Package 'grattan'

February 22, 2018

Type Package

Title Australian Tax Policy Analysis

Version 1.5.3.7

Date 2018-02-22

Maintainer Hugh Parsonage <hugh.parsonage@gmail.com>

URL https://github.com/HughParsonage/grattan

BugReports https://github.com/HughParsonage/grattan/issues

Description Utilities for costing and evaluating Australian tax policy, including high-performance tax and transfer calculators,
a fast method of projecting tax collections, and an interface to common indices from the Australian Bureau of Statistics.
Written to support Grattan Institute's Australian Perspectives program. For access to the 'taxstats' package, please run
install.packages(``taxstats'', repos = ``https://hughparsonage.github.io/drat/'', type = ``source'').
N.B. The 'taxstats' package is approximately 50 MB.

Depends R (>= 3.3.0)

License GPL-2

Imports data.table, hutils, rsdmx, fastmatch, forecast, assertthat, magrittr, Rcpp (>= 0.12.3), utils, zoo

LinkingTo Rcpp

RoxygenNote 6.0.1

Suggests dplyr, dtplyr, ggplot2, ggrepel, knitr, lattice, mgcv, rmarkdown, scales, survey, testthat, taxstats, viridis, yaml

Additional_repositories https://hughparsonage.github.io/drat/

LazyData true

VignetteBuilder knitr

NeedsCompilation yes

Author Hugh Parsonage [aut, cre],
Tim Cameron [aut],
Brendan Coates [aut],

William Young [aut],
Ittima Cherastidtham [dtc],
W. Karsten [ctb],
M. Enrique Garcia [ctb]

Repository CRAN

Date/Publication 2018-02-22 12:13:28 UTC

R topics documented:

`grattan-package` *The grattan package.*

Description

Grattan package

Details

Tax modelling and other common tasks for Australian policy analysts, in support of the Grattan Institute, Melbourne. <https://grattan.edu.au>

Author(s)

<hugh.parsonage+grattanpackage@grattan.edu.au>

<hugh.parsonage@gmail.com>

See Also

Useful links:

- https://github.com/HughParsonage/grattan
- Report bugs at https://github.com/HughParsonage/grattan/issues

age_grouper *Age grouper*

Description

Age grouper

Usage

```
age_grouper(age, interval = 10, min_age = 25, max_age = 75,
  breaks = NULL, labels = NULL)
```

Arguments

age	A numeric age (in years).
interval	How big should the age range be. 25-34 means interval = 10.
min_age	What is the upper bound of the lowest bracket? (`min_age = 25` means 'Under 25' will be the lowest bracket.)
max_age	What is the lower bound of the highest bracket? (`max_age = 75` means '75+' will be the bracket.)
breaks	Specify breaks manually.
labels	Specify the labels manually.

Value

An ordered factor giving age ranges (separated by hyphens) as specified.

Examples

```
age_grouper(42)
age_grouper(42, interval = 5, min_age = 20, max_age = 60)
```

apply_super_caps_and_div293 *Superannuation caps and Division 293 calculations*

Description

Mutate a sample file to reflect particular caps on concessional contributions and applications of Division 293 tax.

Usage

```
apply_super_caps_and_div293(.sample.file,
  colname_concessional = "concessional_contributions",
  colname_div293_tax = "div293_tax",
  colname_new_Taxable_Income = "Taxable_income_for_ECT",
  div293_threshold = 3e+05, cap = 25000, cap2 = 35000,
  age_based_cap = TRUE, cap2_age = 59, ecc = FALSE,
  use_other_contr = FALSE, scale_contr_match_ato = FALSE, .lambda = 0,
  reweight_late_lodgers = FALSE, .mu = 1.05,
  impute_zero_concess_contr = FALSE, .min.Sw.for.SG = 450 * 12,
  .SG_rate = 0.0925, warn_if_colnames_overwritten = TRUE,
  drop_helpers = FALSE, copyDT = TRUE)
```

Arguments

`.sample.file`	A data.table containing at least the variables `sample_file_1314` from the taxstats package.
`colname_concessional`	The name for concessional contributions.
`colname_div293_tax`	The name of the column containing the values of Division 293 tax payable for that taxpayer.
`colname_new_Taxable_Income`	The name of the column containing the new Taxable Income.
`div293_threshold`	The Division 293 threshold.
`cap`	The cap on concessional contributions for all taxpayers if `age_based_cap` is FALSE, or for those below the age threshold otherwise.
`cap2`	The cap on concessional contributions for those above the age threshold. No effect if `age_based_cap` is FALSE.
`age_based_cap`	Is the cap on concessional contributions age-based?
`cap2_age`	The age above which `cap2` applies.
`ecc`	(logical) Should an excess concessional contributions charge be calculated? (Not implemented.)
`use_other_contr`	Make a (poor) assumption that all 'Other contributions' (`MCS_Othr_Contr`) are concessional contributions. This may be a useful upper bound should such contributions be considered important.
`scale_contr_match_ato`	(logical) Should concessional contributions be inflated to match aggregates in 2013-14? That is, should concessional contributions by multiplied by `grattan:::super_contribution_` which was defined to be: $$\frac{\text{Total assessable contributions in SMSF and funds}}{\text{Total contributions in 2013-14 sample file}}$$.

`.lambda` Scalar weight applied to `concessional contributions`. $\lambda = 0$ means no (extra) weight. $\lambda = 1$ means contributions are inflated by the ratio of aggregates to the sample file's total. For $R =$ actual/apparent then the contributions are scaled by $1 + \lambda(R - 1)$.

`reweight_late_lodgers`
(logical) Should WEIGHT be inflated to account for late lodgers?

`.mu` Scalar weight for WEIGHT. ($w' = \mu w$) No effect if `reweight_late_lodgers` is `FALSE`.

`impute_zero_concess_contr`
Should zero concessional contributions be imputed using salary?

`.min.Sw.for.SG` The minimum salary required for super guarantee to be imputed.

`.SG_rate` The super guarantee rate for imputation.

`warn_if_colnames_overwritten`
(logical) Issue a warning if the construction of helper columns will overwrite existing column names in `.sample.file`.

`drop_helpers` (logical) Should columns used in the calculation be dropped before the sample file is returned?

`copyDT` (logical) Should the data table be `copy()`d? If the action of this data table is being compared, possibly useful.

Value

A data table comprising the original sample file (`.sample.file`) with extra superannuation policy-relevant variables for the policy specified by the function.

Author(s)

Hugh Parsonage, William Young

---

aus_pop_qtr *Australia's population*

Description

Australia's population

Usage

```
aus_pop_qtr(date_quarter, allow.projections = TRUE)
```

Arguments

`date_quarter` A character string (YYYY-QQ).

`allow.projections`
If the date is beyond the ABS's confirmed data, should a projection be used?

Value

The population at `date_quarter`, or at the most recent year in the data if projections are disallowed.

`aus_pop_qtr_age` *Australian estimated resident population by age and date*

Description

Australian estimated resident population by age and date

Usage

```
aus_pop_qtr_age(date = NULL, age = NULL, tbl = FALSE, roll = TRUE,
  roll.beyond = FALSE)
```

Arguments

`date`	A vector of dates. If `NULL`, values for all dates are returned in a table. The dates need not be quarters, provided `roll != FALSE`,
`age`	A vector of (integer) ages from 0 to 100 inclusive. If `NULL`, all ages are returned.
`tbl`	Should a table be returned? If `FALSE`, a vector is returned.
`roll`	Should a rolling join be performed?
`roll.beyond`	Should inputs be allowed to go beyond the limits of data (without a warning)? This is passed to `data.table`'s join, so options other than `TRUE` and `FALSE` are available. See `?data.table`.

