Package 'grattan'

June 27, 2022

Type Package

Title Australian Tax Policy Analysis

Version 2.0.0.0

Date 2022-06-27

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

URL https://github.com/HughParsonage/grattan,
https://hughparsonage.github.io/grattan/

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

Description Utilities to cost and evaluate Australian tax policy, including fast projections of personal income tax collections, high-performance tax and transfer calculators, and an interface to common indices from the Australian Bureau of Statistics. Written to support Grattan Institute's Australian Perspectives program, and related projects. Access to the Australian Taxation Office's sample files of personal income tax returns is assumed.

Depends R (>= 3.5.0)

License GPL-2

Imports checkmate, data.table, hutils (>= 1.3.0), hutilscpp (>= 0.9.0), ineq (>= 0.2-10), fastmatch, forecast, fy (>= 0.2.0), assertthat (>= 0.1), magrittr (>= 1.5), utils, zoo (>= 1.5-5)

RoxygenNote 7.1.1

Suggests curl, fst (>= 0.8.4), knitr, rlang, rmarkdown, rsdmx, scales, survey, taxstats, taxstats1516, testthat, tibble, yaml, withr, covr

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

LazyData true

Encoding UTF-8

NeedsCompilation yes

Author Hugh Parsonage [aut, cre],
Tim Cameron [aut],
Brendan Coates [aut],
Matthew Katzen [aut],
William Young [aut],
Ittima Cherastidtham [dtc],
W. Karsten [ctb],
M. Enrique Garcia [ctb],
Matt Cowgill [aut]

Repository CRAN

Date/Publication 2022-06-27 08:30:02 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>

Package options

`grattan.verbose` (`FALSE`) Emit diagnostic messages (via `cat()`)

`grattan.assume1901_2100` (`TRUE`) Assume `yr2fy` receives an integer >= 1901 and <= 2100.

`grattan.taxstats.lib` Package library into which `taxstats` packages will be installed. If `NULL`, a temporary directory is used.

Author(s)

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

`<hugh.parsonage@gmail.com>`

See Also

Useful links:

- `https://github.com/HughParsonage/grattan`
- `https://hughparsonage.github.io/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,
  below = "Below\n",
  exp_min_age = 1L,
  exp_max_age = 100L,
  threshold = 10000L
)
```

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.
`below`	String giving the prefix for the lowest bin. (Only applicable if `breaks` and `labels` are `NULL`.)
`exp_min_age, exp_max_age`	Integers specifying the lowest/highest expected age in `age`. If any values fall outside this range, ages will still work though perhaps slow when `length(age)` >> `threshold`.
`threshold`	An integer, the minimum length at which the calculation will be accelerated.

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)
```

`age_pension_age` *Age of eligibility for the Age Pension*

Description

Age of eligibility for the Age Pension

Usage

```
age_pension_age(when = Sys.Date(), sex = "male")
```

Arguments

`when`	Either a Date (or a character vector coercible to such) or a financial year, when the age of eligibility of Age Pension is requested. Defaults to current date.
`sex`	A character vector the same length as `when`, containing strings `"male"` and `"female "`. May be abbreviated to `"m"` or `"f"` and is case-insensitive.

Value

A numeric vector, the age of eligiblity for the Age Pension for each `when`.

Source

https://guides.dss.gov.au/social-security-guide/3/4/1/10

Examples

```
age_pension_age()  # Current age of eligiblity
age_pension_age("1995-12-31")
age_pension_age("2013-14")
```

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 = 300000,
  cap = 30000,
  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 $R = \text{actual}/\text{apparent}$ 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)
```

`awote` *AWOTE*

Description

Adult weekly ordinary-time earnings

Usage

