.frame, a one-row data.frame of same structure with all three cells set to NA.

Examples

```
ir2 <- iris
unique(ir2$Species)

ir2 <- add_val_labs(ir2,
  vars = "Species", vals = c(
    "setosa",
    "versicolor",
    "virginica"
  ),
  labs = c("se", "ve", "vi")
)
get_val_labs(ir2)
head(use_val_labs(ir2))
get_factor_info(iris) # no such info: iris is not labelr labeled
get_factor_info(ir2) # this one has info: it's labelr labeled
```

---

get_frame_lab *Return a Data Frame's Frame Label*

Description

For a frame-labeled data.frame, `get_frame_lab` returns a derivative 1x2 data.frame that lists the data.frame name and its frame.lab attribute.

Note: `gfl` is a compact alias for `get_frame_lab`: they do the same thing, and the former is easier to type.

Usage

```
get_frame_lab(data)

gfl(data)
```

Arguments

`data`	a data.frame.

Details

`get_frame_lab` returns the overall descriptive "frame label" that is assigned to a data.frame using `add_frame_lab`.

Value

A 1x2 data.frame, consisting of "data.frame" and "frame.lab" values for the supplied data.frame. If the supplied data.frame does not have a frame label, its name also will be used as its frame label (i.e., both entries of the returned 1x2 data.frame will be the data.frame name).

Examples

```
# add frame.lab to mtcars and assign to new data.frame mt2
mt2 <- add_frame_lab(mtcars, frame.lab = "Data extracted from the 1974 Motor
                     Trend US magazine, comprising fuel consumption and 10
                     aspects of automobile design and performance for 32
                     automobiles (1973-74 models). Source: Henderson and
                     Velleman (1981), Building multiple regression models
                     interactively. Biometrics, 37, 391-411.")

attr(mt2, "frame.lab") # check for attribute

# return frame.lab alongside data.frame name as a data.frame
get_frame_lab(mt2)
```

---

get_labs_att *Return Specified Label Attribute, if Present*

Description

`get_labs_att` returns the specified piece of labelr lab(el) attribute meta- data information if it is present.

Usage

```
get_labs_att(data, att)
```

Arguments

`data`	the data.frame you are checking for the presence (or absence) of labelr meta-data.
`att`	the specific label meta-data you are looking for. Default of NULL will return any and all meta-data with name substring "name.labs", "val.labs", or "factor." (period is part of the substring).

Value

A list.

Examples

```
# make toy demographic (gender, raceth, etc.) data set
set.seed(555)
df <- make_demo_data(n = 1000) # another labelr:: function
# let's add variable VALUE labels for variable "raceth"
df <- add_val_labs(df,
  vars = "raceth", vals = c(1:7),
  labs = c("White", "Black", "Hispanic", "Asian", "AIAN", "Multi", "Other"),
  max.unique.vals = 50
)
```

get_name_labs *Return Look-up Table of Variable Names and Name Labels*

Description

For a name-labeled data.frame, `get_name_labs` returns a derivative data.frame that lists each variable and its variable name label.

Note: `gnl` is a compact alias for `get_name_labs`: they do the same thing, and the former is easier to type

Usage

```
get_name_labs(data, vars = NULL)

gnl(data, vars = NULL)
```

Arguments

`data`	a data.frame.
`vars`	a character vector with the name(s) of any specific variable(s) (If NULL, returned data.frame will contain all variable name labels).

Value

A two-column data.frame, consisting of "var" and "lab" columns, where each row corresponds to a unique variable (column) from the user- supplied data.frame.

Examples

```
# create a data set
df <- mtcars

# variable names and their labels
names_labs_vec <- c(
  "mpg" = "Miles/(US) gallon",
  "cyl" = "Number of cylinders",
  "disp" = "Displacement (cu.in.)",
  "hp" = "Gross horsepower",
  "drat" = "Rear axle ratio",
  "wt" = "Weight (1000 lbs)",
  "qsec" = "1/4 mile time",
  "vs" = "Engine (0 = V-shaped, 1 = straight)",
  "am" = "Transmission (0 = automatic, 1 = manual)",
  "gear" = "Number of forward gears",
  "carb" = "Number of carburetors"
)

# assign variable labels
```

```
df <- add_name_labs(df,
  vars = names(names_labs_vec),
  labs = names_labs_vec
)

# see what we have
get_name_labs(df)
```

`get_val_lab1` *Return Look-up Table of One Variable's Value Labels*

Description

For a data.frame with value-labeled variables, `get_val_lab1` returns a derivative data.frame or vector that shows the value-to-label mapping for each unique value of that value-labeled variable.

Usage

```
get_val_lab1(data, var, simplify = FALSE)

gvl1(data, var, simplify = FALSE)
```

Arguments

`data`	a data.frame.
`var`	the unquoted name of the variable (column) for which a value-to-label look-up mapping is sought.
`simplify`	return the mapping as a named vector, not a data.frame (defaults to FALSE).

Details

`get_val1` is a variant of `get_val_labs` that allows you to specify only one var whose value-to-label mapping you wish to look up.

Note 1: As with `get_val_labs()`, `get_val_lab1()` exists to provide a visual, human-interpretable quick look at how value labels map to underlying values and is NOT intended for use in automated querying, subsetting, or other manipulation of those value labels. Further: Unlike `get_val_labs()`, which may return value-to-label mappings for –several– variables of potentially different atomic types, `get_val_lab1()` limits itself to returning the value labels of a –single– variable (column) of the supplied data.frame.

For this reason, and in contrast to the behavior of `get_val_labs()`, if `get_val_lab1()`'s simplify argument is set to FALSE (the default), the returned data.frame will express var values as numeric if this can be done without creating new NA values (i.e., in the sense of `as_numv()`). In contrast, if simplify is TRUE, the look-up table information will be returned as a named character vector.

Note 2: `gvl1` is a compact alias for `get_val_lab1`: they do the same thing, and the former is easier to type

Value

By default, a three-column data.frame, consisting of "var", "vals", and "labs" columns, where each row corresponds to a unique value of var OR – for variables labeled using `add_quant_labs` (or `add_quant1`) – the approximate (i.e., possibly rounded) upper bound of numerical values that fall within that label's range of coverage. If simplify is FALSE, a character vector will returned.

Examples

```
# add val labs to multiple variables at once
# make a "Likert"-type fake data set to demo
# note, by default, add_val_labs() "vars" arg will do partial matching
# in this case, we catch all vars with "x" in their name
set.seed(272)
dflik <- make_likert_data(scale = 1:7)
vals2label <- 1:7
labs2use <- c(
  "VSD",
  "SD",
  "D",
  "N",
  "A",
  "SA",
  "VSA"
)

dflik <- add_val_labs(
  data = dflik, vars = c("x", "y3"), # note the vars args
  vals = vals2label,
  labs = labs2use,
  partial = TRUE
)

# note, all "x" vars get the labs, as does "y3"
get_val_lab1(dflik, x1)

get_val_lab1(dflik, x1, simplify = TRUE)
```

---

get_val_labs *Return Look-up Table of Variable Values and Value Labels*

Description

For a data.frame with value-labeled variables, `get_val_labs` returns a derivative data.frame that shows the value-to-label mapping for each unique value of each value-labeled variable.

Usage

```
get_val_labs(data, var = NULL)

gvl(data, var = NULL)
```

Arguments

data	a data.frame.
var	a character vector with the name(s) of any specific variable(s) (If NULL, returned data.frame will contain all variable value labels).

Details

Note 1: get_val_labs returns a data.frame that is intended strictly to facilitate human-in-the-loop –visual– display and inspection of what (if any) value label has been associated with each variable value. It is –not– intended for use in automated querying or subsetting or as an indicator of of the supplied data.frame's columns' underlying classes or atomic types. In particular, all columns of the –returned– data.frame object are coerced to character for display purposes, as a result of concatenating value information from different variables of potentially different atomic types or classes. For example, all elements of the "vals" column are expressed as character even if the underlying values themselves are numeric.

Note 2: gvl is a compact alias for get_val_labs: they do the same thing, and the former is easier to type

Value

A three-column data.frame, consisting of "var", "vals", and "labs" columns, where each row corresponds to a unique value of a value-labeled variable (column) from the user-supplied data.frame OR – for variables labeled using add_quant_labs (or add_quant1) – the upper bound of numerical values that fall within that label's range of coverage. Note that all variables of the returned data.frame are coerced to character (see Note 1 of details).

Examples

```
# add val labs to multiple variables at once
# make a "Likert"-type fake data set to demo
# note, by default, add_val_labs() "vars" arg will do partial matching
# in this case, we catch all vars with "x" in their name
set.seed(272)
dflik <- make_likert_data(scale = 1:7)
vals2label <- 1:7
labs2use <- c(
  "VSD",
  "SD",
  "D",
  "N",
  "A",
  "SA",
  "VSA"
)

dflik <- add_val_labs(
  data = dflik, vars = c("x", "y3"), # note the vars args
  vals = vals2label,
  labs = labs2use,
  partial = TRUE
```

```
)

# note, all "x" vars get the labs, as does "y3"
get_val_labs(dflik)
get_val_labs(dflik, "x1")
```

---

greml *Determine Which Pattern Elements of One Character Vector Are Found in at Least One Element of A Second Character Vector*

Description

greml takes two character vectors of strings and, for each pattern represented by an element of the first vector, searches all elements of the second vector to see if any of those elements of the second vector matches that pattern element of the first vector.

Usage

```
greml(patterns, x, ignore.case = TRUE, vals = FALSE)
```

Arguments

patterns	a character vector of comma-separated, quoted character strings or regular expressions.
x	a character vector that will be tested for presence/absence of the patterns passed via the patterns argument.
ignore.case	search in a non-case-sensitive fashion if TRUE.
vals	by default, vals = FALSE and will return a named vector that indicates, for each unique element of patterns, whether a match for that pattern was found somewhere in the x vector (one named logical element returned for each patterns argument vector element). If TRUE, vals returns the unique values of patterns that were matched (one character element for each matched patterns argument vector element).

Details

This function accepts a character vector of text substring patterns or regular expressions (patterns argument), and searches a second character vector (x argument) to determine which elements of the first vector (patterns) match at least one element anywhere in the second vector (x). If vals = TRUE (default is FALSE), each matched pattern element is returned; if vals = FALSE, a vector of named logical values equal in length to the object supplied to the patterns argument is returned, indicating for each patterns element whether a match for it was found anywhere in x (TRUE if so, FALSE if not), with the names corresponding to the elements of the patterns vector. If ignore.case = TRUE (the default), neither vector is treated case- sensitively (both are coerced to lower-case before other operations occur).

Used internally by various labelr functions (e.g., use_val_labs). Note that this is the same search and syntax that is performed by gremlr() (gremlr means "greml in reverse"), except that, whereas

greml returns matches in terms of the patterns argument (i.e., which patterns elements match at least one x element), gremlr returns matches in terms of the x argument.

Value

a vector, either character (if vals = TRUE) or logical (if vals = FALSE).

Examples

```
# search for "AB" (case-sensitively) anywhere in subsequent vector
greml(c("AB"), c("ab", "ab", "abc", "z"), vals = TRUE, ignore.case = FALSE)
# character(0)

# search for "AB" (non-case-sensitively; the default) anywhere in next vector
greml(c("AB"), c("ab", "ab", "abc", "z"), vals = TRUE, ignore.case = TRUE)
# [1] "AB"

# other searches
greml(c("AB"), c("ab", "ab", "abc", "z"), vals = TRUE, ignore.case = FALSE)
greml(c("ab"), c("ab", "ab", "abc", "z"), vals = FALSE)
greml(c("ab", "Q"), c("ab", "ab", "abc", "z"), vals = FALSE)
greml(c("a|b", "Q"), c("a", "b", "abc", "z"), vals = FALSE)
greml(c("a|b", "Q"), c("a", "b", "z"), vals = FALSE)
greml(c("a|b", "Q"), c("bq", "z"), vals = FALSE)
greml(c("a|b", "Q"), c("bq", "z"), vals = TRUE)

# compare greml (above) to gremlr() (here)
gremlr(c("AB"), c("ab", "ab", "abc", "z"), vals = TRUE, ignore.case = FALSE)
gremlr(c("ab"), c("ab", "ab", "abc", "z"), vals = FALSE)
gremlr(c("ab", "Q"), c("ab", "ab", "abc", "z"), vals = FALSE)
gremlr(c("a|b", "Q"), c("a", "b", "abc", "z"), vals = FALSE)
gremlr(c("a|b", "Q"), c("a", "b", "z"), vals = FALSE)
gremlr(c("a|b", "Q"), c("bq", "z"), vals = FALSE)
```

---

gremlr *Determine Which Elements of a Character Vector Match at Least One Pattern Contained in Any of the Elements of Another Character Vector*

Description

gremlr accepts two character vectors of strings and, for each element of the second vector, determines whether that element matches any of the patterns supplied via any of the elements of the first vector.

Usage

```
gremlr(patterns, x, ignore.case = TRUE, vals = FALSE)
```

Arguments

patterns	a character vector of comma-separated, quoted character strings or regular expressions.
x	a character vector that will be tested for presence/absence of the patterns passed via the patterns argument.
ignore.case	search in a non-case-sensitive fashion if TRUE.
vals	by default, vals = FALSE and will return a named vector that indicates, for each unique element of x, whether that x element was a match for any element of patterns (one named logical element returned for each x vector element). If TRUE, vals returns the unique values of x that were matched (one character element for each matched x argument vector element).

Details

This function accepts a character vector of text substring patterns or regular expressions (patterns argument), and searches another character vector (x argument) to determine for each element of x, whether that element is a match (in the sense of `base::grepl()`) for any pattern element of of patterns. If vals = TRUE (default is FALSE), each matched x element is returned; if vals = FALSE, a vector of named logical values equal in length to x is returned, indicating for each x element whether it contains any text substring or pattern found in any element of patterns (TRUE if so, FALSE if not), with the names corresponding to the elements of the x vector. Used internally by various labelr functions. If ignore.case = TRUE (the default), neither vector is treated case-sensitively (both are coerced to lower-case before other operations).

