Package 'EpiForsk'

February 10, 2023

Title Code Sharing at the Department of Epidemiological Research at Statens Serum Institut

Version 0.0.1

Description This is a collection of assorted functions and examples collected from various projects. Currently we have functionalities for simplifying overlapping time intervals, Charlson comorbidity score constructors for Danish data, sibling design linear regression functionalities, and a method for calculating the confidence intervals for functions of parameters from a GLM.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.2.3

Imports cowplot, data.table, doParallel, dplyr, foreach, ggplot2, glue, parallel, purrr, rlang, stringr, tibble, tidyr, tidyselect (>= 1.2.0)

Depends R (>= 4.2)

LazyData true

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Anna Laksafoss [aut, cre] (<https://orcid.org/0000-0002-9898-2924>)

Maintainer Anna Laksafoss <adls@ssi.dk>

Repository CRAN

Date/Publication 2023-02-10 10:10:11 UTC

R topics documented:

`EpiForsk-package` *EpiForsk*

Description

This is a collection of assorted functions and examples collected from various projects. Currently we have functionalities for simplifying overlapping time intervals, Charlson comorbidity score constructors for Danish data, sibling design linear regression functionalities, and a method for calculating the confidence intervals for functions of parameters from a GLM.

Details

In the package there are contributions from

- ADLS : Anna Damkjær Laksafoss (https://orcid.org/0000-0002-9898-2924)
- ANDH : Anders Husby (https://orcid.org/0000-0002-7634-8455)
- ASO : Mikael Andersson
- EMTH : Emilia Myrup Thiesson
- KIJA : Kim Daniel Jakobsen

Author(s)

Maintainer: Anna Laksafoss <adls@ssi.dk> (ORCID)

`adls_timevarying_region_data`
Simulated Time-Varying Residence Data

Description

A dataset of simulated time-varying residence and gender data.

Usage

```
adls_timevarying_region_data
```

Format

andh_forest_data:

A data frame with 546 rows and 7 columns describing 100 people:

id an id number

dob date of birth

region region of Denmark

move_in date of moving to region

move_out date of moving away from region

gender gender of the person

claim whether or not the person made a claim here

---

andh_forest_data *Example Data for Husby's Forest Plot Vignette*

Description

A data example for the construction of a multi faceted forest plot.

Usage

```
andh_forest_data
```

Format

andh_forest_data:

A data frame with 18 rows and 12 columns:

type text formattaing, bold/plain

indent number of indents in final formatting

text description text

A_est point estimate in first figure column

A_l lower limit of confidence interval in first figure column

A_u upper limit of confidence interval in first figure column

B_est point estimate in second figure column

B_l lower limit of confidence interval in second figure column

B_u upper limit of confidence interval in second figure column

C_est point estimate in third figure column

C_l lower limit of confidence interval in third figure column

C_u upper limit of confidence interval in third figure column

braid_rows *Bind lists of list of multiple data frames by row*

Description

Row binds the matching innermost dataframes in a list of lists. This is essentially a list inversion `purrr::list_transpose()` with row-binding `dplyr::bind_rows()`

Usage

```
braid_rows(list)
```

Arguments

`list`	A list of lists of `data.frames` where matching innermost elements must be bound together row-wise.

Value

A list of `data.frames` with the combined information from the inputted list.

Examples

```
# A simple example
lst <- lapply(1:5, function(x) {
  list(
    "A" = data.frame("first" = x, "second" = rnorm(x)),
    "B" = data.frame("info" = 1, "other" = 3)
  )
})
braid_rows(lst)

# An example with an additional layer and jagged innermost info
lapply(c(28, 186, 35), function(len) {
  lapply(c("a", "b"), function(x) {
    res <- list(
      "descriptive" = data.frame(
         risk = len,
         event = x,
         var = 1,
         other = 2
       ),
      "results" = data.frame(
         risk = len,
         event = x,
         important = 4:7,
         new = 3:6
      )
    )
    if (len < 30) {
```

```
      res <- c(res, list("additional" = data.frame(helps = "extra data")))
    }
    return(res)
  }) |> braid_rows()
}) |> braid_rows()
```

---

charlson_score *Charlson Score Constructor*

Description

Charlson comorbidity score for Danish ICD-10 and ICD-8 data. This is a SAS-macro ASO translated to R in March of 2022

