Package 'vital'

June 4, 2024

Version 1.0.0

Type Package

Title Tidy Analysis Tools for Mortality, Fertility, Migration and Population Data

Description Analysing vital statistics based on tools consistent with the tidyverse. Tools are provided for data visualization, lifetable calculations, computing net migration numbers, Lee-Carter modelling; functional data modelling and forecasting.

Depends R (>= 4.1.0)

Imports cobs, distributional, dplyr, fable, fabletools (>= 0.3.3), future.apply, generics, ggplot2, HMDHFDplus (>= 2.0.3), mgcv, patchwork, progressr, purrr, rlang, tibble, tidyr, tidyselect, tsibble, vctrs

Suggests demography, feasts, testthat (>= 3.0.0)

License GPL-3

URL https://pkg.robjhyndman.com/vital/, https://github.com/robjhyndman/vital

Encoding UTF-8

LazyData true

RoxygenNote 7.3.1

BugReports https://github.com/robjhyndman/vital/issues

NeedsCompilation no

Author Rob Hyndman [aut, cre, cph] (<https://orcid.org/0000-0002-2140-5352>), Mitchell O'Hara-Wild [ctb] (<https://orcid.org/0000-0001-6729-7695>)

Maintainer Rob Hyndman <Rob.Hyndman@monash.edu>

Repository CRAN

Date/Publication 2024-06-04 16:10:02 UTC

Contents

`age_components` *Extract age components from a model*

Description

For a mable with a single model column, return the model components that are indexed by age.

Usage

```
age_components(object, ...)
```

Arguments

object	A vital mable object with a single model column.
...	Not currently used.

Value

vital object containing the age components from the model.

Examples

```
aus_mortality |>
  dplyr::filter(State == "Victoria", Sex == "female") |>
  model(lee_carter = LC(log(Mortality))) |>
  age_components()
```

---

as_vital *Coerce to a vital object*

Description

A vital object is a type of tsibble that contains vital statistics such as births, deaths, and population counts, and mortality and fertility rates. It is a tsibble with a special class that allows for special methods to be used. The object has an attribute that stores variables names needed for some functions, including age, sex, births, deaths and population.

Usage

```
as_vital(x, ...)

## S3 method for class 'demogdata'
as_vital(x, sex_groups = TRUE, ...)

## S3 method for class 'tbl_ts'
as_vital(
  x,
  .age = NULL,
  .sex = NULL,
  .deaths = NULL,
  .births = NULL,
  .population = NULL,
  reorder = FALSE,
  ...
)

## S3 method for class 'data.frame'
as_vital(
```

```
  x,
  key = NULL,
  index,
  .age = NULL,
  .sex = NULL,
  .deaths = NULL,
  .births = NULL,
  .population = NULL,
  reorder = TRUE,
  ...
)
```

Arguments

x	Object to be coerced to a vital format.
...	Other arguments passed to `tsibble::as_tsibble()`
sex_groups	Logical variable indicating if the groups denote sexes
.age	Character string with name of age variable
.sex	Character string with name of sex variable
.deaths	Character string with name of deaths variable
.births	Character string with name of births variable
.population	Character string with name of population variable
reorder	Logical indicating if the variables should be reordered.
key	Variable(s) that uniquely determine time indices. NULL for empty key, and `c()` for multiple variables. It works with tidy selector (e.g. `tidyselect::starts_with()`).
index	A variable to specify the time index variable.

Value

A tsibble with class `vital`.

Author(s)

Rob J Hyndman

See Also

`tsibble::tsibble()`

Examples

```
# coerce demogdata object to vital
as_vital(demography::fr.mort)

# create a vital with only age as a key
tibble::tibble(
  year = rep(2010:2015, 100),
```

```
  age = rep(0:99, each = 6),
  mx = runif(600, 0, 1)
) |>
  as_vital(
    index = year,
    key = age,
    .age = "age"
  )
```

`aus_fertility` *Australian fertility data*

Description

`aus_fertility` is an annual `vital` object covering the years 1921-2002 with three values:

Fertility:	Fertility rate per woman
Exposure:	Population of women at 30 June each year
Births:	Number of births

Format

Time series of class `vital`

Details

The data is disaggregated using one key:

Age:	Age of mother at time of birth

The extreme age groups (15 and 49) also include a few younger and older mothers respectively.

