Package: huxtableone (via r-universe)

February 3, 2025

Title Descriptive Tables for Observational or Interventional Studies

Version 0.4.4

Description Generating tabular summaries of data in a format suitable for reporting in journal articles is fiddly and slows down more detailed analysis. Comparing two populations with respect to an intervention, and reporting it is a task that can be largely automated.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2.9003

Suggests covr, formatR, here, knitr, rmarkdown, survival, scales, ggplot2, tidyverse, htmltools, testthat

Config/testthat/edition 3

VignetteBuilder knitr

Depends R (>= 2.10)

LazyData true

Imports dplyr, huxtable (>= 5.5.1), magrittr, nortest, rlang, stringr, systemfonts, tibble, tidyr, utils, stats, forcats, binom, broom, glue, purrr, tidyselect (>= 1.2.0), pwr

URL https://ai4ci.github.io/huxtableone/, https://github.com/ai4ci/huxtableone, https://doi.org/10.5281/zenodo.14794616

BugReports https://github.com/ai4ci/huxtableone/issues

Config/pak/sysreqs libfontconfig1-dev libfreetype6-dev texlive libicu-dev libxml2-dev

Repository https://ai4ci.r-universe.dev

RemoteUrl https://github.com/ai4ci/huxtableone

RemoteRef 0.4.4

RemoteSha 6b92dd631eb438f883333395d0aac3f3c814c754

Contents

`as_huxtable.t1_shape` *Convert a* `t1_summary` *object to a* `huxtable`

Description

Convert a `t1_summary` object to a `huxtable`

Usage

```
## S3 method for class 't1_shape'
as_huxtable(
  x,
  ...,
  font_size = getOption("huxtableone.font_size", 8),
  font = getOption("huxtableone.font", "Arial"),
  footer_text = NULL,
  show_binary_value = NULL
)
```

Arguments

x	the t1_summary object as produced by describe_population
...	not used
font_size	(optional) the font size for the table in points
font	(optional) the font family for the table (which will be matched to closest on your system)
footer_text	any text that needs to be added at the end of the table, setting this to FALSE dsables the whole footer (as does options("huxtableone.hide_footer"=TRUE)).
show_binary_value	if set this will filter the display of covariates where the number of possibilities is exactly 2 to this value.

Value

a formatted table as a huxtable

---

as_huxtable.t1_signif *Convert a* t1_signif *S3 class to a huxtable*

Description

This is responsible for printing the significance test results and comparison

Usage

```
## S3 method for class 't1_signif'
as_huxtable(
  x,
  ...,
  layout = "compact",
  override_percent_dp = list(),
  override_real_dp = list(),
  p_format = names(.pvalue.defaults),
```

```
  font_size = getOption("huxtableone.font_size", 8),
  font = getOption("huxtableone.font", "Arial"),
  footer_text = NULL,
  show_binary_value = NULL
)
```

Arguments

x	the t1_signif result as calculated by compare_population(...)
...	not used
layout	(optional) various layouts are defined as default. As of this version of huxtableone they are "relaxed","compact","micro","simple","single","missing". The layouts can be customised using the options options("huxtableone.format_list"=list(...)"), and this is described in more detail in the vignettes.
override_percent_dp	(optional) a named list of overrides for the default precision of formatting percentages, following a c(<colname_1> = 2, <colname_2> = 4, ...) format. columns not present in this list will use the defaults defined in the layout. See the vignette on customisation.
override_real_dp	(optional) a named list of overrides for the default precision of formatting real values, following a c(<colname_1> = 2, <colname_2> = 4, ...) format. columns not present in this list will use the defaults defined in the layout. See the utils::vignette("customisation", package="huxtableone").
p_format	the format of the p-values: one of "sampl", "nejm", "jama", "lancet", "aim" but any value here is overridden by the option("huxtableone.pvalue_formatter"=function(...))
font_size	(optional) the font size for the table in points
font	(optional) the font family for the table (which will be matched to closest on your system)
footer_text	any text that needs to be added at the end of the table, setting this to FALSE dsables the whole footer (as does options("huxtableone.hide_footer"=TRUE)).
show_binary_value	if set this will filter the display of covariates where the number of possibilities is exactly 2 to this value.

Value

a formatted table as a huxtable

Examples

```
library(huxtableone)
tmp = iris %>% dplyr::group_by(Species) %>%
  as_t1_signif(tidyselect::everything()) %>%
  huxtable::as_huxtable()
```

`as_huxtable.t1_summary` *Convert a* `t1_summary` *object to a* `huxtable`

Description

Convert a `t1_summary` object to a `huxtable`

Usage

```
## S3 method for class 't1_summary'
as_huxtable(
  x,
  ...,
  layout = "single",
  override_percent_dp = list(),
  override_real_dp = list(),
  font_size = getOption("huxtableone.font_size", 8),
  font = getOption("huxtableone.font", "Arial"),
  footer_text = NULL,
  show_binary_value = NULL
)
```

Arguments

`x`	the `t1_summary` object as produced by describe_population
`...`	not used
`layout`	(optional) various layouts are defined as default. As of this version of `huxtableone` they are "relaxed","compact","micro","simple","single","missing". The layouts can be customised using the options `options("huxtableone.format_list"=list(...)")`, and this is described in more detail in the vignettes.
`override_percent_dp`	(optional) a named list of overrides for the default precision of formatting percentages, following a `c(<colname_1> = 2, <colname_2> = 4, ...)` format. columns not present in this list will use the defaults defined in the layout. See the vignette on customisation.
`override_real_dp`	(optional) a named list of overrides for the default precision of formatting real values, following a `c(<colname_1> = 2, <colname_2> = 4, ...)` format. columns not present in this list will use the defaults defined in the layout. See the `utils::vignette("customisation", package="huxtableone")`.
`font_size`	(optional) the font size for the table in points
`font`	(optional) the font family for the table (which will be matched to closest on your system)

`footer_text` any text that needs to be added at the end of the table, setting this to FALSE dsables the whole footer (as does `options("huxtableone.hide_footer"=TRUE)`).

`show_binary_value` if set this will filter the display of covariates where the number of possibilities is exactly 2 to this value.

Value

a formatted table as a `huxtable`

`as_t1_shape` *Summarise a data set*

Description

The data set description is a simple summary of the data formats, types and missingness

Usage

```
as_t1_shape(df, ..., label_fn = label_extractor(df), units = extract_units(df))
```

Arguments

`df` a dataframe of individual observations. Grouping, if present, is ignored. (n.b. if you wanted to construct multiple summary tables a `dplyr::group_map()` call could be used)

`...` the columns of variables we wish to summarise. This can be given as a `tidyselect` specification (see `utils::vignette("syntax", package = "tidyselect")`), identifying the columns. Alternatively it can be given as a formula of the nature

`outcome ~ intervention + covariate_1 + covariate_2 + ...`.

which may be more convenient if you are going on to do a model fit. If the latter format the left hand side is ignored (outcomes are not usual in this kind of table).

