# Package 'rtables'

October 6, 2021

**Title** Reporting Tables

**Date** 2021-10-06

**Version** 0.4.0

**Description** Reporting tables often have structure that goes beyond simple rectangular data. The 'rtables' package provides a framework for declaring complex multi-level tabulations and then applying them to data. This framework models both tabulation and the resulting tables as hierarchical, tree-like objects which support sibling sub-tables, arbitrary splitting or grouping of data in row and column dimensions, cells containing multiple values, and the concept of contextual summary computations. A convenient pipeable interface is provided for declaring table layouts and the corresponding computations, and then applying them to data.

**Depends** methods, magrittr, R (>= 2.10)

**Imports** stats, htmltools, grid

**Suggests** dplyr, tibble, tidyr, testthat, xml2, knitr, rmarkdown, flextable, officer

**License** Apache License 2.0 | file LICENSE

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.1.1

**VignetteBuilder** knitr

**URL** https://github.com/roche/rtables, https://roche.github.io/rtables/

**BugReports** https://github.com/roche/rtables/issues

**Collate** '00tabletrees.R' 'Viewer.R' 'argument_conventions.R' 'as_html.R' 'utils.R' 'colby_constructors.R' 'compare_rtables.R' 'deprecated.R' 'format_rcell.R' 'indent.R' 'make_subset_expr.R' 'prune.R' 'rtable.R' 'simple_analysis.R' 'split_funs.R' 'summary.R' 'tree_accessors.R' 'tt_afun_utils.R' 'tt_compare_tables.R' 'tt_compatibility.R' 'tt_deprecated.R' 'tt_dotabulation.R' 'tt_paginate.R' 'tt_pos_and_access.R' 'tt_showmethods.R' 'tt_sort.R' 'tt_test_afuns.R' 'tt_toString.R' 'tt_export.R' 'labels.R' 'index_footnotes.R' 'tt_from_df.R' 'zzz_constants.R'

**NeedsCompilation** no

**Author** Gabriel Becker [aut, cre],
Adrian Waddell [aut],
Daniel Sabanés Bové [ctb],
Maximilian Mordig [ctb]

**Maintainer** Gabriel Becker <gabembecker@gmail.com>

**Repository** CRAN

**Date/Publication** 2021-10-06 21:50:02 UTC

# R topics documented:

---

`add_colcounts` *Add the column population counts to the header*

---

## Description

Add the data derived column counts.

## Usage

```
add_colcounts(lyt, format = "(N=xx)")
```

## Arguments

| | |
|---|---|
| `lyt` | layout object pre-data used for tabulation |
| `format` | FormatSpec. Format associated with this split. Formats can be declared via strings (`"xx.x"`) or function. In cases such as `analyze` calls, they can character vectors or lists of functions. |

### Details

It is often the case that the the column counts derived from the input data to `build_table` is not representative of the population counts. For example, if events are counted in the table and the header should display the number of subjects and not the total number of events. In that case use the `col_count` argument in `build_table` to control the counts displayed in the table header.

### Value

A `PreDataTableLayouts` object suitable for passing to further layouting functions, and to `build_table`.

### Author(s)

Gabriel Becker

### Examples

```
l <- basic_table() %>% split_cols_by("ARM") %>%
    add_colcounts() %>%
    split_rows_by("RACE", split_fun = drop_split_levels) %>%
    analyze("AGE", afun = function(x) list(min = min(x), max = max(x)))
l

build_table(l, DM)
```

---

## `add_existing_table` *Add an already calculated table to the layout*

---

### Description

Add an already calculated table to the layout

### Usage

```
add_existing_table(lyt, tt, indent_mod = 0)
```

### Arguments

| | |
|---|---|
| `lyt` | layout object pre-data used for tabulation |
| `tt` | TableTree (or related class). A TableTree object representing a populated table. |
| `indent_mod` | numeric. Modifier for the default indent position for the structure created by this function(subtable, content table, or row) *and all of that structure's children*. Defaults to 0, which corresponds to the unmodified default behavior. |

### Value

A `PreDataTableLayouts` object suitable for passing to further layouting functions, and to `build_table`.

## Author(s)

Gabriel Becker

## Examples

```
tbl1 <- basic_table() %>%
   split_cols_by("ARM") %>%
   analyze("AGE", afun = mean, format = "xx.xx") %>%
   build_table(DM)

tbl1

tbl2 <- basic_table() %>% split_cols_by("ARM") %>%
   analyze("AGE", afun = sd, format = "xx.xx") %>%
   add_existing_table(tbl1) %>%
   build_table(DM)

tbl2

table_structure(tbl2)

row_paths_summary(tbl2)
```

---

# add_overall_col *Add Overall Column*

---

## Description

This function will *only* add an overall column at the *top* level of splitting, NOT within existing column splits. See add_overall_level for the recommended way to add overall columns more generally within existing splits.

## Usage

```
add_overall_col(lyt, label)
```

## Arguments

| | |
|---|---|
| lyt | layout object pre-data used for tabulation |
| label | character(1). A label (not to be confused with the name) for the object/structure. |

## Value

A PreDataTableLayouts object suitable for passing to further layouting functions, and to build_table.

## See Also

add_overall_level

## Examples

```
l <- basic_table() %>%
   split_cols_by("ARM") %>%
   add_overall_col("All Patients") %>%
   analyze("AGE")

l

build_table(l, DM)
```

---

`add_overall_level` *Add an virtual 'overall' level to split*

---

## Description

Add an virtual 'overall' level to split

## Usage

```
add_overall_level(
  valname = "Overall",
  label = valname,
  extra_args = list(),
  first = TRUE,
  trim = FALSE
)
```

## Arguments

| | |
|---|---|
| `valname` | character(1). 'Value' to be assigned to the implicit all-observations split level. Defaults to `"Overall"` |
| `label` | character(1). A label (not to be confused with the name) for the object/structure. |
| `extra_args` | list. Extra arguments to be passed to the tabulation function. Element position in thte list corresponds to the children of this split. Named elements in the child-specific lists are ignored if they do not match a formal argument of the ttabulation function. |
| `first` | logical(1). Should the implicit level appear first (`TRUE`) or last `FALSE`. Defaults to `TRUE`. |
| `trim` | logical(1). Should splits corresponding with 0 observations be kept when tabulating. |

## Value

a closure suitable for use as a splitting function (`splfun`) when creating a table layout

## Examples

```
l <- basic_table() %>%
   split_cols_by("ARM", split_fun = add_overall_level("All Patients", first = FALSE)) %>%
    analyze("AGE")

build_table(l, DM)


l <- basic_table() %>%
   split_cols_by("ARM") %>%
   split_rows_by("RACE", split_fun = add_overall_level("All Ethnicities")) %>%
   summarize_row_groups(label_fstr = "%s (n)") %>%
   analyze("AGE")

l

build_table(l, DM)
```

---

`all_zero_or_na` *Trimming and Pruning Criteria*

---

## Description

Criteria functions (and constructors thereof) for trimming and pruning tables.

## Usage

```
all_zero_or_na(tr)

all_zero(tr)

content_all_zeros_nas(tt, criteria = all_zero_or_na)

prune_empty_level(tt)

prune_zeros_only(tt)

low_obs_pruner(min, type = c("sum", "mean"))
```

## Arguments

| | |
|---|---|
| `tr` | TableRow (or related class). A TableRow object representing a single row within a populated table. |
| `tt` | TableTree (or related class). A TableTree object representing a populated table. |
| `criteria` | function. Function which takes a TableRow object and returns `TRUE` if that row should be removed. Defaults to `all_zero_or_na` |

| | |
|---|---|
| min | numeric(1). (lob_obs_pruner only). Miminum aggregate count value. Subtables whose combined/average count are below this threshhold will be pruned |
| type | character(1). How count values should be aggregated. Must be "sum" (the default) or "mean" |

## Details

all_zero_or_na returns TRUE (and thus indicates trimming/pruning) for any *non-LabelRow* TableRow which contain only any mix of NA (including NaN), 0, Inf and -Inf values.

all_zero returns TRUE for any non-Label row which contains only (non-missing) zero values.

content_all_zeros_nas Prunes a subtable if a) it has a content table with exactly one row in it, and b) all_zero_or_na returns TRUE for that single content row. In practice, when the default summary/content function was used, this represents pruning any subtable which corresponds to an empty set of the input data (e.g., because a factor variable was used in split_rows_by but not all levels were present in the data).

prune_empty_level combines all_zero_or_na behavior for TableRow objects, content_all_zeros_nas on content_table(tt) for TableTree objects, and an addition check that returns TRUE if the tt has no children.

prune_zeros_only behaves as prune_empty_levels does, except that like all_zero it prunes only in the case of all non-missing zero values.

lob_obs_pruner is a *constructor function* which, when called, returns a pruning criteria function which will prune on content rows by comparing sum or mean (dictated by type)of the count count portions of the cell values (defined as the first value per cell regardless of how many values per cell there are) against min.

## Value

A logical value indicating whether tr should be included (TRUE) or pruned (FALSE) during pruning.

## See Also

prune_table(), trim_rows()

---

# analyze *Generate Rows Analyzing Variables Across Columns*

---

## Description

Adding /analyzed variables/ to our table layout defines the primary tabulation to be performed. We do this by adding calls to analyze and/or analyze_colvars into our layout pipeline. As with adding further splitting, the tabulation will occur at the current/next level of nesting by default.

## Usage

```
analyze(
  lyt,
  vars,
  afun = simple_analysis,
  var_labels = vars,
  table_names = vars,
  format = NULL,
  nested = TRUE,
  inclNAs = FALSE,
  extra_args = list(),
  show_labels = c("default", "visible", "hidden"),
  indent_mod = 0L
)
```

## Arguments

| | |
|---|---|
| lyt | layout object pre-data used for tabulation |
| vars | character vector. Multiple variable names. |
| afun | function. Analysis function, must take x or df as its first parameter. Can optionally take other parameters which will be populated by the tabulation framework. See Details in analyze. |
| var_labels | character. Variable labels for 1 or more variables |
| table_names | character. Names for the tables representing each atomic analysis. Defaults to var. |
| format | FormatSpec. Format associated with this split. Formats can be declared via strings ("xx.x") or function. In cases such as analyze calls, they can character vectors or lists of functions. |
| nested | boolean, Add this as a new top-level split (defining a new subtable directly under root). Defaults to FALSE |
| inclNAs | boolean. Should observations with NA in the var variable(s) be included when performing this analysis. Defaults to FALSE |
| extra_args | list. Extra arguments to be passed to the tabulation function. Element position in thte list corresponds to the children of this split. Named elements in the child-specific lists are ignored if they do not match a formal argument of the ttabulation function. |
| show_labels | character(1). Should the variable labels for corresponding to the variable(s) in vars be visible in the resulting table. |
| indent_mod | numeric. Modifier for the default indent position for the structure created by this function(subtable, content table, or row) *and all of that structure's children*. Defaults to 0, which corresponds to the unmodified default behavior. |

## Details

When non-NULL format is used to specify formats for all generated rows, and can be a character vector, a function, or a list of functions. It will be repped out to the number of rows once this is

known during the tabulation process, but will be overridden by formats specified within `rcell` calls in `afun`.

The analysis function (`afun`) should take as its first parameter either `x` or `df`. Which of these the function accepts changes the behavior when tabulation is performed.

- If `afun`'s first parameter is x, it will receive the corresponding subset *vector* of data from the relevant column (from `var` here) of the raw data being used to build the table.
- If `afun`'s first parameter is `df`, it will receive the corresponding subset *data.frame* (i.e. all columns) of the raw data being tabulated

In addition to differentiation on the first argument, the analysis function can optionally accept a number of other parameters which, *if and only if* present in the formals will be passed to the function by the tabulation machinery. These are as follows:

**.N_col** column-wise N (column count) for the full column being tabulated within

**.N_total** overall N (all observation count, defined as sum of column counts) for the tabulation

**.N_row** row-wise N (row group count) for the group of observations being analyzed (ie with no column-based subsetting)

**.df_row** data.frame for observations in the row group being analyzed (ie with no column-based subsetting)

**.var** variable that is analyzed

**.ref_group** data.frame or vector of subset corresponding to the `ref_group` column including subsetting defined by row-splitting. Optional and only required/meaningful if a `ref_group` column has been defined

**.ref_full** data.frame or vector of subset corresponding to the `ref_group` column without subsetting defined by row-splitting. Optional and only required/meaningful if a `ref_group` column has been defined

**.in_ref_col** boolean indicates if calculation is done for cells withing the reference column

**.spl_context** data.frame, each row gives information about a previous/'ancestor' split state. see below

## Value

A `PreDataTableLayouts` object suitable for passing to further layouting functions, and to `build_table`.

## .spl_context Details

The `.spl_context` `data.frame` gives information about the subsets of data corresponding to the splits within-which the current `analyze` action is nested. Taken together, these correspond to the path that the resulting (set of) rows the analysis function is creating, although the information is in a slighlyt different form. Each split (which correspond to groups of rows in the resulting table) is represented via the following columns:

**split** The name of the split (often the variable being split in the simple case)

**value** The string representation of the value at that split

**full_parent_df** a dataframe containing the full data (ie across all columns) corresponding to the path defined by the combination of `split` and `value` of this row *and all rows above this row*

**all_cols_n** the number of observations corresponding to this row grouping (union of all columns)

***(row-split and analyze contexts only)* <1 column for each column in the table structure** These list columns (named the same as `names(col_exprs(tab))`) contain logical vectors corresponding to the subset of this row's `full_parent_df` corresponding to that column

**cur_col_subset** List column containing logical vectors indicating the subset of that row's `full_parent_df` for the column currently being created by the analysis function

**cur_col_n** integer column containing the observation counts for that split

*note Within analysis functions that accept* `.spl_context`*, the* `all_cols_n` *and* `cur_col_n` *columns of the dataframe will contain the 'true' observation counts corresponding to the row-group and row-group x column subsets of the data. These numbers will not, and currently cannot, reflect alternate column observation counts provided by the* `alt_counts_df`, `col_counts` *or* `col_total` *arguments to* `build_table`

## Note

None of the arguments described in the Details section can be overridden via extra_args or when calling `make_afun`. `.N_col` and `.N_total` can be overridden via the `col_counts` argument to `build_table`. Alternative values for the others must be calculated within `afun` based on a combination of extra arguments and the unmodified values provided by the tabulation framework.

## Author(s)

Gabriel Becker

## Examples

```
l <- basic_table() %>%
    split_cols_by("ARM") %>%
    analyze("AGE", afun = list_wrap_x(summary) , format = "xx.xx")
l
build_table(l, DM)


l <- basic_table() %>%
    split_cols_by("Species") %>%
    analyze(head(names(iris), -1), afun = function(x) {
        list(
            "mean / sd" = rcell(c(mean(x), sd(x)), format = "xx.xx (xx.xx)"),
            "range" = rcell(diff(range(x)), format = "xx.xx")
        )
    })
l
build_table(l, iris)
```

AnalyzeVarSplit *Define a subset tabulation/analysis*

## Description

Define a subset tabulation/analysis

Define a subset tabulation/analysis

## Usage

```
AnalyzeVarSplit(
  var,
  split_label = var,
  afun,
  defrowlab = "",
  cfun = NULL,
  cformat = NULL,
  split_format = NULL,
  inclNAs = FALSE,
  split_name = var,
  extra_args = list(),
  indent_mod = 0L,
  label_pos = "default",
  cvar = ""
)

AnalyzeColVarSplit(
  afun,
  defrowlab = "",
  cfun = NULL,
  cformat = NULL,
  split_format = NULL,
  inclNAs = FALSE,
  split_name = "",
  extra_args = list(),
  indent_mod = 0L,
  label_pos = "default",
  cvar = ""
)

AnalyzeMultiVars(
  var,
  split_label = "",
  afun,
  defrowlab = "",
  cfun = NULL,
```

```
  cformat = NULL,
  split_format = NULL,
  inclNAs = FALSE,
  .payload = NULL,
  split_name = NULL,
  extra_args = list(),
  indent_mod = 0L,
  child_labels = c("default", "topleft", "visible", "hidden"),
  child_names = var,
  cvar = ""
)
```

## Arguments

| | |
|---|---|
| var | string, variable name |
| split_label | string. Label string to be associated with the table generated by the split. Not to be confused with labels assigned to each child (which are based on the data and type of split during tabulation). |
| afun | function. Analysis function, must take x or df as its first parameter. Can optionally take other parameters which will be populated by the tabulation framework. See Details in analyze. |
| defrowlab | character. Default row labels if they are not specified by the return value of afun |
| cfun | list/function/NULL. tabulation function(s) for creating content rows. Must accept x or df as first parameter. Must accept labelstr as the second argument. Can optionally accept all optional arguments accepted by analysis functions. See analyze. |
| cformat | format spec. Format for content rows |
| split_format | FormatSpec. Default format associated with the split being created. |
| inclNAs | boolean. Should observations with NA in the var variable(s) be included when performing this analysis. Defaults to FALSE |
| split_name | string. Name associiated with this split (for pathing, etc) |
| extra_args | list. Extra arguments to be passed to the tabulation function. Element position in thte list corresponds to the children of this split. Named elements in the child-specific lists are ignored if they do not match a formal argument of the ttabulation function. |
| indent_mod | numeric. Modifier for the default indent position for the structure created by this function(subtable, content table, or row) *and all of that structure's children*. Defaults to 0, which corresponds to the unmodified default behavior. |
| label_pos | character(1). Location the variable label should be displayed, Accepts hidden (default for non-analyze row splits), visible, topleft, and - for analyze splits only - default. For analyze calls, default indicates that the variable should be visible if and only if multiple variables are analyzed at the same level of nesting. |
| cvar | character(1). The variable, if any, which the content function should accept. Defaults to NA. |
| .payload | Used internally, not intended to be set by end users. |

| | |
|---|---|
| child_labels | string. One of "default", "visible", "hidden". What should the display behavior be for the labels (ie label rows) of the children of this split. Defaults to "default" which flags the label row as visible only if the child has 0 content rows. |
| child_names | character. Names to be given to the sub splits contained by a compound split (typically a AnalyzeMultiVars split object). |

## Value

An AnalyzeVarSplit object.

An AnalyzeMultiVars split object.

## Author(s)

Gabriel Becker

Gabriel Becker

---

# analyze_against_ref_group

*Add ref_group comparison analysis recipe*

---

## Description

Add ref_group comparison analysis recipe

## Usage

```
analyze_against_ref_group(
  lyt,
  var = NA_character_,
  afun,
  label = if (is.na(var)) "" else var,
  compfun = `-`,
  format = NULL,
  nested = TRUE,
  indent_mod = 0L,
  show_labels = c("default", "hidden", "visible")
)
```

## Arguments

| | |
|---|---|
| lyt | layout object pre-data used for tabulation |
| var | string, variable name |
| afun | function. Analysis function, must take x or df as its first parameter. Can optionally take other parameters which will be populated by the tabulation framework. See Details in analyze. |

| | |
|---|---|
| label | character(1). A label (not to be confused with the name) for the object/structure. |
| compfun | function/string. The comparison function which accepts the analysis function outputs for two different partitions and returns a single value. Defaults to subraction. If a string, taken as the name of a function. |
| format | FormatSpec. Format associated with this split. Formats can be declared via strings ("xx.x") or function. In cases such as analyze calls, they can character vectors or lists of functions. |
| nested | boolean, Add this as a new top-level split (defining a new subtable directly under root). Defaults to FALSE |
| indent_mod | numeric. Modifier for the default indent position for the structure created by this function(subtable, content table, or row) *and all of that structure's children*. Defaults to 0, which corresponds to the unmodified default behavior. |
| show_labels | character(1). Should the variable labels for corresponding to the variable(s) in vars be visible in the resulting table. |

## Details

Please see the baseline vignette for more details.

## Value

A PreDataTableLayouts object suitable for passing to further layouting functions, and to build_table.

## Author(s)

Gabriel Becker

## Examples

```
basic_table() %>%
    split_cols_by("ARM", ref_group = "B: Placebo") %>%
    analyze("AGE", afun = function(x, .ref_group) {
        in_rows(
           "Difference of Averages" = rcell(mean(x) - mean(.ref_group), format = "xx.xx")
        )
    }) %>%
    build_table(DM)
```

---

# analyze_colvars *Generate Rows Analyzing Different Variables Across Columns*

---

## Description

Generate Rows Analyzing Different Variables Across Columns

## Usage

```
analyze_colvars(
  lyt,
  afun,
  format = NULL,
  nested = TRUE,
  extra_args = list(),
  indent_mod = 0L,
  inclNAs = FALSE
)
```

## Arguments

| | |
|---|---|
| lyt | layout object pre-data used for tabulation |
| afun | function or list. Function(s) to be used to calculate the values in each column. the list will be repped out as needed and matched by position with the columns during tabulation. |
| format | FormatSpec. Format associated with this split. Formats can be declared via strings ("xx.x") or function. In cases such as analyze calls, they can character vectors or lists of functions. |
| nested | boolean, Add this as a new top-level split (defining a new subtable directly under root). Defaults to FALSE |
| extra_args | list. Extra arguments to be passed to the tabulation function. Element position in thte list corresponds to the children of this split. Named elements in the child-specific lists are ignored if they do not match a formal argument of the ttabulation function. |
| indent_mod | numeric. Modifier for the default indent position for the structure created by this function(subtable, content table, or row) *and all of that structure's children*. Defaults to 0, which corresponds to the unmodified default behavior. |
| inclNAs | boolean. Should observations with NA in the var variable(s) be included when performing this analysis. Defaults to FALSE |

## Value

A PreDataTableLayouts object suitable for passing to further layouting functions, and to build_table.

## Author(s)

Gabriel Becker

## See Also

split_cols_by_multivar

## Examples

```
library(dplyr)
ANL <- DM %>% mutate(value = rnorm(n()), pctdiff = runif(n()))

## toy example where we take the mean of the first variable and the
## count of >.5 for the second.
colfuns <- list(function(x) rcell(mean(x), format = "xx.x"),
                function(x) rcell(sum(x > .5), format = "xx"))

l <- basic_table() %>%
    split_cols_by("ARM") %>%
    split_cols_by_multivar(c("value", "pctdiff")) %>%
    split_rows_by("RACE", split_label = "ethnicity", split_fun = drop_split_levels) %>%
    summarize_row_groups() %>%
    analyze_colvars(afun = colfuns)

l

build_table(l, ANL)


basic_table() %>% split_cols_by("ARM") %>%
   split_cols_by_multivar(c("value", "pctdiff"), varlabels = c("Measurement", "Pct Diff")) %>%
    split_rows_by("RACE", split_label = "ethnicity", split_fun = drop_split_levels) %>%
    summarize_row_groups() %>%
    analyze_colvars(afun = mean, format = "xx.xx") %>%
    build_table(ANL)
```

---

# append_topleft *Append a description to the 'top-left' materials for the layout*

---

## Description

This function *adds* `newlines` to the current set of "top-left materials".

## Usage

```
append_topleft(lyt, newlines)
```

## Arguments

| | |
|---|---|
| `lyt` | layout object pre-data used for tabulation |
| `newlines` | character. The new line(s) to be added to the materials |

## Details

Adds `newlines` to the set of strings representing the 'top-left' materials declared in the layout (the content displayed to the left of the column labels when the resulting tables are printed).

Top-left material strings are stored and then displayed *exactly as is*, no structure or indenting is applied to them either wheyn they are added or when they are displayed.

## Value

A `PreDataTableLayouts` object suitable for passing to further layouting functions, and to `build_table`.

## Note

Currently, where in the construction of the layout this is called makes no difference, as it is indepenedent of the actual splitting keywords. This may change in the future.

This function is experimental, its name and the details of its behavior are subject to change in future versions.

## See Also

top_left

## Examples

```
library(dplyr)

lyt <- basic_table() %>%
  split_cols_by("ARM") %>%
  split_cols_by("SEX") %>%
  split_rows_by("RACE") %>%
  append_topleft("Ethnicity") %>%
  analyze("AGE") %>%
  append_topleft("  Age")

DM2 <- DM %>% mutate(RACE = factor(RACE), SEX = factor(SEX))

build_table(lyt, DM2)
```

---

# as.vector,TableRow-method

*convert to a vector*

---

## Description

convert to a vector

## Usage

```
## S4 method for signature 'TableRow'
as.vector(x, mode = "any")

## S4 method for signature 'ElementaryTable'
as.vector(x, mode = "any")

## S4 method for signature 'VTableTree'
as.vector(x, mode = "any")
```

## Arguments

| | |
|---|---|
| x | ANY. The object to be converted to a vector |
| mode | character(1). Passed on to `as.vector` |

## Value

a vector of the chosen mode (or an error is raised if more than one row was present).

