rtables - A Guide To The Perplexed

Gabriel Becker

2023-05-18

Writing Analysis Functions

Step 0 - Do Not Use `make_afun`

But You Made It…

Why? Tell me what this function does

`my_afun`

```
## function(x, ...) {
##     raw_res <- inner_fun(x, ...)
##     in_rows(mean_sd = rcell(raw_res[["mean_sd"]],
##                             label = "Mean (SD)",
##                             format = "xx.x (xx.x)"),
##             range = rcell(raw_res[["range"]],
##                           format = "xx.x - xx.x",
##                           label = "range"))
## }
```

Now Tell Me What This Function Does

my_afun2

```
## function (x, ...)
## {
##     .if_in_formals <- function(nm, ifnot = list(), named_lwrap = TRUE) {
##         val <- if (nm %in% fun_fnames)
##             get(nm)
##         else ifnot
##         if (named_lwrap && !identical(val, ifnot))
##             setNames(list(val), nm)
##         else val
##     }
##     custargs <- fun_args
##     in_rc_argl <- .if_in_formals(".in_ref_col")
##     .in_ref_col <- if (length(in_rc_argl) > 0)
##         in_rc_argl[[1]]
##     else FALSE
##     sfunargs <- c(.if_in_formals("x"), .if_in_formals("df"),
##         .if_in_formals(".N_col"), .if_in_formals(".N_total"),
##         .if_in_formals(".N_row"), .if_in_formals(".ref_group"),
##         in_rc_argl, .if_in_formals(".df_row"), .if_in_formals(".var"),
##         .if_in_formals(".ref_full"))
##     allvars <- setdiff(fun_fnames, c("...", names(sfunargs)))
##     if ("..." %in% fun_fnames) {
##         exargs <- eval(parser_helper(text = "list(...)"))
##         custargs[names(exargs)] <- exargs
##         allvars <- unique(c(allvars, names(custargs)))
##     }
##     for (var in allvars) {
##         if (var %in% fun_fnames && eval(parser_helper(text = paste0("!missing(",
##             var, ")"))))
##             sfunargs[[var]] <- get(var)
##         else if (var %in% names(custargs))
##             sfunargs[[var]] <- custargs[[var]]
##     }
##     rawvals <- do.call(fun, sfunargs)
##     final_vals <- if (is.null(.stats))
##         rawvals
##     else rawvals[.stats]
##     if (!is.list(rawvals))
##         stop("make_afun expects a function fun that always returns a list")
##     if (!is.null(.stats))
##         stopifnot(all(.stats %in% names(rawvals)))
##     else .stats <- names(rawvals)
##     if (!is.null(.ungroup_stats) && !all(.ungroup_stats %in%
##         .stats)) {
```

They're the same

```
my_afun(1:10)
```

```
## RowsVerticalSection (in_rows) object print method:
## ----------------------------
##   row_name formatted_cell indent_mod row_label
## 1  mean_sd      5.5 (3.0)          0 Mean (SD)
## 2    range     1.0 - 10.0          0     range
```

```
my_afun2(1:10)
```

```
## RowsVerticalSection (in_rows) object print method:
## ----------------------------
##   row_name formatted_cell indent_mod row_label
## 1  mean_sd      5.5 (3.0)          0 Mean (SD)
## 2    range     1.0 - 10.0          0     range
```

But I can tell you which one I want to debug…

Put another way

```
my_afun2 <- make_afun(inner_fun, .stats = c("mean_sd", "range"),
                      .formats = c(mean_sd = "xx.x (xx.x)",
                                   range = "xx.x - xx.x"),
                      .labels = list(mean_sd = "Mean (SD)",
                                     range = "range"))
```

```
my_afun <- function(x, ...) {
    raw_res <- inner_fun(x, ...)
    in_rows(mean_sd = rcell(raw_res[["mean_sd"]],
                            label = "Mean (SD)",
                            format = "xx.x (xx.x)"),
            range = rcell(raw_res[["range"]],
                          format = "xx.x - xx.x",
                          label = "range"))
}
```

Is the first one easier?

Sure, I guess

Is it **easier enough to justify making debugging virtually impossible**?

Components of Writing an Analysis Function

First Argument

`rtables` behavior depends on what this argument is named

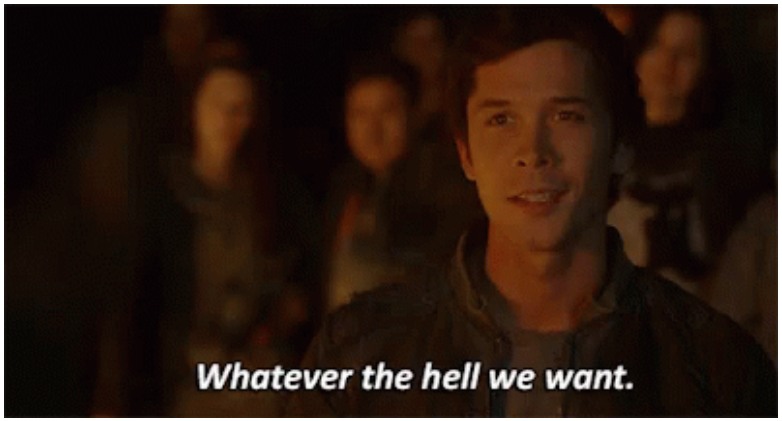
- `df` - receives full facet `data.frame`
- `x` (or anything else) - receives *data vector of variable being analyzed*

Other (Optional, But) Important Parameters

- `.N_col` - Count (taking `alt_counts_df` into account if applicable)
- `.N_row` - Count of observations for this facet
- `.df_row` - the full incoming data (if the first argument was not `df`)
- `.var` - the name of the variable being analyzed (or `NA` in the `analyze_colvars` case
- `.ref_group` - The level of `.var` corresponding to the reference group, if set
- `.ref_full` - the data for the reference group sibling facet
- `.in_ref_col` - boolean indicating whether the current column is the reference col
- `.spl_context` - Splitting history in row space (see next section)