Value

A `data.table` or `vector` with values of the estimated resident population.

Examples

```
aus_pop_qtr_age(date = as.Date("2016-01-01"), age = 42)
```

bto *Beneficiary tax offset*

Description

Beneficiary tax offset

Usage

```
bto(benefit_amount, fy.year = NULL, rate1 = 0.15,
  benefit_threshold = 6000, tax_threshold = 37000, rate2 = 0.15)
```

Arguments

benefit_amount	The amount of Tax Offsetable benefit received by the taxpayer during the income year.
fy.year	The income year. Not used by default.
rate1	The coefficient in Division 2, section 13(2) of the Income Tax Assessment (1936 Act) Regulation 2015 (the regulations).
benefit_threshold	The amount of benefits above which the offset applies.
tax_threshold	The *threshold at the upper conclusion of the lowest marginal tax rate* in the words of the section 13(3) of the regulations.
rate2	The second coefficient in section 13(3) of the regulations.

Value

The beneficiary tax offset.

WARNING

This function disagrees with the ATO online calculator.

CG_population_inflator *Forecasting capital gains*

Description

Forecasting capital gains

Usage

```
CG_population_inflator(x = 1, from_fy, to_fy, forecast.series = "mean",
  cg.series)

CG_inflator(x = 1, from_fy, to_fy, forecast.series = "mean")
```

Arguments

x	To be inflated.
from_fy, to_fy	Financial years designating the inflation period.
forecast.series	One of "mean", "lower", "upper". What estimator to use in forecasts. "lower" and "upper" give the lower and upper boundaries of the 95% prediction interval.
cg.series	(Not implemented.)

Value

For CG_population_inflator, the number of individuals estimated to incur capital gains in fy_year. For CG_inflator, an estimate of the nominal value of (total) capital gains in to_fy relative to the nominal value in from_fy.

---

cpi_inflator *CPI inflator*

Description

CPI inflator

Usage

```
cpi_inflator(from_nominal_price = 1, from_fy, to_fy = "2014-15",
  adjustment = c("seasonal", "none", "trimmed.mean"),
  useABSConnection = FALSE, allow.projection = TRUE)
```

Arguments

from_nominal_price	(numeric) the price (or vector of prices) to be inflated
from_fy	(character) a character vector with each element in the form "2012-13" representing the financial year contemporaneous to the from_nominal_price.
to_fy	(character) a character vector with each element in the form "2012-13" representing the financial year that prices are to be inflated.
adjustment	What CPI index to use ("none" = raw series, "seasonal", or "trimmed" [mean]).

useABSConnection
: Should the function connect with ABS.Stat via an SDMX connection? If FALSE (the default), a pre-prepared index table is used. This is much faster and more reliable (in terms of errors), though of course relies on the package maintainer to keep the tables up-to-date. The internal data is up-to-date as of 2017-Q4. If using useABSConnection = TRUE, ensure you have rsdmx (>= 0.5-10) up-to-date.

allow.projection
: Should projections beyond the ABS's data be allowed?

Value

The value of from_nominal_price in real (to_fy) dollars.

Examples

```
cpi_inflator(100, from_fy = "2005-06", to_fy = "2014-15")
```

---

cpi_inflator_general_date

CPI for general dates

Description

CPI for general dates

Usage

```
cpi_inflator_general_date(from_nominal_price = 1, from_date, to_date, ...)
```

Arguments

from_nominal_price
: (numeric) the nominal prices to be converted to a real price

from_date
: (character, date-like) the 'date' contemporaneous to from_nominal_price. The acceptable forms are 'YYYY', 'YYYY-YY' (financial year), 'YYYY-MM-DD', and 'YYYY-Q[1-4]' (quarters). Note a vector cannot contain a mixture of date forms.

to_date
: (character, date-like) the date at which the real price is valued (where the nominal price equals the real price). Same forms as for from_date

...
: other arguments passed to cpi_inflator_quarters

Value

A vector of real prices in to_date dollars.

cpi_inflator_quarters *CPI inflator when dates are nice*

Description

CPI inflator when dates are nice

Usage

```
cpi_inflator_quarters(from_nominal_price, from_qtr, to_qtr,
  adjustment = c("seasonal", "trimmed", "none"), useABSConnection = FALSE)
```

Arguments

from_nominal_price	(numeric) the nominal prices to be converted to a real price
from_qtr	(date in quarters) the dates contemporaneous to the prices in from_nominal_price. Must be of the form "YYYY-Qq" e.g. "1066-Q2". Q1 = Mar, Q2 = Jun, Q3 = Sep, Q4 = Dec.
to_qtr	(date in quarters) the date to be inflated to, where nominal price = real price. Must be of the form "YYYY-Qq" e.g. "1066-Q2".
adjustment	Should there be an adjustment made to the index? Adjustments include 'none' (no adjustment), 'seasonal', or 'trimmed' [referring to trimmed mean]. By default, `seasonal`.
useABSConnection	Should the function connect with ABS.Stat via an SDMX connection? By default set to `FALSE` in which case a pre-prepared index table is used. This is much faster and more reliable (in terms of errors), though of course relies on the package maintainer to keep the tables up-to-date. The internal data is up-to-date as of 2017-Q4. If using `useABSConnection = TRUE`, ensure you have `rsdmx (>= 0.5-10)` up-to-date.

Value

A vector of real prices.

differentially_uprate_wage *Differential uprating*

Description

Apply differential uprating to projections of the `Sw_amt` variable.

Usage

```
differentially_uprate_wage(wage = 1, from_fy, to_fy, ...)
```

Arguments

wage	A numeric vector to be uprated.
from_fy	The financial year contemporaneous to wage, which must be a financial year of an available sample file – in particular, not after 2013-14.
to_fy	The target of the uprating. Passed to wage_inflator.
...	Other arguments passed wage_inflator.

Details

See vignette("differential-uprating").

Value

The vector wage differentially uprated to to_fy.

Author(s)

Hugh Parsonage and William Young

Examples

```
ws <- c(20e3, 50e3, 100e3)
from <- "2013-14"
to <- "2016-17"
differentially_uprate_wage(ws, from, to)
differentially_uprate_wage(ws, from, to) / wage_inflator(ws, from, to)

# Use a wage series:
if (requireNamespace("taxstats", quietly = TRUE)) {
  library(data.table)
  library(taxstats)
  WageGrowth <- data.table(fy_year = c("2017-18", "2018-19"),
                           r = c(0.0, 0.1))
  Wage201314 <- sample_file_1314[["Sw_amt"]]

  data.table(Wage_201314 = Wage201314,
             Wage_201819 =
               differentially_uprate_wage(Wage201314,
                                          from_fy = "2013-14",
                                          to_fy = "2018-19",
                                          wage.series = WageGrowth))
}
```

gdp *Gross Domestic Product, Australia*

Description

Gross domestic product, at contemporaneous prices (called 'current prices' by the ABS).