## Note

This only works for a table with a single row or a row object.

---

# as_html *Convert an* `rtable` *object to a* `shiny.tag` *html object*

---

## Description

The returned `html` object can be immediately used in shiny and rmarkdown.

## Usage

```
as_html(
  x,
  width = NULL,
  class_table = "table table-condensed table-hover",
  class_tr = "",
  class_td = "",
  class_th = "",
  link_label = NULL
)
```

### Arguments

| | |
|---|---|
| x | rtable object |
| width | width |
| class_table | class for table tag |
| class_tr | class for tr tag |
| class_td | class for td tag |
| class_th | class for th tag |
| link_label | link anchor label (not including tab: prefix) for the table. |

### Value

A shiny.tag object representing x in HTML.

### Examples

```
tbl <- rtable(
  header = LETTERS[1:3],
  format = "xx",
  rrow("r1", 1,2,3),
  rrow("r2", 4,3,2, indent = 1),
  rrow("r3", indent = 2)
)

as_html(tbl)

as_html(tbl, class_table = "table", class_tr = "row")

as_html(tbl, class_td = "aaa")

## Not run:
Viewer(tbl)

## End(Not run)
```

---

## basic_table *Layout with 1 column and zero rows*

---

### Description

Every layout must start with a basic table.

### Usage

```
basic_table(
  title = "",
  subtitles = character(),
  main_footer = character(),
  prov_footer = character(),
  show_colcounts = FALSE
)
```

### Arguments

| | |
|---|---|
| title | character(1). Main title. Ignored for subtables. |
| subtitles | character. Subtitles. Ignored for subtables. |
| main_footer | character. Main global (non-referential) footer materials. |
| prov_footer | character. Provenance-related global footer materials. Generally should not be modified by hand. |
| show_colcounts | logical(1). Should column counts be displayed in the resulting table when this layout is applied to data |

### Value

A PreDataTableLayouts object suitable for passing to further layouting functions, and to build_table.

### Examples

```
lyt <- basic_table() %>%
  analyze("AGE", afun = mean)

build_table(lyt, DM)
```

---

## build_table *Create a table from a layout and data*

---

### Description

Layouts are used to describe a table pre-data. build_rable is used to create a table using a layout and a dataset.

## Usage

```
build_table(
  lyt,
  df,
  alt_counts_df = NULL,
  col_counts = NULL,
  col_total = if (is.null(alt_counts_df)) nrow(df) else nrow(alt_counts_df),
  topleft = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| lyt | layout object pre-data used for tabulation |
| df | dataset (data.frame or tibble) |
| alt_counts_df | dataset (data.frame or tibble). Alternative full data the rtables framework will use (*only*) when calculating column counts. |
| col_counts | numeric (or NULL). Deprecated. If non-null, column counts which override those calculated automatically during tabulation. Must specify "counts" for *all* resulting columns if non-NULL. NA elements will be replaced with the automatically calculated counts. |
| col_total | integer(1). The total observations across all columns. Defaults to nrow(df). |
| topleft | character. Override values for the "top left" material to be displayed during printing. |
| ... | currently ignored. |

## Details

When alt_counts_df is specified, column counts are calculated by applying the exact column subsetting expressions determined when applying column splitting to the main data (df) to alt_counts_df and counting the observations in each resulting subset.

In particular, this means that in the case of splitting based on cuts of the data, any dynamic cuts will have been calculated based on df and simply re-used for the count calculation.

## Value

A TableTree or ElementaryTable object representing the table created by performing the tabulations declared in lyt to the data df.

## Note

When overriding the column counts or totals care must be taken that, e.g., length() or nrow() are not called within tabulation functions, because those will NOT give the overridden counts. Writing/using tabulation functions which accept .N_col and .N_total or do not rely on column counts at all (even implicitly) is the only way to ensure overriden counts are fully respected.

## Author(s)

Gabriel Becker

## Examples

```
l <- basic_table() %>%
  split_cols_by("Species") %>%
  analyze("Sepal.Length", afun = function(x) {
  list(
    "mean (sd)" = rcell(c(mean(x), sd(x)), format = "xx.xx (xx.xx)"),
    "range" = diff(range(x))
  )
})

l

build_table(l, iris)

# analyze multiple variables
l <- basic_table() %>%
  split_cols_by("Species") %>%
  analyze(c("Sepal.Length", "Petal.Width"), afun = function(x) {
  list(
    "mean (sd)" = rcell(c(mean(x), sd(x)), format = "xx.xx (xx.xx)"),
    "range" = diff(range(x))
  )
})

build_table(l, iris)

# an example more relevant for clinical trials
l <- basic_table() %>%
    split_cols_by("ARM") %>%
    analyze("AGE", afun = function(x) {
    setNames(as.list(fivenum(x)), c("minimum", "lower-hinge", "median", "upper-hinge", "maximum"))
    })

build_table(l, DM)

build_table(l, subset(DM, AGE > 40))

# with column counts
l2 <- l %>%
  add_colcounts()
build_table(l2, DM)


# with column counts calculated based on different data
miniDM <- DM[sample(1:NROW(DM), 100),]
build_table(l2, DM, alt_counts_df = miniDM)
```

```
build_table(l, DM, col_counts = 1:3)
```

---

`c,SplitVector-method` *combine SplitVector objects*

---

## Description

These are internal methods that are documented only to satisfy R CMD check. End users should pay no attention to this documentation.

## Usage

```
## S4 method for signature 'SplitVector'
c(x, ...)

split_rows(lyt = NULL, spl, pos, cmpnd_fun = AnalyzeMultiVars)

## S4 method for signature '`NULL`'
split_rows(lyt = NULL, spl, pos, cmpnd_fun = AnalyzeMultiVars)

## S4 method for signature 'PreDataRowLayout'
split_rows(lyt = NULL, spl, pos, cmpnd_fun = AnalyzeMultiVars)

## S4 method for signature 'SplitVector'
split_rows(lyt = NULL, spl, pos, cmpnd_fun = AnalyzeMultiVars)

## S4 method for signature 'PreDataTableLayouts'
split_rows(lyt, spl, pos)

## S4 method for signature 'ANY'
split_rows(lyt, spl, pos)

cmpnd_last_rowsplit(lyt, spl, constructor)

## S4 method for signature '`NULL`'
cmpnd_last_rowsplit(lyt, spl, constructor)

## S4 method for signature 'PreDataRowLayout'
cmpnd_last_rowsplit(lyt, spl, constructor)

## S4 method for signature 'SplitVector'
cmpnd_last_rowsplit(lyt, spl, constructor)

## S4 method for signature 'PreDataTableLayouts'
cmpnd_last_rowsplit(lyt, spl, constructor)

## S4 method for signature 'ANY'
```

```
cmpnd_last_rowsplit(lyt, spl, constructor)

split_cols(lyt = NULL, spl, pos)

## S4 method for signature '`NULL`'
split_cols(lyt = NULL, spl, pos)

## S4 method for signature 'PreDataColLayout'
split_cols(lyt = NULL, spl, pos)

## S4 method for signature 'SplitVector'
split_cols(lyt = NULL, spl, pos)

## S4 method for signature 'PreDataTableLayouts'
split_cols(lyt = NULL, spl, pos)

## S4 method for signature 'ANY'
split_cols(lyt = NULL, spl, pos)

cmpnd_last_colsplit(lyt, spl, constructor)

## S4 method for signature '`NULL`'
cmpnd_last_colsplit(lyt, spl, constructor)

## S4 method for signature 'PreDataColLayout'
cmpnd_last_colsplit(lyt, spl, constructor)

## S4 method for signature 'SplitVector'
cmpnd_last_colsplit(lyt, spl, constructor)

## S4 method for signature 'PreDataTableLayouts'
cmpnd_last_colsplit(lyt, spl, constructor)

## S4 method for signature 'ANY'
cmpnd_last_colsplit(lyt, spl, constructor)

.add_row_summary(
  lyt,
  label,
  cfun,
  child_labels = c("default", "visible", "hidden"),
  cformat = NULL,
  indent_mod = 0L,
  cvar = "",
  extra_args = list()
)

## S4 method for signature 'PreDataTableLayouts'
```

```
.add_row_summary(
  lyt,
  label,
  cfun,
  child_labels = c("default", "visible", "hidden"),
  cformat = NULL,
  indent_mod = 0L,
  cvar = "",
  extra_args = list()
)

## S4 method for signature 'PreDataRowLayout'
.add_row_summary(
  lyt,
  label,
  cfun,
  child_labels = c("default", "visible", "hidden"),
  cformat = NULL,
  indent_mod = 0L,
  cvar = "",
  extra_args = list()
)

## S4 method for signature 'SplitVector'
.add_row_summary(
  lyt,
  label,
  cfun,
  child_labels = c("default", "visible", "hidden"),
  cformat = NULL,
  indent_mod = 0L,
  cvar = "",
  extra_args = list()
)

## S4 method for signature 'Split'
.add_row_summary(
  lyt,
  label,
  cfun,
  child_labels = c("default", "visible", "hidden"),
  cformat = NULL,
  indent_mod = 0L,
  cvar = "",
  extra_args = list()
)

fix_dyncuts(spl, df)
```

```
## S4 method for signature 'Split'
fix_dyncuts(spl, df)

## S4 method for signature 'VarDynCutSplit'
fix_dyncuts(spl, df)

## S4 method for signature 'VTableTree'
fix_dyncuts(spl, df)

## S4 method for signature 'PreDataRowLayout'
fix_dyncuts(spl, df)

## S4 method for signature 'PreDataColLayout'
fix_dyncuts(spl, df)

## S4 method for signature 'SplitVector'
fix_dyncuts(spl, df)

## S4 method for signature 'PreDataTableLayouts'
fix_dyncuts(spl, df)

summarize_rows_inner(obj, depth = 0, indent = 0)

## S4 method for signature 'TableTree'
summarize_rows_inner(obj, depth = 0, indent = 0)

## S4 method for signature 'ElementaryTable'
summarize_rows_inner(obj, depth = 0, indent = 0)

## S4 method for signature 'TableRow'
summarize_rows_inner(obj, depth = 0, indent = 0)

## S4 method for signature 'LabelRow'
summarize_rows_inner(obj, depth = 0, indent = 0)

table_structure_inner(obj, depth = 0, indent = 0, print_indent = 0)

## S4 method for signature 'TableTree'
table_structure_inner(obj, depth = 0, indent = 0, print_indent = 0)

## S4 method for signature 'ElementaryTable'
table_structure_inner(obj, depth = 0, indent = 0, print_indent = 0)

## S4 method for signature 'TableRow'
table_structure_inner(obj, depth = 0, indent = 0, print_indent = 0)

## S4 method for signature 'LabelRow'
```

```
table_structure_inner(obj, depth = 0, indent = 0, print_indent = 0)

next_rpos(obj, nested = TRUE, for_analyze = FALSE)

## S4 method for signature 'PreDataTableLayouts'
next_rpos(obj, nested = TRUE, for_analyze = FALSE)

## S4 method for signature 'PreDataRowLayout'
next_rpos(obj, nested = TRUE, for_analyze = FALSE)

## S4 method for signature 'ANY'
next_rpos(obj, nested)

next_cpos(obj, nested = TRUE)

## S4 method for signature 'PreDataTableLayouts'
next_cpos(obj, nested = TRUE)

## S4 method for signature 'PreDataColLayout'
next_cpos(obj, nested = TRUE)

## S4 method for signature 'ANY'
next_cpos(obj, nested = TRUE)

last_rowsplit(obj)

## S4 method for signature '`NULL`'
last_rowsplit(obj)

## S4 method for signature 'SplitVector'
last_rowsplit(obj)

## S4 method for signature 'PreDataRowLayout'
last_rowsplit(obj)

## S4 method for signature 'PreDataTableLayouts'
last_rowsplit(obj)

rlayout(obj)

## S4 method for signature 'PreDataTableLayouts'
rlayout(obj)

## S4 method for signature 'ANY'
rlayout(obj)

rlayout(object) <- value
```

```
## S4 replacement method for signature 'PreDataTableLayouts'
rlayout(object) <- value

tree_pos(obj)

## S4 method for signature 'VLayoutNode'
tree_pos(obj)

pos_subset(obj)

## S4 method for signature 'TreePos'
pos_subset(obj)

## S4 method for signature 'VLayoutNode'
pos_subset(obj)

pos_splits(obj)

## S4 method for signature 'TreePos'
pos_splits(obj)

## S4 method for signature 'VLayoutNode'
pos_splits(obj)

pos_splvals(obj)

## S4 method for signature 'TreePos'
pos_splvals(obj)

## S4 method for signature 'VLayoutNode'
pos_splvals(obj)

pos_split_labels(obj)

## S4 method for signature 'TreePos'
pos_split_labels(obj)

## S4 method for signature 'VLayoutNode'
pos_split_labels(obj)

split_texttype(obj)

## S4 method for signature 'VarLevelSplit'
split_texttype(obj)

## S4 method for signature 'MultiVarSplit'
split_texttype(obj)
```

```
## S4 method for signature 'AllSplit'
split_texttype(obj)

## S4 method for signature 'RootSplit'
split_texttype(obj)

## S4 method for signature 'NULLSplit'
split_texttype(obj)

## S4 method for signature 'VarStaticCutSplit'
split_texttype(obj)

## S4 method for signature 'VarDynCutSplit'
split_texttype(obj)

## S4 method for signature 'ManualSplit'
split_texttype(obj)

## S4 method for signature 'ANY'
split_texttype(obj)

pos_spltypes(obj)

## S4 method for signature 'TreePos'
pos_spltypes(obj)

## S4 method for signature 'VLayoutNode'
pos_spltypes(obj)

pos_splval_labels(obj)

## S4 method for signature 'TreePos'
pos_splval_labels(obj)

## S4 method for signature 'VLayoutNode'
pos_splval_labels(obj)

spl_payload(obj)

## S4 method for signature 'Split'
spl_payload(obj)

spl_payload(obj) <- value

## S4 replacement method for signature 'Split'
spl_payload(obj) <- value

spl_label_var(obj)
```

```
## S4 method for signature 'VarLevelSplit'
spl_label_var(obj)

## S4 method for signature 'Split'
spl_label_var(obj)

tt_labelrow(obj)

## S4 method for signature 'VTableTree'
tt_labelrow(obj)

tt_labelrow(obj) <- value

## S4 replacement method for signature 'VTableTree'
tt_labelrow(obj) <- value

labelrow_visible(obj)

## S4 method for signature 'VTableTree'
labelrow_visible(obj)

## S4 method for signature 'LabelRow'
labelrow_visible(obj)

## S4 method for signature 'VAnalyzeSplit'
labelrow_visible(obj)

labelrow_visible(obj) <- value

## S4 replacement method for signature 'VTableTree'
labelrow_visible(obj) <- value

## S4 replacement method for signature 'LabelRow'
labelrow_visible(obj) <- value

## S4 replacement method for signature 'VAnalyzeSplit'
labelrow_visible(obj) <- value

label_kids(spl)

## S4 method for signature 'Split'
label_kids(spl)

label_kids(spl) <- value

## S4 replacement method for signature 'Split,character'
label_kids(spl) <- value
```

```
## S4 replacement method for signature 'Split,logical'
label_kids(spl) <- value

vis_label(spl)

## S4 method for signature 'Split'
vis_label(spl)

vis_label(spl) <- value

label_position(spl)

## S4 method for signature 'Split'
label_position(spl)

## S4 method for signature 'VAnalyzeSplit'
label_position(spl)

label_position(spl) <- value

## S4 replacement method for signature 'Split'
label_position(spl) <- value

content_fun(obj)

## S4 method for signature 'Split'
content_fun(obj)

content_fun(object) <- value

## S4 replacement method for signature 'Split'
content_fun(object) <- value

analysis_fun(obj)

## S4 method for signature 'AnalyzeVarSplit'
analysis_fun(obj)

## S4 method for signature 'AnalyzeColVarSplit'
analysis_fun(obj)

analysis_fun(object) <- value

## S4 replacement method for signature 'AnalyzeVarSplit'
analysis_fun(object) <- value

## S4 replacement method for signature 'AnalyzeColVarSplit'
```

```
analysis_fun(object) <- value

split_fun(obj)

## S4 method for signature 'CustomizableSplit'
split_fun(obj)

## S4 method for signature 'Split'
split_fun(obj)

content_extra_args(obj)

## S4 method for signature 'Split'
content_extra_args(obj)

content_extra_args(object) <- value

## S4 replacement method for signature 'Split'
content_extra_args(object) <- value

content_var(obj)

## S4 method for signature 'Split'
content_var(obj)

content_var(object) <- value

## S4 replacement method for signature 'Split'
content_var(object) <- value

avar_inclNAs(obj)

## S4 method for signature 'VAnalyzeSplit'
avar_inclNAs(obj)

avar_inclNAs(obj) <- value

## S4 replacement method for signature 'VAnalyzeSplit'
avar_inclNAs(obj) <- value

spl_labelvar(obj)

## S4 method for signature 'VarLevelSplit'
spl_labelvar(obj)

spl_child_order(obj)

## S4 method for signature 'VarLevelSplit'
```

```
spl_child_order(obj)

spl_child_order(obj) <- value

## S4 replacement method for signature 'VarLevelSplit'
spl_child_order(obj) <- value

## S4 method for signature 'ManualSplit'
spl_child_order(obj)

## S4 method for signature 'MultiVarSplit'
spl_child_order(obj)

## S4 method for signature 'AllSplit'
spl_child_order(obj)

## S4 method for signature 'VarStaticCutSplit'
spl_child_order(obj)

root_spl(obj)

## S4 method for signature 'PreDataAxisLayout'
root_spl(obj)

root_spl(obj) <- value

## S4 replacement method for signature 'PreDataAxisLayout'
root_spl(obj) <- value

spanned_values(obj)

## S4 method for signature 'TableRow'
spanned_values(obj)

## S4 method for signature 'LabelRow'
spanned_values(obj)

spanned_cells(obj)

## S4 method for signature 'TableRow'
spanned_cells(obj)

## S4 method for signature 'LabelRow'
spanned_cells(obj)

spanned_values(obj) <- value

## S4 replacement method for signature 'TableRow'
```

```
spanned_values(obj) <- value

## S4 replacement method for signature 'LabelRow'
spanned_values(obj) <- value

obj_format(obj)

## S4 method for signature 'ANY'
obj_format(obj)

## S4 method for signature 'VTableNodeInfo'
obj_format(obj)

## S4 method for signature 'Split'
obj_format(obj)

obj_format(obj) <- value

## S4 replacement method for signature 'ANY'
obj_format(obj) <- value

## S4 replacement method for signature 'VTableNodeInfo'
obj_format(obj) <- value

## S4 replacement method for signature 'Split'
obj_format(obj) <- value

set_format_recursive(obj, format, override = FALSE)

## S4 method for signature 'TableRow'
set_format_recursive(obj, format, override = FALSE)

## S4 method for signature 'LabelRow'
set_format_recursive(obj, format, override = FALSE)

content_format(obj)

## S4 method for signature 'Split'
content_format(obj)

content_format(obj) <- value

## S4 replacement method for signature 'Split'
content_format(obj) <- value

row_cspans(obj)

## S4 method for signature 'TableRow'
```

```
row_cspans(obj)

## S4 method for signature 'LabelRow'
row_cspans(obj)

row_cspans(obj) <- value

## S4 replacement method for signature 'TableRow'
row_cspans(obj) <- value

## S4 replacement method for signature 'LabelRow'
row_cspans(obj) <- value

cell_cspan(obj)

## S4 method for signature 'CellValue'
cell_cspan(obj)

cell_cspan(obj) <- value

## S4 replacement method for signature 'CellValue'
cell_cspan(obj) <- value

tt_level(obj)

## S4 method for signature 'VNodeInfo'
tt_level(obj)

tt_level(obj) <- value

## S4 replacement method for signature 'VNodeInfo'
tt_level(obj) <- value

## S4 replacement method for signature 'VTableTree'
tt_level(obj) <- value

indent_mod(obj)

## S4 method for signature 'Split'
indent_mod(obj)

## S4 method for signature 'VTableNodeInfo'
indent_mod(obj)

## S4 method for signature 'ANY'
indent_mod(obj)

## S4 method for signature 'RowsVerticalSection'
```

```
indent_mod(obj)

indent_mod(obj) <- value

## S4 replacement method for signature 'Split'
indent_mod(obj) <- value

## S4 replacement method for signature 'VTableNodeInfo'
indent_mod(obj) <- value

content_indent_mod(obj)

## S4 method for signature 'Split'
content_indent_mod(obj)

## S4 method for signature 'VTableNodeInfo'
content_indent_mod(obj)

content_indent_mod(obj) <- value

## S4 replacement method for signature 'Split'
content_indent_mod(obj) <- value

## S4 replacement method for signature 'VTableNodeInfo'
content_indent_mod(obj) <- value

rawvalues(obj)

## S4 method for signature 'ValueWrapper'
rawvalues(obj)

## S4 method for signature 'LevelComboSplitValue'
rawvalues(obj)

## S4 method for signature 'list'
rawvalues(obj)

## S4 method for signature 'ANY'
rawvalues(obj)

## S4 method for signature 'CellValue'
rawvalues(obj)

## S4 method for signature 'TreePos'
rawvalues(obj)

## S4 method for signature 'RowsVerticalSection'
rawvalues(obj)
```

```
value_names(obj)

## S4 method for signature 'ANY'
value_names(obj)

## S4 method for signature 'TreePos'
value_names(obj)

## S4 method for signature 'list'
value_names(obj)

## S4 method for signature 'ValueWrapper'
value_names(obj)

## S4 method for signature 'LevelComboSplitValue'
value_names(obj)

## S4 method for signature 'RowsVerticalSection'
value_names(obj)

value_labels(obj)

## S4 method for signature 'ANY'
value_labels(obj)

## S4 method for signature 'TreePos'
value_labels(obj)

## S4 method for signature 'list'
value_labels(obj)

## S4 method for signature 'RowsVerticalSection'
value_labels(obj)

## S4 method for signature 'ValueWrapper'
value_labels(obj)

## S4 method for signature 'LevelComboSplitValue'
value_labels(obj)

## S4 method for signature 'MultiVarSplit'
value_labels(obj)

spl_varlabels(obj)

## S4 method for signature 'MultiVarSplit'
spl_varlabels(obj)
```

```
spl_varlabels(object) <- value

## S4 replacement method for signature 'MultiVarSplit'
spl_varlabels(object) <- value

splv_extra(obj)

## S4 method for signature 'SplitValue'
splv_extra(obj)

splv_extra(obj) <- value

## S4 replacement method for signature 'SplitValue'
splv_extra(obj) <- value

split_exargs(obj)

## S4 method for signature 'Split'
split_exargs(obj)

split_exargs(obj) <- value

## S4 replacement method for signature 'Split'
split_exargs(obj) <- value

clayout_splits(obj)

## S4 method for signature 'LayoutColTree'
clayout_splits(obj)

## S4 method for signature 'LayoutColLeaf'
clayout_splits(obj)

## S4 method for signature 'VTableNodeInfo'
clayout_splits(obj)

col_extra_args(obj, df = NULL)

## S4 method for signature 'InstantiatedColumnInfo'
col_extra_args(obj, df = NULL)

## S4 method for signature 'PreDataTableLayouts'
col_extra_args(obj, df = NULL)

## S4 method for signature 'PreDataColLayout'
col_extra_args(obj, df = NULL)
```

```
## S4 method for signature 'LayoutColTree'
col_extra_args(obj, df = NULL)

## S4 method for signature 'LayoutColLeaf'
col_extra_args(obj, df = NULL)

disp_ccounts(obj)

## S4 method for signature 'VTableTree'
disp_ccounts(obj)

## S4 method for signature 'InstantiatedColumnInfo'
disp_ccounts(obj)

## S4 method for signature 'PreDataTableLayouts'
disp_ccounts(obj)

## S4 method for signature 'PreDataColLayout'
disp_ccounts(obj)

disp_ccounts(obj) <- value

## S4 replacement method for signature 'VTableTree'
disp_ccounts(obj) <- value

## S4 replacement method for signature 'InstantiatedColumnInfo'
disp_ccounts(obj) <- value

## S4 replacement method for signature 'PreDataColLayout'
disp_ccounts(obj) <- value

## S4 replacement method for signature 'LayoutColTree'
disp_ccounts(obj) <- value

## S4 replacement method for signature 'PreDataTableLayouts'
disp_ccounts(obj) <- value

colcount_format(obj)