Anatomy of an Analysis Function

```
function(x, .N_col, .var, .spl_context, ...) {
```




```
  in_rows(

  )
}
```

in_rows

- Used to construct analysis function output
  - called `RowsVerticalSection` objects
- use `"row label" = <value>` paradigm, or
- `.list`

Basic Example

```
afun <- function(x, .N_col, .spl_context) {
    in_rows("n (%)" = rcell(length(x) * c(1, 1/.N_col),
                            format = "xx (xx.x%)"),
            Mean = rcell(mean(x), format = "xx.x"),
            "Min - Max" = rcell(range(x), format = "xx.x - xx.x"))
}

lyt <- basic_table() %>%
    split_rows_by("ARM") %>%
    analyze("AGE", afun = afun)

build_table(lyt, DM)
```

```
##                   all obs
## ————————————————————————
## A: Drug X
##   n (%)          121 (34.0%)
##   Mean             34.9
##   Min - Max      20.0 - 60.0
## B: Placebo
##   n (%)          106 (29.8%)
##   Mean             33.0
##   Min - Max      21.0 - 55.0
## C: Combination
##   n (%)          129 (36.2%)
##   Mean             34.6
##   Min - Max      22.0 - 53.0
```

But Remember: Whatever We Want

- Conditionally different number of rows^
- Conditionally different formatting

Constraints

- Must generate the same number of rows for all columns in a single row facet

Understanding And Using Split Context

`.spl_context` - Place within the table structure

- `data.frame` which describes each ancestor facet the current facet is nested within
- One row per ancestor state, columns:
  - `split` - name of split
  - `value` - value of split
  - `full_parent_df` - full (all columns) dataframe corresponding to `value` in that split
  - `all_cols_n` - the number of observations across all columns corresponding to `value` in that split
  - <1 column for each column in the table structure, currently named by their leaf names, this will be changes>
  - `cur_col_subset` - the logical vector which subsets `full_parent_df` for the current column
  - `cur_col_n` - number of observations which `cur_col_subset` selects

Using .spl_context

```
cfun <- function(df, labelstr, .spl_context) {
    lastrow <- .spl_context[nrow(.spl_context) - 1, ]
    last_col_count <- lastrow$cur_col_n
    row_count <- nrow(lastrow$full_parent_df[[1]])
    in_rows(c(nrow(df), last_col_count),
            .names = labelstr,
            .labels = sprintf("%s (%d in parent)", labelstr,
                              row_count),
            .formats = "xx / xx")
}

afun <- function(x, .spl_context) {
    lastrow <- .spl_context[nrow(.spl_context), ]
    all_data <- .spl_context$full_parent_df[[1]]
    val <- c(sum(x >= mean(all_data$AGE)),
             lastrow$cur_col_n)
    row_count <-  nrow(lastrow$full_parent_df[[1]])
    in_rows(val,
            .names = "age_analysis",
            .labels = sprintf("counts (%d in parent)",
                              row_count),
            .formats = "xx / xx")
}

lyt <- basic_table(show_colcounts = TRUE) %>%
    split_cols_by("ARM") %>%
    split_rows_by("COUNTRY",
                  split_fun = keep_split_levels(c("CHN", "USA"))) %>%
    summarize_row_groups(cfun = cfun) %>%
    split_rows_by("STRATA1") %>%
    summarize_row_groups(cfun = cfun) %>%
    analyze("AGE", afun = afun)

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

