Improved Data Aggregation and Summary Statistics in R, with *collap* and *qsu*

Sebastian Krantz

February 26, 2019

Abstract

While there already exist a number of different functions and packages in R to aggregate data and compute summary statistics, none of the available solutions offers a flexible method to aggregate multivariate (multi-type) datasets in a single computational step. R also lacks a command to compute summary statistics appropriate to multi-level (panel) data structures, and a simple method to obtain between-or within-transformed datasets for analytical use. In addition, many aggregation solutions don't provide very tidy output, lack automation or flexibility in the syntax and the way inputs can be passed, or perform slow on large datasets. With *collap* and *qsu* I intend to thoroughly fill these gaps while accommodating existing functionality. Both functions can perform a broad range of aggregation and summarizing tasks on a wide variety of data objects, while providing the greatest conceivable flexibility to the user and tidy output. Both functions are built from base R, *collap* is slightly faster than *aggregate* in the default mode. Through an (optional) internal integration with the *data.table* package, both functions can also perform extremely fast when it comes to large datasets.

1 Collap

The creation of *collap* was inspired by the STATA command *collapse*, but *collap* is not simply a reproduction of *collapse* for R, but a more advanced, flexible and faster aggregation command that currently offered in either language. The function is built from base R, and optionally as a wrapper around *data.table*, with the key aims of providing an easier user-interface and a greater range of convenience and functionality without compromizing on speed. Among it's key innovations is the large flexibility in inputs and outputs, and a new approach to data aggregation which recognizes that most datasets are comprised of numeric and categorical variables on which separate operations need to be performed in a multivariate aggregation task. In its custom mode, *collap* provides the full functionality of STATA's *collapse* e.g. the possibility to manually assign different columns of a multivariate dataset to different functions and then aggregating by multiple groups. *collap* in it's default mode however features automatic data type recognition and thus allows the user to simply specify the operation(s) to be performed on numeric and categorical variables. These features, together with multi-function calls, sensible default settings in the arguments, flexible and tidy output, and the possibility to harness the full speed of *data.table*, render *collap* an very convenient tool to use on datasets of all shapes and sizes.

Below I briefly list the key features of *collap* which distinguish it from existing functions such as *aggregate, data.table, plyr, dplyr, doBy::summaryBy, base::by* and the *apply* family. Afterwards I will briefly outline the syntax of the function and then swiftly turn to demonstrate its functionality. I end by benchmarking the function directly against *aggregate* and *data.table*.

1.1 Key Features

- Multivariate data aggregation with datasets of different types (automatic recognition of numeric and categorical variables) + aggregation of data.tables, vectors, and numeric or categorical matrices
- Maximum flexibility in the passing of inputs and the format of the output obtained, powered by a simple and parsimonious syntax
- Fully custom aggregation by passing different aggregator functions to the columns of a dataset
- Possibility to apply multiple aggregator functions to a dataset and obtain the output in a wide- or long format, or as a list of datasets

- Option to obtain between-transformed data (data that is aggregated by group but expanded to the original dimensions and row-order)
- Tidy output (preserved names and column order, rows sorted by aggregation groups)
- Sensible default settings in the arguments (mean for numeric columns, mode for categorical columns, NA's are removed, NaN's accruing during aggregation are replaced with NA's etc..)
- Optional speed improvement with the built-in *data.table* option
- Full compatibility with *data.table*: supplying a *data.table* to *collap* will toggle internal use of *data.table* for aggregation and output as *data.table*
- Option to parallelize the computation of multiple functions for further speed improvement

1.2 Syntax of *collap*

Usage

```
collap(X, by = NULL, FUN = mean, catFUN = Mode, factors = "as.categorical", custom = NULL,
       custom.names = TRUE, collapse = TRUE, sort = TRUE, na.rm = TRUE, replace.nan = TRUE,
       reshape.long = FALSE, show.statistic = TRUE, as.list = FALSE,
       dropcat = FALSE, dropby = FALSE, data.table = FALSE, parallel = FALSE)#, ...
```

Arguments

X	A vector, matrix, list, data.frame or data.table to aggregate (anything that can be coerced to data.frame)
by	Columns to aggregate by, **either contained in X** and indicated using a one-or two sided formula (two-sided if only certain columns in X are to be aggregated), column indices, a vector of column names, or a string of comma-separated column names, **or externally supplied** in form of a vector, list of vectors or data.frame, with the number of elements/rows matching that of X. If 'by' is left empty, columns are fully aggregated.
FUN	Function(s) to apply to numeric columns in X, defaults to the mean. A single function can be supplied without quotes. Multiple functions can be supplied as a character vector, string of comma-separated function names, or as a list of functions (preferably named). Ad-hoc functions can be supplied.
catFUN	Function(s) to apply to categorical columns in X, defaults to the Mode. If all elements in a group defined by 'by' are distinct, the Mode defaults to the first element. Multiple functions can be supplied in the same manor as to 'FUN'.
factors	Specifies treatment of factor variables. Default is treatment as categorical variables. Alternatively factors can be coerced to numerical variables by spcifying "as.numeric", or the factor levels can be extracted and coerced to a numerical variable by specifying "as.numeric.fractor" (internally defined as: as.numeric.factor <- function(x) {as.numeric(levels(x))[x]})
custom	Option to supply a custom vector or list of functions whose length must match the number of columns to be aggregated. Alternatively a named list can be provided with the names being the comma-separated names of the columns to be aggregated by different functions, i.e. list("var1,var2,var3" = mean, var4 = median, "var7,var8" = sd).
custom.names	Interact the column names with the respective function names in 'custom'.
na.rm	Removes missing values from all columns before applying any functions. This is done internally in *collap*, thus it is not required for functions in 'FUN' or 'catFUN' to have a 'na.rm' argument.
replace.nan	Replaces NaN values with NA values. NaN's are frequently generated if na.rm = TRUE, and aggregation takes place over an empty subset.
sort	Sort restores the columns back to their original order after aggregation. If sort = FALSE, the dataset is returned with the 'by' columns in front, and the other columns following in the order of computation (first numeric columns and then categorical columns, or columns in the order they are passed to 'custom').

collapse If collapse = FALSE, the aggregated data will be matched with the original data in the 'by' argument and *collap* will return a dataset that is aggregated but of the same dimensions and row-order as the original data, i.e. a between-transformed dataset.

reshape.long If multiple functions are supplied to either 'FUN' or 'catFUN', by default *collap* returns a wider dataset. If reshape.long = TRUE, then a long form of the dataset is returned with an additional column 'Statistic' indicating the function used for aggregation.

show.statistic If multiple functions are called and reshape.long = TRUE, show.statistic = FALSE can be called to omit the 'Statistic' column and instead make appropriate row.names.

as.list Optionally the output can be requested as a list of vectors or data.frames. There are two options here: If as.list = "by", then a list will be returned whose elements are the aggregated output for each group in 'by'. If multiple functions are supplied to either 'FUN' or 'catFUN', calling as.list = "FUN" will return a list with the dataset aggregated by the different functions. as.list = "by" may come at some slight extra computational cost but as.list = "FUN" does not.

dropcat Drop all categorical variables apart from identifiers in 'by' (i.e. don't perform aggregation on them).

dropby Drop the columns in 'by' from the final output.

data.table By default *collap* is built as a wrapper around *aggregate.data.frame*. Calling this argument will internally use *data.table* as workhorse function, yielding significant speed improvements for large datasets.

parallel If multiple functions are supplied to 'FUN' or 'catFUN', parallel = TRUE will automatically parallelize computation on $k-1$ of the available cores (using the *parLapply* function from the *parallel* package). The argument works together with data.table = TRUE to guarantee maximum performance on tasks involving large datsets and multiple functions.

... Additional arguments supplied to 'FUN', 'catFUN' or to *aggregate.data.frame* in the default mode.

1.3 Demonstration

To demonstrate *collap*, I download 4 US macroeconomic time-series from the Federal Reserve Bank of St. Louis database: The Real Gross Domestic Product (GDPC1), the Civilian Noninstitutional Population (CNP16OV), the Gross Domestic Product: Implicit Price Deflator (GDPDEF), and the Effective Federal Funds Rate (FEDFUNDS). The output shows that real GDP and it's deflator are only available at quarterly frequancy, whereas population and the interest rate are available as monthly series. Furthermore, the 'Date' variable is supplied as a character string.

```
# Download 40 years of US macroeconomic data
library(fImport)
data = as.data.frame(fredSeries(c("GDPC1","CNP16OV","GDPDEF","FEDFUNDS"),
                                from = "1979-01-01"))
data$Date = rownames(data); rownames(data) = NULL
data$Year = as.numeric(substr(data$Date,1,4))
data$Quarter = rep(1:4,100,each=3)[1:nrow(data)]
str(data)

## 'data.frame': 481 obs. of  7 variables:
##  $ GDPC1   : num  6742 NA NA 6749 NA ...
##  $ CNP16OV : num  163516 163726 164027 164162 164459 ...
##  $ GDPDEF  : num  37.5 NA NA 38.4 NA ...
##  $ FEDFUNDS: num  10.1 10.1 10.1 10 10.2 ...
##  $ Date    : chr  "1979-01-01" "1979-02-01" "1979-03-01" "1979-04-01" ...
##  $ Year    : num  1979 1979 1979 1979 1979 ...
##  $ Quarter : int  1 1 1 2 2 2 3 3 3 4 ...
```

