collapse and *data.table*

Harmony and High Performance

Sebastian Krantz

2021-01-04

This vignette focuses on using *collapse* with the popular *data.table* package by Matt Dowle and Arun Srinivasan. In contrast to *dplyr* and *plm* whose methods (*grouped_df*, *pseries*, *pdata.frame*) *collapse* supports, there is no deep integration between *collapse* and *data.table*.

However, *collapse* functions non-destructively handle *data.table*'s and the two packages work similarly on the C/C++ side of things, while largely complementary in functionality.

Purpose and Strengths of each Package

Needless to say both *data.table* and *collapse* are high-performance packages that also work well together, but for effective use it is good to also understand where each has its strengths.

- *data.table* offers an enhanced data frame based class to contain data (including list columns). For this class it provides a concise data manipulation syntax which also includes fast aggregation / slit-apply-combine computing, (rolling, non-equi) joins, keying, reshaping, some time-series functionality like lagging and rolling statistics, set operations on tables and a number of very useful other functions like the fast csv reader, fast switches, list-transpose etc.. *data.table* makes data management, and computations on data very easy and salable, supporting huge datasets in a very memory efficient way. The package caters well to the end user by compressing an enormous amount of functionality into two square brackets `[]`. Some of the exported functions are great for programming and also support other classes, but a lot of the functionality and optimization of *data.table* happens under the hood and can only be accessed through the non-standard evaluation table `[i, j, by]` syntax. This syntax has a cost of about 1-3 milliseconds for each call. Memory efficiency and thread-parallelization make *data.table* the star performer on huge data.
- *collapse* is class-agnostic in nature, supporting vectors, matrices, data frames and non-destructively handling all major R classes and objects. It's functionality focuses on advanced statistical computations, by proving fast column-wise grouped and weighted statistical functions, fast and complex data aggregation and transformations, liner fitting, time series and panel data computations, some advanced summary statistics, and recursive processing of lists of data objects. It also offers some powerful supporting functions to perform fast data manipulation, grouping, factor generation, recoding, handling outliers and missing values. *collapse* supports both *tidyverse* (piped) and base R / standard evaluation programming. It is very programmer friendly by making accessible most of it's internal C/C++ based functionality (like grouping objects). *collapse* functions are simple, access the underlying serial C/C++ code quickly and are very strongly optimized on the R side of things (resulting in basic function execution speeds of 10-50 microseconds). This makes *collapse* ideal for advanced and high-performance statistical computing on matrices and large datasets, and tasks requiring fast programs with repeated function executions.

Interoperating and some Do's and Dont's

Applying *collapse* functions to a data.table always gives a data.table back e.g.

```r
library(collapse)
library(magrittr)
library(data.table)

DT <- qDT(wlddev) # collapse::qDT converts objects to data.table using a shallow copy


DT %>% gby(country) %>% gv(9:12) %>% fmean
#                   country      PCGDP   LIFEEX     GINI        ODA
#   1:            Afghanistan   482.1631 47.88216       NA 1351073448
#   2:                Albania  2710.1591 71.34056 29.66000  300307000
#   3:                Algeria  3474.3201 62.72084 34.36667  583888276
#   4:         American Samoa 10189.2155       NA       NA         NA
#   5:                Andorra 40071.4635       NA       NA         NA
#  ---
# 212: Virgin Islands (U.S.) 36049.6948 73.43375       NA         NA
# 213:     West Bank and Gaza  2268.0728 71.12307 34.52500 1491175200
# 214:            Yemen, Rep.  1084.5999 53.01923 35.46667  639058966
# 215:                 Zambia  1299.2385 49.32509 52.68889  683860862
# 216:               Zimbabwe  1205.5424 54.07304 43.20000  358268214

# Same thing, but notice that fmean give's NA's for missing countries
DT[, lapply(.SD, mean, na.rm = TRUE), keyby = country, .SDcols = 9:12]
#                   country      PCGDP   LIFEEX     GINI        ODA
#   1:            Afghanistan   482.1631 47.88216      NaN 1351073448
#   2:                Albania  2710.1591 71.34056 29.66000  300307000
#   3:                Algeria  3474.3201 62.72084 34.36667  583888276
#   4:         American Samoa 10189.2155      NaN      NaN        NaN
#   5:                Andorra 40071.4635      NaN      NaN        NaN
#  ---
# 212: Virgin Islands (U.S.) 36049.6948 73.43375      NaN        NaN
# 213:     West Bank and Gaza  2268.0728 71.12307 34.52500 1491175200
# 214:            Yemen, Rep.  1084.5999 53.01923 35.46667  639058966
# 215:                 Zambia  1299.2385 49.32509 52.68889  683860862
# 216:               Zimbabwe  1205.5424 54.07304 43.20000  358268214

# This also works without magrittr pipes with the collap() function
collap(DT, ~ country, fmean, cols = 9:12)
#                   country      PCGDP   LIFEEX     GINI        ODA
#   1:            Afghanistan   482.1631 47.88216       NA 1351073448
#   2:                Albania  2710.1591 71.34056 29.66000  300307000
#   3:                Algeria  3474.3201 62.72084 34.36667  583888276
#   4:         American Samoa 10189.2155       NA       NA         NA
#   5:                Andorra 40071.4635       NA       NA         NA
#  ---
# 212: Virgin Islands (U.S.) 36049.6948 73.43375       NA         NA
# 213:     West Bank and Gaza  2268.0728 71.12307 34.52500 1491175200
# 214:            Yemen, Rep.  1084.5999 53.01923 35.46667  639058966
# 215:                 Zambia  1299.2385 49.32509 52.68889  683860862
# 216:               Zimbabwe  1205.5424 54.07304 43.20000  358268214
```

By default, *collapse* orders groups in aggregations, which is equivalent to using `keyby` with *data.table*. `gby / fgroup_by` has an argument `sort = FALSE` to yield an unordered grouping equivalent to *data.table*'s `by` on character data[1].

At this data size *collapse* outperforms *data.table* (which might reverse as data size grows, depending in your computer, the number of *data.table* threads used, and the function in question):

```
library(microbenchmark)

microbenchmark(collapse = DT %>% gby(country) %>% get_vars(9:12) %>% fmean,
               data.table = DT[, lapply(.SD, mean, na.rm = TRUE), keyby = country, .SDcols = 9:12])
# Unit: microseconds
#        expr      min       lq      mean   median        uq       max neval cld
#    collapse  456.512  610.691  921.9761  680.082  821.9895  4796.277   100  a
#  data.table 1971.972 2723.900 5647.3481 3135.787 4179.7850 84344.941   100   b
```

It is critical to never do something like this:

```
DT[, lapply(.SD, fmean), keyby = country, .SDcols = 9:12]
#                    country      PCGDP   LIFEEX     GINI        ODA
#   1:           Afghanistan   482.1631 47.88216       NA 1351073448
#   2:               Albania  2710.1591 71.34056 29.66000  300307000
#   3:               Algeria  3474.3201 62.72084 34.36667  583888276
#   4:        American Samoa 10189.2155       NA       NA         NA
#   5:               Andorra 40071.4635       NA       NA         NA
#  ---
# 212: Virgin Islands (U.S.) 36049.6948 73.43375       NA         NA
# 213:    West Bank and Gaza  2268.0728 71.12307 34.52500 1491175200
# 214:           Yemen, Rep.  1084.5999 53.01923 35.46667  639058966
# 215:                Zambia  1299.2385 49.32509 52.68889  683860862
# 216:              Zimbabwe  1205.5424 54.07304 43.20000  358268214
```

The reason is that *collapse* functions are S3 generic with methods for vectors, matrices and data frames among others. So you incur a method-dispatch for every column and every group the function is applied to.

```
fmean
# function(x, ...) UseMethod("fmean")
# <bytecode: 0x000000000d267ef8>
# <environment: namespace:collapse>
methods(fmean)
# [1] fmean.data.frame  fmean.default     fmean.grouped_df* fmean.list*       fmean.matrix
# see '?methods' for accessing help and source code
```

You may now contend that `base::mean` is also S3 generic, but in this `DT[, lapply(.SD, mean, na.rm = TRUE), by = country, .SDcols = 9:12]` code *data.table* does not use `base::mean`, but `data.table:::gmean`, an internal optimized mean function which is efficiently applied over those groups (see `?data.table::GForce`). `fmean` works similar, and includes this functionality explicitly.

```
str(fmean.data.frame)
# function (x, g = NULL, w = NULL, TRA = NULL, na.rm = TRUE, use.g.names = TRUE, drop = TRUE,
#     ...)
#  - attr(*, "srcref")= 'srcref' int [1:8] 63 21 90 1 21 1 4720 4747
#   ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcfile' <environment: 0x000000000d265398>
```

Here we can see the `x` argument for the data, the `g` argument for grouping vectors, a weight vector `w`, different options `TRA` to transform the original data using the computed means, and some functionality regarding missing values (default: removed / skipped), group names (which are added as row-names to a data frame, but not to a *data.table*) etc. So we can also do

[1] Grouping on numeric variables in *collapse* is always ordered.

```r
fmean(gv(DT, 9:12), DT$country)
#           PCGDP   LIFEEX     GINI        ODA
#   1:   482.1631 47.88216       NA 1351073448
#   2:  2710.1591 71.34056 29.66000  300307000
#   3:  3474.3201 62.72084 34.36667  583888276
#   4: 10189.2155       NA       NA         NA
#   5: 40071.4635       NA       NA         NA
#  ---
# 212: 36049.6948 73.43375       NA         NA
# 213:  2268.0728 71.12307 34.52500 1491175200
# 214:  1084.5999 53.01923 35.46667  639058966
# 215:  1299.2385 49.32509 52.68889  683860862
# 216:  1205.5424 54.07304 43.20000  358268214

# Or
g <- GRP(DT, "country")
add_vars(g[["groups"]], fmean(gv(DT, 9:12), g))
#                     country      PCGDP   LIFEEX     GINI        ODA
#   1:            Afghanistan   482.1631 47.88216       NA 1351073448
#   2:                Albania  2710.1591 71.34056 29.66000  300307000
#   3:                Algeria  3474.3201 62.72084 34.36667  583888276
#   4:         American Samoa 10189.2155       NA       NA         NA
#   5:                Andorra 40071.4635       NA       NA         NA
#  ---
# 212: Virgin Islands (U.S.) 36049.6948 73.43375       NA         NA
# 213:    West Bank and Gaza  2268.0728 71.12307 34.52500 1491175200
# 214:           Yemen, Rep.  1084.5999 53.01923 35.46667  639058966
# 215:                 Zambia  1299.2385 49.32509 52.68889  683860862
# 216:               Zimbabwe  1205.5424 54.07304 43.20000  358268214
```