```
tab
```

```
##                          A: Drug X   B: Placebo   C: Combination
##                           (N=121)     (N=106)        (N=129)
## ———————————————————————————————————————————————————————————————
## CHN (356 in parent)      62 / 121    48 / 106       69 / 129
##   A (179 in parent)       19 / 62     14 / 48        20 / 69
##     counts (53 in parent)  7 / 19      5 / 14         8 / 20
##   B (179 in parent)       19 / 62     16 / 48        22 / 69
##     counts (57 in parent)  9 / 19      5 / 16         9 / 22
##   C (179 in parent)       24 / 62     18 / 48        27 / 69
##     counts (69 in parent)  15 / 24     10 / 18        11 / 27
## USA (356 in parent)      13 / 121    14 / 106       17 / 129
##   A (44 in parent)         5 / 13      3 / 14         9 / 17
##     counts (17 in parent)   3 / 5       1 / 3          5 / 9
##   B (44 in parent)         3 / 13      6 / 14         2 / 17
##     counts (11 in parent)   3 / 3       2 / 6          1 / 2
##   C (44 in parent)         5 / 13      5 / 14         6 / 17
##     counts (16 in parent)   3 / 5       2 / 5          4 / 6
```

Custom Splitting Functions

Splitting Functions

Everyone: Faceting on a categorical variable creates a facet paen for each level

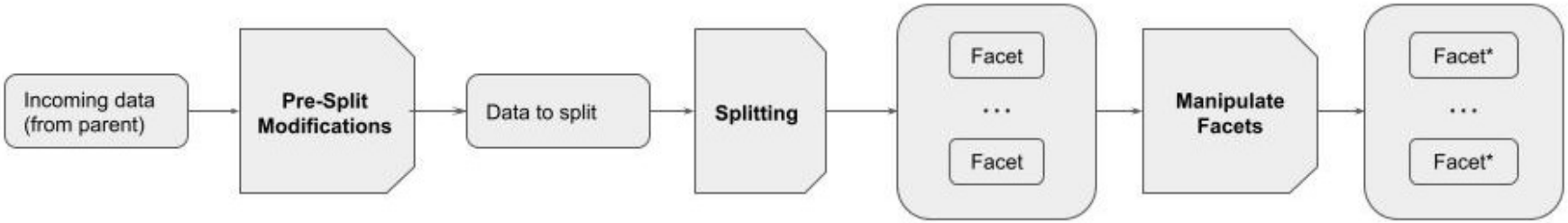
`split_rows_by( , split_fun = )`:

Splitting Is

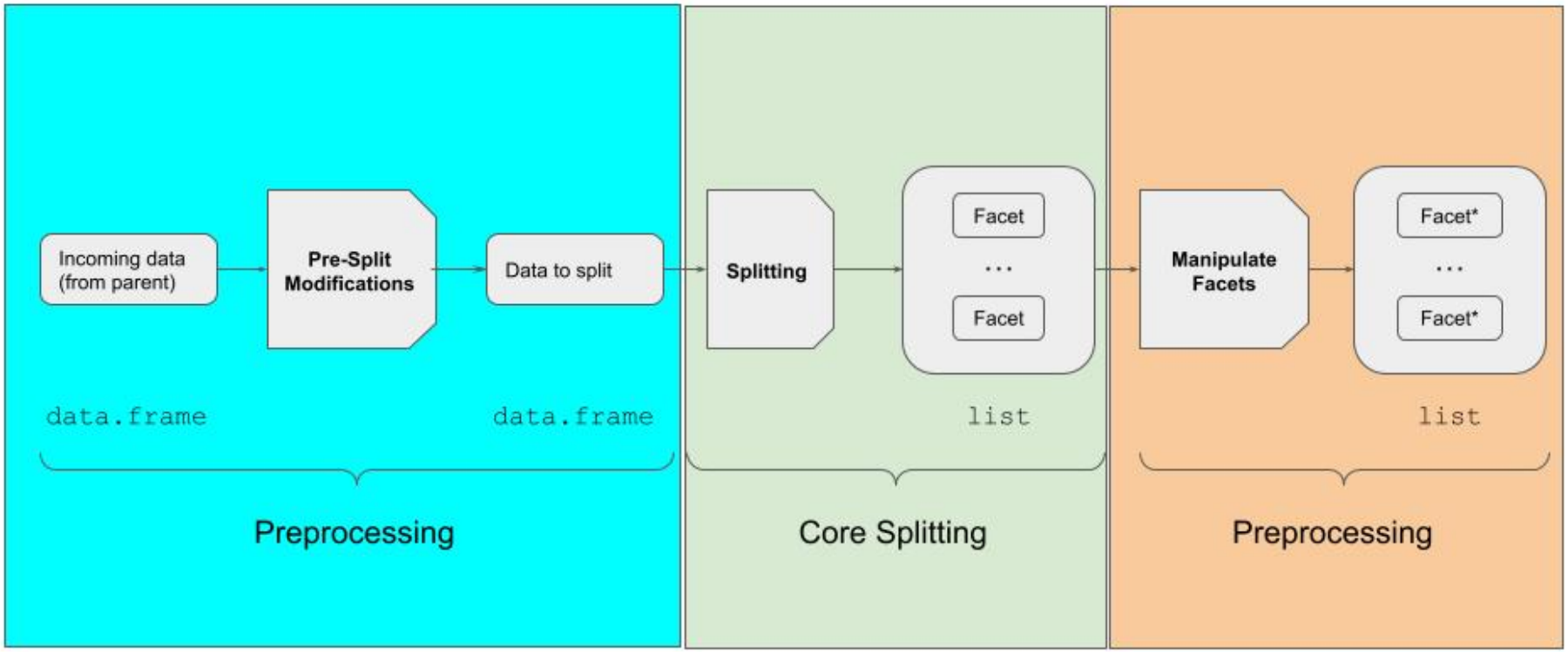
- mapping a `data.frame` to a `list` representing a set of facets

We can draw the generalized faceting process

Like so:



There are 3 ways we can modify this process



Change the data

Change what faceting means

Change what facets are created

Enter `make_split_fun`

`make_split_fun` - Split Function Constructor

Accepts:

- `pre` - a list of functions which accept and return a `data.frame` for use in the preprocessing step
- `core_split` - a function which accepts a data.frame and returns a “split result list”. Only supported in row space
- `post` - a list of functions which accept and return a list representing a split result (set of facets)

`make_split_fun` - Customize What You Want

Everything Else Remains the Default

Some Advice For Using `make_split_fun`

- Write “behavior building-blocks” for use in `pre` and `post`
  - Rather than large multi-behavior functions
  - mix and match
- You should almost never need to override `core_split`
  - If you think you do, you probably don’t

Lets Work Some Examples

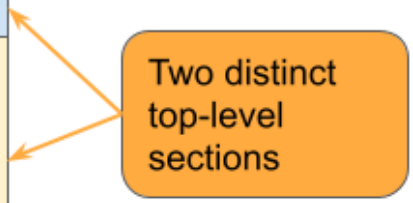
COXT01 With Covariates

Activating `rtables` Vision

Effect/Covariate Included in the Model	n	Treatment Effect Adjusted for Covariate Hazard Ratio	95% CI	p-value	Interaction p-value
Treatment:					
A: Drug X vs control (B: Placebo)	247	0.70	(0.51, 0.96)	0.0293	
Covariate:					
Age	247				0.8626
34		0.70	(0.51, 0.97)		
RACE	247				0.9197
ASIAN		0.75	(0.48, 1.16)		
BLACK OR AFRICAN AMERICAN		0.66	(0.34, 1.28)		
WHITE		0.65	(0.33, 1.27)		

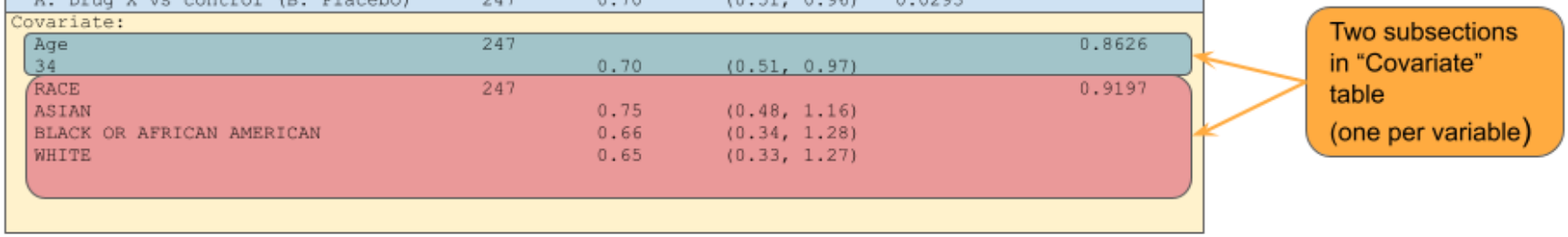
`rtables` Vision - Top Level Row Structure

Effect/Covariate Included in the Model	n	Treatment Effect Adjusted for Covariate Hazard Ratio	95% CI	p-value	Interaction p-value
Treatment:					