`label_fn` (optional) a function for mapping a co-variate column name to printable label. This is by default a no-operation and the output table will contain the dataframe column names as labels. A simple alternative would be some form of dplyr::case_when lookup, or a string function such as stringr::str_to_sentence. (N.b. this function must be vectorised). Any value provided here will be overridden by the `options("huxtableone.labeller" = my_label_fn)` which allows global setting of the labeller.

`units` (optional) a named list of units, following a `c(<colname_1> = "<unit_1>", <colname_2> = "<unit_2>` format. columns not present in this list are assumed to have no units. Units may be involved in the formatting of the summary output.

Value

a `t1_shape` data frame.

Examples

```
tmp = iris %>% as_t1_shape(
  tidyselect::everything()
)
```

`as_t1_signif` *Compares the population against an intervention*

Description

The population comparison is a summary of the co-variates in a data set with no reference to outcome, but comparing intervention groups. It will report summary statistics for continuous and counts for categorical data, for each of the intervention groups, and reports on the significance of the association in relation to the intervention groups. It gives a clear summary of whether data is correlated to intervention.

Usage

```
as_t1_signif(
  df,
  ...,
  label_fn = label_extractor(df),
  units = extract_units(df),
  override_type = list(),
  override_method = list()
)
```

Arguments

`df`	a dataframe of individual observations. If using the `tidyselect` syntax data grouping defines the intervention group and should be present. if the formula interface is used the first variable in the right hand side of the formula is used as the intervention, in which case grouping is ignored.
`...`	the columns of variables we wish to summarise. This can be given as a `tidyselect` specification (see `utils::vignette("syntax", package = "tidyselect")`), identifying the columns. Alternatively it can be given as a formula of the nature `outcome ~ intervention + covariate_1 + covariate_2 + ...` . which may be more convenient if you are going on to do a model fit later. If the latter format the left hand side is ignored (outcomes are not usual in this kind of table).
`label_fn`	(optional) a function for mapping a co-variate column name to printable label. This is by default a no-operation and the output table will contain the dataframe column names as labels. A simple alternative would be some form of dplyr::case_when lookup, or a string function such as stringr::str_to_sentence. (N.b. this function must be vectorised). Any value provided here will be overridden by the `options("huxtableone.labeller" = my_label_fn)` which allows global setting of the labeller.

units (optional) a named list of units, following a c(<colname_1> = "<unit_1>", <colname_2> = "<unit_2> format. columns not present in this list are assumed to have no units. Units may be involved in the formatting of the summary output.

override_type (optional) a named list of data summary types. The default type for a column in a data set are calculated using heurisitics depending on the nature of the data (categorical or continuous), and result of normality tests. if you want to override this the options are "subtype_count","median_iqr","mean_sd","skipped" and you specify this on a column by column bases with a named list (e.g c("Petal.Width"="mean_sd")). Overriding the default does not check the type of data is correct for the summary type and will potentially cause errors if this is not done correctly.

override_method if you want to override the comparison method for a particular variable the options are "chi-sq trend","fisher","t-test","2-sided wilcoxon","2-sided ks","anova","kruskal-wallis","no comparison" and you specify this on a column by column bases with a named list (e.g c("Petal.Width"="t-test"))

Value

a t1_signif dataframe.

Examples

```
tmp = iris %>% dplyr::group_by(Species) %>% as_t1_signif(tidyselect::everything())
tmp = diamonds %>% dplyr::group_by(is_colored) %>% as_t1_signif(tidyselect::everything())
```

---

as_t1_summary *Summarise a population*

Description

The population description is a simple summary of the co-variates in a data set with no reference to outcome, and not comparing intervention (although it might contain intervention rates.) It will report summary statistics for continuous and counts for categorical data,

Usage

```
as_t1_summary(
  df,
  ...,
  label_fn = label_extractor(df),
  units = extract_units(df),
  override_type = list()
)
```

Arguments

df	a dataframe of individual observations. Grouping, if present, is ignored. (n.b. if you wanted to construct multiple summary tables a dplyr::group_map() call could be used)
...	the columns of variables we wish to summarise. This can be given as a tidyselect specification (see utils::vignette("syntax", package = "tidyselect")), identifying the columns. Alternatively it can be given as a formula of the nature outcome ~ intervention + covariate_1 + covariate_2 + ... which may be more convenient if you are going on to do a model fit. If the latter format the left hand side is ignored (outcomes are not usual in this kind of table).
label_fn	(optional) a function for mapping a co-variate column name to printable label. This is by default a no-operation and the output table will contain the dataframe column names as labels. A simple alternative would be some form of dplyr::case_when lookup, or a string function such as stringr::str_to_sentence. (N.b. this function must be vectorised). Any value provided here will be overridden by the options("huxtableone.labeller" = my_label_fn) which allows global setting of the labeller.
units	(optional) a named list of units, following a c(<colname_1> = "<unit_1>", <colname_2> = "<unit_2> format. columns not present in this list are assumed to have no units. Units may be involved in the formatting of the summary output.
override_type	(optional) a named list of data summary types. The default type for a column in a data set are calculated using heurisitics depending on the nature of the data (categorical or continuous), and result of normality tests. if you want to override this the options are "subtype_count","median_iqr","mean_sd","skipped" and you specify this on a column by column bases with a named list (e.g c("Petal.Width"="mean_sd")). Overriding the default does not check the type of data is correct for the summary type and will potentially cause errors if this is not done correctly.

Value

a t1_summary data frame.

Examples

```
tmp = iris %>% as_t1_summary(
  tidyselect::everything(),
  override_type = c(Petal.Length = "mean_sd", Petal.Width = "mean_sd")
)
```

---

as_vars *Reuse tidy-select syntax outside of a tidy-select function*

Description

Reuse tidy-select syntax outside of a tidy-select function

Usage

```
as_vars(tidyselect, data = NULL)
```

Arguments

tidyselect	a tidyselect syntax which will be evaluated in context by looking for a call in the call stack that includes a dataframe as the first argument
data	(optional) a specific dataframe with which to evaluate the tidyselect

Value

a list of symbols resulting from the evaluation of the tidyselect in the context of the current call stack (or a provided data frame)

---

bad_test_cols *A list of columns for a test case*

Description

A list of columns for a test case

Usage

```
bad_test_cols
```

Format

bad_test_cols:
Test data

---

compare_missing *Compares missing data against an intervention in a summary table*

Description

The missing data summary is a simple summary of the missingness of co-variates in a data set with no reference to outcome, but comparing intervention groups. It reports summary counts for missingness in data and reports on the significance of that missingness in relation to the intervention groups, allowing a clear summary of whether data is missing at random compared to the intervention.