Used internally by various labelr functions Note that this is the same search and syntax that is performed by `greml`(`gremlr` is "`greml` in reverse"), except that, whereas `greml` returns matches in terms of the patterns argument, `gremlr` returns matches in terms of x argument.

Value

a vector, either character (if vals = TRUE) or logical (if vals = FALSE).

Examples

```
# search for "AB" (case-sensitively) anywhere in subsequent vector
gremlr(c("AB"), c("ab", "ab", "abc", "z"), vals = TRUE, ignore.case = FALSE)
# character(0)

# search for "AB" (non-case-sensitively; the default) anywhere in next vector
gremlr(c("AB"), c("ab", "ab", "abc", "z"), vals = TRUE, ignore.case = TRUE)
# [1] "ab"  "ab"  "abc"

# other searches
gremlr(c("AB"), c("ab", "ab", "abc", "z"), vals = TRUE, ignore.case = FALSE)
gremlr(c("ab"), c("ab", "ab", "abc", "z"), vals = FALSE)
gremlr(c("ab", "Q"), c("ab", "ab", "abc", "z"), vals = FALSE)
gremlr(c("a|b", "Q"), c("a", "b", "abc", "z"), vals = FALSE)
gremlr(c("a|b", "Q"), c("a", "b", "z"), vals = FALSE)
gremlr(c("a|b", "Q"), c("bq", "z"), vals = FALSE)
```

```
# compare gremlr (above) to greml() (here)
greml(c("AB"), c("ab", "ab", "abc", "z"), vals = TRUE, ignore.case = FALSE)
greml(c("ab"), c("ab", "ab", "abc", "z"), vals = FALSE)
greml(c("ab", "Q"), c("ab", "ab", "abc", "z"), vals = FALSE)
greml(c("a|b", "Q"), c("a", "b", "abc", "z"), vals = FALSE)
greml(c("a|b", "Q"), c("a", "b", "z"), vals = FALSE)
greml(c("a|b", "Q"), c("bq", "z"), vals = FALSE)
```

---

has_avl_labs *Is This a* add_val_labs()*-style Value-labeled Variable (Column)?*

Description

Determine whether a specific variable of a data.frame has value labels associated with it that were added using add_val_labs() or add_val1().

Usage

```
has_avl_labs(data, var)

h11l(data, var)

hql(data, var)
```

Arguments

data	a data.frame.
var	the unquoted name of the variable (column) to check for the presence of add_val_labs()-style (one-to-one) value labels.

Details

h11l is a compact alias for has_avl_labs: they do the same thing, and the former is easier to type

Value

A 1L logical.

Examples

```
# add val labs to multiple variables at once
# make a "Likert"-type fake data set to demo
# note, by default, add_val_labs() "vars" arg will do partial matching
# in this case, we catch all vars with "y" in their name, except "y3"
set.seed(272)
dflik <- make_likert_data(scale = 1:7)
vals2label <- 1:7
labs2use <- c(
  "VSD",
```

```
  "SD",
  "D",
  "N",
  "A",
  "SA",
  "VSA"
)

dflik <- add_val_labs(
  data = dflik, vars = c("y"), # note the vars args
  not.vars = "y3",
  vals = vals2label,
  labs = labs2use,
  partial = TRUE
)

has_avl_labs(dflik, y1) # TRUE

has_avl_labs(dflik, y3) # FALSE, see not.vars arg above
```

---

has_decv *Determine if Vector Has Decimals*

Description

has_decv determines whether a vector has decimal values, using all values for smaller vectors and using a non-random sample of observations for larger vectors.

Usage

```
has_decv(x, sample.after = 1000)
```

Arguments

x	the vector to check for presence of decimals.
sample.after	for larger (length(x)>1000) vectors, test for decimals using a non-random sample of 1000 observations from the vector.

Details

This function is used by core labelr functions to detect vectors that are bad candidates for one-to-one value labeling (as implemented by, e.g., add_val_labs).

Value

a 1L vector indicating whether x has decimal values.

Examples

```
set.seed(123)
x1 <- runif(10)
x2 <- as.character(sample(c(1:20), 10, replace = TRUE))
x3 <- sample(letters, size = 10, replace = TRUE)
df <- data.frame(x1, x2, x3)
head(df, 3)
sapply(df, class)
class(df$x2)

df <- as_num(df)
head(df, 3)
sapply(df, class)

sapply(mtcars, is.double)
sapply(mtcars, is.numeric)
sapply(mtcars, is.integer)
sapply(mtcars, has_decv)
```

---

has_m1_labs *Is This an* add_m1_lab() *Many-to-One-Style Value-labeled Variable (Column)?*

Description

Determine whether a specific variable of a data.frame has many-to-one-style value labels associated with it (i.e., via add_m1_lab() or add1m1()).

Usage

```
has_m1_labs(data, var)

hm1l(data, var)
```

Arguments

data	a data.frame.
var	the unquoted name of the variable (column) to check for the presence of many-to-one-style value labels.

Details

hm1l is a compact alias for has_m1_labs: they do the same thing, and the former is easier to type

Value

A 1L logical.

Examples

```
# add many-to-one style labels for "carb" and one-to-one style for "am"
df <- mtcars

df <- add_m1_lab(df,
  vars = "carb",
  vals = 1:3,
  lab = "<=3",
  max.unique.vals = 10
)

df <- add_m1_lab(df,
  vars = "carb",
  vals = c(4, 6, 8),
  lab = ">=4",
  max.unique.vals = 10
)

df <- add_val_labs(df,
  vars = "am",
  vals = c(0, 1),
  labs = c("autom", "manu"),
  max.unique.vals = 10
)

has_m1_labs(df, carb) # TRUE, carb has m1-style value labels

has_val_labs(df, am) # TRUE, am does have value labels

has_m1_labs(df, am) # FALSE, am's value labels are not not m1-style labels
```

---

has_quant_labs *Is this an* add_quant_labs()*-style Value-labeled Variable (Column)?*

Description

Determine whether a specific variable of a data.frame has value labels associated with it that were added using add_quant_labs() or add_quant1().

Usage

```
has_quant_labs(data, var)
```

Arguments

data	a data.frame.
var	the unquoted name of the variable (column) to check for the presence of add_quant_labs()-style numerical range-based value labels.

Details

hql is a compact alias for has_quant_labs: they do the same thing, and the former is easier to type

Value

A 1L logical.

Examples

```
# copy mtcars to mt2 and assign various types of value labels
mt2 <- mtcars

# add 1-to-1 value labels
mt2 <- add_val_labs(
  data = mt2,
  vars = "am",
  vals = c(0, 1),
  labs = c("automatic", "manual")
)

has_val_labs(mt2, am) # TRUE, it does
has_m1_labs(mt2, am) # FALSE, they are NOT add_m1_lab()-style
has_quant_labs(mt2, am) # FALSE, they are NOT add_quant_labs() -style

# add many-to-1 value labels
mt2 <- add_m1_lab(
  data = mt2,
  vars = "gear",
  vals = 4:5,
  lab = "4+"
)

has_val_labs(mt2, gear) # TRUE, it does
has_m1_labs(mt2, gear) # TRUE, they ARE add_m1_lab()-style
has_quant_labs(mt2, gear) # FALSE, they NOT not add_quant_labs() -style

# add quartile-based numerical range value labels
mt2 <- add_quant_labs(
  data = mt2,
  vars = "disp",
  qtiles = 4
)

has_val_labs(mt2, disp) # TRUE, it does
has_m1_labs(mt2, disp) # FALSE, they are NOT add_m1_lab()-style
has_quant_labs(mt2, disp) # TRUE, they ARE add_quant_labs() -style
```

has_val_labs *Is This a Value-labeled Variable (Column)?*

Description

Determine whether a specific variable of a data.frame has value labels associated with it.

Usage

```
has_val_labs(data, var, type = "any")

hvl(data, var, type = "any")
```

Arguments

data	a data.frame.
var	the unquoted name of the variable (column) to check for the presence of value labels.
type	the type of value label to check the variable for: "any," "1to1," "m1," or "q." If "1to1," check will be for value labels added via `add_val_labs()` or `add_val1()`. If "m1," check will be for value labels added via `add_m1_lab()` or `add1m1()`. If "q," check will be for value labels added via `add_quant_labs()` or `add_quant1()`. If "any," will return TRUE if any of the foregoing value label types is found.

Details

`hvl` is a compact alias for `has_val_labs`: they do the same thing, and the former is easier to type

Value

A 1L logical.

Examples

```
# add val labs to multiple variables at once
# make a "Likert"-type fake data set to demo
# note, by default, add_val_labs() "vars" arg will do partial matching
# in this case, we catch all vars with "y" in their name, except "y3"
set.seed(272)
dflik <- make_likert_data(scale = 1:7)
vals2label <- 1:7
labs2use <- c(
  "VSD",
  "SD",
  "D",
  "N",
  "A",
  "SA",
```

```
  "VSA"
)

dflik <- add_val_labs(
  data = dflik, vars = c("y"), # note the vars args
  not.vars = "y3",
  vals = vals2label,
  labs = labs2use,
  partial = TRUE
)

has_val_labs(dflik, y1) # TRUE

has_val_labs(dflik, y3) # FALSE, see not.vars arg above
```

`headl` *Return First Rows of a Data Frame with Value Labels Visible*

Description

`headl` accepts a labelr value-labeled data.frame and returns the first n value-labeled rows of that data.frame

Usage

```
headl(data, n = 6L)
```

Arguments

`data`	a data.frame.
`n`	the last row to select (i.e., rows 1 through n will be returned).

Details

Whereas `utils::head` returns the first n rows of a data.frame, `headl` does the same thing, substituting value labels for values wherever the former exist. See also `taill` and `somel`.

Value

a data.frame.

Examples

```
# make toy demographic (gender, raceth, etc.) data set
set.seed(555)
df <- make_demo_data(n = 1000) # another labelr:: function
# let's add variable VALUE labels for variable "raceth"
df <- add_val_labs(df,
  vars = "raceth", vals = c(1:7),
```

```
  labs = c("White", "Black", "Hispanic", "Asian", "AIAN", "Multi", "Other"),
  max.unique.vals = 50
)

# let's add variable VALUE labels for variable "gender"
# note that, if we are labeling a single variable, we can use add_val1()
# distinction between add_val1() and add_val_labs() will become more meaningful
# when we get to our Likert example
df <- add_val1(
  data = df, gender, vals = c(0, 1, 2, 3, 4),
  labs = c("M", "F", "TR", "NB", "Diff-Term"), max.unique.vals = 50
)

head(df) # utils::head
headl(df) # same, but with value labels in place of values
```

`init_labs` *Initialize labelr Attributes*

Description

`init_labs` pre-populates a data.frame with "placeholder" labelr label meta- data, which will be overwritten if/when you explicitly assign your own preferred label attributes.

Usage

```
init_labs(data, max.unique.vals = 5000)
```

Arguments

`data`	the data.frame that you will be labeling via functions like `add_val_labs` and `add_name_labs`.
`max.unique.vals`	constrains the variables that may receive value labels to those whose total unique values do not exceed the integer value supplied to this argument. Note that labelr sets a hard ceiling of 5000 on the total number of unique value labels that any variable is permitted to have under any circumstance, as labelr is primarily intended for interactive use with moderately-sized (<=~1M-row) data.frames.

Details

`init_labs` is used inside other labelr functions but is not intended for interactive use at the console.

Value

a data.frame with initial placeholder labelr meta-data added.

Examples

```
# make toy demographic (gender, race, etc.) data set
set.seed(555)
df <- make_demo_data(n = 1000) # another labelr:: function
df2 <- init_labs(df) # df2 is not df
get_all_lab_atts(df) # this is df; is not df2
get_all_lab_atts(df2) # this is df2
```

---

irregular2 *Convert All "Irregular" Data Frame Values to NA or Other Specified Value*

Description

Check all (or specified) columns of a data.frame for the presence of "irregular" values (e.g., NA, Inf, NaN) and, if found, replace them with NA (or some other specified value).

Usage

```
irregular2(
  data,
  vars = NULL,
  to = NA,
  nan.include = TRUE,
  inf.include = TRUE,
  special = c("NA", "NAN", "INF", "-INF")
)
```

Arguments

`data`	a data.frame object.
`vars`	a character vector that corresponds to the name(s) of one or more variables (columns) to which the operational will be applied. If NULL, will be applied to all variables.
`to`	a single value (by default: NA) to which all irregular values will be converted (Note: if arg is character, returned vector will be coerced to character).
`nan.include`	convert NaN values to NA.
`inf.include`	convert Inf and -Inf values to NA.
`special`	additional specific character values that will be converted to NA values.

Details

For purposes of `irregular2`, irregular values consist of: NA values, other arbitrary values you specify, and (by default): NaN, Inf, -Inf, and character variants of same (i.e., upper, lower, or mixed-case variants of "NA","NAN", "INF","-INF"). This function converts all such values to NA (or some other specified value).

Value

a data.frame identical to data, with exception that irregular values have been converted to the value specified in "to" argument (NA, by default).

Examples

```
set.seed(123)
x <- c(NA, Inf, -Inf, NaN, runif(6))
y <- c("a", "inf", "NaN", NA, sample(letters, 6, replace = TRUE))
df <- data.frame(x, y)

head(df, 10)

df_1 <- irregular2(df)

head(df_1, 10)

df_2 <- irregular2(df, vars = "x", to = ";-)")

head(df_2)
```

---

irregular2v *Replace "Irregular" Values of a Vector with Some Other Value*

Description

Check a vector for the presence of "irregular" values (e.g., NA) and, if found, replace with some other (single) user-specified value.

Usage

```
irregular2v(
  x,
  to = 99,
  nan.include = TRUE,
  inf.include = TRUE,
  special = c("NA", "NAN", "INF", "-INF"),
  other = NULL
)
```

Arguments

x	a vector.
to	a single value to which all NA (and other irregular) values should be converted (Note: if arg is character, returned vector will be coerced to character).
nan.include	treat NaN values as NA.
inf.include	treat Inf and -Inf values as NA.

special	a modifiable set of default character values that will be treated as equivalent to NA values.
other	additional user-specified values (of consistent class) that will be treated as equivalent to NA values.