A: Drug X vs control (B: Placebo)	247	0.70	(0.51, 0.96)	0.0293	
Covariate:					
Age	247				0.8626
34		0.70	(0.51, 0.97)		
RACE	247				0.9197
ASIAN		0.75	(0.48, 1.16)		
BLACK OR AFRICAN AMERICAN		0.66	(0.34, 1.28)		
WHITE		0.65	(0.33, 1.27)		



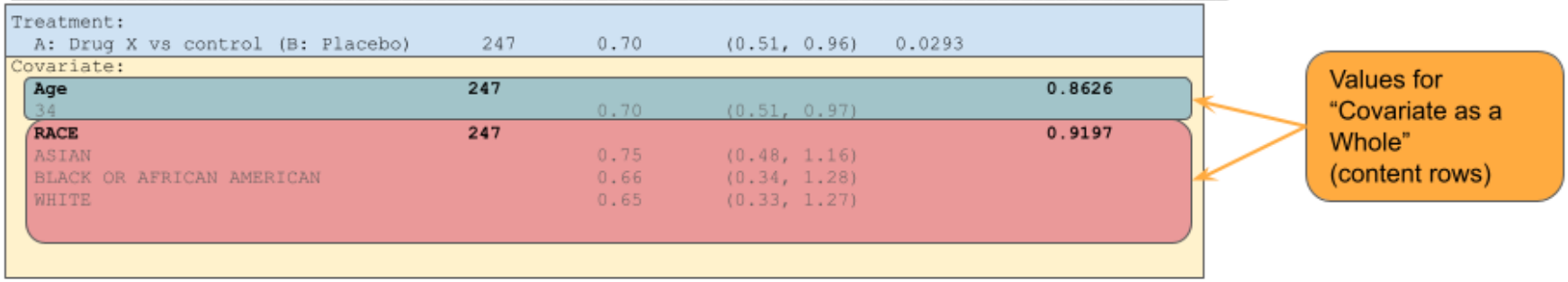
`rtables` Vision - Nested Row Structure

Effect/Covariate Included in the Model	n	Treatment Effect Adjusted for Covariate: Hazard Ratio	95% CI	p-value	Interaction p-value
Treatment:					
A: Drug X vs control (B: Placebo)	247	0.70	(0.51, 0.96)	0.0293	
Covariate:					
Age	247				0.8626
34		0.70	(0.51, 0.97)		
RACE	247				0.9197
ASIAN		0.75	(0.48, 1.16)		
BLACK OR AFRICAN AMERICAN		0.66	(0.34, 1.28)		
WHITE		0.65	(0.33, 1.27)		



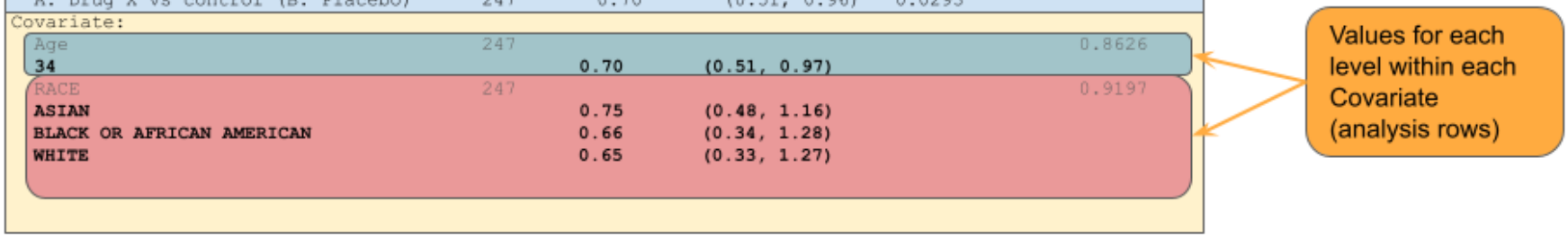
`rtables` Vision - Detecting Group Summaries

Effect/Covariate Included in the Model	n	Treatment Effect Adjusted for Covariate: Hazard Ratio	95% CI	p-value	Interaction p-value
Treatment:					
A: Drug X vs control (B: Placebo)	247	0.70	(0.51, 0.96)	0.0293	
Covariate:					
Age	**247**				**0.8626**
34		0.70	(0.51, 0.97)		
RACE	**247**				**0.9197**
ASIAN		0.75	(0.48, 1.16)		
BLACK OR AFRICAN AMERICAN		0.66	(0.34, 1.28)		
WHITE		0.65	(0.33, 1.27)		

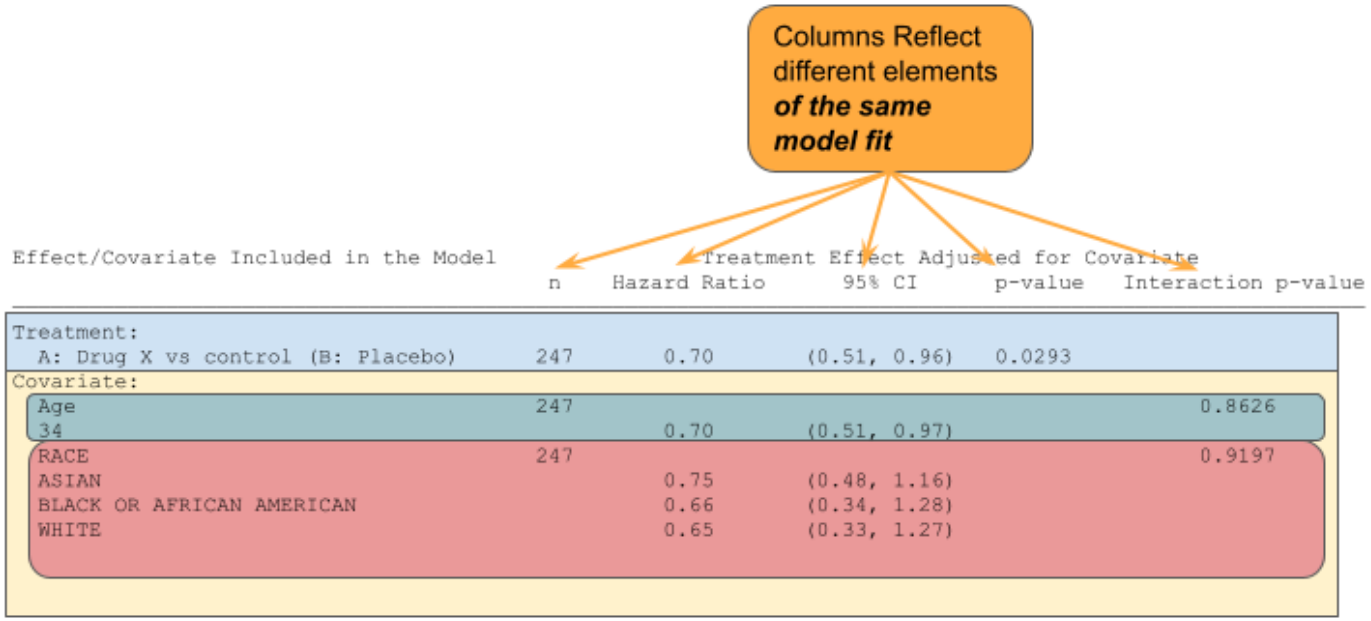


`rtables` Vision - Identifying Analysis Rows

Effect/Covariate Included in the Model	n	Treatment Effect Adjusted for Covariate Hazard Ratio	95% CI	p-value	Interaction p-value
Treatment:					
A: Drug X vs control (B: Placebo)	247	0.70	(0.51, 0.96)	0.0293	
Covariate:					
Age	247				0.8626
34		**0.70**	**(0.51, 0.97)**		
RACE	247				0.9197
ASIAN		**0.75**	**(0.48, 1.16)**		
BLACK OR AFRICAN AMERICAN		**0.66**	**(0.34, 1.28)**		
WHITE		**0.65**	**(0.33, 1.27)**		



`rtables` Vision - Analyzing Column Structure



Effect/Covariate Included in the Model	Treatment Effect Adjusted for Covariate				
	n	Hazard Ratio	95% CI	p-value	Interaction p-value
Treatment:					
A: Drug X vs control (B: Placebo)	247	0.70	(0.51, 0.96)	0.0293	
Covariate:					
Age	247				0.8626
34		0.70	(0.51, 0.97)		
RACE	247				0.9197
ASIAN		0.75	(0.48, 1.16)		
BLACK OR AFRICAN AMERICAN		0.66	(0.34, 1.28)		
WHITE		0.65	(0.33, 1.27)		

Conceptual Analysis Complete

- Top-level Global Analysis (or Content) Row (Treatment)
- Independent Row Sections Per Covariate (AGE, RACE)
  - Separate model fit for each covariate
    - Accessed in multiple columns and rows!
  - Overall Summary Per Covariate (context rows!)
  - Analysis per covariate level (analysis rows)
- Columns access different elements of the same model (per row)
  - same meaning across rows for each column

Key Points

- Do not refit same model over and over
  - That would make us look like clowns
- Do not pre-fit model(s)
  - Cause I don’t feel like it.
  - And I’m the one writing the slides

Core Table Structure

- Column split where each facet contains all the data
  - Can be spoofed with `multivar` or `make_split_fun`
- Initial Top-level `analyze`/`summarize_row_groups`
- `split_rows_by_multivar`
  - We just exported this, but workaround is easy with `make_split_fun`

Problem 1: Clowns Are Fine But Now Is Not The Time

Caching the model fit by covariate