Usage

```
compare_missing(
  df,
  ...,
  label_fn = label_extractor(df),
  p_format = names(.pvalue.defaults),
  font_size = getOption("huxtableone.font_size", 8),
  font = getOption("huxtableone.font", "Arial"),
  significance_limit = 0.05,
  missingness_limit = 0.1,
  footer_text = NULL,
  raw_output = FALSE
)
```

Arguments

df	a dataframe of individual observations. If using the `tidyselect` syntax data grouping defines the intervention group and should be present. if the formula interface is used the first variable in the right hand side of the formula is used as the intervention, in which case grouping is ignored.
...	the columns of variables we wish to summarise. This can be given as a `tidyselect` specification (see `utils::vignette("syntax", package = "tidyselect")`), identifying the columns. Alternatively it can be given as a formula of the nature `outcome ~ intervention + covariate_1 + covariate_2 + ...` . which may be more convenient if you are going on to do a model fit later. If the latter format the left hand side is ignored (outcomes are not usual in this kind of table).
label_fn	(optional) a function for mapping a co-variate column name to printable label. This is by default a no-operation and the output table will contain the dataframe column names as labels. A simple alternative would be some form of dplyr::case_when lookup, or a string function such as stringr::str_to_sentence. (N.b. this function must be vectorised). Any value provided here will be overridden by the `options("huxtableone.labeller" = my_label_fn)` which allows global setting of the labeller.
p_format	the format of the p-values: one of "sampl", "nejm", "jama", "lancet", "aim" but any value here is overridden by the `option("huxtableone.pvalue_formatter"=function(...))`
font_size	(optional) the font size for the table in points
font	(optional) the font family for the table (which will be matched to closest on your system)
significance_limit	the limit at which we reject the hypothesis that the data is missing at random.
missingness_limit	the limit at which too much data is missing to include the predictor.
footer_text	any text that needs to be added at the end of the table, setting this to FALSE dsables the whole footer (as does `options("huxtableone.hide_footer"=TRUE)`).
raw_output	return comparison as tidy dataframe rather than formatted table

Value

a huxtable formatted table.

Examples

```
# this option lets us change the column name for p value from its default
# "P value"
old = options("huxtableone.pvalue_column_name"="p-value")

# missing at random
missing_diamonds %>% dplyr::group_by(is_colored) %>% compare_missing(tidyselect::everything())

# nothing missing
iris %>% dplyr::group_by(Species) %>% compare_missing(tidyselect::everything())

# MNAR: by design missingness is correlated with grouping
mnar_two_class_1000 %>% dplyr::group_by(grouping) %>% compare_missing(tidyselect::everything())

options(old)
```

---

compare_outcomes *Compares multiple outcomes against an intervention in a summary table*

Description

The outcome table is a simple summary of a binary or categorical outcome in a data set compared by intervention groups. The comparison is independent of any covariates, and is a preliminary output prior to more formal statistical analysis or model fitting.

Usage

```
compare_outcomes(
  df,
  ...,
  label_fn = label_extractor(df),
  units = extract_units(df),
  override_type = list(),
  layout = "compact",
  override_percent_dp = list(),
  override_real_dp = list(),
  p_format = names(.pvalue.defaults),
  font_size = getOption("huxtableone.font_size", 8),
  font = getOption("huxtableone.font", "Arial"),
  footer_text = NULL,
  show_binary_value = NULL,
  raw_output = FALSE
)
```

Arguments

df	a dataframe of individual observations. If using the `tidyselect` syntax data grouping defines the intervention group and should be present. if the formula interface is used the first variable in the right hand side of the formula is used as the intervention, in which case grouping is ignored.
...	the outcomes are specified either as a `tidyselect` specification, in which case the grouping of the `df` input determines the intervention and the output is the same as a `compare_population()` call with a tidyselect. Alternatively a set of formulae can be provided that specify the outcomes on the left hand side, e.g. `outcome1 ~ intervention + cov1, outcome2 ~ intervention + cov1, ...` in this case the `intervention` must be the same for all formulae and used to determine the comparison groups.
label_fn	(optional) a function for mapping a co-variate column name to printable label. This is by default a no-operation and the output table will contain the dataframe column names as labels. A simple alternative would be some form of dplyr::case_when lookup, or a string function such as stringr::str_to_sentence. (N.b. this function must be vectorised). Any value provided here will be overridden by the `options("huxtableone.labeller" = my_label_fn)` which allows global setting of the labeller.
units	(optional) a named list of units, following a `c(<colname_1> = "<unit_1>", <colname_2> = "<unit_2>` format. columns not present in this list are assumed to have no units. Units may be involved in the formatting of the summary output.
override_type	(optional) a named list of data summary types. The default type for a column in a data set are calculated using heurisitics depending on the nature of the data (categorical or continuous), and result of normality tests. if you want to override this the options are "subtype_count","median_iqr","mean_sd","skipped" and you specify this on a column by column bases with a named list (e.g `c("Petal.Width"="mean_sd")`). Overriding the default does not check the type of data is correct for the summary type and will potentially cause errors if this is not done correctly.
layout	(optional) various layouts are defined as default. As of this version of `huxtableone` they are "relaxed","compact","micro","simple","single","missing". The layouts can be customised using the options `options("huxtableone.format_list"=list(...)")`, and this is described in more detail in the vignettes.
override_percent_dp	(optional) a named list of overrides for the default precision of formatting percentages, following a `c(<colname_1> = 2, <colname_2> = 4, ...)` format. columns not present in this list will use the defaults defined in the layout. See the vignette on customisation.
override_real_dp	(optional) a named list of overrides for the default precision of formatting real values, following a `c(<colname_1> = 2, <colname_2> = 4, ...)` format. columns not present in this list will use the defaults defined in the layout. See the `utils::vignette("customisation", package="huxtableone")`.
p_format	the format of the p-values: one of "sampl", "nejm", "jama", "lancet", "aim" but any value here is overridden by the `option("huxtableone.pvalue_formatter"=function(...))`
font_size	(optional) the font size for the table in points

font	(optional) the font family for the table (which will be matched to closest on your system)
footer_text	any text that needs to be added at the end of the table, setting this to FALSE dsables the whole footer (as does options("huxtableone.hide_footer"=TRUE)).
show_binary_value	if set this will filter the display of covariates where the number of possibilities is exactly 2 to this value.
raw_output	return comparison as t1_signif dataframe rather than formatted table

Details

It reports summary counts for the outcomes and a measure of significance of the relationship between outcome and intervention. Interpretation of significance tests, should include Bonferroni adjustment.

Value

a huxtable formatted table.