Usage

```
gdp_qtr(date, roll = "nearest")

gdp_fy(fy_year)
```

Arguments

date	A Date vector or character coercible thereto.
roll	Passed to data.table when joining.
fy_year	Character vector of financial years.

Value

For gdp_qtr, the quarterly GDP for the quarter date nearest (or otherwise using roll). For gdp_fy the sum over the quarters in the financial year provided. If fy_year would provide incomplete data (i.e. only sum three or fewer quarters), a warning is issued. Dates or fy_year outside the available data is neither a warning nor an error, but NA.

Source

Australian Bureau of Statistics, Catalogue 5206.0. Series A2304350J.

generic_inflator *Generic inflator*

Description

Used to inflate variables in the sample file when there is no clear existing index. Note this is an unexported function: it is not available to the end-user.

Usage

```
generic_inflator(vars, h, fy.year.of.sample.file = "2012-13",
  nonzero = FALSE, estimator = "mean", pred_interval = 80)
```

Arguments

vars	A character vector of those variables within .sample_file for which forecasts are desired.
h	An integer, how many years ahead should the inflator be targeted.
fy.year.of.sample.file	A string representing the financial year of .sample_file.
nonzero	Should the forecast be taken on all values, or just nonzero values?
estimator	What forecast element should be used: the point estimate ("mean"), or the upper or lower endpoint of a prediction interval?
pred_interval	If estimator is upper or lower, what prediction interval are these the end points of?

Value

A data table of two columns: variable containing vars and inflator equal to the inflator to be applied to that variable to inflate it ahead h years.

---

gni *Gross National Income, Australia*

Description

Gross national income, at contemporaneous prices (called 'current prices' by the ABS).

Usage

```
gni_qtr(date, roll = "nearest")

gni_fy(fy_year)
```

Arguments

date	A Date vector or character coercible thereto.
roll	Passed to data.table when joining.
fy_year	Character vector of financial years.

Value

For gni_qtr, the quarterly GNI for the nearest quarter date. For gni_fy the sum over the quarters in the financial year provided. If fy_year would provide incomplete data (i.e. only sum three or fewer quarters), a warning is issued. Dates or fy_year outside the available data is neither a warning nor an error, but NA.

Source

Australian Bureau of Statistics, Catalogue 5206.0. Series A2304354T.

IncomeTax *IncomeTax*

Description

Calculates the ordinary tax payable given income and tax thresholds and rates. Basic, designed for performance.

Arguments

x	Taxable income.
thresholds	Lower brackets of the tax tables.
rates	Marginal rates

income_tax *Income tax payable*

Description

Income tax payable

Usage

```
income_tax(income, fy.year, age = NULL, family_status = "individual",
  n_dependants = 0L, .dots.ATO = NULL, return.mode = c("numeric",
  "integer"), allow.forecasts = FALSE, .debug = FALSE)
```

Arguments

income	The individual assessable income.
fy.year	The financial year in which the income was earned. Tax years 2000-01 to 2016-17 are provided, as well as the tax years 2017-18 to 2019-20, for convenience, under the assumption the 2017 Budget measures will pass. In particular, the tax payable is calculated under the assumption that the rate of the Medicare levy will rise to 2.5% in the 2019-20 tax year.
age	The individual's age.
family_status	For Medicare and SAPTO purposes.
n_dependants	An integer for the number of children of the taxpayer (for the purposes of the Medicare levy).
.dots.ATO	A data.frame that contains additional information about the individual's circumstances, with columns the same as in the ATO sample files. If .dots.ATO is a data.table, I recommend you enclose it with copy().
return.mode	The mode (numeric or integer) of the returned vector.

`allow.forecasts`
: should dates beyond 2019-20 be permitted? Currently, not permitted.

`.debug`
: (logical, default: `FALSE`) If `TRUE`, returns a `data.table` containing the components. (This argument and its result is liable to change in future versions, possibly without notice.)

Details

The function 'rolling' is inflexible by design. It is designed to guarantee the correct tax payable in a year. For years preceding the introduction of SAPTO, the maximum offset is assumed to apply to those above pensionable age.

Value

The total personal income tax payable.

Author(s)

Tim Cameron, Brendan Coates, Hugh Parsonage, William Young

Examples

```
income_tax(50e3, "2013-14")

## Calculate tax for each lodger in the 2013-14 sample file.
if (requireNamespace("taxstats", quietly = TRUE)) {
  library(data.table)
  library(taxstats)

  s1314 <- as.data.table(sample_file_1314)
  s1314[, tax := income_tax(Taxable_Income, "2013-14", .dots.ATO = s1314)]
}
```

---

income_tax_sapto *Income tax payable as a function of SAPTO*

Description

Income tax payable as a function of SAPTO

Usage

```
income_tax_sapto(income, fy.year, age = 42, family_status = "individual",
  n_dependants = 0L, return.mode = c("numeric", "integer"),
  .dots.ATO = NULL, allow.forecasts = FALSE, sapto.eligible,
  medicare.sapto.eligible, new_sapto_tbl = NULL)
```

Arguments

income	The individual assessable income.
fy.year	The financial year in which the income was earned. Only tax years from 2000-01 to 2016-17 are available.
age	The individual's age.
family_status	For Medicare and SAPTO purposes.
n_dependants	An integer for the number of children of the taxpayer (for the purposes of the Medicare levy).
return.mode	The mode (numeric or integer) of the returned vector.
.dots.ATO	A data.frame that contains additional information about the individual's circumstances, with columns the same as in the ATO sample files. If .dots.ATO is a data.table, I recommend you enclose it with copy().
allow.forecasts	should dates beyond 2016-17 be permitted? Currently, not permitted.
sapto.eligible	Specify explicitly the eligibility for SAPTO. If missing, defaults to ages over 65.
medicare.sapto.eligible	Specify explicitly the eligibility for SAPTO with respect to the Medicare levy for low-income earners. If missing, defaults to ages over 65.
new_sapto_tbl	If not NULL, supplied to new_sapto. Otherwise, fy.year is passed to sapto.

Details

Used to cost simple changes to SAPTO.

---

inflator *Inflate using a general index*

Description

Inflate using a general index

Usage

```
inflator(x = 1, from, to, inflator_table, index.col = "Index",
  time.col = "Time", roll = NULL)
```

Arguments

x	The vector to be inflated.
from	The contemporaneous time of x.
to	The target time (in units of the inflator_table) to which x is to be inflated.
inflator_table	A data.table having columns index.col and time.col.
index.col	The column in inflator_table containing the index used for inflation.
time.col	The column in inflator_table by which times are mapped.
roll	If NULL, inflation is calculated only on exact matches in inflator_table. Otherwise, uses a rolling join. See data.table::data.table.

Value

A vector of inflated values. For example, `inflator_table = grattan:::cpi_seasonal_adjustment`, `index.col = "obsValue"`, `time.col = "obsTime"`, gives the CPI inflator.

---

inverse_average_rate *Inverse average tax rate*

Description

Inverse average tax rate

Usage

```
inverse_average_rate(average_rate, ..., .max = 1e+08)
```

Arguments

`average_rate`	The average tax rate ($\frac{tax}{income}$)
`...`	Parameters passed to `income_tax`.
`.max`	The maximum income to test before ending the search. (Used only to prevent infinite loops.)

