Advanced and Fast Data Transformation with *collapse* : : CHEAT SHEET

Introduction

collapse is a C/C++ based package supporting advanced (grouped, weighted, time series, panel data and recursive) statistical operations in R, with very efficient low-level vectorizations across both groups and columns.

It also offers a flexible, class-agnostic, approach to data transformation in R: handling matrix and data frame based objects in a uniform, attribute preserving, way, and ensuring seamless compatibility with *dplyr* / (grouped) *tibble*, *data.table*, *xts*, *sf* and *plm* classes for panel data ('pseries', 'pdata.frame').

collapse provides full control to the user for statistical programming - with several ways to reach the same outcome and rich optimization possibilities. Its default is `na.rm = TRUE`, and implemented at very low cost at the algorithm level.

Calling `help("collapse-documentation")` brings up a detailed documentation, which is also available online. See also the *fastverse* package/project for a recommended set of complimentary packages and easy package management.

Row/Column Arithmetic (by Reference)

Column-wise sweeping out of vectors/matrices/DFs/lists

`%cr%, %c+%, %c-%, %c*%, %c/%` e.g. `Z = X %c/% rowSums(X)`

Row-wise sweeping vectors from vectors/matrices/DFs/lists

`%rr%, %r+%, %r-%, %r*%, %r/%` e.g. `Z = X %r/% colSums(X)`

Standard (column-wise) math by reference (returns invisibly)

`%+=%, %-=%, %*=%, %/=%` e.g. `X %-=% rowSums(X)`

Same thing, also supports row-wise operations by reference

```
setop(X, "/", rowSums(X))
setop(X, "/", colSums(X), rowwise = TRUE)
```

Transform Data by (Grouped) Replacing or Sweeping out Statistics (by Reference)

A generalisation of rowwise operations, that also supports sweeping by groups e.g. aggregate statistics

```
   TRA(x, STATS, FUN = "-", g = NULL, set = FALSE)
setTRA(x, STATS, FUN = "-", g = NULL)
```

`x` vector, matrix, or (grouped) data frame / list

`STATS` statistics matching (columns of) `x` (i.e. aggregated vector, matrix or data frame / list)

`FUN` integer/string indicating transformation to perform:

Int.	*String*	*Description*
0	`"replace_NA"`	replace missing values in `x`
1	`"replace_fill"`	replace data and missing values in `x`
2	`"replace"`	replace data but preserve missing values in `x`
3	`"-"`	subtract: `x - STATS(g)`
4	`"-+"`	`x - STATS(g) + fmean(STATS, w = GRPN)`
5	`"/"`	divide: `x / STATS(g)`
6	`"%"`	compute percentages: `x * 100/STATS(g)`
7	`"+"`	add: `x + STATS(g)`
8	`"*"`	multiply: `x * STATS(g)`
9	`"%%"`	modulus: `x %% STATS(g)`
10	`"-%%"`	subtract modulus: `x - x %% STATS(g)`

`g` [optional] (list of) vectors / factors or `GRP()` object

`set` TRUE transforms `x` by reference. `setTRA` is equivalent to `invisible(TRA(..., set = TRUE))`

Fast Statistical Functions

Fast functions to perform column–wise grouped and weighted computations on matrix-like objects

```
fmean, fmedian, fmode, fsum, fprod, fsd, fvar
fmin, fmax, fnth, ffirst, flast, fnobs, fndistinct
```

Syntax

```
FUN(x, g = NULL, [w = NULL], TRA = NULL,
    [na.rm = TRUE], use.g.names = TRUE,
    [drop = TRUE], [nthreads = 1L])
```

`x` vector, matrix, or (grouped) data frame / list

`g` [optional] (list of) vectors / factors or GRP() object

`w` [optional] vector of (frequency) weights

`TRA` [optional] operation to transform data with computed statistics (see `FUN` argument to `TRA()` and Examples)