To give us the same result obtained through the high-level functions `gby / fgroup_by` or `collap`. This is however not what *data.table* is doing in `DT[, lapply(.SD, fmean), by = country, .SDcols = 9:12]`. Since `fmean` is not a function it recognizes and is able to optimize, it does something like this,

```r
BY(gv(DT, 9:12), g, fmean) # using collapse::BY
#           PCGDP   LIFEEX     GINI        ODA
#   1:   482.1631 47.88216       NA 1351073448
#   2:  2710.1591 71.34056 29.66000  300307000
#   3:  3474.3201 62.72084 34.36667  583888276
#   4: 10189.2155       NA       NA         NA
#   5: 40071.4635       NA       NA         NA
#  ---
# 212: 36049.6948 73.43375       NA         NA
# 213:  2268.0728 71.12307 34.52500 1491175200
# 214:  1084.5999 53.01923 35.46667  639058966
# 215:  1299.2385 49.32509 52.68889  683860862
# 216:  1205.5424 54.07304 43.20000  358268214
```

which applies `fmean` to every group in every column of the data.

More generally, it is very important to understand that *collapse* is not based around applying functions to data by groups using some universal mechanism: The *dplyr* `data %>% group_by(...) %>% summarize(...) / mutate(...)` and *data.table* `[i, j, by]` syntax are essentially universal mechanisms to apply any function to data by groups. *data.table* additionally internally optimizes some functions (`min, max, mean, median, var, sd, sum, prod, first, last, head, tail`) which they called GForce, `?data.table::GForce`.

collapse instead provides grouped statistical and transformation functions where all grouped computation is done efficiently in C++, and some supporting mechanisms (`fgroup_by`, `collap`) to operate them. In *data.table* words, everything[2] in *collapse*, the *Fast Statistical Functions*, data transformations, time series etc. is GForce optimized.

The full set of optimized grouped statistical and transformation functions in *collapse* is:

```
.FAST_FUN
#  [1] "fmean"      "fmedian"    "fmode"      "fsum"       "fprod"      "fsd"        "fvar"
#  [8] "fmin"       "fmax"       "fnth"       "ffirst"     "flast"      "fNobs"      "fNdistinct"
# [15] "fscale"     "fbetween"   "fwithin"    "fHDbetween" "fHDwithin"  "flag"       "fdiff"
# [22] "fgrowth"
```

Additional optimized grouped functions include `TRA`, `qsu`, `varying`, `fFtest`, `psmat`, `psacf`, `pspacf`, `psccf`.

The nice thing about those GForce (fast) functions provided by *collapse* is that they can be accessed explicitly and programmatically without any overhead as incurred through *data.table*, they cover a broader range of statistical operations (such as mode, distinct values, order statistics), support sampling weights, operate in a class-agnostic way on vectors, matrices, data.frame's and many related classes, and cover transformations (replacing and sweeping, scaling, (higher order) centering, linear fitting) and time series functionality (lags, differences and growth rates, including irregular time series and unbalanced panels).

So if we would want to use `fmean` inside the *data.table*, we should do something like this:

```
# This does not save the grouping columns, we are simply passing a grouping vector to g
# and aggregating the subset of the data table (.SD).
DT[, fmean(.SD, country), .SDcols = 9:12]
#          PCGDP   LIFEEX     GINI        ODA
#   1:   482.1631 47.88216       NA 1351073448
#   2:  2710.1591 71.34056 29.66000  300307000
#   3:  3474.3201 62.72084 34.36667  583888276
#   4: 10189.2155       NA       NA         NA
#   5: 40071.4635       NA       NA         NA
#  ---
# 212: 36049.6948 73.43375       NA         NA
# 213:  2268.0728 71.12307 34.52500 1491175200
# 214:  1084.5999 53.01923 35.46667  639058966
# 215:  1299.2385 49.32509 52.68889  683860862
# 216:  1205.5424 54.07304 43.20000  358268214

# If we want to keep the grouping columns, we need to group .SD first.
DT[, fmean(gby(.SD, country)), .SDcols = c(1L, 9:12)]
#                  country      PCGDP   LIFEEX     GINI        ODA
#   1:           Afghanistan   482.1631 47.88216       NA 1351073448
#   2:               Albania  2710.1591 71.34056 29.66000  300307000
#   3:               Algeria  3474.3201 62.72084 34.36667  583888276
#   4:        American Samoa 10189.2155       NA       NA         NA
#   5:               Andorra 40071.4635       NA       NA         NA
#  ---
# 212: Virgin Islands (U.S.) 36049.6948 73.43375       NA         NA
# 213:    West Bank and Gaza  2268.0728 71.12307 34.52500 1491175200
# 214:           Yemen, Rep.  1084.5999 53.01923 35.46667  639058966
# 215:                Zambia  1299.2385 49.32509 52.68889  683860862
# 216:              Zimbabwe  1205.5424 54.07304 43.20000  358268214
```

[2] Apart from `collapse::BY` which is only an auxiliary function written in base R to perform flexible split-apply combine computing on vectors, matrices and data frames.

Needless to say this kind of programming seems a bit arcane, so there is actually not that great of a scope to use collapse's *Fast Statistical Functions* for aggregations inside *data.table*. I drive this point home with a benchmark:

```
microbenchmark(collapse = DT %>% gby(country) %>% get_vars(9:12) %>% fmean,
               data.table = DT[, lapply(.SD, mean, na.rm = TRUE), keyby = country, .SDcols = 9:12],
               data.table_base = DT[, lapply(.SD, base::mean, na.rm = TRUE), keyby = country, .SDcols =
               hybrid_bad = DT[, lapply(.SD, fmean), keyby = country, .SDcols = 9:12],
               hybrid_ok = DT[, fmean(gby(.SD, country)), .SDcols = c(1L, 9:12)])
# Unit: microseconds
#             expr       min        lq       mean    median        uq       max neval  cld
#         collapse   371.279   450.042   564.5356   497.790   597.750  1830.511   100 a
#       data.table  1718.057  1931.140  2439.3540  2164.305  2903.738  4609.299   100  b
#  data.table_base 12667.204 14092.744 16944.5918 15669.340 18473.340 33735.478   100    d
#       hybrid_bad  6437.578  7336.992  9562.0477  7783.239 10610.222 21015.616   100   c
#        hybrid_ok  1048.683  1225.620  1476.1315  1306.392  1554.283  4109.947   100 a
```

It is evident that *data.table* has some overhead, so there is absolutely no need to do this kind of syntax manipulation.

There is more scope to use *collapse* transformation functions inside *data.table*.

Below some basic examples:

```
# Computing a column containing the sum of ODA received by country
DT[, sum_ODA := sum(ODA, na.rm = TRUE), by = country]
# Same using fsum; "replace_fill" overwrites missing values, "replace" keeps the
DT[, sum_ODA := fsum(ODA, country, TRA = "replace_fill")]
# Same: A native collapse solution using settransform (or its shortcut form)
settfm(DT, sum_ODA = fsum(ODA, country, TRA = "replace_fill"))

# settfm may be more convenient than `:=` for multiple column modifications,
# each involving a different grouping:
  # This computes the percentage of total ODA distributed received by
  # each country both over time and within a given year
settfm(DT, perc_c_ODA = fsum(ODA, country, TRA = "%"),
           perc_y_ODA = fsum(ODA, year, TRA = "%"))
```

The `TRA` argument is available to all *Fast Statistical Functions* (see the macro `.FAST_STAT_FUN`) and offers 10 different replacing and sweeping operations. Note that `TRA()` can also be called directly to replace or sweep with a previously aggregated *data.table*. A set of operators `%rr%`, `%r+%`, `%r-%`, `%r*%`, `%r/%`, `%cr%`, `%c+%`, `%c-%`, `%c*%`, `%c/%` additionally facilitate row- or column-wise replacing or sweeping out vectors of statistics or other *data.table*'s.