```
cached_model <- function(df, cov, .spl_context, cache_env) {
    model_form <- paste0("survival::Surv(AVAL, event) ~ ARM *", cov)
    full_df <- df
    if(!is.null(cache_env[[cov]]))
        model <- cache_env[[cov]]
    else {
        print(model_form)
        model <- survival::coxph(formula = stats::as.formula(model_form),
                                 data = full_df,
                                 ties = "exact")
        cache_env[[cov]] <- model
    }
    model
}
```

Objection! You Can't Do That In An **afun**!

I Can And I Shall

```
cox_model_el_direct <- function(df, .var, .spl_context, labelstr = "",
                                model_el,
                                format = formats[model_el[1]],
                                cache_env,
                                cov_main = FALSE) {
    cov <- tail(.spl_context$value, 1)
    model <- cached_model(df = df, .spl_context = .spl_context, cache_env = cache_env, cov = cov)
    effvar <- "ARM"
    givlevs <- if(is.factor(df[[cov]])) levels(df[[cov]]) else as.character(median(df[[cov]]))
    thing <- tern:::h_coxreg_extract_interaction(effect = "ARM", covar = cov, mod = model,
                                                 data = df, control = control_coxreg(), at = NULL)
    ## interaction stuff don't get main pvals
    thing[,"pval"] <- NA_real_

    if(!cov_main) {
        thing_keep <- thing[thing$level %in% givlevs,]
    } else
        thing_keep <- thing[nchar(thing$level) == 0,]

    retvals <- as.list(apply(thing_keep[model_el], 1, function(x) x, simplify = FALSE))
    nms <- if(cov_main) labelstr else givlevs
    in_rows(.list = retvals, .names = nms,
            .labels = nms,
            .formats = setNames(rep(format, length(nms)), nms),
            .format_na_strs = setNames(rep("", length(nms)), nms))