Since these data need to be at the same quarterly frequency to be useful for macroeconomic analysis, I use *collap* to aggregate them:

```r
head(data)

##      GDPC1 CNP16OV GDPDEF FEDFUNDS       Date Year Quarter
## 1 6741.854  163516 37.476    10.07 1979-01-01 1979       1
## 2       NA  163726     NA    10.06 1979-02-01 1979       1
## 3       NA  164027     NA    10.09 1979-03-01 1979       1
## 4 6749.063  164162 38.394    10.01 1979-04-01 1979       2
## 5       NA  164459     NA    10.24 1979-05-01 1979       2
## 6       NA  164721     NA    10.29 1979-06-01 1979       2

# Collapse data
datac = collap(data, ~ Year + Quarter)
head(datac)

##      GDPC1  CNP16OV GDPDEF FEDFUNDS       Date Year Quarter
## 1 6741.854 163756.3 37.476 10.07333 1979-01-01 1979       1
## 2 6749.063 164447.3 38.394 10.18000 1979-04-01 1979       2
## 3 6799.200 165199.7 39.234 10.94667 1979-07-01 1979       3
## 4 6816.203 166054.7 39.962 13.57667 1979-10-01 1979       4
## 5 6837.641 166762.3 40.801 15.04667 1980-01-01 1980       1
## 6 6696.753 167415.7 41.772 12.68667 1980-04-01 1980       2
```

The output shown provided by *collap* is exactly the same dataset but now at quarterly frequency. *collap* performed this operation by first extracting the "Year" and "Quarter" columns to create groups to aggregate over, then it removed missing values from the data (as na.rm = TRUE by default) and applied the mean (FUN default) to the 4 series, and the mode (catFUN default) to the 'Date' column. The mode chose the first date in each year and quarter since all dates are distinct. *collap* then combined the columns again, put them back into the original order (as sort = TRUE by default), and replaced NaN's with NA's[1] (as replace.nan = TRUE by default). Having outlined basic working principles, I now turn to demonstrate some of the flexibility of *collap* by showing the different ways inputs can be supplied to the function:

```r
# Alternative ways to call the above operation:

datac2 = collap(data,6:7) # using column indices
datac3 = collap(data,c("Year","Quarter")) # a vector of column names
datac4 = collap(data,"Year,Quarter") # a string of comma-separated column names

# These three yield identical output to the formula interface
all(identical(datac,datac2),identical(datac,datac3),identical(datac,datac4))

## [1] TRUE

# One can also supply a vector, list of vectors or data.frame to the 'by' argument
datac5 = collap(data[-(6:7)],data[6:7])
# here however collap is unable to restore the original column order
head(datac5,3)

##   Year Quarter    GDPC1  CNP16OV GDPDEF FEDFUNDS       Date
## 1 1979       1 6741.854 163756.3 37.476 10.07333 1979-01-01
## 2 1979       2 6749.063 164447.3 38.394 10.18000 1979-04-01
## 3 1979       3 6799.200 165199.7 39.234 10.94667 1979-07-01

# the previous output is identical to any of the former if sort = FLASE
head(collap(data, ~ Year + Quarter, sort = FALSE),3)

##   Year Quarter    GDPC1  CNP16OV GDPDEF FEDFUNDS       Date
## 1 1979       1 6741.854 163756.3 37.476 10.07333 1979-01-01
## 2 1979       2 6749.063 164447.3 38.394 10.18000 1979-04-01
## 3 1979       3 6799.200 165199.7 39.234 10.94667 1979-07-01
```

The two-sided formula interface is useful to aggregate only certain columns, i.e. here only real GDP and it's deflator:

[1]NaN's occur when one aggregates over missing values with na.rm = TRUE, i.e. mean(c(NA,NA), na.rm = TRUE) gives NaN. This replacement is only done if replace.nan = TRUE. Setting this argument to FALSE gives a slight speed improvement.

```r
# A two-sided formula serves to aggregate only a subset of the data
head(collap(data, GDPC1 + GDPDEF ~ Year + Quarter),3)

##      GDPC1 GDPDEF Year Quarter
## 1 6741.854 37.476 1979       1
## 2 6749.063 38.394 1979       2
## 3 6799.200 39.234 1979       3
```

With the 'dropcat' and 'dropby' arguments, *collap* offers additional flexibility for certain cases. The 'dropcat' argument can be used to drop all categorical variables (except for those in 'by') prior to aggregation. This is particularly handy when considering that many datasets from statistical agencies provide not only main identifiers, but also some other identifiers and variables providing information about the dataset such as regional codes, series codes etc. which are often categorical. With 'dropcat' these variables can now be dropped, allowing the user to maintain only the identifiers and the aggregated numerical data. Similarly the 'dropby' argument allows the user to drop the aggregation identifiers supplied to 'by'. This is useful in cases where for example an external aggregation ID is supplied which should not be part of the resulting dataset, or when a single column is aggregated and the output is desired in form of a vector. The 'collapse' argument gives between-transformed data, which can be used to run a between-regression, or to obtain within-transformed data by subtracting it from the original data.

```r
# 'dropcat' removes categorical columns (here 'Date')
head(collap(data, ~ Year + Quarter, dropcat = TRUE),3)

##      GDPC1  CNP16OV GDPDEF FEDFUNDS Year Quarter
## 1 6741.854 163756.3 37.476 10.07333 1979       1
## 2 6749.063 164447.3 38.394 10.18000 1979       2
## 3 6799.200 165199.7 39.234 10.94667 1979       3

# 'dropby' omits the 'by' columns from the output, here taking 5-year averages of the data
head(collap(data, round(data$Year/5)*5, dropby = TRUE),3)

##      GDPC1  CNP16OV   GDPDEF  FEDFUNDS       Date   Year Quarter
## 1 6818.057 168752.8 44.11150 13.296667 1979-01-01 1980.5     2.5
## 2 7887.799 178428.7 54.24225  8.175000 1983-01-01 1985.0     2.5
## 3 9292.875 188779.8 63.44695  6.818667 1988-01-01 1990.0     2.5

# This gives a vector of quarterly GDP
head(collap(data, GDPC1 ~ Year + Quarter, dropby = TRUE))

## [1] 6741.854 6749.063 6799.200 6816.203 6837.641 6696.753

# Setting collapse = FALSE gives between-transformed data, the original row-order is restored
head(collap(data, ~ Year + Quarter, collapse = FALSE))

##      GDPC1  CNP16OV GDPDEF FEDFUNDS       Date Year Quarter
## 1 6741.854 163756.3 37.476 10.07333 1979-01-01 1979       1
## 2 6741.854 163756.3 37.476 10.07333 1979-01-01 1979       1
## 3 6741.854 163756.3 37.476 10.07333 1979-01-01 1979       1
## 4 6749.063 164447.3 38.394 10.18000 1979-04-01 1979       2
## 5 6749.063 164447.3 38.394 10.18000 1979-04-01 1979       2
## 6 6749.063 164447.3 38.394 10.18000 1979-04-01 1979       2
```

When more than one function is called, by default *collap* outputs a wider dataset, but the order of columns is still kept as long as sort = TRUE[2]. If reshape.long = TRUE and multiple functions are passed to either 'FUN' or 'catFUN', the data are returned in long form and unaffected columns are duplicated. If multiple functions are supplied to both 'FUN' and 'catFUN', the data are always returned in the wide-form, even if reshape.long = TRUE.

```r
library(dplyr) # dplyr contains the functions 'first' and 'last'

# Applying multiple functions to numeric variables
head(collap(data, ~ Year + Quarter,"mean,length"),3)
```

[2] Calling *length* here serves to count the number of non-missing observations aggregated over to produce each value in the output table, since na.rm = TRUE by default.

```
##   GDPC1.mean GDPC1.length CNP16OV.mean CNP16OV.length GDPDEF.mean GDPDEF.length FEDFUNDS.mean
## 1   6741.854            1     163756.3              3      37.476             1      10.07333
## 2   6749.063            1     164447.3              3      38.394             1      10.18000
## 3   6799.200            1     165199.7              3      39.234             1      10.94667
##   FEDFUNDS.length       Date Year Quarter
## 1               3 1979-01-01 1979       1
## 2               3 1979-04-01 1979       2
## 3               3 1979-07-01 1979       3

# If sort = FALSE, variables are sorted in the order of computation
head(collap(data, ~ Year + Quarter,"mean,length", sort = FALSE),3)

##   Year Quarter GDPC1.mean CNP16OV.mean GDPDEF.mean FEDFUNDS.mean GDPC1.length CNP16OV.length
## 1 1979       1   6741.854     163756.3      37.476      10.07333            1              3
## 2 1979       2   6749.063     164447.3      38.394      10.18000            1              3
## 3 1979       3   6799.200     165199.7      39.234      10.94667            1              3
##   GDPDEF.length FEDFUNDS.length       Date
## 1             1               3 1979-01-01
## 2             1               3 1979-04-01
## 3             1               3 1979-07-01

# If reshape.long = TRUE, data are returned in a long format, were 'Statistic'
# serves as an identifier and unaffected columns (here 'Date') are duplicated
head(collap(data, ~ Year + Quarter,"mean,length", reshape.long = TRUE),3)

##   Statistic    GDPC1  CNP16OV GDPDEF FEDFUNDS       Date Year Quarter
## 1      mean 6741.854 163756.3 37.476 10.07333 1979-01-01 1979       1
## 2      mean 6749.063 164447.3 38.394 10.18000 1979-04-01 1979       2
## 3      mean 6799.200 165199.7 39.234 10.94667 1979-07-01 1979       3

# The same holds true for multiple categorical functions, numeric columns are duplicated
head(collap(data, ~ Year + Quarter, catFUN = "Mode,first,last", reshape.long = TRUE),3)

##   Statistic    GDPC1  CNP16OV GDPDEF FEDFUNDS       Date Year Quarter
## 1      Mode 6741.854 163756.3 37.476 10.07333 1979-01-01 1979       1
## 2      Mode 6749.063 164447.3 38.394 10.18000 1979-04-01 1979       2
## 3      Mode 6799.200 165199.7 39.234 10.94667 1979-07-01 1979       3

# If multiple functions are supplied to 'FUN' and 'catFUN', wide data are always returned
head(collap(data, ~ Year + Quarter,"mean,length","Mode,first,last"),3)

##   GDPC1.mean GDPC1.length CNP16OV.mean CNP16OV.length GDPDEF.mean GDPDEF.length FEDFUNDS.mean
## 1   6741.854            1     163756.3              3      37.476             1      10.07333
## 2   6749.063            1     164447.3              3      38.394             1      10.18000
## 3   6799.200            1     165199.7              3      39.234             1      10.94667
##   FEDFUNDS.length  Date.Mode Date.first  Date.last Year Quarter
## 1               3 1979-01-01 1979-01-01 1979-03-01 1979       1
## 2               3 1979-04-01 1979-04-01 1979-06-01 1979       2
## 3               3 1979-07-01 1979-07-01 1979-09-01 1979       3
```