`drop` drop matrix / data frame dimensions. default `TRUE`

Examples

```
fmean(AirPassengers)  # Vector
## [1] 280.2986
fmean(AirPassengers, w = cycle(AirPassengers))  # Weighted mean
## [1] 284.3397
fmean(EuStockMarkets) # Matrix
##      DAX      SMI      CAC     FTSE
## 2530.657 3376.224 2227.828 3565.643
fmean(EuStockMarkets, drop = FALSE) # Don't drop dimensions
##          DAX      SMI      CAC     FTSE
## [1,] 2530.657 3376.224 2227.828 3565.643
fmean(airquality)     # Data Frame (can also use drop = FALSE)
##      Ozone    Solar.R       Wind       Temp      Month        Day
##  42.129310 185.931507   9.957516  77.882353   6.993464  15.803922
fmean(iris[1:4], g = iris$Species) # Grouped
##            Sepal.Length Sepal.Width Petal.Length Petal.Width
## setosa            5.006       3.428        1.462       0.246
## versicolor        5.936       2.770        4.260       1.326
## virginica         6.588       2.974        5.552       2.026
X = iris[1:4]; g = iris$Species; w <- abs(rnorm(nrow(X)))
fmean(X, g, w)  # Grouped and weighted (random weights)
##            Sepal.Length Sepal.Width Petal.Length Petal.Width
## setosa         5.011663    3.467638     1.504067   0.2525002
## versicolor     5.930365    2.773558     4.238593   1.3136082
## virginica      6.588903    2.978017     5.552375   2.0221178
## Transformations: here centering data on the weighted group median
TRA(X, fmedian(X, g, w), "-", g) |> head(3)
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1          0.1         0.0         -0.1           0
## 2         -0.1         -0.5         -0.1           0
## 3         -0.3         -0.3         -0.2           0
fmedian(X, g, w, TRA = "-") |> head(3) # Same thing: more compact
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1          0.1         0.0         -0.1           0
## 2         -0.1         -0.5         -0.1           0
## 3         -0.3         -0.3         -0.2           0
fmedian(X, g, w, "-", set = TRUE) # Modify in-place (same as setTRA())
head(iris, 3) # Changed iris too, as X = iris[1:4] did a shallow copy
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          0.1         0.0         -0.1           0  setosa
## 2         -0.1         -0.5         -0.1           0  setosa
## 3         -0.3         -0.3         -0.2           0  setosa
```

Basic Computing with R Functions

Apply R functions to rows or columns (by groups)

`dapply(x, FUN, ..., MARGIN = 2)` - column/row apply

`BY(x, g, FUN, ...)` - split-apply-combine computing

Grouping and Ordering

Optimized functions for grouping, ordering, unique values, splitting & recombining, and dealing with factors

`GRP()` - create a grouping object (class 'GRP'): pass to g arg.

```
g <- GRP(iris, ~ Species) # or GRP(iris$Species) or GRP(iris["Species"])
fndistinct(iris[1:4], g)  # Computation without grouping overhead
##            Sepal.Length Sepal.Width Petal.Length Petal.Width
## setosa               15          16            9           6
## versicolor           21          14           19           9
## virginica            21          13           20          12
```

`fgroup_by()` - attach 'GRP' object to data: a class-agnostic grouped frame supporting fast computations

```
mtcars |> fgroup_by(cyl, vs, am) |> ss(1:2)
##               mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4      21   6  160 110  3.9 2.620 16.46  0  1    4    4
## Mazda RX4 Wag  21   6  160 110  3.9 2.875 17.02  0  1    4    4
##
## Grouped by:  cyl, vs, am  [7 | 5 (3.8) 1-12]
# Group Stats: [N. groups | mean (sd) min-max of group sizes]
# Fast Functions also have a grouped_df method: here wt-weighted medians
mtcars |> fgroup_by(cyl, vs, am) |> fmedian(wt) |> head(3)
##   cyl vs am sum.wt  mpg  disp hp drat  qsec gear carb
## 1   4  0  1  2.140 26.0 120.3 91 4.43 16.70    5    2
## 2   4  1  0  8.805 22.8 140.8 95 3.70 20.01    4    2
## 3   4  1  1 14.198 30.4  79.0 66 4.08 18.61    4    1
```