Details

For purposes of irregular2v, irregular values consist of: NA values, other arbitrary values you specify, and (by default): NaN, Inf, -Inf, and character variants of same (i.e., upper, lower, or mixed-case variants of "NA","NAN", "INF","-INF"). This function is used insider core labelr functions to manage such values, but the typical labelr user will have no explicit need to use it as part of an interactive session.

Value

a vector identical to x, with exception that NA (and other irregular) values have been converted to the value specified in "to" argument.

Examples

```
set.seed(123)
x <- c(NA, Inf, -Inf, NaN, runif(6))
x
x1 <- irregular2v(x, to = 33)
x1

set.seed(123)
x1 <- c(NA, Inf, -Inf, NaN, runif(6))
x
x1 <- irregular2v(x, to = 33, nan.include = TRUE, inf.include = FALSE)
x1

set.seed(123)
x <- c(NA, "INF", "in", "nan", "NA", sample(letters, 5))
x
x1 <- irregular2v(x, to = "<-X->")
x1
```

---

is_numable *Test Whether Character Vector Is "Suitable" for Numeric Conversion*

Description

is_numable determines whether a character vector can be coerced to numeric without generating new NA values.

Usage

```
is_numable(x, nan2na = TRUE, inf2na = TRUE)
```

Arguments

x	a character vector.
nan2na	treat NaN (including, e.g., "nan") values as NA values.
inf2na	treat Inf, -Inf values (including, e.g., "inf") as NA values. treated as equivalent to NA values.

Details

Core labelr functions coerce integers to characters and back, which `is_numable` facilitates.

Value

a 1L (scalar) logical vector.

Examples

```
set.seed(123)
x1 <- runif(10)
x2 <- as.character(sample(c(1:20), 10, replace = TRUE))
x2_num_test <- is_numable(x2)
x2_num_test
x3 <- sample(LETTERS, 10, replace = TRUE)
x3_num_test <- is_numable(x3)
x3_num_test
```

---

lab_int_to_factor *Convert a Value-labeled Integer Variable Column to a Factor Variable Column*

Description

`lab_int_to_factor` converts a value-labeled integer variable to a factor, using labelr value labels as factor level labels and returning the modified data.frame.

Usage

```
lab_int_to_factor(data, var, ordered = FALSE)

int2f(data, var, ordered = FALSE)
```

Arguments

data	a data.frame object.
var	the (unquoted) name of a value-labeled integer variable found in data.
ordered	logical flag to determine if resulting factor levels should be regarded as ordered (in ascending order of the integer values of var).

Details

Note 1: `int2f` is a compact alias for `lab_int_to_factor`: they do the same thing, and the former is easier to type.

Note 2: This function can be used to produce ordered factors but will not do so by default (see argument ordered).

Note 3: This function's effects are NOT straightforwardly "undone" by `factor_to_lab_int()` See the latter's documentation for more information and an example demonstration.

Value

a data.frame.

Examples

```
class(iris[["Species"]])

iris_sp_int <- factor_to_lab_int(iris, Species)
class(iris_sp_int[["Species"]])

get_val_labs(iris_sp_int, "Species")
iris_sp_fac <- lab_int_to_factor(iris_sp_int, Species)

class(iris_sp_fac[["Species"]])

levels(iris_sp_fac[["Species"]])

# copy data.frame mtcars to mt2
mt2 <- mtcars

# add value labels to mtcars$carb and assign data.frame to object mt2
mt2 <- add_val_labs(
  data = mt2,
  vars = "carb",
  vals = c(1, 2, 3, 4, 6, 8),
  labs = c(
    "1c", "2c", # a tad silly, but these val labels will demo the principle
    "3c", "4c",
    "6c", "8c"
  )
)

# carb as labeled integer
class(mt2$carb)
levels(mt2$carb)
head(mt2$carb, 3)
lm(mpg ~ carb, data = mt2)
(adj_r2_int <- summary(lm(mpg ~ carb, data = mt2))$adj.r.squared)
AIC(lm(mpg ~ carb, data = mt2))

# carb as factor
carb_fac <- mt2 # copy mt2 to new data.frame carb_fac
```

```
carb_fac <- lab_int_to_factor(carb_fac, carb) # alias int2f() also works
class(carb_fac$carb)
levels(carb_fac$carb)
head(carb_fac$carb, 3)
lm(mpg ~ carb, data = carb_fac)
(adj_r2_fac <- summary(lm(mpg ~ carb, data = carb_fac))$adj.r.squared)
AIC(lm(mpg ~ carb, data = carb_fac))
```

`make_demo_data` *Construct a Fake Demographic Data Frame*

Description

`make_demo_data` generates a data.frame with select (entirely fictional) "demographic" variables purely for the purposes of demonstrating or exploring common labelr behaviors and uses and is not designed to accurately emulate or represent the frequencies or relationships among demographic variables.

Usage

```
make_demo_data(
  n = 1000,
  age.mean = 43,
  age.sd = 15,
  gend.prob = c(0.45, 0.45, 0.045, 0.045, 0.01),
  raceth.prob = c(1/7, 1/7, 1/7, 1/7, 1/7, 1/7, 1/7),
  edu.prob = c(0.03, 0.32, 0.29, 0.24, 0.12),
  rownames = TRUE
)
```

Arguments

`n`	number of observations (rows) of hypothetical data set to create.
`age.mean`	mean value of (fictional) age variable (assuming a normal distribution) recorded in a hypothetical data set.
`age.sd`	standard deviation of (fictional) age variable (assuming a normal distribution) recorded in a hypothetical data set.
`gend.prob`	probabilities of four categories of a gender identity variable recorded in a hypothetical data set.
`raceth.prob`	probabilities of categories of a hypothetical race/ethnicity variable recorded in a hypothetical data set.
`edu.prob`	probabilities of categories of a hypothetical "highest level of education" variable recorded in a hypothetical data set.
`rownames`	create memorable but arbitrary rownames for inspection (if TRUE).

Value

a data.frame.

Examples

```
# make toy demographic (gender, race, etc.) data set
set.seed(555)
df <- make_demo_data(n = 1000)
df <- add_val_labs(df,
  vars = "raceth", vals = c(1:7),
  labs = c("White", "Black", "Hispanic", "Asian", "AIAN", "Multi", "Other"),
  max.unique.vals = 50
)
head(df)
summary(df)
```

---

make_likert_data *Construct a Fake Likert Survey Response Data Frame*

Description

make_likert_data generates a data.frame with select (entirely fictional) numerically coded responses of fictional people to fictional survey items to demonstrate and explore labelr functionalities.

Usage

```
make_likert_data(n = 1000, scale = 1:7, rownames = TRUE)
```

Arguments

n	number of observations (rows) of hypothetical data set to create.
scale	the sequence of distinct integer values describing the raw / naive numerical codings of Likert-type survey items.
rownames	create memorable but arbitrary rownames for inspection (if TRUE).

Details

Data is entirely fictional and strictly for purposes of demonstrating labelr.

Value

a data.frame.

Examples

```
# add_val_labs() "vars" arg will do partial matching if partial = TRUE
# in this case, we catch all vars with "x" in their name
set.seed(272)
dflik <- make_likert_data(scale = 1:7)
vals2label <- 1:7
labs2use <- c(
  "VSD",
  "SD",
  "D",
  "N",
  "A",
  "SA",
  "VSA"
)

dflik <- add_val_labs(
  data = dflik, vars = c("x", "y3"), # note the vars args
  vals = vals2label,
  labs = labs2use,
  partial = TRUE
)

# note, all "x" vars get the labs, as does "y3"
# see vars = args above
lik1 <- use_val_labs(dflik)
head(lik1)
```

---

recode_vals *Recode Values of a Free-standing Vector*

Description

Takes a stand-alone vector (x), and recodes select values (bef) to some other set of values (aft), returning the recoded vector.

Usage

```
recode_vals(x, bef, aft, default.lab = "bef", unique = FALSE)
```

Arguments

x	an integer, character, factor, or logical vector.
bef	the "before" (i.e., current) values of x to be recoded.
aft	the "after" (recoded) values to be substituted in the returned vector in place of the positionally corresponding bef values of the x vector ("positionally corresponding" means that the first element of aft is the replacement (recode) for all x instances of the first element of bef, and so on for the respective second bef

and aft elements, etc.; see examples). variables to which value labels will be added.

`default.lab` the "aft" value to be used for values of x for which no "bef" value is specified. default.lab = "bef" (the default) will use (retain) the existing value of x as its own recode, coercing to character as needed. For example, if the value x=4 is observed in x but is not included in the "bef" argument, the returned vector will have values of 4 (integer) or "4" (character), depending on whether the recodes that –are– supplied are numeric (then 4) or character (then "4").

`unique` if TRUE, return only the mapping itself (bef argument values as names, aft argument values as values), else if FALSE (default), return the full recoded vector of x values.

Details

While labelr users do not need to engage `recode_vals` directly, it is the underlying function that powers certain core labelr functions, including `add_val_labs` and `add_name_labs`. The bef argument identifies the values of x to recode, and aft argument indicates what each bef value should be recoded to (order matters: bef=c("a", "b", "c"), aft=c(1, 2, 3) means that "a" values of x will be recoded to 1 values in returned vector, "b" values will be recoded to 2, and "c" values will be recoded to 3).

Value

A vector of length equal length of supplied vector, with x values found in bef argument switched to the corresponding values found in the aft argument.

Examples

```
z <- mtcars$gear
z[1] <- NA
z
recode_vals(z, c(5, 3, 4), c("five", "three", "four"))
irsp <- iris$Species[c(1:3, 60:62, 148:150)]
irsp
recode_vals(irsp, c("setosa", "versicolor", "virginica"), c("SE", "VE", "VI"))
class(irsp) # factor
class(recode_vals(
  irsp, c("setosa", "versicolor", "virginica"),
  c("SE", "VE", "VI")
)) # coerced to character
set.seed(112)
x_logic <- sample(c(TRUE, FALSE), 10, replace = TRUE)
x_logic
recode_vals(x_logic, bef = c(FALSE), c("Fake News!"))
```

restore_factor_info *Restore Factor Status, Levels to a Character Column of a Labeled Data Frame*

Description

restore_factor_info searches a data.frame for labelr-specific factor meta-data (added by add_factor_info()) and, if found, uses that information to coerce a character vector that was formerly a factor back into a factor, with former levels and (if applicable) "ordered" factor status, as well.

Usage

```
restore_factor_info(data)
```

Arguments

data	the data.frame to which labelr-specific factor attribute meta- data may have been applied via add_factor_info.

Value

A data.frame.

Examples

```
# this function does not strictly require prior or other use of labelr
zz <- add_factor_info(iris) # we'll find out what this does
sapply(zz, class) # Species is a factor
zz$Species <- as.character(zz) # now it's a character
sapply(zz, class) # yup, it's a character
zz <- restore_factor_info(zz) # we'll find out what this does
sapply(zz, class) # now it's back to a factor
levels(zz$Species) # levels are back, too.
```

sbrac *Safely Extract Elements of a Labeled Data Frame*

Description

sbrac allows one to do "bracket-like" row and/or column selection (without actual brackets) on a labelr-labeled data.frame in a manner that preserves its labelr label attributes.

Usage

```
sbrac(data, ri = NULL, ci = NULL, partial = FALSE)
```

Arguments

data	the data.frame from which columns will be selected.
ri	row indices (integer positional or logical) or row.names (or partial row.names if partial = TRUE) to be selected.
ci	column indices (integer positional or logical) or column / variable names (or partial column names if partial = TRUE) to be selected.
partial	if TRUE, any row or column that contains the relevant character substring will be selected (e.g., sbrac(mtcars, c("Maz"), c("di")) will return all "Mazda" car rows and the column "disp").

Details

sbrac allows one to select rows and columns of a data.frame in a bracket- like fashion, using integers (positional indices), logicals, or (partial) character names (row names and column/variable names). For example, sbrac(df, 1:5, 2:4) returns a data.frame (with any labelr attribute meta-data intact), consisting of rows 1-5 and columns 2-4 of data.frame df; while sbrac(mtcars, "Maz", "a", partial = TRUE) returns all variables with substring "a" in their names for all rows with substring "Maz" in their row names. Integer indices (only) can be enclosed in -c() for negative index selection (i.e., "select not these"), where, e.g., sbrac(df, -c(1,2), "x", partial = TRUE) means select all rows of df other than rows 1 and 2 and all columns that feature the letter "x" in their names. See also ssubset, sfilter, sselect, sdrop, ssort, and srename, as well as slab and flab for other labelr attribute-preserving subsetting tools.

Value

a labelr label attribute-preserving data.frame, consisting of the selected rows and/or columns index-selected.

Examples

```
# create a copy of the mtcars data set
mtc2 <- mtcars

# variable names and their labels
names_labs_vec <- c(
  "mpg" = "Miles/(US) gallon",
  "cyl" = "Number of cylinders",
  "disp" = "Displacement (cu.in.)",
  "hp" = "Gross horsepower",
  "drat" = "Rear axle ratio",
  "wt" = "Weight (1000 lbs)",
  "qsec" = "1/4 mile time",
  "vs" = "Engine (0 = V-shaped, 1 = straight)",
  "am" = "Transmission (0 = automatic, 1 = manual)",
  "gear" = "Number of forward gears",
  "carb" = "Number of carburetors"
)

# assign variable labels
```

```
mtc2 <- add_name_labs(mtc2,
  vars = names(names_labs_vec),
  labs = names_labs_vec
)

# examples of sbrac() functionality
sbrac(mtc2, 1:4, ) # commas used in a bracket-like way: row 1:4 and all cols
sbrac(mtc2, , 1:4) # commas used in a bracket-like way: all rows and cols 1:4
sbrac(mtc2, 1, 2) # 1 is row, 2 is col
sbrac(mtc2, -c(8:32), -c(1:8)) # select NOT-these rows and cols (-)
sbrac(mtc2, 1:5, 1:2) # rows 1-5, cols 1 and 2

# if partial = TRUE, partial matching to get all Mazda or Merc + all
# ...vars with "ar" in name
sbrac(mtc2, c("Mazda", "Merc"), c("ar"), partial = TRUE) # see what this does
mtc3 <- sbrac(mtc2, c("45"), 1:2, partial = TRUE) # see what this does
get_labs_att(mtc3, "name.labs") # name.labs still there
```

`scbind` *Safely Combine Data Frames Column-wise*

Description

`scbind` allows one to bind columns together into a data.frame, while preserving any labelr labels of the inputted data.frames.