The code below demonstrates the fully custom mode, which STATA users will find familiar from *collapse*. It should be noted that it is not possible to supply a named list of functions to 'custom' the way it can be supplied to 'FUN' or 'catFUN'. Whenever a named list is supplied to 'custom', *collap* will interpret the names as column names and search for them in the dataset.

```
# Fully custom aggregation
head(collap(data, ~ Year + Quarter,
            custom = list("GDPC1,GDPDEF" = mean, FEDFUNDS = function(x)length(unique(x)))),3)

##      GDPC1 GDPDEF FEDFUNDS Year Quarter
## 1 6741.854 37.476        3 1979       1
## 2 6749.063 38.394        3 1979       2
## 3 6799.200 39.234        3 1979       3

# Users should note that when a named list is passed to 'custom', the names will always
# be interpreted as column names matching those in the data
```

```r
# Using quotes around functions adds names, as long as custom.names = TRUE
# The same column can also be assigned to multiple functions (here FEDFUNDS):
head(collap(data, ~ Year + Quarter,
            custom = list("GDPC1,GDPDEF,FEDFUNDS" = "mean", FEDFUNDS = "median")),3)

##   GDPC1.mean GDPDEF.mean FEDFUNDS.mean FEDFUNDS.median Year Quarter
## 1   6741.854      37.476      10.07333           10.07 1979       1
## 2   6749.063      38.394      10.18000           10.24 1979       2
## 3   6799.200      39.234      10.94667           10.94 1979       3

# Alternatively: Using a vector, list or comma-separated string of functions
# of length ncol(X)-length(by)
head(collap(data, ~ Year + Quarter, custom = "mean,median,mean,median,first"),3)

##   GDPC1.mean CNP16OV.median GDPDEF.mean FEDFUNDS.median Date.first Year Quarter
## 1   6741.854         163726      37.476           10.07 1979-01-01 1979       1
## 2   6749.063         164459      38.394           10.24 1979-04-01 1979       2
## 3   6799.200         165198      39.234           10.94 1979-07-01 1979       3

# Without the names
head(collap(data, ~ Year + Quarter, custom = "mean,median,mean,median,first",
            custom.names = FALSE),3)

##      GDPC1 CNP16OV GDPDEF FEDFUNDS       Date Year Quarter
## 1 6741.854  163726 37.476    10.07 1979-01-01 1979       1
## 2 6749.063  164459 38.394    10.24 1979-04-01 1979       2
## 3 6799.200  165198 39.234    10.94 1979-07-01 1979       3

# Using a list of functions of length ncol(X)-length(by)
head(collap(data, ~ Year + Quarter,
            custom = list(mean,median,mean,function(x)length(unique(x)),first)),3)

##      GDPC1 CNP16OV GDPDEF FEDFUNDS       Date Year Quarter
## 1 6741.854  163726 37.476        3 1979-01-01 1979       1
## 2 6749.063  164459 38.394        3 1979-04-01 1979       2
## 3 6799.200  165198 39.234        3 1979-07-01 1979       3
```

Now I provide a taste of uses of *collap* with different data objects. Some of these examples are a bit unconventional, especially since *qsu* is better adapted to compute summary statistics, but they serve to demonstrate the flexibility of *collap*.

```r
# Leaving 'by' unpecified fully aggregates the data
collap(data)

##      GDPC1  CNP16OV   GDPDEF FEDFUNDS       Date     Year  Quarter
## 1 12180.56 209992.2 76.52082 4.890374 1979-01-01 1998.543 2.496881

# Collap works with matrices
round(collap(as.matrix(data[-5])),2) # This outputs a vector

##     GDPC1   CNP16OV    GDPDEF   FEDFUNDS      Year   Quarter
##  12180.56 209992.23     76.52       4.89   1998.54      2.50

head(collap(as.matrix(data[-5]), ~ Year + Quarter),3) # This outputs a matrix

##         GDPC1  CNP16OV GDPDEF FEDFUNDS Year Quarter
## [1,] 6741.854 163756.3 37.476 10.07333 1979       1
## [2,] 6749.063 164447.3 38.394 10.18000 1979       2
## [3,] 6799.200 165199.7 39.234 10.94667 1979       3

# Collap also works with vectors, here same as calling sd(data$GDPC1, na.rm = TRUE)
collap(data$GDPC1, fun = sd) # This gives a scalar

## data.GDPC1
##   12180.56
```

```r
# Using two vectors
collap(data$GDPC1, data$Quarter, "mean,min,max")

##   data.Quarter data.GDPC1.mean data.GDPC1.min data.GDPC1.max
## 1            1        12098.39       6741.854       18323.96
## 2            2        12186.08       6696.753       18511.58
## 3            3        12263.02       6688.794       18664.97
## 4            4        12174.59       6802.497       18223.76

# Using a list. If the list is not named, column names will be "Group.1", "Group.2"
head(collap(data$GDPC1, list(Year = data$Year, Quarter = data$Quarter)),3)

##   Year Quarter data.GDPC1
## 1 1979       1   6741.854
## 2 1979       2   6749.063
## 3 1979       3   6799.200

# This computes the time-correlation for each variable, averaged across the 4 quarters
collap(collap(collap(data,"Year,Quarter"),"Quarter",function(x)cor(x,seq(x))))

##       GDPC1   CNP16OV    GDPDEF   FEDFUNDS       Date Year Quarter
## 1 0.9950782 0.9983412 0.9978126 -0.8705986 1979-01-01    1     2.5
```

As noted before, if a *data.table* is passed to *collap*, *collap* will automatically resort to the fast *data.table* method for aggregation (same as setting data.table = TRUE) and also output a *data.table*. If the user sets data.table = TRUE, and the input is not a *data.table*, *collap* will internally use *data.table* for aggregation, but output an object of the original class. The code below briefly demonstrates the as.list argument, which can come in handy for certain tasks, if output in a list formal is preferred.

```r
# If multiple functions are called, as.list = "FUN" returns a separate dataset for each
str(collap(data,~Year+Quarter,"mean,length", as.list = "FUN"))

## List of 2
##  $ mean  :'data.frame': 161 obs. of  7 variables:
##   ..$ GDPC1   : num [1:161] 6742 6749 6799 6816 6838 ...
##   ..$ CNP16OV : num [1:161] 163756 164447 165200 166055 166762 ...
##   ..$ GDPDEF  : num [1:161] 37.5 38.4 39.2 40 40.8 ...
##   ..$ FEDFUNDS: num [1:161] 10.1 10.2 10.9 13.6 15 ...
##   ..$ Date    : chr [1:161] "1979-01-01" "1979-04-01" "1979-07-01" "1979-10-01" ...
##   ..$ Year    : num [1:161] 1979 1979 1979 1979 1980 ...
##   ..$ Quarter : int [1:161] 1 2 3 4 1 2 3 4 1 2 ...
##  $ length:'data.frame': 161 obs. of  7 variables:
##   ..$ GDPC1   : int [1:161] 1 1 1 1 1 1 1 1 1 1 ...
##   ..$ CNP16OV : int [1:161] 3 3 3 3 3 3 3 3 3 3 ...
##   ..$ GDPDEF  : int [1:161] 1 1 1 1 1 1 1 1 1 1 ...
##   ..$ FEDFUNDS: int [1:161] 3 3 3 3 3 3 3 3 3 3 ...
##   ..$ Date    : chr [1:161] "1979-01-01" "1979-04-01" "1979-07-01" "1979-10-01" ...
##   ..$ Year    : num [1:161] 1979 1979 1979 1979 1980 ...
##   ..$ Quarter : int [1:161] 1 2 3 4 1 2 3 4 1 2 ...

# as.list = "by" provides a list of datasets for each 'by'-group
head(collap(data,~Year+Quarter,"length,mean,sd,min,max", as.list = "by", reshape.long = T),2)

## $`1979.1`
##     Statistic    GDPC1      CNP16OV GDPDEF    FEDFUNDS       Date
## 1      length    1.000       3.0000  1.000  3.00000000 1979-01-01
## 162      mean 6741.854 163756.3333 37.476 10.07333333 1979-01-01
## 323        sd       NA     256.8469     NA  0.01527525 1979-01-01
## 484       min 6741.854 163516.0000 37.476 10.06000000 1979-01-01
## 645       max 6741.854 164027.0000 37.476 10.09000000 1979-01-01
##
## $`1979.2`
##     Statistic    GDPC1      CNP16OV GDPDEF   FEDFUNDS       Date
## 2      length    1.000       3.0000  1.000  3.0000000 1979-04-01
## 163      mean 6749.063 164447.3333 38.394 10.1800000 1979-04-01
## 324        sd       NA     279.6826     NA  0.1493318 1979-04-01
```

```
## 485       min 6749.063 164162.0000 38.394 10.0100000 1979-04-01
## 646       max 6749.063 164721.0000 38.394 10.2900000 1979-04-01
```