Usage

```
charlson_score(
  data,
  Person_ID,
  diagnosis_variable,
  time_variable = NULL,
  end_date = NULL,
  days_before_end_date = NULL,
  amount_output = "total"
)
```

Arguments

data	A data.frame with at least an id variable and a variable with all diagnosis codes. The data should be in the long format (only one variable with diagnoses, but several lines per person is OK).
Person_ID	<data-masking> An unquoted expression naming the id variable in data. This variable must always be specified.
diagnosis_variable	<data-masking> An unquoted expression naming the diagnosis variable in data. This variable must always be specified.
time_variable	<data-masking> An unquoted expression naming the diagnosis time variable in data if needed. The time_variable must be in a date format. When time_variable is specified, end_date must also be specified.
end_date	<data-masking> An unquoted expression naming the end of time-period to search for relevant diagnoses or a single date specifying the end date. If end_date names a variable, this variable must be in a date format.
days_before_end_date	A numeric specifying the number of days look-back from end_date to search for relevant diagnoses.

`amount_output` A character specifying whether all created index variables should be returned. When `amount_output` is "total" (the default) only the resulting Charlson scores are returned, otherwise all disease- specific index variables are returned.

Details

The `charlson_score()` function calculates the Charlson Charlson Comorbidity Index for each person. Three different variations on the score has been implemented:

- "cc"Article from Quan et al. (Coding Algorithms for Defining Comorbidities in ICD-9 and ICD-10 Administrative Data, Med Care 2005:43: 1130-1139), the same HTR and others have used - ICD10 only
- "ch"Article from Christensen et al. (Comparison of Charlson comorbidity index with SAPS and APACHE sources for prediction of mortality following intensive care, Clinical Epidemiology 2011:3 203-211), include ICD8 and ICD10 but the included diagnoses are not the same as in Quan
- "cd"Article from Sundarajan et al. (New ICD-10 version of Charlson Comorbidity Index predicted in-hospital mortality, Journal of clinical Epidemiology 57 (2004) 1288-1294, include ICD10 = Charlson-Deyo including cancer

Value

If `Person_ID` and `diagnosis_variable` are the only specifications, the function will calculate the different versions of the Charlson score on all data available for each person, regardless of timing etc. This is OK if only relevant records are included.

NOTE

The diagnoses to use in this function at the current state should be either ICD-8, but preferably ICD-10. The ICD-10 codes should start with two letters, where the first one is "D". Furthermore, the code should only have letters and digits (i.e. the form "DA000" not "DA00.0")

Author(s)

ASO & ADLS

Examples

```
# An example dataset

test_data <- data.frame(
  IDs = c(
    1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
    17, 18, 19, 20, 21, 22, 22, 23, 23, 24, 24, 24, 24, 24
  ),
  Diags = c(
    "DZ36", "DZ38", "DZ40", "DZ42", "DC20", "DI252",
    "DP290", "DI71", "DH340", "DG30", "DJ40", "DM353",
```

```
    "DK26", "DK700", "DK711", "DE106", "DE112", "DG82",
    "DZ940", "DC80", "DB20", "DK74", "DK704", "DE101",
    "DE102", "DB20", "DK74", "DK704", "DE101", "DE102"
 ),
 time = as.Date(c(
   "2001-01-30", "2004-05-20", "2007-01-02", "2013-12-01",
   "2017-04-30", "2001-01-30", "2004-05-20", "2007-01-02",
   "2013-12-01", "2017-04-30", "2001-01-30", "2004-05-20",
   "2007-01-02", "2013-12-01", "2017-04-30", "2001-01-30",
   "2004-05-20", "2007-01-02", "2013-12-01", "2017-04-30",
   "2001-01-30", "2004-05-20", "2007-01-02", "2013-12-01",
   "2017-04-30", "2001-01-30", "2004-05-20", "2007-01-02",
   "2013-12-01", "2017-04-30"
 )),
 match_date = as.Date(c(
   "2001-10-15", "2005-10-15", "2011-10-15", "2021-10-15",
   "2021-10-15", "2001-10-15", "2005-10-15", "2011-10-15",
   "2021-10-15", "2021-10-15", "2001-10-15", "2005-10-15",
   "2011-10-15", "2021-10-15", "2021-10-15", "2001-10-15",
   "2005-10-15", "2011-10-15", "2021-10-15", "2021-10-15",
   "2001-10-15", "2005-10-15", "2011-10-15", "2021-10-15",
   "2021-10-15", "2001-10-15", "2005-10-15", "2011-10-15",
   "2021-10-15", "2021-10-15"
 ))
)

# Minimal example
charlson_score(
  data = test_data,
  Person_ID = IDs,
  diagnosis_variable = Diags
)

# Minimal example with all index diagnosis variables
charlson_score(
  data = test_data,
  Person_ID = IDs,
  diagnosis_variable = Diags,
  amount_output = "all"
)

# Imposing uniform date restrictions to diagnoses
charlson_score(
  data = test_data,
  Person_ID = IDs,
  diagnosis_variable = Diags,
  time_variable = time,
  end_date = as.Date("2012-01-01")
)

# Imposing differing date restriction to diagnoses
charlson_score(
  data = test_data,
```

```
  Person_ID = IDs,
  diagnosis_variable = Diags,
  time_variable = time,
  end_date = match_date
)

# Imposing both a start and end to the lookup period for
# relevant diagnoses
charlson_score(
  data = test_data,
  Person_ID = IDs,
  diagnosis_variable = Diags,
  time_variable = time,
  end_date = match_date,
  days_before_end_date = 365.25
)
```

