Package 'Colossus'

October 30, 2025

Type Package

Title ``Risk Model Regression and Analysis with Complex Non-Linear Models"

Version 1.4.3

URL https://ericgiunta.github.io/Colossus/, https://github.com/ericgiunta/Colossus

BugReports https://github.com/ericgiunta/Colossus/issues

Description Performs survival analysis using general non-linear models. Risk models can be the sum or product of terms. Each term is the product of exponential/linear functions of covariates. Additionally sub-terms can be defined as a sum of exponential, linear threshold, and step functions. Cox Proportional hazards, Poisson, and Fine-Gray competing risks regression are supported. This work was sponsored by NASA Grants 80NSSC19M0161 and 80NSSC23M0129 through a subcontract from the National Council on Radiation Protection and Measurements (NCRP). The computing for this project was performed on the Beocat Research Cluster at Kansas State University, which is funded in part by NSF grants CNS-1006860, EPS-1006860, EPS-0919443, ACI-1440548, CHE-1726332, and NIH P20GM113109.

License GPL (>= 3)

Imports Rcpp, data.table, parallel, stats, utils, rlang, methods, callr, stringr, processx, dplyr, tibble, lubridate

LinkingTo Rcpp, RcppEigen, testthat

SystemRequirements make, C++17

RoxygenNote 7.3.3

Encoding UTF-8

Suggests knitr, rmarkdown, testthat, xml2, ggplot2, pandoc, spelling, survival, splines, dtplyr

Config/testthat/edition 3

VignetteBuilder knitr

Language en-US

Biarch TRUE

NeedsCompilation yes

Author Eric Giunta [aut, cre] (ORCID: <https://orcid.org/0000-0002-1577-766X>),
Amir Bahadori [ctb] (ORCID: <https://orcid.org/0000-0002-4589-105X>),
Dan Andresen [ctb] (ORCID: <https://orcid.org/0000-0003-2345-6695>),
Linda Walsh [ctb] (ORCID: <https://orcid.org/0000-0001-7399-9191>),
Benjamin French [ctb] (ORCID: <https://orcid.org/0000-0001-9265-5378>),
Lawrence Dauer [ctb] (ORCID: <https://orcid.org/0000-0002-5629-8462>),
John Boice Jr [ctb] (ORCID: <https://orcid.org/0000-0002-8755-1299>),
Kansas State University [cph],
NASA [fnd],
NCRP [fnd],
NRC [fnd]

Maintainer Eric Giunta <egiunta@ksu.edu>

Repository CRAN

Date/Publication 2025-10-30 15:50:02 UTC

Contents

`apply_norm` *Automatically applies a normalization to either an input or output*

Description

`apply_norm` applies a normalization factor

Usage

```
apply_norm(df, norm, names, input, values, model_control)
```

Arguments

`df`	The data.table with columns to be normalized
`norm`	The normalization option used, currently max or mean
`names`	columns for elements of the model, used to identify data columns
`input`	boolean if the normalization is being performed on the input values or on an output
`values`	list of values using during normalization
`model_control`	controls which alternative model options are used, see the Control_Options vignette for further details

Value

returns list with the normalized values

See Also

Other Data Cleaning Functions: `Check_Iters()`, `Date_Shift()`, `Event_Count_Gen()`, `Event_Time_Gen()`, `Joint_Multiple_Events()`, `Replace_Missing()`, `Time_Since()`, `factorize()`, `gen_time_dep()`

`CaseControlRun` *Fully runs a case-control regression model, returning the model and results*

Description

`CaseControlRun` uses a formula, data.table, and list of controls to prepare and run a Colossus matched case-control regression function

Usage

```
CaseControlRun(
  model,
  df,
  a_n = list(c(0)),
  keep_constant = c(0),
  control = list(),
  conditional_threshold = 50,
  gradient_control = list(),
  single = FALSE,
  observed_info = FALSE,
  cons_mat = as.matrix(c(0)),
  cons_vec = c(0),
  norm = "null",
  ...
)
```