1.4 Benchmark

I finally examine the performance of *collap*, in its default mode and with the help of the built-in data.table option, and compare it to *aggregate.data.frame* and *data.table*. I let *aggregate.data.frame* and *data.table* perform exactly the same computational steps as *collap*, although I do not bring the output of these functions in the same form (i.e. no column binding and sorting, and no replacement of NaN's with NA's). The benchmark comes in three steps: A microbenchmark on the dataset considered so far, a benchmark with a long dataset[3], and a benchmark with a wide dataset.

```
library(microbenchmark)
library(data.table)
dim(data)

## [1] 481   7

print(microbenchmark( # 100 replications microbenchmark
# C = Collap | AG = Aggregate | CDT = Collap + Data Table | DT = Data Table
C  = collap(data, ~ Year + Quarter),
AG ={aggregate.data.frame(data[1:4], data[6:7], FUN = mean, na.rm = TRUE)
     aggregate.data.frame(data[5], data[6:7], FUN = Mode, na.rm = TRUE)},
CDT=collap(data, ~ Year + Quarter, data.table = TRUE),
DT ={setDT(data)[,lapply(.SD,mean,na.rm=TRUE), keyby = "Year,Quarter", .SDcols=1:4]
     setDT(data)[,lapply(.SD,Mode,na.rm=TRUE), keyby = "Year,Quarter", .SDcols=5]
     setDF(data)}), digits = 3)

## Unit: milliseconds
##  expr   min    lq  mean median    uq  max neval  cld
##     C 11.48 11.88 12.70  12.13 13.25 17.0   100   c
##    AG 12.83 13.32 13.93  13.55 14.53 16.5   100    d
##   CDT  5.34  5.77  6.19   6.01  6.26 11.3   100  b
##    DT  3.91  4.12  4.64   4.33  4.60 11.9   100 a
```

The microbenchmark shows that in the default mode *collap* performs slightly faster than *aggregate.data.frame*, and is about 2 milliseconds slower than *data.table* in the data.table mode.

```
# Generating long data:
for (i in 1:13) data = rbind(data,data)
dim(data)

## [1] 3940352       7

print(microbenchmark( # 10 replications benchmark
# C = Collap | AG = Aggregate | CDT = Collap + Data Table | DT = Data Table
C  = collap(data, ~ Year + Quarter),
AG ={aggregate.data.frame(data[1:4], data[6:7], FUN = mean, na.rm = TRUE)
    aggregate.data.frame(data[5], data[6:7], FUN = Mode, na.rm = TRUE)},
CDT=collap(data, ~ Year + Quarter, data.table = TRUE),
DT ={setDT(data)[,lapply(.SD,mean,na.rm=TRUE), keyby = "Year,Quarter", .SDcols=1:4]
     setDT(data)[,lapply(.SD,Mode,na.rm=TRUE), keyby = "Year,Quarter", .SDcols=5]
     setDF(data)}, times = 10), digits = 3)

## Unit: milliseconds
##  expr   min    lq  mean median    uq   max neval cld
##     C 13950 14050 14588  14489 15091 15445    10  b
##    AG 14778 14979 15283  15142 15536 16306    10   c
##   CDT   799   801   901    864   982  1119    10 a
##    DT   754   768   814    801   861   931    10 a
```

The benchmark with the long dataset of approx. 4 million observations shows that the built-in data.table option endows *collap* with a significant edge over *aggregate.data.frame* (0.9 seconds vs. 18

[3]Obtained by duplicating and row-binding the dataset at hand.

seconds for this task), and is only neglegibly slower than *data.table* itself. For the wide data benchmark I use the World Bank Development Indicators, a dataset providing around 1450 development indicators following 264 geographical entities grouped into 7 World Regions over 57 years. Below I aggregate this dataset by region and year:

```
# The World Bank Development Indicators
dim(WDI)

## [1] 15048  1457

ind = match(c("region","year"),names(WDI)) # Columns to aggregate by
nu = setdiff(which(sapply(WDI,is.numeric)),ind) # Numeric variables
nnu = setdiff(seq(ncol(WDI)),c(ind,nu)) # categorical variables

print(microbenchmark( # 10 replications benchmark
# C = Collap | AG = Aggregate | CDT = Collap + Data Table | DT = Data Table
C  = collap(WDI, ind),
AG ={aggregate.data.frame(WDI[nu], WDI[ind], FUN = mean, na.rm = TRUE)
     aggregate.data.frame(WDI[nnu], WDI[ind], FUN = Mode, na.rm = TRUE)},
CDT=collap(WDI, ind, data.table = TRUE),
DT ={setDT(WDI)[,lapply(.SD,mean,na.rm=TRUE), keyby = "region,year", .SDcols=nu]
     setDT(WDI)[,lapply(.SD,Mode,na.rm=TRUE), keyby = "region,year", .SDcols=nnu]
     setDF(WDI)}, times = 10), digits = 3)

## Unit: milliseconds
##  expr   min    lq  mean median    uq   max neval cld
##     C 10651 10810 11287  11357 11681 11866    10   b
##    AG 10454 10570 11028  10832 11640 11889    10   b
##   CDT   563   571   594    576   603   724    10  a
##    DT   549   567   592    582   606   695    10  a
```

The results again are vary similar, *collap* here is about the same speed as *aggregate.data.frame* and also just as fast as *data.table* - a blazing 0.6 seconds for this dataset - revealing the efficient programming behind it and rendering it a very useful tool even for advanced R users working on large datasets or data.tables.

Amongst others I have not demonstrated the parallel option. Generally speaking the speed improvement it brings is modest on two-core machines, but when several functions are applied and the dataset is long and large, *collap* with the data.table and parallel options enabled can outperform *data.table*.

1.5 Conclusion

collap represents a new data-aggregation tool that offers a significant combination of extended functionality, performance and convenience that was previously unavailable in R in this area. Based on my own use I am convinced that this command will enhance the workflow and become a personal favourite of many data analysts.

2 Qsu

Qsu, which stands shorthand for *quick-summary*, is an advanced and fast summary command for cross-sectional and multilevel (panel) data. It's key feature is that it not only provides arbitrary summary statistics by group, but also within-and between groups, and also within and between subgroups defined by a group. *qsu* also provides an easy and fast method to obtain within-transformed data, a feature many will find handy. Again below I briefly list the key advanteges of *qsu* over existing functions such as *base::summary, base::by, psych::describe, psych::describeBy, FSA::Summarize, Rmisc::summarySE, doby::summaryBy, pastecs::stat.desc, Hmisc::describe, stats::xtabs, fBasic::basicStats*, the *apply* family etc., then I will briefly outline the syntax of the function and swiftly turn to demonstrate its functionality.

2.1 Key Features

- Parsimonious and speedy default summary (output familiar to STATA users from *summarize*). Users can also request an extended set of statistics including skewness and kurtosis, and specify an arbitrary number of quantiles to be computed
- Multilevel (panel)-data summary (i.e. *overall, between* entities and *within* entities summary) (familar from *xtsummarize* STATA command), the *xt*-option
- Summary by Groups, the *by*-option, can be combined with the *xt*-option for subgroups
- Fully customizable set of summary statistics, works with the *xt*- and *by*-options (i.e. any function or set of functions that takes a data-vector as input and returns a vector of statistics can be used with *qsu*)
- Option to apply a transformation like scaling or log to the numeric columns of a dataset, transformations can be taken overall or by group
- Ability to display variable labels in the summary, i.e. for STATA, SPSS or SAS datasets imported into R using the *haven* package, or downloaded using *WDI* or other API's that supply labels
- Maximum flexibility in input and output specification
- Tidy output in a convenient format
- Option to output the transformed data used to compute the summary

2.2 Syntax of *qsu*

Usage

```
qsu(X, by = NULL, xt = NULL, FUN = NULL, Q = FALSE, Ext = FALSE, trans = NULL, trans.by = FALSE,
    ndigits = 2, na.rm = TRUE, pretty = FALSE, labels = FALSE, factors = "as.categorical",
    combine.by = FALSE, combine.xt = TRUE, within.add.mean = TRUE, show.trans = TRUE,
    data.out = FALSE, data.out.drop = FALSE, xt.data.table = FALSE)
```

By default *qsu* computes the following statistics: N = Number of Observations, D = Number of distinct values, *Mean*, *SD* = Standard Deviation, *Min* = Minimum value, *Max* = Maximum value. The latter four are only computed for numerical variables. If one or multiple grouping variable is supplied to *xt*, by default *qsu* will show classical (overall) statistics, but also compute statistics between and within groups. The most common form of multilevel data is longitudinal data which follows individuals or entities i over time t (but t could just be another grouping variable). Denote $\mathbf{x}_{it}$ the original data, then $\bar{\mathbf{x}}_i$ is the between-transformed data, where the time-mean for each individual was taken, and $\mathbf{x}_{it} - \bar{\mathbf{x}}_i + \bar{\bar{\mathbf{x}}}$ is the within-transformed (demeaned) data (the overall mean $\bar{\bar{\mathbf{x}}}$ is added back to make results comparable). Providing summary statistics of $\bar{\mathbf{x}}_i$ and $\mathbf{x}_{it} - \bar{\mathbf{x}}_i + \bar{\bar{\mathbf{x}}}$ in addition to $\mathbf{x}_{it}$ has the advantage that it uncoveres the structure of the longitudinal data in terms of the number of individuals and the average number of time-periods. Of particular interest in this summary is the standard deviation, which now decomposes overall variability into variability between individual averages, and variability within individuals over time. This variance decomposition, amongst other things, allows one to see which variables are time varying and which time-invariant individual characteristics, and it allows the researcher to gauge what proportion of the variance in model variables would be lost by employing a fixed effects estimator. If a multilevel dataset is characterized by more than two identifiers, i.e. $\mathbf{x}_{jit}$, one can supply, j, i, t, ij, it

or jt to the xt option. One could also supply for example j to the *by* option, and i to the xt option. In that setup within and between transformed statistics over i will be computed separately for each group defined by j. For example if j is a region, i district and t a year, then this would show the variation between districts and over time for each region.