Value

The minimum income at which the average tax rate exceeds `average_rate`.

Examples

```
inverse_average_rate(0.2, fy.year = "2014-15")
```

---

inverse_income *Inverse income tax functions*

Description

Inverse income tax functions

Usage

```
inverse_income(tax, fy.year = "2012-13", zero.tax.income = c("maximum",
  "zero", "uniform", numeric(1)), ...)
```

Arguments

tax	The tax payable.
fy.year	The relevant financial year.
zero.tax.income	A character vector, ("maximum", "zero", "uniform", numeric(1)) Given that many incomes map to zero taxes, the income_tax function is not invertible there. As a consequence, the inverse function's value must be specified for tax = 0. "maximum" returns the maximum integer income one can have with a zero tax liability; "zero" returns zero for any tax of zero; "uniform" provides a random integer from zero to the maximum income with a zero tax. The value can also be specified explicitly.
...	Other arguments passed to income_tax. If tax or fy.year are vectors, these should be named vectors.

Details

This function has an error of $2.

Value

The approximate taxable income given the tax payable for the financial year. See Details.

---

is.fy *Convenience functions for dealing with financial years*

Description

Convenience functions for dealing with financial years

Arguments

yr_ending	An integer representing a year.
fy.yr	A string suspected to be a financial year.
date	A string or date for which the financial year is desired. Note that yr2fy does not check its argument is an integer.

Value

For is.fy, a logical, whether its argument is a financial year. The following forms are allowed: 2012-13, 201213, 2012 13, only. For fy.year, yr2fy, and date2fy, the financial year. For the inverses, a numeric corresponding to the year.

Examples

```
is.fy("2012-13")
is.fy("2012-14")
yr2fy(2012)
fy2yr("2015-16")
date2fy("2014-08-09")
```

`lf_inflator` *Labour force inflators*

Description

Labour force inflators

Usage

```
lf_inflator_fy(labour_force = 1, from_fy = "2012-13", to_fy,
  useABSConnection = FALSE, allow.projection = TRUE, use.month = 1L,
  forecast.series = c("mean", "upper", "lower", "custom"),
  forecast.level = 95, lf.series = NULL)

lf_inflator(labour_force = 1, from_date = "2013-06-30", to_date,
  useABSConnection = FALSE)
```

Arguments

`labour_force`	A numeric vector.
`from_fy`	Financial year of `labour_force`.
`to_fy`	Financial year for which the labour force is predicted.
`useABSConnection`	Should the function connect with ABS.Stat via an SDMX connection? If `FALSE` (the default), a pre-prepared index table is used. This is much faster and more reliable (in terms of errors), though of course relies on the package maintainer to keep the tables up-to-date. The internal data was updated on 2018-02-21 to include data up to 2018-01-01.
`allow.projection`	Logical. Should projections be allowed?
`use.month`	An integer (corresponding to the output of `data.table::month`) representing the month of the series used for the inflation.
`forecast.series`	Whether to use the forecast mean, or the upper or lower boundaries of the prediction intervals.
`forecast.level`	The prediction interval to be used if `forecast.series` is `upper` or `lower`.

lf.series	If forecast.series = 'custom', a data.table with two variables, fy_year and r. The variable fy_year consists of all financial years between the last financial year in the (known) labour force series and to_fy **inclusive**. The variable r consists of rates of labour force growth assumed in each fy_year, which must be 1 in the first year (to connect with the original labour force series).
from_date	The date of labour_force.
to_date	Dates as a character vector.

Details

lf_inflator is used on dates. The underlying data series is available every month.

Value

The relative labour force between to_date and for_date or to_fy and from_fy, multiplied by labour_force.

Author(s)

Hugh Parsonage and Tim Cameron

Source

ABS Cat 6202.0 http://www.abs.gov.au/ausstats/abs@.nsf/mf/6202.0?OpenDocument.

Examples

```
lf_inflator_fy(labour_force = 1, from_fy = "2012-13", to_fy = "2013-14")

library(data.table)
# Custom 1% growth over 2017-18 -> 2018.19
lf_inflator_fy(from_fy = "2017-18",
               to_fy = "2018-19",
               forecast.series = "custom",
               lf.series = data.table(fy_year = c("2017-18", "2018-19"),
                                      r = c(0, 0.01)))
## Not run:
lf_inflator(labour_force = 1, from_date = "2013-06-30", to_date = "2014-06-30")

## End(Not run)
```

---

lito *Low Income Tax Offset*

Description

The Low Income Tax Offset (LITO) is a non-refundable tax offset to reduce ordinary personal income tax for low-income earners.

Usage

```
.lito(input)

lito(income, max_lito = 445, lito_taper = 0.015, min_bracket = 37000)
```

Arguments

input	A keyed data.table containing the financial year and the input of every observation for which the LITO should be calculated. The input must have the following structure. **The structure will not be checked.** **fy_year** The financial year the LITO parameters should be obtained. This must be the key of the data.table. **income** The Taxable Income of the individual. **ordering** An integer sequence from 1 to `nrow(input)` which will be the order of the output.
income	Income of taxpayer
max_lito	The maximum LITO available.
lito_taper	The amount by which LITO should be shaded out or reduced for every additional dollar of taxable income.
min_bracket	The income at which the `lito_taper` applies.

Value

For `.lito`, the a numeric vector equal to the offset for each income and each financial year in `input`. For `lito`, a numeric vector equal to the offset for each income given the LITO parameters.

---

max_super_contr_base *Maximum superannuation contribution base*

Description

Data maximum super contribution base.

Usage

```
max_super_contr_base
```

Format

A data frame with 25 rows and 2 variables:

fy_year The financial year.

max_sg_per_qtr Maximum superannuation guarantee per quarter.

Source

ATO.

MedicareLevy *Medicare levy in C++*

Description

Medicare levy. Experimental function in C++, equivalent to `medicare_levy`.

Arguments

`income, SpouseIncome, isFamily, NDependants, lowerThreshold, upperThreshold, lowerFamilyThreshold,`	As in `medicare_levy`.
`rate, taper`	The parameters for the specific year or hypothetical requested.

Details

For `yr > 2018`, the 2017-18 values are used.

medicare_levy *Medicare levy*

Description

The (actual) amount payable for the Medicare levy.

Usage

```
medicare_levy(income, fy.year = "2013-14", Spouse_income = 0,
  sapto.eligible = FALSE, sato = NULL, pto = NULL,
  family_status = "individual", n_dependants = 0, .checks = TRUE)
```

Arguments

`income`	The taxable income. A vector of numeric values.
`fy.year`	The financial year. A character vector satisfying `is.fy`.
`Spouse_income`	The spouse's adjusted income.
`sapto.eligible`	(logical) Is the taxpayer eligible for SAPTO? See Details.
`sato`	Is the taxpayer eligible for the Senior Australians Tax Offset?
`pto`	Is the taxpayer eligible for the Pensions Tax Offset?
`family_status`	What is the taxpayer's family status: family or individual?
`n_dependants`	Number of children dependant on the taxpayer.
`.checks`	Should checks of certain arguments be made? Provided to improve performance when checks are not necessary.