Source

Australian Human Mortality Database. https://aushd.org

Examples

```
library(ggplot2)
aus_fertility
aus_fertility |>
  autoplot(Fertility) +
  ylab("Fertility rate")
```

`aus_mortality` *Australian mortality data*

Description

`aus_mortality` is an annual `vital` with three values:

Mortality:	Mortality rate
Exposure:	Population at 30 June each year
Deaths:	Number of deaths

Format

Time series of class `vital`

Details

The data is disaggregated using four keys:

Age:	Age at death
Sex:	male or female
State:	State of Australia
Code:	Short code for state

The age group 100 also includes people who died aged older than 100. The data up to 1970 were taken from the Australian Demographic Data Bank (https://pkg.robjhyndman.com/addb/). From 1971, the data come from the Australian Human Mortality Database (https://aushd.org). There may be some discontinuities introduced due to different methods being used to prepare the data before and after 1971. Note that "ACTOT" includes both the ACT and overseas territories and is only available up to 2003. The data exclusively from the ACT begins in 1971.

Source

Australian Human Mortality Database

Examples

```
library(ggplot2)
aus_mortality
aus_mortality |>
  dplyr::filter(State=="Victoria", Sex != "total") |>
  autoplot(Exposure) +
  ylab("Population at 30 June (thousands)")
```

autoplot.fbl_vtl_ts *Plot forecasts from a vital model*

Description

Produces a plot showing forecasts obtained from a model applied to a vital object.

Usage

```
## S3 method for class 'fbl_vtl_ts'
autoplot(object, ...)
```

Arguments

object	A fable object obtained from a vital model.
...	Further arguments ignored.

Value

A ggplot2 object.

Author(s)

Rob J Hyndman

Examples

```
library(ggplot2)
aus_mortality |>
 dplyr::filter(State == "Victoria") |>
 model(ave = FMEAN(Mortality)) |>
 forecast(h = 10) |>
 autoplot() + scale_y_log10()
```

autoplot.mdl_vtl_df *Plot output from a vital model*

Description

Produces a plot showing a model applied to a vital object. This can be applied to one type of model only. So use select() to choose the model column to plot. If there are multiple keys, separate models will be identified by colour.

Usage

```
## S3 method for class 'mdl_vtl_df'
autoplot(object, ...)
```

Arguments

object	A mable object obtained from a vital.
...	Further arguments ignored.

Value

A ggplot2 object.

Author(s)

Rob J Hyndman

Examples

```
library(ggplot2)
aus_mortality |>
 dplyr::filter(State == "Victoria") |>
 model(ave = FMEAN(Mortality)) |>
 autoplot() + scale_y_log10()
```

---

autoplot.vital *Rainbow plot of demographic data against age*

Description

Produce rainbow plot (coloured by time index) of demographic variable against against age.

Usage

```
## S3 method for class 'vital'
autoplot(object, .vars = NULL, age = attributes(object)$agevar, ...)
```

Arguments

object	A vital including an age variable and the variable you wish to plot.
.vars	The name of the variable you wish to plot.
age	The name of the age variable. If not supplied, the function will attempt to find it.
...	Further arguments not used.

Value

A ggplot2 object.

Author(s)

Rob J Hyndman

References

Hyndman, Rob J & Shang, Han Lin (2010) Rainbow plots, bagplots, and boxplots for functional data. *Journal of Computational and Graphical Statistics*, 19(1), 29-45. https://robjhyndman.com/publications/rainbow-fda/

Examples

```
autoplot(aus_fertility, Fertility)
```

`collapse_ages` *Collapse upper ages into a single age group. Counts are summed while rates are recomputed where possible.*

Description

Collapse upper ages into a single age group. Counts are summed while rates are recomputed where possible.

Usage

```
collapse_ages(.data, max_age = 100)
```

Arguments

`.data`	A vital object including an age variable
`max_age`	Maximum age to include in the collapsed age group.

Details

If the object includes deaths, population and mortality rates, then deaths and population are summed and mortality rates are recomputed as deaths/population. But if the object contains mortality rates but not deaths and population, then the last rate remains unchanged (and a warning is generated).

Value

A vital object with the same variables as `.data`, but with the upper ages collapsed into a single age group.

Author(s)

Rob J Hyndman

Examples

```
aus_mortality |>
 dplyr::filter(State == "Victoria", Sex == "female") |>
 collapse_ages(max_age = 85)
```

FDM *Functional data model*

Description

Functional data model of mortality or fertility rates as a function of age. `FDM()` returns a functional data model applied to the formula's response variable as a function of age.

Usage

```
FDM(formula, order = 6, ts_model_fn = fable::ARIMA, coherent = FALSE, ...)
```

Arguments

`formula`	Model specification.
`order`	Number of principal components to fit.
`ts_model_fn`	Univariate time series modelling function for the coefficients. Any model that works with the fable package is ok. Default is `fable::ARIMA()`.
`coherent`	If TRUE, fitted models are stationary, other than for the case of a key variable taking the value `geometric_mean`. This is designed to work with vitals produced using `make_pr()`. Default is FALSE. It only works when `ts_model_fn` is `ARIMA()`.
`...`	Not used.

Value

A model specification.