## S4 method for signature 'InstantiatedColumnInfo'
colcount_format(obj)

## S4 method for signature 'VTableNodeInfo'
colcount_format(obj)

## S4 method for signature 'PreDataColLayout'
colcount_format(obj)
```

```
## S4 method for signature 'PreDataTableLayouts'
colcount_format(obj)

colcount_format(obj) <- value

## S4 replacement method for signature 'InstantiatedColumnInfo'
colcount_format(obj) <- value

## S4 replacement method for signature 'VTableNodeInfo'
colcount_format(obj) <- value

## S4 replacement method for signature 'PreDataColLayout'
colcount_format(obj) <- value

## S4 replacement method for signature 'PreDataTableLayouts'
colcount_format(obj) <- value

spl_cuts(obj)

## S4 method for signature 'VarStaticCutSplit'
spl_cuts(obj)

spl_cutlabels(obj)

## S4 method for signature 'VarStaticCutSplit'
spl_cutlabels(obj)

spl_cutfun(obj)

## S4 method for signature 'VarDynCutSplit'
spl_cutfun(obj)

spl_cutlabelfun(obj)

## S4 method for signature 'VarDynCutSplit'
spl_cutlabelfun(obj)

spl_is_cmlcuts(obj)

## S4 method for signature 'VarDynCutSplit'
spl_is_cmlcuts(obj)

spl_varnames(obj)

## S4 method for signature 'MultiVarSplit'
spl_varnames(obj)

spl_varnames(object) <- value
```

```
## S4 replacement method for signature 'MultiVarSplit'
spl_varnames(object) <- value

## S4 method for signature 'VTableTree'
print(x, ...)

## S4 method for signature 'VTableTree'
show(object)
```

## Arguments

| | |
|---|---|
| x | The object. |
| ... | Splits or SplitVector objects |
| lyt | layout object pre-data used for tabulation |
| spl | Split. The split. |
| pos | numeric(1). Intended for internal use. |
| cmpnd_fun | function. Intended for internal use. |
| constructor | function. |
| label | character(1). A label (not to be confused with the name) for the object/structure. |
| cfun | list/function/NULL. tabulation function(s) for creating content rows. Must accept x or df as first parameter. Must accept labelstr as the second argument. Can optionally accept all optional arguments accepted by analysis functions. See analyze. |
| child_labels | string. One of "default", "visible", "hidden". What should the display behavior be for the labels (ie label rows) of the children of this split. Defaults to "default" which flags the label row as visible only if the child has 0 content rows. |
| cformat | format spec. Format for content rows |
| indent_mod | numeric. Modifier for the default indent position for the structure created by this function(subtable, content table, or row) *and all of that structure's children*. Defaults to 0, which corresponds to the unmodified default behavior. |
| cvar | character(1). The variable, if any, which the content function should accept. Defaults to NA. |
| extra_args | list. Extra arguments to be passed to the tabulation function. Element position in thte list corresponds to the children of this split. Named elements in the child-specific lists are ignored if they do not match a formal argument of the ttabulation function. |
| df | dataset (data.frame or tibble) |
| obj | The object. |
| depth | depth in tree |
| indent | indent |
| print_indent | indent for print |

| | |
|---|---|
| nested | boolean, Add this as a new top-level split (defining a new subtable directly under root). Defaults to FALSE |
| for_analyze | logical(1). |
| object | The object to modify in-place |
| value | The new value |
| format | FormatSpec. Format associated with this split. Formats can be declared via strings ("xx.x") or function. In cases such as analyze calls, they can character vectors or lists of functions. |
| override | logical(1). |

## Value

Various, but should be considered implementation details.

## Examples

```
library(dplyr)

iris2 <- iris %>%
  group_by(Species) %>%
  mutate(group = as.factor(rep_len(c("a", "b"), length.out = n()))) %>%
  ungroup()

l <- basic_table() %>%
  split_cols_by("Species") %>%
  split_cols_by("group") %>%
 analyze(c("Sepal.Length", "Petal.Width"), afun = list_wrap_x(summary) , format = "xx.xx")

tbl <- build_table(l, iris2)

summarize_rows(tbl)
```

---

# cbind_rtables *cbind two rtables*

---

## Description

cbind two rtables

## Usage

```
cbind_rtables(x, ...)
```

## Arguments

| | |
|---|---|
| x | A table or row object |
| ... | 1 or more further objects of the same class as x |

## Value

A formal table object.

## Examples

```
x <- rtable(c("A", "B"), rrow("row 1", 1,2), rrow("row 2", 3, 4))

y <- rtable("C", rrow("row 1", 5), rrow("row 2", 6))

z <- rtable("D", rrow("row 1", 9), rrow("row 2", 10))

t1 <- cbind_rtables(x, y)
t1

t2 <- cbind_rtables(x, y, z)
t2

col_paths_summary(t1)
col_paths_summary(t2)
```

---

# CellValue *Cell Value constructor*

---

## Description

Cell Value constructor

## Usage

```
CellValue(
  val,
  format = NULL,
  colspan = 1L,
  label = NULL,
  indent_mod = NULL,
  footnotes = NULL
)
```

## Arguments

| | |
|---|---|
| val | ANY. value in the cell exactly as it should be passed to a formatter or returned when extracted |
| format | FormatSpec. Format associated with this split. Formats can be declared via strings ("xx.x") or function. In cases such as analyze calls, they can character vectors or lists of functions. |
| colspan | integer(1). Columnspan value. |

| | |
|---|---|
| label | character(1). A label (not to be confused with the name) for the object/structure. |
| indent_mod | numeric. Modifier for the default indent position for the structure created by this function(subtable, content table, or row) *and all of that structure's children*. Defaults to 0, which corresponds to the unmodified default behavior. |
| footnotes | list or NULL. Referential footnote messages for the cell. |

## Value

An object representing the value within a single cell within a populated table. The underlying structure of this object is an implementation detail and sholud not be relied upon beyond calling accessors for the class.

---

# cell_values *Retrieve cell values by row and column path*

---

## Description

Retrieve cell values by row and column path

## Usage

```
cell_values(tt, rowpath = NULL, colpath = NULL, omit_labrows = TRUE)

## S4 method for signature 'VTableTree'
cell_values(tt, rowpath = NULL, colpath = NULL, omit_labrows = TRUE)

## S4 method for signature 'TableRow'
cell_values(tt, rowpath = NULL, colpath = NULL, omit_labrows = TRUE)

## S4 method for signature 'LabelRow'
cell_values(tt, rowpath = NULL, colpath = NULL, omit_labrows = TRUE)

value_at(tt, rowpath = NULL, colpath = NULL)

## S4 method for signature 'VTableTree'
value_at(tt, rowpath = NULL, colpath = NULL)

## S4 method for signature 'TableRow'
value_at(tt, rowpath = NULL, colpath = NULL)

## S4 method for signature 'LabelRow'
value_at(tt, rowpath = NULL, colpath = NULL)
```

## Arguments

| | |
|---|---|
| tt | TableTree (or related class). A TableTree object representing a populated table. |
| rowpath | character. Path in row-split space to the desired row(s). Can include "@content". |
| colpath | character. Path in column-split space to the desired column(s). Can include "*". |
| omit_labrows | logical(1). Should label rows underneath rowpath be omitted (TRUE, the default), or return empty lists of cell "values" (FALSE). |

## Value

for cell_values, a *list* (regardless of the type of value the cells hold). if rowpath defines a path to a single row, cell_values returns the list of cell values for that row, otherwise a list of such lists, one for each row captured underneath rowpath. This occurs after subsetting to colpath has occured.

For value_at the "unwrapped" value of a single cell, or an error, if the combination of rowpath and colpath do not define the location of a single cell in tt.

## Note

cell_values will return a single cell's value wrapped in a list. Use value_at to recieve the "bare" cell value.

## Examples

```
 l <- basic_table() %>% split_cols_by("ARM") %>%
   split_cols_by("SEX") %>%
   split_rows_by("RACE") %>%
   summarize_row_groups() %>%
   split_rows_by("STRATA1") %>%
   analyze("AGE")

library(dplyr) ## for mutate
tbl <- build_table(l, DM %>% mutate(SEX = droplevels(SEX), RACE = droplevels(RACE)))

row_paths_summary(tbl)
col_paths_summary(tbl)

cell_values(tbl, c("RACE", "ASIAN", "STRATA1", "B"), c("ARM", "A: Drug X", "SEX", "F"))

# it's also possible to access multiple values by being less specific
cell_values(tbl, c("RACE", "ASIAN", "STRATA1"), c("ARM", "A: Drug X", "SEX", "F"))
cell_values(tbl, c("RACE", "ASIAN"), c("ARM", "A: Drug X", "SEX", "M"))


## any arm, male columns from the ASIAN content (ie summary) row
cell_values(tbl, c("RACE", "ASIAN", "@content"), c("ARM", "B: Placebo", "SEX", "M"))
cell_values(tbl, c("RACE", "ASIAN", "@content"), c("ARM", "*", "SEX", "M"))

## all columns
cell_values(tbl,  c("RACE", "ASIAN", "STRATA1", "B"))
```

```
## all columns for the Combination arm
cell_values(tbl,  c("RACE", "ASIAN", "STRATA1", "B"), c("ARM", "C: Combination"))

cvlist <- cell_values(tbl, c("RACE", "ASIAN", "STRATA1", "B", "AGE", "Mean"),
                      c("ARM", "B: Placebo", "SEX", "M"))
cvnolist <- value_at(tbl,  c("RACE", "ASIAN", "STRATA1", "B", "AGE", "Mean"),
                     c("ARM", "B: Placebo", "SEX", "M"))
stopifnot(identical(cvlist[[1]], cvnolist))
```

---

clayout *Column information/structure accessors*

---

## Description

Column information/structure accessors

## Usage

```
clayout(obj)

## S4 method for signature 'VTableNodeInfo'
clayout(obj)

## S4 method for signature 'PreDataTableLayouts'
clayout(obj)

## S4 method for signature 'ANY'
clayout(obj)

clayout(object) <- value

## S4 replacement method for signature 'PreDataTableLayouts'
clayout(object) <- value

col_info(obj)

## S4 method for signature 'VTableNodeInfo'
col_info(obj)

col_info(obj) <- value

## S4 replacement method for signature 'TableRow'
col_info(obj) <- value

## S4 replacement method for signature 'ElementaryTable'
col_info(obj) <- value

## S4 replacement method for signature 'TableTree'
```

```
col_info(obj) <- value

coltree(obj, df = NULL, rtpos = TreePos())

## S4 method for signature 'InstantiatedColumnInfo'
coltree(obj, df = NULL, rtpos = TreePos())

## S4 method for signature 'PreDataTableLayouts'
coltree(obj, df = NULL, rtpos = TreePos())

## S4 method for signature 'PreDataColLayout'
coltree(obj, df = NULL, rtpos = TreePos())

## S4 method for signature 'LayoutColTree'
coltree(obj, df = NULL, rtpos = TreePos())

## S4 method for signature 'VTableTree'
coltree(obj, df = NULL, rtpos = TreePos())

## S4 method for signature 'TableRow'
coltree(obj, df = NULL, rtpos = TreePos())

col_exprs(obj, df = NULL)

## S4 method for signature 'PreDataTableLayouts'
col_exprs(obj, df = NULL)

## S4 method for signature 'PreDataColLayout'
col_exprs(obj, df = NULL)

## S4 method for signature 'InstantiatedColumnInfo'
col_exprs(obj, df = NULL)

col_counts(obj)

## S4 method for signature 'InstantiatedColumnInfo'
col_counts(obj)

## S4 method for signature 'VTableNodeInfo'
col_counts(obj)

col_counts(obj) <- value

## S4 replacement method for signature 'InstantiatedColumnInfo'
col_counts(obj) <- value

## S4 replacement method for signature 'VTableNodeInfo'
col_counts(obj) <- value
```

```
col_total(obj)

## S4 method for signature 'InstantiatedColumnInfo'
col_total(obj)

## S4 method for signature 'VTableNodeInfo'
col_total(obj)

col_total(obj) <- value

## S4 replacement method for signature 'InstantiatedColumnInfo'
col_total(obj) <- value

## S4 replacement method for signature 'VTableNodeInfo'
col_total(obj) <- value
```

## Arguments

| | |
|---|---|
| obj | ANY. The object for the accessor to access or modify |
| object | The object to modify in-place |
| value | The new value |
| df | data.frame/NULL. Data to use if the column information is being generated from a Pre-Data layout object |
| rtpos | TreePos. Root position. |

## Value

A LayoutColTree object.

Various column information, depending on the accessor used.

---

# clear_indent_mods *Clear All Indent Mods from a Table*

---

## Description

Clear All Indent Mods from a Table

## Usage

```
clear_indent_mods(tt)

## S4 method for signature 'VTableTree'
clear_indent_mods(tt)

## S4 method for signature 'TableRow'
clear_indent_mods(tt)
```

### Arguments

| | |
|---|---|
| tt | TableTree (or related class). A TableTree object representing a populated table. |

### Value

The same class as tt, with all indent mods set to zero.

### Examples

```
t1 <- basic_table() %>%
 summarize_row_groups("STUDYID",label_fstr = "overall summary") %>%
 split_rows_by("AEBODSYS",  child_labels = "visible") %>%
 summarize_row_groups("STUDYID", label = "subgroup summary") %>%
 analyze("AGE", indent_mod = -1L) %>%
 build_table(ex_adae)
t1
clear_indent_mods(t1)
```

---

## collect_leaves *Collect leaves of a table tree*

---

### Description

Collect leaves of a table tree

### Usage

```
collect_leaves(tt, incl.cont = TRUE, add.labrows = FALSE)

## S4 method for signature 'TableTree'
collect_leaves(tt, incl.cont = TRUE, add.labrows = FALSE)

## S4 method for signature 'ElementaryTable'
collect_leaves(tt, incl.cont = TRUE, add.labrows = FALSE)

## S4 method for signature 'VTree'
collect_leaves(tt, incl.cont = TRUE, add.labrows = FALSE)

## S4 method for signature 'VLeaf'
collect_leaves(tt, incl.cont = TRUE, add.labrows = FALSE)

## S4 method for signature '`NULL`'
collect_leaves(tt, incl.cont = TRUE, add.labrows = FALSE)

## S4 method for signature 'ANY'
collect_leaves(tt, incl.cont = TRUE, add.labrows = FALSE)
```

### Arguments

| | |
|---|---|
| tt | TableTree (or related class). A TableTree object representing a populated table. |
| incl.cont | logical. Include rows from content tables within the tree. Defaults to TRUE |
| add.labrows | logical. Include label rows. Defaults to FALSE |

### Value

A list of TableRow objects for all rows in the table

---

## compare_rtables *Compare two rtables*

---

### Description

Prints a matrix where . means cell matches, X means cell does cells do not match, + cell (row) is missing, and - cell (row) should not be there. If structure is set to TRUE, C indicates columnar structure mismatch, R indicates row-structure mismatch, and S indicates mismatch in both row and column structure.

### Usage

```
compare_rtables(
  object,
  expected,
  tol = 0.1,
  comp.attr = TRUE,
  structure = FALSE
)
```

### Arguments

| | |
|---|---|
| object | rtable to test |
| expected | rtable expected |
| tol | numerical tolorance |
| comp.attr | boolean. Compare format of cells. Other attributes are silently ignored. |
| structure | boolean. Should structure (in the form of column and row paths to cells) be compared. Currently defaults to FALSE, but this is subject to change in future versions. |

### Value

a matrix of class "rtables_diff" representing the differences between object and expected as described above.

## Note

In its current form compare_rtables does not take structure into account, only row and cell position.

## Examples

```
t1 <- rtable(header = c("A", "B"), format = "xx", rrow("row 1", 1, 2))
t2 <- rtable(header = c("A", "B", "C"), format = "xx", rrow("row 1", 1, 2, 3))

compare_rtables(object = t1, expected = t2)

if(interactive()){
Viewer(t1, t2)
}

 expected <- rtable(
    header = c("ARM A\nN=100", "ARM B\nN=200"),
    format = "xx",
    rrow("row 1", 10, 15),
    rrow(),
    rrow("section title"),
   rrow("row colspan", rcell(c(.345543, .4432423), colspan = 2, format = "(xx.xx, xx.xx)"))
 )

expected

 object <- rtable(
    header = c("ARM A\nN=100", "ARM B\nN=200"),
    format = "xx",
    rrow("row 1", 10, 15),
    rrow("section title"),
   rrow("row colspan", rcell(c(.345543, .4432423), colspan = 2, format = "(xx.xx, xx.xx)"))
 )

compare_rtables(object, expected, comp.attr = FALSE)

 object <- rtable(
    header = c("ARM A\nN=100", "ARM B\nN=200"),
    format = "xx",
    rrow("row 1", 10, 15),
    rrow(),
    rrow("section title")
 )

 compare_rtables(object, expected)

 object <- rtable(
    header = c("ARM A\nN=100", "ARM B\nN=200"),
    format = "xx",
    rrow("row 1", 14, 15.03),
    rrow(),
```

```
   rrow("section title"),
  rrow("row colspan", rcell(c(.345543, .4432423), colspan = 2, format = "(xx.xx, xx.xx)"))
)

compare_rtables(object, expected)

object <- rtable(
   header = c("ARM A\nN=100", "ARM B\nN=200"),
   format = "xx",
   rrow("row 1", 10, 15),
   rrow(),
   rrow("section title"),
  rrow("row colspan", rcell(c(.345543, .4432423), colspan = 2, format = "(xx.x, xx.x)"))
)

compare_rtables(object, expected)
```

---

compat_args *Compatability Arg Conventions*

---

## Description

Compatability Arg Conventions

## Usage

```
compat_args(
  .lst,
  FUN,
  col_by,
  row_by,
  row.name,
  format,
  indent,
  col_wise_args,
  label
)
```

## Arguments

| | |
|---|---|
| .lst | list. An already-collected list of arguments tot be used instead of the elements of .... Arguments passed via ... will be ignored if this is specified. |
| FUN | function. Tabulation fucntion. Will be passed subsets of x defined by the combination of col_by and row_by and returns corresponding cell value |
| col_by | (factor or data.frame if a factor of length nrow(x) that defines which levels in col_by define a column. |
| row_by | rows in x to take per row in the resulting table |

| | |
|---|---|
| row.name | if NULL then the FUN argument is deparsed and used as row.name of the rrow |
| format | if FUN does not return a formatted rcell then the format is applied |
| indent | deprecated. |
| col_wise_args | a named list containing collections (e.g. vectors or lists) with data elements for each column of the resulting table. The data elements are then passed to the named argument FUN corresponding to the element name of the outer list. Hence, the length and order of each collection must match the levels in col_by. See examples. |
| label | character(1). A label (not to be confused with the name) for the object/structure. |

## Value

NULL (this is an argument template dummy function)

## See Also

Other conventions: constr_args(), gen_args(), lyt_args(), sf_args()

---

constr_args *Constructor Arg Conventions*

---

## Description

Constructor Arg Conventions

## Usage

```
constr_args(
  kids,
  cont,
  lev,
  iscontent,
  cinfo,
  labelrow,
  vals,
  cspan,
  label_pos,
  cindent_mod,
  cvar,
  label,
  cextra_args,
  child_names,
  title,
  subtitles,
  main_footer,
  prov_footer,
  footnotes
)
```

## Arguments

| | |
|---|---|
| kids | list. List of direct children. |
| cont | ElementaryTable. Content table. |
| lev | integer. Nesting level (roughly, indentation level in practical terms). |
| iscontent | logical. Is the TableTree/ElementaryTable being constructed the content table for another TableTree. |
| cinfo | InstantiatedColumnInfo (or NULL). Column structure for the object being created. |
| labelrow | LabelRow. The LabelRow object to assign to this Table. Consructed from label by default if not specified. |
| vals | list. cell values for the row |
| cspan | integer. Column span. 1 indicates no spanning. |
| label_pos | character(1). Location the variable label should be displayed, Accepts hidden (default for non-analyze row splits), visible, topleft, and - for analyze splits only - default. For analyze calls, default indicates that the variable should be visible if and only if multiple variables are analyzed at the same level of nesting. |
| cindent_mod | numeric(1). The indent modifier for the content tables generated by this split. |
| cvar | character(1). The variable, if any, which the content function should accept. Defaults to NA. |
| label | character(1). A label (not to be confused with the name) for the object/structure. |
| cextra_args | list. Extra arguments to be passed to the content function when tabulating row group summaries. |
| child_names | character. Names to be given to the sub splits contained by a compound split (typically a AnalyzeMultiVars split object). |
| title | character(1). Main title. Ignored for subtables. |
| subtitles | character. Subtitles. Ignored for subtables. |
| main_footer | character. Main global (non-referential) footer materials. |
| prov_footer | character. Provenance-related global footer materials. Generally should not be modified by hand. |
| footnotes | list or NULL. Referential footnotes to be applied at current level |

## Value

NULL (this is an argument template dummy function)

## See Also

Other conventions: compat_args(), gen_args(), lyt_args(), sf_args()

## content_table *Retrieve or set Content Table from a TableTree*

### Description

Returns the content table of `obj` if it is a `TableTree` object, or `NULL` otherwise

### Usage

```
content_table(obj)

## S4 method for signature 'TableTree'
content_table(obj)

## S4 method for signature 'ANY'
content_table(obj)

content_table(obj) <- value

## S4 replacement method for signature 'TableTree,ElementaryTable'
content_table(obj) <- value
```

### Arguments

| | |
|---|---|
| `obj` | TableTree. The TableTree |
| `value` | ElementaryTable. The new content table for `obj`. |

### Value

the `ElementaryTable` containing the (top level) *content rows* of `obj` ( or `NULL` if `obj` is not a formal table object).

## cont_n_allcols *Score functions for sorting TableTrees*

### Description

Score functions for sorting TableTrees

### Usage

```
cont_n_allcols(tt)

cont_n_onecol(j)
```

### Arguments

| | |
|---|---|
| tt | TableTree (or related class). A TableTree object representing a populated table. |
| j | numeric(1). Number of column to be scored |

### Value

A single numeric value indicating score according to the relevant metric for tt, to be used when sorting.

---

custom_split_funs *Custom Split Functions Split functions provide the work-horse for* rtables*'s generalized partitioning. These functions accept a (sub)set of incoming data, a split object, and return 'splits' of that data.*

---

### Description

Custom Split Functions

Split functions provide the work-horse for rtables's generalized partitioning. These functions accept a (sub)set of incoming data, a split object, and return 'splits' of that data.

### Custom Splitting Function Details

User-defined custom split functions can perform any type of computation on the incoming data provided that they meed the contract for generating 'splits' of the incoming data 'based on' the split object.

Split functions are functions that accept:

**df** data.frame of incoming data to be split

**spl** a Split object. this is largely an internal detail custom functions will not need to worry about, but obj_name(spl), for example, will give the name of the split as it will appear in paths in the resulting table

**vals** Any pre-calculated values. If given non-null values, the values returned should match these. Should be NULL in most cases and can likely be ignored

**labels** Any pre-calculated value labels. Same as above for values

**trim** If TRUE, resulting splits that are empty should be removed

**(Optional) .spl_context** a data.frame describing previously performed splits which collectively arrived at df

The function must then output a named list with the following elements:

**values** The vector of all values corresponding to the splits of df

**datasplit** a list of data.frames representing the groupings of the actual observations from df.

**labels** a character vector giving a string label for each value listed in the values element above

**(Optional) extras** If present, extra arguments to be passed to summary and analysis functions whenever they are executed on the corresponding element of `datasplit` or a subset thereof

One way to generate custom splitting functions is to wrap existing split functions and modify either the incoming data before they are called, or their outputs.

---

`df_to_tt` *Create ElementaryTable from data.frame*

---

## Description

Create ElementaryTable from data.frame

## Usage

```
df_to_tt(df)
```

## Arguments

| | |
|---|---|
| `df` | data.frame. |

## Value

an `ElementaryTable` object with unnested columns corresponding to `names(df)` and row labels corresponding to `row.names(df)`

## Examples

```
df_to_tt(mtcars)
```

---

`DM` *DM data*

---

## Description

DM data

## Usage