```
awote(
  Date = NULL,
  fy.year = NULL,
  rollDate = "nearest",
  isMale = NA,
  isAdult = TRUE,
  isOrdinary = TRUE
)
```

Arguments

`Date, fy.year`	When the AWOTE is desired.
`rollDate`	How should the `Date` be joined to the source data? Passed to `data.table`.
`isMale`	(logical, default: `NA`) `TRUE` for male weekly earnings, `FALSE` for female, `NA` for the weekly earnings of both sexes.
`isAdult`	(logical, default: `TRUE`) Use adult weekly earnings?
`isOrdinary`	Use ordinary weekly earnings?

Examples

```
awote()  # Current AWOTE
```

`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`.

`compare_avg_tax_rates` *Compare average tax rates by percentile*

Description

To determine the effects of bracket creep on a proposed tax policy, a common task is calculate the change in the average tax rates for each percentile. This function accepts a sample file and a baseline sample file, and returns a 100-row table giving the mean change in average tax rates for each percentile, compared to the baseline.

Usage

```
compare_avg_tax_rates(DT, baseDT, by = "id", ids = NULL)
```

Arguments

`DT`	A single `data.table` containing columns `new_tax`, `Taxable_Income`, `baseline_tax`.
`baseDT`	A `data.table` of a single cross-section of taxpayers from which baseline percentiles can be produced.
`by`	How to separate `DT`
`ids`	Subset `DT` by `by`.

`cpi_inflator` *CPI inflator*

Description

CPI inflator

Usage

```
cpi_inflator(
  from_nominal_price = 1,
  from_fy = NULL,
  to_fy = NULL,
  adjustment = c("seasonal", "none", "trimmed.mean"),
  useABSConnection = FALSE,
  allow.projection = TRUE,
  accelerate.above = 100000L
)
```

Arguments

`from_nominal_price`	(numeric) the price (or vector of prices) to be inflated
`from_fy, to_fy`	(character) a character vector with each element in the form "2012-13" representing the financial years between which the CPI inflator is desired. If both `from_fy` and `to_fy` are `NULL` (the default), `from_fy` is set to the previous financial year and `to_fy` to the current financial year, with a warning. Setting only one is an error.
`adjustment`	What CPI index to use ("none" = raw series, "seasonal", or "trimmed" [mean]).
`useABSConnection`	Defunct. ABS host not reliable. The internal data was updated on 2022-01-03 to 2021-Q3.
`allow.projection`	Should projections beyond the ABS's data be allowed?

`accelerate.above`	An integer setting the threshold for 'acceleration'. When the maximum length of the arguments exceeds this value, calculate each unique value individually then combine. Set to 100,000 as a rule of thumb beyond which calculation speeds benefit dramatically. Can be set to `Inf` to disable acceleration.

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 was updated on 2022-01-03 to 2021-Q3. Using `useABSConnection = TRUE` is no longer supported for server issues.

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 2016-17.
`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.

income_tax *Income tax payable*

Description

Income tax payable

Usage