`GRPN()`, `fgroup_vars()`, `fungroup()` - get group count, grouping columns/variables, and ungroup data

`qF()`, `qG()` - quick `as.factor`, and vector grouping object of class 'qG': a factor-light without levels attribute

`group()` - (multivariate) group id ('qG') in appearance order

`groupid()` - run-length-type group id ('qG')

`seqid()` - group-id from integer-sequences ('qG')

`radixorder[v]()` - (multivariate) radix-based ordering

`finteraction()` - fast factor interactions (or return 'qG')

`fdroplevels()` - fast removal of unused factor levels

`f[n]unique()` - fast unique values / rows (by columns)

`gsplit()` - fast splitting vector based on 'GRP' objects

`greorder()` - efficiently reorder `y = unlist(gsplit(x, g))` such that `identical(greorder(y, g), x)`

collapse optimizes grouping using both factors / 'qG' objects and 'GRP' objects. 'GRP' objects contain most information and are thus most efficient for complex computations.

```
X <- iris[1:4]; v <- as.character(iris$Species)
f <- qF(v, na.exclude = FALSE) # Adds 'na.included' class: no NA checks
gv <- group(v) # 'qG' object: first appearance order, with 'na.included'
microbenchmark(fmode(X, v), fmode(X, f), fmode(X, gv), fmode(X, g))

## Unit: microseconds
##          expr     min       lq     mean   median       uq     max neval
##   fmode(X, v) 11.890 12.9150 15.17697 13.3455 13.7350 162.073   100
##   fmode(X, f)  9.225  9.8195 11.33035 10.0860 10.4550  92.947   100
##  fmode(X, gv)  8.569  9.3480 10.73667  9.6555 10.1065  73.021   100
##   fmode(X, g)  6.683  7.2980  7.71620  7.5440  7.7490  13.489   100
```

Quick Conversions

Fast and exact conversion of common data objects

`qM()`, `qDF()`, `qDT()`, `qTBL()` - convert vectors, arrays, data.frames or lists to matrix, data.frame, data.table or tibble

`m[r|c]tl()` - matrix rows/cols to list, data.frame or data.table

`qF()`, `as_numeric_factor()`, `as_character_factor()` - convert to/from factors or all factors in a list / data.frame

Fast Data Manipulation

Minimal overhead implementations

`fselect[<-]()` - select/replace columns

`fsubset()` - subset data (rows and columns)

`ss()` - fast alternative to `[`, particularly for data frames

`[row|col]order[v]()` - reorder (sort) rows and columns

`fmutate()`, `fsummarise()` - *dplyr*-like, incl. `across()` feature

`[f|set]transform[v][<-]()` - transform cols (by reference)

`fcompute[v]()` - compute new cols dropping existing ones

`[f|set]rename()` - rename (any object with 'names' attribute)

`[set]relabel()` - assign/change variable labels ('label' attr.)

`get_vars[<-]()` - select/replace columns (standard eval.)

`[num|cat|char|fact|logi|date]_vars[<-]()` - select/replace columns by data type or retrieve names/indices