```
DM
```

## Format

rds (data.frame)

ElementaryTable-class *TableTree classes*

## Description

TableTree classes

Table Constructors and Classes

## Usage

```
ElementaryTable(
  kids = list(),
  name = "",
  lev = 1L,
  label = "",
  labelrow = LabelRow(lev = lev, label = label, vis = !isTRUE(iscontent) &&
    !is.na(label) && nzchar(label)),
  rspans = data.frame(),
  cinfo = NULL,
  iscontent = NA,
  var = NA_character_,
  format = NULL,
  indent_mod = 0L,
  title = "",
  subtitles = character(),
  main_footer = character(),
  prov_footer = character()
)

TableTree(
  kids = list(),
  name = if (!is.na(var)) var else "",
  cont = EmptyElTable,
  lev = 1L,
  label = name,
 labelrow = LabelRow(lev = lev, label = label, vis = nrow(cont) == 0 && !is.na(label)
    && nzchar(label)),
  rspans = data.frame(),
  iscontent = NA,
  var = NA_character_,
  cinfo = NULL,
  format = NULL,
  indent_mod = 0L,
  title = "",
  subtitles = character(),
  main_footer = character(),
```

```
  prov_footer = character()
)
```

## Arguments

| | |
|---|---|
| kids | list. List of direct children. |
| name | character(1). Name of the split/table/row being creatted. Defaults to same as the corresponding llabel, but is not required to be. |
| lev | integer. Nesting level (roughly, indentation level in practical terms). |
| label | character(1). A label (not to be confused with the name) for the object/structure. |
| labelrow | LabelRow. The LabelRow object to assign to this Table. Consructed from `label` by default if not specified. |
| rspans | data.frame. Currently stored but otherwise ignored. |
| cinfo | InstantiatedColumnInfo (or NULL). Column structure for the object being created. |
| iscontent | logical. Is the TableTree/ElementaryTable being constructed the content table for another TableTree. |
| var | string, variable name |
| format | FormatSpec. Format associated with this split. Formats can be declared via strings (`"xx.x"`) or function. In cases such as `analyze` calls, they can character vectors or lists of functions. |
| indent_mod | numeric. Modifier for the default indent position for the structure created by this function(subtable, content table, or row) *and all of that structure's children*. Defaults to 0, which corresponds to the unmodified default behavior. |
| title | character(1). Main title. Ignored for subtables. |
| subtitles | character. Subtitles. Ignored for subtables. |
| main_footer | character. Main global (non-referential) footer materials. |
| prov_footer | character. Provenance-related global footer materials. Generally should not be modified by hand. |
| cont | ElementaryTable. Content table. |

## Value

A formal object representing a populated table.

## Author(s)

Gabriel Becker

Gabriel Becker

## EmptyColInfo *Empty table, column, split objects*

### Description

Empty objects of various types to compare against efficiently.

## export_as_pdf *Export as PDF*

### Description

The PDF output is based on the ASCII output created with `toString`

### Usage

```
export_as_pdf(
  tt,
  file,
  width = 11.7,
  height = 8.3,
  margins = c(4, 4, 4, 4),
  fontsize = 8,
  paginate = TRUE,
  lpp = NULL,
  ...
)
```

### Arguments

| | |
|---|---|
| `tt` | TableTree (or related class). A TableTree object representing a populated table. |
| `file` | file to write, must have `.pdf` extension |
| `width` | the width and height of the graphics region in inches |
| `height` | the width and height of the graphics region in inches |
| `margins` | A numeric vector interpreted in the same way as `par(mar)` in base graphics. |
| `fontsize` | the size of text (in points) |
| `paginate` | logical(1). Should `tt` be paginated before writing the file. |
| `lpp` | numeric. Maximum lines per page including (re)printed header and context rows |
| `...` | arguments passed on to `paginate_table` |

### See Also

`export_as_txt`

## Examples

```
lyt <- basic_table() %>%
  split_cols_by("ARM") %>%
  analyze(c("AGE", "BMRKR2", "COUNTRY"))

tbl <- build_table(lyt, ex_adsl)

## Not run:
tf <- tempfile(fileext = ".pdf")
export_as_pdf(tbl, file = tf, height = 4)
tf <- tempfile(fileext = ".pdf")
export_as_pdf(tbl, file = tf, lpp = 8)

## End(Not run)
```

---

`export_as_tsv` *Create Enriched flat value table with paths*

---

## Description

This function creates a flat tabular file of cell values and corresponding paths via `path_enriched_df`. I then writes that data.frame out as a tsv file.

## Usage

```
export_as_tsv(
  tt,
  file = NULL,
  path_fun = collapse_path,
  value_fun = collapse_values
)

import_from_tsv(file)
```

## Arguments

| | |
|---|---|
| `tt` | TableTree (or related class). A TableTree object representing a populated table. |
| `file` | character(1). The path of the file to written to or read from. |
| `path_fun` | function. Function to transform paths into single-string row/column names. |
| `value_fun` | function. Functiont to transform cell values into cells of the data.frame. Defaults to `collapse_values` which creates strings where multi-valued cells are collapsed together, separated by `\|`. |

## Details

By default (ie when `value_func` is not specified, List columns where at least one value has length > 1 are collapsed to character vectors by collapsing the list element with `"|"`.

## Value

NULL silently for export_as_tsv, a data.frame with re-constituted list values for export_as_tsv.

## Note

There is currently no round-trip capability for this type of export. You can read values exported this way back in via import_from_tsv but you will receive only the data.frame version back, NOT a TableTree.

---

# export_as_txt *Export as plain text with page break symbol*

---

## Description

Export as plain text with page break symbol

## Usage

```
export_as_txt(
  tt,
  file = NULL,
  paginate = FALSE,
  ...,
  page_break = "\\s\\n"
)
```

## Arguments

| | |
|---|---|
| tt | TableTree (or related class). A TableTree object representing a populated table. |
| file | character(1). File to write. |
| paginate | logical(1). Should tt be paginated before writing the file. |
| ... | Passed directly to paginate_table |
| page_break | character(1). Page break symbol (defaults to outputting "\s"). |

## Value

file (this function is called for the side effect of writing the file.

## See Also

export_as_pdf

## Examples

```
lyt <- basic_table() %>%
  split_cols_by("ARM") %>%
  analyze(c("AGE", "BMRKR2", "COUNTRY"))

tbl <- build_table(lyt, ex_adsl)

cat(export_as_txt(tbl, file = NULL, paginate = TRUE, lpp = 8))

## Not run:
tf <- tempfile(fileext = ".txt")
export_as_txt(tbl, file = tf)
system2("cat", tf)

## End(Not run)
```

---

# ex_adsl *Simulated CDISC Alike Data for Examples*

---

## Description

Simulated CDISC Alike Data for Examples

## Usage

```
ex_adsl

ex_adae

ex_adaette

ex_adtte

ex_adcm

ex_adlb

ex_admh

ex_adqs

ex_adrs

ex_advs
```

### Format

rds (data.frame)

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 1934 rows and 48 columns.

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 1200 rows and 42 columns.

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 1200 rows and 42 columns.

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 1934 rows and 41 columns.

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 8400 rows and 59 columns.

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 1934 rows and 41 columns.

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 14000 rows and 49 columns.

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 2400 rows and 41 columns.

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 16800 rows and 59 columns.

---

`format_rcell` *Convert the contets of an* `rcell` *to a string using the* `format` *information*

---

### Description

Convert the contets of an `rcell` to a string using the `format` information

### Usage

```
format_rcell(x, format, output = c("ascii", "html"))
```

### Arguments

| | |
|---|---|
| `x` | an object of class `rcell` |
| `format` | if `FUN` does not return a formatted `rcell` then the `format` is applied |
| `output` | output type |

### Value

formatted text representing the cell `x`.

### Examples

```
x <- rcell(pi, format = "xx.xx")
x

format_rcell(x, output = "ascii")
```

gen_args *General Argument Conventions*

## Description

General Argument Conventions

## Usage

```
gen_args(
  df,
  alt_counts_df,
  spl,
  pos,
  tt,
  tr,
  verbose,
  colwidths,
  obj,
  x,
  value,
  object,
  path,
  label,
  label_pos,
  cvar,
  topleft,
  ...
)
```

## Arguments

| | |
|---|---|
| df | dataset (data.frame or tibble) |
| alt_counts_df | dataset (data.frame or tibble). Alternative full data the rtables framework will use (*only*) when calculating column counts. |
| spl | A Split object defining a partitioning or analysis/tabulation of the data. |
| pos | numeric. Which top-level set of nested splits should the new layout feature be added to. Defaults to the current |
| tt | TableTree (or related class). A TableTree object representing a populated table. |
| tr | TableRow (or related class). A TableRow object representing a single row within a populated table. |
| verbose | logical(1). Should extra debugging messages be shown. Defaults to FALSE. |
| colwidths | numeric vector. Column widths for use with vertical pagination. Currently ignored. |
| obj | ANY. The object for the accessor to access or modify |

| | |
|---|---|
| x | An object |
| value | The new value |
| object | The object to modify in-place |
| path | character. A vector path for a position within the structure of a tabletree. Each element represents a subsequent choice amongst the children of the previous choice. |
| label | character(1). A label (not to be confused with the name) for the object/structure. |
| label_pos | character(1). Location the variable label should be displayed, Accepts hidden (default for non-analyze row splits), visible, topleft, and - for analyze splits only - default. For analyze calls, `default` indicates that the variable should be visible if and only if multiple variables are analyzed at the same level of nesting. |
| cvar | character(1). The variable, if any, which the content function should accept. Defaults to NA. |
| topleft | character. Override values for the "top left" material to be displayed during printing. |
| ... | Passed on to metthods or tabulation functions. |

## Value

NULL (this is an argument template dummy function)

## See Also

Other conventions: `compat_args()`, `constr_args()`, `lyt_args()`, `sf_args()`

---

# get_formatted_cells *get formatted cells*

---

## Description

get formatted cells

## Usage

```
get_formatted_cells(obj)

## S4 method for signature 'TableTree'
get_formatted_cells(obj)

## S4 method for signature 'ElementaryTable'
get_formatted_cells(obj)

## S4 method for signature 'TableRow'
get_formatted_cells(obj)

## S4 method for signature 'LabelRow'
get_formatted_cells(obj)
```

### Arguments

| | |
|---|---|
| obj | ANY. The object for the accessor to access or modify |

### Value

the formatted print-strings for all (body) cells in obj.

### Examples

```
library(dplyr)

iris2 <- iris %>%
  group_by(Species) %>%
  mutate(group = as.factor(rep_len(c("a", "b"), length.out = n()))) %>%
  ungroup()

tbl <- basic_table() %>%
  split_cols_by("Species") %>%
  split_cols_by("group") %>%
 analyze(c("Sepal.Length", "Petal.Width"), afun = list_wrap_x(summary) , format = "xx.xx") %>%
  build_table(iris2)

get_formatted_cells(tbl)
```

---

## indent *Change indentation of all rrows in an rtable*

---

### Description

Change indentation of all rrows in an rtable

### Usage

```
indent(x, by = 1)
```

### Arguments

| | |
|---|---|
| x | rtable object |
| by | integer to increase indentation of rows. Can be negative. If final indentation is smaller than 0 then the indentation is set to 0. |

### Value

x with its indent modifier incremented by by.

## Examples

```
is_setosa <- iris$Species == "setosa"
mtbl <- rtable(
  header = rheader(
   rrow(row.name = NULL, rcell("Sepal.Length", colspan = 2), rcell("Petal.Length", colspan=2)),
    rrow(NULL, "mean", "median", "mean", "median")
  ),
  rrow(
    row.name = "All Species",
    mean(iris$Sepal.Length), median(iris$Sepal.Length),
    mean(iris$Petal.Length), median(iris$Petal.Length),
    format = "xx.xx"
  ),
  rrow(
    row.name = "Setosa",
    mean(iris$Sepal.Length[is_setosa]), median(iris$Sepal.Length[is_setosa]),
    mean(iris$Petal.Length[is_setosa]), median(iris$Petal.Length[is_setosa]),
    format = "xx.xx"
  )
)
indent(mtbl)
indent(mtbl, 2)
```

---

`indent_string` *Indent Strings*

---

## Description

Used in rtables to indent row names for the ASCII output.

## Usage

```
indent_string(x, indent = 0, incr = 2, including_newline = TRUE)
```

## Arguments

| | |
|---|---|
| `x` | a character vector |
| `indent` | a vector of length `length(x)` with non-negative integers |
| `incr` | non-negative integer: number of spaces per indent level |
| `including_newline` | boolean: should newlines also be indented |

## Value

x indented by left-padding with codeindent*incr white-spaces.

## Examples

```
indent_string("a", 0)
indent_string("a", 1)
indent_string(letters[1:3], 0:2)
indent_string(paste0(letters[1:3], "\n", LETTERS[1:3]), 0:2)
```

---

`insert_row_at_path` *Insert Row at Path*

---

## Description

Insert a row into an existing table directly before or directly after an existing data (i.e., non-content and non-label) row, specified by its path.

## Usage

```
insert_row_at_path(tt, path, value, after = FALSE)

## S4 method for signature 'VTableTree,DataRow'
insert_row_at_path(tt, path, value, after = FALSE)

## S4 method for signature 'VTableTree,ANY'
insert_row_at_path(tt, path, value)
```

## Arguments

| | |
|---|---|
| `tt` | TableTree (or related class). A TableTree object representing a populated table. |
| `path` | character. A vector path for a position within the structure of a tabletree. Each element represents a subsequent choice amongst the children of the previous choice. |
| `value` | The new value |
| `after` | logical(1). Should `value` be added as a row directly before (`FALSE`, the default) or after (`TRUE`) the row specified by `path`. |

## See Also

DataRow rrow

## Examples

```
lyt <- basic_table() %>%
  split_rows_by("COUNTRY", split_fun = keep_split_levels(c("CHN", "USA"))) %>%
  analyze("AGE")

tab <- build_table(lyt, DM)
```

```
tab2 <- insert_row_at_path(tab, c("COUNTRY", "CHN", "AGE", "Mean"),
                           rrow("new row", 555))
tab2
tab2 <- insert_row_at_path(tab2, c("COUNTRY", "CHN", "AGE", "Mean"),
                           rrow("new row redux", 888),
                           after = TRUE)
tab2
```

---

`insert_rrow` *[DEPRECATED] insert rrows at (before) a specific location*

---

## Description

This function is deprecated and will be removed in a future release of rtables. Please use `insert_row_at_path` or `label_at_path` instead.

## Usage

```
insert_rrow(tbl, rrow, at = 1, ascontent = FALSE)
```

## Arguments

| | |
|---|---|
| `tbl` | rtable |
| `rrow` | rrow to append to rtable |
| `at` | position into which to put the rrow, defaults to beginning (ie 1) |
| `ascontent` | logical. Currently ignored. |

## Value

A TableTree of the same specific class as `tbl`

## Note

Label rows (ie a row with no data values, only a row.name) can only be inserted at positions which do not already contain a label row when there is a non-trivial nested row structure in `tbl`

## See Also

Other compatability: `rtable()`

## Examples

```
o <- options(warn = 0)
tbl <- basic_table() %>%
    split_cols_by("Species") %>%
    analyze("Sepal.Length") %>%
    build_table(iris)

insert_rrow(tbl, rrow("Hello World"))
insert_rrow(tbl, rrow("Hello World"), at = 2)

tbl2 <- basic_table() %>%
    split_cols_by("Species") %>%
    split_rows_by("Species") %>%
    analyze("Sepal.Length") %>%
    build_table(iris)

insert_rrow(tbl2, rrow("Hello World"))
insert_rrow(tbl2, rrow("Hello World"), at = 2)
insert_rrow(tbl2, rrow("Hello World"), at = 4)

insert_rrow(tbl2, rrow("new row", 5, 6, 7))

insert_rrow(tbl2, rrow("new row", 5, 6, 7), at = 3)

options(o)
```

---

# InstantiatedColumnInfo-class

*InstantiatedColumnInfo*

---

## Description

InstantiatedColumnInfo

## Usage

```
InstantiatedColumnInfo(
  treelyt = LayoutColTree(),
  csubs = list(expression(TRUE)),
  extras = list(list()),
  cnts = NA_integer_,
  total_cnt = NA_integer_,
  dispcounts = FALSE,
  countformat = "(N=xx)",
  topleft = character()
)
```

### Arguments

| | |
|---|---|
| treelyt | LayoutColTree. |
| csubs | list. List of subsetting expressions |
| extras | list. Extra arguments associated with the columns |
| cnts | integer. Counts. |
| total_cnt | integer(1). Total observations represented across all columns. |
| dispcounts | logical. Should the counts be displayed as header info when the associated table is printed. |
| countformat | string. Format for the counts if thtey are displayed |
| topleft | character. Override values for the "top left" material to be displayed during printing. |

### Value

an InstantiateadColumnInfo object.

---

## in_rows *Create multiple rows in analysis or summary functions*

---

### Description

define the cells that get placed into multiple rows in afun

### Usage

```
in_rows(
  ...,
  .list = NULL,
  .names = NULL,
  .labels = NULL,
  .formats = NULL,
  .indent_mods = NULL,
  .cell_footnotes = list(NULL),
  .row_footnotes = list(NULL)
)
```

### Arguments

| | |
|---|---|
| ... | single row defining expressions |
| .list | list. list cell content, usually rcells, the .list is concatenated to ... |
| .names | character or NULL. Names of the returned list/structure. |
| .labels | character or NULL. labels for the defined rows |
| .formats | character or NULL. Formats for the values |

.indent_mods integer or NULL. Indent modificatons for the defined rows.

.cell_footnotes list. Referential footnote messages to be associated by name with *cells*

.row_footnotes list. Referential footnotes messages to be associated by name with *rows*

## Value

an RowsVerticalSection object (or NULL). The details of this object should be considered an internal implementation detail.

## Note

currently the .name argument is not used

## See Also

analyze

## Examples

```
in_rows(1, 2, 3, .names = c("a", "b", "c"))
in_rows(1, 2, 3, .labels = c("a", "b", "c"))
in_rows(1, 2, 3, .names = c("a", "b", "c"), .labels = c("AAA", "BBB", "CCC"))

in_rows(.list = list(a = 1, b = 2, c = 3))
in_rows(1, 2, .list = list(3), .names = c("a", "b", "c"))

basic_table() %>%
  split_cols_by("ARM") %>%
  analyze("AGE", afun = function(x) {
    in_rows(
       "Mean (sd)" = rcell(c(mean(x), sd(x)), format = "xx.xx (xx.xx)"),
       "Range" = rcell(range(x), format = "xx.xx - xx.xx")
    )
  }) %>%
  build_table(ex_adsl)
```

---

# is_rcell_format *Check if a format is a valid rcell format*

---

## Description

Check if a format is a valid rcell format

## Usage

```
is_rcell_format(x, stop_otherwise = FALSE)
```

### Arguments

| | |
|---|---|
| x | either format string or an object returned by sprintf_format |
| stop_otherwise | logical, if x is not a format should an error be thrown |

### Value

TRUE if x is NULL, a supported format string, or a function; FALSE otherwise.

### Note

No check if the function is actually a formatter is performed.

---

## is_rtable *Check if an object is a valid rtable*

---

### Description

Check if an object is a valid rtable

### Usage

```
is_rtable(x)
```

### Arguments

| | |
|---|---|
| x | an object |

### Value

TRUE if x is a formal Table object, FALSE otherwise.

### Examples

```
is_rtable(build_table(basic_table(), iris))
```

LabelRow *Row classes and constructors*

## Description

Row classes and constructors

Row constructors and Classes

## Usage

```
LabelRow(
  lev = 1L,
  label = "",
  name = label,
  vis = !is.na(label) && nzchar(label),
  cinfo = EmptyColInfo,
  indent_mod = 0L
)

.tablerow(
  vals = list(),
  name = "",
  lev = 1L,
  label = name,
  cspan = rep(1L, length(vals)),
  cinfo = EmptyColInfo,
  var = NA_character_,
  format = NULL,
  klass,
  indent_mod = 0L,
  footnotes = list()
)

DataRow(...)

ContentRow(...)
```

## Arguments

| | |
|---|---|
| lev | integer. Nesting level (roughly, indentation level in practical terms). |
| label | character(1). A label (not to be confused with the name) for the object/structure. |
| name | character(1). Name of the split/table/row being creatted. Defaults to same as the corresponding llabel, but is not required to be. |
| vis | logical. Should the row be visible (LabelRow only). |
| cinfo | InstantiatedColumnInfo (or NULL). Column structure for the object being created. |

| | |
|---|---|
| indent_mod | numeric. Modifier for the default indent position for the structure created by this function(subtable, content table, or row) *and all of that structure's children*. Defaults to 0, which corresponds to the unmodified default behavior. |
| vals | list. cell values for the row |
| cspan | integer. Column span. 1 indicates no spanning. |
| var | string, variable name |
| format | FormatSpec. Format associated with this split. Formats can be declared via strings ("xx.x") or function. In cases such as analyze calls, they can character vectors or lists of functions. |
| klass | Internal detail. |
| footnotes | list or NULL. Referential footnotes to be applied at current level |
| ... | passed to shared constructor (.tablerow). |

## Value

A formal object representing a table row of the constructed type.

## Author(s)

Gabriel Becker

---

# label_at_path *Label at Path*

---

## Description

Gets or sets the label at a path

## Usage

```
label_at_path(tt, path)

label_at_path(tt, path) <- value
```

## Arguments

| | |
|---|---|
| tt | TableTree (or related class). A TableTree object representing a populated table. |
| path | character. A vector path for a position within the structure of a tabletree. Each element represents a subsequent choice amongst the children of the previous choice. |
| value | The new value |

## Details

If path resolves to a single row, the label for that row is retrieved or set. If, instead, path resolves to a subtable, the text for the row-label associated with that path is retrieved or set. In the subtable case, if the label text is set to a non-NA value, the labelrow will be set to visible, even if it was not before. Similarly, if the label row text for a subtable is set to NA, the label row will bet set to non-visible, so the row will not appear at all when the table is printed.

## Note

When changing the row labels for content rows, it is important to path all the way to the *row*. Paths ending in "@content" will not exhibit the behavior you want, and are thus an error. See row_paths for help determining the full paths to content rows.

## Examples

```
lyt <- basic_table() %>%
  split_rows_by("COUNTRY", split_fun = keep_split_levels(c("CHN", "USA"))) %>%
  analyze("AGE")

tab <- build_table(lyt, DM)

label_at_path(tab, c("COUNTRY", "CHN"))

label_at_path(tab, c("COUNTRY", "USA")) <- "United States"
tab
```

---

# length,CellValue-method

*Length of a Cell value*

---

## Description

Length of a Cell value

## Usage

```
## S4 method for signature 'CellValue'
length(x)
```

## Arguments

| | |
|---|---|
| x | x. |

## Value

Always returns 1L

## list_rcell_format_labels

*List with valid rcell formats labels grouped by 1d, 2d and 3d*

### Description

Currently valid format labels can not be added dynamically. Format functions must be used for special cases

### Usage

```
list_rcell_format_labels()
```

### Value

A nested list, with elements listing the supported 1d, 2d, and 3d format strings.

### Examples

```
list_rcell_format_labels()
```

## list_wrap_x

*Returns a function that coerces the return values of f to a list*

### Description

Returns a function that coerces the return values of f to a list

### Usage

```
list_wrap_x(f)

list_wrap_df(f)
```

### Arguments

| | |
|---|---|
| f | The function to wrap. |

### Details

`list_wrap_x` generates a wrapper which takes `x` as its first argument, while `list_wrap_df` generates an otherwise identical wrapper function whose first argument is named `df`.

We provide both because when using the functions as tabulation in `analyze`, functions which take `df` as their first argument are passed the full subset dataframe, while those which accept anything else notably including `x` are passed only the relevant subset of the variable being analyzed.

## Value

A function which calls `f` and converts the result to a list of `CellValue` objects.

## Author(s)

Gabriel Becker

## Examples

```
summary(iris$Sepal.Length)

f <- list_wrap_x(summary)
f(x = iris$Sepal.Length)

f2 <- list_wrap_df(summary)
f2(df = iris$Sepal.Length)
```

---

# lyt_args *Layouting Function Arg Conventions*

---

## Description

Layouting Function Arg Conventions

## Usage