Similarly, we can use the following vector valued functions

```
setdiff(.FAST_FUN, .FAST_STAT_FUN)
# [1] "fscale"     "fbetween"   "fwithin"    "fHDbetween" "fHDwithin"  "flag"       "fdiff"
# [8] "fgrowth"
```

for very efficient data transformations:

```
# Centering GDP
DT[, demean_PCGDP := PCGDP - mean(PCGDP, na.rm = TRUE), by = country]
DT[, demean_PCGDP := fwithin(PCGDP, country)]

# Lagging GDP
DT[order(year), lag_PCGDP := shift(PCGDP, 1L), by = country]
```

```r
DT[, lag_PCGDP := flag(PCGDP, 1L, country, year)]

# Computing a growth rate
DT[order(year), growth_PCGDP := (PCGDP / shift(PCGDP, 1L) - 1) * 100, by = country]
DT[, lag_PCGDP := fgrowth(PCGDP, 1L, 1L, country, year)] # 1 lag, 1 iteration

# Several Growth rates
DT[order(year), paste0("growth_", .c(PCGDP, LIFEEX, GINI, ODA)) := (.SD / shift(.SD, 1L) - 1) * 100,
   by = country, .SDcols = 9:12]

# Same thing using collapse
DT %<>% tfm(gv(., 9:12) %>% fgrowth(1L, 1L, country, year) %>% add_stub("growth_"))

# Or even simpler using settransform and the Growth operator
settfmv(DT, 9:12, G, 1L, 1L, country, year, apply = FALSE)

head(DT)
#        country iso3c       date year decade     region     income  OECD PCGDP LIFEEX GINI       ODA
# 1: Afghanistan   AFG 1961-01-01 1960   1960 South Asia Low income FALSE    NA 32.292   NA 114440000
# 2: Afghanistan   AFG 1962-01-01 1961   1960 South Asia Low income FALSE    NA 32.742   NA 233350000
# 3: Afghanistan   AFG 1963-01-01 1962   1960 South Asia Low income FALSE    NA 33.185   NA 114880000
# 4: Afghanistan   AFG 1964-01-01 1963   1960 South Asia Low income FALSE    NA 33.624   NA 236450000
# 5: Afghanistan   AFG 1965-01-01 1964   1960 South Asia Low income FALSE    NA 34.060   NA 302480000
# 6: Afghanistan   AFG 1966-01-01 1965   1960 South Asia Low income FALSE    NA 34.495   NA 370250000
#        sum_ODA perc_c_ODA perc_y_ODA demean_PCGDP lag_PCGDP growth_PCGDP growth_LIFEEX growth_GINI
# 1: 78362260000  0.1460397  0.4534359           NA        NA           NA            NA          NA
# 2: 78362260000  0.2977837  0.7746781           NA        NA           NA      1.393534          NA
# 3: 78362260000  0.1466012  0.3674502           NA        NA           NA      1.353002          NA
# 4: 78362260000  0.3017396  0.6966903           NA        NA           NA      1.322887          NA
# 5: 78362260000  0.3860021  0.8739274           NA        NA           NA      1.296693          NA
# 6: 78362260000  0.4724851  0.9918527           NA        NA           NA      1.277158          NA
#    growth_ODA G1.PCGDP G1.LIFEEX G1.GINI    G1.ODA
# 1:         NA       NA        NA      NA        NA
# 2:  103.90598       NA  1.393534      NA 103.90598
# 3:  -50.76923       NA  1.353002      NA -50.76923
# 4:  105.82347       NA  1.322887      NA 105.82347
# 5:   27.92557       NA  1.296693      NA  27.92557
# 6:   22.40479       NA  1.277158      NA  22.40479
```

Since transformations (`:=` operations) are not highly optimized in *data.table*, *collapse* will be faster in most circumstances. Also time series functionality in *collapse* is significantly faster as it does not require data to be ordered or balanced to compute. For example `flag` computes an ordered lag without sorting the entire data first.

```r
# Lets generate a large dataset and benchmark this stuff
DT_large <- replicate(1000, qDT(wlddev), simplify = FALSE) %>%
    lapply(tfm, country = paste(country, rnorm(1))) %>%
    rbindlist

# 12.7 million Obs
fdim(DT_large)
# [1] 12744000       12

microbenchmark(
```

```
  S1 = DT_large[, sum_ODA := sum(ODA, na.rm = TRUE), by = country],
  S2 = DT_large[, sum_ODA := fsum(ODA, country, TRA = "replace_fill")],
  S3 = settfm(DT_large, sum_ODA = fsum(ODA, country, TRA = "replace_fill")),
  W1 = DT_large[, demean_PCGDP := PCGDP - mean(PCGDP, na.rm = TRUE), by = country],
  W2 = DT_large[, demean_PCGDP := fwithin(PCGDP, country)],
  L1 = DT_large[order(year), lag_PCGDP := shift(PCGDP, 1L), by = country],
  L2 = DT_large[, lag_PCGDP := flag(PCGDP, 1L, country, year)],
  L3 = DT_large[, lag_PCGDP := shift(PCGDP, 1L), by = country], # Not ordered
  L4 = DT_large[, lag_PCGDP := flag(PCGDP, 1L, country)], # Not ordered
  times = 5
)
# Unit: milliseconds
#  expr       min        lq      mean    median        uq       max neval   cld
#    S1  990.4583 1039.2520 1238.4211 1041.5560 1212.7413 1908.0978     5   b
#    S2  674.0067  693.1418  746.4946  717.7288  810.4792  837.1163     5 ab
#    S3  616.6776  708.5758  765.4463  714.6488  890.4242  896.9050     5 ab
#    W1 2511.6147 2858.5451 2929.2528 2909.0332 3101.2354 3265.8355     5   c
#    W2  511.9472  529.6110  593.5401  601.6944  649.9968  674.4512     5 a
#    L1 6951.0171 7579.7876 7842.6786 8098.4062 8134.0731 8450.1088     5     e
#    L2 1007.1748 1146.5024 1191.0843 1197.8687 1260.3636 1343.5121     5 ab
#    L3 5049.3341 5331.6968 5554.5611 5487.8310 5832.1174 6071.8263     5    d
#    L4  619.9049  673.6296  713.7116  691.5862  691.7518  891.6857     5 ab

rm(DT_large)
gc()
#            used  (Mb) gc trigger   (Mb)  max used   (Mb)
# Ncells  2096237 112.0    3930408  210.0   3930408  210.0
# Vcells 21575104 164.7  281847526 2150.4 350699072 2675.7
```

Further *collapse* features supporting *data.table*'s

As mentioned, `qDT` is a flexible and very fast function to create / column-wise convert R objects to *data.table*'s. You can also row-wise convert a matrix to data.table using `mrtl`:

```
# Creating a matrix from mtcars
m <- qM(mtcars)
str(m)
#  num [1:32, 1:11] 21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
#  - attr(*, "dimnames")=List of 2
#   ..$ : chr [1:32] "Mazda RX4" "Mazda RX4 Wag" "Datsun 710" "Hornet 4 Drive" ...
#   ..$ : chr [1:11] "mpg" "cyl" "disp" "hp" ...

# Demonstrating another nice feature of qDT
qDT(m, row.names.col = "car") %>% head
#                  car  mpg cyl disp  hp drat    wt  qsec vs am gear carb
# 1:         Mazda RX4 21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
# 2:     Mazda RX4 Wag 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
# 3:        Datsun 710 22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
# 4:    Hornet 4 Drive 21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
# 5: Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
# 6:           Valiant 18.1   6  225 105 2.76 3.460 20.22  1  0    3    1

# Row-wise conversion to data.table
```

```r
mrtl(m, names = TRUE, return = "data.table") %>% head(2)
#    Mazda RX4 Mazda RX4 Wag Datsun 710 Hornet 4 Drive Hornet Sportabout Valiant Duster 360 Merc 240D
# 1:        21            21       22.8           21.4              18.7    18.1       14.3      24.4
# 2:         6             6        4.0            6.0               8.0     6.0        8.0       4.0
#    Merc 230 Merc 280 Merc 280C Merc 450SE Merc 450SL Merc 450SLC Cadillac Fleetwood
# 1:     22.8     19.2      17.8       16.4       17.3        15.2               10.4
# 2:      4.0      6.0       6.0        8.0        8.0         8.0                8.0
#    Lincoln Continental Chrysler Imperial Fiat 128 Honda Civic Toyota Corolla Toyota Corona
# 1:                10.4              14.7     32.4        30.4           33.9          21.5
# 2:                 8.0               8.0      4.0         4.0            4.0           4.0
#    Dodge Challenger AMC Javelin Camaro Z28 Pontiac Firebird Fiat X1-9 Porsche 914-2 Lotus Europa
# 1:             15.5        15.2       13.3             19.2      27.3            26         30.4
# 2:              8.0         8.0        8.0              8.0       4.0             4          4.0
#    Ford Pantera L Ferrari Dino Maserati Bora Volvo 142E
# 1:           15.8         19.7            15       21.4
# 2:            8.0          6.0             8        4.0
```

The computational efficiency of these functions makes them very useful to use in *data.table* based workflows.

```r
# Benchmark
microbenchmark(qDT(m, "car"), mrtl(m, TRUE, "data.table"))
# Unit: microseconds
#                          expr    min     lq     mean median      uq    max neval cld
#                 qDT(m, "car") 12.049 12.495 13.61080 12.942 13.3880 50.426   100   b
#  mrtl(m, TRUE, "data.table")  5.801  6.694  7.40351  7.140  7.3635 41.055   100  a
```

For example we could regress the growth rate of GDP per capita on the Growth rate of life expectancy in each country and save results in a *data.table*:

```r
library(lmtest)

wlddev %>% fselect(country, PCGDP, LIFEEX) %>%
  # This counts missing values on PCGDP and LIFEEX only
  na_omit(cols = -1L) %>%
  # This removes countries with less than 20 observations
  fsubset(fNobs(PCGDP, country, "replace_fill") > 20L) %>%
  qDT %>%
  # Run estimations by country using data.table
  .[, qDT(coeftest(lm(G(PCGDP) ~ G(LIFEEX))), "Coef"), keyby = country] %>% head
#    country        Coef   Estimate Std. Error    t value    Pr(>|t|)
# 1: Albania (Intercept) -4.5601369   2.642304 -1.7258186 0.093458980
# 2: Albania   G(LIFEEX) 23.7190961   7.721912  3.0716609 0.004171056
# 3: Algeria (Intercept)  0.7775839   1.873481  0.4150477 0.679751420
# 4: Algeria   G(LIFEEX)  0.7589775   1.777133  0.4270798 0.671019328
# 5:  Angola (Intercept) -4.0318568   1.867152 -2.1593617 0.037966192
# 6:  Angola   G(LIFEEX)  4.0824305   1.321935  3.0882227 0.003994191
```