Author(s)

Rob J Hyndman

References

Hyndman, R. J., and Ullah, S. (2007) Robust forecasting of mortality and fertility rates: a functional data approach. *Computational Statistics & Data Analysis*, 5, 4942-4956. https://robjhyndman.com/publications/funcfor/ Hyndman, R. J., Booth, H., & Yasmeen, F. (2013). Coherent mortality forecasting: the product-ratio method with functional time series models. *Demography*, 50(1), 261-283. https://robjhyndman.com/publications/coherentfdm/

Examples

```
hu <- norway_mortality |>
  dplyr::filter(Sex == "Female", Year > 2010) |>
  smooth_mortality(Mortality) |>
  model(hyndman_ullah = FDM(log(.smooth)))
report(hu)
autoplot(hu)
```

FMEAN *Functional mean model*

Description

`FMEAN()` returns an iid functional model applied to the formula's response variable as a function of age.

Usage

```
FMEAN(formula, ...)
```

Arguments

`formula`	Model specification.
`...`	Not used.

Value

A model specification.

Author(s)

Rob J Hyndman

Examples

```
fmean <- aus_mortality |>
  dplyr::filter(State == "Victoria", Sex == "female") |>
  model(mean = FMEAN(Mortality))
report(fmean)
autoplot(fmean) + ggplot2::scale_y_log10()
```

FNAIVE *Functional naive model*

Description

FNAIVE() returns an random walk functional model applied to the formula's response variable as a function of age.

Usage

```
FNAIVE(formula, ...)
```

Arguments

formula	Model specification.
...	Not used.

Value

A model specification.

Author(s)

Rob J Hyndman

Examples

```
fnaive <- aus_mortality |>
  dplyr::filter(State == "Victoria", Sex == "female") |>
  model(fit = FNAIVE(Mortality))
report(fnaive)
autoplot(fnaive) + ggplot2::scale_y_log10()
```

forecast.FDM *Produce forecasts from a vital model*

Description

The forecast function allows you to produce future predictions of a vital model, where the response is a function of age. The forecasts returned contain both point forecasts and their distribution.

Usage

```
## S3 method for class 'FDM'
forecast(
  object,
  new_data = NULL,
  h = NULL,
  point_forecast = list(.mean = mean),
  simulate = FALSE,
  bootstrap = FALSE,
  times = 5000,
  ...
)

## S3 method for class 'LC'
forecast(
  object,
  new_data = NULL,
  h = NULL,
  point_forecast = list(.mean = mean),
  simulate = FALSE,
  bootstrap = FALSE,
  times = 5000,
  ...
)

## S3 method for class 'FMEAN'
forecast(
  object,
  new_data = NULL,
  h = NULL,
  point_forecast = list(.mean = mean),
  simulate = FALSE,
  bootstrap = FALSE,
  times = 5000,
  ...
)

## S3 method for class 'FNAIVE'
forecast(
  object,
  new_data = NULL,
  h = NULL,
  point_forecast = list(.mean = mean),
  simulate = FALSE,
  bootstrap = FALSE,
  times = 5000,
  ...
)
```

```
## S3 method for class 'mdl_vtl_df'
forecast(
  object,
  new_data = NULL,
  h = NULL,
  point_forecast = list(.mean = mean),
  simulate = FALSE,
  bootstrap = FALSE,
  times = 5000,
  ...
)
```

Arguments

object	A mable containing one or more models.
new_data	A tsibble containing future information used to forecast.
h	Number of time steps ahead to forecast. This can be used instead of new_data when there are no covariates in the model. It is ignored if new_data is provided.
point_forecast	A list of functions used to compute point forecasts from the forecast distribution.
simulate	If TRUE, then forecast distributions are computed using simulation from a parametric model.
bootstrap	If TRUE, then forecast distributions are computed using simulation with resampling.
times	The number of sample paths to use in estimating the forecast distribution when bootstrap = TRUE.
...	Additional arguments passed to the specific model method.

Value

A fable containing the following columns:

- .model: The name of the model used to obtain the forecast. Taken from the column names of models in the provided mable.
- The forecast distribution. The name of this column will be the same as the dependent variable in the model(s). If multiple dependent variables exist, it will be named .distribution.
- Point forecasts computed from the distribution using the functions in the point_forecast argument.
- All columns in new_data, excluding those whose names conflict with the above.

Author(s)

Rob J Hyndman and Mitchell O'Hara-Wild

Examples

```
aus_mortality |>
 dplyr::filter(State == "Victoria", Sex == "female") |>
 model(naive = FNAIVE(Mortality)) |>
 forecast(h = 10)
```

`generate.mdl_vtl_df` *Generate responses from a mable*

Description

Use a fitted model to simulate future data with similar behaviour to the response.