```
lyt_args(
  lyt,
  var,
  vars,
  label,
  labels_var,
  varlabels,
  varnames,
  split_format,
  nested,
  format,
  cfun,
  cformat,
  split_fun,
  split_name,
  split_label,
  afun,
  inclNAs,
  valorder,
  ref_group,
```

```
  compfun,
  label_fstr,
  child_labels,
  extra_args,
  name,
  cuts,
  cutlabels,
  cutfun,
  cutlabelfun,
  cumulative,
  indent_mod,
  show_labels,
  label_pos,
  var_labels,
  cvar,
  table_names,
  topleft
)
```

## Arguments

| | |
|---|---|
| lyt | layout object pre-data used for tabulation |
| var | string, variable name |
| vars | character vector. Multiple variable names. |
| label | character(1). A label (not to be confused with the name) for the object/structure. |
| labels_var | string, name of variable containing labels to be displayed for the values of var |
| varlabels | character vector. Labels for vars |
| varnames | character vector. Names for vars which will appear in pathing. When vars are all unique this will be the variable names. If not, these will be variable names with suffixes as necessary to enforce uniqueness. |
| split_format | FormatSpec. Default format associated with the split being created. |
| nested | boolean, Add this as a new top-level split (defining a new subtable directly under root). Defaults to FALSE |
| format | FormatSpec. Format associated with this split. Formats can be declared via strings ("xx.x") or function. In cases such as analyze calls, they can character vectors or lists of functions. |
| cfun | list/function/NULL. tabulation function(s) for creating content rows. Must accept x or df as first parameter. Must accept labelstr as the second argument. Can optionally accept all optional arguments accepted by analysis functions. See analyze. |
| cformat | format spec. Format for content rows |
| split_fun | function/NULL. custom splitting function See custom_split_funs |
| split_name | string. Name associiated with this split (for pathing, etc) |

| | |
|---|---|
| split_label | string. Label string to be associated with the table generated by the split. Not to be confused with labels assigned to each child (which are based on the data and type of split during tabulation). |
| afun | function. Analysis function, must take x or df as its first parameter. Can optionally take other parameters which will be populated by the tabulation framework. See Details in analyze. |
| inclNAs | boolean. Should observations with NA in the var variable(s) be included when performing this analysis. Defaults to FALSE |
| valorder | character vector. Order that the split children should appear in resulting table. |
| ref_group | character. Value of var to be taken as the ref_group/control to be compared against. |
| compfun | function/string. The comparison function which accepts the analysis function outputs for two different partitions and returns a single value. Defaults to subraction. If a string, taken as the name of a function. |
| label_fstr | string. An sprintf style format string containing. For non-comparison splits, it can contain up to one "%s" which takes the current split value and generates the row/column label. Comparison-based splits it can contain up to two "%s". |
| child_labels | string. One of "default", "visible", "hidden". What should the display behavior be for the labels (ie label rows) of the children of this split. Defaults to "default" which flags the label row as visible only if the child has 0 content rows. |
| extra_args | list. Extra arguments to be passed to the tabulation function. Element position in thte list corresponds to the children of this split. Named elements in the child-specific lists are ignored if they do not match a formal argument of the ttabulation function. |
| name | character(1). Name of the split/table/row being creatted. Defaults to same as the corresponding llabel, but is not required to be. |
| cuts | numeric. Cuts to use |
| cutlabels | character (or NULL). Labels for the cuts |
| cutfun | function. Function which accepts the *full vector* of var values and returns cut points to be used (via cut) when splitting data during tabulation |
| cutlabelfun | function. Function which returns either labels for the cuts or NULL when passed the return value of cutfun |
| cumulative | logical. Should the cuts be treated as cumulative. Defaults to FALSE |
| indent_mod | numeric. Modifier for the default indent position for the structure created by this function(subtable, content table, or row) *and all of that structure's children*. Defaults to 0, which corresponds to the unmodified default behavior. |
| show_labels | character(1). Should the variable labels for corresponding to the variable(s) in vars be visible in the resulting table. |
| label_pos | character(1). Location the variable label should be displayed, Accepts hidden (default for non-analyze row splits), visible, topleft, and - for analyze splits only - default. For analyze calls, default indicates that the variable should be visible if and only if multiple variables are analyzed at the same level of nesting. |

| | |
|---|---|
| var_labels | character. Variable labels for 1 or more variables |
| cvar | character(1). The variable, if any, which the content function should accept. Defaults to NA. |
| table_names | character. Names for the tables representing each atomic analysis. Defaults to var. |
| topleft | character. Override values for the "top left" material to be displayed during printing. |

## Value

NULL (this is an argument template dummy function)

## See Also

Other conventions: compat_args(), constr_args(), gen_args(), sf_args()

---

make_afun *Create custom analysis function wrapping existing function*

---

## Description

Create custom analysis function wrapping existing function

## Usage

```
make_afun(
  fun,
  .stats = NULL,
  .formats = NULL,
  .labels = NULL,
  .indent_mods = NULL,
  .ungroup_stats = NULL,
  ...,
  .null_ref_cells = ".in_ref_col" %in% names(formals(fun))
)
```

## Arguments

| | |
|---|---|
| fun | function. The function to be wrapped in a new customized analysis fun. Should return named list. |
| .stats | character. Names of elements to keep from fun's full output. |
| .formats | ANY. vector/list of formats to override any defaults applied by fun. |
| .labels | character. Vector of labels to override defaults returned by fun |
| .indent_mods | integer. Named vector of indent modifiers for the generated rows. |
| .ungroup_stats | character. Vector of names, which must match elements of .stats |

`...` dots. Additional arguments to `fun` which effectively become new defaults. These can still be overriden by extra args within a split.

`.null_ref_cells` logical(1). Should cells for the reference column be NULL-ed by the returned analysis function. Defaults to `TRUE` if `fun` accepts `.in_ref_col` as a formal argument. Note this argument occurs after `...` so it must be *fully* specified by name when set.

## Value

A function suitable for use in `analyze` with element selection, reformatting, and relabeling performed automatically.

## Note

setting `.ungroup_stats` to non-null changes the *structure* of the value(s) returned by `fun`, rather than just labeling (`.labels`), formatting (`.formats`), and selecting amongst (`.stats`) them. This means that subsequent `make_afun` calls to customize the output further both can and must operate on the new structure, *NOT* the original structure returned by `fun`. See the final pair of examples below.

## See Also

`analyze()`

## Examples

```
s_summary <- function(x) {
  stopifnot(is.numeric(x))

  list(
    n = sum(!is.na(x)),
    mean_sd = c(mean = mean(x), sd = sd(x)),
    min_max = range(x)
  )
}

s_summary(iris$Sepal.Length)

a_summary <- make_afun(
  fun = s_summary,
  .formats = c(n = "xx", mean_sd = "xx.xx (xx.xx)", min_max = "xx.xx - xx.xx"),
  .labels = c(n = "n", mean_sd = "Mean (sd)", min_max = "min - max")
)

a_summary(x = iris$Sepal.Length)

a_summary2 <- make_afun(a_summary, .stats = c("n", "mean_sd"))

a_summary2(x = iris$Sepal.Length)
```

```
a_summary3 <- make_afun(a_summary, .formats = c(mean_sd = "(xx.xxx, xx.xxx)"))



s_foo <- function(df, .N_col, a = 1, b = 2) {
   list(
      nrow_df = nrow(df),
      .N_col = .N_col,
      a = a,
      b = b
   )
}

s_foo(iris, 40)

a_foo <- make_afun(s_foo, b = 4,
 .formats = c(nrow_df = "xx.xx", ".N_col" = "xx.", a = "xx", b = "xx.x"),
 .labels = c(nrow_df = "Nrow df", ".N_col" = "n in cols", a = "a value", b = "b value"),
 .indent_mods = c(nrow_df = 2L, a = 1L)
)

a_foo(iris, .N_col = 40)
a_foo2 <- make_afun(a_foo, .labels = c(nrow_df = "Number of Rows"))
a_foo(iris, .N_col = 40)

#grouping and further customization
s_grp <- function(df, .N_col, a = 1, b = 2) {
   list(
      nrow_df = nrow(df),
      .N_col = .N_col,
      letters = list(a = a,
                     b = b)
   )
}
a_grp <- make_afun(s_grp, b = 3, .labels = c(nrow_df = "row count", .N_col = "count in column"),
                   .formats = c(nrow_df = "xx.", .N_col = "xx."),
                   .indent_mod = c(letters = 1L),
                   .ungroup_stats ="letters")
a_grp(iris, 40)
a_aftergrp <- make_afun(a_grp, .stats = c("nrow_df", "b"), .formats = c(b = "xx."))
a_aftergrp(iris, 40)

s_ref <- function(x, .in_ref_col, .ref_group) {
   list(
         mean_diff = mean(x) - mean(.ref_group)
       )
}

a_ref <- make_afun(s_ref, .labels = c( mean_diff = "Mean Difference from Ref"))
a_ref(iris$Sepal.Length, .in_ref_col = TRUE, 1:10)
a_ref(iris$Sepal.Length, .in_ref_col = FALSE, 1:10)
```

make_row_df *Make row and column layout summary data.frames for use during pagination*

## Description

Used for Pagination

## Usage

```
make_row_df(
  tt,
  colwidths = NULL,
  visible_only = TRUE,
  rownum = 0,
  indent = 0L,
  path = character(),
  incontent = FALSE,
  repr_ext = 0L,
  repr_inds = integer(),
  sibpos = NA_integer_,
  nsibs = NA_integer_,
  nrowrefs = 0L,
  ncellrefs = 0L,
  nreflines = 0L
)

## S4 method for signature 'VTableTree'
make_row_df(
  tt,
  colwidths = NULL,
  visible_only = TRUE,
  rownum = 0,
  indent = 0L,
  path = character(),
  incontent = FALSE,
  repr_ext = 0L,
  repr_inds = integer(),
  sibpos = NA_integer_,
  nsibs = NA_integer_
)

## S4 method for signature 'TableRow'
make_row_df(
  tt,
```

```
  colwidths = NULL,
  visible_only = TRUE,
  rownum = 0,
  indent = 0L,
  path = "root",
  incontent = FALSE,
  repr_ext = 0L,
  repr_inds = integer(),
  sibpos = NA_integer_,
  nsibs = NA_integer_
)

## S4 method for signature 'LabelRow'
make_row_df(
  tt,
  colwidths = NULL,
  visible_only = TRUE,
  rownum = 0,
  indent = 0L,
  path = "root",
  incontent = FALSE,
  repr_ext = 0L,
  repr_inds = integer(),
  sibpos = NA_integer_,
  nsibs = NA_integer_
)

make_col_df(tt, visible_only = TRUE)
```

## Arguments

| | |
|---|---|
| tt | TableTree (or related class). A TableTree object representing a populated table. |
| colwidths | numeric. Internal detail do not set manually. |
| visible_only | logical(1). Should only visible aspects of the table structure be reflected in this summary. Defaults to TRUE. |
| rownum | numeric(1). Internal detail do not set manually. |
| indent | integer(1). Internal detail do not set manually. |
| path | character. A vector path for a position within the structure of a tabletree. Each element represents a subsequent choice amongst the children of the previous choice. |
| incontent | logical(1). Internal detail do not set manually. |
| repr_ext | integer(1). Internal detail do not set manually. |
| repr_inds | integer. Internal detail do not set manually. |
| sibpos | integer(1). Internal detail do not set manually. |
| nsibs | integer(1). Internal detail do not set manually. |
| nrowrefs | integer(1). Internal detail do not set manually. |

ncellrefs integer(1). Internal detail do not set manually.

nreflines integer(1). Internal detail do not set manually.

## Details

When visible_only is TRUE, the resulting data.frame will have exactly one row per visible row in the table. This is useful when reasoning about how a table will print, but does not reflect the full pathing space of the structure (though the paths which are given will all work as is).

When visible_only is FALSE, every structural element of the table (in row-space) will be reflected in the returned data.frame, meaning the full pathing-space will be represented but some rows in the layout summary will not represent printed rows in the table as it is displayed.

## Value

a data.frame of row/column-structure information used by the pagination machinery.

## Note

the technically present root tree node is excluded from the summary returne dby both make_row_df and make_col_df, as it is simply the row/column structure of tt and thus not useful for pathing or pagination.

---

# ManualSplit *Manually defined split*

---

## Description

Manually defined split

## Usage

```
ManualSplit(
  levels,
  label,
  name = "manual",
  extra_args = list(),
  indent_mod = 0L,
  cindent_mod = 0L,
  cvar = "",
  cextra_args = list(),
  label_pos = "visible"
)
```

### Arguments

| | |
|---|---|
| levels | character. Levels of the split (ie the children of the manual split) |
| label | character(1). A label (not to be confused with the name) for the object/structure. |
| name | character(1). Name of the split/table/row being creatted. Defaults to same as the corresponding llabel, but is not required to be. |
| extra_args | list. Extra arguments to be passed to the tabulation function. Element position in thte list corresponds to the children of this split. Named elements in the child-specific lists are ignored if they do not match a formal argument of the ttabulation function. |
| indent_mod | numeric. Modifier for the default indent position for the structure created by this function(subtable, content table, or row) *and all of that structure's children*. Defaults to 0, which corresponds to the unmodified default behavior. |
| cindent_mod | numeric(1). The indent modifier for the content tables generated by this split. |
| cvar | character(1). The variable, if any, which the content function should accept. Defaults to NA. |
| cextra_args | list. Extra arguments to be passed to the content function when tabulating row group summaries. |
| label_pos | character(1). Location the variable label should be displayed, Accepts hidden (default for non-analyze row splits), visible, topleft, and - for analyze splits only - default. For analyze calls, `default` indicates that the variable should be visible if and only if multiple variables are analyzed at the same level of nesting. |

### Value

A `ManualSplit` object.

### Author(s)

Gabriel Becker

---

## manual_cols *Manual column declaration*

---

### Description

Manual column declaration

### Usage

```
manual_cols(..., .lst = list(...))
```

### Arguments

| | |
|---|---|
| ... | One or more vectors of levels to appear in the column splace. If more than one set of levels is given, the values of the second are nested within each value of the first, and so on. |
| .lst | A list of sets of levels, by default populated via `list(...)`. |

### Value

An InstantiatedColumnInfo object, suitable for use declaring the column structure for a manually constructed table.

### Author(s)

Gabriel Becker

### Examples

```
# simple one level column space
rows = lapply(1:5, function(i) {
   DataRow(rep(i, times  = 3))})
tab = TableTree(kids = rows, cinfo = manual_cols(split = c("a", "b", "c")))
tab

# manually declared nesting
tab2 = TableTree(kids = list(DataRow(as.list(1:4))),
                 cinfo = manual_cols(Arm = c("Arm A", "Arm B"),
                                     Gender = c("M", "F")))

tab2
```

---

## matrix_form *Transform rtable to a list of matrices which can be used for outputting*

---

### Description

Although rtables are represented as a tree data structure when outputting the table to ASCII or HTML it is useful to map the rtable to an in between state with the formatted cells in a matrix form.

### Usage

```
matrix_form(tt, indent_rownames = FALSE)
```

### Arguments

| | |
|---|---|
| tt | TableTree (or related class). A TableTree object representing a populated table. |
| indent_rownames | logical(1), if TRUE the column with the row names in the `strings` matrix of has indented row names (strings pre-fixed) |

## Details

The strings in the return object are defined as follows: row labels are those determined by `summarize_rows` and cell values are determined using `get_formatted_cells`. (Column labels are calculated using a non-exported internal funciton.

## Value

A list with the following elements:

**strings** The content, as it should be printed, of the top-left material, column headers, row labels , and cell values of `tt`

**spans** The column-span information for each print-string in the strings matrix

**aligns** The text alignment for each print-string in the strings matrix

**display** Whether each print-string in the strings matrix should be printed or not.

**row_info** the data.frame generated by `summarize_rows(tt)`

With an additional `nrow_header` attribute indicating the number of pseudo "rows" the column structure defines.

## Examples

```
library(dplyr)

iris2 <- iris %>%
  group_by(Species) %>%
  mutate(group = as.factor(rep_len(c("a", "b"), length.out = n()))) %>%
  ungroup()

l <- basic_table() %>%
  split_cols_by("Species") %>%
  split_cols_by("group") %>%
 analyze(c("Sepal.Length", "Petal.Width"), afun = list_wrap_x(summary) , format = "xx.xx")

l

tbl <- build_table(l, iris2)

matrix_form(tbl)
```

---

# MultiVarSplit *Split between two or more different variables*

---

## Description

Split between two or more different variables

## Usage

```
MultiVarSplit(
  vars,
  split_label = "",
  varlabels = NULL,
  varnames = NULL,
  cfun = NULL,
  cformat = NULL,
  split_format = NULL,
  split_name = "multivars",
  child_labels = c("default", "visible", "hidden"),
  extra_args = list(),
  indent_mod = 0L,
  cindent_mod = 0L,
  cvar = "",
  cextra_args = list(),
  label_pos = "visible"
)
```

## Arguments

| | |
|---|---|
| vars | character vector. Multiple variable names. |
| split_label | string. Label string to be associated with the table generated by the split. Not to be confused with labels assigned to each child (which are based on the data and type of split during tabulation). |
| varlabels | character vector. Labels for vars |
| varnames | character vector. Names for vars which will appear in pathing. When vars are all unique this will be the variable names. If not, these will be variable names with suffixes as necessary to enforce uniqueness. |
| cfun | list/function/NULL. tabulation function(s) for creating content rows. Must accept x or df as first parameter. Must accept labelstr as the second argument. Can optionally accept all optional arguments accepted by analysis functions. See analyze. |
| cformat | format spec. Format for content rows |
| split_format | FormatSpec. Default format associated with the split being created. |
| split_name | string. Name associiated with this split (for pathing, etc) |
| child_labels | string. One of "default", "visible", "hidden". What should the display behavior be for the labels (ie label rows) of the children of this split. Defaults to "default" which flags the label row as visible only if the child has 0 content rows. |
| extra_args | list. Extra arguments to be passed to the tabulation function. Element position in thte list corresponds to the children of this split. Named elements in the child-specific lists are ignored if they do not match a formal argument of the ttabulation function. |

| | |
|---|---|
| indent_mod | numeric. Modifier for the default indent position for the structure created by this function(subtable, content table, or row) *and all of that structure's children*. Defaults to 0, which corresponds to the unmodified default behavior. |
| cindent_mod | numeric(1). The indent modifier for the content tables generated by this split. |
| cvar | character(1). The variable, if any, which the content function should accept. Defaults to NA. |
| cextra_args | list. Extra arguments to be passed to the content function when tabulating row group summaries. |
| label_pos | character(1). Location the variable label should be displayed, Accepts hidden (default for non-analyze row splits), visible, topleft, and - for analyze splits only - default. For analyze calls, `default` indicates that the variable should be visible if and only if multiple variables are analyzed at the same level of nesting. |

## Value

A `MultiVarSplit` object.

## Author(s)

Gabriel Becker

---

`names,VTableNodeInfo-method`
*Names of a TableTree*

---

## Description

Names of a TableTree

## Usage

```
## S4 method for signature 'VTableNodeInfo'
names(x)

## S4 method for signature 'InstantiatedColumnInfo'
names(x)

## S4 method for signature 'LayoutColTree'
names(x)

## S4 method for signature 'VTableTree'
row.names(x)
```

## Arguments

| | |
|---|---|
| x | the object. |

### Details

For TableTrees with more than one level of splitting in columns, the names are defined to be the top-level split values repped out across the columns that they span.

### Value

The column names of `x`, as defined in the details above.

---

## no_colinfo *Exported for use in tern*

---

### Description

Does the table/row/InstantiatedColumnInfo object contain no column structure information?

### Usage

```
no_colinfo(obj)

## S4 method for signature 'VTableNodeInfo'
no_colinfo(obj)

## S4 method for signature 'InstantiatedColumnInfo'
no_colinfo(obj)
```

### Arguments

| | |
|---|---|
| `obj` | ANY. The object for the accessor to access or modify |

### Value

`TRUE` if the object has no/empty instantiated column information, `FALSE` otherwise.

---

## nrow,VTableTree-method *Table Dimensions*

---

### Description

Table Dimensions

## Usage

```
## S4 method for signature 'VTableTree'
nrow(x)

## S4 method for signature 'TableRow'
nrow(x)

## S4 method for signature 'VTableNodeInfo'
ncol(x)

## S4 method for signature 'TableRow'
ncol(x)

## S4 method for signature 'LabelRow'
ncol(x)

## S4 method for signature 'InstantiatedColumnInfo'
ncol(x)

## S4 method for signature 'VTableNodeInfo'
dim(x)
```

## Arguments

| | |
|---|---|
| x | TableTree or ElementaryTable object |

## Value

the number of rows (nrow), columns (ncol) or both (dim) of the object.

## Examples

```
tbl <- basic_table() %>%
  split_cols_by("ARM") %>%
  analyze(c("SEX", "AGE")) %>%
  build_table(ex_adsl)

dim(tbl)
nrow(tbl)
ncol(tbl)

NROW(tbl)
NCOL(tbl)
```

obj_avar *Row attribute accessors*

## Description

Row attribute accessors

## Usage

```
obj_avar(obj)

## S4 method for signature 'TableRow'
obj_avar(obj)

## S4 method for signature 'ElementaryTable'
obj_avar(obj)

row_cells(obj)

## S4 method for signature 'TableRow'
row_cells(obj)

row_cells(obj) <- value

## S4 replacement method for signature 'TableRow'
row_cells(obj) <- value

row_values(obj)

## S4 method for signature 'TableRow'
row_values(obj)

row_values(obj) <- value

## S4 replacement method for signature 'TableRow'
row_values(obj) <- value

## S4 replacement method for signature 'LabelRow'
row_values(obj) <- value
```

## Arguments

| | |
|---|---|
| obj | ANY. The object for the accessor to access or modify |
| value | The new value |

## Value

various, depending on the accessor called.

---

obj_name *Label and Name accessors*

---

## Description

Label and Name accessors

## Usage

```
obj_name(obj)

## S4 method for signature 'VNodeInfo'
obj_name(obj)

## S4 method for signature 'Split'
obj_name(obj)

obj_name(obj) <- value

## S4 replacement method for signature 'VNodeInfo'
obj_name(obj) <- value

## S4 replacement method for signature 'Split'
obj_name(obj) <- value

obj_label(obj)

## S4 method for signature 'Split'
obj_label(obj)

## S4 method for signature 'ANY'
obj_label(obj)

## S4 method for signature 'TableRow'
obj_label(obj)

## S4 method for signature 'VTableTree'
obj_label(obj)

## S4 method for signature 'ValueWrapper'
obj_label(obj)

obj_label(obj) <- value
```

```
## S4 replacement method for signature 'Split'
obj_label(obj) <- value

## S4 replacement method for signature 'TableRow'
obj_label(obj) <- value

## S4 replacement method for signature 'ValueWrapper'
obj_label(obj) <- value

## S4 replacement method for signature 'ANY'
obj_label(obj) <- value

## S4 replacement method for signature 'VTableTree'
obj_label(obj) <- value
```

## Arguments

| | |
|---|---|
| obj | ANY. The object. |
| value | character(1). The new value |

## Value

the name or label of obj for getters, or obj after modification for setters.

---

# pag_tt_indices *Pagination of a TableTree*

---

## Description

Pagination of a TableTree

## Usage

```
pag_tt_indices(
  tt,
  lpp = 15,
  min_siblings = 2,
  nosplitin = character(),
  colwidths = NULL,
  verbose = FALSE
)

paginate_table(
  tt,
  lpp = 15,
  min_siblings = 2,
```

```
  nosplitin = character(),
  colwidths = NULL,
  verbose = FALSE
)
```

## Arguments

| | |
|---|---|
| tt | TableTree (or related class). A TableTree object representing a populated table. |
| lpp | numeric. Maximum lines per page including (re)printed header and context rows |
| min_siblings | numeric. Minimum sibling rows which must appear on either side of pagination row for a mid-subtable split to be valid. Defaults to 2. |
| nosplitin | character. List of names of sub-tables where page-breaks are not allowed, regardless of other considerations. Defaults to none. |
| colwidths | numeric vector. Column widths for use with vertical pagination. Currently ignored. |
| verbose | logical(1). Should extra debugging messages be shown. Defaults to FALSE. |

## Value

for pag_tt_indices a list of paginated-groups of row-indices of tt. For paginate_table, The subtables defined by subsetting by the indices defined by pag_tt_indices.

## Note

This is our first take on pagination. We will refine pagination in subsequent releases. Currently only pagination in the row space work. Pagination in the column space will be added in the future.

## Examples