`add_vars[<-]()` - add or column-bind columns

Examples

```
mtcars |> fsubset(mpg > fnth(mpg, 0.95), disp:wt, cylinders = cyl)
##                disp hp drat    wt cylinders
## Fiat 128       78.7 66 4.08 2.200         4
## Toyota Corolla 71.1 65 4.22 1.835         4
mtcars |> colorder(cyl, vs, am, pos = 'after') |> head(2)
##                mpg cyl vs am disp  hp drat    wt  qsec gear carb
## Mazda RX4       21   6  0  1  160 110  3.9 2.620 16.46    4    4
## Mazda RX4 Wag   21   6  0  1  160 110  3.9 2.875 17.02    4    4
i <- base::invisible # These are equivalent, the second option is faster:
mtcars |> fgroup_by(cyl, vs, am) |> fmutate(sum_mpg = fsum(mpg)) |> i()
mtcars |> fmutate(sum_mpg = fsum(mpg, list(cyl, vs, am), TRA = 1)) |> i()
# These are also equivalent (weighted means), again the second is faster
mtcars |> fgroup_by(cyl) |> fmutate(across(disp:drat, fmean, wt)) |> i()
mtcars |> ftransformv(disp:drat, fmean, cyl, wt, 1, apply = FALSE) |> i()
# ftransform()/fcompute() support list input and ignore attached groupings
mtcars %>% fgroup_by(cyl) %>% ftransform(fselect(., hp:qsec) %>%
           fmedian(TRA = 1) %>% fungroup() %>% fsum(TRA = "/")) |> i()
# Again a faster equivalent: note the use of 'set' to avoid a deep copy
mtcars %>% ftransform(fselect(., hp:qsec) %>% fmedian(cyl, TRA = 1) %>%
                      fsum(TRA = "/", set = TRUE)) %>% i()
# Aggregation: weighted standard deviations
mtcars |> fgroup_by(vs) |> fsummarise(across(disp:drat, fsd, w = wt))
##   vs      disp       hp      drat
## 1  0 101.80094 54.79388 0.4249447
## 2  1  56.30073 23.17952 0.4915196
# Grouped linear models: .apply = FALSE applies functions to DF subset
qTBL(mtcars) |> fgroup_by(vs) |> fsummarise(across(disp:drat,
     function(x) list(models = list(lm(disp ~., x))), .apply = FALSE))
## # A tibble: 2 x 2
##      vs models
##   <dbl> <list>
## 1     0 <lm>
## 2     1 <lm>
# Adding some columns. Use ftransform<- to also replace existing ones
add_vars(iris) <- num_vars(iris) |> fsum(TRA = '%') |> add_stub("perc_")
```

Multi-Type Aggregation

Convenient interface to complex multi-type aggregations

```
collap(data, by, FUN = fmean, catFUN = fmode,
       cols = NULL, w = NULL, wFUN = fsum,
       custom = NULL, keep.col.order = TRUE, ...)
```

```
# Population weighted mean (PCGDP, LIFEEX) & mode (country), and sum(POP)
collap(wlddev, country + PCGDP + LIFEEX ~ income, w = ~ POP)
##         country              income      PCGDP   LIFEEX          POP
## 1 United States         High income 31284.7366 75.69257  58840837058
## 2      Ethiopia          Low income   557.1427 53.50608  20949161394
## 3         India Lower middle income  1238.8280 60.58651 113837684528
## 4         China Upper middle income  4145.6844 68.26984 119606023798
```

Advanced Transformations

Common transformations (in econometrics)

Scaling, Centering and Averaging

```
fscale(x, g = NULL, w = NULL, na.rm = TRUE,
       mean = 0, sd = 1, ...)
fwithin(x, g = NULL, w = NULL, na.rm = TRUE,
        mean = 0, theta = 1, ...)
fbetween(x, g = NULL, w = NULL, na.rm = TRUE,
         fill = FALSE, ...)
```

Higher-Dimensional Centering/Avg. and Linear Prediction

```
fhdwithin(x, fl, w = NULL, na.rm = TRUE,
          fill = FALSE, lm.method = "qr", ...)
```

`fhdbetween()` - same arguments as `fhdwithin()`

Statistical Operators (function shorthands with extra features)
`STD()`, `W()`, `B()`, `HDW()`, `HDB()`