}
```

`.spl_context` Tells Us What Covariate We're In

```
cox_model_el_direct <- function(df, .var, .spl_context, labelstr = "",
                                model_el,
                                format = formats[model_el[1]],
                                cache_env,
                                cov_main = FALSE) {
    cov <- tail(.spl_context$value, 1)
    model <- cached_model(df = df, .spl_context = .spl_context, cache_env = cache_env, cov = cov)
    effvar <- "ARM"
    givlevs <- if(is.factor(df[[cov]])) levels(df[[cov]]) else as.character(median(df[[cov]]))
    thing <- tern:::h_coxreg_extract_interaction(effect = "ARM", covar = cov, mod = model,
                                                 data = df, control = control_coxreg(), at = NULL)
    ## interaction stuff don't get main pvals
    thing[,"pval"] <- NA_real_

    if(!cov_main) {
        thing_keep <- thing[thing$level %in% givlevs,]
    } else
        thing_keep <- thing[nchar(thing$level) == 0,]

    retvals <- as.list(apply(thing_keep[model_el], 1, function(x) x, simplify = FALSE))
    nms <- if(cov_main) labelstr else givlevs
    in_rows(.list = retvals, .names = nms,
            .labels = nms,
            .formats = setNames(rep(format, length(nms)), nms),
            .format_na_strs = setNames(rep("", length(nms)), nms))

}
```

Fit The Model - If We Feel Like It

```
cox_model_el_direct <- function(df, .var, .spl_context, labelstr = "",
                                model_el,
                                format = formats[model_el[1]],
                                cache_env,
                                cov_main = FALSE) {
    cov <- tail(.spl_context$value, 1)
    model <- cached_model(df = df, .spl_context = .spl_context, cache_env = cache_env, cov = cov)
    effvar <- "ARM"
    givlevs <- if(is.factor(df[[cov]])) levels(df[[cov]]) else as.character(median(df[[cov]]))
    thing <- tern:::h_coxreg_extract_interaction(effect = "ARM", covar = cov, mod = model,
                                                 data = df, control = control_coxreg(), at = NULL)
    ## interaction stuff don't get main pvals
    thing[,"pval"] <- NA_real_

    if(!cov_main) {
        thing_keep <- thing[thing$level %in% givlevs,]
    } else
        thing_keep <- thing[nchar(thing$level) == 0,]

    retvals <- as.list(apply(thing_keep[model_el], 1, function(x) x, simplify = FALSE))
    nms <- if(cov_main) labelstr else givlevs
    in_rows(.list = retvals, .names = nms,
            .labels = nms,
            .formats = setNames(rep(format, length(nms)), nms),
            .format_na_strs = setNames(rep("", length(nms)), nms))

}
```

Semi-Politely Decline To Reinvent The Wheel

```
cox_model_el_direct <- function(df, .var, .spl_context, labelstr = "",
                                model_el,
                                format = formats[model_el[1]],
                                cache_env,
                                cov_main = FALSE) {
    cov <- tail(.spl_context$value, 1)
    model <- cached_model(df = df, .spl_context = .spl_context, cache_env = cache_env, cov = cov)
    effvar <- "ARM"
    givlevs <- if(is.factor(df[[cov]])) levels(df[[cov]]) else as.character(median(df[[cov]]))
    thing <- tern:::h_coxreg_extract_interaction(effect = "ARM", covar = cov, mod = model,
                                                 data = df, control = control_coxreg(), at = NULL)
    ## interaction stuff don't get main pvals
    thing[,"pval"] <- NA_real_

    if(!cov_main) {
        thing_keep <- thing[thing$level %in% givlevs,]
    } else
        thing_keep <- thing[nchar(thing$level) == 0,]
    retvals <- as.list(apply(thing_keep[model_el], 1, function(x) x, simplify = FALSE))
    nms <- if(cov_main) labelstr else givlevs
    in_rows(.list = retvals, .names = nms,
            .labels = nms,
            .formats = setNames(rep(format, length(nms)), nms),
            .format_na_strs = setNames(rep("", length(nms)), nms))

}
```

Handle Summary Vs Analysis Row Differences

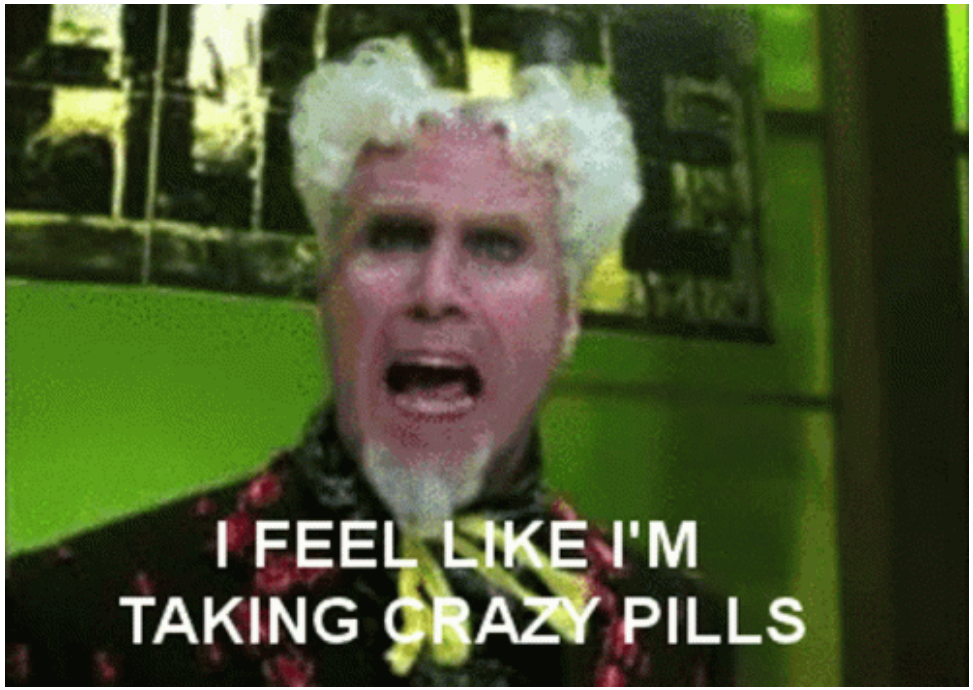
```
cox_model_el_direct <- function(df, .var, .spl_context, labelstr = "",
                                model_el,
                                format = formats[model_el[1]],
                                cache_env,
                                cov_main = FALSE) {
    cov <- tail(.spl_context$value, 1)
    model <- cached_model(df = df, .spl_context = .spl_context, cache_env = cache_env, cov = cov)
    effvar <- "ARM"
    givlevs <- if(is.factor(df[[cov]])) levels(df[[cov]]) else as.character(median(df[[cov]]))
    thing <- tern:::h_coxreg_extract_interaction(effect = "ARM", covar = cov, mod = model,
                                                 data = df, control = control_coxreg(), at = NULL)
    ## interaction stuff don't get main pvals
    thing[,"pval"] <- NA_real_
    if(!cov_main) {
        thing_keep <- thing[thing$level %in% givlevs,]
    } else
        thing_keep <- thing[nchar(thing$level) == 0,]
    retvals <- as.list(apply(thing_keep[model_el], 1, function(x) x, simplify = FALSE))
    nms <- if(cov_main) labelstr else givlevs
    in_rows(.list = retvals, .names = nms,
            .labels = nms,
            .formats = setNames(rep(format, length(nms)), nms),
            .format_na_strs = setNames(rep("", length(nms)), nms))