Arguments

`model`	either a formula written for the get_form function, or the model result from the get_form function.
`df`	a data.table containing the columns of interest
`a_n`	list of initial parameter values, used to determine the number of parameters. May be either a list of vectors or a single vector.
`keep_constant`	binary values to denote which parameters to change
`control`	list of parameters controlling the convergence, see the Control_Options vignette for details

conditional_threshold
: threshold above which unconditional logistic regression is used to calculate likelihoods in a matched group

gradient_control
: a list of control options for the gradient descent algorithm. If any value is given, a gradient descent algorithm is used instead of Newton-Raphson. See the Control_Options vignette for details

single
: a boolean to denote that only the log-likelihood should be calculated and returned, no derivatives or iterations

observed_info
: a boolean to denote that the observed information matrix should be used to calculate the standard error for parameters, not the expected information matrix

cons_mat
: Matrix containing coefficients for a system of linear constraints, formatted as matrix

cons_vec
: Vector containing constants for a system of linear constraints, formatted as vector

norm
: methods used to normalize the covariates. Default is 'null' for no normalization. Other options include 'max' to normalize by the absolute maximum and 'mean' to normalize by the mean

...
: can include the named entries for the control list parameter

Value

returns a class fully describing the model and the regression results

See Also

Other Case Control Wrapper Functions: RunCaseControlRegression_Omnibus()

Examples

```
library(data.table)
df <- data.table::data.table(
  "UserID" = c(112, 114, 213, 214, 115, 116, 117),
  "Starting_Age" = c(18, 20, 18, 19, 21, 20, 18),
  "Ending_Age" = c(30, 45, 57, 47, 36, 60, 55),
  "Cancer_Status" = c(0, 0, 1, 0, 1, 0, 0),
  "a" = c(0, 1, 1, 0, 1, 0, 1),
  "b" = c(1, 1.1, 2.1, 2, 0.1, 1, 0.2),
  "c" = c(10, 11, 10, 11, 12, 9, 11),
  "d" = c(0, 0, 0, 1, 1, 1, 1),
  "e" = c(0, 0, 1, 0, 0, 0, 1)
)
control <- list(
  "ncores" = 2, "lr" = 0.75, "maxiters" = c(1, 1),
  "halfmax" = 1
)
formula <- CaseCon_Strata(Cancer_Status, e) ~
  loglinear(a, b, c, 0) + plinear(d, 0) + multiplicative()
res <- CaseControlRun(formula, df,
```

```
  a_n = list(c(1.1, -0.1, 0.2, 0.5), c(1.6, -0.12, 0.3, 0.4)),
  control = control
)
```

`Check_Iters` *Automatically checks the number of starting guesses*

Description

`Check_Iters` checks the number of iterations and number of guesses, and corrects

Usage

```
Check_Iters(control, a_n)
```

Arguments

`control`	list of parameters controlling the convergence, see the Control_Options vignette for details
`a_n`	list of initial parameter values, used to determine the number of parameters. May be either a list of vectors or a single vector.

Value

returns a list with the corrected control list and a_n

See Also

Other Data Cleaning Functions: `Date_Shift()`, `Event_Count_Gen()`, `Event_Time_Gen()`, `Joint_Multiple_Events()`, `Replace_Missing()`, `Time_Since()`, `apply_norm()`, `factorize()`, `gen_time_dep()`

`ColossusCoxSurv` *Interprets basic cox survival formula RHS*

Description

`ColossusCoxSurv` assigns and interprets interval columns for cox model. This functions is called using the arguments for Cox in the right-hand side of the formula. Uses an interval start time, end time, and event status. These are expected to be in order or named: tstart, tend, and event. The Fine-Gray and Stratified versions use strata and weight named options or the last two entries.

Usage

```
ColossusCoxSurv(...)
```

Arguments

...	entries for a cox survival object, tstart, tend, and event. Either in order or named. If unnamed and two entries, tend and event are assumed.

Value

returns list with interval endpoints and event

See Also

Other Formula Interpretation: `ColossusLogitSurv()`, `ColossusPoisSurv()`, `get_form()`, `get_form_joint()`

`ColossusLogitSurv` *Interprets basic logistic survival formula RHS with no grouping*

Description

`ColossusLogitSurv` assigns and interprets columns for trials and events in logistic model with no grouping.