```
income_tax(
  income,
  fy.year = NULL,
  age = NULL,
  .dots.ATO = NULL,
  System = NULL,
  return.mode = c("numeric", "integer")
)
```

Arguments

`income`	The individual assessable income.
`fy.year`	The financial year in which the income was earned. Tax years 2000-01 to 2018-19 are supported, as well as the tax year 2019-20, for convenience. If `fy.year` is not given, the current financial year is used by default.
`age`	The individual's age. Ignored if `.dots.ATO` is provided (and contains an age variable such as `age_range` or `Birth_year`).
`.dots.ATO`	A data.frame that contains additional information about the individual's circumstances, with columns the same as in the ATO sample files. Age variables in `.dots.ATO` take precedence over `age` and providing both is a warning.
`System`	A `tax-system` created by `System()` or `NULL`, the default, corresponding to the tax system of the given year.
`return.mode`	The mode (numeric or integer) of the returned vector.

Details

The function is inflexible by design. It is designed to return the correct tax payable in a year, not to model the tax payable under different tax settings. (Use `model_income_tax` for that purpose.)

The function aims to produce the personal income tax payable for the inputs given in the tax year `fy.year`. The function is specified to produce the most accurate calculation of personal income tax given the variables in the ATO's 2% sample files. However, many components are absent from these files, while other components could not be computed reliably.

For the 2018-19 tax year, the function calculates

tax on ordinary taxable income The tax as specified in Schedule 7 of the *Income Tax Rates Act 1986* (Cth).

Medicare levy See `medicare_levy` for details.

LITO See `lito` for details.

SAPTO See `sapto`. For years preceding the introduction of SAPTO, the maximum offset is assumed to apply to those above age 65 (since the sample files only provide 5-year age groups).

SBTO See `small_business_tax_offset` for details.

Historical levies The flood levy and the temporary budget repair levy.

Notably, when used with a 2% sample file, the function will not be able to correctly account for different tax rates and offsets among taxpayers with dependants since the sample files (as of 2015-16) do not have this information.

Value

The total personal income tax payable.

Author(s)

Tim Cameron, Brendan Coates, Matthew Katzen, Hugh Parsonage, William Young

Examples

```
## Income tax payable on a taxable income of 50,000
## for the 2013-14 tax year
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)]
}
```

---

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,
  max.length = 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`.
`max.length`	(Internal use only). If not `NULL`, the maximum length of `x`, `from`, and `to` known in advance. May be provided to improve the performance if known.

Value

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

`install_taxstats` *Install 'taxstats' files*

Description

The taxstats packages provide the sample files as released by the ATO. These packages are used for testing, but are not available through CRAN as they are too large.

Usage

```
install_taxstats(pkg = c("taxstats"), ...)
```

Arguments

`pkg`	The package to install such as `"taxstats"` or `"taxstats1516"`.
`...`	Arguments passed to `install.packages`.

`inverse_average_rate` *Inverse average tax rate*

Description

Inverse average tax rate

Usage

```
inverse_average_rate(average_rate, ..., .max = 100000000)
```

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

From grattan v1.7.1.4, these are reexports from the `fy-package`.

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.
`assume1901_2100`	For `yr2fy`, assume that `yr_ending` is between 1901 and 2100, for performance. By default, set to `getOption("grattan.assume1901_2100", TRUE)`.

Details

The following forms are permitted: `2012-13`, `201213`, `2012 13`, only. However, the `2012-13` form is preferred and will improve performance.

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.

`fy.year` is a deprecated alias for `yr2fy`, the latter is slightly more efficient, as well as more declarative.

`fy2yr` converts a financial year to the year ending: `fy2yr("2016-17")` returns 2017. `yr2fy` is the inverse: `yr2fy(fy2yr("2016-17")) == "2016-17"`.

`fy2date` converts a financial year to the 30 June of the financial year ending.

`date2fy` converts a date to the corresponding financial 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

This function's behaviour has changed due to COVID-19. In particular, the trend labour force status is no longer available.

Usage

```
lf_inflator_fy(
  labour_force = 1,
  from_fy = NULL,
  to_fy = NULL,
  useABSConnection = FALSE,
  allow.projection = TRUE,
  use.month = 1L,
  forecast.series = c("mean", "upper", "lower", "custom"),
  forecast.level = 95,
  lf.series = NULL,
  .lf_indices = NULL,
  accelerate.above = 100000L
)

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

Arguments

`labour_force`	A numeric vector.
`from_fy, to_fy`	(character) a character vector with each element in the form "2012-13" representing the financial years between which the labour force inflator is desired. If both `from_fy` and `to_fy` are `NULL` (the default), `from_fy` is set to the previous financial year and `to_fy` to the current financial year, with a warning. Setting only one is an error.
`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. If the SDMX connection fails, a message is emitted (not a warning) and the function contines as if `useABSConnection = FALSE`. The internal data was updated on 2022-01-03 to 2021-11-01.

