

# Package ‘grattan’

July 2, 2017

**Type** Package

**Title** Perform Common Quantitative Tasks for Australian Analysts and to Support Grattan Institute Analysis

**Version** 1.5.1.1

**Date** 2017-07-02

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

**URL** <https://github.com/HughParsonage/grattan>

**BugReports** <https://github.com/HughParsonage/grattan/issues>

**Description** A series of functions focused on costing and evaluating Australian tax policy in support of the 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 (>= 2.10)

**License** GPL-2

**Imports** data.table, dplyr (>= 0.5.0), rsdmx, forecast, lubridate, assertthat, magrittr, Rcpp (>= 0.12.3), zoo, lazyeval, purrr

**LinkingTo** Rcpp

**RoxygenNote** 6.0.1

**Suggests** testthat, taxstats, rmarkdown, dtplyr, ggplot2, scales, broom, knitr, survey, viridis, ggrepel

**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]

**Repository** CRAN

**Date/Publication** 2017-07-02 05:44:59 UTC

## R topics documented:

age_grouper . . . . .	3
apply_super_caps_and_div293 . . . . .	4
aus_pop_qtr . . . . .	6
aus_pop_qtr_age . . . . .	6
bto . . . . .	7
CG_population_inflator . . . . .	8
cpi_inflator . . . . .	8
cpi_inflator_general_date . . . . .	9
cpi_inflator_quarters . . . . .	10
differentially_uprate_wage . . . . .	11
gdp . . . . .	12
generic_inflator . . . . .	12
gni . . . . .	13
income_tax . . . . .	14
income_tax_sapto . . . . .	15
inflator . . . . .	16
inverse_average_rate . . . . .	16
inverse_income . . . . .	17
is.fy . . . . .	18
lf_inflator . . . . .	18
lito . . . . .	20
max_super_contr_base . . . . .	20
medicare_levy . . . . .	21
model_new_caps_and_div293 . . . . .	22
new_income_tax . . . . .	24
new_medicare_levy . . . . .	24
new_sapto . . . . .	25
npv . . . . .	26
pmax3 . . . . .	27
pmaxC . . . . .	27
pmaxV . . . . .	28
pminC . . . . .	28
pminV . . . . .	29
prohibit_length0_vectors . . . . .	29
prohibit_unequal_length_vectors . . . . .	30
prohibit_vector_recycling . . . . .	30
project . . . . .	31
project_to . . . . .	32
rebate_income . . . . .	33
residential_property_prices . . . . .	33
sapto . . . . .	34
student_repayment . . . . .	35

<i>age_grouper</i>	3
wage_inflator . . . . .	36
weighted_ntile . . . . .	37
<b>Index</b>	<b>38</b>

---

<i>age_grouper</i>	<i>Age grouper</i>
--------------------	--------------------

---

## Description

Age grouper

## Usage

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

## Arguments

<i>age</i>	A numeric age (in years).
<i>interval</i>	How big should the age range be. 25-34 means interval = 10.
<i>min_age</i>	What is the upper bound of the lowest bracket? ( <i>min_age</i> = 25 means 'Under 25' will be the lowest bracket.)
<i>max_age</i>	What is the lower bound of the highest bracket? ( <i>max_age</i> = 75 means '75+' will be the bracket.)
<i>breaks</i>	Specify breaks manually.
<i>labels</i>	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, div293 = TRUE, warn_if_colnames_overwritten = TRUE,
  drop_helpers = FALSE, copyDT = TRUE)
```

## Arguments

<code>.sample.file</code>	A data.table containing at least the variables <code>sample_file_1314</code> from the <code>taxs-tats</code> package.
<code>colname_concessional</code>	The name for concessional contributions.
<code>colname_div293_tax</code>	The name of the column containing the values of Division 293 tax payable for that taxpayer.
<code>colname_new_Taxable_Income</code>	The name of the column containing the new Taxable Income.
<code>div293_threshold</code>	The Division 293 threshold.
<code>cap</code>	The cap on concessional contributions for all taxpayers if <code>age_based_cap</code> is <code>FALSE</code> , or for those below the age threshold otherwise.
<code>cap2</code>	The cap on concessional contributions for those above the age threshold. No effect if <code>age_based_cap</code> is <code>FALSE</code> .
<code>age_based_cap</code>	Is the cap on concessional contributions age-based?
<code>cap2_age</code>	The age above which <code>cap2</code> applies.
<code>ecc</code>	(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 be 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  $R = \text{actual/apparent}$  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.

div293

(logical) Should Division 293 tax be calculated? If FALSE, `.sample.file` is returned immediately, with a warning (that you're using this function pointlessly!).

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	<i>Australia's population</i>
-------------	-------------------------------

---

### Description

Australia's population

### Usage