Arguments

X A vector, matrix, data.frame or data.table to summarize (anything that can be coerced to data.frame)

by Groups to summarize by, **either contained in X** and indicated using a one-or two sided formula (two-sided if only certain columns in X are to be aggregated), column indices, a vector of column names, or a string of comma-separated column names, **or externally supplied** in form of a vector, list of vectors or data.frame, with the number of elements/rows matching that of X.

xt Groups to compute statistics overall, between and within. The same flexibility as with the 'by' argument applies. If used together with 'by', a subgroup of 'by' should be used. If a two-sided formula is used together with 'by', it does not matter whether the LHS variables are specified in the 'by', 'xt' or in both arguments.

FUN Custom function(s) to apply to all columns in X apart from columns in the 'by' or 'xt' arguments. Functions must take a vector and return a vector of statistics. A single function can be supplied without quotes. Multiple functions can be supplied as a character vector, string of comma-separated function names, or as a named list of functions. Ad-hoc functions can be supplied. 'FUN' when it is used overrides the default set of statistics and the 'Q' and 'Ext' arguments.

Q Number of quantiles to compute.

Ext Request an Extended set of statistics including the *median*, the *skewness* and the *kurtosis*

trans A transformation function applied to the numeric columns of the data (for example *log*, *scale*, *diff* or growth rates)

trans.by If the 'by' option is used, 'trans' can be applied to groups separately (i.e. one could use it to obtain growth rates for multiple countries in a long country-time $\times$ variables dataset)

ndigits Number of digits to show. If set to NULL, all digits will be shown.

na.rm Internally removes missing values before applying any functions or transformations. It is not required for functions to have a 'na.rm' argument.

pretty Returns result as a character matrix where trailing zeros are eliminated and large numbers are written in standard (as opposed to scientific) notation.

labels Show variable labels next to statistics. If labels = TRUE, X must be a data.frame with variable labels stored as attributes [attr(X$var1,"label")<-"label1"] etc. Alternatively, a character vector of labels of length ncol(X) can be passed to the labels argument.

factors Specifies the treatment of factor variables. Default is treatment as categorical variables. Alternatively factors can be coerced to numerical variables by spcifying "as.numeric", or the factor levels can be extracted and coerced to a numerical variable by specifying "as.numeric.fractor" (internally defined as: as.numeric.factor <- function(x) {as.numeric(levels(x))[x]})

combine.by If the 'by' option is used, combine.by = TRUE gives a compact output instead of a list.

combine.xt If the 'xt' option is used combine.xt = FALSE returns a list with overall, between group and within group statistics.

within.add.mean By default, within-group statistics are computed as $\mathbf{x}_{it} - \bar{\mathbf{x}}_i + \bar{\bar{\mathbf{x}}}$. If within.add.mean = FALSE, The within-transformed dataset is obtained as $\mathbf{x}_{it} - \bar{\mathbf{x}}_i$, which is a more classical within-transformation used i.e. for fixed-effects regression.

data.out Output transformed data used to compute the summary. If the 'xt' option is used, the output will be a named list of three datasets: An overall dataset (= the original dataset if trans = NULL), an aggregated dataset for the between-statistics, and a within-transformed dataset. All datasets come with the original column order, the aggregated dataset is sorted by the 'xt' identifiers, and the within-transformed dataset has the same row-order as the original dataset. In the aggregated dataset categorical variables were aggregated using the mode, while in the within-transformed dataset categorical variables are unaffected/untransformed.

data.out.drop Drop all identifiers supplied to 'by' or 'xt' before returning the dataset.

xt.data.table If the 'xt' option is used, *qsu* internally utilizes *collap* to aggregate the data and compute the within-transformed dataset. If xt.data.table = TRUE, *collap* will internally use *data.table*, yielding a much faster computation on large datasets.

2.3 Demonstration

To demonstrate *qsu*, I take a classic example of multilevel data, and download 3 series from the World Bank Development Indicators database: The GDP per capita in constant 2010 US$, the life expectancy at birth in years and the GINI index. Following the newest update the *WDI* package also downloads the labels for these series and stores them in a similar way to the *haven* library when importing STATA, SPSS or SAS files that typically contain labels.

```
# Case Study: Income, Health and Inequality
library(WDI)
data = WDI(indicator = c('NY.GDP.PCAP.KD','SP.DYN.LE00.IN','SI.POV.GINI'), extra = TRUE)
data = data[c(2,8,12,3:6)]; names(data)[5:7] = c("PCGDP","LIFEEX","GINI")
str(data)

## 'data.frame': 15576 obs. of  7 variables:
##  $ country: chr  "Arab World" "Arab World" "Arab World" "Arab World" ...
##  $ region : Factor w/ 8 levels "Aggregates","East Asia & Pacific",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ income : Factor w/ 5 levels "Aggregates","High income",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ year   : int  2008 2012 2010 2011 1974 1975 2009 1977 1978 1979 ...
##  $ PCGDP  : atomic  5900 6248 5918 5991 NA ...
##   ..- attr(*, "label")= chr "GDP per capita (constant 2010 US$)"
##  $ LIFEEX : atomic  69.6 70.4 70 70.2 54.8 ...
##   ..- attr(*, "label")= chr "Life expectancy at birth, total (years)"
##  $ GINI   : atomic  NA NA NA NA NA NA NA NA NA NA ...
##   ..- attr(*, "label")= chr "GINI index (World Bank estimate)"
```

In the default mode, *qsu* provides a simple set of summary statistics in an easily readable format. 'D' denotes the number of distinct values, showing that the dataset tracks 264 countries and regional entities over 59 years, 1960-2018. The data-coverage on the GINI index is very low. Down below the 'pretty' argument is set to eliminate trailing zeros and replaces NA's with '-', the labels argument can be used to display variable labels if provided, and specifying factors = as.numeric coerces factor variables (here region and income) to numeric before summarizing them.

```
# Default summary
qsu(data)

##              N     D     Mean       SD     Min       Max
## country 15576   264       NA       NA      NA        NA
## region  15399     8       NA       NA      NA        NA
## income  15399     5       NA       NA      NA        NA
## year    15576    59  1989.00    17.03 1960.00   2018.00
## PCGDP   11358 11248 10632.06 17053.84  131.65 191586.64
## LIFEEX  13747 12555    63.54    11.16   18.91     85.42
## GINI     1356   363    39.40     9.68   16.20     65.80

# Display labels, pretty printing and treat factors as numeric
qsu(data, labels = TRUE, pretty = TRUE, factors = as.numeric)

##              N     D     Mean       SD     Min       Max                                   Label
## country 15576   264        -        -       -         -                                    <NA>
## region  15399     8     3.92     2.38       1         8                                    <NA>
```

```
## income  15399     5     2.96     1.43      1         5                                 <NA>
## year    15576    59     1989    17.03    1960      2018                                 <NA>
## PCGDP   11358 11248 10632.06 17053.84  131.65 191586.64      GDP per capita (constant 2010 US$)
## LIFEEX  13747 12555    63.54    11.16   18.91     85.42 Life expectancy at birth, total (years)
## GINI     1356   363     39.4     9.68    16.2      65.8       GINI index (World Bank estimate)
```

The quantile argument 'Q' takes away 'Min' and 'Max' from the summary and shows the specified number of quantiles. If an extended set of statistics is requested by setting Ext = TRUE, the median, skewness and kurtosis are added to the summary. These statistics are internally defined and need not be loaded from the *moments* library. Of course 'Q' and 'Ext' can be used jointly as the third example shows.

```
# Compute 4 quantiles
qsu(data, Q = 4, pretty = TRUE, factors = as.numeric)

##             N     D     Mean       SD      0%     25%     50%      75%      100%
## country 15576   264        -        -       -       -       -        -         -
## region  15399     8     3.92     2.38       1       2       3        5         8
## income  15399     5     2.96     1.43       1       2       3        4         5
## year    15576    59     1989    17.03    1960    1974    1989     2004      2018
## PCGDP   11358 11248 10632.06 17053.84  131.65 1190.32 3455.63 12217.94 191586.64
## LIFEEX  13747 12555    63.54    11.16   18.91   55.51   66.32       72     85.42
## GINI     1356   363     39.4     9.68    16.2    31.7    37.4     46.8      65.8

# An extended set of statistics
qsu(data, Ext = TRUE, pretty = TRUE, factors = as.numeric)

##             N     D     Mean  Median       SD     Min       Max  Skew  Kurt
## country 15576   264        -       -        -       -         -     -     -
## region  15399     8     3.92       3     2.38       1         8  0.61  2.14
## income  15399     5     2.96       3     1.43       1         5  0.16  1.63
## year    15576    59     1989    1989    17.03    1960      2018     0   1.8
## PCGDP   11358 11248 10632.06 3455.63 17053.84  131.65 191586.64  3.26  18.8
## LIFEEX  13747 12555    63.54   66.32    11.16   18.91     85.42 -0.61  2.58
## GINI     1356   363     39.4    37.4     9.68    16.2      65.8  0.46  2.29

# A very rich summary, adjusting the number of digits
qsu(data, Q = 8, Ext = TRUE, pretty = TRUE, factors = as.numeric, ndigits = 0)

##             N     D  Mean    SD   0% 12.5%  25% 37.5%  50% 62.5%   75% 87.5%   100% Skew Kurt
## country 15576   264     -     -    -     -    -     -    -     -     -     -      -    -    -
## region  15399     8     4     2    1     1    2     3    3     4     5     8      8    1    2
## income  15399     5     3     1    1     1    2     2    3     4     4     5      5    0    2
## year    15576    59  1989    17 1960  1967 1974  1982 1989  1996  2004  2011   2018    0    2
## PCGDP   11358 11248 10632 17054  132   603 1190  2017 3456  5948 12218 27913 191587    3   19
## LIFEEX  13747 12555    64    11   19    49   56    61   66    70    72    75     85   -1    3
## GINI     1356   363    39    10   16    28   32    34   37    42    47    53     66    0    2
```