`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).
`.lf_indices`	(Internal use only.) A `data.table` sent directly to `inflator` without any checks.
`accelerate.above`	An integer setting the threshold for 'acceleration'. When the maximum length of the arguments exceeds this value, calculate each unique value individually then combine. Set to 100,000 as a rule of thumb beyond which calculation speeds benefit dramatically. Can be set to `Inf` to disable acceleration.
`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)

Tim Cameron, Matthew Katzen, and Hugh Parsonage

Source

ABS Cat 6202.0 `https://www.abs.gov.au/statistics/labour/employment-and-unemployment/labour-force-australia/latest-release`.

Examples

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

library(data.table)
# Custom 1% growth over 2018-19 -> 2019-20
lf_inflator_fy(from_fy = "2018-19",
               to_fy = "2019-20",
                    forecast.series = "custom",
                    lf.series = data.table(fy_year = c("2018-19", "2019-20"),
                                           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.

N.B. Since v2.0.0, `lito` only calculates the actual LITO, rather than an offset with custom parameters. For such functionality, use (unexported) `Offset`.

Usage

```
lito(income, fy.year = NULL)

lmito(income, fy.year = NULL)
```

Arguments

`income`	The income on which the offset is applied.
`fy.year`	The financial year for which the LITO is desired.

Value

The LITO or LMITO for the given income and tax year.

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.

`medicare_levy` *Medicare levy*

Description

The (actual) amount payable for the Medicare levy.

Usage

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

Arguments

`income`	`numeric(N)` The income for medicare levy purposes of the taxpayer.
`fy.year`	`character(1)` **or** `character(N)` **or** `fy(N)` **or** `fy(1)` The tax year in which `income` was earned. A vector satisfying `fy::validate_fys_permitted`.
`Spouse_income`	`numeric(1)` **or** `numeric(N)` The income of the taxpayer's spouse. Missing values are imputed to zeroes. Values are truncated to integer.
`sapto.eligible`	`logical(1)` **or** `logical(N)` Is the taxpayer entitled to the SAPTO thresholds? Missing values are imputed to `FALSE`.
`sato, pto`	Is the taxpayer eligible for the Senior Australians Tax Offset or Pensions Tax Offset? `pto = TRUE` not supported and will be set to `FALSE`, with a warning.
`family_status`	(Deprecated: use 'is_married' and 'n_dependants' instead)

`n_dependants`	`integer(N)` **or** `integer(1)` Number of dependants the taxpayer has. If nonzero, the taxpayer is entitled to the family thresholds of the Medicare levy, and each dependant child increases the thresholds.
`is_married`	`logical(N)` Is the taxpayer married? Married individuals (or those whose `Spouse_income` > `0`) are deemed to be families when determining cut-off thresholds.
`.checks`	Whether or not to perform checks on inputs.

Details

The Medicare levy for individuals is imposed by the *Medicare Levy Act 1986* (Cth). The function only calculates the levy for individuals (not trusts). It includes the s 7 *Levy in cases of small incomes*, including the differences for those eligible for `sapto`. s 8 *Amount of levy—person who has spouse or dependants* (though the number of dependants is not a variable in the sample files).

The function does **not** include the Medicare levy surcharge; it assumes that all persons (who would potentially be liable for it) avoided it.

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.

See `http://classic.austlii.edu.au/au/legis/cth/consol_act/mla1986131/` for the *Medicare Levy Act 1986* (Cth).