Usage

```
scbind(...)
```

Arguments

`...`	data.frames to be column-bound

Details

Precedence is given to the labels of earlier-appearing arguments, such that, if both an earlier and a later data.frame include a label attribute with the same name, the attribute from the earlier data.frame will be preserved, and the same-named attribute later data.frame(s) will be discarded.

Value

a data.frame.

Examples

```
# assign mtcars to df
df <- mtcars

# add value labels to "am"
df <- add_val_labs(
  data = df,
  vars = "am",
  vals = c(0, 1),
  labs = c("automatic", "manual")
)

# add numerical range value labels to "mpg"
df <- add_quant1(df, mpg, qtiles = 4)

df_a <- sselect(df, "am")
df_b <- sselect(df, "mpg")
df_c <- sselect(mtcars, "cyl")
df_all <- scbind(df_a, df_b, df_c)

head(df_all)

get_val_labs(df_all)
```

---

schange *Safely Change or Add a Data Frame Variable (Column)*

Description

schange allows one to modify or add a column to a data.frame while preserving the labelr attributes attached to the inputted data.frame.

Usage

```
schange(data, ...)
```

Arguments

data	a data.frame.
...	an expression that will create or update a column in data.

Details

Note that, while schange preserves label meta-data of the supplied data.frame, it does not update or add any new labels. Therefore, if you are altering the range of values for an extant variable or adding a new variable to the data.frame, you will need to explicitly instantiate any new or modified labels that you desire via follow-up calls to functions such as `add_val_labs()`, `drop_val_labs()`, etc.

Value

a data.frame.

Examples

```
df <- mtcars
# now, add value labels
df <- add_val_labs(
  data = df,
  vars = "am",
  vals = c(0, 1),
  labs = c("automatic", "manual")
)

df <- add_val_labs(
  data = df,
  vars = "carb",
  vals = c(1, 2, 3, 4, 6, 8),
  labs = c(
    "1-carb", "2-carbs",
    "3-carbs", "4-carbs",
    "6-carbs", "8-carbs"
  )
)

df <- schange(df, mpg_sq = mpg^2) # create var "mpg_sq"
df <- schange(df, am = ifelse(am == 0, 2, am)) # modify/replace var "am"

head(df, 4) # show that data.frame modifications have been made
get_all_lab_atts(df) # labels are intact; "val.labs.am" needs updating!
df <- drop_val_labs(
  data = df,
  vars = "am"
)

df <- add_val_labs(
  data = df,
  vars = "am",
  vals = c(1, 2),
  labs = c("manual", "automatic")
)
get_val_labs(df)
```

---

sdrop *Safely Drop Specified Columns of a Labeled Data Frame*

Description

sdrop allows one to remove columns from a data.frame, returning the remaining columns as a data.frame that preserves the labelr attributes attached to the inputted data.frame.

Usage

```
sdrop(data, ...)
```

Arguments

data	the data.frame from which columns will be removed.
...	comma-separated, unquoted column/variable names to be discarded (e.g., cyl, mpg, not c("cyl", "mpg")), with no other special characters or symbols, such as quotes, parentheses, colons, minus signs, exclamation points, or other operators.

Details

This function accepts a data.frame, followed by a set of comma-separated, non-quoted column names to be discarded and returns the remaining columns as a data.frame that preserves labelr attribute information. NOTE: This command does NOT allow for positive specification of columns to be retained; rather, all variables not specified will be retained by default. Further, `sdrop` does not supported quoted column names, dplyr-like helper functions or other special selection syntax or idioms. See also `ssubset`, `sselect`, or `sbrac`); see also `sfilter`, `ssort`, `srename`, `slab`, and `flab`.

Value

a labelr label attribute-preserving data.frame consisting of the remaining (i.e., non-specified, non-discarded) subset of columns of the supplied data.frame.

Examples

```
# make toy demographic (gender, raceth, etc.) data set
set.seed(555)
df <- make_demo_data(n = 1000) # another labelr:: function
# let's add variable VALUE labels for variable "raceth"
df <- add_val_labs(df,
  vars = "raceth", vals = c(1:7),
  labs = c("White", "Black", "Hispanic", "Asian", "AIAN", "Multi", "Other"),
  max.unique.vals = 50
)

head(df, 3)
check_labs_att(df, "val.labs.raceth") # "raceth" lab specifically TRUE

dfless <- sdrop(df, id, raceth) # drop the vars id and raceth
head(dfless, 3) # selection worked
check_labs_att(dfless, "val.labs.raceth") # "raceth" value labels are gone
```

sfilter *Safely Filter Rows of a Labeled Data Frame*

Description

sfilter allows one to filter-subset a data.frame, selecting only rows that satisfy conditions (after the fashion of base::subset() or dplyr::filter()), returning the selected rows as a data.frame that preserves the labelr attributes attached to the inputted data.frame.

Usage

```
sfilter(data, condition)
```

Arguments

data	the data.frame from which columns will be selected.
condition	row-filtering conditions along the lines of base::subset() and/or dplyr::select().

Details

This function accepts a data.frame, followed by conditional filtering instructions and returns the selected rows (and all inputted data.frame columns) as a data.frame that preserves the labelr attribute information of the originally supplied data.frame. See ssubset for a variant that combines sfilter row-filtering and sselect column selection in a single function. See sbrac for a labelr attribute-preserving approach to row and/or column indexing. See also sdrop, ssort, srename, slab, and flab.

Value

a labelr label attribute-preserving data.frame consisting of the selected rows that meet the filtering condition(s).

Examples

```
# make toy demographic (gender, raceth, etc.) data set
set.seed(555)
df <- make_demo_data(n = 1000) # another labelr:: function
# let's add variable VALUE labels for variable "raceth"
df <- add_val_labs(df,
  vars = "raceth", vals = c(1:7),
  labs = c("White", "Black", "Hispanic", "Asian", "AIAN", "Multi", "Other"),
  max.unique.vals = 50
)

head(df, 3)
check_labs_att(df, "val.labs.raceth") # "raceth" lab specifically TRUE

dffil1 <- sfilter(df, raceth %in% c(1, 2, 3) & id < 50 & gender == 1)
```

```
head(dffil1, 3)
check_labs_att(dffil1, "val.labs.raceth") # "raceth" lab specifically TRUE

dffil2 <- sfilter(df, !raceth %in% 1:5 | (age == 60))
head(dffil2, 8)
check_labs_att(dffil2, "val.labs.raceth") # "raceth" lab specifically TRUE
```

---

sgen *Safely Generate a Data Frame Variable (Column)*

Description

sgen allows one to add a column to a data.frame while preserving the labelr attributes attached to the inputted data.frame.

Usage

```
sgen(data, ...)
```

Arguments

data	a data.frame.
...	an expression that will create a new column in the data.frame.

Details

Note that, while sgen preserves label meta-data of the supplied data.frame, it does not update or add any new labels. Therefore, you will need to explicitly instantiate any new or modified labels that you desire via follow-up calls to functions such as add_val_labs(), drop_val_labs(), etc.

You may not use sgen to replace a variable already present in a the supplied data.frame. For that, see sreplace or schange.

Value

a data.frame.

Examples

```
df <- mtcars
# now, add value labels
df <- add_val_labs(
  data = df,
  vars = "am",
  vals = c(0, 1),
  labs = c("automatic", "manual")
)

df <- add_val_labs(
```

```
  data = df,
  vars = "carb",
  vals = c(1, 2, 3, 4, 6, 8),
  labs = c(
    "1-carb", "2-carbs",
    "3-carbs", "4-carbs",
    "6-carbs", "8-carbs"
  )
)

df <- sgen(df, mpg_sq = mpg^2) # create var "mpg_sq"
df <- sgen(df, am2 = ifelse(am == 0, 2, am)) # create var "am2"

head(df, 4) # show that data.frame modifications have been made
get_all_lab_atts(df)

df <- add_quant_labs(
  data = df,
  vars = "mpg_sq",
  vals = c(200, 400, 600, 1000, 1500),
  labs = NULL
)

df <- add_val_labs(
  data = df,
  vars = "am2",
  vals = c(1, 2),
  labs = c("manual", "automatic")
)


get_val_labs(df)
```

---

slab *Subset a Data Frame Using Value Labels*

Description

slab ("subset using labels") allows one to filter rows and select columns from a data.frame based on variable-specific value label attributes.

Usage

```
slab(data, condition, ...)
```

Arguments

data	the data.frame from which columns rows will be filtered (and, possibly, columns selected)

condition — row-filtering conditions along the lines of base::subset() and/or dplyr::filter(). Note: Row-filtering conditions (to include condition==NULL) must be supplied. Conditions involving value-labeled variables must be expressed in terms of the value labels (see examples), else try ssubset.

... — Optionally supply one or more unquoted, comma-separated column names that identify columns to be retained (Note: If no columns are listed, all columns will be retained). Note: While row-filtering conditions may leverage standard operators (e.g., &, |, ==, !=), the column-selection portion of call may not incorporate special characters or symbols, such as quotes, parentheses, colons, minus signs, exclamation points, or other operators. Only positive selection of columns is permitted (negative selection – i.e., "select not / select-all-except" specified columns is not supported)

Details

slab does base::subset-style data subsetting using variable value label meta-data that are associated with variable values but are not themselves values (i.e., will not appear in response to View(), head(), etc.). In other words, value labels are supplied to the slab() call to direct the filtering process, but those value labels are not displayed in the cells of the returned data.frame – the raw values themselves are. This functionality may be useful for interactively subsetting a data.frame, where character value labels may be more intuitive and easily recalled than the underlying variable values themselves (e.g., raceth=="White" & gender="F" may be more intuitive or readily recalled than raceth==3 & gender==2).

slabtakes as its arguments a labelr value-labeled data.frame, followed by condition-based row-filtering instructions (required) and a list of unquoted names of variables to be retained (optional; all variables returned by default).

Note 1: When using slab, any conditional row-filtering syntax involving value-labeled variables must be expressed in terms of those variables' value labels, not the raw values themselves. Filtering on non-value-labeled variables is also permitted, with those variables' filtering conditions being expressed in terms of raw values. Further, slab() calls may reference both types of columns (i.e., value-labeled variables and non-value-labeled variables), provided filtering conditions for the former are expressed in terms of value labels.

Note 2: slab (and labelr more broadly) is intended for moderate-sized (or smaller) data.frames, defined loosely as those with a few million or fewer rows. With a conventional (c. 2024) laptop, labelr operations on modest- sized (~100K rows) take seconds (or less); with larger (> a few million rows) data.frames, labelr may take several minutes (or run out of memory and fail altogether!), depending on the complexity of the call and the number and type of cells implicated in it.

See also flab, use_val_labs, add_val_labs, add_val1,add_quant_labs, add_quant1, get_val_labs, drop_val_labs. For label-preserving subsetting tools that subset in terms of raw values (not value labels), see sfilter, sbrac, ssubset, sdrop.

Value

a labelr label attribute-preserving data.frame consisting of the selected rows that meet the filtering condition(s) and the columns whose (unquoted, comma-separated) names are passed to dots (...) (or all columns if no column names are passed).

Examples

```
# make toy demographic (gender, raceth, etc.) data set
set.seed(555)
df <- make_demo_data(n = 1000)

# let's add variable VALUE labels for variable "raceth"
df <- add_val_labs(df,
  vars = "raceth", vals = c(1:7),
  labs = c("White", "Black", "Hispanic", "Asian", "AIAN", "Multi", "Other"),
  max.unique.vals = 50
)

# let's add variable VALUE labels for variable "gender"
# note that, if we are labeling a single variable, we can use add_val1()
# distinction between add_val1() and add_val_labs() will become more
# meaningful when we get to our Likert example
df <- add_val1(
  data = df, gender, vals = c(0, 1, 2, 3, 4),
  labs = c("M", "F", "TR", "NB", "Diff-Term"), max.unique.vals = 50
)

# see what we did
# get_val_labs(df)
get_val_labs(df, "gender")
get_val_labs(df, "raceth")

# use --labels-- to subset w/ slab() ("*S*ubset using *lab*els")
dflab <- slab(df, raceth == "Asian" & gender == "F", id, gender)
head(dflab, 4)

# equivalently, use --values--- to filter w/ sfilter() ("*S*afe filter")
dfsf <- ssubset(df, raceth == 3 & gender == 1, gender, raceth)
head(dfsf, 4)
```

`smerge` *Safely Merge Two Data Frames*

Description

`smerge` allows one to merge two data.frames, one or both of which may be labeled, preserving the labelr attributes of the inputted data.frames.

Usage

```
smerge(x, y, ...)
```

Arguments

`x`	a data.frame to be merged with y.

y	a data.frame to be merged with x.
...	additional arguments passed to `base::merge()`

Details

Precedence is given to the labels of the first data.frame (argument x), such that, if both data.frames include a label attribute with the same name, the attribute from data.frame x will be preserved, and the same-named attribute from data.frame y will be discarded.

Value

a data.frame.