The functionality offered by the 'by' argument is pretty standard, apart from the greater range of possible input formats that can be supplied, just as for *collap*[4]. The 'combine.by' argument provides a handy extension to obtain the output in a more convenient format.

```
# remove aggregate political entities, '%>%' is from the dplyr package
data = subset(data, region!="Aggregates") %>% droplevels
# The by argument
qsu(data, PCGDP + LIFEEX + GINI ~ income)

## income: High income
##           N    D     Mean       SD    Min       Max
## PCGDP  3038 3038 28974.73 22910.72 944.29 191586.64
## LIFEEX 3682 3459    73.22     5.51  42.67     85.42
## GINI    478  205    34.32     7.86  21.00     58.90
## ---------------------------------------------------------------------------
## income: Low income
```

[4]I already demonstrated the flexibility in inputs with *collap* and won't repear this demonstration here.

```
##           N    D   Mean     SD    Min     Max
## PCGDP  1405 1405 596.80 308.21 131.65 1506.30
## LIFEEX 1881 1819  49.62   8.89  27.61   74.43
## GINI    109   89  41.47   6.79  28.90   65.80
## ---------------------------------------------------------------------------
## income: Lower middle income
##           N    D    Mean     SD    Min     Max
## PCGDP  2120 2120 1583.37 890.74 150.22 4662.88
## LIFEEX 2628 2542   58.56   9.39  18.91   76.25
## GINI    330  209   40.07   9.36  24.00   63.20
## ---------------------------------------------------------------------------
## income: Upper middle income
##           N    D    Mean      SD    Min      Max
## PCGDP  2432 2432 4849.75 2959.23 131.96 20333.94
## LIFEEX 2877 2742   65.97    7.65  36.74    79.83
## GINI    439  257   43.91    9.75  16.20    64.80

# by + combine.by
head(qsu(data, PCGDP + LIFEEX + GINI ~ region, combine.by = TRUE))

##                                 N    D     Mean       SD    Min       Max
## East Asia & Pacific.PCGDP    1391 1391 10337.05 14094.83 131.96  72183.30
## East Asia & Pacific.LIFEEX   1717 1683    65.65    10.12  18.91     84.28
## East Asia & Pacific.GINI       92   74    38.51     5.37  27.80     55.40
## Europe & Central Asia.PCGDP  2084 2084 25664.81 26181.67 367.05 191586.64
## Europe & Central Asia.LIFEEX 2886 2749    71.93     5.46  45.37     85.42
## Europe & Central Asia.GINI    588  177    31.90     4.74  16.20     48.40
```

One feature of *qsu* is that it always seeks to provide output in a convenient format, for example if a single function is used to summarize multiple variables by some group, the output comes in a matrix format similar to the output *collap* offers. If multiple functions are provided, the statistics form the columns and the variables and groups are interacted in the row-names, as in the example above. In that case a wide-format can only be obtained by employing *collap* itself. If multiple groups are used together with 'combine.by' they are also interacted to provide output a long format.

```
# Checking the data availability by country
head(qsu(data, PCGDP + LIFEEX + GINI ~ country, FUN = length, combine.by = TRUE),3)

##             PCGDP LIFEEX GINI
## Afghanistan    16     57    0
## Albania        38     57    5
## Algeria        58     57    3

# An extention of this format could be achieved with collap
head(collap(data, PCGDP + LIFEEX + GINI ~ country, FUN = "length,mean", sort = FALSE),3)

##       country PCGDP.length LIFEEX.length GINI.length PCGDP.mean LIFEEX.mean GINI.mean
## 1 Afghanistan           16            57           0   482.1631    47.88216        NA
## 2     Albania           38            57           5  2710.1591    71.34056  29.66000
## 3     Algeria           58            57           3  3474.3201    62.72084  34.36667

# Inequality by region and income level
head(qsu(data, GINI ~ region + income, FUN = "length,mean,sd", combine.by = TRUE))

##                                           length  mean   sd
## East Asia & Pacific.High income               13 32.80 1.22
## East Asia & Pacific.Lower middle income       37 36.21 4.83
## East Asia & Pacific.Upper middle income       42 42.30 3.64
## Europe & Central Asia.High income            343 30.83 3.66
## Europe & Central Asia.Low income               6 32.13 1.71
## Europe & Central Asia.Lower middle income     93 32.70 5.32

# Only by income level, this time the output is a vector
head(qsu(data, GINI ~ income, FUN = mean, combine.by = TRUE))

##         High income          Low income Lower middle income Upper middle income
##               34.32               41.47               40.07               43.91
```

Of course *qsu* also works with other summary commands, such as *base::summary* or the *quantile* function.

```
# Using base::summary
head(qsu(data, PCGDP + LIFEEX + GINI ~ region, FUN = summary, combine.by = TRUE))

##                                 Min. 1st Qu.   Median     Mean  3rd Qu.      Max.
## East Asia & Pacific.PCGDP    131.96 1600.61  2928.13 10337.05 14574.99  72183.30
## East Asia & Pacific.LIFEEX    18.91   59.89    66.92    65.65    72.59     84.28
## East Asia & Pacific.GINI      27.80   34.60    37.55    38.51    42.25     55.40
## Europe & Central Asia.PCGDP  367.05 5668.88 18993.11 25664.81 36343.96 191586.64
## Europe & Central Asia.LIFEEX  45.37   69.01    71.59    71.93    75.70     85.42
## Europe & Central Asia.GINI    16.20   28.10    31.65    31.90    35.23     48.40
```

The biggest innovation of *qsu* is of course the 'xt' argument, which leads *qsu* to output three sets of statistics for each variable: The standard overall sample statistics, the between-country statistics and the within-country statistics. For the within-summary not the number of observations, but the average number of time-periods $T = N_{\text{overall}}/N_{\text{between}}$ per individual entity is shown. The three identifiers region, income and year form a balanced panel, each tracking 216 entities over 59 years. If the panel is balanced, then the overall, between and within-entitiy means are equal, while if the panel is unbalanced only the overall and within entities means are equal[5]. The standard deviations show that region and income are time-invariant and year is country-invariant. The 'Trans' column can in the summary can be removed by calling show.trans = FALSE. The summary of the three variables below shows that for GDP per capita and life expectancy we have data on around 205 countries with on average around 50 years of data, while the GINI index is only recorded in 161 countries with 8 years of data on average. These variables are not balanced yielding a between-mean slightly different from the overall mean. The standard deviations show that all three variables elicit a significantly larger amount of variation between countries than within-countries/over time.

```
# The xt argument, here showing only the identifiers
head(qsu(data, xt = ~ country, factors = as.numeric),9)

##             Trans   N/T  D    Mean    SD     Min     Max
## region    overall 12744  7    3.52  2.17    1.00    7.00
## region.B  between   216  7    3.52  2.17    1.00    7.00
## region.W   within    59  1    3.52  0.00    3.52    3.52
## income    overall 12744  4    2.37  1.22    1.00    4.00
## income.B  between   216  4    2.37  1.22    1.00    4.00
## income.W   within    59  1    2.37  0.00    2.37    2.37
## year      overall 12744 59 1989.00 17.03 1960.00 2018.00
## year.B    between   216  1 1989.00  0.00 1989.00 1989.00
## year.W     within    59 59 1989.00 17.03 1960.00 2018.00

# A more compact view, showing the three variables
qsu(data, xt = PCGDP + LIFEEX + GINI ~ country, show.trans = FALSE, pretty = TRUE, ndigits = 1)

##            N/T     D    Mean      SD      Min      Max
## PCGDP     8995  8995 11563.7 18348.4    131.6 191586.6
## PCGDP.B    203   203 12488.9 19628.4    255.4 141165.1
## PCGDP.W     44  8993 11563.7    6335 -30529.1  75348.1
## LIFEEX   11068 10048    63.8    11.4     18.9     85.4
## LIFEEX.B   207   207    64.5      10     39.3     85.4
## LIFEEX.W    53 10996    63.8     5.8     33.5     83.9
## GINI      1356   363    39.4     9.7     16.2     65.8
## GINI.B     161   160    39.6     8.4     23.4     61.7
## GINI.W       8  1130    39.4       3       24     54.8
```

Analogous to the 'by' argument, the 'xt' argument also has an associated 'combine.xt' argument which is TRUE by default in order to yield this compact format. If combine.xt = FALSE, *qsu* will output a list with separate overall, between and within statistics.

[5]This is so by definition since the overall mean is added back to the within-transformed data. If within.add.mean = FALSE, the within mean will be 0 for all variables.

```r
# xt without combine.xt: Here showing only the within-country summary
qsu(data, xt = PCGDP + LIFEEX + GINI ~ country, combine.xt = FALSE)$within

##            T     D     Mean      SD       Min      Max
## PCGDP  44.31  8993 11563.65 6334.95 -30529.09 75348.07
## LIFEEX 53.47 10996    63.84    5.83     33.47    83.86
## GINI    8.42  1130    39.40    3.04     23.96    54.80
```

If only a single function is supplied, *qsu* again gives the output in a more convenient format, allowing us to compare the variation of the three variables between countries and over time directly. Similarly to *collap*, if the function name is provided in quotes, it is interacted with the column names. Now one problem in comparing the variability of GDP per capita, life expectancy and inequality of different countries is that these variables come at different scales. The 'trans' argument can therefore be used to scale the data, which will set the overall standard deviations of all variables to 1. It is now evident that the greatest variation between countries is in terms of GDP per capita, while the greatest development within countries was in terms of life expectancy. Overall, the GINI coefficient shows the lowest amount of variation between and within countries.