```
aus_pop_qtr(date_quarter, allow.projections = TRUE, fertility = c("high",
  "medium", "low"), mortality = c("high.LifeExpectancy",
  "medium.LifeExpectancy"))
```

### 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?
fertility	What fertility assumption should be used? Must be one of high, medium, low.
mortality	The assumption of future life expectancy. Must be high.LifeExpectancy or medium.LifeExpectancy.

### Value

The population at date\_quarter, or at the year if a projections.

---

aus_pop_qtr_age	<i>Australian estimated resident population by age and date</i>
-----------------	---

---

### 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 <code>roll != FALSE</code> ,
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 <code>data.table</code> 's <code>join</code> , so options other than TRUE and FALSE are available. See <code>?data.table</code> .

**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	<i>Beneficiary tax offset</i>
-----	-------------------------------

---

**Description**

Beneficiary tax offset

**Usage**

```
bto(benefit_amount, fy.year)
```

**Arguments**

benefit_amount	The amount of Tax Offsetable benefit received by the taxpayer during the income year.
fy.year	The income year.

**Value**

The beneficiary tax offset.

**WARNING**

This function disagrees with the ATO online calculator.

---

CG_population_inflator	<i>Forecasting capital gains</i>
------------------------	----------------------------------

---

**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	<i>CPI inflator</i>
--------------	---------------------

---

**Description**

CPI inflator

**Usage**

```
cpi_inflator(from_nominal_price = 1, from_fy, to_fy = "2014-15",
  adjustment = c("seasonal", "none", "trimmed.mean"),
  useABSCConnection = 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? 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.
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 = "seasonal", 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 mean'.
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.

**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**

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

**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)
```

---

gdp	<i>Gross Domestic Product, Australia</i>
-----	--

---

### 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 <code>data.table</code> 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	<i>Generic inflator</i>
------------------	-------------------------

---

### Description

Used to inflate variables in the sample file when there is no clear existing index.

### Usage

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

**Arguments**

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

---

<code>gni</code>	<i>Gross National Income, Australia</i>
------------------	---

---

**Description**

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

**Usage**

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

```
gni_fy(fy_year)
```

**Arguments**

<code>date</code>	A Date vector or character coercible thereto.
<code>roll</code>	Passed to <code>data.table</code> when joining.
<code>fy_year</code>	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.

---

income_tax	<i>Income tax payable</i>
------------	---------------------------

---

**Description**

Income tax payable

**Usage**

```
income_tax(income, fy.year, age = 42, family_status = "individual",
  n_dependants = 0L, .dots.ATO = NULL, return.mode = c("numeric",
  "integer"), allow.forecasts = 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.

**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

---

income_tax_sapto	<i>Income tax payable as a function of SAPTO</i>
------------------	--

---

## Description

Income tax payable as a function of SAPTO

## Usage

```
income_tax_sapto(income, fy.year, age = 42, family_status = "individual",
  n_dependants = 0L, .dots.ATO = NULL, return.mode = c("numeric",
    "integer"), 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).
.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 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 <a href="#">new_sapto</a> . Otherwise, fy.year is passed to <a href="#">sapto</a> .

## Details

Used to cost simple changes to SAPTO.

---

inflator	<i>Inflate using a general index</i>
----------	--------------------------------------

---

**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	<i>Inverse average tax rate</i>
----------------------	---------------------------------

---

**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 <a href="#">income_tax</a> .
.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	<i>Inverse income tax functions</i>
----------------	-------------------------------------

---

**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,
  useABSCConnection = 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,
  useABSCConnection = 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 an sdmx connection be used to get ABS data?
allow.projection	Logical. Should projections be allowed?
use.month	An integer (corresponding to the output of <code>data.table::month</code> ) 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 <b>inclusive</b> . 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 from\_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")
## Not run:
lf_inflator(labour_force = 1, from_date = "2013-06-30", to_date = "2014-06-30")

## End(Not run)
```