Examples

```
# Grouped scaling
iris |> fgroup_by(Species) |> fscale() |> head(2)
```

```
##   Species Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1  setosa    0.2666745   0.1899414   -0.3570112  -0.4364923
## 2  setosa   -0.3007180  -1.1290958   -0.3570112  -0.4364923
```

```
STD(iris, ~ Species, stub = FALSE) |> invisible() # Same thing + faster
# Grouped and weighted scaling. Operators support formulas and keep ids
STD(mtcars, mpg + carb ~ cyl, w = ~ wt) |> head(2)
```

```
##               cyl    wt   STD.mpg STD.carb
## Mazda RX4       6 2.620 0.9691687 0.386125
## Mazda RX4 Wag   6 2.875 0.9691687 0.386125
```

```
# Much shorter than fsubset(mpg > fmean(mpg, cyl, TRA = "replace"))
mtcars |> fsubset(mpg > B(mpg, cyl)) |> head(2)
```

```
##               mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4      21   6  160 110  3.9 2.620 16.46  0  1    4    4
## Mazda RX4 Wag  21   6  160 110  3.9 2.875 17.02  0  1    4    4
```

```
# Regression with cyl fixed effects - a la Mundlak (1978)
lm(mpg ~ carb + B(carb, cyl), data = mtcars) |> coef()
```

```
##  (Intercept)          carb B(carb, cyl)
##    34.829652    -0.465511    -4.775032
```

```
# Fast grouped (vs) bivariate regression slopes: mpg ~ carb
mtcars |> fgroup_by(vs) |> fmutate(dm_carb = W(carb)) |>
  fsummarise(beta = fsum(mpg, dm_carb) %/=% fsum(dm_carb^2))
```

```
##   vs       beta
## 1  0 -0.5557241
## 2  1 -2.0706468
```

```
# Residuals from regressing on 'Petal' vars and 'Species' FE
fhdwithin(iris[1:2], iris[3:5]) |> head(2)
```

```
##   Sepal.Length Sepal.Width
## 1   0.14989286   0.1102684
## 2  -0.05010714  -0.3897316
```

```
# Detrending with country-level cubic polynomials
HDW(wlddev, PCGDP + LIFEEX + POP ~ iso3c * poly(year, 3)) |> head(2)
```

```
##    HDW.PCGDP HDW.LIFEEX   HDW.POP
## 43 -258.4069  0.2360285 -317459.1
## 44 -119.5600  0.1136432  -33900.2
```

```
# Note: HD centering/prediction and polynomials requires package 'fixest'
```

Linear Models

Fast (barebones) linear model fitting with 6 different solvers
`flm(y, X, w = NULL, add.icpt = FALSE, method = "lm")`

Fast R^2-based F-test of exclusion restrictions for lm's (with FE)
`fFtest(y, exc, X = NULL, w = NULL, full.df = TRUE)`

Both functions also have formula interfaces:

```
flm(cbind(mpg, disp) ~ hp + carb, weights = wt, mtcars)
```

```
##                     mpg       disp
## (Intercept) 28.48401839  42.155002
## hp          -0.06834996   2.101036
## carb         0.33207257 -38.183910
```

```
# Test the exclusion of cyl-dummies and hp.
fFtest(mpg ~ qF(cyl) + hp | carb + qF(am), weights = wt, mtcars)
```

```
##                R-Sq. DF1 DF2 F-Stat. P-Value
## Full Model     0.812   5  26  22.479   0.000
## Restricted Model 0.674   2  29  30.041   0.000
## Exclusion Rest.  0.138   3  26   6.351   0.002
```

Time Series and Panel Series

Fast and flexible indexed series and data frames: a modern upgrade of *plm*'s 'pseries' and 'pdata.frame'

Turn DF into an 'indexed_frame' using id and/or time vars
`data_ix = findex_by(data, id1, ..., time)`

`data_ix$indexed_series` - columns are 'indexed_series'

`index_df = findex(data_ix)` - retrieve 'index_df': DF of ids

`index_df = with(data_ix, findex(indexed_series))` - can fetch 'index_df' from 'indexed_series' in any caller environment

`data = unindex(data_ix)` - unindex (also 'indexed_series')

`reindex(data, index = index_df)` - reindex / new pointers

'indexed_series' can be 1-or-2D atomic objects. Vectors / time series / matrices can also be indexed directly using:
`reindex(vec/mat, index = vec/index_df)`