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,
  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,
  lito_multi = NULL,
  offsets = NULL,
  Budget2018_lamington = FALSE,
  Budget2019_lamington = NA,
  Budget2018_lito_202223 = FALSE,
  Budget2018_watr = FALSE,
  Budget2019_watr = FALSE,
  sapto_eligible = NULL,
  sapto_max_offset = NULL,
  sapto_lower_threshold = NULL,
  sapto_taper = NULL,
  sapto_max_offset_married = NULL,
  sapto_lower_threshold_married = NULL,
  sapto_taper_married = NULL,
  sbto_discount = NULL,
  cgt_discount_rate = NULL,
  calc_baseline_tax = TRUE,
  return. = c("sample_file", "tax", "sample_file.int"),
  clear_tax_cols = TRUE,
  warn_upper_thresholds = TRUE,
  .debug = FALSE
)
```

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. Must be a valid tax year and one for which `income_tax` has been programmed.

`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`
: (deprecated) The maximum offset available for low incomes.

`lito_taper`
: (deprecated) The taper to apply beyond `lito_min_bracket`.

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

`lito_multi`
: No longer supported.

`offsets`
: A list of lists created by `set_offsets`. If `NULL`, the default, the list is populated by the offsets in `baseline_fy`.

`Budget2018_lamington`
: No longer supported

`Budget2019_lamington`
: No longer supported.

`Budget2018_lito_202223`	No longer supported.
`Budget2018_watr`	No longer supported
`Budget2019_watr`	No longer supported.
`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`.
`sapto_max_offset_married, sapto_lower_threshold_married, sapto_taper_married`	As above, but applied to members of a couple
`sbto_discount`	The `tax_discount` in `small_business_tax_offset`.
`cgt_discount_rate`	(numeric(1)) The capital gains tax discount rate, currently 50%.
`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`, `sample_file`, or `sample_file.int`. If `tax`, the tax payable under the settings; if `sample_file`, the `sample_file`, but with variables `tax` and possibly `new_taxable_income`; if `sample_file.int`, same as `sample_file` but `new_tax` is coerced to integer.
`clear_tax_cols`	If `TRUE`, the default, then `return. = sample_file` implies any columns called `new_tax` or `baseline_tax` in `sample_file` are dropped silently.
`warn_upper_thresholds`	If `TRUE`, the default, then any inconsistency between `baseline_fy` and the upper thresholds result in a warning. Set to `FALSE`, if the `lower_thresholds` may take priority.
`.debug`	Return a data.table of `new_tax`. Experimental so cannot be relied in future versions.

Examples

```
library(data.table)
library(hutils)

# With new tax-free threshold of $20,000:
if (requireNamespace("taxstats", quietly = TRUE) && FALSE) {
  library(taxstats)
  library(magrittr)
  model_income_tax(sample_file_1314,
                   "2013-14",
                   ordinary_tax_thresholds = c(0, 20e3, 37e3, 80e3, 180e3)) %>%
    select_grep("tax", "Taxable_Income")

}
```

model_new_caps_and_div293 *Modelling superannuation changes*

Description

Model changes to the contributions cap, Division 293 threshold and related modelling. Note: defaults are relevant to pre-2017 for compatibility.

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_contr_tax = "15%",
  new_div293_threshold = 300000,
  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 = 300000
)

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_contr_tax`	A string to determine the contributions tax.
`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 the concessional contributions by multiplied by the internal constant `grattan:::super_contribution_inflator_1314`, 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 the variables in `.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.

Examples

```
if (requireNamespace("taxstats", quietly = TRUE)) {
  library(data.table)
  s1314 <- taxstats::sample_file_1314
  s1314[, WEIGHT := 50L]
  revenue_from_new_cap_and_div293(s1314, new_cap = 12e3, "2016-17")
  revenue_from_new_cap_and_div293(s1314, new_contr_tax = "mr - 15%", "2016-17")
}
```

`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)
```

`progressivity` *Compute the progressivity*

Description

Compute the progressivity

Usage

```
progressivity(income, tax, measure = c("Reynolds-Smolensky", "Kakwani"))
```

Arguments

`income`	Pre-tax income.
`tax`	Tax paid.
`measure`	Currently, only "Reynolds-Smolensky" progressivity is calculated: $$G_Y - G_Z$$ where G_Y is the Gini coefficient of `income` and G_X is the Gini coefficient of post-tax income.