Usage

```
## S3 method for class 'mdl_vtl_df'
generate(x, new_data = NULL, h = NULL, bootstrap = FALSE, times = 1, ...)
```

Arguments

`x`	A mable.
`new_data`	Future data needed for generation (should include the time index and exogenous regressors)
`h`	The simulation horizon (can be used instead of `new_data` for regular time series with no exogenous regressors).
`bootstrap`	If `TRUE`, then forecast distributions are computed using simulation with resampled errors.
`times`	The number of replications.
`...`	Additional arguments

Details

Innovations are sampled by the model's assumed error distribution. If `bootstrap` is `TRUE`, innovations will be sampled from the model's residuals.

Value

A vital object with simulated values.

Author(s)

Rob J Hyndman and Mitchell O'Hara-Wild

Examples

```
aus_mortality |>
  dplyr::filter(State == "Victoria") |>
  model(lc = LC(Mortality)) |>
  generate(times = 3, bootstrap = TRUE)
```

`interpolate.mdl_vtl_df` *Interpolate missing values using a vital model*

Description

Uses a fitted vital model to interpolate missing values from a dataset.

Usage

```
## S3 method for class 'mdl_vtl_df'
interpolate(object, new_data, ...)
```

Arguments

`object`	A mable containing a single model column.
`new_data`	A dataset with the same structure as the data used to fit the model.
`...`	Other arguments passed to interpolate methods.

Value

A vital object with missing values interpolated.

Author(s)

Rob J Hyndman

Examples

```
act_female <- aus_mortality |>
 dplyr::filter(Code == "ACTOT", Sex == "female")
act_female |>
 model(mean = FMEAN(Mortality)) |>
 interpolate(act_female)
```

LC *Lee-Carter model*

Description

Lee-Carter model of mortality or fertility rates. `LC()` returns a Lee-Carter model applied to the formula's response variable as a function of age. This produces a standard Lee-Carter model by default, although many other options are available. Missing rates are set to the geometric mean rate for the relevant age.

Usage

```
LC(
  formula,
  adjust = c("dt", "dxt", "e0", "none"),
  jump_choice = c("fit", "actual"),
  scale = FALSE,
  ...
)
```

Arguments

`formula`	Model specification. It should include the log of the variable to be modelled. See the examples.
`adjust`	method to use for adjustment of coefficients k_t. Possibilities are `"dt"` (Lee-Carter method, the default), `"dxt"` (BMS method), `"e0"` (Lee-Miller method based on life expectancy) and `"none"`.
`jump_choice`	Method used for computation of jump-off point for forecasts. Possibilities: `"actual"` (use actual rates from final year) and `"fit"` (use fitted rates). The original Lee-Carter method used `"fit"` (the default), but Lee and Miller (2001) and most other authors prefer `"actual"`.
`scale`	If TRUE, `bx` and `kt` are rescaled so that `kt` has drift parameter = 1.
`...`	Not used.

Value

A model specification.

Author(s)

Rob J Hyndman

References

Basellini, U, Camarda, C G, and Booth, H (2022) Thirty years on: A review of the Lee-Carter method for forecasting mortality. *International Journal of Forecasting*, 39(3), 1033-1049.

Booth, H., Maindonald, J., and Smith, L. (2002) Applying Lee-Carter under conditions of variable mortality decline. *Population Studies*, **56**, 325-336.

Lee, R D, and Carter, L R (1992) Modeling and forecasting US mortality. *Journal of the American Statistical Association*, 87, 659-671.

Lee R D, and Miller T (2001). Evaluating the performance of the Lee-Carter method for forecasting mortality. *Demography*, 38(4), 537–549.

Examples

```
lc <- aus_mortality |>
  dplyr::filter(State == "Victoria", Sex == "female") |>
  model(lee_carter = LC(log(Mortality)))
report(lc)
autoplot(lc)
```

---

life_expectancy *Compute life expectancy from age-specific mortality rates*

Description

Returns remaining life expectancy at a given age (0 by default).

Usage

```
life_expectancy(.data, from_age = 0, mortality)
```

Arguments

`.data`	A vital object including an age variable and a variable containing mortality rates.
`from_age`	Age at which life expectancy to be calculated. Either a scalar or a vector of ages.
`mortality`	Variable in `.data` containing Mortality rates (mx). If omitted, the variable with name `mx`, `Mortality` or `Rate` will be used (not case sensitive).

Value

A `vital` object with life expectancy in column `ex`.

Author(s)

Rob J Hyndman

References

Chiang CL. (1984) *The life table and its applications*. Robert E Krieger Publishing Company: Malabar.

Keyfitz, N, and Caswell, H. (2005) *Applied Mathematical Demography*, Springer-Verlag: New York.