Examples

```
# assign mtcars to df
df <- mtcars

# assign the rownames to a column
id <- rownames(df)

df <- cbind(id, df)

# split the data.frame into two
df_a <- df[c("id", "am")]
df_b <- df[c("id", "mpg")]

# add value labels to df_a$am
df_a <- add_val_labs(
  data = df_a,
  vars = "am",
  vals = c(0, 1),
  labs = c("automatic", "manual")
)

# add numerical range value labels to df_b$mpg
df_b <- add_quant1(df_b, mpg, qtiles = 4)

# now, safely merge them
df_m <- smerge(df_a, df_b)

head(df_m)

get_val_labs(df_m)
```

---

somel *Return a Random Sample of Data Frame Rows with Value Labels Visible*

Description

somel accepts a labelr value-labeled data.frame and returns a random sample of n value-labeled rows of that data.frame

Usage

```
somel(data, n = 6L)
```

Arguments

data	a data.frame.
n	the number of random rows of the data.frame to return.

Details

somel is inspired by the function some from the car package. See also headl and taill.

Value

a data.frame.

Examples

```
# make toy demographic (gender, raceth, etc.) data set
set.seed(555)
df <- make_demo_data(n = 1000) # another labelr:: function
# let's add variable VALUE labels for variable "raceth"
df <- add_val_labs(df,
  vars = "raceth", vals = c(1:7),
  labs = c("White", "Black", "Hispanic", "Asian", "AIAN", "Multi", "Other"),
  max.unique.vals = 50
)

# let's add variable VALUE labels for variable "gender"
# note that, if we are labeling a single variable, we can use add_val1()
# distinction between add_val1() and add_val_labs() will become more meaningful
# when we get to our Likert example
df <- add_val1(
  data = df, gender, vals = c(0, 1, 2, 3, 4),
  labs = c("M", "F", "TR", "NB", "Diff-Term"), max.unique.vals = 50
)

somel(df) # six random rows with value labels visible
```

`sort_val_labs` *Sort Ascending Any Variable Value Labels*

Description

`sort_val_labs` sorts the presentation order of variable value label meta-data.

Usage

```
sort_val_labs(data)
```

Arguments

`data`	a data.frame

Details

This function is used in internally by other labelr functions to ensure that value label meta-data is sorted in a logical, intuitive order. It is not intended for interactive use.

Value

a data.frame

Examples

```
# note that this example is trivial, as value labels already are in order
df <- mtcars
df <- add_val_labs(
  data = df,
  vars = "am",
  vals = c(0, 1),
  labs = c("automatic", "manual")
)

df <- add_val_labs(
  data = df,
  vars = "carb",
  vals = c(1, 2, 3, 4, 6, 8),
  labs = c(
    "1-carb", "2-carbs",
    "3-carbs", "4-carbs",
    "6-carbs", "8-carbs"
  )
)

df <- sort_val_labs(df)
```

srbind *Safely Combine Data Frames Row-wise*

Description

srbind allows one to bind rows together into a data.frame, while preserving any labelr labels of the inputted data.frames.

Usage

```
srbind(...)
```

Arguments

...	data.frames to be row-bound

Details

Precedence is given to the labels of earlier-appearing arguments, such that, if both an earlier and a later data.frame include a label attribute with the same name, the attribute from the earlier data.frame will be preserved, and the same-named attribute later data.frame(s) will be discarded.

Value

a data.frame.

Examples

```
# assign mtcars to df
df <- mtcars

# assign the rownames to a column
id <- rownames(df)

df <- cbind(id, df)

# now, add value labels
df <- add_val_labs(
  data = df,
  vars = "am",
  vals = c(0, 1),
  labs = c("automatic", "manual")
)

df <- add_val_labs(
  data = df,
  vars = "carb",
  vals = c(1, 2, 3, 4, 6, 8),
  labs = c(
```

```
    "1-carb", "2-carbs",
    "3-carbs", "4-carbs",
    "6-carbs", "8-carbs"
  )
)

# split the data.frame into three
aa <- df[1:5, ]
bb <- df[6:11, ]
cc <- df[12:32, ]
# put them back together
df2 <- srbind(aa, bb, cc, cc)

get_val_labs(df2)
```

---

srename *Safely Rename a Variable and Preserve Its Value Labels*

Description

Note: srename renames and existing variable and preserves its value labels, overwriting an existing variable only if option force = TRUE.

Usage

```
srename(data, old.name, new.name, force = FALSE)
```

Arguments

data	a data.frame.
old.name	the unquoted name of the existing variable being renamed (to new.name).
new.name	the unquoted name that will be used to rename the variable specified in the old.name argument.
force	if a variable with the same name as new.name already exists in the data.frame, allow it to be overwritten. If FALSE, this will not be allowed, and an error will be issued.

Details

Any non-labelr R operation that changes a variable's (column's) name or that copies its contents to another variable (column) with a different name will not associate the original variable's value labels with the new variable name. To mitigate this, srename allows one to rename a data.frame variable while preserving its value labels – that is, by associating the old.name's value labels with the new.name. If the old.name variable (column) has a name label (in "name.labs" attribute), the column name associated with that name label will be changed from old.name to new.name.

Value

A data.frame.

Examples

```
# make toy demographic (gender, raceth, etc.) data set
set.seed(555)
df <- make_demo_data(n = 1000) # another labelr:: function
# let's add variable VALUE labels for variable "raceth"
df <- add_val_labs(df,
  vars = "raceth", vals = c(1:7),
  labs = c("White", "Black", "Hispanic", "Asian", "AIAN", "Multi", "Other"),
  max.unique.vals = 50
)

head(df, 4)
df <- srename(df, old.name = gender, new.name = genid)
df <- srename(df, old.name = raceth, new.name = racid)
df <- srename(df, old.name = x1, new.name = var1)
head(df, 4)
```

---

sreplace *Safely Replace a Data Frame Variable (Column)*

Description

sreplace allows one to replace a data.frame column with new values, while preserving the labelr attributes attached to the inputted data.frame.

Usage

```
sreplace(data, ...)
```

Arguments

data	a data.frame.
...	an expression that replaces a column in the data.frame.

Details

Note that, while sreplace preserves label meta-data of the supplied data.frame, it does not update or add any new labels. Therefore, if you are altering the range of values for an extant variable, you will need to explicitly instantiate any new or modified labels that you desire via follow-up calls to functions such as add_val_labs(), drop_val_labs(), etc.

You may not use sreplace to generate a new variable (i.e., one not already present in a the supplied data.frame). For that, see sgen or schange.

Value

a data.frame.

Examples

```
df <- mtcars
# now, add value labels
df <- add_val_labs(
  data = df,
  vars = "am",
  vals = c(0, 1),
  labs = c("automatic", "manual")
)

df <- add_val_labs(
  data = df,
  vars = "carb",
  vals = c(1, 2, 3, 4, 6, 8),
  labs = c(
    "1-carb", "2-carbs",
    "3-carbs", "4-carbs",
    "6-carbs", "8-carbs"
  )
)

df <- sreplace(df, mpg = mpg^2) # replace var "mpg"
df <- sreplace(df, am = ifelse(am == 0, 2, am)) # replace var "am"

head(df, 4) # show that data.frame modifications have been made
get_all_lab_atts(df)

df <- add_quant_labs(
  data = df,
  vars = "mpg",
  vals = c(200, 400, 600, 1000, 1500),
  labs = NULL
)

df <- drop_val_labs(
  data = df,
  vars = "am"
)

df <- add_val_labs(
  data = df,
  vars = "am",
  vals = c(1, 2),
  labs = c("manual", "automatic")
)

get_val_labs(df)
```

`sselect` *Safely Select Specified Columns of a Labeled Data Frame*

Description

sselect allows one to subset (select) columns from a data.frame, returning the selected columns as a data.frame that preserves the labelr attributes attached to the inputted data.frame.

Usage

```
sselect(data, ...)
```

Arguments

data	the data.frame from which columns will be selected.
...	comma-separated, unquoted column/variable names to be selected (e.g., cyl, mpg, not c("cyl", "mpg")), with no other special characters or symbols, such as quotes, parentheses, colons, minus signs, exclamation points, or other operators.

Details

This function accepts a data.frame, followed by a set of comma-separated, non-quoted column names to be retained and returns the selected columns in a data.frame that preserves labelr attribute information. NOTE: This command does not allow for negative selection, quoted columns, or dplyr-like helper functions or special selection idioms, but: see sdrop for negative selection ("return all columns except these"); see sbrac for a more flexible subsetting command; and see also ssubset, sfilter, ssort, and srename, as well as slab and flab.

Value

a labelr label attribute-preserving data.frame consisting of the selected subset of columns.

Examples

```
# make toy demographic (gender, raceth, etc.) data set
set.seed(555)
df <- make_demo_data(n = 1000) # another labelr:: function
# let's add variable VALUE labels for variable "raceth"
df <- add_val_labs(df,
  vars = "raceth", vals = c(1:7),
  labs = c("White", "Black", "Hispanic", "Asian", "AIAN", "Multi", "Other"),
  max.unique.vals = 50
)

head(df, 3)
check_labs_att(df, "val.labs.raceth") # "raceth" lab specifically TRUE

dfless <- sselect(df, id, raceth) # select only the vars id and raceth
head(dfless, 3) # selection worked
check_labs_att(dfless, "val.labs.raceth") # "raceth" value labels preserved
```

ssort *Safely Sort (Re-order) a Labeled Data Frame*

Description

ssort allows one to sort (after the fashion of base::order or dplyr::arrange) the rows of a data.frame based on column values.

Usage

```
ssort(data, vars, descending = FALSE, na.last = TRUE, fact.to.char = TRUE)
```

Arguments

data	the data.frame to be sorted.
vars	the variables to be sorted on, specified as a quoted character vector of variable names or an integer vector of column position indices.
descending	whether to sort the given variable of vars in descending or ascending order. Default is FALSE, which will be recycled to all vars arguments.
na.last	force NA values to appear last in a variable's sort order if TRUE (default).
fact.to.char	coerce all factor variables to character variables.

Details

This function accepts a data.frame, followed by a quoted vector of column names (or an integer vector of column position indices), followed by an indication of which are to be sorted ascending (default) or descending. If multiple columns are supplied to vars, sorting prioritizes the columns that appear earlier, with values of subsequent columns being sorted within distinct values of earlier columns. Note: ssort is fast enough on small data.frames and very slow on "larger" (>500K records) data.frames, particularly for more complex or demanding sort requests. Other R packages may provide faster sorting while preserving labelr attributes.

Value

a labelr label attribute-preserving data.frame consisting of the re-sorted data.frame.

Examples

```
# make toy demographic (gender, raceth, etc.) data set
set.seed(555)
df <- make_demo_data(n = 1000) # another labelr:: function

# let's add variable VALUE labels for variable "raceth"
df <- add_val_labs(df,
  vars = "raceth", vals = c(1:7),
  labs = c("White", "Black", "Hispanic", "Asian", "AIAN", "Multi", "Other"),
  max.unique.vals = 50
```

```
)

head(df, 3)
check_labs_att(df, "val.labs.raceth") # "raceth" lab specifically TRUE

dfsort1 <- ssort(df, c("raceth", "gender", "age"), descending = c(TRUE, FALSE, FALSE))

head(dfsort1, 20)

check_labs_att(dfsort1, "val.labs.raceth") # "raceth" lab specifically TRUE

dfsort2 <- ssort(df, c("age", "gender"))

head(dfsort2, 20)

check_labs_att(dfsort2, "val.labs.raceth") # "raceth" lab specifically TRUE

dfsort3 <- ssort(df, c("raceth"))

head(dfsort3, 10)

check_labs_att(dfsort3, "val.labs.raceth") # "raceth" lab specifically TRUE
```

`ssubset` *Safely Subset a Labeled Data Frame*

Description

`ssubset` allows one to simultaneously filter-select rows that satisfy conditions AND return only selected columns as a data.frame that preserves the labelr attributes attached to the inputted data.frame.

Usage

```
ssubset(data, condition, ...)
```

Arguments

`data`	the data.frame from which columns rows will be filtered (and, possibly, columns selected).
`condition`	row-filtering conditions along the lines of base::subset() and/or dplyr::select(). Note: Row-filtering conditions (to include condition==NULL) must be supplied.
`...`	Optionally supply one or more unquoted, comma-separated column names that identify columns to be retained (Note: If no columns are listed, all columns will be retained). Note: While row-filtering conditions may leverage standard operators (e.g., &, |, ==, !=), the column-selection portion of call may not incorporate special characters or symbols, such as quotes, parentheses, colons, minus signs, exclamation points, or other operators.

Details

Combining `sfilter` and `sselect` functionality (along the lines of base::subset(), this function accepts a data.frame, followed by conditional filtering instructions (required) and (optional) comma-separated unquoted column names to be selected (see examples), returning the selected rows and columns as a data.frame that preserves the labelr attribute information of the originally supplied data.frame. See `ssubset` for a variant that combines `sfilter` row-filtering and `sselect` column selection in a single function. See also `sdrop`, `sbrac`, `ssort`, `srename`, `slab` and `flab`.

Value

a labelr label attribute-preserving data.frame consisting of the selected rows that meet the filtering condition(s).

Examples

```
# make toy demographic (gender, raceth, etc.) data set
set.seed(555)
df <- make_demo_data(n = 1000) # another labelr:: function
# let's add variable VALUE labels for variable "raceth"
df <- add_val_labs(df,
  vars = "raceth", vals = c(1:7),
  labs = c("White", "Black", "Hispanic", "Asian", "AIAN", "Multi", "Other"),
  max.unique.vals = 50
)

head(df, 3)
check_labs_att(df, "val.labs.raceth") # "raceth" lab specifically TRUE

dfsub1 <- ssubset(df, raceth == 2 & age > 70, id, raceth, gender)
head(dfsub1, 3)
check_labs_att(dfsub1, "val.labs.raceth") # "raceth" lab specifically TRUE

dfsub2 <- ssubset(df, raceth %in% c(2:4), age, raceth)
head(dfsub2, 3)
check_labs_att(dfsub2, "val.labs.raceth") # "raceth" lab specifically TRUE
# even if NULL, must supply explicit condition argument
head(ssubset(df, condition = NULL, age, raceth), 3) # better to just use sselect()
head(ssubset(df, age == 60), 3) # skip column selection (will return all cols)
```

---

strip_labs *Strip All labelr Meta-data from a Data Frame*

Description

`strip_labs` removes all labelr meta-data attributes ("name.labs", "val.labs", "frame.lab", and "factor.") from a data.frame, because you no longer want/need it for some reason or another.