---

lito	<i>Low Income Tax Offset</i>
------	------------------------------

---

### 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. <b>The structure will not be checked.</b>  <b>fy_year</b> The financial year the LITO parameters should be obtained. This must be the key of the data.table. <b>income</b> The Taxable Income of the individual. <b>ordering</b> 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	<i>Maximum superannuation contribution base</i>
----------------------	---

---

### 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.

---

medicare_levy	<i>Medicare levy</i>
---------------	----------------------

---

**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**

<code>income</code>	The taxable income. A vector of numeric values.
<code>fy.year</code>	The financial year. A character vector satisfying <code>is.fy</code> .
<code>Spouse_income</code>	The spouse's adjusted income.
<code>sapto.eligible</code>	(logical) Is the taxpayer eligible for SAPTO? See Details.
<code>sato</code>	Is the taxpayer eligible for the Senior Australians Tax Offset?
<code>pto</code>	Is the taxpayer eligible for the Pensions Tax Offset?
<code>family_status</code>	What is the taxpayer's family status: family or individual?
<code>n_dependants</code>	Number of children dependant on the taxpayer.
<code>.checks</code>	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\_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 <b>proposed</b> cap on concessional contributions for all taxpayers if age_based_cap is FALSE, or for those below the age threshold otherwise.
new_cap2	The <b>proposed</b> cap on concessional contributions for those above the age threshold. No effect if age_based_cap is FALSE.
new_age_based_cap	Is the <b>proposed</b> 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 <b>proposed</b> 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 be multiplied by $\frac{\text{grattan:::super\_contribution}}{\text{which was defined to be:}}$
	$\frac{\text{Total assessable contributions in SMSF and funds}}{\text{Total contributions in 2013-14 sample file}}$
	.
.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 $R = \text{actual/apparent}$ 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 <b>comparator</b> cap on concessional contributions for all taxpayers if age_based_cap is FALSE, or for those below the age threshold otherwise.
prv_cap2	The <b>comparator</b> cap on concessional contributions for those above the age threshold. No effect if age_based_cap is FALSE.
prv_age_based_cap	Is the <b>comparator</b> 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 <b>comparator</b> 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.

---

<code>new_income_tax</code>	<i>New income tax payable Income tax payable with new tax brackets, tax rates etc</i>
-----------------------------	---

---

**Description**

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

**Usage**

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

**Arguments**

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

**Value**

The income according to the new parameters.

---

<code>new_medicare_levy</code>	<i>New medicare levy</i>
--------------------------------	--------------------------

---

**Description**

Use a different way to calculate medicare levy.

**Usage**

```
new_medicare_levy(parameter_table)
```



**Arguments**

parameter_table	A <code>data.table</code> 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 <code>n_dependants</code> .
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 <code>grattan::sapto_tbl</code> , keyed on <code>family_status</code> .
sapto.eligible	Is the individual eligible for sapto?
Spouse_income	Spouse income whose unutilized SAPTO may be added to the current taxpayer. Must match <code>family_status</code> ; i.e. can only be nonzero when <code>family_status != "single"</code> .
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)
```

---

pmax3	<i>Threeway parallel maximum</i>
-------	----------------------------------

---

**Description**

Returns the parallel maximum of three

**Arguments**

x, y, z            Numeric vectors of identical lengths.

**Value**

The parallel maximum of the vectors.

---

pmaxC	<i>Parallel maximum</i>
-------	-------------------------

---

**Description**

A faster pmax().

**Arguments**

x                    A numeric vector.  
a                    A single numeric value.

**Value**

The parallel maximum of the input values.

**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	<i>Parallel maximum</i>
-------	-------------------------

---

**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	<i>Parallel maximum</i>
-------	-------------------------

---

**Description**

A faster `pmin()`.

**Arguments**

x	A numeric vector.
a	A single numeric value.

**Value**

The parallel minimum of the input values.

**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 vector has 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).

**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	<i>A function for simple projections of tables of Australian Taxation Office tax returns.</i>
---------	---

---

## 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 = "2013-14",
        WEIGHT = 50L, excl_vars, forecast.dots = list(estimator = "mean",
        pred_interval = 80), wage.series = NULL, lf.series = NULL,
        .recalculate.inflators = FALSE, .copyDT = 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.
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 <a href="#">wage_inflator</a> .
lf.series	See <a href="#">lf_inflator_fy</a> .
.recalculate.inflators	Should generic_inflator() or CG_inflator be called to project the other variables? Adds time.
.copyDT	Should a copy() of sample_file be made? If set to FALSE, will update sample_file.

**Details**

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

**Value**

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

**Examples**