---

compare_population *Compares the population against an intervention in a summary table*

Description

The population comparison is a summary of the co-variates in a data set with no reference to outcome, but comparing intervention groups. It will report summary statistics for continuous and counts for categorical data, for each of the intervention groups, and reports on the significance of the association in relation to the intervention groups. It gives a clear summary of whether data is correlated to intervention.

Usage

```
compare_population(
  df,
  ...,
  label_fn = label_extractor(df),
  units = extract_units(df),
  override_type = list(),
  override_method = list(),
  layout = "compact",
  override_percent_dp = list(),
  override_real_dp = list(),
  p_format = names(.pvalue.defaults),
  font_size = getOption("huxtableone.font_size", 8),
  font = getOption("huxtableone.font", "Arial"),
  footer_text = NULL,
  show_binary_value = NULL,
```

```
  raw_output = FALSE
)
```

Arguments

df	a dataframe of individual observations. If using the tidyselect syntax data grouping defines the intervention group and should be present. if the formula interface is used the first variable in the right hand side of the formula is used as the intervention, in which case grouping is ignored.
...	the columns of variables we wish to summarise. This can be given as a tidyselect specification (see utils::vignette("syntax", package = "tidyselect")), identifying the columns. Alternatively it can be given as a formula of the nature outcome ~ intervention + covariate_1 + covariate_2 + ... which may be more convenient if you are going on to do a model fit later. If the latter format the left hand side is ignored (outcomes are not usual in this kind of table).
label_fn	(optional) a function for mapping a co-variate column name to printable label. This is by default a no-operation and the output table will contain the dataframe column names as labels. A simple alternative would be some form of dplyr::case_when lookup, or a string function such as stringr::str_to_sentence. (N.b. this function must be vectorised). Any value provided here will be overridden by the options("huxtableone.labeller" = my_label_fn) which allows global setting of the labeller.
units	(optional) a named list of units, following a c(<colname_1> = "<unit_1>", <colname_2> = "<unit_2> format. columns not present in this list are assumed to have no units. Units may be involved in the formatting of the summary output.
override_type	(optional) a named list of data summary types. The default type for a column in a data set are calculated using heurisitics depending on the nature of the data (categorical or continuous), and result of normality tests. if you want to override this the options are "subtype_count","median_iqr","mean_sd","skipped" and you specify this on a column by column bases with a named list (e.g c("Petal.Width"="mean_sd")). Overriding the default does not check the type of data is correct for the summary type and will potentially cause errors if this is not done correctly.
override_method	if you want to override the comparison method for a particular variable the options are "chi-sq trend","fisher","t-test","2-sided wilcoxon","2-sided ks","anova","kruskal-wallis","no comparison" and you specify this on a column by column bases with a named list (e.g c("Petal.Width"="t-test"))
layout	(optional) various layouts are defined as default. As of this version of huxtableone they are "relaxed","compact","micro","simple","single","missing". The layouts can be customised using the options options("huxtableone.format_list"=list(...)"), and this is described in more detail in the vignettes.
override_percent_dp	(optional) a named list of overrides for the default precision of formatting percentages, following a c(<colname_1> = 2, <colname_2> = 4, ...) format. columns not present in this list will use the defaults defined in the layout. See the vignette on customisation.

override_real_dp
: (optional) a named list of overrides for the default precision of formatting real values, following a `c(<colname_1> = 2, <colname_2> = 4, ...)` format. columns not present in this list will use the defaults defined in the layout. See the `utils::vignette("customisation", package="huxtableone")`.

p_format
: the format of the p-values: one of "sampl", "nejm", "jama", "lancet", "aim" but any value here is overridden by the `option("huxtableone.pvalue_formatter"=function(...))`

font_size
: (optional) the font size for the table in points

font
: (optional) the font family for the table (which will be matched to closest on your system)

footer_text
: any text that needs to be added at the end of the table, setting this to FALSE dsables the whole footer (as does `options("huxtableone.hide_footer"=TRUE)`).

show_binary_value
: if set this will filter the display of covariates where the number of possibilities is exactly 2 to this value.

raw_output
: return comparison as `t1_signif` dataframe rather than formatted table

Value

a `huxtable` formatted table.

Examples

```
# the heuristics detect that Petals in the iris data set are not normally
# distributed and hence report median and IQR:
iris %>% dplyr::group_by(Species) %>% compare_population(tidyselect::everything())

# Missing data
old = options("huxtableone.show_pvalue_method"=FALSE)
missing_diamonds %>%
  dplyr::group_by(is_colored) %>%
  compare_population(-color, layout="relaxed")

tmp = missing_diamonds %>% explicit_na() %>% dplyr::group_by(is_colored)
tmp %>% compare_population(-color,
    footer_text = c(
      "IQR: Interquartile range; CI: Confidence interval",
      "Line two")
    )

options(old)
```

---

count_table *Group data count and calculate proportions by column.*

Description

Group data count and calculate proportions by column.

Usage

```
count_table(
  df,
  rowGroupVars,
  colGroupVars,
  numExpr = dplyr::n(),
  denomExpr = dplyr::n(),
  totalExpr = dplyr::n(),
  subgroupLevel = length(rowGroupVars),
  glue = list(`Count [%] (N={sprintf("%d",N)})` =
    "{sprintf(\"%d/%d [%1.1f%%]\", x, n, mean*100)}"),
  label_fn = label_extractor(df),
  font_size = getOption("huxtableone.font_size", 8),
  font = getOption("huxtableone.font", "Arial")
)
```

Arguments

df	a dataframe of linelist items
rowGroupVars	the rows of the table. The last one of these is the denominator grouping
colGroupVars	the column groupings of the table.
numExpr	defines how the numerator is defined in the context of the column and row groups (e.g. dplyr::n())
denomExpr	defines how the numerator is defined in the context of the column and row (ungrouped one level)
totalExpr	defines how the column level total is defined
subgroupLevel	defines how the numerator grouping is defined in terms of the row groupings
glue	a named list of column value specifications.
label_fn	(optional) a function for mapping a co-variate column name to printable label. This is by default a no-operation and the output table will contain the dataframe column names as labels. A simple alternative would be some form of dplyr::case_when lookup, or a string function such as stringr::str_to_sentence. (N.b. this function must be vectorised). Any value provided here will be overridden by the `options("huxtableone.labeller" = my_label_fn)` which allows global setting of the labeller.
font_size	(optional) the font size for the table in points
font	(optional) the font family for the table (which will be matched to closest on your system)

Value

a huxtable with the count and proportions of the rows groups

Examples

```
diamonds %>% count_table(dplyr::vars(cut,clarity), dplyr::vars(color), subgroupLevel = 1)
```

cut_integer *Cut and label an integer valued quantity*

Description

Deals with some annoying issues classifying integer data sets, such as ages, into groups. where you want to specify just the change over points as integers and clearly label the resulting ordered factor.