Value

The progressivity measure. Positive for progressive tax systems, and higher the value the more progressive the system.

Examples

```
I <- c(10e3, 20e3, 50e3, 100e3, 150e3)
progressivity(I, 0.3 * I) # zero
progressivity(I, income_tax(I, "2017-18"))
```

`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.

`project` *Simple projections of the annual 2% samples of Australian Taxation Office tax returns.*

Description

Simple projections of the annual 2% samples of Australian Taxation Office tax returns.

Usage

```
project(
  sample_file,
  h = 0L,
  fy.year.of.sample.file = NULL,
  WEIGHT = 50L,
  excl_vars = NULL,
  forecast.dots = list(estimator = "mean", pred_interval = 80),
  wage.series = NULL,
  lf.series = NULL,
  use_age_pop_forecast = FALSE,
  .recalculate.inflators = NA,
  .copyDT = TRUE,
  check_fy_sample_file = TRUE,
  differentially_uprate_Sw = NA,
  r_super_balance = 1.05
)
```

Arguments

`sample_file`	A `data.table` matching a 2% sample file from the ATO. See package `taxstats` for an example.
`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`, `2014-15`, `2015-16`, or `2016-17`.
`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` (if requested).
`lf.series`	See `lf_inflator_fy`.
`use_age_pop_forecast`	Should the inflation of the number of taxpayers be moderated by the number of resident persons born in a certain year? If `TRUE`, younger ages will grow at a slightly higher rate beyond 2018 than older ages.
`.recalculate.inflators`	(logical, default: `NA`). Should `generic_inflator()` or `CG_inflator` be called to project the other variables? Adds time. Default `NA` means `TRUE` if the pre-calculated inflators are available, `FALSE` otherwise.
`.copyDT`	(logical, default: `TRUE`) Should a `copy()` of `sample_file` be made? If set to `FALSE`, will update `sample_file` in place, which may be necessary when memory is constrained, but is dangerous as it modifies the original data and its projection. (So if you run the same code twice you may end up with a projection `2h` years ahead, not `h` years.)
`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, 2014-15, 2015-16, 2016-17, or 2017-18, 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.
`differentially_uprate_Sw`	(logical, default: `NA`) Should the salary and wage column (`Sw_amt`) be differentially uprated using (`differentially_uprate_wage`)? Default of `NA` means use differential uprating is used when `fy.year.of.sample.file` <= `"2016-17"`. It is known that the Treasury stopped using differential uprating by 2019. Selecting `TRUE` for `fy.year.of.sample.file` > `"2016-17"` is an error as the precalculated values are not available.

`r_super_balance`
: The factor to inflate super balances by (annualized). Set to `1.05` for backwards compatibility. The annual superannuation bulletin of June 2019 from APRA reported 7.3% growth of funds with more than fund members over the previous 5 years and 7.9% growth over the previous ten years.

Details

Currently components of taxable income are individually inflated based on their historical trends in the ATO sample files, with the exception of:

inflated using `differentially_uprate_wage`**.** `Sw_amt`

inflated using `wage_inflator` `Alow_ben_amt`, `ETP_txbl_amt`, `Rptbl_Empr_spr_cont_amt`, `Non_emp_spr_amt`, `MCS_Emplr_Contr`, `MCS_Prsnl_Contr`, `MCS_Othr_Contr`

inflated using `cpi_inflator` `WRE_car_amt`, `WRE_trvl_amt`, `WRE_uniform_amt`, `WRE_self_amt`, `WRE_other_amt`

inflated by `lf_inflator_fy` `WEIGHT`

inflated by `CG_inflator` `Net_CG_amt`, `Tot_CY_CG_amt`

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

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. However, more recent data is of course preferable.

Value

A sample file with the same number of rows as `sample_file` but with inflated values as a forecast for the sample file in `to_fy`. If `WEIGHT` is not already a column of `sample_file`, it will be added and its sum will be the predicted number of taxpayers in `to_fy`.