```
s_summary <- function(x) {
 if (is.numeric(x)) {
     in_rows(
         "n" = rcell(sum(!is.na(x)), format = "xx"),
         "Mean (sd)" = rcell(c(mean(x, na.rm = TRUE), sd(x, na.rm = TRUE)),
                             format = "xx.xx (xx.xx)"),
         "IQR" = rcell(IQR(x, na.rm = TRUE), format = "xx.xx"),
         "min - max" = rcell(range(x, na.rm = TRUE), format = "xx.xx - xx.xx")
     )
 } else if (is.factor(x)) {

     vs <- as.list(table(x))
     do.call(in_rows, lapply(vs, rcell, format = "xx"))

 } else (
     stop("type not supported")
 )
}
```

```
lyt <- basic_table() %>%
split_cols_by(var = "ARM") %>%
    analyze(c("AGE", "SEX", "BEP01FL", "BMRKR1", "BMRKR2", "COUNTRY"), afun = s_summary)

tbl <- build_table(lyt, ex_adsl)
tbl

nrow(tbl)

row_paths_summary(tbl)

tbls <- paginate_table(tbl)

w_tbls <- propose_column_widths(tbl) # so that we have the same column widths

tmp <- lapply(tbls, print, widths = w_tbls)

tmp <- lapply(tbls, function(tbli) {
  cat(toString(tbli, widths = w_tbls))
  cat("\n\n")
  cat("~~~~ PAGE BREAK ~~~~")
  cat("\n\n")
})
```

---

`path_enriched_df` *Transform TableTree object to Path-Enriched data.frame*

---

## Description

Transform TableTree object to Path-Enriched data.frame

## Usage

```
path_enriched_df(tt, path_fun = collapse_path, value_fun = collapse_values)
```

## Arguments

| | |
|---|---|
| `tt` | TableTree (or related class). A TableTree object representing a populated table. |
| `path_fun` | function. Function to transform paths into single-string row/column names. |
| `value_fun` | function. Functiont to transform cell values into cells of the data.frame. Defaults to `collapse_values` which creates strings where multi-valued cells are collapsed together, separated by `\|`. |

## Value

A data frame of `tt`'s cell values (processed by `value_fun`, with columns named by the full column paths (processed by `path_fun` and an additional `row_path` column with the row paths (processed by by `path_fun`).

## Examples

```
lyt <- basic_table() %>%
  split_cols_by("ARM") %>%
  analyze(c("AGE", "BMRKR2"))

tbl <- build_table(lyt, ex_adsl)
path_enriched_df(tbl)
```

---

propose_column_widths *Propose Column Widths of an* rtable *object*

---

## Description

The row names are also considered a column for the output

## Usage

```
propose_column_widths(x, mat_form = matrix_form(x, indent_rownames = TRUE))
```

## Arguments

| | |
|---|---|
| x | rtable object |
| mat_form | object as created with matrix_form |

## Value

a vector of column widths based on the content of x (or mat_form if explictly provided) for use in printing and, in the future, in pagination.

## Examples

```
library(dplyr)

iris2 <- iris %>%
  group_by(Species) %>%
  mutate(group = as.factor(rep_len(c("a", "b"), length.out = n()))) %>%
  ungroup()

l <- basic_table() %>%
  split_cols_by("Species") %>%
  split_cols_by("group") %>%
 analyze(c("Sepal.Length", "Petal.Width"), afun = list_wrap_x(summary) , format = "xx.xx")

tbl <- build_table(l, iris2)

propose_column_widths(tbl)
```

## prune_table *Recursively prune a TableTree*

### Description

Recursively prune a TableTree

### Usage

```
prune_table(
  tt,
  prune_func = prune_empty_level,
  stop_depth = NA_real_,
  depth = 0
)
```

### Arguments

| | |
|---|---|
| tt | TableTree (or related class). A TableTree object representing a populated table. |
| prune_func | function. A Function to be called on each subtree which returns TRUE if the entire subtree should be removed. |
| stop_depth | numeric(1). The depth after which subtrees should not be checked for pruning. Defaults to NA which indicates pruning should happen at all levels |
| depth | numeric(1). Used internally, not intended to be set by the end user. |

### Value

A TableTree pruned via recursive application of prune_func.

### See Also

prune_empty_level()

## rbindl_rtables *rbind TableTree and related objects*

### Description

rbind TableTree and related objects

## Usage

```
rbindl_rtables(x, gap = 0, check_headers = FALSE)

## S4 method for signature 'VTableNodeInfo'
rbind(..., deparse.level = 1)

## S4 method for signature 'VTableNodeInfo,ANY'
rbind2(x, y)
```

## Arguments

| | |
|---|---|
| x | VTableNodeInfo. TableTree, ElementaryTable or TableRow object. |
| gap | deprecated. Ignored. |
| check_headers | deprecated. Ignored. |
| ... | ANY. Elements to be stacked. |
| deparse.level | numeric(1). Currently Ignored. |
| y | VTableNodeInfo. TableTree, ElementaryTable or TableRow object. |

## Value

A formal table object.

## Examples

```
mtbl <- rtable(
   header = rheader(
    rrow(row.name = NULL, rcell("Sepal.Length", colspan = 2), rcell("Petal.Length", colspan=2)),
      rrow(NULL, "mean", "median", "mean", "median")
   ),
   rrow(
     row.name = "All Species",
     mean(iris$Sepal.Length), median(iris$Sepal.Length),
     mean(iris$Petal.Length), median(iris$Petal.Length),
     format = "xx.xx"
   )
 )

 mtbl2 <- with(subset(iris, Species == 'setosa'), rtable(
   header = rheader(
    rrow(row.name = NULL, rcell("Sepal.Length", colspan = 2), rcell("Petal.Length", colspan=2)),
      rrow(NULL, "mean", "median", "mean", "median")
   ),
   rrow(
     row.name = "Setosa",
     mean(Sepal.Length), median(Sepal.Length),
     mean(Petal.Length), median(Petal.Length),
     format = "xx.xx"
   )
 ))
```

```
 rbind(mtbl, mtbl2)
rbind(mtbl, rrow(), mtbl2)
rbind(mtbl, rrow("aaa"), indent(mtbl2))
```

---

rcell *Cell value constructors*

---

## Description

Construct a cell value and associate formatting, labeling, indenting, and column spanning information with it.

## Usage

```
rcell(
  x,
  format = NULL,
  colspan = 1L,
  label = NULL,
  indent_mod = NULL,
  footnotes = NULL
)

non_ref_rcell(
  x,
  is_ref,
  format = NULL,
  colspan = 1L,
  label = NULL,
  indent_mod = NULL,
  refval = NULL
)
```

## Arguments

| | |
|---|---|
| x | ANY. Cell value |
| format | if FUN does not return a formatted rcell then the format is applied |
| colspan | integer(1). Columnspan value. |
| label | character(1). Label or Null. If non-null, it will be looked at when determining row labels. |
| indent_mod | numeric. Modifier for the default indent position for the structure created by this function(subtable, content table, or row) *and all of that structure's children*. Defaults to 0, which corresponds to the unmodified default behavior. |
| footnotes | list or NULL. Referential footnote messages for the cell. |
| is_ref | logical(1). Are we in the reference column (ie .in_ref_col shoul be passed to this argument) |
| refval | ANY. Value to use when in the reference column. Defaults to NULL |

## Details

non_ref_rcell provides the common *blank for cells in the reference column, this value otherwise*, and should be passed the value of .in_ref_col when it is used.

## Value

An object representing the value within a single cell within a populated table. The underlying structure of this object is an implementation detail and sholud not be relied upon beyond calling accessors for the class.

## Note

currently column spanning is only supported for defining header structure.

---

# remove_split_levels *Split functions*

---

## Description

Split functions

## Usage

```
remove_split_levels(excl)

keep_split_levels(only, reorder = TRUE)

drop_split_levels(df, spl, vals = NULL, labels = NULL, trim = FALSE)

drop_and_remove_levels(excl)

reorder_split_levels(neworder, newlabels = neworder, drlevels = TRUE)

trim_levels_in_group(innervar, drop_outlevs = TRUE)
```

## Arguments

| | |
|---|---|
| excl | character. Levels to be excluded (they will not be reflected in the resulting table structure regardless of presence in the data). |
| only | character. Levels to retain (all others will be dropped). |
| reorder | logical(1). Should the order of only be used as the order of the children of the split. defaults to TRUE |
| df | dataset (data.frame or tibble) |
| spl | A Split object defining a partitioning or analysis/tabulation of the data. |
| vals | ANY. For internal use only. |

| | |
|---|---|
| labels | character. Labels to use for the remaining levels instead of the existing ones. |
| trim | logical(1). Should splits corresponding with 0 observations be kept when tabulating. |
| neworder | character. New order or factor levels. |
| newlabels | character. Labels for (new order of) factor levels |
| drlevels | logical(1). Should levels in the data which do not appear in neworder be dropped. Defaults to TRUE |
| innervar | character(1). Variable whose factor levels should be trimmed (e.g., empty levels dropped) *separately within each grouping defined at this point in the structure* |
| drop_outlevs | logical(1). Should empty levels in the variable being split on (ie the 'outer' variable, not innervar) be dropped? Defaults to TRUE |

## Value

a closure suitable for use as a splitting function (splfun) when creating a table layout

## Custom Splitting Function Details

User-defined custom split functions can perform any type of computation on the incoming data provided that they meed the contract for generating 'splits' of the incoming data 'based on' the split object.

Split functions are functions that accept:

**df** data.frame of incoming data to be split

**spl** a Split object. this is largely an internal detail custom functions will not need to worry about, but obj_name(spl), for example, will give the name of the split as it will appear in paths in the resulting table

**vals** Any pre-calculated values. If given non-null values, the values returned should match these. Should be NULL in most cases and can likely be ignored

**labels** Any pre-calculated value labels. Same as above for values

**trim** If TRUE, resulting splits that are empty should be removed

**(Optional) .spl_context** a data.frame describing previously performed splits which collectively arrived at df

The function must then output a named list with the following elements:

**values** The vector of all values corresponding to the splits of df

**datasplit** a list of data.frames representing the groupings of the actual observations from df.

**labels** a character vector giving a string label for each value listed in the values element above

**(Optional) extras** If present, extra arguments to be passed to summary and analysis functions whenever they are executed on the corresponding element of datasplit or a subset thereof

One way to generate custom splitting functions is to wrap existing split functions and modify either the incoming data before they are called, or their outputs.

## Examples

```
l <- basic_table() %>%
  split_cols_by("ARM") %>%
 split_rows_by("COUNTRY", split_fun = remove_split_levels(c("USA", "CAN", "CHE", "BRA"))) %>%
  analyze("AGE")

build_table(l, DM)

l <- basic_table() %>%
  split_cols_by("ARM") %>%
  split_rows_by("COUNTRY", split_fun = keep_split_levels(c("USA", "CAN", "BRA"))) %>%
  analyze("AGE")

build_table(l, DM)
l <- basic_table() %>%
  split_cols_by("ARM") %>%
  split_rows_by("SEX", split_fun = drop_split_levels) %>%
  analyze("AGE")

build_table(l, DM)
l <- basic_table() %>%
  split_cols_by("ARM") %>%
  split_rows_by("SEX", split_fun = drop_and_remove_levels(c("M", "U"))) %>%
  analyze("AGE")

build_table(l, DM)
```

---

# rheader *Create a header*

---

## Description

Create a header

## Usage

```
rheader(..., format = "xx", .lst = NULL)
```

## Arguments

| | |
|---|---|
| `...` | row specifications (either as character vectors or the output from `rrow` or `DataRow`, `LabelRow`, etc. |
| `format` | if `FUN` does not return a formatted `rcell` then the `format` is applied |
| `.lst` | list. An already-collected list of arguments tot be used instead of the elements of `....` Arguments passed via `...` will be ignored if this is specified. |

## Value

a `InstantiatedColumnInfo` object.

## Examples

```
h1 <- rheader(c("A", "B", "C"))

h2 <- rheader(
  rrow(NULL, rcell("group 1", colspan = 2), rcell("group 2", colspan = 2)),
  rrow(NULL, "A", "B", "A", "B")
)

h1

h2
```

---

`row_footnotes` *Referential Footnote Accessors*

---

## Description

Get and set referential footnotes on aspects of a built table

## Usage

```
row_footnotes(obj)

## S4 method for signature 'TableRow'
row_footnotes(obj)

## S4 method for signature 'RowsVerticalSection'
row_footnotes(obj)

row_footnotes(obj) <- value

## S4 replacement method for signature 'TableRow'
row_footnotes(obj) <- value

## S4 method for signature 'ElementaryTable'
row_footnotes(obj)

cell_footnotes(obj)

## S4 method for signature 'CellValue'
cell_footnotes(obj)

## S4 method for signature 'TableRow'
cell_footnotes(obj)
```

```
## S4 method for signature 'LabelRow'
cell_footnotes(obj)

## S4 method for signature 'ElementaryTable'
cell_footnotes(obj)

cell_footnotes(obj) <- value

## S4 replacement method for signature 'CellValue'
cell_footnotes(obj) <- value

## S4 replacement method for signature 'DataRow'
cell_footnotes(obj) <- value

## S4 replacement method for signature 'ContentRow'
cell_footnotes(obj) <- value

col_fnotes_here(obj)

## S4 method for signature 'LayoutColTree'
col_fnotes_here(obj)

## S4 method for signature 'LayoutColLeaf'
col_fnotes_here(obj)

col_fnotes_here(obj) <- value

## S4 replacement method for signature 'LayoutColTree'
col_fnotes_here(obj) <- value

## S4 replacement method for signature 'LayoutColLeaf'
col_fnotes_here(obj) <- value

ref_index(obj)

## S4 method for signature 'RefFootnote'
ref_index(obj)

ref_index(obj) <- value

## S4 replacement method for signature 'RefFootnote'
ref_index(obj) <- value

fnotes_at_path(obj, rowpath = NULL, colpath = NULL, reset_idx = TRUE) <- value

## S4 replacement method for signature 'VTableTree,character'
fnotes_at_path(obj, rowpath = NULL, colpath = NULL, reset_idx = TRUE) <- value
```

```
## S4 replacement method for signature 'VTableTree,`NULL`'
fnotes_at_path(obj, rowpath = NULL, colpath = NULL, reset_idx = TRUE) <- value
```

### Arguments

| | |
|---|---|
| obj | ANY. The object for the accessor to access or modify |
| value | The new value |
| rowpath | character or NULL. Path within row structure. NULL indicates the footnote should go on the column rather than cell. |
| colpath | character or NULL. Path within column structure. NULL indicates footnote should go on the row rather than cell |
| reset_idx | logical(1). Should the numbering for referential footnotes be immediately recalculated. Defaults to TRUE. |

---

## row_paths *Return List with Table Row/Col Paths*

---

### Description

Return List with Table Row/Col Paths

### Usage

```
row_paths(x)

col_paths(x)
```

### Arguments

| | |
|---|---|
| x | an rtable object |

### Value

a list of paths to each row/column within x

### Examples

```
tbl <- basic_table() %>%
  split_cols_by("ARM") %>%
  analyze(c("SEX", "AGE")) %>%
  build_table(ex_adsl)

tbl

row_paths(tbl)
col_paths(tbl)

cell_values(tbl, c("AGE", "Mean"),  c("ARM", "B: Placebo"))
```

`row_paths_summary` *Print Row/Col Paths Summary*

## Description

Print Row/Col Paths Summary

## Usage

```
row_paths_summary(x)

col_paths_summary(x)
```

## Arguments

| | |
|---|---|
| `x` | an rtable object |

## Value

A data.frame summarizing the row- or column-structure of `x`.

## Examples

```
library(dplyr)

ex_adsl_MF <- ex_adsl %>% filter(SEX %in% c("M", "F"))

tbl <- basic_table() %>%
  split_cols_by("ARM") %>%
  split_cols_by("SEX", split_fun = drop_split_levels) %>%
  analyze(c("AGE", "BMRKR2")) %>%
  build_table(ex_adsl_MF)

tbl

df <- row_paths_summary(tbl)

df

col_paths_summary(tbl)

# manually constructed table
tbl3 <- rtable(
   rheader(
     rrow("row 1", rcell("a", colspan = 2),
     rcell("b", colspan = 2)
   ),
   rrow("h2", "a", "b", "c", "d")),
   rrow("r1", 1, 2, 1, 2), rrow("r2", 3, 4, 2,1)
```

```
)
col_paths_summary(tbl3)
```

---

`rrow` *row*

---

## Description

row

## Usage

```
rrow(row.name = "", ..., format = NULL, indent = 0)
```

## Arguments

| | |
|---|---|
| `row.name` | if `NULL` then the `FUN` argument is deparsed and used as `row.name` of the `rrow` |
| `...` | cell values |
| `format` | if `FUN` does not return a formatted `rcell` then the `format` is applied |
| `indent` | deprecated. |

## Value

A row object of the context-appropriate type (label or data)

## Examples

```
rrow("ABC", c(1,2), c(3,2), format = "xx (xx.%)")
rrow("")
```

---

`rrowl` *rrowl*

---

## Description

rrowl

## Usage

```
rrowl(row.name, ..., format = NULL, indent = 0)
```

### Arguments

| | |
|---|---|
| row.name | if NULL then the FUN argument is deparsed and used as row.name of the rrow |
| ... | values in vector/list form |
| format | if FUN does not return a formatted rcell then the format is applied |
| indent | deprecated. |

### Value

A row object of the context-appropriate type (label or data)

### Examples

```
rrowl("a", c(1,2,3), format = "xx")
rrowl("a", c(1,2,3), c(4,5,6), format = "xx")


rrowl("N", table(iris$Species))
rrowl("N", table(iris$Species), format = "xx")

x <- tapply(iris$Sepal.Length, iris$Species, mean, simplify = FALSE)

rrow(row.name = "row 1", x)
rrow("ABC", 2, 3)

rrowl(row.name = "row 1", c(1, 2), c(3,4))
rrow(row.name = "row 2", c(1, 2), c(3,4))
```

---

## rtable *Create a Table*

---

### Description

Create a Table

### Usage

```
rtable(header, ..., format = NULL)

rtablel(header, ..., format = NULL)
```

### Arguments

| | |
|---|---|
| header | Information defining the header (column strucure) of the table. This can be as row objects (legacy), character vectors or a InstantiatedColumnInfo object. |
| ... | Rows to place in the table. |
| format | if FUN does not return a formatted rcell then the format is applied |

## Value

a formal table object of the appropriate type (`ElementaryTable` or `TableTree`)

## See Also

Other compatability: `insert_rrow()`

## Examples

```
rtable(
  header = LETTERS[1:3],
  rrow("one to three", 1, 2, 3),
  rrow("more stuff", rcell(pi, format = "xx.xx"), "test", "and more")
)


# Table with multirow header
sel <- iris$Species == "setosa"
mtbl <- rtable(
  header = rheader(
    rrow(row.name = NULL, rcell("Sepal.Length", colspan = 2),
         rcell("Petal.Length", colspan=2)),
    rrow(NULL, "mean", "median", "mean", "median")
  ),
  rrow(
    row.name = "All Species",
    mean(iris$Sepal.Length), median(iris$Sepal.Length),
    mean(iris$Petal.Length), median(iris$Petal.Length),
    format = "xx.xx"
  ),
  rrow(
    row.name = "Setosa",
    mean(iris$Sepal.Length[sel]), median(iris$Sepal.Length[sel]),
    mean(iris$Petal.Length[sel]), median(iris$Petal.Length[sel])
  )
)

mtbl

names(mtbl) # always first row of header

# Single row header

tbl <- rtable(
  header = c("Treatement\nN=100", "Comparison\nN=300"),
  format = "xx (xx.xx%)",
  rrow("A", c(104, .2), c(100, .4)),
  rrow("B", c(23, .4), c(43, .5)),
  rrow(""),
  rrow("this is a very long section header"),
  rrow("estimate", rcell(55.23, "xx.xx", colspan = 2)),
```

```
  rrow("95% CI", indent = 1, rcell(c(44.8, 67.4), format = "(xx.x, xx.x)", colspan = 2))
)
tbl

row.names(tbl)
names(tbl)


# Subsetting
tbl[1, ]
tbl[, 1]

tbl[1,2]
tbl[2, 1]

tbl[3,2]
tbl[5,1]
tbl[5,2]

# # Data Structure methods
dim(tbl)
nrow(tbl)
ncol(tbl)
names(tbl)


# Colspans

tbl2 <- rtable(
  c("A", "B", "C", "D", "E"),
  format = "xx",
  rrow("r1", 1, 2, 3, 4, 5),

  rrow("r2", rcell("sp2", colspan = 2), "sp1", rcell("sp2-2", colspan = 2))
)

tbl2
```

---

`select_all_levels` *Add Combination Levels to split*

---

## Description

Add Combination Levels to split

## Usage

```
select_all_levels

add_combo_levels(combosdf, trim = FALSE, first = FALSE, keep_levels = NULL)
```

## Arguments

| | |
|---|---|
| combosdf | data.frame/tbl_df. Columns valname, label, levelcombo, exargs. Of which levelcombo and exargs are list columns. Passing the select_all_levels object as a value in the comblevels column indicates that an overall/all-observations level should be created. |
| trim | logical(1). Should splits corresponding with 0 observations be kept when tabulating. |
| first | logical(1). Should the created split level be placed first in the levels (TRUE) or last (FALSE, the default). |
| keep_levels | character or NULL. If non-NULL, the levels to retain across both combination and individual levels. |

## Format

An object of class AllLevelsSentinel of length 0.

## Value

a closure suitable for use as a splitting function (splfun) when creating a table layout

## Note

Analysis or summary functions for which the order matters should never be used within the tabulation framework.

## Examples

```
library(tibble)
combodf <- tribble(
    ~valname, ~label, ~levelcombo, ~exargs,
    "A_B", "Arms A+B", c("A: Drug X", "B: Placebo"), list(),
    "A_C", "Arms A+C", c("A: Drug X", "C: Combination"), list())

l <- basic_table() %>%
    split_cols_by("ARM", split_fun = add_combo_levels(combodf)) %>%
    add_colcounts() %>%
    analyze("AGE")

build_table(l, DM)

la <- basic_table() %>%
   split_cols_by("ARM", split_fun = add_combo_levels(combodf, keep_levels = c("A_B", "A_C"))) %>%
    add_colcounts() %>%
    analyze("AGE")

build_table(la, DM)

smallerDM <- droplevels(subset(DM, SEX %in% c("M", "F") &
                         grepl("^(A|B)", ARM)))
l2 <- basic_table() %>%
```

```
    split_cols_by("ARM", split_fun = add_combo_levels(combodf[1,])) %>%
    split_cols_by("SEX", split_fun = add_overall_level("SEX_ALL", "All Genders")) %>%
    add_colcounts() %>%
    analyze("AGE")

l3 <-  basic_table() %>%
    split_cols_by("ARM", split_fun = add_combo_levels(combodf)) %>%
    add_colcounts() %>%
    split_rows_by("SEX", split_fun = add_overall_level("SEX_ALL", "All Genders")) %>%
    summarize_row_groups() %>%
    analyze("AGE")

build_table(l3, smallerDM)
```

---

`sf_args` *Split Function Arg Conventions*

---

## Description

Split Function Arg Conventions

## Usage

```
sf_args(trim, label, first)
```

## Arguments

| | |
|---|---|
| `trim` | logical(1). Should splits corresponding with 0 observations be kept when tabulating. |
| `label` | character(1). A label (not to be confused with the name) for the object/structure. |
| `first` | logical(1). Should the created split level be placed first in the levels (`TRUE`) or last (`FALSE`, the default). |

## Value

NULL (this is an argument template dummy function)

## See Also

Other conventions: `compat_args()`, `constr_args()`, `gen_args()`, `lyt_args()`

`simple_analysis` *Default tabulation*

## Description

This function is used when `analyze` is invoked

## Usage

```
simple_analysis(x, ...)