Usage

```
cut_integer(
  x,
  cut_points,
  glue = "{label}",
  lower_limit = -Inf,
  upper_limit = Inf,
  ...
)
```

Arguments

x	a vector of integer valued numbers, e.g. ages, counts
cut_points	a vector of integer valued cut points which define the lower, inclusive boundary of each group
glue	a glue spec that may be used to generate a label. It can use {low}, {high}, {next_low}, or {label} as values.
lower_limit	the minimum value we should include (this is inclusive for the bottom category) (default -Inf)
upper_limit	the maximum value we should include (this is also inclusive for the top category) (default Inf)
...	not used

Value

an ordered factor of the integer

Examples

```
cut_integer(stats::rbinom(20,20,0.5), c(5,10,15))
cut_integer(floor(stats::runif(100,-10,10)), cut_points = c(2,3,4,6), lower_limit=0, upper_limit=10)
cut_integer(1:10, cut_points = c(1,3,9))
```

default.format *Default table layout functions*

Description

Customisation of output can use one of these entries as a starting point. A custom layout should look like one of the entries in level 2 of this nested list, containing 4 named entries, one for each type of table summary.

Usage

```
default.format
```

Format

`default.format`**:**

A names list of lists:

level one The name of the table layout

level two The name of the summary type required. one of `subtype_count`, `median_iqr`,`mean_sd`,`skipped`

level three a named list of `column=glue specification` pairs. The `column` (itself a glue spec) might reference `N_total`, `N_present` or `.unit` but typically will be a fixed string- it defines the name of the table column to generate. The `glue specification` defines the layout of that column, and can use summary statistics as below

subtype_count can use `level`, `prob.0.5`, `prob.0.025`, `prob.0.975`, `unit`, `n`, `N`. `n` is subgroup count, `N` is data count.

median_iqr can use `q.0.5`, `q.0.25`, ..., `unit`, `n`, `N - n` excludes missing, `N` does not.

mean_sd can use `mean`, `sd`, `unit`, `n`, `N - n` excludes missing, `N` does not.

skipped can use `n`, `N - n` excludes missing, `N` does not.

describe_data *Describe the data types and consistence*

Description

The population description is a simple summary of the co-variates in a data set with no reference to outcome, and not comparing intervention (although it might contain intervention rates.) It will report summary statistics for continuous and counts for categorical data,

Usage

```
describe_data(
  df,
  ...,
  label_fn = label_extractor(df),
  units = extract_units(df),
  layout = "single",
  font_size = getOption("huxtableone.font_size", 8),
  font = getOption("huxtableone.font", "Arial"),
  footer_text = NULL,
  raw_output = FALSE
)
```

Arguments

df	a dataframe of individual observations. Grouping, if present, is ignored. (n.b. if you wanted to construct multiple summary tables a dplyr::group_map() call could be used)
...	the columns of variables we wish to summarise. This can be given as a tidyselect specification (see utils::vignette("syntax", package = "tidyselect")), identifying the columns. Alternatively it can be given as a formula of the nature outcome ~ intervention + covariate_1 + covariate_2 + ... which may be more convenient if you are going on to do a model fit. If the latter format the left hand side is ignored (outcomes are not usual in this kind of table).
label_fn	(optional) a function for mapping a co-variate column name to printable label. This is by default a no-operation and the output table will contain the dataframe column names as labels. A simple alternative would be some form of dplyr::case_when lookup, or a string function such as stringr::str_to_sentence. (N.b. this function must be vectorised). Any value provided here will be overridden by the options("huxtableone.labeller" = my_label_fn) which allows global setting of the labeller.
units	(optional) a named list of units, following a c(<colname_1> = "<unit_1>", <colname_2> = "<unit_2> format. columns not present in this list are assumed to have no units. Units may be involved in the formatting of the summary output.
layout	(optional) various layouts are defined as default. As of this version of huxtableone they are "relaxed","compact","micro","simple","single","missing". The layouts can be customised using the options options("huxtableone.format_list"=list(...)"), and this is described in more detail in the vignettes.
font_size	(optional) the font size for the table in points
font	(optional) the font family for the table (which will be matched to closest on your system)
footer_text	any text that needs to be added at the end of the table, setting this to FALSE dsables the whole footer (as does options("huxtableone.hide_footer"=TRUE)).
raw_output	return comparison as t1_signif dataframe rather than formatted table

Value

a huxtable formatted table.

Examples

```
# Overriding the heuristics is possible:
iris %>% describe_data(tidyselect::everything())

diamonds %>% dplyr::group_by(is_colored) %>% describe_data(tidyselect::everything())
```

---

describe_population *Describe the population in a summary table*

Description

The population description is a simple summary of the co-variates in a data set with no reference to outcome, and not comparing intervention (although it might contain intervention rates.) It will report summary statistics for continuous and counts for categorical data,

Usage

```
describe_population(
  df,
  ...,
  label_fn = label_extractor(df),
  units = extract_units(df),
  override_type = list(),
  layout = "single",
  override_percent_dp = list(),
  override_real_dp = list(),
  font_size = getOption("huxtableone.font_size", 8),
  font = getOption("huxtableone.font", "Arial"),
  footer_text = NULL,
  show_binary_value = NULL,
  raw_output = FALSE
)
```

Arguments

df	a dataframe of individual observations. Grouping, if present, is ignored. (n.b. if you wanted to construct multiple summary tables a dplyr::group_map() call could be used)
...	the columns of variables we wish to summarise. This can be given as a tidyselect specification (see utils::vignette("syntax", package = "tidyselect")), identifying the columns. Alternatively it can be given as a formula of the nature outcome ~ intervention + covariate_1 + covariate_2 + ...

which may be more convenient if you are going on to do a model fit. If the latter format the left hand side is ignored (outcomes are not usual in this kind of table).

`label_fn` (optional) a function for mapping a co-variate column name to printable label. This is by default a no-operation and the output table will contain the dataframe column names as labels. A simple alternative would be some form of dplyr::case_when lookup, or a string function such as stringr::str_to_sentence. (N.b. this function must be vectorised). Any value provided here will be overridden by the `options("huxtableone.labeller" = my_label_fn)` which allows global setting of the labeller.

`units` (optional) a named list of units, following a `c(<colname_1> = "<unit_1>", <colname_2> = "<unit_2>` format. columns not present in this list are assumed to have no units. Units may be involved in the formatting of the summary output.

`override_type` (optional) a named list of data summary types. The default type for a column in a data set are calculated using heurisitics depending on the nature of the data (categorical or continuous), and result of normality tests. if you want to override this the options are "subtype_count","median_iqr","mean_sd","skipped" and you specify this on a column by column bases with a named list (e.g `c("Petal.Width"="mean_sd")`). Overriding the default does not check the type of data is correct for the summary type and will potentially cause errors if this is not done correctly.