If we only need the coefficients, not the standard errors, we can also use `collapse::flm` together with `mrtl`:

```r
wlddev %>% fselect(country, PCGDP, LIFEEX) %>%
  na_omit(cols = -1L) %>%
  fsubset(fNobs(PCGDP, country, "replace_fill") > 20L) %>%
  qDT %>%
  .[, mrtl(flm(fgrowth(PCGDP)[-1L],
               cbind(Intercept = 1,
                     LIFEEX = fgrowth(LIFEEX)[-1L])), TRUE),
```

```r
    keyby = country] %>% head
#                 country  Intercept     LIFEEX
# 1:              Albania -4.5601369 23.7190961
# 2:              Algeria  0.7775839  0.7589775
# 3:               Angola -4.0318568  4.0824305
# 4: Antigua and Barbuda -8.2491200 38.0097698
# 5:            Argentina  1.9612901 -2.5533460
# 6:              Armenia -1.0018625 13.2867613
```

... which provides a significant speed gain here:

```r
microbenchmark(

A = wlddev %>% fselect(country, PCGDP, LIFEEX) %>%
  na_omit(cols = -1L) %>%
  fsubset(fNobs(PCGDP, country, "replace_fill") > 20L) %>%
  qDT %>%
  .[, qDT(coeftest(lm(G(PCGDP) ~ G(LIFEEX))), "Coef"), keyby = country],

B = wlddev %>% fselect(country, PCGDP, LIFEEX) %>%
  na_omit(cols = -1L) %>%
  fsubset(fNobs(PCGDP, country, "replace_fill") > 20L) %>%
  qDT %>%
  .[, mrtl(flm(fgrowth(PCGDP)[-1L],
             cbind(Intercept = 1,
                   LIFEEX = fgrowth(LIFEEX)[-1L])), TRUE),
    keyby = country]
)
# Unit: milliseconds
#  expr       min        lq      mean    median        uq       max neval cld
#     A 280.44652 313.24532 372.91496 347.49577 399.92101 723.02574   100   b
#     B  10.13653  11.55181  14.58432  12.31043  15.62628  35.42274   100  a
```

Another feature to highlight at this point are *collapse*'s list processing functions, in particular `rsplit`, `rapply2d`, `get_elem` and `unlist2d`. `rsplit` is an efficient recursive generalization of `split`:

```r
DT_list <- rsplit(DT, country + year + PCGDP + LIFEEX ~ region + income)

# Note: rsplit(DT, year + PCGDP + LIFEEX ~ region + income, flatten = TRUE)
# would yield a simple list with interacted categories (like split)

str(DT_list, give.attr = FALSE)
# List of 7
#  $ East Asia & Pacific       :List of 3
#   ..$ High income        :Classes 'data.table' and 'data.frame':  767 obs. of  4 variables:
#   .. ..$ country: chr [1:767] "Australia" "Australia" "Australia" "Australia" ...
#   .. ..$ year   : int [1:767] 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 ...
#   .. ..$ PCGDP  : num [1:767] 19387 19478 19255 20063 21046 ...
#   .. ..$ LIFEEX : num [1:767] 70.8 71 70.9 70.9 70.9 ...
#   ..$ Lower middle income:Classes 'data.table' and 'data.frame':  767 obs. of  4 variables:
#   .. ..$ country: chr [1:767] "Cambodia" "Cambodia" "Cambodia" "Cambodia" ...
#   .. ..$ year   : int [1:767] 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 ...
#   .. ..$ PCGDP  : num [1:767] NA NA NA NA NA NA NA NA NA NA ...
#   .. ..$ LIFEEX : num [1:767] 41.2 41.4 41.5 41.7 41.9 ...
#   ..$ Upper middle income:Classes 'data.table' and 'data.frame':  590 obs. of  4 variables:
```

```
#   .. ..$ country: chr [1:590] "American Samoa" "American Samoa" "American Samoa" "American Samoa" ...
#   .. ..$ year   : int [1:590] 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 ...
#   .. ..$ PCGDP  : num [1:590] NA NA NA NA NA NA NA NA NA NA ...
#   .. ..$ LIFEEX : num [1:590] NA NA NA NA NA NA NA NA NA NA ...
#  $ Europe & Central Asia     :List of 4
#   ..$ High income        :Classes 'data.table' and 'data.frame':  2183 obs. of  4 variables:
#   .. ..$ country: chr [1:2183] "Andorra" "Andorra" "Andorra" "Andorra" ...
#   .. ..$ year   : int [1:2183] 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 ...
#   .. ..$ PCGDP  : num [1:2183] NA NA NA NA NA NA NA NA NA NA ...
#   .. ..$ LIFEEX : num [1:2183] NA NA NA NA NA NA NA NA NA NA ...
#   ..$ Low income         :Classes 'data.table' and 'data.frame':  59 obs. of  4 variables:
#   .. ..$ country: chr [1:59] "Tajikistan" "Tajikistan" "Tajikistan" "Tajikistan" ...
#   .. ..$ year   : int [1:59] 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 ...
#   .. ..$ PCGDP  : num [1:59] NA NA NA NA NA NA NA NA NA NA ...
#   .. ..$ LIFEEX : num [1:59] 56.2 56.6 57 57.4 57.8 ...
#   ..$ Lower middle income:Classes 'data.table' and 'data.frame':  354 obs. of  4 variables:
#   .. ..$ country: chr [1:354] "Georgia" "Georgia" "Georgia" "Georgia" ...
#   .. ..$ year   : int [1:354] 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 ...
#   .. ..$ PCGDP  : num [1:354] NA NA NA NA NA ...
#   .. ..$ LIFEEX : num [1:354] 63.7 64.1 64.5 64.9 65.3 ...
#   ..$ Upper middle income:Classes 'data.table' and 'data.frame':  826 obs. of  4 variables:
#   .. ..$ country: chr [1:826] "Albania" "Albania" "Albania" "Albania" ...
#   .. ..$ year   : int [1:826] 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 ...
#   .. ..$ PCGDP  : num [1:826] NA NA NA NA NA NA NA NA NA NA ...
#   .. ..$ LIFEEX : num [1:826] 62.3 63.3 64.2 64.9 65.5 ...
#  $ Latin America & Caribbean :List of 4
#   ..$ High income        :Classes 'data.table' and 'data.frame':  1062 obs. of  4 variables:
#   .. ..$ country: chr [1:1062] "Antigua and Barbuda" "Antigua and Barbuda" "Antigua and Barbuda" "Ant
#   .. ..$ year   : int [1:1062] 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 ...
#   .. ..$ PCGDP  : num [1:1062] NA NA NA NA NA NA NA NA NA NA ...
#   .. ..$ LIFEEX : num [1:1062] 62.1 62.6 63 63.4 63.8 ...
#   ..$ Low income         :Classes 'data.table' and 'data.frame':  59 obs. of  4 variables:
#   .. ..$ country: chr [1:59] "Haiti" "Haiti" "Haiti" "Haiti" ...
#   .. ..$ year   : int [1:59] 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 ...
#   .. ..$ PCGDP  : num [1:59] 1018 969 1025 986 950 ...
#   .. ..$ LIFEEX : num [1:59] 42.1 42.7 43.3 43.8 44.4 ...
#   ..$ Lower middle income:Classes 'data.table' and 'data.frame':  236 obs. of  4 variables:
#   .. ..$ country: chr [1:236] "Bolivia" "Bolivia" "Bolivia" "Bolivia" ...
#   .. ..$ year   : int [1:236] 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 ...
#   .. ..$ PCGDP  : num [1:236] 995 997 1033 1082 1102 ...
#   .. ..$ LIFEEX : num [1:236] 42.1 42.5 42.8 43.1 43.5 ...
#   ..$ Upper middle income:Classes 'data.table' and 'data.frame':  1121 obs. of  4 variables:
#   .. ..$ country: chr [1:1121] "Belize" "Belize" "Belize" "Belize" ...
#   .. ..$ year   : int [1:1121] 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 ...
#   .. ..$ PCGDP  : num [1:1121] 1072 1093 1115 1138 1161 ...
#   .. ..$ LIFEEX : num [1:1121] 60 60.5 61.1 61.7 62.2 ...
#  $ Middle East & North Africa:List of 4
#   ..$ High income        :Classes 'data.table' and 'data.frame':  472 obs. of  4 variables:
#   .. ..$ country: chr [1:472] "Bahrain" "Bahrain" "Bahrain" "Bahrain" ...
#   .. ..$ year   : int [1:472] 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 ...
#   .. ..$ PCGDP  : num [1:472] NA NA NA NA NA NA NA NA NA NA ...
#   .. ..$ LIFEEX : num [1:472] 51.9 53.2 54.6 55.9 57.2 ...
#   ..$ Low income         :Classes 'data.table' and 'data.frame':  118 obs. of  4 variables:
```

```
#   .. ..$ country: chr [1:118] "Syrian Arab Republic" "Syrian Arab Republic" "Syrian Arab Republic" "Sy
#   .. ..$ year   : int [1:118] 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 ...
#   .. ..$ PCGDP  : num [1:118] NA NA NA NA NA NA NA NA NA NA ...
#   .. ..$ LIFEEX : num [1:118] 52 52.6 53.2 53.8 54.4 ...
#   ..$ Lower middle income:Classes 'data.table' and 'data.frame':  295 obs. of  4 variables:
#   .. ..$ country: chr [1:295] "Djibouti" "Djibouti" "Djibouti" "Djibouti" ...
#   .. ..$ year   : int [1:295] 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 ...
#   .. ..$ PCGDP  : num [1:295] NA NA NA NA NA NA NA NA NA NA ...
#   .. ..$ LIFEEX : num [1:295] 44 44.5 44.9 45.3 45.7 ...
#   ..$ Upper middle income:Classes 'data.table' and 'data.frame':  354 obs. of  4 variables:
#   .. ..$ country: chr [1:354] "Algeria" "Algeria" "Algeria" "Algeria" ...
#   .. ..$ year   : int [1:354] 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 ...
#   .. ..$ PCGDP  : num [1:354] 2466 2078 1628 2133 2201 ...
#   .. ..$ LIFEEX : num [1:354] 46.1 46.6 47.1 47.5 48 ...
#  $ North America             :List of 1
#   ..$ High income:Classes 'data.table' and 'data.frame':  177 obs. of  4 variables:
#   .. ..$ country: chr [1:177] "Bermuda" "Bermuda" "Bermuda" "Bermuda" ...
#   .. ..$ year   : int [1:177] 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 ...
#   .. ..$ PCGDP  : num [1:177] 27838 28437 29007 28641 31042 ...
#   .. ..$ LIFEEX : num [1:177] NA NA NA NA NA ...
#  $ South Asia                :List of 3
#   ..$ Low income         :Classes 'data.table' and 'data.frame':  118 obs. of  4 variables:
#   .. ..$ country: chr [1:118] "Afghanistan" "Afghanistan" "Afghanistan" "Afghanistan" ...
#   .. ..$ year   : int [1:118] 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 ...
#   .. ..$ PCGDP  : num [1:118] NA NA NA NA NA NA NA NA NA NA ...
#   .. ..$ LIFEEX : num [1:118] 32.3 32.7 33.2 33.6 34.1 ...
#   ..$ Lower middle income:Classes 'data.table' and 'data.frame':  295 obs. of  4 variables:
#   .. ..$ country: chr [1:295] "Bangladesh" "Bangladesh" "Bangladesh" "Bangladesh" ...
#   .. ..$ year   : int [1:295] 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 ...
#   .. ..$ PCGDP  : num [1:295] 371 382 391 379 408 ...
#   .. ..$ LIFEEX : num [1:295] 45.8 46.5 47.1 47.7 48.3 ...
#   ..$ Upper middle income:Classes 'data.table' and 'data.frame':  59 obs. of  4 variables:
#   .. ..$ country: chr [1:59] "Maldives" "Maldives" "Maldives" "Maldives" ...
#   .. ..$ year   : int [1:59] 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 ...
#   .. ..$ PCGDP  : num [1:59] NA NA NA NA NA NA NA NA NA NA ...
#   .. ..$ LIFEEX : num [1:59] 37.4 38 38.6 39.3 40 ...
#  $ Sub-Saharan Africa        :List of 4
#   ..$ High income        :Classes 'data.table' and 'data.frame':  59 obs. of  4 variables:
#   .. ..$ country: chr [1:59] "Seychelles" "Seychelles" "Seychelles" "Seychelles" ...
#   .. ..$ year   : int [1:59] 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 ...
#   .. ..$ PCGDP  : num [1:59] 2830 2617 2763 2966 3064 ...
#   .. ..$ LIFEEX : num [1:59] NA NA NA NA NA NA NA NA NA NA ...
#   ..$ Low income         :Classes 'data.table' and 'data.frame':  1593 obs. of  4 variables:
#   .. ..$ country: chr [1:1593] "Benin" "Benin" "Benin" "Benin" ...
#   .. ..$ year   : int [1:1593] 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 ...
#   .. ..$ PCGDP  : num [1:1593] 520 529 504 519 544 ...
#   .. ..$ LIFEEX : num [1:1593] 37.3 37.7 38.2 38.7 39.1 ...
#   ..$ Lower middle income:Classes 'data.table' and 'data.frame':  826 obs. of  4 variables:
#   .. ..$ country: chr [1:826] "Angola" "Angola" "Angola" "Angola" ...
#   .. ..$ year   : int [1:826] 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 ...
#   .. ..$ PCGDP  : num [1:826] NA NA NA NA NA NA NA NA NA NA ...
#   .. ..$ LIFEEX : num [1:826] 33.3 33.6 33.9 34.3 34.6 ...
#   ..$ Upper middle income:Classes 'data.table' and 'data.frame':  354 obs. of  4 variables:
```

```
#   .. ..$ country: chr [1:354] "Botswana" "Botswana" "Botswana" "Botswana" ...
#   .. ..$ year   : int [1:354] 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 ...
#   .. ..$ PCGDP  : num [1:354] 391 406 422 436 453 ...
#   .. ..$ LIFEEX : num [1:354] 50.7 51 51.4 51.7 52.1 ...
```