Preston, S.H., Heuveline, P., and Guillot, M. (2001) *Demography: measuring and modeling population processes*. Blackwell

See Also

`life_table()`

Examples

```
# Compute Victorian life expectancy for females over time
aus_mortality |>
  dplyr::filter(Code == "VIC", Sex == "female") |>
  life_expectancy()
```

`life_table` *Compute period lifetables from age-specific mortality rates*

Description

All available years and ages are included in the tables. $qx = mx/(1 + ((1-ax) * mx))$ as per Chiang (1984). Warning: the code has only been tested for data based on single-year age groups.

Usage

```
life_table(.data, mortality)
```

Arguments

`.data`	A `vital` including an age variable and a variable containing mortality rates.
`mortality`	Variable in `.data` containing Mortality rates (mx). If omitted, the variable with name `mx`, `Mortality` or `Rate` will be used (not case sensitive).

Value

A vital object containing the index, keys, and the new life table variables `mx`, `qx`, `lx`, `dx`, `Lx`, `Tx` and `ex`.

Author(s)

Rob J Hyndman

References

Chiang CL. (1984) *The life table and its applications*. Robert E Krieger Publishing Company: Malabar.

Keyfitz, N, and Caswell, H. (2005) *Applied mathematical demography*, Springer-Verlag: New York.

Preston, S.H., Heuveline, P., and Guillot, M. (2001) *Demography: measuring and modeling population processes*. Blackwell

Examples

```
# Compute Victorian life table for females in 2003
aus_mortality |>
  dplyr::filter(Code == "VIC", Sex == "female", Year == 2003) |>
  life_table()
```

`make_pr` *Do a product/ratio transformation*

Description

Make a new vital containing products and ratios of a measured variable by a key variable. The most common use case of this function is for mortality rates by sex. That is, we want to compute the geometric mean of age-specific mortality rates, along with the ratio of mortality to the geometric mean for each sex. The latter are equal to the male/female and female/male ratios of mortality rates.

Usage

```
make_pr(.data, .var, key = Sex)
```

Arguments

`.data`	A vital object
`.var`	A bare variable name of the measured variable to use.
`key`	A bare variable name specifying the key variable to use.

Details

When a measured variable takes value 0, it is set to 10^-6 to avoid infinite values in the ratio.

Value

A vital object

References

Hyndman, R.J., Booth, H., & Yasmeen, F. (2013). Coherent mortality forecasting: the product-ratio method with functional time series models. *Demography*, 50(1), 261-283.

Examples

```
pr <- aus_mortality |>
  dplyr::filter(Year > 2015, Sex != "total") |>
  make_pr(Mortality)
pr |>
  dplyr::filter(Sex == "geometric_mean", Code == "VIC") |>
  autoplot(Mortality) +
  ggplot2::scale_y_log10()
```

---

make_sd *Do a sum/difference transformation*

Description

Make a new vital containing means and differences of a measured variable by a key variable. The most common use case of this function is for migration numbers by sex. That is, we want to compute the age-specific mean migration, along with the difference of migration to the mean for each sex. The latter are equal to half the male/female and female/male differences of migration numbers.

Usage

```
make_sd(.data, .var, key = Sex)
```

Arguments

.data	A vital object
.var	A bare variable name of the measured variable to use.
key	A bare variable name specifying the key variable to use.

Value

A vital object

References

Hyndman, R.J., Booth, H., & Yasmeen, F. (2013). Coherent mortality forecasting: the product-ratio method with functional time series models. *Demography*, 50(1), 261-283.

Examples

```
mig <- net_migration(norway_mortality, norway_births) |>
  dplyr::filter(Sex != "Total")
sd <- mig |>
  make_sd(NetMigration)
sd |>
  autoplot(NetMigration)
```

`model.vital` *Estimate models for vital data*

Description

Trains specified model definition(s) on a dataset. This function will estimate the a set of model definitions (passed via `...`) to each series within `.data` (as identified by the key structure). The result will be a mable (a model table), which neatly stores the estimated models in a tabular structure. Rows of the data identify different series within the data, and each model column contains all models from that model definition. Each cell in the mable identifies a single model.

Usage

```
## S3 method for class 'vital'
model(.data, ..., .safely = TRUE)
```

Arguments

`.data`	A vital object including an age variable.
`...`	Definitions for the models to be used. All models must share the same response variable.
`.safely`	If a model encounters an error, rather than aborting the process a NULL model will be returned instead. This allows for an error to occur when computing many models, without losing the results of the successful models.

Value

A mable containing the fitted models.

Parallel

It is possible to estimate models in parallel using the future package. By specifying a `future::plan()` before estimating the models, they will be computed according to that plan.

Progress

Progress on model estimation can be obtained by wrapping the code with `progressr::with_progress()`. Further customisation on how progress is reported can be controlled using the `progressr` package.