`layout` (optional) various layouts are defined as default. As of this version of `huxtableone` they are "relaxed","compact","micro","simple","single","missing". The layouts can be customised using the options `options("huxtableone.format_list"=list(...)")`, and this is described in more detail in the vignettes.

`override_percent_dp`
(optional) a named list of overrides for the default precision of formatting percentages, following a `c(<colname_1> = 2, <colname_2> = 4, ...)` format. columns not present in this list will use the defaults defined in the layout. See the vignette on customisation.

`override_real_dp`
(optional) a named list of overrides for the default precision of formatting real values, following a `c(<colname_1> = 2, <colname_2> = 4, ...)` format. columns not present in this list will use the defaults defined in the layout. See the `utils::vignette("customisation", package="huxtableone")`.

`font_size` (optional) the font size for the table in points

`font` (optional) the font family for the table (which will be matched to closest on your system)

`footer_text` any text that needs to be added at the end of the table, setting this to FALSE dsables the whole footer (as does `options("huxtableone.hide_footer"=TRUE)`).

`show_binary_value`
if set this will filter the display of covariates where the number of possibilities is exactly 2 to this value.

`raw_output` return comparison as `t1_signif` dataframe rather than formatted table

Value

a `huxtable` formatted table.

Examples

```
# the heuristics detect that Petals in the iris data set are not normally
# distributed and hence report median and IQR:
iris %>% describe_population(tidyselect::everything())

# Overriding the heuristics is possible:
iris %>% describe_population(
  tidyselect::everything(),
  override_type = c(Petal.Length = "mean_sd", Petal.Width = "mean_sd")
)

# The counts sometimes seem redundant if there is no missing information:
diamonds %>% describe_population(tidyselect::everything())

# however in a data set with missing values the denominators are important:
missing_diamonds %>% describe_population(tidyselect::everything())

# for factor levels we can make the missing values more explicit
missing_diamonds %>% explicit_na() %>%
  describe_population(tidyselect::everything())

# in the output above the price variable is not # presented the way we would
# like so here we override the number of decimal places shown for the price
# variable while we are at it we will use a mid point for the decimal point,
# and make the variable labels sentence case.

old = options("huxtableone.dp"="\u00B7")
missing_diamonds %>%
  explicit_na() %>%
  describe_population(
    tidyselect::everything(),
    label_fn=stringr::str_to_sentence,
    override_real_dp=list(price=6)
  )
options(old)
```

`diamonds` *A copy of the diamonds dataset*

Description

with a binary class is_coloured based on the color column

Usage

```
diamonds
```

Format

`diamonds`:
Test data

`explicit_na` *Make NA values in factor columns explicit*

Description

Converts NA values in any factors in the dataframe into a new level - This is a thin wrapper for `forcats::fct_explicit_na()` but with missing value level added regardless of whether any values missing. This forces an empty row in count tables.

Usage

```
explicit_na(df, na_level = "<missing>", hide_if_empty = FALSE)
```

Arguments

`df`	the data frame
`na_level`	a label for NA valued factors
`hide_if_empty`	dont add a missing data category if no data is missing

Value

the dataframe with all factor columns containing explicit na values

Examples

```
# before
missing_diamonds %>% dplyr::group_by(cut) %>% dplyr::count()
# after
missing_diamonds %>% explicit_na() %>% dplyr::group_by(cut) %>% dplyr::count()
```

`extract_comparison` *Get summary comparisons and statistics between variables as raw data.*

Description

Get summary comparisons and statistics between variables as raw data.

Usage

```
extract_comparison(
  df,
  ...,
  label_fn = label_extractor(df),
  override_type = list(),
  p_format = names(.pvalue.defaults),
  override_method = list(),
  power_analysis = FALSE,
  override_power = list(),
  raw_output = FALSE
)
```

Arguments

df	a dataframe of individual observations. If using the `tidyselect` syntax data grouping defines the intervention group and should be present. if the formula interface is used the first variable in the right hand side of the formula is used as the intervention, in which case grouping is ignored.
...	the outcomes are specified either as a `tidyselect` specification, in which case the grouping of the `df` input determines the intervention and the output is the same a `compare_population()` call with a tidyselect. Alternatively a set of formulae can be provided that specify the outcomes on the left hand side, e.g. `outcome1 ~ intervention + cov1, outcome2 ~ intervention + cov1, ...` in this case the `intervention` must be the same for all formulae and used to determine the comparison groups.
label_fn	(optional) a function for mapping a co-variate column name to printable label. This is by default a no-operation and the output table will contain the dataframe column names as labels. A simple alternative would be some form of dplyr::case_when lookup, or a string function such as stringr::str_to_sentence. (N.b. this function must be vectorised). Any value provided here will be overridden by the `options("huxtableone.labeller" = my_label_fn)` which allows global setting of the labeller.
override_type	(optional) a named list of data summary types. The default type for a column in a data set are calculated using heurisitics depending on the nature of the data (categorical or continuous), and result of normality tests. if you want to override this the options are "subtype_count","median_iqr","mean_sd","skipped" and you specify this on a column by column bases with a named list (e.g `c("Petal.Width"="mean_sd")`). Overriding the default does not check the type of data is correct for the summary type and will potentially cause errors if this is not done correctly.
p_format	the format of the p-values: one of "sampl", "nejm", "jama", "lancet", "aim" but any value here is overridden by the `option("huxtableone.pvalue_formatter"=function(...))`
override_method	if you want to override the comparison method for a particular variable the options are "chi-sq trend","fisher","t-test","2-sided wilcoxon","2-sided ks","anova","kruskal-wallis","no comparison" and you specify this on a column by column bases with a named list (e.g `c("Petal.Width"="t-test")`)

power_analysis	conduct sample size based power analysis.
override_power	if you want to override the power calculation method for a particular variable the options are "fisher","t-test","2-sided wilcoxon","2-sided ks","anova","kruskal-wallis","no comparison" and you specify this on a column by column bases with a named list (e.g `c("Petal.Width"="t-test")`)
raw_output	return comparison as `t1_signif` dataframe rather than formatted table

Value

a list of accessor functions for the summary data allowing granular access to the results of the analysis:

- `comparison$compare(.variable, .characteristic = NULL)` - prints a comparison between the different intervention groups for the specified variable (and optionally the given characteristic if it is a categorical variable).
- `comparison$filter(.variable, .intervention = NULL, .characteristic = NULL)` extracts a given variable (e.g. `gender`), optionally for a given level of intervention (e.g. `control`) and if categorical a given characteristic (e.g. `male`). This will output a dataframe with all the calculated summary variables, for all qualifying intervention, variable and characteristic combinations, significance tests (and power analyses) for the qualifying variable (comparing intervention groups).
- `comparison$signif_tests(.variable)` - extracts for a given variable (e.g. `gender`) the significance tests (and optionally power analyses) of the univariate comparison between different interventions and the variable.
- `comparison$summary_stats(.variable, .intervention = NULL, .characteristic = NULL)` extracts a given variable (e.g. `gender`), optionally for a given level of intervention (e.g. `control`) and if categorical a given characteristic (e.g. `male`). This returns only the summary stats for all qualifying intervention, variable and characteristic combinations.