Details

The Seniors and Pensioners Tax Offset was formed in 2012-13 as an amalgam of the Senior Australians Tax Offset and the Pensions Tax Offset. Medicare rates before 2012-13 were different based on these offsets. For most taxpayers, eligibility would be based on whether your age is over the pension age (currently 65). If `sato` and `pto` are `NULL`, `sapto.eligible` stands for eligibility for the `sato` and not `pto`. If `sato` or `pto` are not `NULL` for such years, only `sato` is currently considered. Supplying `pto` independently is currently a warning.

Value

The Medicare levy payable for that taxpayer.

`model_income_tax` *Modelled Income Tax*

Description

The income tax payable if tax settings are changed.

Usage

```
model_income_tax(sample_file, baseline_fy, n_dependants = 0L,
  elasticity_of_taxable_income = NULL, ordinary_tax_thresholds = NULL,
  ordinary_tax_rates = NULL, medicare_levy_taper = NULL,
  medicare_levy_rate = NULL, medicare_levy_lower_threshold = NULL,
  medicare_levy_upper_threshold = NULL,
  medicare_levy_lower_sapto_threshold = NULL,
  medicare_levy_upper_sapto_threshold = NULL,
  medicare_levy_lower_family_threshold = NULL,
  medicare_levy_upper_family_threshold = NULL,
  medicare_levy_lower_family_sapto_threshold = NULL,
  medicare_levy_upper_family_sapto_threshold = NULL,
  medicare_levy_lower_up_for_each_child = NULL, lito_max_offset = NULL,
  lito_taper = NULL, lito_min_bracket = NULL, sapto_eligible = NULL,
  sapto_max_offset = NULL, sapto_lower_threshold = NULL,
  sapto_taper = NULL, calc_baseline_tax = TRUE, return. = c("sample_file",
  "tax"))
```

Arguments

`sample_file`	A sample file having at least as many variables as the 2012-13 sample file.
`baseline_fy`	If a parameter is not selected, the parameter's value in this tax year is used.
`n_dependants`	The number of dependants for each entry in `sample_file`.

elasticity_of_taxable_income
: Either NULL (the default), or a numeric vector the same length of sample_file (or length-1) providing the elasticity of taxable income for each observation in sample_file;

$$\frac{\Delta z/z}{\Delta \tau/(1-\tau)}$$

where z is taxable income and τ is tax payable.

For example, if, for a given taxpayer, the tax settings would otherwise result in a 2% decrease of disposable income under the tax settings to be modelled, and elasticity_of_taxable_income is set to 0.1, the Taxable_Income is reduced by 0.2% before the tax rates are applied.

If NULL, an elasticity of 0 is used.

ordinary_tax_thresholds
: A numeric vector specifying the lower bounds of the brackets for "ordinary tax" as defined by the Regulations. The first element should be zero if there is a tax-free threshold.

ordinary_tax_rates
: The marginal rates of ordinary tax. The first element should be zero if there is a tax-free threshold. Since the temporary budget repair levy was imposed on a discrete tax bracket when it applied, it is not included in this function.

medicare_levy_taper
: The taper that applies between the _lower and _upper thresholds.

medicare_levy_rate
: The ordinary rate of the Medicare levy for taxable incomes above medicare_levy_upper_threshold.

medicare_levy_lower_threshold
: Minimum taxable income at which the Medicare levy will be applied.

medicare_levy_upper_threshold
: Minimum taxable income at which the Medicare levy will be applied at the full Medicare levy rate (2% in 2015-16). Between this threshold and the medicare_levy_lower_threshold, a tapered rate applies, starting from zero and climbing to medicare_levy_rate.

medicare_levy_lower_sapto_threshold, medicare_levy_upper_sapto_threshold
: The equivalent values for SAPTO-eligible individuals (not families).

medicare_levy_lower_family_threshold, medicare_levy_upper_family_threshold
: The equivalent values for families.

medicare_levy_lower_family_sapto_threshold, medicare_levy_upper_family_sapto_threshold
: The equivalent values for SAPTO-eligible individuals in a family.

medicare_levy_lower_up_for_each_child
: The amount to add to the _family_thresholds for each dependant child.

lito_max_offset
: The maximum offset available for low incomes.

lito_taper
: The taper to apply beyond lito_min_bracket.

lito_min_bracket
: The taxable income at which the value of the offset starts to reduce (from lito_max_offset).

sapto_eligible
: Whether or not each taxpayer in sample_file is eligible for SAPTO. If NULL, the default, then eligibility is determined by age_range in sample_file; *i.e.*, if age_range <= 1 then the taxpayer is assumed to be eligible for SAPTO.

sapto_max_offset
: The maximum offset available through SAPTO.

sapto_lower_threshold
: The threshold at which SAPTO begins to reduce (from sapto_max_offset).

sapto_taper
: The taper rate beyond sapto_lower_threshold.

calc_baseline_tax
: (logical, default: TRUE) Should the income tax in baseline_fy be included as a column in the result?

return.
: What should the function return? One of tax or sample_file. If tax, the tax payable under the settings; if sample_file, the sample_file, but with variables tax and possibly new_taxable_income.

---

model_new_caps_and_div293
Modelling superannuation changes

Description

Modelling superannuation changes

Usage

```
model_new_caps_and_div293(.sample.file, fy.year, new_cap = 30000,
  new_cap2 = 35000, new_age_based_cap = TRUE, new_cap2_age = 49,
  new_ecc = FALSE, new_div293_threshold = 3e+05, use_other_contr = FALSE,
  scale_contr_match_ato = FALSE, .lambda = 0,
  reweight_late_lodgers = TRUE, .mu = 1.05,
  impute_zero_concess_contr = TRUE, .min.Sw.for.SG = 450 * 12,
  .SG_rate = 0.0925, prv_cap = 30000, prv_cap2 = 35000,
  prv_age_based_cap = TRUE, prv_cap2_age = 49, prv_ecc = FALSE,
  prv_div293_threshold = 3e+05)

n_affected_from_new_cap_and_div293(..., adverse_only = TRUE)

revenue_from_new_cap_and_div293(...)
```

Arguments

.sample.file
: A data.table whose variables include those in taxstats::sample_file_1314.

fy.year
: The financial year tax scales.

new_cap
: The **proposed** cap on concessional contributions for all taxpayers if age_based_cap is FALSE, or for those below the age threshold otherwise.

new_cap2
: The **proposed** cap on concessional contributions for those above the age threshold. No effect if age_based_cap is FALSE.

`new_age_based_cap`
: Is the **proposed** cap on concessional contributions age-based?

`new_cap2_age`
: The age above which `new_cap2` applies.

`new_ecc`
: (logical) Should an excess concessional contributions charge be calculated? (Not implemented.)

`new_div293_threshold`
: The **proposed** Division 293 threshold.

`use_other_contr`
: Should `MCS_Othr_Contr` be used to calculate Division 293 liabilities?

`scale_contr_match_ato`
: (logical) Should concessional contributions be inflated to match aggregates in 2013-14? That is, should concessional contributions by multiplied by `grattan:::super_contribution_` which was defined to be:

$$\frac{\text{Total assessable contributions in SMSF and funds}}{\text{Total contributions in 2013-14 sample file}}$$

.

`.lambda`
: Scalar weight applied to `concessional contributions`. $\lambda = 0$ means no (extra) weight. $\lambda = 1$ means contributions are inflated by the ratio of aggregates to the sample file's total. For $R =$ actual/apparent then the contributions are scaled by $1 + \lambda(R - 1)$.