`fct_confint` *Confidence set for functions of model parameters*

Description

Computes confidence sets of functions of model parameters by computing a confidence set of the model parameters and returning the codomain of the provided function given the confidence set of model parameters as domain.

Usage

```
fct_confint(
  object,
  f,
  which_parm = rep(TRUE, length(coef(object))),
  level = 0.95,
  ...
)

## S3 method for class 'lm'
fct_confint(
  object,
  f,
  which_parm = rep(TRUE, length(coef(object))),
  level = 0.95,
  len = 0.1,
  n_grid = 0L,
  k = 1000L,
  parallel = FALSE,
```

```
  n_cores = 10,
  return_beta = FALSE,
  verbose = FALSE,
  ...
)

## S3 method for class 'glm'
fct_confint(
  object,
  f,
  which_parm = rep(TRUE, length(coef(object))),
  level = 0.95,
  len = 0.1,
  n_grid = 0L,
  k = 1000L,
  parallel = FALSE,
  n_cores = 10,
  return_beta = FALSE,
  verbose = FALSE,
  ...
)

## S3 method for class 'lms'
fct_confint(
  object,
  f,
  which_parm = rep(TRUE, length(coef(object))),
  level = 0.95,
  len = 0.1,
  n_grid = 0L,
  k = 1000L,
  parallel = FALSE,
  n_cores = 10,
  return_beta = FALSE,
  verbose = FALSE,
  ...
)

## Default S3 method:
fct_confint(
  object,
  f,
  which_parm = rep(TRUE, length(coef(object))),
  level = 0.95,
  ...
)
```

Arguments

object	A fitted model object.
f	A function taking the parameter vector as its single argument, and returning a numeric vector.
which_parm	Either a logical vector the same length as the coefficient vector, with TRUE indicating a coefficient is used by f, or an integer vector with the indices of the coefficients used by f.
level	The confidence level required.
...	Additional argument(s) passed to methods.
len	numeric, the radius of the sphere or box used to define directions in which to look for boundary points of the parameter confidence set.
n_grid	Either NULL or an integer vector of length 1 or the number of TRUE/indices in which_parm. Specifies the number of grid points in each dimension of a grid with endpoints defined by len. If NULL or 0L, will instead sample k points uniformly on a sphere.
k	If n_grid is NULL or 0L, the number of points to sample uniformly from a sphere.
parallel	Logical, if TRUE parallel computing is used when solving for points on the boundary of the parameter confidence set.
n_cores	An integer specifying the number of threads to use for parallel computing.
return_beta	Logical, if TRUE returns both the confidence limits and the parameter values used from the boundary of the parameter confidence set.
verbose	Logical, if TRUE prints information about the number of points on the boundary used to calculate the confidence limits.

Details

Assume the response Y and predictors X are given by a generalized linear model, that is, they fulfill the assumptions

$$E(Y|X) = \mu(X^T\beta)$$
$$V(Y|X) = \psi\nu(\mu(X^T\beta))$$
$$Y|X \sim \varepsilon(\theta, \nu_\psi).$$

Here μ is the mean value function, ν is the variance function, and ψ is the dispersion parameter in the exponential dispersion model $\varepsilon(\theta, \nu_\psi)$, where θ is the canonical parameter and ν_ψ is the structure measure. Then it follows from the central limit theorem that

$$hat(\beta) \sim N(\beta, (X^TWX)^{-1})$$

will be a good approximation in large samples, where X^TWX is the Fisher information of the exponential dispersion model.