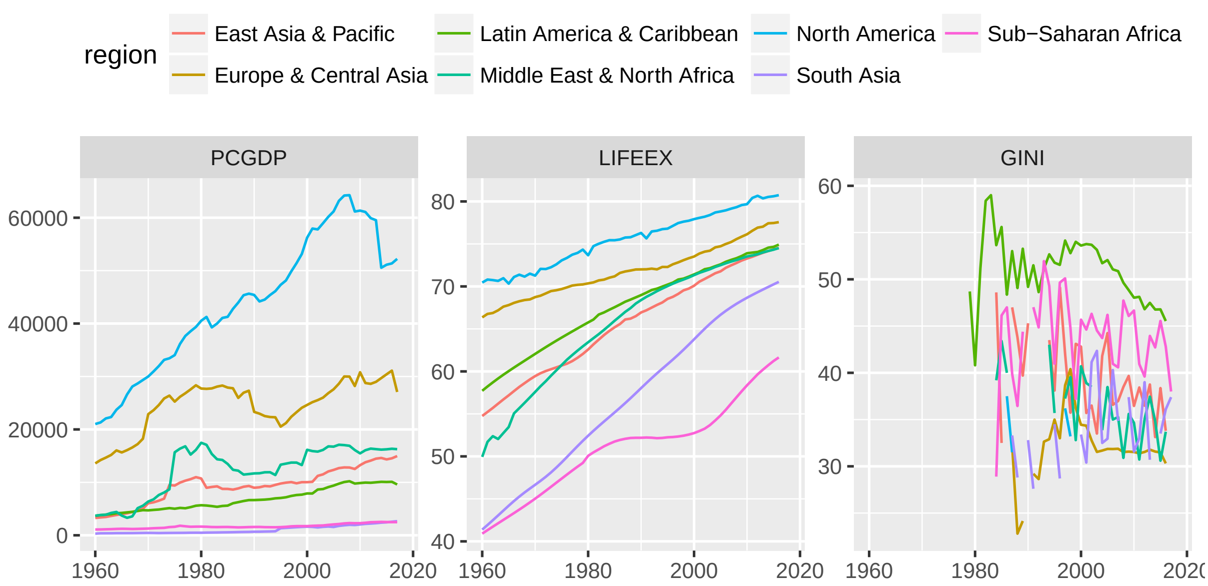
```r
# Only examining the SD
qsu(data, xt = PCGDP + LIFEEX + GINI ~ country, FUN = "sd")

##        overall.sd between.sd within.sd
## PCGDP    18348.41   19628.37   6334.95
## LIFEEX      11.45      10.02      5.83
## GINI         9.68       8.37      3.04

# Putting this on a standardized scale
qsu(data, xt = PCGDP + LIFEEX + GINI ~ country, FUN = "sd", trans = scale)

##        overall.sd between.sd within.sd
## PCGDP           1       1.07      0.35
## LIFEEX          1       0.88      0.51
## GINI            1       0.86      0.31
```

Using now also the 'by' argument, the variations of the three variables can be explored for the 7 World Regions individually[6]. The statistics show that the greatest within-country changes in GDP per capita were in North America, the greatest changes in life expectancy were in South Asia, and the greatest changes in inequality were in Africa. The relative variation between and within countries for each region can be examined through setting trans.by = TRUE, which will apply the scaling to each region separately.

```r
# Decomposing variation in inequality between and within countries by region
qsu(data, PCGDP + LIFEEX + GINI ~ region, ~ country, FUN = "sd", combine.by = TRUE, trans = scale)

##                                    overall.sd between.sd within.sd
## East Asia & Pacific.PCGDP                0.77       0.67      0.35
## East Asia & Pacific.LIFEEX               0.88       0.67      0.57
## East Asia & Pacific.GINI                 0.55       0.48      0.24
## Europe & Central Asia.PCGDP              1.43       1.54      0.57
## Europe & Central Asia.LIFEEX             0.48       0.41      0.30
## Europe & Central Asia.GINI               0.49       0.43      0.25
## Latin America & Caribbean .PCGDP         0.37       0.42      0.13
## Latin America & Caribbean .LIFEEX        0.64       0.49      0.46
## Latin America & Caribbean .GINI          0.55       0.53      0.36
## Middle East & North Africa.PCGDP         1.00       1.08      0.34
## Middle East & North Africa.LIFEEX        0.83       0.53      0.66
## Middle East & North Africa.GINI          0.53       0.48      0.23
## North America.PCGDP                      1.00       0.81      0.75
## North America.LIFEEX                     0.30       0.15      0.28
## North America.GINI                       0.41       0.51      0.17
## South Asia.PCGDP                         0.09       0.11      0.03
## South Asia.LIFEEX                        0.97       0.54      0.83
## South Asia.GINI                          0.45       0.37      0.27
## Sub-Saharan Africa .PCGDP                0.14       0.12      0.07
## Sub-Saharan Africa .LIFEEX               0.75       0.54      0.54
## Sub-Saharan Africa .GINI                 0.86       0.74      0.46
```

[6] It does not matter here whether the three variables are indicated on the LHS of the formulas passed to 'by' or to 'xt'.

```r
# Scaling by region
qsu(data, PCGDP + LIFEEX + GINI ~ region, ~ country, FUN = "sd", combine.by = TRUE,
    trans = scale, trans.by = TRUE)

##                                      overall.sd between.sd within.sd
## East Asia & Pacific.PCGDP                     1       0.87      0.45
## East Asia & Pacific.LIFEEX                    1       0.76      0.65
## East Asia & Pacific.GINI                      1       0.87      0.43
## Europe & Central Asia.PCGDP                   1       1.08      0.40
## Europe & Central Asia.LIFEEX                  1       0.86      0.63
## Europe & Central Asia.GINI                    1       0.88      0.52
## Latin America & Caribbean .PCGDP              1       1.15      0.34
## Latin America & Caribbean .LIFEEX             1       0.76      0.73
## Latin America & Caribbean .GINI               1       0.97      0.66
## Middle East & North Africa.PCGDP              1       1.08      0.34
## Middle East & North Africa.LIFEEX             1       0.63      0.79
## Middle East & North Africa.GINI               1       0.90      0.43
## North America.PCGDP                           1       0.81      0.75
## North America.LIFEEX                          1       0.48      0.93
## North America.GINI                            1       1.26      0.42
## South Asia.PCGDP                              1       1.33      0.37
## South Asia.LIFEEX                             1       0.56      0.85
## South Asia.GINI                               1       0.82      0.61
## Sub-Saharan Africa .PCGDP                     1       0.85      0.51
## Sub-Saharan Africa .LIFEEX                    1       0.73      0.72
## Sub-Saharan Africa .GINI                      1       0.85      0.53

# A rough visual demonstration of what we are looking at
library(reshape2); library(ggplot2)
D = collap(data, ~ region + year, dropcat = TRUE) %>% melt(1:2)
qplot(year, value, color = region, geom = "line", data = D) + labs(x=NULL,y=NULL) +
      facet_wrap(~variable, scales = "free") + theme(legend.position = "top")
```



Below the variation in inequality is decomposed by income group. The analysis clearly shows that by far the greatest within-country variation in inequality is in low income countries, while the greatest between country variation is in upper middle income countries.

```r
# Decomposing variation in inequality by income group
qsu(data, GINI ~ income, ~ country, FUN = "sd", combine.by = TRUE)

##                        overall.sd between.sd within.sd
## High income                  7.86       6.86      1.94
## Low income                   6.79       5.16      4.69
```

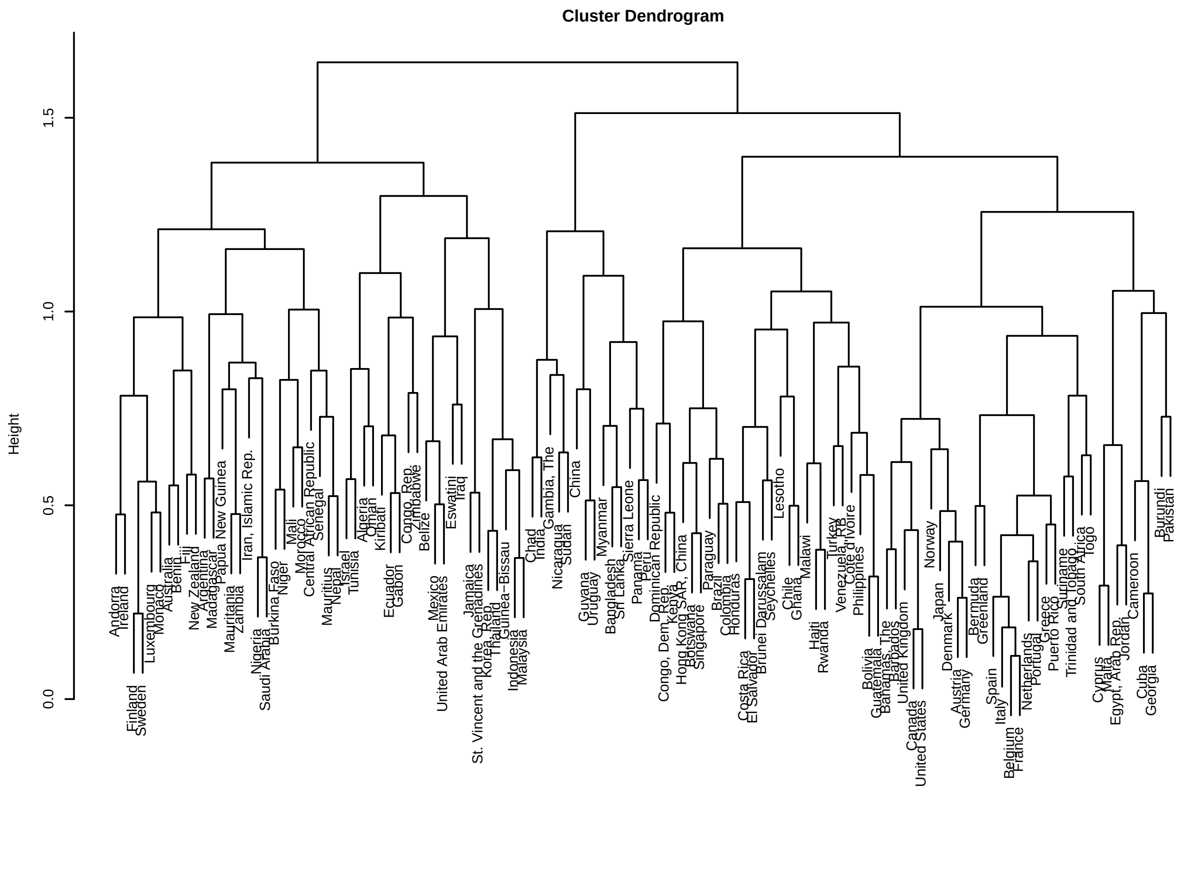
```
## Lower middle income       9.36       7.60      3.53
## Upper middle income       9.75       9.66      3.12

# Putting this on a standardized scale
qsu(data, GINI ~ income, ~ country, FUN = "sd", combine.by = TRUE,
    trans = scale, trans.by = TRUE)

##                       overall.sd between.sd within.sd
## High income                    1       0.87      0.25
## Low income                     1       0.76      0.69
## Lower middle income            1       0.81      0.38
## Upper middle income            1       0.99      0.32
```