}
```

Inform `rtables` It Can Take Back Over Now

```
cox_model_el_direct <- function(df, .var, .spl_context, labelstr = "",
                                model_el,
                                format = formats[model_el[1]],
                                cache_env,
                                cov_main = FALSE) {
    cov <- tail(.spl_context$value, 1)
    model <- cached_model(df = df, .spl_context = .spl_context, cache_env = cache_env, cov = cov)
    effvar <- "ARM"
    givlevs <- if(is.factor(df[[cov]])) levels(df[[cov]]) else as.character(median(df[[cov]]))
    thing <- tern:::h_coxreg_extract_interaction(effect = "ARM", covar = cov, mod = model,
                                                 data = df, control = control_coxreg(), at = NULL)
    ## interaction stuff don't get main pvals
    thing[,"pval"] <- NA_real_
    if(!cov_main) {
        thing_keep <- thing[thing$level %in% givlevs,]
    } else
        thing_keep <- thing[nchar(thing$level) == 0,]
    retvals <- as.list(apply(thing_keep[model_el], 1, function(x) x, simplify = FALSE))
    nms <- if(cov_main) labelstr else givlevs
    in_rows(.list = retvals, .names = nms,
            .labels = nms,
            .formats = setNames(rep(format, length(nms)), nms),
            .format_na_strs = setNames(rep("", length(nms)), nms))
}
```

I Know, I Know

You, probably:



But Watch This:

```
myvars <- list("n", "hr", c("lcl", "ucl"), "pval", "pval_inter")
myvarlabs <- c("n", "Hazard Ratio", "95% CI", "p-value (eff)", "p-value (inter)")
formats <- c(n = "xx", hr = "xx.xx", lcl = "(xx.xx, xx.xx)", pval = "xx.xxxx", pval_inter = "xx.xxxx")
env <- new.env()

lyt <- basic_table() %>%
    split_cols_by_multivar(rep("STUDYID", length = length(myvars)),
                           varlabels = myvarlabs,
                           extra_args = list(model_el = myvars,
                                             cache_env = replicate(length(myvars), list(env)))) %>%
    summarize_row_groups(cfun = cox_model_main_el_direct) %>%
    rtables:::split_rows_by_multivar(c("AGE", "RACE"),
                                     varlabels = c("Age", "Ethnicity"),
                                     split_label = "Covariate:",
                                     indent_mod = -1) %>%
    summarize_row_groups(cfun = cox_model_el_direct,
                         extra_args = list(cov_main = TRUE)) %>%
    analyze_colvars(afun = cox_model_el_direct)
```

No Way!

Yes Way.

```
build_table(lyt, anl)
```

```
## [1] "survival::Surv(AVAL, event) ~ ARM *ARM"
## [1] "survival::Surv(AVAL, event) ~ ARM *AGE"
## [1] "survival::Surv(AVAL, event) ~ ARM *RACE"
```

```
##                                n    Hazard Ratio      95% CI      p-value (eff)   p-value (inter)
## ———————————————————————————————————————————————————————————————————————————————————————————————
## B: Placebo vs A: Drug X        247       0.70       (0.51, 0.96)      0.0293
## Covariate:
##   Age                          247                                                      0.8626
##     34                                  0.70       (0.51, 0.97)
##   Ethnicity                    247                                                      0.9197
##     ASIAN                             0.75       (0.48, 1.16)
##     BLACK OR AFRICAN AMERICAN         0.66       (0.34, 1.28)
##     WHITE                             0.65       (0.33, 1.27)
```

Conditionally-present Analyses (different in different facets)

Example - SEX analyzed only for strata B and C

	A: Drug X	B: Placebo	C: Combination
A			
AGE			
n	38	44	40
mean (sd)	33.1 (5.7)	35.1 (7.9)	34.2 (6.2)
B			
AGE			
n	47	45	43
mean (sd)	33.9 (7.2)	36.0 (9.1)	36.3 (8.4)
SEX			
F	25	27	21
M	21	17	21
U	1	1	1
UNDIFFERENTIATED	0	0	0
C			
AGE			
n	49	45	49
mean (sd)	34.2 (6.6)	35.2 (6.6)	35.6 (8.2)
SEX			
F	33	26	27
M	14	19	19
U	1	0	2
UNDIFFERENTIATED	1	0	1

Layout-Based Solution

None, yet, but issue filed:
https://github.com/Roche/rtables/issues/537

Post-processing Solution

Assigning `NULL` anywhere in the table structure removes the substructure (including any children)