From this, the combinant

$$(hat(\beta) - \beta)^T X^T W X (hat(\beta) - \beta)$$

is an approximate pivot, with a χ^2_p distribution. Then

$$C_\beta = \{\beta | (hat(\beta) - \beta)^T X^T W X (hat(\beta) - \beta) < \chi^2_p(1-\alpha)\}$$

is an approximate $(1-\alpha)$-confidence set for the parameter vector β. Similarly, confidence sets for sub-vectors of β can be obtained by the fact that marginal distributions of normal distributions are again normally distributed, where the mean vector and covariance matrix are appropriate subvectors and submatrices.

Finally, a confidence set for the transformed parameters $f(\beta)$ is obtained as

$$\{f(\beta)|\beta \in C_\beta\}$$

Note this is a conservative confidence set, since parameters outside the confidence set of β can be mapped to the confidence set of the transformed parameter.

To determine C_β, `fct_confint()` finds the boundary by taking a number of points around $hat(\beta)$ and projecting them onto the boundary. Therefore, the confidence set of the transformed parameter will only be valid if the boundary of C_β is mapped to the boundary of the confidence set for the transformed parameter.

The points projected to the boundary are either laid out in a grid around $hat(\beta)$, with the number of points in each direction determined by `n_grid`, or uniformly at random on a hypersphere, with the number of points determined by `k`. The radius of the grid/sphere is determined by `len`.

Value

A tibble with columns estimate, conf.low, and conf.high or if return_beta is `TRUE`, a list with the tibble and the beta values on the boundary used to calculate the confidence limits.

Author(s)

KIJA

Examples

```
1+1
```

---

flatten_date_intervals

Flatten Date Intervals

Description

A tidyverse compatible function for simplifying time interval data

Usage

```
flatten_date_intervals(
  data,
  id,
  in_date,
  out_date,
```

```
  status = NULL,
  overlap_handling = "most_recent",
  lag = 0
)
```

Arguments

`data`	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from dbplyr or dtplyr).
`id`	<tidy-select> One or more unquoted expression naming the id variables in data.
`in_date`	<data-masking> One unquoted expressions naming the start date variable in data.
`out_date`	<data-masking> One unquoted expression naming the end date variable in data.
`status`	<tidy-select> One or more unquoted expressions naming a status variable in data, such as region or hospitalization reason.
`overlap_handling`	A character naming the method for handling overlaps within an individuals time when `status` has been specified. • "none": No special handling of the overlapping time intervals within person is done. • "first": The `status` mentioned first, that is, has the smallest `in_date`, dominates. • "most_recent" (default): The most recent `status`, that is, the one with the largest `in_date`, dominates. When the most recent `status` is fully contained within an older (and different) `status` then the `out_date` associated with the most recent `in_date` is kept, but the remaining time from the older `status` is removed. See examples below. We currently don't have a method that lets the most recent status dominate and then potentially return to an older longer running status. If this is needed, please contact ADLS.
`lag`	A numeric, giving the number of days allowed between time intervals that should be collapsed into one.

Details

This functions identifies overlapping time intervals within individual and collapses them into distinct and disjoint intervals. When `status` is specified these intervals are both individual and status specific.

If `lag` is specified then intervals must be more then `lag` time units apart to be considered distinct.

Value

A data frame with the `id`, `status` if specified and simplified `in_date` and `out_date`. The returned data is sorted by `id` and `in_date`.

Author(s)