`is_irregular()` - irregularity in any index[ed] obj. or time vec

Example: Indexing Panel Data

```
wldi <- wlddev |> findex_by(iso3c, year) # Balanced: 216 countries
fsubset(wldi, 1:2, iso3c, year, PCGDP:POP)
```

```
##   iso3c year PCGDP LIFEEX GINI       ODA     POP
## 1   AFG 1960    NA 32.446   NA 116769997 8996973
## 2   AFG 1961    NA 32.962   NA 232080002 9169410
##
## Indexed by:  iso3c [1] | year [2 (61)]
```

```
# Index stats: [N. ids] | [N. periods (tot.N. periods: (max-min)/GCD)]
LIFEEXi = wldi$LIFEEX # Indexed series
str(LIFEEXi, strict.width = "cut")
```

```
##  'indexed_series' num [1:13176] 32.4 33 33.5 34 34.5 ...
##  - attr(*, "index_df")=Classes 'index_df', 'pindex' and 'data.frame'..
##    ..$ iso3c: Factor w/ 216 levels "ABW","AFG","AGO",..: 2 2 2 2 2 2 ..
##    ..$ year : Ord.factor w/ 61 levels "1960"<"1961"<..: 1 2 3 4 5 6 7..
```

```
LIFEEXi[1:7] # Subsetting indexed series
```

```
## [1] 32.446 32.962 33.471 33.971 34.463 34.948 35.430
##
## Indexed by:  iso3c [1] | year [7 (61)]
```

```
c(is_irregular(LIFEEXi), is_irregular(LIFEEXi[-5])) # Is irregular?
```

```
## [1] FALSE  TRUE
```

Note: 'indexed_series' and frames are supported via existing 'pseries'/'pdata.frame' methods for time series/panel functions.

Fast functions to perform time-based computations on (irregular) time series and (unbalanced) panel data

Lags/Leads, Differences, Growth Rates and Cumulative Sums

```
flag(x, n = 1, g = NULL, t = NULL, fill = NA, ...)
fdiff(x, n = 1, diff = 1, g = NULL, t = NULL,
      fill = NA, log = FALSE, rho = 1, ...)
fgrowth(x, n = 1, diff = 1, g = NULL, t = NULL, fill
 = NA, logdiff = FALSE, scale = 100, power = 1, ...)
fcumsum(x, g = NULL, o = NULL, na.rm = TRUE,
        fill = FALSE, check.o = TRUE, ...)
```

Statistical Operators: `L()`, `F()`, `D()`, `Dlog()`, `G()`

Example: Computing Growth Rates

```
# Ad-hoc use: note that G() supports formulas which fgrowth() doesn't
fgrowth(AirPassengers) |> head()
```

```
## [1]        NA  5.357143 11.864407 -2.272727 -6.201550 11.570248
```

```
G(wlddev, c(1, 10), by = PCGDP ~ iso3c, t = ~ year) |> ss(11:12)
```

```
##   iso3c year G1.PCGDP L10G1.PCGDP
## 1   AFG 1970       NA          NA
## 2   AFG 1971       NA          NA
```

```
wlddev |> fgroup_by(iso3c) |> fselect(iso3c, year, PCGDP, LIFEEX) |>
  fmutate(PCGDP_growth = fgrowth(PCGDP, t = year)) |> head(2)
```

```
##   iso3c year PCGDP LIFEEX PCGDP_growth
## 1   AFG 1960    NA 32.446           NA
## 2   AFG 1961    NA 32.962           NA
```

```
settransform(wlddev, PCGDP_growth = G(PCGDP, g = iso3c, t = year))
# Note: can omit t -> requires consecutive observations and groups
# Usage with indexed series / frames:
```

```
G(wldi) |> head(2) # default: compute growth of num_vars(), keep ids
```

```
##   iso3c year G1.decade G1.PCGDP G1.LIFEEX G1.GINI   G1.ODA   G1.POP
## 1   AFG 1960        NA       NA        NA      NA       NA       NA
## 2   AFG 1961         0       NA  1.590335      NA 98.74969 1.916611
##
## Indexed by:  iso3c [1] | year [2 (61)]
```

```
settransform(wldi, PCGDP_growth = fgrowth(PCGDP))
lm(G(PCGDP) ~ L(G(LIFEEX), 0:2), wldi) |> summary() |> coef() |> round(3)
```

```
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)            1.718      0.081  21.256    0.000
## L(G(LIFEEX), 0:2)--    0.062      0.175   0.353    0.724
## L(G(LIFEEX), 0:2)L1    0.368      0.220   1.672    0.095
## L(G(LIFEEX), 0:2)L2    0.254      0.173   1.468    0.142
```