We can use `rapply2d` to apply a function to each data frame / data.table in an arbitrary nested structure:

```
# This runs region-income level regressions, with country fixed effects
# following Mundlak (1978)
lm_summary_list <- DT_list %>%
  rapply2d(lm, formula = G(PCGDP) ~ G(LIFEEX) + B(G(LIFEEX), country)) %>%
  # Summarizing the results
  rapply2d(summary, classes = "lm")

# This is a nested list of linear model summaries
str(lm_summary_list, give.attr = FALSE)
# List of 7
#  $ East Asia & Pacific       :List of 3
#   ..$ High income        :List of 12
#   .. ..$ call         : language FUN(formula = ..1, data = y)
#   .. ..$ terms        :Classes 'terms', 'formula'  language G(PCGDP) ~ G(LIFEEX) + B(G(LIFEEX), count
#   .. ..$ residuals    : Named num [1:415] -1.39 -2.64 2.7 3.41 2.43 ...
#   .. ..$ coefficients : num [1:3, 1:4] -0.47 1.381 7.4 0.649 0.724 ...
#   .. ..$ aliased      : Named logi [1:3] FALSE FALSE FALSE
#   .. ..$ sigma        : num 4.6
#   .. ..$ df           : int [1:3] 3 412 3
#   .. ..$ r.squared    : num 0.0897
#   .. ..$ adj.r.squared: num 0.0853
#   .. ..$ fstatistic   : Named num [1:3] 20.3 2 412
#   .. ..$ cov.unscaled : num [1:3, 1:3] 2.00e-02 -4.68e-05 -4.13e-02 -4.68e-05 2.48e-02 ...
#   .. ..$ na.action    : 'omit' Named int [1:352] 1 58 59 60 61 62 63 64 65 66 ...
#   ..$ Lower middle income:List of 12
#   .. ..$ call         : language FUN(formula = ..1, data = y)
#   .. ..$ terms        :Classes 'terms', 'formula'  language G(PCGDP) ~ G(LIFEEX) + B(G(LIFEEX), count
#   .. ..$ residuals    : Named num [1:501] -40.7 2.61 -1.21 -3 -2.29 ...
#   .. ..$ coefficients : num [1:3, 1:4] 0.187 0.91 2.619 0.89 0.888 ...
#   .. ..$ aliased      : Named logi [1:3] FALSE FALSE FALSE
#   .. ..$ sigma        : num 6.6
#   .. ..$ df           : int [1:3] 3 498 3
#   .. ..$ r.squared    : num 0.0167
#   .. ..$ adj.r.squared: num 0.0128
#   .. ..$ fstatistic   : Named num [1:3] 4.24 2 498
#   .. ..$ cov.unscaled : num [1:3, 1:3] 0.018183 -0.000238 -0.024148 -0.000238 0.018093 ...
#   .. ..$ na.action    : 'omit' Named int [1:266] 1 2 3 4 5 6 7 8 9 10 ...
#   ..$ Upper middle income:List of 12
#   .. ..$ call         : language FUN(formula = ..1, data = y)
#   .. ..$ terms        :Classes 'terms', 'formula'  language G(PCGDP) ~ G(LIFEEX) + B(G(LIFEEX), count
#   .. ..$ residuals    : Named num [1:294] -31.49 -10.87 3.58 11.85 10.86 ...
#   .. ..$ coefficients : num [1:3, 1:4] 1.898 -0.481 3.404 0.536 0.501 ...
#   .. ..$ aliased      : Named logi [1:3] FALSE FALSE FALSE
#   .. ..$ sigma        : num 4.6
#   .. ..$ df           : int [1:3] 3 291 3
#   .. ..$ r.squared    : num 0.0452
#   .. ..$ adj.r.squared: num 0.0386
```

```
#   .. ..$ fstatistic   : Named num [1:3] 6.89 2 291
#   .. ..$ cov.unscaled : num [1:3, 1:3] 0.013585 0.000335 -0.018191 0.000335 0.011841 ...
#   .. ..$ na.action    : 'omit' Named int [1:296] 1 2 3 4 5 6 7 8 9 10 ...
#  $ Europe & Central Asia     :List of 4
#   ..$ High income        :List of 12
#   .. ..$ call         : language FUN(formula = ..1, data = y)
#   .. ..$ terms        :Classes 'terms', 'formula'  language G(PCGDP) ~ G(LIFEEX) + B(G(LIFEEX), count
#   .. ..$ residuals    : Named num [1:1214] 2.698 -0.596 0.966 3.01 0.211 ...
#   .. ..$ coefficients : num [1:3, 1:4] 3.199 -0.194 -2.218 0.38 0.251 ...
#   .. ..$ aliased      : Named logi [1:3] FALSE FALSE FALSE
#   .. ..$ sigma        : num 3.31
#   .. ..$ df           : int [1:3] 3 1211 3
#   .. ..$ r.squared    : num 0.00287
#   .. ..$ adj.r.squared: num 0.00122
#   .. ..$ fstatistic   : Named num [1:3] 1.74 2 1211
#   .. ..$ cov.unscaled : num [1:3, 1:3] 0.01318 -0.00103 -0.04478 -0.00103 0.00577 ...
#   .. ..$ na.action    : 'omit' Named int [1:969] 1 2 3 4 5 6 7 8 9 10 ...
#   ..$ Low income         :List of 12
#   .. ..$ call         : language FUN(formula = ..1, data = y)
#   .. ..$ terms        :Classes 'terms', 'formula'  language G(PCGDP) ~ G(LIFEEX) + B(G(LIFEEX), count
#   .. ..$ residuals    : Named num [1:31] 3.345 1.032 17.941 0.176 6.946 ...
#   .. ..$ coefficients : num [1:2, 1:4] -4.53 11.48 2.27 4.42 -2 ...
#   .. ..$ aliased      : Named logi [1:3] FALSE FALSE TRUE
#   .. ..$ sigma        : num 9.37
#   .. ..$ df           : int [1:3] 2 29 3
#   .. ..$ r.squared    : num 0.189
#   .. ..$ adj.r.squared: num 0.161
#   .. ..$ fstatistic   : Named num [1:3] 6.75 1 29
#   .. ..$ cov.unscaled : num [1:2, 1:2] 0.0587 -0.0768 -0.0768 0.2224
#   .. ..$ na.action    : 'omit' Named int [1:28] 1 2 3 4 5 6 7 8 9 10 ...
#   ..$ Lower middle income:List of 12
#   .. ..$ call         : language FUN(formula = ..1, data = y)
#   .. ..$ terms        :Classes 'terms', 'formula'  language G(PCGDP) ~ G(LIFEEX) + B(G(LIFEEX), count
#   .. ..$ residuals    : Named num [1:176] 2.44723 1.5123 -0.00885 0.45619 7.76215 ...
#   .. ..$ coefficients : num [1:3, 1:4] 0.433 5.375 0.995 1.605 1.166 ...
#   .. ..$ aliased      : Named logi [1:3] FALSE FALSE FALSE
#   .. ..$ sigma        : num 7.69
#   .. ..$ df           : int [1:3] 3 173 3
#   .. ..$ r.squared    : num 0.112
#   .. ..$ adj.r.squared: num 0.102
#   .. ..$ fstatistic   : Named num [1:3] 10.9 2 173
#   .. ..$ cov.unscaled : num [1:3, 1:3] 0.04353 -0.00139 -0.1401 -0.00139 0.02298 ...
#   .. ..$ na.action    : 'omit' Named int [1:178] 1 2 3 4 5 6 58 59 60 61 ...
#   ..$ Upper middle income:List of 12
#   .. ..$ call         : language FUN(formula = ..1, data = y)
#   .. ..$ terms        :Classes 'terms', 'formula'  language G(PCGDP) ~ G(LIFEEX) + B(G(LIFEEX), count
#   .. ..$ residuals    : Named num [1:397] 0.812 -2.093 -4.025 -6.405 -3.361 ...
#   .. ..$ coefficients : num [1:3, 1:4] 2.907 3.942 -3.026 0.803 0.854 ...
#   .. ..$ aliased      : Named logi [1:3] FALSE FALSE FALSE
#   .. ..$ sigma        : num 8.51
#   .. ..$ df           : int [1:3] 3 394 3
#   .. ..$ r.squared    : num 0.0517
#   .. ..$ adj.r.squared: num 0.0469
```

```
#   .. ..$ fstatistic   : Named num [1:3] 10.7 2 394
#   .. ..$ cov.unscaled : num [1:3, 1:3] 0.008913 0.000103 -0.01683 0.000103 0.010069 ...
#   .. ..$ na.action    : 'omit' Named int [1:429] 1 2 3 4 5 6 7 8 9 10 ...
#  $ Latin America & Caribbean :List of 4
#   ..$ High income         :List of 12
#   .. ..$ call         : language FUN(formula = ..1, data = y)
#   .. ..$ terms        :Classes 'terms', 'formula'  language G(PCGDP) ~ G(LIFEEX) + B(G(LIFEEX), count
#   .. ..$ residuals    : Named num [1:517] 1.72 5.75 6.26 2.32 -1.26 ...
#   .. ..$ coefficients : num [1:3, 1:4] 1.182 0.107 2.167 0.701 0.962 ...
#   .. ..$ aliased      : Named logi [1:3] FALSE FALSE FALSE
#   .. ..$ sigma        : num 4.89
#   .. ..$ df           : int [1:3] 3 514 3
#   .. ..$ r.squared    : num 0.00265
#   .. ..$ adj.r.squared: num -0.00123
#   .. ..$ fstatistic   : Named num [1:3] 0.684 2 514
#   .. ..$ cov.unscaled : num [1:3, 1:3] 0.0206 0.00126 -0.05589 0.00126 0.03875 ...
#   .. ..$ na.action    : 'omit' Named int [1:545] 1 2 3 4 5 6 7 8 9 10 ...
#   ..$ Low income          :List of 12
#   .. ..$ call         : language FUN(formula = ..1, data = y)
#   .. ..$ terms        :Classes 'terms', 'formula'  language G(PCGDP) ~ G(LIFEEX) + B(G(LIFEEX), count
#   .. ..$ residuals    : Named num [1:56] -3.84 6.73 -2.89 -2.68 1.01 ...
#   .. ..$ coefficients : num [1:2, 1:4] 0.01235 -0.72315 1.7478 2.27714 0.00706 ...
#   .. ..$ aliased      : Named logi [1:3] FALSE FALSE TRUE
#   .. ..$ sigma        : num 3.96
#   .. ..$ df           : int [1:3] 2 54 3
#   .. ..$ r.squared    : num 0.00186
#   .. ..$ adj.r.squared: num -0.0166
#   .. ..$ fstatistic   : Named num [1:3] 0.101 1 54
#   .. ..$ cov.unscaled : num [1:2, 1:2] 0.195 -0.242 -0.242 0.331
#   .. ..$ na.action    : 'omit' Named int [1:3] 1 58 59
#   ..$ Lower middle income:List of 12
#   .. ..$ call         : language FUN(formula = ..1, data = y)
#   .. ..$ terms        :Classes 'terms', 'formula'  language G(PCGDP) ~ G(LIFEEX) + B(G(LIFEEX), count
#   .. ..$ residuals    : Named num [1:219] -1.198 2.214 3.396 0.596 1.518 ...
#   .. ..$ coefficients : num [1:3, 1:4] -1.6 -0.227 3.517 3.113 0.768 ...
#   .. ..$ aliased      : Named logi [1:3] FALSE FALSE FALSE
#   .. ..$ sigma        : num 4.03
#   .. ..$ df           : int [1:3] 3 216 3
#   .. ..$ r.squared    : num 0.00377
#   .. ..$ adj.r.squared: num -0.00546
#   .. ..$ fstatistic   : Named num [1:3] 0.408 2 216
#   .. ..$ cov.unscaled : num [1:3, 1:3] 0.5966 0.0077 -0.7315 0.0077 0.0363 ...
#   .. ..$ na.action    : 'omit' Named int [1:17] 1 58 59 60 61 62 63 64 65 117 ...
#   ..$ Upper middle income:List of 12
#   .. ..$ call         : language FUN(formula = ..1, data = y)
#   .. ..$ terms        :Classes 'terms', 'formula'  language G(PCGDP) ~ G(LIFEEX) + B(G(LIFEEX), count
#   .. ..$ residuals    : Named num [1:956] 0.102 0.1392 0.1791 0.1765 0.0504 ...
#   .. ..$ coefficients : num [1:3, 1:4] 2.0093 -0.1695 0.0472 0.385 0.5152 ...
#   .. ..$ aliased      : Named logi [1:3] FALSE FALSE FALSE
#   .. ..$ sigma        : num 4.23
#   .. ..$ df           : int [1:3] 3 953 3
#   .. ..$ r.squared    : num 0.000143
#   .. ..$ adj.r.squared: num -0.00196
```

```