## S4 method for signature 'numeric'
simple_analysis(x, ...)

## S4 method for signature 'logical'
simple_analysis(x, ...)

## S4 method for signature 'factor'
simple_analysis(x, ...)

## S4 method for signature 'ANY'
simple_analysis(x, ...)
```

## Arguments

| | |
|---|---|
| `x` | the *already split* data being tabulated for a particular cell/set of cells |
| `...` | passed on directly |

## Details

This function has the following behavior given particular types of inputs:

**numeric** calls `mean` on `x`

**logical** calls `sum` on `x`

**factor** calls `length` on `x`

`in_rows` is called on the resulting value(s).

All other classes of input currently lead to an error.

## Value

an `RowsVerticalSection` object (or `NULL`). The details of this object should be considered an internal implementation detail.

## Author(s)

Gabriel Becker and Adrian Waddell

## Examples

```
simple_analysis(1:3)
simple_analysis(iris$Species)
simple_analysis(iris$Species == "setosa")
```

---

`sort_at_path` *Sort substructure of a TableTree at a particular Path in the Tree.*

---

## Description

Sort substructure of a TableTree at a particular Path in the Tree.

## Usage

```
sort_at_path(
  tt,
  path,
  scorefun,
  decreasing = NA,
  na.pos = c("omit", "last", "first")
)
```

## Arguments

| | |
|---|---|
| `tt` | TableTree (or related class). A TableTree object representing a populated table. |
| `path` | character. A vector path for a position within the structure of a tabletree. Each element represents a subsequent choice amongst the children of the previous choice. |
| `scorefun` | function. Scoring function, should accept the type of children directly under the position at `path` (either VTableTree, VTableRow, or VTableNodeInfo, which covers both) and return a numeric value to be sorted. |
| `decreasing` | logical(1). Should the the scores generated by `scorefun` be sorted in decreasing order. If unset (the default of `NA`), it is set to `TRUE` if the generated scores are numeric and `FALSE` if they are characters. |
| `na.pos` | character(1). What should be done with children (subtrees/rows) with `NA` scores. Defaults to `"omit"`, which removes them, other allowed values are `"last"` and `"first"` which indicate where they should be placed in the order. |

## Details

The `path` here can include `"*"` as a step, which means taht each child at that step will be *separately* sorted based on `scorefun` and the remaining `path` entries. This can occur multiple times in a path.

## Value

A TableTree with the same structure as `tt` with the exception that the requested sorting has been done at `path`

split_cols_by *Declaring a column-split based on levels of a variable*

## Description

Will generate children for each subset of a categorical variable

## Usage

```
split_cols_by(
  lyt,
  var,
  labels_var = var,
  split_label = var,
  split_fun = NULL,
  format = NULL,
  nested = TRUE,
  child_labels = c("default", "visible", "hidden"),
  extra_args = list(),
  ref_group = NULL
)
```

## Arguments

| | |
|---|---|
| lyt | layout object pre-data used for tabulation |
| var | string, variable name |
| labels_var | string, name of variable containing labels to be displayed for the values of var |
| split_label | string. Label string to be associated with the table generated by the split. Not to be confused with labels assigned to each child (which are based on the data and type of split during tabulation). |
| split_fun | function/NULL. custom splitting function See custom_split_funs |
| format | FormatSpec. Format associated with this split. Formats can be declared via strings ("xx.x") or function. In cases such as analyze calls, they can character vectors or lists of functions. |
| nested | boolean, Add this as a new top-level split (defining a new subtable directly under root). Defaults to FALSE |
| child_labels | string. One of "default", "visible", "hidden". What should the display behavior be for the labels (ie label rows) of the children of this split. Defaults to "default" which flags the label row as visible only if the child has 0 content rows. |
| extra_args | list. Extra arguments to be passed to the tabulation function. Element position in thte list corresponds to the children of this split. Named elements in the child-specific lists are ignored if they do not match a formal argument of the ttabulation function. |
| ref_group | character(1) or NULL. Level of var which should be considered ref_group/reference |

## Value

A `PreDataTableLayouts` object suitable for passing to further layouting functions, and to `build_table`.

## Custom Splitting Function Details

User-defined custom split functions can perform any type of computation on the incoming data provided that they meed the contract for generating 'splits' of the incoming data 'based on' the split object.

Split functions are functions that accept:

**df** data.frame of incoming data to be split

**spl** a Split object. this is largely an internal detail custom functions will not need to worry about, but `obj_name(spl)`, for example, will give the name of the split as it will appear in paths in the resulting table

**vals** Any pre-calculated values. If given non-null values, the values returned should match these. Should be NULL in most cases and can likely be ignored

**labels** Any pre-calculated value labels. Same as above for `values`

**trim** If `TRUE`, resulting splits that are empty should be removed

**(Optional) .spl_context** a data.frame describing previously performed splits which collectively arrived at `df`

The function must then output a `named list` with the following elements:

**values** The vector of all values corresponding to the splits of `df`

**datasplit** a list of data.frames representing the groupings of the actual observations from `df`.

**labels** a character vector giving a string label for each value listed in the `values` element above

**(Optional) extras** If present, extra arguments to be passed to summary and analysis functions whenever they are executed on the corresponding element of `datasplit` or a subset thereof

One way to generate custom splitting functions is to wrap existing split functions and modify either the incoming data before they are called, or their outputs.

## Author(s)

Gabriel Becker

## Examples

```
lyt <- basic_table() %>%
  split_cols_by("ARM") %>%
  analyze(c("AGE", "BMRKR2"))

build_table(lyt, ex_adsl)

# Let's look at the splits in more detail

l <- basic_table() %>% split_cols_by("ARM")
```

```
l

# add an analysis (summary)
l2 <- l %>%
    analyze(c("AGE", "COUNTRY"), afun = list_wrap_x(summary) , format = "xx.xx")
l2

build_table(l2, DM)

# By default sequentially adding layouts results in nesting
library(dplyr)
DM_MF <- DM %>% filter(SEX %in% c("M", "F")) %>% mutate(SEX = droplevels(SEX))

l3 <- basic_table() %>% split_cols_by("ARM") %>%
  split_cols_by("SEX") %>%
  analyze(c("AGE", "COUNTRY"), afun = list_wrap_x(summary), format = "xx.xx")
l3

 build_table(l3, DM_MF)

# nested=TRUE vs not
l4 <- basic_table() %>% split_cols_by("ARM") %>%
 split_rows_by("SEX", split_fun = drop_split_levels) %>%
 split_rows_by("RACE", split_fun = drop_split_levels) %>%
 analyze("AGE")

l4
build_table(l4, DM)

l5 <- basic_table() %>% split_cols_by("ARM") %>%
 split_rows_by("SEX", split_fun= drop_split_levels) %>%
 analyze("AGE") %>%
 split_rows_by("RACE", nested=FALSE, split_fun = drop_split_levels) %>%
 analyze("AGE")

l5
build_table(l5, DM)
```

---

`split_cols_by_cuts` *Split on static or dynamic cuts of the data*

---

## Description

Create columns (or row splits) based on values (such as quartiles) of `var`.

## Usage

```
split_cols_by_cuts(
```

```
  lyt,
  var,
  cuts,
  cutlabels = NULL,
  split_label = var,
  nested = TRUE,
  cumulative = FALSE
)

split_rows_by_cuts(
  lyt,
  var,
  cuts,
  cutlabels = NULL,
  split_label = var,
  nested = TRUE,
  cumulative = FALSE,
  label_pos = "hidden"
)

split_cols_by_cutfun(
  lyt,
  var,
  cutfun = qtile_cuts,
  cutlabelfun = function(x) NULL,
  split_label = var,
  format = NULL,
  nested = TRUE,
  extra_args = list(),
  cumulative = FALSE
)

split_cols_by_quartiles(
  lyt,
  var,
  split_label = var,
  format = NULL,
  nested = TRUE,
  extra_args = list(),
  cumulative = FALSE
)

split_rows_by_quartiles(
  lyt,
  var,
  split_label = var,
  format = NULL,
  nested = TRUE,
```

```
  child_labels = c("default", "visible", "hidden"),
  extra_args = list(),
  cumulative = FALSE,
  indent_mod = 0L,
  label_pos = "hidden"
)

split_rows_by_cutfun(
  lyt,
  var,
  cutfun = qtile_cuts,
  cutlabelfun = function(x) NULL,
  split_label = var,
  format = NULL,
  nested = TRUE,
  child_labels = c("default", "visible", "hidden"),
  extra_args = list(),
  cumulative = FALSE,
  indent_mod = 0L,
  label_pos = "hidden"
)
```

## Arguments

| | |
|---|---|
| lyt | layout object pre-data used for tabulation |
| var | string, variable name |
| cuts | numeric. Cuts to use |
| cutlabels | character (or NULL). Labels for the cutst |
| split_label | string. Label string to be associated with the table generated by the split. Not to be confused with labels assigned to each child (which are based on the data and type of split during tabulation). |
| nested | boolean, Add this as a new top-level split (defining a new subtable directly under root). Defaults to FALSE |
| cumulative | logical. Should the cuts be treated as cumulative. Defaults to FALSE |
| label_pos | character(1). Location the variable label should be displayed, Accepts hidden (default for non-analyze row splits), visible, topleft, and - for analyze splits only - default. For analyze calls, default indicates that the variable should be visible if and only if multiple variables are analyzed at the same level of nesting. |
| cutfun | function. Function which accepts the full vector of var values and returns cut points to be passed to cut. |
| cutlabelfun | function. Function which returns either labels for the cuts or NULL when passed the return value of cutfun |
| format | FormatSpec. Format associated with this split. Formats can be declared via strings ("xx.x") or function. In cases such as analyze calls, they can character vectors or lists of functions. |

| | |
|---|---|
| extra_args | list. Extra arguments to be passed to the tabulation function. Element position in thte list corresponds to the children of this split. Named elements in the child-specific lists are ignored if they do not match a formal argument of the ttabulation function. |
| child_labels | string. One of "default", "visible", "hidden". What should the display behavior be for the labels (ie label rows) of the children of this split. Defaults to "default" which flags the label row as visible only if the child has 0 content rows. |
| indent_mod | numeric. Modifier for the default indent position for the structure created by this function(subtable, content table, or row) *and all of that structure's children*. Defaults to 0, which corresponds to the unmodified default behavior. |

## Details

For dynamic cuts, the cut is transformed into a static cut by build_table *based on the full dataset*, before proceeding. Thus even when nested within another split in column/row space, the resulting split will reflect the overall vaalues (e.g., quartiles) in the dataset, NOT the values for subset it is nested under.

## Value

A PreDataTableLayouts object suitable for passing to further layouting functions, and to build_table.

## Author(s)

Gabriel Becker

## Examples

```
library(dplyr)

# split_cols_by_cuts
l <- basic_table() %>%
    split_cols_by("ARM") %>%
    split_cols_by_cuts("AGE", split_label = "Age",
                       cuts = c(0, 25, 35, 1000),
                       cutlabels = c("young", "medium", "old")) %>%
    analyze(c("BMRKR2", "STRATA2")) %>%
    append_topleft("counts")

build_table(l, ex_adsl)


# split_rows_by_cuts
l <- basic_table() %>%
    split_cols_by("ARM") %>%
    split_rows_by_cuts("AGE", split_label = "Age",
                  cuts = c(0, 25, 35, 1000),
                  cutlabels = c("young", "medium", "old")) %>%
    analyze(c("BMRKR2", "STRATA2")) %>%
    append_topleft("counts")
```

```
build_table(l, ex_adsl)


# split_cols_by_quartiles

l <- basic_table() %>%
    split_cols_by("ARM") %>%
    split_cols_by_quartiles("AGE", split_label = "Age") %>%
    analyze(c("BMRKR2", "STRATA2")) %>%
    append_topleft("counts")

build_table(l, ex_adsl)

# split_rows_by_quartiles
l <- basic_table() %>%
    split_cols_by("ARM") %>%
    add_colcounts() %>%
    split_rows_by_quartiles("AGE", split_label = "Age") %>%
    analyze("BMRKR2") %>%
    append_topleft(c("Age Quartiles", " Counts BMRKR2"))

build_table(l, ex_adsl)
```

---

`split_cols_by_multivar` *Associate Multiple Variables with Columns*

---

## Description

In some cases, the variable to be ultimately analyzed is most naturally defined on a column, not a row basis. When we need columns to reflect different variables entirely, rather than different levels of a single variable, we use `split_cols_by_multivar`

## Usage

```
split_cols_by_multivar(
  lyt,
  vars,
  varlabels = vars,
  varnames = NULL,
  nested = TRUE
)
```

## Arguments

| | |
|---|---|
| lyt | layout object pre-data used for tabulation |
| vars | character vector. Multiple variable names. |
| varlabels | character vector. Labels for vars |
| varnames | character vector. Names for vars which will appear in pathing. When vars are all unique this will be the variable names. If not, these will be variable names with suffixes as necessary to enforce uniqueness. |
| nested | boolean, Add this as a new top-level split (defining a new subtable directly under root). Defaults to FALSE |

## Value

A PreDataTableLayouts object suitable for passing to further layouting functions, and to build_table.

## Author(s)

Gabriel Becker

## See Also

analyze_colvars

## Examples

```
library(dplyr)
ANL <- DM %>% mutate(value = rnorm(n()), pctdiff = runif(n()))

## toy example where we take the mean of the first variable and the
## count of >.5 for the second.
colfuns <- list(function(x) in_rows(mean = mean(x), .formats = "xx.x"),
                function(x) in_rows("# x > 5" = sum(x > .5), .formats = "xx"))

l <- basic_table() %>%
    split_cols_by("ARM") %>%
    split_cols_by_multivar(c("value", "pctdiff")) %>%
    split_rows_by("RACE", split_label = "ethnicity", split_fun = drop_split_levels) %>%
    summarize_row_groups() %>%
    analyze_colvars(afun = colfuns)

l

build_table(l, ANL)
```

split_rows_by *Add Rows according to levels of a variable*

## Description

Add Rows according to levels of a variable

## Usage

```
split_rows_by(
  lyt,
  var,
  labels_var = var,
  split_label = var,
  split_fun = NULL,
  format = NULL,
  nested = TRUE,
  child_labels = c("default", "visible", "hidden"),
  label_pos = "hidden",
  indent_mod = 0L
)
```

## Arguments

| | |
|---|---|
| lyt | layout object pre-data used for tabulation |
| var | string, variable name |
| labels_var | string, name of variable containing labels to be displayed for the values of var |
| split_label | string. Label string to be associated with the table generated by the split. Not to be confused with labels assigned to each child (which are based on the data and type of split during tabulation). |
| split_fun | function/NULL. custom splitting function See custom_split_funs |
| format | FormatSpec. Format associated with this split. Formats can be declared via strings ("xx.x") or function. In cases such as analyze calls, they can character vectors or lists of functions. |
| nested | boolean, Add this as a new top-level split (defining a new subtable directly under root). Defaults to FALSE |
| child_labels | string. One of "default", "visible", "hidden". What should the display behavior be for the labels (ie label rows) of the children of this split. Defaults to "default" which flags the label row as visible only if the child has 0 content rows. |
| label_pos | character(1). Location the variable label should be displayed, Accepts hidden (default for non-analyze row splits), visible, topleft, and - for analyze splits only - default. For analyze calls, default indicates that the variable should be visible if and only if multiple variables are analyzed at the same level of nesting. |

`indent_mod` numeric. Modifier for the default indent position for the structure created by this function(subtable, content table, or row) *and all of that structure's children*. Defaults to 0, which corresponds to the unmodified default behavior.

## Value

A `PreDataTableLayouts` object suitable for passing to further layouting functions, and to `build_table`.

## Custom Splitting Function Details

User-defined custom split functions can perform any type of computation on the incoming data provided that they meed the contract for generating 'splits' of the incoming data 'based on' the split object.

Split functions are functions that accept:

**df** data.frame of incoming data to be split

**spl** a Split object. this is largely an internal detail custom functions will not need to worry about, but `obj_name(spl)`, for example, will give the name of the split as it will appear in paths in the resulting table

**vals** Any pre-calculated values. If given non-null values, the values returned should match these. Should be NULL in most cases and can likely be ignored

**labels** Any pre-calculated value labels. Same as above for `values`

**trim** If `TRUE`, resulting splits that are empty should be removed

**(Optional) .spl_context** a data.frame describing previously performed splits which collectively arrived at `df`

The function must then output a `named list` with the following elements:

**values** The vector of all values corresponding to the splits of `df`

**datasplit** a list of data.frames representing the groupings of the actual observations from `df`.

**labels** a character vector giving a string label for each value listed in the `values` element above

**(Optional) extras** If present, extra arguments to be passed to summary and analysis functions whenever they are executed on the corresponding element of `datasplit` or a subset thereof

One way to generate custom splitting functions is to wrap existing split functions and modify either the incoming data before they are called, or their outputs.

## Note

If `var` is a factor with empty unobserved levels and `labels_var` is specified, it must also be a factor with the same number of levels as `var`. Currently the error that occurs when this is not hte case is not very informative, but that will change in the future.

## Author(s)

Gabriel Becker

## Examples

```
l <- basic_table() %>%
    split_cols_by("ARM") %>%
    split_rows_by("RACE", split_fun = drop_split_levels) %>%
    analyze("AGE", mean, var_labels = "Age", format = "xx.xx")

build_table(l, DM)


basic_table() %>%
    split_cols_by("ARM") %>%
    split_rows_by("RACE") %>%
    analyze("AGE", mean, var_labels = "Age", format = "xx.xx") %>%
    build_table(DM)


l <- basic_table() %>%
    split_cols_by("ARM") %>%
    split_cols_by("SEX") %>%
    summarize_row_groups(label_fstr = "Overall (N)") %>%
    split_rows_by("RACE", split_label = "Ethnicity", labels_var = "ethn_lab",
                  split_fun = drop_split_levels) %>%
    summarize_row_groups("RACE", label_fstr = "%s (n)") %>%
    analyze("AGE", var_labels = "Age", afun = mean, format = "xx.xx")

l

library(dplyr)
DM2 <- DM %>%
    filter(SEX %in% c("M", "F")) %>%
    mutate(
        SEX = droplevels(SEX),
        gender_lab = c("F" = "Female", "M" = "Male",
                       "U" = "Unknown", "UNDIFFERENTIATED" = "Undifferentiated")[SEX],
        ethn_lab = c(
            "ASIAN" = "Asian",
            "BLACK OR AFRICAN AMERICAN" = "Black or African American",
            "WHITE" = "White",
            "AMERICAN INDIAN OR ALASKA NATIVE" = "American Indian or Alaska Native",
             "MULTIPLE" = "Multiple",
             "NATIVE HAWAIIAN OR OTHER PACIFIC ISLANDER" =
                 "Native Hawaiian or Other Pacific Islander",
             "OTHER" = "Other", "UNKNOWN" = "Unknown"
        )[RACE]
    )

build_table(l, DM2)
```

`spl_context` *spl_context within analysis and split functions .spl_context in analysis and split functions*

## Description

spl_context within analysis and split functions

.spl_context in analysis and split functions

## .spl_context Details

The `.spl_context` `data.frame` gives information about the subsets of data corresponding to the splits within-which the current `analyze` action is nested. Taken together, these correspond to the path that the resulting (set of) rows the analysis function is creating, although the information is in a slighlyt different form. Each split (which correspond to groups of rows in the resulting table) is represented via the following columns:

**split** The name of the split (often the variable being split in the simple case)

**value** The string representation of the value at that split

**full_parent_df** a dataframe containing the full data (ie across all columns) corresponding to the path defined by the combination of `split` and `value` of this row *and all rows above this row*

**all_cols_n** the number of observations corresponding to this row grouping (union of all columns)

***(row-split and analyze contexts only)* <1 column for each column in the table structure** These list columns (named the same as `names(col_exprs(tab))`) contain logical vectors corresponding to the subset of this row's `full_parent_df` corresponding to that column

**cur_col_subset** List column containing logical vectors indicating the subset of that row's `full_parent_df` for the column currently being created by the analysis function

**cur_col_n** integer column containing the observation counts for that split

*note Within analysis functions that accept* `.spl_context`*, the* `all_cols_n` *and* `cur_col_n` *columns of the dataframe will contain the 'true' observation counts corresponding to the row-group and row-group x column subsets of the data. These numbers will not, and currently cannot, reflect alternate column observation counts provided by the* `alt_counts_df`, `col_counts` *or* `col_total` *arguments to* `build_table`

`sprintf_format` *Specify a rcell format based on sprintf formattig rules*

## Description

Format the rcell data with `sprintf` formatting strings

## Usage

```
sprintf_format(format)
```

### Arguments

| | |
|---|---|
| format | character(1). A format string passed to sprintf. |

### Value

A formating function which wraps and will apply the specified printf style format string format.

### See Also

sprintf

### Examples

```
basic_table() %>%
  split_cols_by("ARM") %>%
  analyze("AGE", function(x) {
    in_rows(
      "mean_sd" = c(mean(x), sd(x)),
      "range" = range(x),
      .formats = c(mean_sd = sprintf_format("%.4f - %.2f"), range = "xx.xx - xx.xx")
    )
  }) %>%
  build_table(DM)

rcell(100, format = sprintf_format("(N=%i)"))

rcell(c(4,9999999999), format = sprintf_format("(%.2f, >999.9)"))

rtable(LETTERS[1:2], rrow("", 1 ,2), format = sprintf_format("%.2f"))
```

---

## summarize_rows *summarize_rows*

---

### Description

summarize_rows

### Usage

```
summarize_rows(obj)
```

### Arguments

| | |
|---|---|
| obj | VTableTree. |

### Value

A data.frame summarizing the rows in obj.

summarize_row_groups *Add a content row of summary counts*

## Description

Add a content row of summary counts

## Usage

```
summarize_row_groups(
  lyt,
  var = "",
  label_fstr = "%s",
  format = "xx (xx.x%)",
  cfun = NULL,
  indent_mod = 0L,
  extra_args = list()
)
```

## Arguments

| | |
|---|---|
| lyt | layout object pre-data used for tabulation |
| var | string, variable name |
| label_fstr | string. An sprintf style format string containing. For non-comparison splits, it can contain up to one "%s" which takes the current split value and generates the row/column label. Comparison-based splits it can contain up to two "%s". |
| format | FormatSpec. Format associated with this split. Formats can be declared via strings ("xx.x") or function. In cases such as analyze calls, they can character vectors or lists of functions. |
| cfun | list/function/NULL. tabulation function(s) for creating content rows. Must accept x or df as first parameter. Must accept labelstr as the second argument. Can optionally accept all optional arguments accepted by analysis functions. See analyze. |
| indent_mod | numeric. Modifier for the default indent position for the structure created by this function(subtable, content table, or row) *and all of that structure's children*. Defaults to 0, which corresponds to the unmodified default behavior. |
| extra_args | list. Extra arguments to be passed to the tabulation function. Element position in thte list corresponds to the children of this split. Named elements in the child-specific lists are ignored if they do not match a formal argument of the ttabulation function. |

## Details

If format expects 2 values (i.e. xx appears twice in the format string, then both raw and percent of column total counts are calculated. Otherwise only raw counts are used.

cfun must accept df as its first argument and will receive the subset data.frame corresponding with the row- and column-splitting for the cell being calculated. Must accept labelstr as the second parameter, which accepts the label of the level of the parent split currently being summarized. Can additionally take any optional argument supported by analysis functions. (see analyze).

## Value

A PreDataTableLayouts object suitable for passing to further layouting functions, and to build_table.

## Author(s)

Gabriel Becker

## Examples

```
DM2 <- subset(DM, COUNTRY %in% c("USA", "CAN", "CHN"))

l <- basic_table() %>% split_cols_by("ARM") %>%
    split_rows_by("COUNTRY", split_fun = drop_split_levels) %>%
    summarize_row_groups(label_fstr = "%s (n)") %>%
    analyze("AGE", afun = list_wrap_x(summary) , format = "xx.xx")
l

tbl <- build_table(l, DM2)

tbl

row_paths_summary(tbl) # summary count is a content table


## use a cfun and extra_args to customize summarization
## behavior
sfun <- function(x, labelstr, trim) {
    in_rows(
        c(mean(x, trim = trim), trim),
        .formats = "xx.x (xx.x%)",
        .labels = sprintf("%s (Trimmed mean and trim %%)",
                          labelstr)
    )
}

l2 <- basic_table() %>% split_cols_by("ARM") %>%
    split_rows_by("COUNTRY", split_fun = drop_split_levels) %>%
    add_colcounts() %>%
    summarize_row_groups("AGE", cfun = sfun,
                         extra_args = list(trim = .2)) %>%
    analyze("AGE", afun = list_wrap_x(summary) , format = "xx.xx") %>%
    append_topleft(c("Country", "  Age"))

tbl2 <- build_table(l2, DM2)
tbl2
```

table_structure *Summarize Table*

## Description

Summarize Table

## Usage