`psacf()`, `pspacf()`, `psccf()` - panel series ACF/PACF/CCF

`psmat()` - panel data to array conversion/reshaping

Summary Statistics

`qsu()` - fast (grouped, weighted, panel-decomposed) summary statistics for cross-sectional and panel data

```
# Panel data statistics: overall, on group-means and group-centered data
qsu(iris, pid = Sepal.Length ~ Species, higher = TRUE)
```

```
##          N/T    Mean      SD     Min     Max     Skew    Kurt
## Overall  150  5.8433  0.8281     4.3     7.9   0.3118  2.4264
## Between    3  5.8433  0.7951   5.006   6.588  -0.2112     1.5
## Within    50  5.8433  0.5113  4.1553  7.1553   0.1187  3.2633
```

`qtab()` - faster `table()` function, incl. weights & custom funs

`descr()` - detailed statistical description of data.frame

`varying()` - check variation within groups (panel-ids)

`pwcor()`, `pwcov()`, `pwnobs()` - pairwise correlations, covariance and obs. (with P-value and pretty printing)

List Processing

Functions to process (nested) lists (of data objects)

`ldepth()` - level of nesting of list

`is_unlistable()` - is list composed of atomic objects

`has_elem()` - search if list contains certain elements

`get_elem()` - pull out elements from list / subset list

`atomic_elem[<-]()`, `list_elem[<-]()` - get list with atomic / sub-list elements, examining only first level of list

`reg_elem()`, `irreg_elem()` - get full list tree leading to atomic ('regular') or non-atomic ('irregular') elements

`rsplit()` - efficient (recursive) splitting

`t_list()` - efficient list transpose (transpose lists of lists)

`rapply2d()` - recursive apply to lists of data objects

`unlist2d()` - recursive row-binding to data.frame

Example: Nested Linear Models

```
(dl <- mtcars |> rsplit(mpg + hp + carb ~ vs + am)) |> str(max.level = 2)
```

```
## List of 2
##  $ 0:List of 2
##   ..$ 0:'data.frame': 12 obs. of  3 variables:
##   ..$ 1:'data.frame': 6 obs. of  3 variables:
##  $ 1:List of 2
##   ..$ 0:'data.frame': 7 obs. of  3 variables:
##   ..$ 1:'data.frame': 7 obs. of  3 variables:
```

```
nest_lm <- dl |> rapply2d(lm, formula = mpg ~ .)
(nest_coef <- nest_lm |> rapply2d(summary, classes = "lm") |>
    get_elem("coefficients")) |> str(give.attr = FALSE, strict = "cut")
```

```
## List of 2
##  $ 0:List of 2
##   ..$ 0: num [1:3, 1:4] 15.8791 0.0683 -4.5715 3.655 0.0345 ...
##   ..$ 1: num [1:3, 1:4] 26.9556 -0.0319 -0.308 2.293 0.0149 ...
##  $ 1:List of 2
##   ..$ 0: num [1:3, 1:4] 30.896903 -0.099403 -0.000332 3.346033 0.035..
##   ..$ 1: num [1:3, 1:4] 37.0012 -0.1155 0.4762 7.3316 0.0894 ...
```

```
nest_coef |> unlist2d(c("vs", "am"), row.names = "variable") |> head(2)
```

```
##   vs am    variable    Estimate Std. Error  t value    Pr(>|t|)
## 1  0  0 (Intercept) 15.87914500 3.65495315 4.344555 0.001865018
## 2  0  0          hp  0.06832467 0.03449076 1.980956 0.078938069
```