As a final step in this part of the analysis, the long-term correlations between the three variables are examined. For this the data is aggregated to decadal averages using *collap*, and then *qsu* is used to obtain aggregated and within country transformed versions of this dataset. The overall, between country and within country correlations of the three variables are then easily computed. The correlations show that overall and between countries inequality is negatively correlated with income and life expectancy, while within countries there is a zero relationship between income and inequality. A stylized fact that emerged in the economics literature is that the between-country correlation of growth and inequality is negative while the within-country relationship is positive. More recent empirical work however also shows that this relationship is highly non-linear. A general pattern in this data is that the between-country correlations are greater than the within-country correlations - a major point of critique for cross-country analysis.

```
# Reduce dataset to 10-Year averages
dataD = collap(data[5:7], data.frame(data[1:3], decade = round(data$year/10)*10))
# Obtain between and within transformed data:
datBW = qsu(dataD, xt = ~ country, within.add.mean = FALSE, data.out = TRUE)
lapply(datBW, head, 5)

## $overall
##       country     region     income decade    PCGDP   LIFEEX GINI
## 1 Afghanistan South Asia Low income   1960       NA 33.39967   NA
## 2 Afghanistan South Asia Low income   1970       NA 36.70089   NA
## 3 Afghanistan South Asia Low income   1980       NA 42.03909   NA
## 4 Afghanistan South Asia Low income   1990       NA 49.69089   NA
## 5 Afghanistan South Asia Low income   2000 349.7596 55.61818   NA
##
## $between
##          country                     region              income decade      PCGDP   LIFEEX     GINI
## 1    Afghanistan                 South Asia          Low income   1990   480.3213 48.86557       NA
## 2        Albania      Europe & Central Asia Upper middle income   1990  2951.0325 71.73397 29.63333
## 3        Algeria Middle East & North Africa Upper middle income   1990  3528.0841 63.35445 34.36667
## 4 American Samoa        East Asia & Pacific Upper middle income   1990 10125.6670       NA       NA
## 5        Andorra      Europe & Central Asia         High income   1990 40598.7349       NA       NA
##
## $within
##       country     region     income decade     PCGDP      LIFEEX GINI
## 1 Afghanistan South Asia Low income    -30        NA -15.4659040   NA
## 2 Afghanistan South Asia Low income    -20        NA -12.1646818   NA
## 3 Afghanistan South Asia Low income    -10        NA  -6.8264798   NA
## 4 Afghanistan South Asia Low income      0        NA   0.8253182   NA
## 5 Afghanistan South Asia Low income     10 -130.5616   6.7526111   NA

# Compute long-term correlations
data.frame(lapply(datBW,function(x)round(cor(x[5:7],use = "pairwise.complete.obs"),2)))

##        overall.PCGDP overall.LIFEEX overall.GINI between.PCGDP between.LIFEEX between.GINI
## PCGDP           1.00           0.57        -0.39          1.00           0.60        -0.41
## LIFEEX          0.57           1.00        -0.40          0.60           1.00        -0.43
## GINI           -0.39          -0.40         1.00         -0.41          -0.43         1.00
##        within.PCGDP within.LIFEEX within.GINI
## PCGDP          1.00          0.33        0.00
## LIFEEX         0.33          1.00       -0.14
## GINI           0.00         -0.14        1.00
```

As a last part of the demonstration I show below that *qsu* can also be used for certain data wrangling tasks, such as computing growth rates of one or multiple variables in multilevel datasets or obtaining a matrix of values from a column in a multilevel dataset. I conceed that a function like *plyr* may be just as adept to this task, but the example is neat: Below I hierarchically cluster economies based on the correlatiof their GDP growth rates, and then use the average R^2 of a countries growth with all other countries to find the 20 most and the 20 least internationally integrated economies based on this metric.

```
data = data[order(data$country,data$year),]
# Compute growth rate for each country
data$PCGR = qsu(data, PCGDP ~ country, trans = function(x)(x-dplyr::lag(x))/dplyr::lag(x)*100,
                combine.by = TRUE, trans.by = TRUE, data.out = TRUE, data.out.drop = TRUE)
# This creates a matrix of growth rates:
GRmat = t(qsu(data, PCGR ~ country, FUN = function(x)x, na.rm = FALSE, combine.by = TRUE))
rownames(GRmat) = unique(data$year)
# Keep countries with more than 40 years of data
keep = apply(GRmat,2,function(x)sum(!is.na(x)))>40
GRmat = GRmat[,keep]; dim(GRmat)

## [1]  59 120

# Preview:
GRmat[1:5,1:5]

##      Algeria Andorra Argentina Australia Austria
## 1960      NA      NA        NA        NA      NA
## 1961  -15.73      NA      3.75      0.47    4.96
## 1962  -21.65      NA     -2.41     -1.15    2.02
## 1963   31.01      NA     -6.77      4.20    3.47
## 1964    3.16      NA      8.46      4.90    5.42

# Compute pairwise correlations between country growth rates
GRcormat = cor(GRmat, use = "pairwise.complete.obs")
# Use this as a distance matrix for hierarchical clustering ->
# Uncover the structure of the World Economy based on growth rates
par(mar = c(0.5,4,2,0.1), cex = 0.5)
plot(hclust(as.dist(1-GRcormat), method="complete"))
```



```r
# 20 most internationally integrated economies based on R^2 of growth rates
head(sort(apply(GRcormat^2,2,mean), decreasing = TRUE),20)

##         France        Belgium          Italy        Austria         Cyprus    Netherlands
##     0.12840163     0.12260422     0.11513234     0.10695267     0.10496995     0.10074666
##        Germany       Portugal          Spain       Barbados         Canada        Finland
##     0.09767770     0.09671805     0.09385140     0.09327988     0.09182866     0.08757321
##      Guatemala          Japan         Greece        Denmark United Kingdom  United States
##     0.08565475     0.08499938     0.08381706     0.08365582     0.08122768     0.08065376
##    Puerto Rico         Sweden
##     0.07972002     0.07790377

# 20 least internationally integrated economies based on R^2 of growth rates
head(sort(apply(GRcormat^2,2,mean), decreasing = FALSE),20)

## Central African Republic             Gambia, The                    Iraq                Zimbabwe
##               0.02050957              0.02280265              0.02380236              0.02439412
##                   Malawi            Sierra Leone                  Rwanda                    Chad
##               0.02461948              0.02554266              0.02566427              0.02795084
##                  Morocco                   Benin                Pakistan                 Burundi
##               0.02802228              0.02850689              0.02895090              0.02925478
##         Papua New Guinea                   Niger                Cameroon                 Algeria
##               0.02975633              0.02987594              0.03006668              0.03028474
##            Guinea-Bissau      Dominican Republic                Eswatini                    Mali
##               0.03030095              0.03129271              0.03132721              0.03144876
```

2.4 A Note on Performance

I do not show official benchmarks results for *qsu* since for most of it's functionality there is no function to directly compare it to. I have however tested it on the WDI dataset used in the *collap* benchmark and found the following: In the default mode calling *qsu* on the WDI dataset takes about 0.5 seconds, whereas using *base::summary* takes about 1.1 seconds. For the xt method and with xt.data.table = TRUE, *qsu* takes about 2.6 seconds to provide a complete overall, between and within country summary of the WDI dataset. The aggregate method takes longer at around 10 seconds. If only the data is requested with data.out = TRUE, within.add.mean = FALSE and xt.data.table = TRUE, *qsu* takes only about 1.4 seconds to output the aggregated and within-transformed datasets. Given that *data.table* itself takes about 0.6 seconds just to aggregate this dataset by country, 1.4 seconds for a within transformed dataset of this size is very fast.

2.5 Conclusion

qsu is an advanced summary command for cross-sectional and multilevel (panel) data, which also offers a significant edge over existing summary functions in terms of functionality, flexibility of use and performance. The seamless integration of the 'by', 'xt', 'FUN' and 'trans' arguments, together with its intelligent reshaping of outputs into a parsimonious format and the possibility to quickly compute and output transformed data, will make it, together with *collap*, a prefered tool, at the very least for everyone frequently working with multilevel data in R.