Usage

```
strip_labs(data)
```

Arguments

data	a data.frame object.

Details

Some labelr functions automatically use strip_labs, but you should only use it if you no longer want or need a given data.frame's labelr meta-data (i.e., labels). If you have saved your labelr attributes (using get_all_lab_atts), you can restore them to the data.frame using add_lab_atts).

Value

a data.frame object with label attribute meta-data stripped from it.

Examples

```
# make toy demographic (gender, raceth, etc.) data set
set.seed(555)
df <- make_demo_data(n = 1000) # another labelr:: function
# let's add variable VALUE labels for variable "raceth"
df <- add_val_labs(df,
  vars = "raceth", vals = c(1:7),
  labs = c("White", "Black", "Hispanic", "Asian", "AIAN", "Multi", "Other"),
  max.unique.vals = 50
)

get_val_labs(df, "raceth") # it's here
df <- strip_labs(df) # this removes labs from df
get_val_labs(df, "raceth") # it's gone
check_any_lab_atts(df) # FALSE (means "no labs here")
```

---

tabl *Construct Value Label-Friendly Frequency Tables*

Description

tabl calculates raw or weighted frequency counts (or proportions) over arbitrary categorical values (including integer values), which may be expressed in terms of raw variable values or labelr label values.

Usage

```
tabl(
  data,
  vars = NULL,
  labs.on = TRUE,
  qtiles = 4,
  prop.digits = NULL,
  wt = NULL,
```

```
  div.by = NULL,
  max.unique.vals = 10,
  sort.freq = TRUE,
  zero.rm = FALSE,
  irreg.rm = FALSE,
  wide.col = NULL
)
```

Arguments

data	a data.frame.
vars	a quoted character vector of variable names of variables you wish to include in defining category groups to tabulate over in the table. If NULL tabl will attempt to construct a table over all combinations of all non-decimal-having variables in the data.frame that do not exceed the max.unique.vals threshold. Additionally, note the effects of the qtiles argument.
labs.on	if TRUE (the default), then value labels – rather than the raw variable values – will be displayed in the returned table for any value-labeled variables. Variables need not be value-labeled: This command (with this option set to TRUE or FALSE) will work even when no variables are value-labeled.
qtiles	if not NULL, must be a 1L integer between 2 and 100 indicating the number of quantile categories to employ in temporarily (for purposes of tabulation) auto-value-labeling numeric columns that exceed the max.unique.vals threshold. If NULL, no such auto-value-labeling will take place. Note: When labs.on = TRUE, any pre-existing variable value labels will be used in favor of the quantile value labels generated by this argument. By default, qtiles = 4, and the automatically generated quantile category levels will be labeled as "q025" (i.e., first quartile), "q050", "q075", and "q100".
prop.digits	if non-NULL, cell percentages (proportions) will be returned instead of frequency counts, and these will be rounded to the digit specified (e.g., prop.digits = 3 means a value of 0.157 would be returned for a cell that accounted for 8 observations if the total number of observations were 51). If NULL (the default), frequency counts will be returned.
wt	an optional vector that includes cell counts or some other idiosyncratic "importance" weight. If NULL, no weighting will be employed.
div.by	Divide the returned counts by a constant for scaling purposes. This may be a number (e.g., div.by = 10 to divide by 10) or a character that follows the convention "number followed by 'K', 'M', or 'B'", where, e.g., "10K" is translated as 10000, "1B" is translated as 1000000000, etc.
max.unique.vals	Integer to specify the maximum number of unique values of a variable that may be observed for that variable to be included in tabulations. Note that labelr sets a hard ceiling of 5000 on the total number of unique value labels that any variable is permitted to have under any circumstance, as labelr is primarily intended for interactive use with moderately-sized data.frames. See the qtiles argument for an approach to incorporating many-valued numeric variables that exceed the max.unique.vals threshold.

`sort.freq` By default, returned table rows are sorted in descending order of cell frequency (most frequent categories/combinations first). If set to FALSE, table rows will be sorted by the the distinct values of the vars (in the order vars are specified).

`zero.rm` If TRUE, zero-frequency vars categories/combinations (i.e., those not observed in the data.frame) will be filtered from the table. For tables that would produce more than 10000 rows, this is done automatically.

`irreg.rm` If TRUE, tabulations exclude cases where any applicable variable (see vars argument) features any of the following "irregular" values: NA, NaN, Inf, -Inf, or any non-case-sensitive variation on "NA", "NAN", "INF", or "-INF." If FALSE, all "irregular" values (as just defined) are assigned to a "catch-all" category of NA that is featured in the returned table (if/where present).

`wide.col` If non-NULL, this is the quoted name of a single column / var of supplied data.frame whose distinct values (category levels) you wish to be columns of the returned table. For example, if you are interested in a cross-tab of "edu" (highest level of education) and "race" (a race/ethnicity variable), you could supply vars= c("edu") and wide.col = "race", and the different racial-ethnic group categories would appear as distinct columns, with "edu" category levels appearing as distinct rows, and cell values representing the cross-tabbed cell "edu" level frequencies for the respective "race" groups (see examples). You may supply one wide.col at most.

Details

This function creates a labelr-friendly data.frame representation of multi-variable tabular data, where either value labels or values can be displayed (use of value labels is the default), and where various convenience options are provided, such as using frequency weights, using proportions instead of counts, rounding those percentages, automatically expressing many-valued, non-value-labeled numerical variables in terms of quantile category groups, or pivoting / casting one of the categorical variables' levels (labels) to serve as columns in a cross-tab-like table.

Value

a data.frame.

Examples

```
# assign mtcars to new data.frame df
df <- mtcars

# add na values to make things interesting
df[1, 1:11] <- NA
rownames(df)[1] <- "Missing Car"

# add value labels
df <- add_val_labs(
  data = df,
  vars = "am",
  vals = c(0, 1),
  labs = c("automatic", "manual")
```

```
)

df <- add_val_labs(
  data = df,
  vars = "carb",
  vals = c(1, 2, 3, 4, 6, 8),
  labs = c(
    "1-carb", "2-carbs",
    "3-carbs", "4-carbs",
    "6-carbs", "8-carbs"
  )
)

# var arg can be unquoted if using add_val1()
# note that this is not add_val_labs(); add_val1() has "var" arg instead of "vars
df <- add_val1(
  data = df,
  var = cyl, # note, "var," not "vars" arg
  vals = c(4, 6, 8),
  labs = c(
    "four-cyl",
    "six-cyl",
    "eight-cyl"
  )
)

df <- add_val_labs(
  data = df,
  vars = "gear",
  vals = 3:5,
  labs = c(
    "3-speed",
    "4-speed",
    "5-speed"
  )
)


# lookup mapping
get_val_labs(df)

# introduce other "irregular" values
df$am[1] <- NA

df[2, "am"] <- NaN
df[3, "am"] <- -Inf
df[5, "cyl"] <- "NAN"

# take a look
head(df)

# demonstrate tabl() frequency tabulation function
```

```
# this is the "first call" that will be referenced repeatedly below
# labels on, sort by variable values, suppress/exclude NA/irregular values
# ...return counts
tabl(df,
  vars = c("cyl", "am"),
  labs.on = TRUE, # use variable value labels
  sort.freq = FALSE, # sort by vars values (not frequencies)
  irreg.rm = TRUE, # NAs and the like are suppressed
  prop.digits = NULL
) # return counts, not proportions

# same as "first call", except now value labels are off
tabl(df,
  vars = c("cyl", "am"),
  labs.on = FALSE, # use variable values
  sort.freq = FALSE, # sort by vars values (not frequencies)
  irreg.rm = TRUE, # NAs and the like are suppressed
  prop.digits = NULL
) # return counts, not proportions

# same as "first call," except now proportions instead of counts
tabl(df,
  vars = c("cyl", "am"),
  labs.on = TRUE, # use variable value labels
  sort.freq = FALSE, # sort by vars values (not frequencies)
  irreg.rm = TRUE, # NAs and the like are suppressed
  prop.digits = 3
) # return proportions, rounded to 3rd decimal

# same as "first call," except now sort by frequency counts
tabl(df,
  vars = c("cyl", "am"),
  labs.on = TRUE, # use variable value labels
  sort.freq = TRUE, # sort in order of descending frequency
  irreg.rm = TRUE, # NAs and the like are suppressed
  prop.digits = NULL
) # return proportions, rounded to 3rd decimal

# same as "first call," except now use weights
set.seed(2944) # for reproducibility
df$freqwt <- sample(10:50, nrow(df), replace = TRUE) # create (fake) freq wts
tabl(df,
  vars = c("cyl", "am"),
  wt = "freqwt", # use frequency weights
  labs.on = TRUE, # use variable value labels
  sort.freq = FALSE, # sort by vars values (not frequencies)
  irreg.rm = FALSE, # NAs and the like are included/shown
  prop.digits = NULL
) # return counts, not proportions

df$freqwt <- NULL # we don't need this anymore

# now, with extremely large weights to illustrate div.by
```

```
set.seed(428441) # for reproducibility
df$freqwt <- sample(1000000:10000000, nrow(df), replace = TRUE) # large freq wts
tabl(df,
  vars = c("cyl", "am"),
  wt = "freqwt", # use frequency weights
  labs.on = TRUE, # use variable value labels
  sort.freq = FALSE, # sort by vars values (not frequencies)
  irreg.rm = FALSE, # NAs and the like are included/shown
  prop.digits = NULL
) # return counts, not proportions

# show div by - Millions
tabl(df,
  vars = c("cyl", "am"),
  wt = "freqwt", # use frequency weights
  labs.on = TRUE, # use variable value labels
  sort.freq = FALSE, # sort by vars values (not frequencies)
  irreg.rm = FALSE, # NAs and the like are included/shown
  prop.digits = NULL, # return counts, not proportions
  div.by = "1M"
) # one million

# show div by - Tens of millions
tabl(df,
  vars = c("cyl", "am"),
  wt = "freqwt", # use frequency weights
  labs.on = TRUE, # use variable value labels
  sort.freq = FALSE, # sort by vars values (not frequencies)
  irreg.rm = FALSE, # NAs and the like are included/shown
  prop.digits = NULL, # return counts, not proportions
  div.by = "10M"
) # ten million

# show div by - 10000
tabl(df,
  vars = c("cyl", "am"),
  wt = "freqwt", # use frequency weights
  labs.on = TRUE, # use variable value labels
  sort.freq = FALSE, # sort by vars values (not frequencies)
  irreg.rm = FALSE, # NAs and the like are included/shown
  prop.digits = NULL, # return counts, not proportions
  div.by = 10000
) # ten thousand; could've used div.by = "10K"

# show div by - 10000, but different syntax
tabl(df,
  vars = c("cyl", "am"),
  wt = "freqwt", # use frequency weights
  labs.on = TRUE, # use variable value labels
  sort.freq = FALSE, # sort by vars values (not frequencies)
  irreg.rm = FALSE, # NAs and the like are included/shown
  prop.digits = NULL, # return counts, not proportions
  div.by = "10K"
```

```
) # ten thousand; could've used div.by = 10000

df$freqwt <- NULL # we don't need this anymore

# turn labels off, to make this more compact
# do not show zero values (zero.rm)
# do not show NA values (irreg.rm)
# many-valued numeric variables will be converted to quantile categories by
# ...qtiles argument
tabl(df,
  vars = c("am", "gear", "carb", "mpg"),
  qtiles = 4, # many-valued numerics converted to quantile
  labs.on = FALSE, # use values, not variable value labels
  sort.freq = FALSE, # sort by vars values (not frequencies)
  irreg.rm = TRUE, # NAs and the like are suppressed
  zero.rm = TRUE, # variable combinations that never occur are suppressed
  prop.digits = NULL, # return counts, not proportions
  max.unique.vals = 10
) # drop from table any var with >10 distinct values

# same as above, but include NA/irregular category values,
# zero.rm is TRUE; include unobserved (zero-count) category combinations
tabl(df,
  vars = c("am", "gear", "carb", "mpg"),
  qtiles = 4,
  labs.on = FALSE, # use values, not variable value labels
  sort.freq = TRUE, # sort by frequency
  irreg.rm = FALSE, # preserve/include NAs and irregular values
  zero.rm = FALSE, # include non-observed combinations
  prop.digits = NULL, # return counts, not proportions
  max.unique.vals = 10
) # drop from table any var with >10 distinct values

# show cross-tab view with wide.col arg
tabl(df,
  vars = c("cyl", "am"),
  labs.on = TRUE, # use variable value labels
  sort.freq = TRUE, # sort by vars values (not frequencies)
  irreg.rm = TRUE, # NAs and the like are suppressed
  prop.digits = NULL, # return counts, not proportions
  wide.col = "am"
) # use "am" as a column variable in a cross-tab view

tabl(df,
  vars = c("cyl", "am"),
  labs.on = TRUE, # use variable value labels
  sort.freq = TRUE, # sort by vars values (not frequencies)
  irreg.rm = TRUE, # NAs and the like are suppressed
  prop.digits = NULL, # return counts, not proportions
  wide.col = "cyl"
) # use "cyl" as a column variable in a cross-tab view

# verify select counts using base::subset()
```

```
nrow(subset(df, am == 0 & cyl == 4))
nrow(subset(df, am == 0 & cyl == 8))
nrow(subset(df, am == 1 & cyl == 8))
nrow(subset(df, am == 0 & cyl == 6))
nrow(subset(df, am == 1 & cyl == 6))

# will work on an un-labeled data.frame
tabl(mtcars, vars = c("am", "gear", "carb", "mpg"))
```

`taill` *Return Last Rows of a Data Frame with Value Labels Visible*

Description

`taill` accepts a labelr value-labeled data.frame and returns the last n value-labeled rows of that data.frame

Usage

```
taill(data, n = 6L)
```

Arguments

`data`	a data.frame.
`n`	the number of consecutive rows at the end / bottom of the data.frame to return.