---

extract_units *Extracts units set as dataframe column attributes*

Description

Extracts units set as dataframe column attributes

Usage

```
extract_units(df)
```

Arguments

df	the data frame from `set_units()`

Value

a named list of column / unit pairs.

Examples

```
iris = iris %>% set_units(-Species, units="mm")
iris %>% extract_units()
```

---

format_pvalue *Format a p-value*

Description

Uses the default formatter set globally in `options("huxtableone.pvalue_formatter")` in preference the one defined by `p_format` which is only used if no default is set.

Usage

```
format_pvalue(p.value, p_format = names(.pvalue.defaults))
```

Arguments

`p.value`	the p-value to be formatted
`p_format`	a name of a p-value formatter (one of sampl, nejm, jama, lancet, aim)

Value

a formatted P-value

---

get_footer_text *Get footer text if available*

Description

The functions in `huxtableone` will record the methods used for reporting in a scientific paper. This is both for normality assumption tests and for significance tests.

Usage

```
get_footer_text(df_output)
```

Arguments

`df_output`	a data frame that is the output of a `huxtableone` function

Value

the footnotes if they exist as a list (NULL otherwise)

Examples

```
iris %>% describe_population(tidyselect::everything()) %>% get_footer_text()
iris %>% dplyr::group_by(Species) %>%
  compare_population(tidyselect::everything()) %>% get_footer_text()
```

`group_comparison` *Extract one or more comparisons for inserting into text.*

Description

At some point we need to take information from the tables produced by `huxtableone` and place it into the main text of the document. It is annoying if this cannot be done automatically. the `group_comparison()` function enables extraction of one or more head to head comparisons and provides a fairly flexible mechanism for building the precise format desired.

Usage

```
group_comparison(
  t1_signif,
  variable = NULL,
  subgroup = NULL,
  intervention = NULL,
  percent_fmt = "%1.1f%%",
  p_format = names(.pvalue.defaults),
  no_summary = FALSE,
  summary_glue = NULL,
  summary_arrange = NULL,
  summary_sep = ", ",
  summary_last = " versus ",
  no_signif = FALSE,
  signif_glue = NULL,
  signif_sep = NULL,
  signif_last = NULL
)
```

Arguments

`t1_signif`	a `t1_signif` as produced by `as_t1_signif()` or `compare_population(..., raw_output = TRUE)`.
`variable`	a variable or set of variables to compare. If missing a set of approriate values is displayed based on the columns of `t1_signif`
`subgroup`	a subgroup or set of subgroups to compare.
`intervention`	the side or sides of the intervention to select. N.b. using this effectively prevents any statistical comparison as only one side will be available.
`percent_fmt`	a `sprintf` format string that is applied to probability fields in the summary data to convert to percentages.

p_format	the format of the p-values: one of "sampl", "nejm", "jama", "lancet", "aim" but any value here is overridden by the option("huxtableone.pvalue_formatter"=function(...))
no_summary	only extract significance test values
summary_glue	a glue specification that maps the summary statistics to a readable string.
summary_arrange	an expression by which to order the summary output
summary_sep	a separator to combine the summary output (see glue::glue_collapse())
summary_last	a separator to combine the last 2 summary outputs (see glue::glue_collapse())
no_signif	do not try and include significance in the output. Sometimes this is the only option if there is not enough of the comparison to retained by the variable, subgroup, and intervention filters. (Specifically if there is only a comparison between different subgroups, as the p-values will be for the different comparison between intervention groups.)
signif_glue	a glue specification that maps the combined summary output with the result of the significance tests, to given a complete comparison.
signif_sep	a separator to combine complete comparisons (see glue::glue_collapse())
signif_last	a separator to combine the last 2 complete comparisons (see glue::glue_collapse())

Value

ideally a single string but various things will be returned depending on hos much input is constrained, and sometimes will provide guidance about what next to do. The intention is the function to be used interactively until a satisfactory result is obtained.

Examples

```
tmp = diamonds %>%
  dplyr::group_by(is_colored) %>%
  set_units(price,units="£") %>%
  compare_population(-color, raw_output=TRUE)

# The tabular output is retrieved by converting to a huxtable
# as_huxtable(tmp, layout="simple")

# An unqualified group_comparison call gives informative messages
# about what can be compared:
tmp %>% group_comparison()

# filtering down the data gets us to a specific comparison:
tmp %>% group_comparison(variable = "cut", subgroup="Fair") %>% dplyr::glimpse()

# With further interactive exploration the
# data available for that comparison can be made into a glue string
tmp %>% group_comparison(variable = "cut", subgroup="Fair", intervention = "clear",
  summary_glue = "{is_colored}: {x}/{n} ({prob.0.5}%)",
 signif_glue = "{variable}={subgroup}; {text}; Overall p-value for '{variable}': {p.value}.")

# group comparisons above using many individual subgroups are a bit confusing because
```

```
# the p-value is at the variable level. This is less of an issue for continuous
# or binary values.
tmp %>% group_comparison(
  variable = "price",
  summary_glue = "{is_colored}: {unit}{q.0.5}; IQR: {q.0.25} \u2014 {q.0.75} (n={n})",
  signif_glue = "{variable}: {text}; P-value {p.value}.")

# Sometimes we only want to extract a p-value:
tmp %>%
  group_comparison(variable = "cut", subgroup="Fair", no_summary=TRUE) %>%
  dplyr::glimpse()
```

`label_extractor` *Extract labels from a dataframe column attributes*

Description

Retrieve column labels are embedded as an attribute of each column.