```
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"
}
```

---

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 = "2013-14", ...)
```

**Arguments**

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

**Value**

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



---

rebate_income	<i>Rebate income</i>
---------------	----------------------

---

**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	<i>Residential property prices in Australia</i>
-----------------------------	---

---

**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	<i>Seniors and Pensioner Tax Offset</i>
-------	---

---

**Description**

Seniors and Pensioner Tax Offset

**Usage**

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

**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.

---

student_repayment	<i>HELP / HECS repayment amounts</i>
-------------------	--------------------------------------

---

**Description**

HELP / HECS repayment amounts

**Usage**

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

**Arguments**

repayment\_income

The repayment income of the individual, equal to

$TaxableIncome + Totalnetinvestmentloss(inclNetrentalloss) + reportablefringebenefitsamount$

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://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](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? 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.
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 <b>inclusive</b> . The variable r consists of rates of wage growth assumed in each fy_year.

**Value**

The wage inflation between the two years.

---

weighted_ntile	<i>Weighted quantiles</i>
----------------	---------------------------

---

**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

## \*Topic **datasets**

max\_super\_contr\_base, [20](#)  
residential\_property\_prices, [33](#)  
.lito (lito), [20](#)

age\_grouper, [3](#)  
apply\_super\_caps\_and\_div293, [4](#)  
aus\_pop\_qtr, [6](#)  
aus\_pop\_qtr\_age, [6](#)

bto, [7](#)

CG\_inflator (CG\_population\_inflator), [8](#)  
CG\_population\_inflator, [8](#)  
cpi\_inflator, [8](#)  
cpi\_inflator\_general\_date, [9](#)  
cpi\_inflator\_quarters, [10](#)

date2fy (is.fy), [18](#)  
differentially\_uprate\_wage, [11](#)

fv (npv), [26](#)  
fy.year (is.fy), [18](#)  
fy2date (is.fy), [18](#)  
fy2yr (is.fy), [18](#)

gdp, [12](#)  
gdp\_fy (gdp), [12](#)  
gdp\_qtr (gdp), [12](#)  
generic\_inflator, [12](#)  
gni, [13](#)  
gni\_fy (gni), [13](#)  
gni\_qtr (gni), [13](#)

income\_tax, [14](#), [16](#)  
income\_tax\_sapto, [15](#)  
inflator, [16](#)  
inverse\_average\_rate, [16](#)  
inverse\_income, [17](#)  
irr (npv), [26](#)  
is.fy, [18](#)

lf\_inflator, [18](#)  
lf\_inflator\_fy, [31](#)  
lf\_inflator\_fy (lf\_inflator), [18](#)  
lito, [20](#)

max\_super\_contr\_base, [20](#)  
medicare\_levy, [21](#)  
model\_new\_caps\_and\_div293, [22](#)  
  
n\_affected\_from\_new\_cap\_and\_div293  
    (model\_new\_caps\_and\_div293), [22](#)  
new\_income\_tax, [24](#)  
new\_medicare\_levy, [24](#)  
new\_sapto, [15](#), [25](#)  
npv, [26](#)

pmax3, [27](#)  
pmaxC, [27](#)  
pmaxV, [28](#)  
pminC, [28](#)  
pminV, [29](#)  
pmt (npv), [26](#)  
prohibit\_length0\_vectors, [29](#)  
prohibit\_unequal\_length\_vectors, [30](#)  
prohibit\_vector\_recycling, [30](#)  
project, [31](#), [32](#)  
project\_to, [32](#)  
pv (npv), [26](#)

rebate\_income, [33](#)  
residential\_property\_prices, [33](#)  
revenue\_from\_new\_cap\_and\_div293  
    (model\_new\_caps\_and\_div293), [22](#)

sapto, [15](#), [34](#)  
student\_repayment, [35](#)

wage\_inflator, [11](#), [31](#), [36](#)  
weighted\_ntile, [37](#)

yr2fy (is.fy), [18](#)