Details

Whereas `utils::tail` returns the last n rows of a data.frame, `taill` does the same thing, substituting value labels for values wherever the former exist. See also `headl` and `somel`.

Value

a data.frame.

Examples

```
# make toy demographic (gender, raceth, etc.) data set
set.seed(555)
df <- make_demo_data(n = 1000) # another labelr:: function
# let's add variable VALUE labels for variable "raceth"
df <- add_val_labs(df,
  vars = "raceth", vals = c(1:7),
  labs = c("White", "Black", "Hispanic", "Asian", "AIAN", "Multi", "Other"),
  max.unique.vals = 50
)

# let's add variable VALUE labels for variable "gender"
# note that, if we are labeling a single variable, we can use add_val1()
```

```
# distinction between add_val1() and add_val_labs() will become more meaningful
# when we get to our Likert example
df <- add_val1(
  data = df, gender, vals = c(0, 1, 2, 3, 4),
  labs = c("M", "F", "TR", "NB", "Diff-Term"), max.unique.vals = 50
)

tail(df) # utils::tail
taill(df) # same, but with value labels in place of values
```

`transfer_labs` *Transfer Labels from One Variable (Column) Name to Another*

Description

Note: `transfer_labs` searches a data.frame's name.labs and val.labs attributes and transfers the labels associated with one variable name to another, so that the first variable no longer has name or value labels associated with it, and so that whatever name or value labels previously were associated with it are now associated with the second variable.

Usage

```
transfer_labs(data, from, to)
```

Arguments

`data`	a data.frame.
`from`	the unquoted variable name from which labels will be transferred. Note, even if the variable itself has been dropped from the data.frame (to include being renamed), its label attribute meta-data may still be present and available for use by this function (use `get_all_lab_atts()` to see).
`to`	the unquoted name of the variable to which the labels will be transferred.

Details

Certain non-labelr data management functions will preserve the labelr labels that are attached to the passed data.frame, but they will not update those labels to reflect any changes the function makes to the variable(s). For example, if one were to use dplyr::rename to change the name of a value-labeled variable from old name "x1" to new name "satisfaction", the labelr attributes associated with "x1" would not be transferred to label "satisfaction." `transfer_labs` allows one to transfer those labels, dis- associating them with the old name (here, "x1") and associating them with new name (here, "satisfaction").

Value

A data.frame.

Examples

```
# make toy demographic (gender, raceth, etc.) data set
set.seed(555)
df <- make_demo_data(n = 1000) # another labelr:: function

# let's add variable VALUE labels for variable "raceth"
df <- add_val_labs(df,
  vars = "raceth", vals = c(1:7),
  labs = c("White", "Black", "Hispanic", "Asian", "AIAN", "Multi", "Other"),
  max.unique.vals = 50
)

df <- add_val1(
  data = df, gender, vals = c(0, 1, 2, 3, 4),
  labs = c("M", "F", "TR", "NB", "Diff-Term"), max.unique.vals = 50
)

# let's add variable NAME labels
df <- add_name_labs(df, name.labs = c(
  "age" = "Age in years",
  "raceth" = "racial-ethnic group category",
  "gender" = "gender identity"
))
head(df, 4)
get_name_labs(df)
get_val_labs(df)

df <- dplyr::rename(df, race = raceth) # new name is on left of = sign
df <- dplyr::rename(df, gend = gender) # new name is on left of = sign

head(df, 4)
get_name_labs(df)
get_val_labs(df)

df <- transfer_labs(df, from = raceth, to = race) # labs info transferred from raceth
df <- transfer_labs(df, from = gender, to = gend) # labs info transferred to gend
df <- transfer_labs(df, from = gend, to = nothere) # var nothere does not exist!

head(df, 4)
get_name_labs(df)
get_val_labs(df)
```

---

use_name_labs *Swap Name Labels for Variable Names*

Description

Replace data.frame variable names with their corresponding name labels (previously assigned using add_name_labs).

Usage

```
use_name_labs(data, vars = NULL)

unl(data, vars = NULL)
```

Arguments

data	the data.frame whose name labels you wish to "use" (aka swap, turn on, activate, etc.).
vars	the names of the columns (variables) to which name labels will be applied. If NULL, all available name labels will be swapped in for the corresponding variable (column) names.

Details

Note: unl is a compact alias for use_name_labs: they do the same thing, and the former is easier to type

use_name_labs works with add_name_labs, get_name_labs, use_var_names, and drop_name_labs, to facilitate the creation, accessing, substitution (swap out, swap back in), and destruction of variable name labels for variable names. Each variable (column) of a data.frame can receive one and only one "name label," which typically is a noun phrase that expounds the meaning of contents of the variable's name (e.g., "Weight in ounces at birth" might be a name label for a column called "wgt"). add_name_labs associates these labels with variables in a data.frame, use_name_labs applies or "turns on" those name labels, i.e., swaps out variable names for corresponding labels, and you can assign the name-label-swapped data.frame to an object, or you may use it strictly for display purposes (e.g., head(use_name_labs(df), 5)). Because they are intended to be more descriptive than column names, they tend to be more verbose – possibly so verbose as to undermine their value or convenience for anything other than an on-demand "What is this variable again?" cheat sheet via get_name_labs(). That said, this may have some uses (see examples).

Value

A data.frame, with (all or the select) name labels swapped in for the variable names.

Examples

```
# variable names and their labels
names_labs_vec <- c(
  "mpg" = "Miles/(US) gallon",
  "cyl" = "Number of cylinders",
  "disp" = "Displacement (cu.in.)",
  "hp" = "Gross horsepower",
  "drat" = "Rear axle ratio",
  "wt" = "Weight (1000 lbs)",
  "qsec" = "1/4 mile time",
  "vs" = "Engine (0 = V-shaped, 1 = straight)",
  "am" = "Transmission (0 = automatic, 1 = manual)",
  "gear" = "Number of forward gears",
  "carb" = "Number of carburetors"
```

```
)

# add the above name labeling scheme
mt2 <- add_name_labs(mtcars, name.labs = names_labs_vec)

# use the name labeling scheme (i.e., swap out column/variable names for
# ...their name labels)
mt2 <- use_name_labs(mt2)

# compare these two - concision vs. informativeness
as.data.frame(sapply(mtcars, mean))
as.data.frame(sapply(mt2, mean))

# compare the plot labeling we get with mtcars
with(mtcars, hist(mpg))

get_name_labs(mt2) # get the lab of interest, and paste it into `` below
with(mt2, hist(`Miles/(US) gallon`))

# regression - this is easier to type
lm(mpg ~ cyl, data = mtcars)

# regression with name labs - more painful to type/copy-paste, but may be
# ...the more informative labels are worth it (your mileage/mpg may vary)
# let's see the name labels, then copy paste mpg and cyl labs from console to
# ...where we need them in the lm() call
get_name_labs(mt2) # copy from this call's console output
lm(`Miles/(US) gallon` ~ `Number of cylinders`, data = mt2) # paste into `` here

# same results, more informative labels, more steps/hand-jamming pain
# can also turn them on (semi) permanently
# ...then you can use mt2$ syntax in RStudio, and RStudio will autocomplete,
# then you can backspace delete the "mt2$"
# if you like
mt2 <- use_name_labs(mt2)
lm(`Miles/(US) gallon` ~ `Number of cylinders`, data = mt2)
lm(mpg ~ cyl, data = use_var_names(mt2))

# let's turn them back off
mt2 <- use_var_names(mt2) # use_var_names() as "undo" of use_name_labs()

# back to our previous variable names
head(mt2)
# even with name labels "off," mt2 retains labelr attribute meta-data
# ...which we can strip away using strip_labs()
identical(strip_labs(mt2), mtcars) # and we're back
```

---

use_val_lab1 *Replace a Single Data Frame Column's Values with Its Value Labels*

Description

For a single value-labeled column of a data.frame, replace all of its values with the corresponding value labels and return the modified data.frame.

Usage

```
use_val_lab1(data, var)

uvl1(data, var)
```

Arguments

data	the data.frame.
var	the unquoted name of the column (variable) whose values you wish to replace with the corresponding value labels.

Details

Note 1: use_val_lab1 is a variant of use_val_labs that allows you to specify only one variable at a time but that allows you to pass its name without quoting it (compare use_val_lab1(mtcars, am) to use_val_labs(mtcars, "am")).

Note 2: uvl1 is a compact alias for use_val_lab1: they do the same thing, and the former is easier to type.

Note 3: This command is intended exclusively for interactive use. In particular, the var argument must be the literal name of a single variable (column) found in the supplied data.frame and may NOT be, e.g., the name of a character vector that contains the variable (column name) of interest. If you wish to supply a character vector with the names of variables (columns) of interest, use use_val_labs().

use_val_lab1 replaces a single, value-labeled data.frame column with a "value labels-on" version of that column. Here, "labels-on" means that the column's original values are replaced with the corresponding value labels. Note that the modified column is a simple, self-contained character variable that cannot itself be converted or reverted back to the original ("labels-off") values of its parent/source column.

Value

A data.frame consisting of the originally supplied data.frame, with the var argument variable's values replaced with its value labels.

Examples

```
# swap in "am" value labels for values in mtcars
df <- mtcars # copy of mtcars

# now, add value labels
df <- add_val1(
  data = df,
  var = am,
```

```
  vals = c(0, 1),
  labs = c("automatic", "manual")
)

# switch out "am" values for value labels, assign to df_plus
df_plus <- use_val_lab1(df, am)
head(df_plus[c("am")])
```

`use_val_labs` *Swap Variable Value Labels for Variable Values*

Description

Replace the actual values of data.frame variables with the corresponding value labels (previous assigned using `add_val_labs` or a related function).

Usage

```
use_val_labs(data, vars = NULL)

uvl(data, vars = NULL)
```

Arguments

`data`	a data.frame.
`vars`	the names of the columns (variables) for which value labels will will replace original values in the returned data.frame.

Details

Note: `uvl` is a compact alias for `use_val_labs`: they do the same thing, and the former is easier to type.

`use_val_labs` takes a variable value-labeled data.frame and substitutes each (labeled) variable's value labels for its values, returning a data.frame whose dimensions, names, and members are the same as the inputted data.frame. This may be useful if one wishes to view data.frame information using the (potentially) more intuitively meaningful value labels (e.g., gender=1 values displayed as "Male" instead of 1).

Warning: `use_val_labs` will replace existing variable values with value labels and cannot be undone. If you wish to preserve variable values, be sure to assign the result of `use_val_labs` to a new object. For other ways to leverage value labels for common data management or inspection tasks, while preserving raw data values in returned object, see `add_lab_cols`, `add_lab_dummies`, `flab`, `slab`, `tabl`, `headl`, `taill`, and `somel`.

Value

A data.frame, with (all or the select) variable value labels substituted for original variable values and any affected variables coerced to character if they were not already.

Examples

```
# Example #1 - mtcars example, one variable at a time
# one variable at a time, mtcars
df <- mtcars
# now, add value labels
df <- add_val_labs(
  data = df,
  vars = "am",
  vals = c(0, 1),
  labs = c("automatic", "manual")
)

df <- add_val_labs(
  data = df,
  vars = "carb",
  vals = c(1, 2, 3, 4, 6, 8),
  labs = c(
    "1-carb", "2-carbs",
    "3-carbs", "4-carbs",
    "6-carbs", "8-carbs"
  )
)

# var arg can be unquoted if using add_val1()
# note that this is not add_val_labs(); add_val1() has "var" (not "vars) arg
df <- add_val1(
  data = df,
  var = cyl, # note, "var," not "vars" arg
  vals = c(4, 6, 8),
  labs = c(
    "four-cyl",
    "six-cyl",
    "eight-cyl"
  )
)

df <- add_val_labs(
  data = df,
  vars = "gear",
  vals = c(3, 4),
  labs = c(
    "3-speed",
    "4-speed"
  )
)

# Oops, we forgot 5-speeds; let's finish the job.
df <- add_val_labs(
  data = df,
  vars = "gear",
  vals = 5,
  labs = "5-speed"
```

```
)

head(use_val_labs(df), 3) # they're there

# Example #2 - (Fake) Likert Data
# add val labs to multiple variables at once
# make a "Likert"-type fake data set to demo
# note, by default, add_val_labs() "vars" arg will do partial matching
# in this case, we catch all vars with "x" in their name
set.seed(272)
dflik <- make_likert_data(scale = 1:7)
vals2label <- 1:7
labs2use <- c(
  "VSD",
  "SD",
  "D",
  "N",
  "A",
  "SA",
  "VSA"
)

dflik <- add_val_labs(
  data = dflik, vars = c("x", "y3"), # note the vars args
  vals = vals2label,
  labs = labs2use,
  partial = TRUE
)

# note, all "x" vars get the labs, as does "y3"
# see vars = args above
lik1 <- use_val_labs(dflik)
head(lik1)
# keep a copy
dflik_conv <- use_val_labs(dflik)
head(dflik_conv, 3)
```

`use_var_names` *Swap (back) Original Variable Names for Name Labels*

Description

"Undo" or reverse `use_name_labs` operation, restoring the data.frame's original variable names (assuming they were previously swapped out for labels using `use_name_labs`).

Usage

```
use_var_names(data, vars = NULL)

uvn(data, vars = NULL)
```

Arguments

data	a data.frame.
vars	the names of the columns (variables) to which the action will be applied. If NULL, all current variable names (presumably, the labels you turned on using use_name_labs will be swapped out for their original variable names available names.

Details

Note: uvn is a compact alias for use_var_names: they do the same thing, and the former is easier to type

use_var_names works with add_name_labs, get_name_labs, use_name_labs, and drop_name_labs, to facilitate the creation, accessing, substitution (swap out, swap back in), and destruction of variable name labels for variable names. Each variable (column) of a data.frame can receive one and only one "name label," which typically is a noun phrase that expounds the meaning of contents of the variable's name (e.g., "Weight in ounces at birth" might be a name label for a column called "wgt"). add_name_labs associates these labels with variables in a data.frame, use_var_names "turns off" the name labels are currently being used as variable names as a result of calling use_name_labs – that is, use_var_names "undoes" or reverses use_name_labs – assuming none of your intervening calls have altered or dropped your data.frame's name label meta-data attributes.