Author(s)

Rob J Hyndman and Mitchell O'Hara-Wild

Examples

```
aus_mortality |>
  dplyr::filter(State == "Victoria", Sex == "female") |>
  model(
    naive = FNAIVE(Mortality),
    mean = FMEAN(Mortality)
  )
```

`net_migration` *Calculate net migration from a vital object*

Description

Calculate net migration from a vital object

Usage

```
net_migration(deaths, births)
```

Arguments

`deaths`	A vital object containing at least a time index, age, population at 1 January, and death rates.
`births`	A vital object containing at least a time index and number of births per time period. It is assumed that the population variable is the same as in the deaths object, and that the same keys other than age are present in both objects.

Value

A vital object containing population, estimated deaths (not actual deaths) and net migration, using the formula Net Migration = Population - lag(Population cohort) - Deaths + Births. Births are returned as Population at Age -1, and deaths are estimated from the life table

References

Hyndman and Booth (2008) Stochastic population forecasts using functional data models for mortality, fertility and migration. *International Journal of Forecasting*, 24(3), 323-342.

Examples

```
net_migration(norway_mortality, norway_births)
## Not run:
# Files downloaded from the [Human Mortality Database](https://mortality.org)
deaths <- read_hmd_files(c("Population.txt", "Mx_1x1.txt"))
births <- read_hmd_file("Births.txt")
mig <- net_migration(deaths, births)

## End(Not run)
```

`norway_births` *Norwegian mortality and births data*

Description

`norway_births` is an annual `vital` object covering the years 1846-2022, as provided by the Human Mortality Database on 21 April 2024.

`norway_fertality` is an annual `vital` covering the years 1967-2022, as provided by the Human Fertility Database on 21 April 2024.

`norway_mortality` is an annual `vital` covering the years 1846-2022, as provided by the Human Mortality Database on 21 April 2024.

Format

Time series of class `vital`

Source

Human Mortality Database https://mortality.org

Human Fertility Database https://www.humanfertility.org

Examples

```
library(ggplot2)
# Births
norway_births
norway_births |>
  autoplot(Births)
# Deaths
norway_mortality
norway_mortality |>
  dplyr::filter(Age < 85, Year < 1900, Sex != "Total") |>
  autoplot(Mortality) +
  scale_y_log10()
# Fertility
norway_fertility
norway_fertility |>
  autoplot(Fertility)
```

read_hfd *Read data directly from HFD and construct a* vital *object for use in other functions*

Description

read_hfd reads single-year and single-age data from the Human Fertility Database (HFD https://www.humanfertility.org) and constructs a vital object suitable for use in other functions. This function uses HMDHFDplus::readHFDweb() to download the required data. It is designed to handle age-specific fertility rates. It may be extended to handle other types of data in the future.

Usage

```
read_hfd(country, username, password, variables = "asfrRR")
```

Arguments

country	Directory abbreviation from the HMD. For instance, Norway = "NOR".
username	HFD username (case-sensitive)
password	HFD password (case-sensitive)
variables	List of variables to download from the HFD. By default, the age-specific fertility rate (asfrRR) is downloaded.

Details

In order to read the data, users are required to create an account with the HFD website (https://www.humanfertility.org), and obtain a valid username and password.

Value

read_hfd returns a vital object combining the downloaded data.

Author(s)

Rob J Hyndman

Examples

```
## Not run:
norway <- read_hfd(
  country = "NOR",
  username = "Nora.Weigh@mymail.com",
  password = "FF!5xeEFa6"
)

## End(Not run)
```

read_hfd_files *Read data from files downloaded from HFD and construct a* vital *object for use in other functions*

Description

read_hfd_files reads single-year and single-age data from files downloaded from the Human Mortality Database (HFD https://www.humanfertility.org) and constructs a vital object suitable for use in other functions. This function uses HMDHFDplus::readHFD() to parse the files.

Usage

```
read_hfd_files(files)
```

Arguments

files	Vector of file names containing data downloaded from the HFD. The file names are used to determine what they contain. If the file names are as per the HFD, then the function will automatically determine the contents. If it is unclear what a file contains, the columns will be named according to the filename. If the data contains a mixture of age-specific and non-age-specific variables, then the non-age-specific data will be repeated for each age. If you have HMD files for many countries, all with the same names, then you should put them in separate folders to avoid confusion, and to save changing all the filenames.

Value

read_hfd_files returns a vital object combining the downloaded data.

Author(s)

Rob J Hyndman

Examples

```
## Not run:
# File downloaded from the [Human Fertility Database](https://www.humanfertility.org)
fertility <- read_hfd_files("NORasfrRR.txt")

## End(Not run)
```

read_hmd *Read data directly from HMD and construct a* vital *object for use in other functions*

Description

read_hmd reads single-year and single-age data from the Human Mortality Database (HMD https://www.mortality.org) and constructs a vital object suitable for use in other functions. This function uses HMDHFDplus::readHMDweb() to download the required data. It is designed to handle Deaths, Population, Exposure, Death Rates and Births. By default, Deaths, Population, Exposure and Death Rates are downloaded. It is better to handle Births separately as they are not age-specific.