`reweight_late_lodgers`
: (logical) Should WEIGHT be inflated to account for late lodgers?

`.mu`
: Scalar weight for WEIGHT. ($w' = \mu w$) No effect if `reweight_late_lodgers` is `FALSE`.

`impute_zero_concess_contr`
: Should zero concessional contributions be imputed using salary?

`.min.Sw.for.SG`
: The minimum salary required for super guarantee to be imputed.

`.SG_rate`
: The super guarantee rate for imputation.

`prv_cap`
: The **comparator** cap on concessional contributions for all taxpayers if `age_based_cap` is FALSE, or for those below the age threshold otherwise.

`prv_cap2`
: The **comparator** cap on concessional contributions for those above the age threshold. No effect if `age_based_cap` is FALSE.

`prv_age_based_cap`
: Is the **comparator** cap on concessional contributions age-based?

`prv_cap2_age`
: The age above which `new_cap2` applies.

`prv_ecc`
: (logical) Should an excess concessional contributions charge be calculated? (Not implemented.)

`prv_div293_threshold`
: The **comparator** Division 293 threshold.

`...`
: Passed to `model_new_caps_and_div293`.

`adverse_only`
: Count only individuals who are adversely affected by the change.

Value

For `model_new_caps_and_div293`, A data.frame, comprising `.sample.file`, the superannuation variables generated by `apply_super_caps_and_div293`, and two variables `prv_revenue` and `new_revenue` which give the tax (income tax, super tax, and division 293 tax) payable by that taxpayer in the comparator scenario and the proposed scenario, respectively.

For `n_affected_from_new_cap_and_div293`, the number of individuals affected by the proposed changes.

For `revenue_from_new_cap_and_div293`, the extra revenue expected from the proposed changes.

`new_income_tax` *New income tax payable Income tax payable with new tax brackets, tax rates etc*

Description

New income tax payable Income tax payable with new tax brackets, tax rates etc

Usage

```
new_income_tax(income, new_tax_tbl)
```

Arguments

`income`	A vector of taxable incomes.
`new_tax_tbl`	A `data.table` with columns `lower_bracket` and `marginal_rate` for the new brackets and marginal rates.

Value

The income according to the new parameters.

`new_medicare_levy` *New medicare levy*

Description

Use a different way to calculate medicare levy.

Usage

```
new_medicare_levy(parameter_table)
```

Arguments

parameter_table
: A data.table containing

  switches The value in a row specifying which different medicare function is to apply.

  lower_threshold What is the lower medicare threshold, below which no medicare levy is applied, above which a tapering rate applies.

  taper What is the taper above lower_threshold.

  rate The medicare levy applicable above the medicare thresholds.

  lower_up_for_each_child How much the lower threshold should increase with each n_dependants.

  lower_family_threshold The threshold as applied to families (i.e. couples)

Value

A function similar to medicare_levy.

---

new_sapto *SAPTO with user-defined thresholds*

Description

SAPTO with user-defined thresholds

Usage

```
new_sapto(rebate_income, new_sapto_tbl, sapto.eligible = TRUE,
  Spouse_income = 0, fill = 0, family_status = "single")
```

Arguments

rebate_income	The rebate income of the individual.
new_sapto_tbl	Having the same columns as grattan:::sapto_tbl, keyed on family_status.
sapto.eligible	Is the individual eligible for sapto?
Spouse_income	Spouse income whose unutilized SAPTO may be added to the current taxpayer. Must match family_status; i.e. can only be nonzero when family_status != "single".
fill	If SAPTO was not applicable, what value should be used?
family_status	Family status of the individual.

npv *Financial functions*

Description

Financial functions from Excel. These functions are equivalent to the Excel functions of the same name (in uppercase).

Usage

```
npv(rate, values)

irr(x, start = 0.1)

fv(rate, nper, pmt, pv = 0, type = 0)

pv(rate, nper, pmt, fv = 0, type = 0)

pmt(rate, nper, pv, fv = 0, type = 0)
```

Arguments

rate	Discount or interest rate.
values	Income stream.
x	Cash flow.
start	Initial guess to start the iterative process.
nper	Number of periods
pmt	Payments.
pv	Present value.
type	Factor.
fv	Future value.

Author(s)

Enrique Garcia M. <egarcia@egm.as>

Karsten W. <k.weinert@gmx.net>

Examples

```
npv(0.07, c(1, 2))
irr(x = c(1, -1), start = 0.1)
fv(0.04, 7, 1, pv = 0.0, type = 0)
pv(rate = 0.08, nper = 7, pmt = 1, fv = 0.0, type = 0)
pmt(rate = 0.025, nper = 7, pv = 0, fv = 0.0, type = 0)
```

Offset — *General offset in C++*

Description

Calculate the offset given a threshold, a maximum offset, and a taper.

Arguments

x	A vector of incomes etc.
y	The maximum offset available; the offset when x is zero.
a	The maximum value of x at which the maximum offset is available.
m	The taper rate (the **negative** slope).

pmax3 — *Threeway parallel maximum*

Description

Returns the parallel maximum of three

Arguments

x, y, z	Numeric vectors of identical lengths.

Value

The parallel maximum of the vectors.

pmaxC — *Parallel maximum*

Description

A faster pmax().

Arguments

x	A numeric vector.
a	A single numeric value.

Value

The parallel maximum of the input values. `pmax0(x)` is shorthand for `pmaxC(x, 0)`, i.e. convert negative values in `x` to 0.

Note

This function will always be faster than `pmax(x, a)` when `a` is a single value, but can be slower than `pmax.int(x, a)` when `x` is short. Use this function when comparing a numeric vector with a single value.

`pmaxV` *Parallel maximum*

Description

A faster `pmax()`.

Arguments

`x`	A numeric vector.
`y`	A numeric vector, the same length as x.

Value

The parallel maximum of the input values.

`pminC` *Parallel maximum*

Description

A faster `pmin()`.

Arguments

`x`	A numeric vector.
`a`	A single numeric value.

Value

The parallel minimum of the input values. The `0` versions are shortcuts for `a = 0`.

Note

This function will always be faster than `pmin(x, a)` when `a` is a single value, but can be slower than `pmin.int(x, a)` when `x` is short. Use this function when comparing a numeric vector with a single value.

pminV *Parallel maximum*

Description

A faster `pmin()`.

Arguments

`x`	A numeric vector.
`y`	A numeric vector, the same length as x.

Value

The parallel maximum of the input values.

prohibit_length0_vectors *Prohibit zero lengths*

Description

Tests whether any vectors have zero length.

Usage

```
prohibit_length0_vectors(...)
```

Arguments

`...`	A list of vectors

Value

An error message if any of the vectors `...` have zero length.

prohibit_unequal_length_vectors *Prohibit unequal length vectors*

Description

Tests whether all vectors have the same length.

Usage

```
prohibit_unequal_length_vectors(...)
```

Arguments

...	Vectors to test.

Value

An error message unless all of ... have the same length in which case NULL, invisibly.

prohibit_vector_recycling *Prohibit vector recycling*

Description

Tests (harshly) whether the vectors can be recycled safely.

Usage

```
prohibit_vector_recycling(...)
```

Arguments

...	A list of vectors

Value

An error message if the vectors are of different length (unless the alternative length is 1). The functions differ in their return values on success: prohibit_vector_recycling.MAXLENGTH returns the maximum of the lengths whereas prohibit_vector_recyling returns NULL. (Both functions return their values invisibly.)