ADLS, EMTH & ASO

Examples

```
### The flatten function works with both dates and numeric

dat <- data.frame(
   ID    = c(1, 1, 1, 2, 2, 3, 3, 4),
   START = c(1, 2, 5, 3, 6, 2, 3, 6),
   END   = c(3, 3, 7, 4, 9, 3, 5, 8))
dat |> flatten_date_intervals(ID, START, END)

dat <- data.frame(
   ID    = c(1, 1, 1, 2, 2, 3, 3, 4, 4),
   START = as.Date(c("2012-02-15", "2005-12-13", "2006-01-24",
                     "2002-03-14", "1997-02-27",
                     "2008-08-13", "1998-09-23",
                     "2005-01-12", "2007-05-10")),
   END   = as.Date(c("2012-06-03", "2007-02-05", "2006-08-22",
                     "2005-02-26", "1999-04-16",
                     "2008-08-22", "2015-01-29",
                     "2007-05-07", "2008-12-12")))
dat |> flatten_date_intervals(ID, START, END)



###  Allow for a 5 days lag between

dat |> flatten_date_intervals(ID, START, END, lag = 5)



### Adding status information

dat <- data.frame(
   ID     = c(1, 1, 1, 2, 2, 3, 3, 4, 4),
   START  = as.Date(c("2012-02-15", "2005-12-13", "2006-01-24",
                      "2002-03-14", "1997-02-27",
                      "2008-08-13", "1998-09-23",
                      "2005-01-12", "2007-05-10")),
   END    = as.Date(c("2012-06-03", "2007-02-05", "2006-08-22",
                      "2005-02-26", "1999-04-16",
                      "2008-08-22", "2015-01-29",
                     "2007-05-07", "2008-12-12")),
   REGION = c("H", "H", "N", "S", "S", "M", "N", "S", "S"))

# Note the difference between the the different overlap_handling methods
dat |> flatten_date_intervals(ID, START, END, REGION, "none")
dat |> flatten_date_intervals(ID, START, END, REGION, "first")
dat |> flatten_date_intervals(ID, START, END, REGION, "most_recent")
```

`lms` *Wrapper around lm for sibling design*

Description

Fits a linear model using demeaned data. Useful for sibling design.

Usage

```
lms(formula, data, grp_id, obs_id = NULL, ...)

## S3 method for class 'lms'
print(x, digits = max(3L, getOption("digits") - 3L), ...)
```

Arguments

`formula`	A formula, used to create a model matrix with demeaned columns.
`data`	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from dbplyr or dtplyr).
`grp_id`	<data-masking> One unquoted expression naming the id variable in data defining the groups to demean, e.g. sibling groups.
`obs_id`	<data-masking> Optional, One unquoted expression naming an id variable to keep track of the input data order.
`...`	Additional arguments to be passed to lm(). In print, additional arguments are ignored without warning.
`x`	An S3 object with class lms.
`digits`	The number of significant digits to be passed to format(coef(x), .) when print()ing.

Details

`lms` estimates parameters in the linear model

$$y_{ij_i} = \alpha_i + x_{ij_i}^T \beta + \varepsilon_{ij_i}$$

where α_i is a group (e.g. sibling group) specific intercept and x_{ij_i} are covariate values for observation j_i in group i. $\varepsilon_{ij_i} \sim N(0, \sigma^2)$ is a normally distributed error term. It is assumed that interest is in estimating the vector β while α_{ij_i} are nuisance parameters. Estimation of β uses the mean deviation method, where

$$y_{ij_i}^{'} = y_{ij_i} - y_i$$

is regressed on

$$x_{ij_i}^{'} = x_{ij_i} - x_i.$$

Here y_i and x_i refers to the mean of y and x in group i.
`lms` can keep track of observations by providing a unique identifier column to `obs_id`. `lms` will return `obs_id` so it matches the order of observations in model.
`lms` only supports syntactic covariate names. Using a non-syntactic name risks returning an error, e.g if names end in + or -.

Value

A list with class `c("lms", "lm")`. Contains the output from `lm` applied to demeaned data according to `formula`, as well as the original data and the provided formula.

Author(s)

KIJA

Examples

```
sib_id <- sample(200, 1000, replace = TRUE)
sib_out <- rnorm(200)
x1 <- rnorm(1000)
x2 <- rnorm(1000) + sib_out[sib_id] + x1
y <- rnorm(1000, 1, 0.5) + 2 * sib_out[sib_id] - x1 + 2 * x2
data <- data.frame(
  x1 = x1,
  x2 = x2,
  y = y,
  sib_id = sib_id,
  obs_id = 1:1000
)
mod_lm <- lm(y ~ x1 + x2, data) # OLS model
mod_lm_grp <- lm(y ~ x1 + x2 + factor(sib_id), data) # OLS with grp
mod_lms <- lms(y ~ x1 + x2, data, sib_id, obs_id) # conditional model
summary(mod_lm)
coef(mod_lm_grp)[1:3]
summary(mod_lms)
print(mod_lms)
```

Index