Usage

```
label_extractor(df, ..., attribute = "label")
```

Arguments

`df`	a dataframe containing some labels
`...`	additional string manipulation functions to apply e.g. `tolower`
`attribute`	the name of the label containing attribute (defaults to `"label"`)

Value

a labelling function. This is specific to the dataframe provided in `df`

Examples

```
iris = set_labels(iris, c(
    "Sepal Length", "Sepal Width",
    "Petal Length", "Petal Width",  "Species"
 ))
fn = label_extractor(iris,tolower)
fn(colnames(iris))
```

make_factors *Convert discrete data to factors*

Description

It is simpler for presentation and sometimes more correct for discrete valued data to be represented as factors. Such discrete valued data might be logical values, character values, or numeric values with a limited number of levels (e.g. scores). this function lets you convert (a subset of) data frame columns into factors using

Usage

```
make_factors(
  df,
  ...,
  .logical = c("yes", "no"),
  .numeric = "{name}={value}",
  .character = NULL
)
```

Arguments

df	a data frame
...	either a `tidyselect` specification or a formula with the right hand side defining the columns to convert (left hand side is ignored)
.logical	(optional) a length 2 vector defining the levels of TRUE, then FALSE.
.numeric	(optional) if provided it must either be a named list e.g. `c(column_name = "{name}:{value}", ..., .default="{value}")` pairs which define the way in which numeric columns are converted to factor levels. If a single value is given then all numerics are converted in the same way (this is the default). If there are some values that you are not certain you want to convert setting a limit on the maximum number of levels in a generated factor may be a good idea (i.e. `options("huxtableone.max_discrete_levels"=16)`) otherwise all values are converted
.character	in general character columns are converted into a factor with the default levels. To explicitly set levels a named list can be given here which `c(colname_1 = c("level_1", "level_2", ...), colname_2 = ...)`

Value

a dataframe with the columns converted to factors

Examples

```
iris %>%
  make_factors(tidyselect::ends_with("Length"), .numeric = "{name}={round(value)}") %>%
  dplyr::glimpse()

# Convert everything in diamonds to be a factor, rounding all
# the numeric values and converting all the names to upper case
tmp = diamonds %>%
  dplyr::mutate(is_colored = color > "F") %>%
  make_factors(tidyselect::everything(), .numeric="{toupper(name)}={round(value)}")

# as we included `price` which has very many levels one factor is unuseable with 11602 levels:
length(levels(tmp$price))

# we could explicitly exclude it from the `tidyselect` syntax `...` parameter:
diamonds %>% dplyr::mutate(is_colored = color > "F") %>%
  make_factors(-price, .numeric="{toupper(name)}={round(value)}") %>%
  dplyr::glimpse()

# or alternatively we set a limit on the maximum number of factors, which
# in this example picks up the `depth` and `table` columns as exceeding this
# new limit:

old = options("huxtableone.max_discrete_levels"=16)
diamonds %>% dplyr::mutate(is_colored = color > "F") %>%
  make_factors(tidyselect::everything(), .numeric="{toupper(name)}={round(value)}") %>%
  dplyr::glimpse()

options(old)

# converting a character vector. Here we specify `.character` as a list giving the
# possible levels of `alpha2`. Values outside of this list are converted to `NA`

set.seed(100)
eg_character = tibble::tibble(
  alpha1 =  sample(letters,50,replace=TRUE),
  alpha2 = sample(LETTERS,50,replace=TRUE)
)

eg_character %>%
  make_factors(tidyselect::everything(), .character = list(alpha2 = LETTERS[3:20]))
```

`missing_diamonds` *A copy of the diamonds dataset*

Description

with 10% of entries replaced by NA and a binary class is_coloured based on the color column

Usage

```
missing_diamonds
```

Format

`missing_diamonds`:
Test data

---

mnar_two_class_1000 *Missing not at random 2 class 1000 items*

Description

A random data test dataset with 2 classes (groupings column) one of which has 10% missing data and the other has 20%

Usage

```
mnar_two_class_1000
```

Format

`mnar_two_class_1000`:
Test data

---

multi_class_negative *A multi-class dataset with equal random samples in each class*

Description

A multi-class dataset with equal random samples in each class

Usage

```
multi_class_negative
```

Format

`multi_class_negative`:
Test data

one_class_test_100 *A single-class dataset with 100 items of random data*

Description

columns contain a set of random data of different types e.g. uniform continuous, normal, binomial, multinomial.

Usage

```
one_class_test_100
```

Format

`one_class_test_100`:
Test data

one_class_test_1000 *A single-class dataset with 1000 items of random data*

Description

columns contain a set of random data of different types e.g. uniform continuous, normal, binomial, multinomial.

Usage

```
one_class_test_1000
```

Format

`one_class_test_1000`:
Test data

`remove_missing` *Remove variables that fail a missing data test from models*

Description

Comparing missingness by looking at a table is good but we also want to update models to exclude missing data from the predictors.

Usage

```
remove_missing(
  df,
  ...,
  label_fn = label_extractor(df),
  significance_limit = 0.05,
  missingness_limit = 0.1
)
```

Arguments

`df`	a dataframe of individual observations. If using the `tidyselect` syntax data grouping defines the intervention group and should be present. if the formula interface is used the first variable in the right hand side of the formula is used as the intervention, in which case grouping is ignored.
`...`	a list of formulae that specify the models that we want to check
`label_fn`	(optional) a function for mapping a co-variate column name to printable label. This is by default a no-operation and the output table will contain the dataframe column names as labels. A simple alternative would be some form of dplyr::case_when lookup, or a string function such as stringr::str_to_sentence. (N.b. this function must be vectorised). Any value provided here will be overridden by the `options("huxtableone.labeller" = my_label_fn)` which allows global setting of the labeller.
`significance_limit`	the limit at which we reject the hypothesis that the data is missing at random.
`missingness_limit`	the limit at which too much data is missing to include the predictor.

Value

a list of formulae with missing parameters removed

Examples

```
df = iris %>%
  dplyr::mutate(Petal.Width = ifelse(
    stats::runif(dplyr::n()) < dplyr::case_when(
      Species == "setosa" ~ 0.2,
```

```
      Species == "virginica" ~ 0.1,
      TRUE~0
    ),
    NA,
    Petal.Width
  ))
remove_missing(df, ~ Species + Petal.Width + Sepal.Width, ~ Species + Petal.Length + Sepal.Length)
```

`set_labels` *Set a label attribute*

Description

Set a label attribute

Usage

```
set_labels(df, labels, attribute = "label")
```

Arguments

`df`	a dataframe
`labels`	a vector of labels, one for each column
`attribute`	the name of the label attribute (defaults to `"label"`)

Value

the same dataframe with each column labelled

Examples

```
iris = set_labels(iris,
  c("Sepal Length", "Sepal Width",
    "Petal Length", "Petal Width",  "Species"
   ))
fn = label_extractor(iris,tolower)
fn(colnames(iris))
```

set_units *Title*

Description

Title

Usage

```
set_units(df, ..., units)
```

Arguments

df	a dataframe
...	a tidyselect specification or a formula
units	a list of unit as strings which must be either 1 or the same length as the columns matched by the tidyselect.

Value

the dataframe with the unit attribute updated

Examples

```
iris = iris %>% set_units(-Species, units="mm")
iris %>% extract_units()
```

test_cols *A list of columns for a test case*

Description

A list of columns for a test case

Usage

```
test_cols
```

Format

test_cols:
Test data

two_class_test *A two-class dataset with random data*

Description

columns contain a set of random data of different types e.g. uniform continuous, normal, binomial, multinomial. in grouping 1 there is 100 items in grouping 2 there are 1000 items

Usage

```
two_class_test
```

Format

one_class_test_100:
Test data

Index