Usage

```
read_hmd(
  country,
  username,
  password,
  variables = c("Deaths", "Exposures", "Population", "Mx")
)
```

Arguments

country	Directory abbreviation from the HMD. For instance, Australia = "AUS".
username	HMD username (case-sensitive)
password	HMD password (case-sensitive)
variables	List of variables to download from the HMD. If the data contains a mixture of age-specific and non-age-specific variables, then the non-age-specific data will be repeated for each age.

Details

In order to read the data, users are required to create an account with the HMD website (https://www.mortality.org), and obtain a valid username and password.

Value

read_hmd returns a vital object combining the downloaded data.

Author(s)

Rob J Hyndman

Examples

```
## Not run:
norway <- read_hmd(
  country = "NOR",
  username = "Nora.Weigh@mymail.com",
  password = "FF!5xeEFa6"
)
norway_births <- read_hmd(
  country = "NOR",
  username = "Nora.Weigh@mymail.com",
  password = "FF!5xeEFa6",
  variables = "Births"
)

## End(Not run)
```

`read_hmd_files` *Read data from files downloaded from HMD and construct a* `vital` *object for use in other functions*

Description

`read_hmd_files` reads single-year and single-age data from files downloaded from the Human Mortality Database (HMD https://www.mortality.org) and constructs a `vital` object suitable for use in other functions. This function uses `HMDHFDplus::readHMD()` to parse the files.

Usage

```
read_hmd_files(files)
```

Arguments

`files`	Vector of file names containing data downloaded from the HMD. The file names are used to determine what they contain. If the file names are as per the HMD, then the function will automatically determine the contents. If it is unclear what a file contains, the columns will be named according to the filename. If the data contains a mixture of age-specific and non-age-specific variables, then the non-age-specific data will be repeated for each age. If you have HMD files for many countries, all with the same names, then you should put them in separate folders to avoid confusion, and to save changing all the filenames.

Value

`read_hmd_files` returns a `vital` object combining the downloaded data.

Author(s)

Rob J Hyndman

Examples

```
## Not run:
# Files downloaded from the [Human Mortality Database](https://mortality.org)
mortality <- read_hmd_files(
  c("Deaths_1x1.txt", "Exposures_1x1.txt", "Population.txt", "Mx_1x1.txt")
)
births <- read_hmd_files("Births.txt")

## End(Not run)
```

`smooth_spline` *Functions to smooth demographic data*

Description

These smoothing functions allow smoothing of a variable in a vital object. The vital object is returned along with some additional columns containing information about the smoothed variable: usually `.smooth` containing the smoothed values, and `.smooth_se` containing the corresponding standard errors.

Usage

```
smooth_spline(.data, .var, age_spacing = 1, k = -1)

smooth_mortality(.data, .var, age_spacing = 1, b = 65, power = 0.4, k = 30)

smooth_fertility(.data, .var, age_spacing = 1, lambda = 1e-10)

smooth_loess(.data, .var, age_spacing = 1, span = 0.2)
```

Arguments

`.data`	A vital object
`.var`	name of variable to smooth
`age_spacing`	Spacing between ages for smoothed vital. Default is 1.
`k`	Number of knots to use for penalized regression spline estimate.
`b`	Lower age for monotonicity. Above this, the smooth curve is assumed to be monotonically increasing.
`power`	Power transformation for age variable before smoothing. Default is 0.4 (for mortality data).
`lambda`	Penalty for constrained regression spline.
`span`	Span for loess smooth.

Details

`smooth_mortality()` use penalized regression splines applied to log mortality with a monotonicity constraint above age b. The methodology is based on Wood (1994). `smooth_fertility()` uses weighted regression B-splines with a concavity constraint, based on He and Ng (1999). The function `smooth_loess()` uses locally quadratic regression, while `smooth_spline()` uses penalized regression splines.

Value

vital with added columns containing smoothed values and their standard errors

Author(s)

Rob J Hyndman

References

Hyndman, R.J., and Ullah, S. (2007) Robust forecasting of mortality and fertility rates: a functional data approach. *Computational Statistics & Data Analysis*, 51, 4942-4956. https://robjhyndman.com/publications/funcfor/

Examples

```
library(dplyr)
aus_mortality |>
filter(State == "Victoria", Sex == "female", Year > 2000) |>
smooth_mortality(Mortality)
aus_fertility |>
 filter(Year > 2000) |>
smooth_fertility(Fertility)
```

`time_components` *Extract time components from a model*

Description

For a mable with a single model column, return the model components that are indexed by time.

Usage

```
time_components(object, ...)
```

Arguments

`object`	A vital mable object with a single model column.
`...`	Not currently used.

Value

tsibble object containing the time components from the model.