#   .. ..$ fstatistic   : Named num [1:3] 0.0681 2 953
#   .. ..$ cov.unscaled : num [1:3, 1:3] 0.008283 0.000483 -0.015813 0.000483 0.014836 ...
#   .. ..$ na.action    : 'omit' Named int [1:165] 1 58 59 60 117 118 119 176 177 178 ...
#  $ Middle East & North Africa:List of 4
#   ..$ High income         :List of 12
#   .. ..$ call         : language FUN(formula = ..1, data = y)
#   .. ..$ terms        :Classes 'terms', 'formula'  language G(PCGDP) ~ G(LIFEEX) + B(G(LIFEEX), count
#   .. ..$ residuals    : Named num [1:310] -10.844 -12.117 2.016 0.844 -8.769 ...
#   .. ..$ coefficients : num [1:3, 1:4] 1.8 3.83 -2.94 1.16 1.06 ...
#   .. ..$ aliased      : Named logi [1:3] FALSE FALSE FALSE
#   .. ..$ sigma        : num 8.61
#   .. ..$ df           : int [1:3] 3 307 3
#   .. ..$ r.squared    : num 0.0414
#   .. ..$ adj.r.squared: num 0.0351
#   .. ..$ fstatistic   : Named num [1:3] 6.62 2 307
#   .. ..$ cov.unscaled : num [1:3, 1:3] 0.0182 0.00135 -0.025 0.00135 0.01504 ...
#   .. ..$ na.action    : 'omit' Named int [1:162] 1 2 3 4 5 6 7 8 9 10 ...
#   ..$ Low income          :List of 12
#   .. ..$ call         : language FUN(formula = ..1, data = y)
#   .. ..$ terms        :Classes 'terms', 'formula'  language G(PCGDP) ~ G(LIFEEX) + B(G(LIFEEX), count
#   .. ..$ residuals    : Named num [1:26] -2.172 1.655 -0.723 3.193 3.244 ...
#   .. ..$ coefficients : num [1:2, 1:4] -13.32 28.2 6.53 14.47 -2.04 ...
#   .. ..$ aliased      : Named logi [1:3] FALSE FALSE TRUE
#   .. ..$ sigma        : num 5.68
#   .. ..$ df           : int [1:3] 2 24 3
#   .. ..$ r.squared    : num 0.137
#   .. ..$ adj.r.squared: num 0.101
#   .. ..$ fstatistic   : Named num [1:3] 3.8 1 24
#   .. ..$ cov.unscaled : num [1:2, 1:2] 1.32 -2.88 -2.88 6.49
#   .. ..$ na.action    : 'omit' Named int [1:92] 1 2 3 4 5 6 7 8 9 10 ...
#   ..$ Lower middle income:List of 12
#   .. ..$ call         : language FUN(formula = ..1, data = y)
#   .. ..$ terms        :Classes 'terms', 'formula'  language G(PCGDP) ~ G(LIFEEX) + B(G(LIFEEX), count
#   .. ..$ residuals    : Named num [1:179] -1.159 -2.252 4.333 5.389 -0.925 ...
#   .. ..$ coefficients : num [1:3, 1:4] 3.074 1.286 -1.554 1.188 0.701 ...
#   .. ..$ aliased      : Named logi [1:3] FALSE FALSE FALSE
#   .. ..$ sigma        : num 4.47
#   .. ..$ df           : int [1:3] 3 176 3
#   .. ..$ r.squared    : num 0.0191
#   .. ..$ adj.r.squared: num 0.00794
#   .. ..$ fstatistic   : Named num [1:3] 1.71 2 176
#   .. ..$ cov.unscaled : num [1:3, 1:3] 0.070589 -0.000639 -0.082158 -0.000639 0.02456 ...
#   .. ..$ na.action    : 'omit' Named int [1:116] 1 2 3 4 5 6 7 8 9 10 ...
#   ..$ Upper middle income:List of 12
#   .. ..$ call         : language FUN(formula = ..1, data = y)
#   .. ..$ terms        :Classes 'terms', 'formula'  language G(PCGDP) ~ G(LIFEEX) + B(G(LIFEEX), count
#   .. ..$ residuals    : Named num [1:246] -18.053 -23.964 28.706 0.876 1.165 ...
#   .. ..$ coefficients : num [1:3, 1:4] 4.613 0.827 -3.458 3.931 1.399 ...
#   .. ..$ aliased      : Named logi [1:3] FALSE FALSE FALSE
#   .. ..$ sigma        : num 14.2
#   .. ..$ df           : int [1:3] 3 243 3
#   .. ..$ r.squared    : num 0.00264
#   .. ..$ adj.r.squared: num -0.00557
```

```
#   .. ..$ fstatistic   : Named num [1:3] 0.321 2 243
#   .. ..$ cov.unscaled : num [1:3, 1:3] 0.076158 0.000698 -0.094605 0.000698 0.009642 ...
#   .. ..$ na.action    : 'omit' Named int [1:108] 1 58 59 60 117 118 119 120 121 122 ...
#  $ North America              :List of 1
#   ..$ High income:List of 12
#   .. ..$ call         : language FUN(formula = ..1, data = y)
#   .. ..$ terms        :Classes 'terms', 'formula'  language G(PCGDP) ~ G(LIFEEX) + B(G(LIFEEX), count
#   .. ..$ residuals    : Named num [1:125] 4.331 -3.606 1.301 0.101 -0.476 ...
#   .. ..$ coefficients : num [1:3, 1:4] 4.362 -0.968 -9.885 2.178 0.614 ...
#   .. ..$ aliased      : Named logi [1:3] FALSE FALSE FALSE
#   .. ..$ sigma        : num 2.31
#   .. ..$ df           : int [1:3] 3 122 3
#   .. ..$ r.squared    : num 0.0313
#   .. ..$ adj.r.squared: num 0.0154
#   .. ..$ fstatistic   : Named num [1:3] 1.97 2 122
#   .. ..$ cov.unscaled : num [1:3, 1:3] 8.88e-01 -8.29e-05 -3.65 -8.29e-05 7.05e-02 ...
#   .. ..$ na.action    : 'omit' Named int [1:52] 1 2 3 4 5 6 7 8 9 10 ...
#  $ South Asia                 :List of 3
#   ..$ Low income         :List of 12
#   .. ..$ call         : language FUN(formula = ..1, data = y)
#   .. ..$ terms        :Classes 'terms', 'formula'  language G(PCGDP) ~ G(LIFEEX) + B(G(LIFEEX), count
#   .. ..$ residuals    : Named num [1:70] -0.143 -6.864 3.149 -1.887 6.779 ...
#   .. ..$ coefficients : num [1:3, 1:4] 113.08 -1.04 -88.58 67.81 1.36 ...
#   .. ..$ aliased      : Named logi [1:3] FALSE FALSE FALSE
#   .. ..$ sigma        : num 3.66
#   .. ..$ df           : int [1:3] 3 67 3
#   .. ..$ r.squared    : num 0.0734
#   .. ..$ adj.r.squared: num 0.0457
#   .. ..$ fstatistic   : Named num [1:3] 2.65 2 67
#   .. ..$ cov.unscaled : num [1:3, 1:3] 344.155 2.955 -280.751 2.955 0.138 ...
#   .. ..$ na.action    : 'omit' Named int [1:48] 1 2 3 4 5 6 7 8 9 10 ...
#   ..$ Lower middle income:List of 12
#   .. ..$ call         : language FUN(formula = ..1, data = y)
#   .. ..$ terms        :Classes 'terms', 'formula'  language G(PCGDP) ~ G(LIFEEX) + B(G(LIFEEX), count
#   .. ..$ residuals    : Named num [1:259] -0.171 -0.767 -6.554 4.427 -4.754 ...
#   .. ..$ coefficients : num [1:3, 1:4] 1.3742 -0.0169 2.3037 0.6744 0.4604 ...
#   .. ..$ aliased      : Named logi [1:3] FALSE FALSE FALSE
#   .. ..$ sigma        : num 3.36
#   .. ..$ df           : int [1:3] 3 256 3
#   .. ..$ r.squared    : num 0.0303
#   .. ..$ adj.r.squared: num 0.0228
#   .. ..$ fstatistic   : Named num [1:3] 4 2 256
#   .. ..$ cov.unscaled : num [1:3, 1:3] 0.040224 -0.000657 -0.04521 -0.000657 0.018743 ...
#   .. ..$ na.action    : 'omit' Named int [1:36] 1 58 59 60 61 62 63 64 65 66 ...
#   ..$ Upper middle income:List of 12
#   .. ..$ call         : language FUN(formula = ..1, data = y)
#   .. ..$ terms        :Classes 'terms', 'formula'  language G(PCGDP) ~ G(LIFEEX) + B(G(LIFEEX), count
#   .. ..$ residuals    : Named num [1:21] 1.995 2.675 1.824 0.429 -2.042 ...
#   .. ..$ coefficients : num [1:2, 1:4] 2.589 0.853 3.351 3.674 0.773 ...
#   .. ..$ aliased      : Named logi [1:3] FALSE FALSE TRUE
#   .. ..$ sigma        : num 7.66
#   .. ..$ df           : int [1:3] 2 19 3
#   .. ..$ r.squared    : num 0.00283
```

```
#   .. ..$ adj.r.squared: num -0.0497
#   .. ..$ fstatistic   : Named num [1:3] 0.0539 1 19
#   .. ..$ cov.unscaled : num [1:2, 1:2] 0.191 -0.182 -0.182 0.23
#   .. ..$ na.action    : 'omit' Named int [1:38] 1 2 3 4 5 6 7 8 9 10 ...
#  $ Sub-Saharan Africa        :List of 4
#   ..$ High income        :List of 12
#   .. ..$ call         : language FUN(formula = ..1, data = y)
#   .. ..$ terms        :Classes 'terms', 'formula'  language G(PCGDP) ~ G(LIFEEX) + B(G(LIFEEX), count
#   .. ..$ residuals    : Named num [1:36] -11.209 -5.094 -2.981 0.536 7.877 ...
#   .. ..$ coefficients : num [1:2, 1:4] 2.501 -0.727 0.838 0.596 2.984 ...
#   .. ..$ aliased      : Named logi [1:3] FALSE FALSE TRUE
#   .. ..$ sigma        : num 4.98
#   .. ..$ df           : int [1:3] 2 34 3
#   .. ..$ r.squared    : num 0.0419
#   .. ..$ adj.r.squared: num 0.0137
#   .. ..$ fstatistic   : Named num [1:3] 1.49 1 34
#   .. ..$ cov.unscaled : num [1:2, 1:2] 0.0283 -0.00275 -0.00275 0.01432
#   .. ..$ na.action    : 'omit' Named int [1:23] 1 2 3 4 5 6 7 8 9 10 ...
#   ..$ Low income         :List of 12
#   .. ..$ call         : language FUN(formula = ..1, data = y)
#   .. ..$ terms        :Classes 'terms', 'formula'  language G(PCGDP) ~ G(LIFEEX) + B(G(LIFEEX), count
#   .. ..$ residuals    : Named num [1:1163] 0.741 -5.822 2.116 3.902 2.461 ...
#   .. ..$ coefficients : num [1:3, 1:4] -0.31 0.565 0.665 0.679 0.136 ...
#   .. ..$ aliased      : Named logi [1:3] FALSE FALSE FALSE
#   .. ..$ sigma        : num 5.75
#   .. ..$ df           : int [1:3] 3 1160 3
#   .. ..$ r.squared    : num 0.0168
#   .. ..$ adj.r.squared: num 0.0151
#   .. ..$ fstatistic   : Named num [1:3] 9.88 2 1160
#   .. ..$ cov.unscaled : num [1:3, 1:3] 1.39e-02 1.70e-06 -1.55e-02 1.70e-06 5.59e-04 ...
#   .. ..$ na.action    : 'omit' Named int [1:430] 1 58 59 60 117 118 119 176 177 178 ...
#   ..$ Lower middle income:List of 12
#   .. ..$ call         : language FUN(formula = ..1, data = y)
#   .. ..$ terms        :Classes 'terms', 'formula'  language G(PCGDP) ~ G(LIFEEX) + B(G(LIFEEX), count
#   .. ..$ residuals    : Named num [1:693] -7.23 -3.02 1.08 3 0.85 ...
#   .. ..$ coefficients : num [1:3, 1:4] 2.97 1.127 -3.478 0.772 0.218 ...
#   .. ..$ aliased      : Named logi [1:3] FALSE FALSE FALSE
#   .. ..$ sigma        : num 5.29
#   .. ..$ df           : int [1:3] 3 690 3
#   .. ..$ r.squared    : num 0.0419
#   .. ..$ adj.r.squared: num 0.0391
#   .. ..$ fstatistic   : Named num [1:3] 15.1 2 690
#   .. ..$ cov.unscaled : num [1:3, 1:3] 2.13e-02 7.37e-05 -3.31e-02 7.37e-05 1.70e-03 ...
#   .. ..$ na.action    : 'omit' Named int [1:133] 1 2 3 4 5 6 7 8 9 10 ...
#   ..$ Upper middle income:List of 12
#   .. ..$ call         : language FUN(formula = ..1, data = y)
#   .. ..$ terms        :Classes 'terms', 'formula'  language G(PCGDP) ~ G(LIFEEX) + B(G(LIFEEX), count
#   .. ..$ residuals    : Named num [1:280] 0.311 0.534 -0.287 0.508 -0.58 ...
#   .. ..$ coefficients : num [1:3, 1:4] 1.272 0.571 3.614 2.025 0.676 ...
#   .. ..$ aliased      : Named logi [1:3] FALSE FALSE FALSE
#   .. ..$ sigma        : num 11.6
#   .. ..$ df           : int [1:3] 3 277 3
#   .. ..$ r.squared    : num 0.00866
```

```
#   .. ..$ adj.r.squared: num 0.00151
#   .. ..$ fstatistic   : Named num [1:3] 1.21 2 277
#   .. ..$ cov.unscaled : num [1:3, 1:3] 0.030322 0.000224 -0.045458 0.000224 0.003379 ...
#   .. ..$ na.action    : 'omit' Named int [1:74] 1 58 59 60 61 62 63 64 65 66 ...
```