Value

A data.frame.

Examples

```
# variable names and their labels
names_labs_vec <- c(
  "mpg" = "Miles/(US) gallon",
  "cyl" = "Number of cylinders",
  "disp" = "Displacement (cu.in.)",
  "hp" = "Gross horsepower",
  "drat" = "Rear axle ratio",
  "wt" = "Weight (1000 lbs)",
  "qsec" = "1/4 mile time",
  "vs" = "Engine (0 = V-shaped, 1 = straight)",
  "am" = "Transmission (0 = automatic, 1 = manual)",
  "gear" = "Number of forward gears",
  "carb" = "Number of carburetors"
)

# add the above name labeling scheme
mt2 <- add_name_labs(mtcars, name.labs = names_labs_vec)

# use the name labeling scheme (i.e., swap out column/variable names for
# ...their name labels)
mt2 <- use_name_labs(mt2)

# compare these two - concision vs. informativeness
```

```
as.data.frame(sapply(mtcars, mean))
as.data.frame(sapply(mt2, mean))

# compare the plot labeling we get with mtcars
with(mtcars, hist(mpg))

get_name_labs(mt2) # get the lab of interest, and paste it into `` below
with(mt2, hist(`Miles/(US) gallon`))

# regression - this is easier to type
lm(mpg ~ cyl, data = mtcars)

# regression with name labs - more painful to type/copy-paste, but maybe
# ...the more informative labels are worth it (your mileage may vary)
# let's see the name labels, then copy paste mpg and cyl labs from console to
# ...where we need them in the lm() call
get_name_labs(mt2) # copy from this call's console output
lm(`Miles/(US) gallon` ~ `Number of cylinders`, data = mt2) # paste into `` here

# same results, more informative labels, more steps/hand-jamming pain
# can also turn them on (semi) permanently
# ...then you can use mt2$ syntax in Rstudio, and Rstudio will autocomplete,
# then you can backspace delete the "mt2$"
# if you like
mt2 <- use_name_labs(mt2)
lm(`Miles/(US) gallon` ~ `Number of cylinders`, data = mt2)
lm(mpg ~ cyl, data = use_var_names(mt2))

# let's turn them back off
mt2 <- use_var_names(mt2) # use_var_names() as "undo" of use_name_labs()

# back to our previous variable names
head(mt2)
# even with name labels "off," mt2 retains labelr attribute meta-data
# ...which we can strip away using strip_labs()
identical(strip_labs(mt2), mtcars) # and we're back
```

---

v *Specify Column Names without Quoting Them*

Description

Alternative to the base `c()` combine operator that allows one to select columns by passing unquoted comma-separated column names instead of quoted, comma-separated column names.

Usage

```
v(...)
```

Arguments

...	a vector of unquoted, comma-separated column names.

Details

Does not support or combine with other subsetting operators, such as negative indexing or colon: names must be full, individual column names, separated by commas (see examples).

Value

A character vector of quoted, comma-separated, column names.

Examples

```
mt2a <- mtcars[c("am", "cyl", "mpg")]
mt2b <- mtcars[v(am, cyl, mpg)]
identical(mt2a, mt2b) # TRUE

# silly demo
mtlabs <- mtcars
mtlabs <- add_val_labs(
  data = mtlabs,
  vars = "am",
  vals = c(0, 1),
  labs = v(a, m) # equivalent to c("a", "m")
)

get_val_labs(mtlabs)
```

---

val_labs_vec *Replace a Variable's Values with Its Value Labels and Return as a Vector*

Description

Select a single, value-labeled data.frame column (variable), replace each of its values with the corresponding value labels, and return the result as a vector.

Usage

```
val_labs_vec(data, var)

vlv(data, var)
```

Arguments

data	a data.frame.
var	the unquoted name of the column (variable) whose values will be converted to the associated value labels in the returned vector.

Details

Note 1: `vlv` is a compact alias for `val_labs_vec`: they do the same thing, and the former is easier to type.

Note 2: This command is intended exclusively for interactive use. In particular, the var argument must be the literal name of a single variable (column) found in the supplied data.frame and may NOT be, e.g., the name of a character vector that contains the variable (column name) of interest.

`val_labs_vec` works with other labelr functions to facilitate creation, modification, accessing, use, and destruction of variable-specific value labels. This functionality is equivalent to calling `use_val_labs` with a single variable passed to the vars argument, except that the latter returns the entire data.frame with that variable modified, while `val_labs_vec` returns only that single modified variable itself (as a vector)

Value

A vector containing the original data.frame variable (var), after its values have been converted to their corresponding value labels.

Examples

```
df <- mtcars
# add value labels
df <- add_val_labs(
  data = df,
  var = "am",
  vals = c(0, 1),
  labs = c("automatic", "manual")
)

am_labs <- val_labs_vec(df, am)

length(df$am)

class(df$am)

df$am

length(am_labs)

class(am_labs)

am_labs
```

`with_both_labs` *Overlay Variable Name and Value Labels Onto Arbitrary R Function Calls*

Description

`with_both_labs` instructs R function calls to swap in variable name labels for column names AND variable value labels for variable values in the objects they return or side effects they produce.

Note: `wbl` is a compact alias for `with_both_labs`: they do the same thing, and the former is easier to type

Usage

```
with_both_labs(data, ...)

wbl(data, ...)
```

Arguments

`data`	a data.frame with variable name labels and variable value labels.
`...`	an additional expression passed to dots (quotes and dollar signs are not needed or permitted).

Details

`with_both_labs` (see also alias `wbl`) is intended for interactive use. With it, you pass a name-labeled data.frame followed, followed by a comma, followed by an unquoted R expression (function call) to be evaluated within that data.frame, and both name and value labels will be substituted for their corresponding, respective column names and variable value in any returned object or side effects. Your function call (expression) should refer to columns of the data.frame passed via your data argument, NOT their name labels, as the intent is to allow you to pass functions in terms of the (typically much more concise and familiar) column names while having the results displayed / presented in terms of the more informative (but more verbose and typically non-standard) name labels. See examples.

Caution 1: Typically, `with_name_labs` will be more appropriate than `with_both_labs`, since conversion of variables' values to their corresponding labels frequently entails conversion from numeric to character.

Caution 2: `with_both_labs` is a rudimentary function that leverages basic regular expressions and `eval(parse(text=))` to substitute name labels for variable names behind the scenes. It appears to be robust to a range of the most common commands and operators (e.g., formula or modeling operators, such as ~ , * , +, :, =, and |). However, it is intended strictly as a convenience for relatively simple, interactive, single-line-expression, data exploration, description, or simple model-fitting use cases. It is expressly NOT intended for: (1) multi-step workflows or pipes, (2) expressions that require or make reference to objects existing outside the supplied data.frame, or (3) data management operations aimed at modifying the supplied data.frame. Again, see the examples for the types of expressions/use cases envisioned.

Value

the value of the evaluated `expr`, with name and value labels substituted for variable (column) names and values, respectively.

Examples

```
# assign mtcars to new data.frame mt2
mt2 <- mtcars

# add name labs
mt2 <- add_name_labs(mt2,
  name.labs = c(
    "mpg" = "Miles/(US) gallon",
    "cyl" = "Number of cylinders",
    "disp" = "Displacement (cu.in.)",
    "hp" = "Gross horsepower",
    "drat" = "Rear axle ratio",
    "wt" = "Weight (1000 lbs)",
    "qsec" = "1/4 mile time",
    "vs" = "Engine (0 = V-shaped, 1 = straight)",
    "am" = "Transmission (0 = automatic, 1 = manual)",
    "gear" = "Number of forward gears",
    "carb" = "Number of carburetors"
  )
)


# add many-to-1 value labels
mt2 <- add_m1_lab(
  data = mt2,
  vars = "gear",
  vals = 4:5,
  lab = "4+"
)

# add many-to-1 value labels
mt2 <- add_val_labs(
  data = mt2,
  vars = "am",
  vals = c(0, 1),
  lab = c("auto", "man")
)
with_both_labs(mt2, t.test(mpg ~ am))
with_both_labs(mt2, lm(mpg ~ am))
with_both_labs(mt2, xtabs(~gear))
xtabs(~ mt2$gear)
```

`with_name_labs` *Overlay Variable Name Labels Onto Arbitrary R Function Calls*

Description

`with_name_labs` instructs R function calls to swap in variable name labels for column names in the objects they return or side effects they produce.

Usage

```
with_name_labs(data, ...)

wnl(data, ...)
```

Arguments

data	a data.frame with name labels.
...	an additional expression passed to dots (quotes and dollar signs are not needed or permitted).

Details

Note: wnl is a compact alias for with_name_labs: they do the same thing, and the former is easier to type

with_name_labs is intended for interactive use. With it, you pass a name-labeled data.frame, followed by a comma, followed by an unquoted R expression (function call) to be evaluated within that data.frame, and name labels will be substituted for column names in any returned object or side effects. Your function call (expression) should refer to columns of the data.frame, NOT their name labels, as the intent is to allow you to pass functions in terms of the (typically much more concise and familiar) column names while having the results displayed / presented in terms of the more informative (but more verbose and typically non-standard) name labels. See examples.

Caution: with_name_labs is a rudimentary function that leverages basic, fairly literal (and potentially brittle!) regular expressions and eval(parse(text=)) to substitute name labels for variable names behind the scenes. It appears to be robust to a range of common commands and operators (e.g., ~ , * , +, :, ::, =, and |). However, it is intended strictly as a convenience for relatively simple, interactive, single-line-expression use cases, involving data exploration, description, or simple model-fitting. It is expressly NOT intended for: (1) multi-step workflows or pipes, (2) expressions that require or make reference to objects existing outside the supplied data.frame, or (3) data management operations aimed at modifying (e.g., subsetting, merging, renaming) – as opposed to merely describing or analyzing – the supplied data.frame. Again, see the examples for the types of expressions/use cases envisioned.

Value

the value of the evaluated expr, with name labels substituted for variable (column) names.

Examples

```
# assign mtcars to new data.frame mt2
mt2 <- mtcars

# add name labs
mt2 <- add_name_labs(mt2,
  name.labs = c(
    "mpg" = "Miles/(US) gallon",
    "cyl" = "Number of cylinders",
    "disp" = "Displacement (cu.in.)",
    "hp" = "Gross horsepower",
```

```
    "drat" = "Rear axle ratio",
    "wt" = "Weight (1000 lbs)",
    "qsec" = "1/4 mile time",
    "vs" = "Engine (0 = V-shaped, 1 = straight)",
    "am" = "Transmission (0 = automatic, 1 = manual)",
    "gear" = "Number of forward gears",
    "carb" = "Number of carburetors"
  )
)


with_name_labs(mt2, t.test(mpg ~ am))
with_name_labs(mt2, lm(mpg ~ am))
with_name_labs(mt2, summary(mt2))
with_name_labs(mt2, cor(mt2, use = "pairwise.complete.obs"))
with_name_labs(mt2, xtabs(~gear))
xtabs(~ mt2$gear)
with_name_labs(mt2, cor(mpg, carb))
with_name_labs(mt2, hist(mpg))
with_name_labs(mt2, plot(mpg, carb))
with_name_labs(mt2, head(gear))
with_name_labs(mt2, summary(mt2))
```

`with_val_labs` *Evaluate an Expression in a Value Labels-on Data Environment*

Description

`with_val_labs` wraps a data.frame in `use_val_labs` and wraps the resulting data.frame in `base::with` in support of `base::with`-like non-standard evaluation (see examples).

Note: `wvl` is a compact alias for `with_val_labs`: they do the same thing, and the former is easier to type

Usage

```
with_val_labs(data, ...)

wvl(data, ...)
```

Arguments

`data`	a data.frame with value-labeled columns.
`...`	additional arguments passed to dots, typically an expression involving a function called on unquoted variable(s) (see examples).

Details

`with_val_labs` (see also alias `wvl`) is useful for applying certain nominal-variable-friendly functions (chiefly, `table` and the like) to value-labeled data.frames. See also `base::with`. See also `tabl`, `with_name_labs`, and `with_both_labs`.

Value

the value of the evaluated `expr`, with value labels substituted for variable values.

Examples

```
# make toy demographic (gender, raceth, etc.) data set
set.seed(555)
df <- make_demo_data(n = 1000) # another labelr:: function
# let's add variable VALUE labels for variable "raceth"
df <- add_val_labs(df,
  vars = "raceth", vals = c(1:7),
  labs = c("White", "Black", "Hispanic", "Asian", "AIAN", "Multi", "Other"),
  max.unique.vals = 50
)

# let's add variable VALUE labels for variable "gender"
# note that, if we are labeling a single variable, we can use add_val1()
# distinction between add_val1() and add_val_labs() will become more meaningful
# when we get to our Likert example
df <- add_val1(
  data = df, gender, vals = c(0, 1, 2, 3, 4),
  labs = c("M", "F", "TR", "NB", "Diff-Term"), max.unique.vals = 50
)

# "with_val_labs" - with()-like function that swaps value labels out for value values
# compare with(df, ...) to with_val_labs(df,...)
with(df, table(gender, raceth)) # without labels

# the same data (note that presentation order changes d/t alphabetical ordering)
with_val_labs(df, table(gender, raceth)) # with labels
with(use_val_labs(df), table(gender, raceth)) # above is shorthand for this

# just raceth
with(df, table(raceth)) # with
with_val_labs(df, table(raceth)) # with_val_labs

# another use case
with(df, unique(raceth)) # with
with_val_labs(df, unique(raceth)) # with_val_labs

# another
with(df, modelr::typical(raceth)) # numerical median!
with_val_labs(df, modelr::typical(raceth)) # modal label (not the median!)
```

Index