```
table_structure(x, detail = c("subtable", "row"))
```

## Arguments

| | |
|---|---|
| x | a table object |
| detail | either row or subtable |

## Value

currently no return value. Called for the side-effect of printing a row- or subtable-structure summary of x.

## Examples

```
library(dplyr)

iris2 <- iris %>%
  group_by(Species) %>%
  mutate(group = as.factor(rep_len(c("a", "b"), length.out = n()))) %>%
  ungroup()

l <- basic_table() %>%
  split_cols_by("Species") %>%
  split_cols_by("group") %>%
 analyze(c("Sepal.Length", "Petal.Width"), afun = list_wrap_x(summary) , format = "xx.xx")

tbl <- build_table(l, iris2)
tbl

row_paths(tbl)

table_structure(tbl)

table_structure(tbl, detail = "row")
```

top_left *Top Left Material (Experimental)*

## Description

A TableTree object can have *top left material* which is a sequence of strings which are printed in the area of the table between the column header display and the label of the first row. These functions acccess and modify that material.

## Usage

```
top_left(obj)

## S4 method for signature 'VTableTree'
top_left(obj)

## S4 method for signature 'InstantiatedColumnInfo'
top_left(obj)

## S4 method for signature 'PreDataTableLayouts'
top_left(obj)

top_left(obj) <- value

## S4 replacement method for signature 'VTableTree'
top_left(obj) <- value

## S4 replacement method for signature 'InstantiatedColumnInfo'
top_left(obj) <- value

## S4 replacement method for signature 'PreDataTableLayouts'
top_left(obj) <- value
```

## Arguments

| | |
|---|---|
| obj | ANY. The object for the accessor to access or modify |
| value | The new value |

## Value

A character vector representing the top-left material of obj (or obj after modification, in the case of the setter).

`toString,VTableTree-method`
*Convert an* `rtable` *object to a string*

## Description

Convert an `rtable` object to a string

## Usage

```
## S4 method for signature 'VTableTree'
toString(x, widths = NULL, col_gap = 3)
```

## Arguments

| | |
|---|---|
| `x` | table object |
| `widths` | widths of row.name and columns columns |
| `col_gap` | gap between columns |

## Value

a string representation of `x` as it appears when printed.

## Examples

```
library(dplyr)

iris2 <- iris %>%
  group_by(Species) %>%
  mutate(group = as.factor(rep_len(c("a", "b"), length.out = n()))) %>%
  ungroup()

l <- basic_table() %>%
  split_cols_by("Species") %>%
  split_cols_by("group") %>%
 analyze(c("Sepal.Length", "Petal.Width"), afun = list_wrap_x(summary) , format = "xx.xx")

tbl <- build_table(l, iris2)

cat(toString(tbl, col_gap = 3))
```

## tree_children *Retrieve or set the direct children of a Tree-style object*

### Description

Retrieve or set the direct children of a Tree-style object

### Usage

```
tree_children(x)

## S4 method for signature 'VTree'
tree_children(x)

## S4 method for signature 'VTableTree'
tree_children(x)

## S4 method for signature 'VLeaf'
tree_children(x)

tree_children(x) <- value

## S4 replacement method for signature 'VTree'
tree_children(x) <- value

## S4 replacement method for signature 'VTableTree'
tree_children(x) <- value
```

### Arguments

| | |
|---|---|
| x | An object with a Tree structure |
| value | New list of children. |

### Value

List of direct children of x

## trim_levels_to_map *Trim Levels to map*

### Description

This split function constructor creatse a split function which trims levels of a variable to reflect restrictions on the possible combinations of two or more variables which are split by (along the same axis) within a layout.

## Usage

```
trim_levels_to_map(map = NULL)
```

## Arguments

| | |
|---|---|
| map | data.frame. A data.frame defining allowed combinations of variables. Any combination at the level of this split not present in the map will be removed from the data, both for the variable being split and those present in the data but not associated with this split or any parents of it. |

## Details

When splitting occurs, the map is subset to the values of all previously performed splits. The levels of the variable being split are then pruned to only those still present within this subset of the map representing the current hierarchical splitting context.

Splitting is then performed via the keep_split_levels split function.

Each resulting element of the partition is then further trimmed by pruning values of any remaining variables specified in the map to those values allowed under the combination of the previous and current split.

## Value

a fun

## See Also

trim_levels_in_group

## Examples

```
map <- data.frame(
      LBCAT = c("CHEMISTRY", "CHEMISTRY", "CHEMISTRY", "IMMUNOLOGY"),
      PARAMCD = c("ALT", "CRP", "CRP", "IGA"),
      ANRIND = c("LOW", "LOW", "HIGH", "HIGH"),
      stringsAsFactors = FALSE
  )

  lyt <- basic_table() %>%
      split_rows_by("LBCAT") %>%
      split_rows_by("PARAMCD", split_fun = trim_levels_to_map(map = map)) %>%
      analyze("ANRIND")
  tbl1 <- build_table(lyt, ex_adlb)
```

## trim_rows *Trim rows from a populated table without regard for table structure*

### Description

Trim rows from a populated table without regard for table structure

### Usage

```
trim_rows(tt, criteria = all_zero_or_na)
```

### Arguments

| | |
|---|---|
| tt | TableTree (or related class). A TableTree object representing a populated table. |
| criteria | function. Function which takes a TableRow object and returns TRUE if that row should be removed. Defaults to all_zero_or_na |

### Value

The table with rows that have only NA or 0 cell values removed

### Note

Visible LabelRows are including in this trimming, which can lead to either all label rows being trimmed or label rows remaining when all data rows have been trimmed, depending on what criteria returns when called on a LabelRow object. To avoid this, use the structurally-aware prune_table machinery instead.

### See Also

prune_table()

## trim_zero_rows *Trim Zero Rows*

### Description

Trim Zero Rows

### Usage

```
trim_zero_rows(tbl)
```

### Arguments

| | |
|---|---|
| tbl | table object |

## Value

an rtable object

---

`tt_at_path` *Get or set table elements at specified path*

---

## Description

Get or set table elements at specified path

## Usage

```
tt_at_path(tt, path, ...)

## S4 method for signature 'VTableTree'
tt_at_path(tt, path, ...)

tt_at_path(tt, path, ...) <- value

## S4 replacement method for signature 'VTableTree,ANY,VTableTree'
tt_at_path(tt, path, ...) <- value

## S4 replacement method for signature 'VTableTree,ANY,`NULL`'
tt_at_path(tt, path, ...) <- value

## S4 replacement method for signature 'VTableTree,ANY,TableRow'
tt_at_path(tt, path, ...) <- value
```

## Arguments

| | |
|---|---|
| `tt` | TableTree (or related class). A TableTree object representing a populated table. |
| `path` | character. A vector path for a position within the structure of a tabletree. Each element represents a subsequent choice amongst the children of the previous choice. |
| `...` | unused. |
| `value` | The new value |

tt_to_flextable *Create a FlexTable object representing an rtables TableTree*

## Description

Create a FlexTable object representing an rtables TableTree

## Usage

```
tt_to_flextable(
  tt,
  paginate = FALSE,
  lpp = NULL,
  ...,
  colwidths = propose_column_widths(tt),
  total_width = 5
)
```

## Arguments

| | |
|---|---|
| tt | TableTree (or related class). A TableTree object representing a populated table. |
| paginate | logical(1). Should tt be paginated and exported as multiple flextables. Defaults to FALSE |
| lpp | numeric. Maximum lines per page including (re)printed header and context rows |
| ... | Passed on to metthods or tabulation functions. |
| colwidths | numeric vector. Column widths for use with vertical pagination. Currently ignored. |
| total_width | numeric(1). Total width in inches for the resulting flextable(s). Defaults to 5. |

## Value

a flextable object

## Examples

```
analysisfun <- function(x, ...) {
    in_rows(row1 = 5,
            row2 = c(1, 2),
            .row_footnotes = list(row1 = "row 1 - row footnote"),
            .cell_footnotes = list(row2 = "row 2 - cell footnote"))
}

lyt <- basic_table(title = "Title says Whaaaat", subtitles = "Oh, ok.",
               main_footer = "ha HA! Footer!") %>%
split_cols_by("ARM") %>%
analyze("AGE", afun = analysisfun)
```

```
tbl <-  build_table(lyt, ex_adsl)
ft <- tt_to_flextable(tbl)
ft
```

---

`update_ref_indexing` *Update footnote indexes on a built table*

---

## Description

Re-indexes footnotes within a built table

## Usage

```
update_ref_indexing(tt)
```

## Arguments

| | |
|---|---|
| `tt` | TableTree (or related class). A TableTree object representing a populated table. |

## Details

After adding or removing referential footnotes manually, or after subsetting a table, the reference indexes (ie the number associated with specific footnotes) may be incorrect. This function recalculates these based on the full table.

## Note

In the future this should not generally need to be called manually.

---

`value_formats` *Value Formats*

---

## Description

Returns a matrix of formats for the cells in a table

## Usage

```
value_formats(obj, default = obj_format(obj))

## S4 method for signature 'ANY'
value_formats(obj, default = obj_format(obj))

## S4 method for signature 'TableRow'
value_formats(obj, default = obj_format(obj))
```

```
## S4 method for signature 'LabelRow'
value_formats(obj, default = obj_format(obj))

## S4 method for signature 'VTableTree'
value_formats(obj, default = obj_format(obj))
```

## Arguments

| | |
|---|---|
| obj | A table or row object. |
| default | FormatSpec. |

## Value

Matrix (storage mode list) containing the effective format for each cell position in the table (including 'virtual' cells implied by label rows, whose formats are always NULL)

## Examples

```
lyt <- basic_table() %>%
split_rows_by("RACE", split_fun = keep_split_levels(c("ASIAN", "WHITE"))) %>%
analyze("AGE")

tbl <- build_table(lyt, DM)
value_formats(tbl)
```

---

# VarLevelSplit-class *Split on levels within a variable*

---

## Description

Split on levels within a variable

## Usage

```
VarLevelSplit(
  var,
  split_label,
  labels_var = NULL,
  cfun = NULL,
  cformat = NULL,
  split_fun = NULL,
  split_format = NULL,
  valorder = NULL,
  split_name = var,
  child_labels = c("default", "visible", "hidden"),
  extra_args = list(),
  indent_mod = 0L,
```

```
  label_pos = c("topleft", "hidden", "visible"),
  cindent_mod = 0L,
  cvar = "",
  cextra_args = list()
)

VarLevWBaselineSplit(
  var,
  ref_group,
  labels_var = var,
  split_label,
  split_fun = NULL,
  label_fstr = "%s - %s",
  cfun = NULL,
  cformat = NULL,
  cvar = "",
  split_format = NULL,
  valorder = NULL,
  split_name = var,
  extra_args = list()
)
```

## Arguments

| | |
|---|---|
| var | string, variable name |
| split_label | string. Label string to be associated with the table generated by the split. Not to be confused with labels assigned to each child (which are based on the data and type of split during tabulation). |
| labels_var | string, name of variable containing labels to be displayed for the values of var |
| cfun | list/function/NULL. tabulation function(s) for creating content rows. Must accept x or df as first parameter. Must accept labelstr as the second argument. Can optionally accept all optional arguments accepted by analysis functions. See analyze. |
| cformat | format spec. Format for content rows |
| split_fun | function/NULL. custom splitting function See custom_split_funs |
| split_format | FormatSpec. Default format associated with the split being created. |
| valorder | character vector. Order that the split children should appear in resulting table. |
| split_name | string. Name associiated with this split (for pathing, etc) |
| child_labels | string. One of "default", "visible", "hidden". What should the display behavior be for the labels (ie label rows) of the children of this split. Defaults to "default" which flags the label row as visible only if the child has 0 content rows. |
| extra_args | list. Extra arguments to be passed to the tabulation function. Element position in thte list corresponds to the children of this split. Named elements in the child-specific lists are ignored if they do not match a formal argument of the ttabulation function. |

| | |
|---|---|
| indent_mod | numeric. Modifier for the default indent position for the structure created by this function(subtable, content table, or row) *and all of that structure's children*. Defaults to 0, which corresponds to the unmodified default behavior. |
| label_pos | character(1). Location the variable label should be displayed, Accepts hidden (default for non-analyze row splits), visible, topleft, and - for analyze splits only - default. For analyze calls, `default` indicates that the variable should be visible if and only if multiple variables are analyzed at the same level of nesting. |
| cindent_mod | numeric(1). The indent modifier for the content tables generated by this split. |
| cvar | character(1). The variable, if any, which the content function should accept. Defaults to NA. |
| cextra_args | list. Extra arguments to be passed to the content function when tabulating row group summaries. |
| ref_group | character. Value of `var` to be taken as the ref_group/control to be compared against. |
| label_fstr | string. An sprintf style format string containing. For non-comparison splits, it can contain up to one `"%s"` which takes the current split value and generates the row/column label. Comparison-based splits it can contain up to two `"%s"`. |

## Value

a `VarLevelSplit` object.

## Author(s)

Gabriel Becker

---

`VarStaticCutSplit-class`

*Splits for cutting by values of a numeric variable*

---

## Description

Splits for cutting by values of a numeric variable

## Usage

```
VarStaticCutSplit(
  var,
  split_label = var,
  cuts,
  cutlabels = NULL,
  cfun = NULL,
  cformat = NULL,
  split_format = NULL,
  split_name = var,
```

```
  child_labels = c("default", "visible", "hidden"),
  extra_args = list(),
  indent_mod = 0L,
  cindent_mod = 0L,
  cvar = "",
  cextra_args = list(),
  label_pos = "visible"
)

CumulativeCutSplit(
  var,
  split_label,
  cuts,
  cutlabels = NULL,
  cfun = NULL,
  cformat = NULL,
  split_format = NULL,
  split_name = var,
  child_labels = c("default", "visible", "hidden"),
  extra_args = list(),
  indent_mod = 0L,
  cindent_mod = 0L,
  cvar = "",
  cextra_args = list(),
  label_pos = "visible"
)

VarDynCutSplit(
  var,
  split_label,
  cutfun,
  cutlabelfun = function(x) NULL,
  cfun = NULL,
  cformat = NULL,
  split_format = NULL,
  split_name = var,
  child_labels = c("default", "visible", "hidden"),
  extra_args = list(),
  cumulative = FALSE,
  indent_mod = 0L,
  cindent_mod = 0L,
  cvar = "",
  cextra_args = list(),
  label_pos = "visible"
)
```

## Arguments

| | |
|---|---|
| var | string, variable name |

| | |
|---|---|
| split_label | string. Label string to be associated with the table generated by the split. Not to be confused with labels assigned to each child (which are based on the data and type of split during tabulation). |
| cuts | numeric. Cuts to use |
| cutlabels | character (or NULL). Labels for the cuts |
| cfun | list/function/NULL. tabulation function(s) for creating content rows. Must accept x or df as first parameter. Must accept labelstr as the second argument. Can optionally accept all optional arguments accepted by analysis functions. See analyze. |
| cformat | format spec. Format for content rows |
| split_format | FormatSpec. Default format associated with the split being created. |
| split_name | string. Name associiated with this split (for pathing, etc) |
| child_labels | string. One of "default", "visible", "hidden". What should the display behavior be for the labels (ie label rows) of the children of this split. Defaults to "default" which flags the label row as visible only if the child has 0 content rows. |
| extra_args | list. Extra arguments to be passed to the tabulation function. Element position in thte list corresponds to the children of this split. Named elements in the child-specific lists are ignored if they do not match a formal argument of the ttabulation function. |
| indent_mod | numeric. Modifier for the default indent position for the structure created by this function(subtable, content table, or row) *and all of that structure's children*. Defaults to 0, which corresponds to the unmodified default behavior. |
| cindent_mod | numeric(1). The indent modifier for the content tables generated by this split. |
| cvar | character(1). The variable, if any, which the content function should accept. Defaults to NA. |
| cextra_args | list. Extra arguments to be passed to the content function when tabulating row group summaries. |
| label_pos | character(1). Location the variable label should be displayed, Accepts hidden (default for non-analyze row splits), visible, topleft, and - for analyze splits only - default. For analyze calls, default indicates that the variable should be visible if and only if multiple variables are analyzed at the same level of nesting. |
| cutfun | function. Function which accepts the *full vector* of var values and returns cut points to be used (via cut) when splitting data during tabulation |
| cutlabelfun | function. Function which returns either labels for the cuts or NULL when passed the return value of cutfun |
| cumulative | logical. Should the cuts be treated as cumulative. Defaults to FALSE |

## Value

a VarStaticCutSplit, CumulativeCutSplit, or VarDynCutSplit object.

vars_in_layout *List Variables required by a pre-data table layout*

## Description

List Variables required by a pre-data table layout

## Usage

```
vars_in_layout(lyt)

## S4 method for signature 'PreDataTableLayouts'
vars_in_layout(lyt)

## S4 method for signature 'PreDataAxisLayout'
vars_in_layout(lyt)

## S4 method for signature 'SplitVector'
vars_in_layout(lyt)

## S4 method for signature 'Split'
vars_in_layout(lyt)

## S4 method for signature 'CompoundSplit'
vars_in_layout(lyt)

## S4 method for signature 'ManualSplit'
vars_in_layout(lyt)
```

## Arguments

| | |
|---|---|
| lyt | The Layout (or a component thereof) |

## Details

This will walk the layout declaration and return a vector of the names of the unique variables that are used in any of the following ways:

- Variable being split on (directly or via cuts)
- Element of a Multi-variable column split
- Content variable
- Value-label variable

## Value

A character vector containing the unique variables explicitly used in the layout (see Notes).

## Note

This function will not detect dependencies implicit in analysis or summary functions which accept `df` and then rely on the existence of particular variables not being split on/ analyzed.

The order these variable names appear within the return vector is undefined and should not be relied upon.

## Examples

```
lyt <- basic_table() %>%
    split_cols_by("ARM") %>%
    split_cols_by("SEX") %>%
    summarize_row_groups(label_fstr = "Overall (N)") %>%
    split_rows_by("RACE", split_label = "Ethnicity", labels_var = "ethn_lab",
                  split_fun = drop_split_levels) %>%
    summarize_row_groups("RACE", label_fstr = "%s (n)") %>%
    analyze("AGE", var_labels = "Age", afun = mean, format = "xx.xx")

vars_in_layout(lyt)
```

---

# var_labels *Get Label Attributes of Variables in a* `data.frame`

---

## Description

Variable labels can be stored as a `label` attribute for each variable. This functions returns a named character vector with the variable labels (empty sting if not specified)

## Usage

```
var_labels(x, fill = FALSE)
```

## Arguments

| | |
|---|---|
| `x` | a `data.frame` object |
| `fill` | boolean in case the `label` attribute does not exist if `TRUE` the variable names is returned, otherwise `NA` |

## Value

a named character vector with the variable labels, the names correspond to the variable names

## Examples

```
x <- iris
var_labels(x)
var_labels(x) <- paste("label for", names(iris))
var_labels(x)
```

## var_labels<- *Set Label Attributes of All Variables in a* data.frame

### Description

Variable labels can be stored as a label attribute for each variable. This functions sets all non-missing (non-NA) variable labels in a data.frame

### Usage

```
var_labels(x) <- value
```

### Arguments

| | |
|---|---|
| x | a data.frame object |
| value | new variable labels, NA removes the variable label |

### Value

modifies the variable labels of x

### Examples

```
x <- iris
var_labels(x)
var_labels(x) <- paste("label for", names(iris))
var_labels(x)

if(interactive()){
View(x) # in RStudio data viewer labels are displayed
}
```

## var_labels_remove *Remove Variable Labels of a* data.frame

### Description

Removing labels attributes from a variables in a data frame

### Usage

```
var_labels_remove(x)
```

### Arguments

| | |
|---|---|
| x | a data.frame object |

### Value

the same data frame as x stripped of variable labels

### Examples

```
x <- var_labels_remove(iris)
```

---

## var_relabel *Copy and Change Variable Labels of a* data.frame

---

### Description

Relabel a subset of the variables

### Usage

```
var_relabel(x, ...)
```

### Arguments

| | |
|---|---|
| x | a data.frame object |
| ... | name-value pairs, where name corresponds to a variable name in x and the value to the new variable label |

### Value

a copy of x with changed labels according to ...

### Examples

```
x <- var_relabel(iris, Sepal.Length = "Sepal Length of iris flower")
var_labels(x)
```

---

## Viewer *Dispaly an* rtable *object in the Viewer pane in RStudio or in a browser*

---

### Description

The table will be displayed using the bootstrap styling for tables.

### Usage

```
Viewer(x, y = NULL, row.names.bold = FALSE, ...)
```

### Arguments

| | |
|---|---|
| x | object of class rtable or shiny.tag (defined in htmltools) |
| y | optional second argument of same type as x |
| row.names.bold | row.names.bold boolean, make rownames bold |
| ... | arguments passed to as_html |

### Value

not meaningful. Called for the side effect of opening a browser or viewer pane.

### Examples

```
if(interactive()) {
sl5 <- factor(iris$Sepal.Length > 5, levels = c(TRUE, FALSE),
   labels = c("S.L > 5", "S.L <= 5"))

df <- cbind(iris, sl5 = sl5)

tbl <- basic_table() %>%
   split_cols_by("sl5") %>%
   analyze("Sepal.Length") %>%
   build_table(df)

Viewer(tbl)
Viewer(tbl, tbl)


tbl2 <-htmltools::tags$div(
  class = "table-responsive",
  as_html(tbl, class_table = "table")
)

Viewer(tbl, tbl2)

}
```

---

## with_label *Return an object with a label attribute*

---

### Description

Return an object with a label attribute

### Usage

```
with_label(x, label)
```

### Arguments

| | |
|---|---|
| x | an object |
| label | label attribute to to attached to x |

### Value

x labeled by label. Note: the exact mechanism of labeling should be considered an internal implementation detail, but the label will always be retrieved via obj_label.

### Examples

```
x <- with_label(c(1,2,3), label = "Test")
obj_label(x)
```

---

## [<-,VTableTree,ANY,ANY,list-method

*retrieve and assign elements of a TableTree*

---

### Description

retrieve and assign elements of a TableTree

### Usage

```
## S4 replacement method for signature 'VTableTree,ANY,ANY,list'
x[i, j, ...] <- value

## S4 replacement method for signature 'VTableTree,ANY,ANY,CellValue'
x[i, j, ...] <- value

## S4 method for signature 'VTableTree,logical,logical'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'VTableTree,logical,ANY'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'VTableTree,logical,missing'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'VTableTree,ANY,logical'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'VTableTree,ANY,missing'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'VTableTree,missing,ANY'
x[i, j, ..., drop = FALSE]
```

```
## S4 method for signature 'VTableTree,ANY,character'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'VTableTree,character,ANY'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'VTableTree,character,character'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'VTableTree,missing,numeric'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'VTableTree,numeric,numeric'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'VTableTree,list'
x[[i, j, ...]]
```

## Arguments

| | |
|---|---|
| x | TableTree |
| i | index |
| j | index |
| ... | Includes |
| | ***keep_topleft*** logical(1) ([ only) Should the 'top-left' material for the table be retained after subsetting. Defaults to NA, which retains the material if all rows are included (ie subsetting was by column), and drops it otherwise. |
| | ***keep_titles*** logical(1) Should title and non-referential footer information be retained. Defaults to FALSE |
| | ***reindex_refs*** logical(1). Should referential footnotes be re-indexed as if the resulting subset is the entire table. Defaults to TRUE |
| value | Replacement value (list, TableRow, or TableTree) |
| drop | logical(1). Should the value in the cell be returned if only one cell is selected by the combination of i and j. Defaults to FALSE |

## Value

a TableTree (or ElementaryTable) object, unless a single cell was selected with drop=TRUE, in which case the (possibly multi-valued) fully stripped raw value of the selected cell.

## Examples

```
l <- basic_table() %>%
   split_cols_by("ARM") %>%
   analyze(c("SEX", "AGE"))
```

```
tbl <- build_table(l, DM)

tbl

tbl[1, ]
tbl[1:2, 2]

tbl[2, 1]
tbl[2, 1, drop = TRUE]

tbl[, 1]

tbl[-2, ]
tbl[, -1]

tbl[2, 1] <- rcell(999)
tbl[2, ] <- list(rrow("FFF", 888, 666, 777))
tbl[3, ] <- list(-111, -222, -333)
tbl
```

# Index