Examples

```
# install_taxstats()
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 — *Simple projections of the annual 2% samples of Australian Taxation Office tax returns.*

Description

Simple projections of the annual 2% samples of Australian Taxation Office tax returns.

Usage

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

Arguments

`sample_file`	A `data.table` matching a 2% sample file from the ATO. See package `taxstats` for an example.
`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 with the same number of rows as `sample_file` but with inflated values as a forecast for the sample file in `to_fy`. If `WEIGHT` is not already a column of `sample_file`, it will be added and its sum will be the predicted number of taxpayers in `to_fy`.

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/

`require_taxstats` *Attach a 'taxstats' package*

Description

Used in lieu of simply `library(taxstats)` to handle cases where it is not installed, but should not be installed to the user's default library (as during CRAN checks).

Usage

```
require_taxstats()

require_taxstats1516()
```

Value

`TRUE`, invisibly, for success. Used for its side-effect: attaching the taxstats package.

`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. Last updated 2018-07-06.

Usage

```
residential_property_prices
```

Format

A `data.table` of three columns and 522 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. `https://www.abs.gov.au/statistics/economy/price-indexes-and-inflation/residential-property-price-indexes-eight-capital-cities/latest-release`.

`revenue_foregone` *Revenue foregone from a modelled sample file*

Description

Revenue foregone from a modelled sample file

Usage

```
revenue_foregone(dt, revenue_positive = TRUE, digits = NULL)
```

Arguments

`dt`	A `data.table` from `model_income_tax`.
`revenue_positive`	If `TRUE`, the default, tax increase (revenue) is positive and tax cuts are negative.
`digits`	If not `NULL`, affects the print method of the value.

`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",
  on_sapto_cd = "A",
  .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.
`on_sapto_cd`	SAPTO claim code type (for non-veterans). A letter A-E. A = single, B = lived apart due to illness and spouse was eligible, C = lived apart but spouse ineligible, D = lived together, both eligible for sapto, E = lived together, spouse ineligible. Only `"A"` and `"D"` are supported.
`.check`	Run checks for consistency of values. For example, ensuring no single individuals have positive `Spouse_income`.

set_offset *Set offsets*

Description

Create parameters for tax offsets.

Usage

```
set_offset(
  offset_1st = integer(1),
  thresholds = integer(),
  tapers = double(),
  refundable = logical(1)
)

set_offsets(
  ...,
  yr = NULL,
  lito_max_offset = NULL,
  lito_taper = NULL,
  lito_min_bracket = NULL,
  lito_multi = NULL
)

the_MAX_N_OFFSETN()
```

Arguments

`offset_1st`	`integer(1)` The offset available for zero income.
`thresholds`	`integer(N)` An sorted integer vector, the thresholds above which each taper applies.
`tapers`	`double(N)` The tapers above each `threshold`. Positive tapers mean that the offset reduces for every dollar above the corresponding threshold.
`refundable`	`bool(1)` If `FALSE`, the default, offsets are non-refundable, meaning that the offset cannot reduce the tax below zero.
`...`	A set of offsets created by `set_offset`. May not exceed `the_MAX_N_OFFSETN()`.
`yr`	`NULL` / `integer(1)` If `NULL`, only the offsets created by `...` are used. Otherwise, inherits offsets (such as LITO and LMITO) from the corresponding year.
`lito_max_offset`, `lito_taper`, `lito_min_bracket`, `lito_multi`	deprecated arguments to adjust (single-threshold) LITO.

Value

set_offset	A list of four elements, offset_1st, thresholds, tapers, refundable.
set_offsets	A list of lists created by set_offset.
the_MAX_N_OFFSETN	The maximum number of offsets that may be used.

---

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 2 million.

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-a`

`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 `https://github.com/HughParsonage/grattan/blob/master/data-raw/parlinfo/small-biz-explanatory-memo-2015.pdf` from the original `http://parlinfo.aph.gov.au/parlInfo/download/legislation/ems/r5494_ems_0a26ca86-9c3f-4ffa-9b8`

`System` *FUNCTION_TITLE*

Description

FUNCTION_DESCRIPTION

Usage