Examples

```
aus_mortality |>
  dplyr::filter(State == "Victoria", Sex == "female") |>
  model(lee_carter = LC(log(Mortality))) |>
  time_components()
```

---

total_fertility_rate *Compute total fertility rate from age-specific fertility rates*

Description

Total fertility rate is the expected number of babies per woman in a life-time given the fertility rate at each age of a woman's life.

Usage

```
total_fertility_rate(.data, fertility)
```

Arguments

`.data`	A vital object including an age variable and a variable containing fertility rates.
`fertility`	Variable in `.data` containing fertility rates. If omitted, the variable with name `fx`, `Fertility` or `Rate` will be used (not case sensitive).

Value

A vital object with total fertility in column `tfr`.

Author(s)

Rob J Hyndman

Examples

```
# Compute Australian total fertility rates over time
aus_fertility |>
  total_fertility_rate()
```

undo_pr *Undo a product/ratio transformation*

Description

Make a new vital from products and ratios of a measured variable by a key variable. The most common use case of this function is for computing mortality rates by sex, from the sex ratios and geometric mean of the rates.

Usage

```
undo_pr(.data, .var, key = Sex, times = 2000)
```

Arguments

.data	A vital object
.var	A bare variable name of the measured variable to use.
key	A bare variable name specifying the key variable to use. This key variable must include the value geometric_mean.
times	When the variable is a distribution, the product must be computed by simulation. This argument specifies the number of simulations to use.

Details

Note that when a measured variable takes value 0, the geometric mean is set to 10^-6 to avoid infinite values in the ratio. Therefore, when the transformation is undone, the results will not be identical to the original in the case that the original data was 0.

Value

A vital object

References

Hyndman, R.J., Booth, H., & Yasmeen, F. (2013). Coherent mortality forecasting: the product-ratio method with functional time series models. *Demography*, 50(1), 261-283.

Examples

```
# Make products and ratios
orig_data <- aus_mortality |>
  dplyr::filter(Year > 2015, Sex != "total", Code == "NSW")
pr <- orig_data |>
  make_pr(Mortality)
# Compare original data with product/ratio version
orig_data
pr
# Undo products and ratios
pr |> undo_pr(Mortality)
```

undo_sd *Undo a mean/difference transformation*

Description

Make a new vital from means and differences of a measured variable by a key variable. The most common use case of this function is for computing migration numbers by sex, from the sex differences and mean of the numbers.

Usage

```
undo_sd(.data, .var, key = Sex, times = 2000)
```

Arguments

.data	A vital object
.var	A bare variable name of the measured variable to use.
key	A bare variable name specifying the key variable to use. This key variable must include the value geometric_mean.
times	When the variable is a distribution, the product must be computed by simulation. This argument specifies the number of simulations to use.

Value

A vital object

References

Hyndman, R.J., Booth, H., & Yasmeen, F. (2013). Coherent mortality forecasting: the product-ratio method with functional time series models. *Demography*, 50(1), 261-283.

Examples

```
# Make sums and differences
mig <- net_migration(norway_mortality, norway_births) |>
  dplyr::filter(Sex != "Total")
sd <- mig |>
  make_sd(NetMigration)
# Undo products and ratios
sd |> undo_sd(NetMigration)
```

vital *Create a vital object*

Description

A vital object is a type of tsibble that contains vital statistics such as births, deaths, and population counts, and mortality and fertility rates. It is a tsibble with a special class that allows for special methods to be used. The object has an attribute that stores variables names needed for some functions, including age, sex, births, deaths and population.

Usage

```
vital(
  ...,
  key = NULL,
  index,
  .age = NULL,
  .sex = NULL,
  .deaths = NULL,
  .births = NULL,
  .population = NULL,
  regular = TRUE,
  .drop = TRUE
)
```

Arguments

`...`	A set of name-value pairs
`key`	Variable(s) that uniquely determine time indices. NULL for empty key, and `c()` for multiple variables. It works with tidy selector (e.g. `tidyselect::starts_with()`)
`index`	A variable to specify the time index variable.
`.age`	Character string with name of age variable
`.sex`	Character string with name of sex variable
`.deaths`	Character string with name of deaths variable
`.births`	Character string with name of births variable
`.population`	Character string with name of population variable
`regular`	Regular time interval (`TRUE`) or irregular (`FALSE`). The interval is determined by the greatest common divisor of index column, if `TRUE`.
`.drop`	If `TRUE`, empty key groups are dropped.

Value

A tsibble with class `vital`.

Author(s)

Rob J Hyndman

See Also

tsibble::tsibble()

Examples

```
# create a vital with only age as a key
vital(
  year = rep(2010:2015, 100),
  age = rep(0:99, each = 6),
  mx = runif(600, 0, 1),
  index = year,
  key = age,
  .age = "age"
)
```

Index