Recode and Replace Values

`recode_num()`, `recode_char()` - recode numeric / character values (+ regex recoding) in matrix-like objects

`replace_[NA|Inf|outliers]()` - replace special values

`pad()` - add (missing) observations / rows i.e. expand objects

(Memory) Efficient Programming

Functions for (memory) efficient R programming

`any|all[v|NA]`, `which[v|NA]`, `%[=|!]=%`, `copyv`, `setv`, `alloc`
`missing_cases`, `na_[insert|rm|omit]`, `vlengths`, `vtypes`, `vgcd`, `frange`, `fnlevels`, `fn[row|col]`, `fdim`, `seq_[row|col]`

```
fsubset(wlddev, year %==% 2010) # 2x faster fsubset(wlddev, year == 2010)
attach(mtcars) # Efficient sub-assignment by reference, various options...
setv(am, 0, vs); setv(am, 1:10, vs); setv(am, 1:10, vs[10:20])
```

Small (Helper) Functions

Functions for (meta-)programming and attributes

`.c`, `massign`, `%=%`, `vlabels[<-]`, `setLabels`, `vclasses`, `namlab`, `[add|rm]_stub`, `%!in%`, `ckmatch`, `all_identical`, `all_obj_equal`, `all_funs`, `set[Dim|Row|Col]names`, `unattrib`, `setAttrib`, `copyAttrib`, `copyMostAttrib`

```
.c(var1, var2, var3) # Non-standard concatenation
```

```
## [1] "var1" "var2" "var3"
```

```
.c(values, vectors) %=% eigen(cov(mtcars)) # Multiple Assignment
# Variable labels: vlabels[<-], [set]relabel() etc. namlab() shows summary
namlab(wlddev[c(2, 9)], N = TRUE, Ndist = TRUE, class = TRUE)
```

```
##   Variable   Class     N Ndist                                Label
## 1    iso3c  factor 13176   216                         Country Code
## 2    PCGDP numeric  9470  9470 GDP per capita (constant 2010 US$)
```

API Extensions

Shorthands for frequently used functions

`fselect -> slt, fsubset -> sbt, fmutate -> mtt, [f/set]transform[v] -> [set]tfm[v], fsummarise -> smr, across -> acr, fgroup_by -> gby, finteraction -> itn, findex_by -> iby, findex -> ix, frename -> rnm, get_vars -> gv, num_vars -> nv, add_vars -> av`

Namespace masking

Can set `option(collpse_mask = c(...))` with a vector of functions starting with f-, to export versions without f-, masking base R or *dplyr*. A few keywords exist to mask multiple functions, see `help("collapse-options")`. This allows clean & fast code, but poses additional namespace challenges:

```
# Masking all f- functions and specials n = GRPN and table = qtab
options(collapse_mask = "all")
library(collapse)
# The following is 100% collapse code, apart from the base pipe

wlddev |>
  subset(year >= 1990) |>
  group_by(year) |>
  summarise(n = n(), across(PCGDP:GINI, mean, w = POP))

with(mtcars, table(cyl, vs, am))
sum(mtcars)
diff(EuStockMarkets)
droplevels(wlddev)
mean(nv(iris), g = iris$Species)
scale(nv(GGDC10S), g = GGDC10S$Variable)
unique(GGDC10S, cols = c("Variable", "Country"))
range(wlddev$date)

wlddev |>
  index_by(iso3c, year) |>
  mutate(PCGDP_lag = lag(PCGDP),
         PCGDP_diff = PCGDP - PCGDP_lag,
         PCGDP_growth = growth(PCGDP)) |> unindex()
```

The best way to set this option is inside an `.Rprofile` file placed in the user or project directory. Use it carefully.