We can turn this list into a *data.table* again by calling first `get_elem` to recursively extract the coefficient matrices and then `unlist2d` to recursively bind them to a new *data.table*:

```
lm_summary_list %>%
  get_elem("coefficients") %>%
  unlist2d(idcols = .c(Region, Income),
           row.names = "Coef",
           DT = TRUE) %>% head
#                  Region              Income                   Coef   Estimate Std. Error    t value
# 1: East Asia & Pacific         High income            (Intercept) -0.4698378  0.6492898 -0.7236180
# 2: East Asia & Pacific         High income              G(LIFEEX)  1.3810085  0.7238075  1.9079775
# 3: East Asia & Pacific         High income B(G(LIFEEX), country)  7.4001516  1.5909029  4.6515420
# 4: East Asia & Pacific Lower middle income            (Intercept)  0.1867100  0.8902693  0.2097231
# 5: East Asia & Pacific Lower middle income              G(LIFEEX)  0.9104478  0.8880453  1.0252267
# 6: East Asia & Pacific Lower middle income B(G(LIFEEX), country)  2.6189241  1.4996172  1.7463951
#        Pr(>|t|)
# 1: 4.697109e-01
# 2: 5.708899e-02
# 3: 4.445740e-06
# 4: 8.339696e-01
# 5: 3.057539e-01
# 6: 8.135888e-02
```