Source

http://stackoverflow.com/a/9335687/1664978

Examples

```
## Not run:
# Returns nothing because they are of the same length
prohibit_vector_recycling(c(2, 2), c(2, 2))
# Returns nothing also, because the only different length is 1
prohibit_vector_recycling(c(2, 2), 1)
# Returns an error:
prohibit_vector_recycling(c(2, 2), 1, c(3, 3, 3))

## End(Not run)
```

`project` *A function for simple projections of tables of Australian Taxation Office tax returns.*

Description

A function for simple projections of tables of Australian Taxation Office tax returns.

Usage

```
project(sample_file, h = 0L, fy.year.of.sample.file = NULL, WEIGHT = 50L,
  excl_vars, forecast.dots = list(estimator = "mean", pred_interval = 80),
  wage.series = NULL, lf.series = NULL, .recalculate.inflators = FALSE,
  .copyDT = TRUE, check_fy_sample_file = TRUE)
```

Arguments

`sample_file`	A sample file, most likely the 2012-13 sample file. It is intended that to be the most recent.
`h`	An integer. How many years should the sample file be projected?
`fy.year.of.sample.file`	The financial year of `sample_file`. If `NULL`, the default, the number is inferred from the number of rows of `sample_file` to be one of `2012-13`, `2013-14`, or `2014-15`.
`WEIGHT`	The sample weight for the sample file. (So a 2% file has `WEIGHT` = 50.)
`excl_vars`	A character vector of column names in `sample_file` that should not be inflated. Columns not present in the 2013-14 sample file are not inflated and nor are the columns `Ind`, `Gender`, `age_range`, `Occ_code`, `Partner_status`, `Region`, `Lodgment_method`, and `PHI_Ind`.
`forecast.dots`	A list containing parameters to be passed to `generic_inflator`.
`wage.series`	See `wage_inflator`. Note that the `Sw_amt` will uprated by `differentially_uprate_wage`.
`lf.series`	See `lf_inflator_fy`.
`.recalculate.inflators`	(logical, default: `FALSE`. Should `generic_inflator()` or `CG_inflator` be called to project the other variables? Adds time.

`.copyDT` (logical, default: `TRUE`) Should a `copy()` of `sample_file` be made? If set to `FALSE`, will update `sample_file`.

`check_fy_sample_file`
(logical, default: `TRUE`) Should `fy.year.of.sample.file` be checked against `sample_file`? By default, `TRUE`, an error is raised if the base is not 2012-13, 2013-14, or 2014-15 and a warning is raised if the number of rows in `sample_file` is different to the known number of rows in the sample files.

Details

We recommend you use `sample_file_1213` over `sample_file_1314`, unless you need the superannuation variables, as the latter suggests lower-than-recorded tax collections.

Superannuation variables are inflated by a fixed rate of 5% p.a.

Value

A sample file of the same number of rows as `sample_file` with inflated values (including WEIGHT).

Examples

```
# install.packages('taxstats', repos = 'https://hughparsonage.github.io/drat')
if (requireNamespace("taxstats", quietly = TRUE) &&
    requireNamespace("data.table", quietly = TRUE)) {
  library(taxstats)
  library(data.table)
  sample_file <- copy(sample_file_1314)
  sample_file_1617 <- project(sample_file,
                              h = 3L, # to "2016-17"
                              fy.year.of.sample.file = "2013-14")
}
```

`project_to` *A function for simple projections of sample files*

Description

A function for simple projections of sample files

Usage

```
project_to(sample_file, to_fy, fy.year.of.sample.file = NULL, ...)
```

Arguments

sample_file	A sample file, most likely the 2012-13 sample file. It is intended that to be the most recent.
to_fy	A string like "1066-67" representing the financial year for which forecasts of the sample file are desired.
fy.year.of.sample.file	The financial year of sample_file. See project for the default.
...	Other arguments passed to project.

Value

A sample file of the same number of rows as sample_file with inflated values (including WEIGHT).

---

rebate_income *Rebate income*

Description

Rebate income

Usage

```
rebate_income(Taxable_Income, Rptbl_Empr_spr_cont_amt = 0,
  All_deductible_super_contr = 0, Net_fincl_invstmt_lss_amt = 0,
  Net_rent_amt = 0, Rep_frng_ben_amt = 0)
```

Arguments

Taxable_Income	the taxable income
Rptbl_Empr_spr_cont_amt	The reportable employer superannuation contributions amount
All_deductible_super_contr	deductible personal superannuation contributions
Net_fincl_invstmt_lss_amt	Net financial investment loss
Net_rent_amt	(for Rental deductions)
Rep_frng_ben_amt	Reportable fringe-benefits

Source

https://www.ato.gov.au/Individuals/Tax-return/2015/Tax-return/Tax-offset-questions-T1-T2/Rebate-income-2015/

residential_property_prices *Residential property prices in Australia*

Description

Residential property prices indexes for the capital cities of Australia, and a weighted average for the whole country.

Usage

```
residential_property_prices
```

Format

A `data.table` of three columns and 486 observations:

Date Date of the index

City Capital city (or `Australia (weighted average)`)

Residential_property_price_index An index (100 = 2011-12-01) measuring the price change in all residential dwellings.

Source

ABS Cat 6416.0. http://www.abs.gov.au/ausstats/abs@.nsf/mf/6416.0.

sapto *Seniors and Pensioner Tax Offset*

Description

Seniors and Pensioner Tax Offset

Usage

```
sapto(rebate_income, fy.year, fill = 0, sapto.eligible = TRUE,
  Spouse_income = 0, family_status = "single", .check = TRUE)
```

Arguments

`rebate_income`	The rebate income of the individual.
`fy.year`	The financial year in which sapto is to be calculated.
`fill`	If SAPTO was not applicable, what value should be used?
`sapto.eligible`	Is the individual eligible for sapto?
`Spouse_income`	Spouse income whose unutilized SAPTO may be added to the current taxpayer. Must match `family_status`; i.e. can only be nonzero when `family_status != "single"`.
`family_status`	Family status of the individual.
`.check`	Run checks for consistency of values. For example, ensuring no single individuals have positive `Spouse_income`.

`sapto_rcpp` *SAPTO in C++*

Description

An implementation of SAPTO in C++.

Usage

```
sapto_rcpp(RebateIncome, MaxOffset, LowerThreshold, TaperRate, SaptoEligible,
  SpouseIncome, IsMarried)
```

Arguments

`RebateIncome, MaxOffset, LowerThreshold, TaperRate, SaptoEligible, SpouseIncome, IsMarried`	As in `sapto`.

`sapto_rcpp_singleton` *SAPTO singleton*

Description

Length-one version of `SAPTO` in C++.

Usage

```
sapto_rcpp_singleton(rebate_income, max_offset, lower_threshold, taper_rate,
  sapto_eligible, Spouse_income, is_married)
```

Arguments

`rebate_income, max_offset, lower_threshold, taper_rate, sapto_eligible, Spouse_income, is_married`	As in `sapto`.

sapto_rcpp_yr *SAPTO for specific years in C++*

Description

Fast way to calculate SAPTO for multiple people when the year is known in advance. Speed is by cheating and entering in the year's parameters literally.