```
System(
  yr,
  ordinary_tax_thresholds = NULL,
  ordinary_tax_rates = NULL,
  medicare_levy_taper = NULL,
  medicare_levy_rate = NULL,
  medicare_levy_lower_threshold = NULL,
  medicare_levy_lower_sapto_threshold = NULL,
  medicare_levy_lower_family_threshold = NULL,
  medicare_levy_lower_family_sapto_threshold = NULL,
  medicare_levy_lower_up_for_each_child = NULL,
  medicare_levy_upper_sapto_threshold = NULL,
  medicare_levy_upper_family_threshold = NULL,
  medicare_levy_upper_family_sapto_threshold = NULL,
  medicare_levy_upper_threshold = NULL,
  Offsets = NULL,
  sapto_max_offset = NULL,
  sapto_lower_threshold = NULL,
  sapto_taper = NULL,
  sapto_max_offset_married = NULL,
  sapto_lower_threshold_married = NULL,
  sapto_taper_married = NULL,
  sapto_pension_age = NULL,
  fix = 0L
)
```

Arguments

`yr` `integer(1)` A year.

`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_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.
`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`.
`Offsets`	List of offsets created by `set_offsets`.
`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`.
`sapto_max_offset_married`, `sapto_lower_threshold_married`, `sapto_taper_married`	As above, but applied to members of a couple.
`sapto_pension_age`	The age at and above which the SAPTO is to apply.
`fix`	`integer(1)` If `0L`, the default, an error will be emitted if parameters are inconsistent; if `1L`, inconsistencies will be fixed.

Details

A list describing a tax system

validate_date *Verifying validity of dates*

Description

Many functions expect Dates. Determining that they are validly entered is often quite computationally costly, relative to the core calculations. These internal functions provide mechanisms to check validity quickly, while still providing clear, accurate error messages.

Usage

```
validate_date(date_to_verify, from = NULL, to = NULL, deparsed = "Date")
```

Arguments

`date_to_verify`	(character) A user-provided value, purporting to be character vector of dates.
`from, to`	Indicating the range of years valid for `date_to_verify`. Default set to -Inf and Inf respectively (i.e. there is no bound)
`deparsed`	The name of variable to appear in error messages.

Value

`date_to_verify` as a Date object, provided it can be converted to a Date and all elements are within the bounds `from` and `to`.

Examples

```
validate_date("2020-01-01")
```

`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 = NULL,
  to_fy = NULL,
  useABSConnection = FALSE,
  allow.projection = TRUE,
  forecast.series = c("mean", "upper", "lower", "custom"),
  forecast.level = 95,
  wage.series = NULL,
  accelerate.above = 100000L
)
```

Arguments

`wage`	The amount to be inflated (1 by default).
`from_fy, to_fy`	(character) a character vector with each element in the form "2012-13" representing the financial years between which the CPI inflator is desired. If both `from_fy` and `to_fy` are `NULL` (the default), `from_fy` is set to the previous financial year and `to_fy` to the current financial year, with a warning. Setting only one is an error.
`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. If the SDMX connection fails, a message is emitted (not a warning) and the function contines as if `useABSConnection = FALSE`. The internal data was updated on 2022-01-03 to 2021-Q2.

`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`.
`accelerate.above`	An integer setting the threshold for 'acceleration'. When the maximum length of the arguments exceeds this value, calculate each unique value individually then combine. Set to 100,000 as a rule of thumb beyond which calculation speeds benefit dramatically. Can be set to `Inf` to disable acceleration.

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)
```

Index