The fact that this is a nested list of matrices, and that we can save both the names of the lists at each level of nesting and the row- and column- names of the matrices make `unlist2d` a significant generalization of `rbindlist`[3].

But why do all this fuzz if we could have simply done:?

```
DT[, qDT(coeftest(lm(G(PCGDP) ~ G(LIFEEX) + B(G(LIFEEX), country))), "Coef"),
   keyby = .(region, income)] %>% head
#                  region              income                   Coef   Estimate Std. Error    t value
# 1: East Asia & Pacific         High income            (Intercept) -0.4698378  0.6492898 -0.7236180
# 2: East Asia & Pacific         High income              G(LIFEEX)  1.3810085  0.7238075  1.9079775
# 3: East Asia & Pacific         High income B(G(LIFEEX), country)  7.4001516  1.5909029  4.6515420
# 4: East Asia & Pacific Lower middle income            (Intercept)  0.1867100  0.8902693  0.2097231
# 5: East Asia & Pacific Lower middle income              G(LIFEEX)  0.9104478  0.8880453  1.0252267
# 6: East Asia & Pacific Lower middle income B(G(LIFEEX), country)  2.6189241  1.4996172  1.7463951
#        Pr(>|t|)
# 1: 4.697109e-01
# 2: 5.708899e-02
# 3: 4.445740e-06
# 4: 8.339696e-01
# 5: 3.057539e-01
# 6: 8.135888e-02
```

Well we might want to do more things with that list of linear models first before tidying it, so this is a more general workflow. We might also be interested in additional statistics like the R-squared or the F-statistic:

[3] `unlist2d` can similarly bind nested lists of arrays, data frames or *data.table*'s

```r
DT_sum <- lm_summary_list %>%
get_elem("coef|r.sq|fstat", regex = TRUE) %>%
  unlist2d(idcols = .c(Region, Income, Statistic),
           row.names = "Coef",
           DT = TRUE)

head(DT_sum)
#                  Region      Income     Statistic                  Coef   Estimate Std. Error
# 1: East Asia & Pacific High income  coefficients           (Intercept) -0.4698378  0.6492898
# 2: East Asia & Pacific High income  coefficients             G(LIFEEX)  1.3810085  0.7238075
# 3: East Asia & Pacific High income  coefficients B(G(LIFEEX), country)  7.4001516  1.5909029
# 4: East Asia & Pacific High income     r.squared                  <NA>         NA         NA
# 5: East Asia & Pacific High income adj.r.squared                  <NA>         NA         NA
# 6: East Asia & Pacific High income    fstatistic                  <NA>         NA         NA
#       t value     Pr(>|t|)         V1     value numdf dendf
# 1: -0.723618 4.697109e-01         NA        NA    NA    NA
# 2:  1.907977 5.708899e-02         NA        NA    NA    NA
# 3:  4.651542 4.445740e-06         NA        NA    NA    NA
# 4:        NA           NA 0.08968990        NA    NA    NA
# 5:        NA           NA 0.08527092        NA    NA    NA
# 6:        NA           NA         NA 20.29651     2   412

# Reshaping to long form:
DT_sum %>%
  melt(1:4, na.rm = TRUE) %>%
  roworderv(1:2) %>% head(20)
#                   Region               Income     Statistic                  Coef   variable
#  1: East Asia & Pacific          High income  coefficients           (Intercept)   Estimate
#  2: East Asia & Pacific          High income  coefficients             G(LIFEEX)   Estimate
#  3: East Asia & Pacific          High income  coefficients B(G(LIFEEX), country)   Estimate
#  4: East Asia & Pacific          High income  coefficients           (Intercept) Std. Error
#  5: East Asia & Pacific          High income  coefficients             G(LIFEEX) Std. Error
#  6: East Asia & Pacific          High income  coefficients B(G(LIFEEX), country) Std. Error
#  7: East Asia & Pacific          High income  coefficients           (Intercept)    t value
#  8: East Asia & Pacific          High income  coefficients             G(LIFEEX)    t value
#  9: East Asia & Pacific          High income  coefficients B(G(LIFEEX), country)    t value
# 10: East Asia & Pacific          High income  coefficients           (Intercept)   Pr(>|t|)
# 11: East Asia & Pacific          High income  coefficients             G(LIFEEX)   Pr(>|t|)
# 12: East Asia & Pacific          High income  coefficients B(G(LIFEEX), country)   Pr(>|t|)
# 13: East Asia & Pacific          High income     r.squared                  <NA>         V1
# 14: East Asia & Pacific          High income adj.r.squared                  <NA>         V1
# 15: East Asia & Pacific          High income    fstatistic                  <NA>      value
# 16: East Asia & Pacific          High income    fstatistic                  <NA>      numdf
# 17: East Asia & Pacific          High income    fstatistic                  <NA>      dendf
# 18: East Asia & Pacific Lower middle income  coefficients           (Intercept)   Estimate
# 19: East Asia & Pacific Lower middle income  coefficients             G(LIFEEX)   Estimate
# 20: East Asia & Pacific Lower middle income  coefficients B(G(LIFEEX), country)   Estimate
#             value
#  1: -4.698378e-01
#  2:  1.381008e+00
#  3:  7.400152e+00
#  4:  6.492898e-01
#  5:  7.238075e-01
```

```
#  6:  1.590903e+00
#  7: -7.236180e-01
#  8:  1.907977e+00
#  9:  4.651542e+00
# 10:  4.697109e-01
# 11:  5.708899e-02
# 12:  4.445740e-06
# 13:  8.968990e-02
# 14:  8.527092e-02
# 15:  2.029651e+01
# 16:  2.000000e+00
# 17:  4.120000e+02
# 18:  1.867100e-01
# 19:  9.104478e-01
# 20:  2.618924e+00
```

As a final example of this kind, lets suppose we are interested in the within-country correlations of all these variables by region and income group:

```
DT[, qDT(pwcor(W(.SD, country)), "Variable"),
   keyby = .(region, income), .SDcols = PCGDP:ODA] %>% head
#                 region              income Variable    W.PCGDP   W.LIFEEX     W.GINI       W.ODA
# 1: East Asia & Pacific         High income  W.PCGDP  1.0000000  0.7446393  0.7939673 -0.25771300
# 2: East Asia & Pacific         High income W.LIFEEX  0.7446393  1.0000000  0.4242715 -0.37949818
# 3: East Asia & Pacific         High income   W.GINI  0.7939673  0.4242715  1.0000000          NA
# 4: East Asia & Pacific         High income    W.ODA -0.2577130 -0.3794982         NA  1.00000000
# 5: East Asia & Pacific Lower middle income  W.PCGDP  1.0000000  0.4154066 -0.1200012 -0.02541587
# 6: East Asia & Pacific Lower middle income W.LIFEEX  0.4154066  1.0000000 -0.1668932  0.10671141
```

In summary: The list processing features, statistical capabilities and efficient converters of *collapse* and the flexibility of *data.table* work well together, facilitating more complex workflows.

Additional Benchmarks

See here or here.

These are all run on a 2 core laptop, so I honestly don't know how *collapse* scales on powerful multi-core machines. My own limited computational resources are part of the reason I did not opt for a thread-parallel package from the start. But a multi-core version of *collapse* will eventually be released, maybe by end of 2021.

References

Mundlak, Yair. 1978. "On the Pooling of Time Series and Cross Section Data." *Econometrica* 46 (1): 69–85.