```
lyt <- basic_table() %>%
    split_cols_by("ARM") %>%
    split_rows_by("STRATA1") %>%
    analyze(c("AGE", "SEX"))

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

```
##                      A: Drug X   B: Placebo   C: Combination
## ———————————————————————————————————————————————————————————
## A
##   AGE
##     Mean               33.08       35.11         34.23
##   SEX
##     F                   21          24            18
##     M                   16          19            20
##     U                    1          1             1
##     UNDIFFERENTIATED     0          0             1
## B
##   AGE
##     Mean               33.85       36.00         36.33
##   SEX
##     F                   25          27            21
##     M                   21          17            21
##     U                    1          1             1
##     UNDIFFERENTIATED     0          0             0
## C
##   AGE
##     Mean               34.22       35.18         35.63
##   SEX
##     F                   33          26            27
##     M                   14          19            19
##     U                    1          0             2
##     UNDIFFERENTIATED     1          0             1
```

```
row_paths_summary(tbl)
```

```
## rowname                 node_class    path
## ──────────────────────────────────────────────────────────────────
## A                       LabelRow      STRATA1, A
##   AGE                   LabelRow      STRATA1, A, AGE
##     Mean                DataRow       STRATA1, A, AGE, Mean
##   SEX                   LabelRow      STRATA1, A, SEX
##     F                   DataRow       STRATA1, A, SEX, F
##     M                   DataRow       STRATA1, A, SEX, M
##     U                   DataRow       STRATA1, A, SEX, U
##     UNDIFFERENTIATED    DataRow       STRATA1, A, SEX, UNDIFFERENTIATED
## B                       LabelRow      STRATA1, B
##   AGE                   LabelRow      STRATA1, B, AGE
##     Mean                DataRow       STRATA1, B, AGE, Mean
##   SEX                   LabelRow      STRATA1, B, SEX
##     F                   DataRow       STRATA1, B, SEX, F
##     M                   DataRow       STRATA1, B, SEX, M
##     U                   DataRow       STRATA1, B, SEX, U
##     UNDIFFERENTIATED    DataRow       STRATA1, B, SEX, UNDIFFERENTIATED
## C                       LabelRow      STRATA1, C
##   AGE                   LabelRow      STRATA1, C, AGE
##     Mean                DataRow       STRATA1, C, AGE, Mean
##   SEX                   LabelRow      STRATA1, C, SEX
##     F                   DataRow       STRATA1, C, SEX, F
##     M                   DataRow       STRATA1, C, SEX, M
##     U                   DataRow       STRATA1, C, SEX, U
##     UNDIFFERENTIATED    DataRow       STRATA1, C, SEX, UNDIFFERENTIATED
```

```
tt_at_path(tbl, c("STRATA1", "A", "SEX")) <- NULL
tbl
```

```
##                        A: Drug X   B: Placebo   C: Combination
## ———————————————————————————————————————————————————————————————
## A
##   AGE
##     Mean                 33.08       35.11          34.23
## B
##   AGE
##     Mean                 33.85       36.00          36.33
##   SEX
##     F                     25          27             21
##     M                     21          17             21
##     U                      1           1              1
##     UNDIFFERENTIATED       0           0              0
## C
##   AGE
##     Mean                 34.22       35.18          35.63
##   SEX
##     F                     33          26             27
##     M                     14          19             19
##     U                      1           0              2
##     UNDIFFERENTIATED       1           0              1
```

Key Take-Aways

- Pathing makes specifying the things you want ot find easy and deterministic
    - including when you’re trying to remove them
- Obvious place for further extension of the layouting API

Table Analyzing different subsets of the same data

Example

	A: Drug X	B: Placebo	C: Combination
All Patients	134 (100.0%)	134 (100.0%)	132 (100.0%)
Mean	5.97	5.70	5.62
Older Females	30 (22.4%)	33 (24.6%)	35 (26.5%)
Mean	5.70	5.69	5.29
Younger Males	24 (17.9%)	20 (14.9%)	31 (23.5%)
Mean	7.68	5.88	5.07

Layout-Based Solution

Custom splitting

```
custom_core_split <- function(df, ...) {
    stopifnot(all(c("AGE", "SEX") %in% names(df)))

    make_split_result(
        values = c("ALL", "OLDF", "YOUNGM"),
        datasplit = list(ALL = df,
                         OLDF = subset(df, SEX == "F" & AGE > mean(AGE)),
                         YOUNGM = subset(df, SEX == "M" & AGE < mean(AGE))),
        labels = c("All Patients", "Older Females", "Younger Males"))
}

mysplfun <- make_split_fun(core_split = custom_core_split)

lyt <- basic_table() %>%
    split_cols_by("ARM") %>%
    split_rows_by("STUDYID", split_fun = mysplfun) %>%
    summarize_row_groups() %>%
    analyze("BMRKR1")

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

Result

```
tbl
```

```
##                 A: Drug X      B: Placebo    C: Combination
## ————————————————————————————————————————————————————————
## All Patients    134 (100.0%)   134 (100.0%)    132 (100.0%)
##   Mean             5.97           5.70            5.62
## Older Females    30 (22.4%)     33 (24.6%)      35 (26.5%)
##   Mean             5.70           5.69            5.29
## Younger Males    24 (17.9%)     20 (14.9%)      31 (23.5%)
##   Mean             7.68           5.88            5.07
```

Resulting Table Structure

```
table_structure(tbl)
```

```
## [TableTree] STUDYID
##  [TableTree] ALL [cont: 1 x 3]
##   [ElementaryTable] BMRKR1 (1 x 3)
##  [TableTree] OLDF [cont: 1 x 3]
##   [ElementaryTable] BMRKR1 (1 x 3)
##  [TableTree] YOUNGM [cont: 1 x 3]
##   [ElementaryTable] BMRKR1 (1 x 3)
```

Key Take Aways

- custom split functions can do *anything*
  - including define the different subsets you want to analyze!
- doesn't matter what you "split on" when overriding core behavior
  - cause you're fully overriding core behavior…