Arguments

`RebateIncome, IsMarried, SpouseIncome`
As in `sapto`.

small_business_tax_offset *Small Business Tax Offset*

Description

Small Business Tax Offset

Usage

```
small_business_tax_offset(taxable_income, basic_income_tax_liability,
  .dots.ATO = NULL, aggregated_turnover = NULL,
  total_net_small_business_income = NULL, fy_year = NULL,
  tax_discount = NULL)
```

Arguments

`taxable_income` Individual's assessable income.

`basic_income_tax_liability`
Tax liability (in dollars) according to the method in the box in s 4.10(3) of the *Income Tax Assessment Act 1997* (Cth). In general, `basic_income_tax_liability` is the ordinary tax minus offsets. In particular, it does not include levies (such as the Medicare levy or the Temporary Budget Repair Levy).

$$\text{Income Tax} = \text{Taxable income} \times \text{Rate} - \text{Tax offsets}$$

For example, in 2015-16, an individual with an assessable income of \$100,000 had a basic tax liability of approximately \$25,000.

`.dots.ATO` A `data.table` of tax returns. If provided, it must contain the variables `Total_PP_BE_amt`, `Total_PP_BI_amt`, `Total_NPP_BE_amt`, `Total_NPP_BI_amt`. If both `.dots.ATO` and either `aggregated_turnover` or `total_net_small_business_income` are provided, `.dots.ATO` takes precedence, with a warning.

If `.dots.ATO` contains the variable `Tot_net_small_business_inc`, it is used instead of the income variables.

aggregated_turnover
: A numeric vector the same length as taxable_income. Only used to determine whether or not the offset is applicable; that is, the offset only applies if aggregated turnover is less than \$2M.

  Aggregated turnover of a taxpayer is the sum of the following:

  - the taxpayer's annual turnover for the income year,
  - the annual turnover of any entity connected with the taxpayer's, for that part of the income year that the entity is connected with the taxpayer's
  - the annual turnover of any entity that is an affiliate of the taxpayer, for that part of the income year that the entity is affiliated with the taxpayer's
  - When you calculate aggregated turnover for an income year, do not include either:
    - the annual turnover of other entities for any period of time that the entities are either not connected with the taxpayer or are not the taxpayer's affiliate, or
    - amounts resulting from any dealings between these entities for that part of the income year that the entity is connected or affiliated with the taxpayer.

  https://www.ato.gov.au/Business/Research-and-development-tax-incentive/Claiming-the-tax-offset/Steps-to-claiming-the-tax-offset/Step-3---Calculate-your-ag

total_net_small_business_income
: Total net business income within the meaning of the Act. For most taxpayers, this is simply any net income from a business they own (or their share of net income from a business in which they have an interest). The only difference being in the calculation of the net business income of some minors (vide Division 6AA of Part III of the Act).

fy_year
: The financial year for which the small business tax offset is to apply.

tax_discount
: If you do not wish to use the legislated discount rate from a particular fy_year, you can specify it via tax_discount. If both are provided, tax_discount prevails, with a warning.

Source

Basic income tax method s4-10(3) http://classic.austlii.edu.au/au/legis/cth/consol_act/itaa1997240/s4.10.html. Explanatory memorandum http://parlinfo.aph.gov.au/parlInfo/download/legislation/ems/r5494_ems_0a26ca86-9c3f-4ffa-9b81-219ac09be454/upload_pdf/503041.pdf.

---

student_repayment *HELP / HECS repayment amounts*

Description

HELP / HECS repayment amounts

Usage

```
student_repayment(repayment_income, fy.year, debt)
```

Arguments

repayment_income	The repayment income of the individual, equal to Taxable Income + Total net investment loss (incl Net rental loss) + reportable fringe benefits amounts + Reportable super contributions + exempt foreign income
fy.year	The financial year repayment_income was earned.
debt	The amount of student debt held.

Details

The student repayments for `fy.year = '2018-19'` assume the measures in Budget 2017 will pass.

Value

The repayment amount.

Author(s)

Ittima Cherastidtham and Hugh Parsonage

Source

https://www.ato.gov.au/Rates/HELP,-TSL-and-SFSS-repayment-thresholds-and-rates/?page=2#HELP_repayment_thresholds_and_rates_2013_14m https://docs.education.gov.au/system/files/doc/other/ed17-0138_-_he_-_glossy_budget_report_acc.pdf

Examples

```
student_repayment(50e3, "2013-14", debt = 10e3)
# 0 since below the threshold

student_repayment(60e3, "2013-14", debt = 10e3)
# above the threshold

student_repayment(60e3, "2013-14", debt = 0)
# above the threshold, but no debt
```

wage_inflator *Inflation using the Wage Price Index.*

Description

Predicts the inflation of hourly rates of pay, between two financial years.

Usage

```
wage_inflator(wage = 1, from_fy, to_fy, useABSConnection = FALSE,
  allow.projection = TRUE, forecast.series = c("mean", "upper", "lower",
  "custom"), forecast.level = 95, wage.series = NULL)
```

Arguments

wage	The amount to be inflated (1 by default).
from_fy	A character vector of the form "2012-13" representing the FY ending that the wage index is to be taken (i.e. Q4 in that year). FY year must be 1996-97 or later.
to_fy	The FY ending that the wage index is to be taken.
useABSConnection	Should the function connect with ABS.Stat via an SDMX connection? If FALSE (the default), a pre-prepared index table is used. This is much faster and more reliable (in terms of errors), though of course relies on the package maintainer to keep the tables up-to-date. The internal data was updated on 2018-02-21 to include data up to 2017-Q4.
allow.projection	If set to TRUE the forecast package is used to project forward, if required.
forecast.series	Whether to use the forecast mean, or the upper or lower boundaries of the prediction intervals. A fourth option custom allows manual forecasts to be set.
forecast.level	The prediction interval to be used if forecast.series is upper or lower.
wage.series	If forecast.series = 'custom', how future years should be inflated. The future wage series can be provided in two ways: (1) a single value, to be the assumed rate of wage inflation in years beyond the known series, or (2) a data.table with two variables, fy_year and r. If (2), the variable fy_year must be a vector of all financial years after the last financial year in the (known) wage series and the latest to_fy **inclusive**. The variable r consists of rates of wage growth assumed in each fy_year.

Value

The wage inflation between the two years.

Examples

```
# Wage inflation
wage_inflator(from_fy = "2013-14", to_fy = "2014-15")

# Custom wage inflation
wage_inflator(from_fy = "2016-17",
              to_fy = "2017-18",
              forecast.series = "custom",
              wage.series = 0.05)
```

`weighted_ntile` *Weighted quantiles*

Description

Weighted quantiles

Usage

```
weighted_ntile(vector, weights = rep(1, length(vector)), n)
```

Arguments

`vector`	The vector for which quantiles are desired.
`weights`	The weights associated with the vector. None should be `NA` or zero.
`n`	The number of quantiles desired.

Details

With a short-length vector, or with weights of a high variance, the results may be unexpected.

Value

A vector of integers corresponding to the ntiles. (As in `dplyr::ntile`.)

Examples

```
weighted_ntile(1:10, n = 5)
weighted_ntile(1:10, weights = c(rep(4, 5), rep(1, 5)), n = 5)
```

Index