Usage

```
ColossusLogitSurv(...)
```

Arguments

...	entries for a Logistic object, trials and events. trials not provided assumes one trial per row.

Value

returns list with event

See Also

Other Formula Interpretation: `ColossusCoxSurv()`, `ColossusPoisSurv()`, `get_form()`, `get_form_joint()`

ColossusPoisSurv *Interprets basic poisson survival formula RHS*

Description

ColossusPoisSurv assigns and interprets interval columns for poisson model. This functions is called using the arguments for Poisson or Poisson_Strata in the right-hand side of the formula. Uses an person-year column, number of events, and any strata columns. The first two are expected to be in order or named: pyr and event. Anything beyond the event name is assumed to be strata if Poisson_Strata is used.

Usage

```
ColossusPoisSurv(...)
```

Arguments

...	entries for a Poisson object with or without strata, pyr, event, and any strata columns. Either in order or named. The first two are assumed to be pyr and event, the rest assumed to be strata columns

Value

returns list with duration, strata if used, and event

See Also

Other Formula Interpretation: ColossusCoxSurv(), ColossusLogitSurv(), get_form(), get_form_joint()

CoxRun *Fully runs a cox or fine-gray regression model, returning the model and results*

Description

CoxRun uses a formula, data.table, and list of controls to prepare and run a Colossus cox or fine-gray regression function

Usage

```
CoxRun(
  model,
  df,
  a_n = list(c(0)),
  keep_constant = c(0),
  control = list(),
```

```
  gradient_control = list(),
  single = FALSE,
  observed_info = FALSE,
  cons_mat = as.matrix(c(0)),
  cons_vec = c(0),
  norm = "null",
  ...
)
```

Arguments

model	either a formula written for the get_form function, or the model result from the get_form function.
df	a data.table containing the columns of interest
a_n	list of initial parameter values, used to determine the number of parameters. May be either a list of vectors or a single vector.
keep_constant	binary values to denote which parameters to change
control	list of parameters controlling the convergence, see the Control_Options vignette for details
gradient_control	a list of control options for the gradient descent algorithm. If any value is given, a gradient descent algorithm is used instead of Newton-Raphson. See the Control_Options vignette for details
single	a boolean to denote that only the log-likelihood should be calculated and returned, no derivatives or iterations
observed_info	a boolean to denote that the observed information matrix should be used to calculate the standard error for parameters, not the expected information matrix
cons_mat	Matrix containing coefficients for a system of linear constraints, formatted as matrix
cons_vec	Vector containing constants for a system of linear constraints, formatted as vector
norm	methods used to normalize the covariates. Default is 'null' for no normalization. Other options include 'max' to normalize by the absolute maximum and 'mean' to normalize by the mean
...	can include the named entries for the control list parameter

Value

returns a class fully describing the model and the regression results

See Also

Other Cox Wrapper Functions: `CoxRunMulti()`, `LikelihoodBound.coxres()`, `RunCoxRegression_Omnibus()`

Examples

```
library(data.table)
df <- data.table::data.table(
  "UserID" = c(112, 114, 213, 214, 115, 116, 117),
  "Starting_Age" = c(18, 20, 18, 19, 21, 20, 18),
  "Ending_Age" = c(30, 45, 57, 47, 36, 60, 55),
  "Cancer_Status" = c(0, 0, 1, 0, 1, 0, 0),
  "a" = c(0, 1, 1, 0, 1, 0, 1),
  "b" = c(1, 1.1, 2.1, 2, 0.1, 1, 0.2),
  "c" = c(10, 11, 10, 11, 12, 9, 11),
  "d" = c(0, 0, 0, 1, 1, 1, 1),
  "e" = c(0, 0, 1, 0, 0, 0, 1)
)
control <- list(
  "ncores" = 2, "lr" = 0.75, "maxiters" = c(1, 1),
  "halfmax" = 1
)
formula <- Cox(Starting_Age, Ending_Age, Cancer_Status) ~
  loglinear(a, b, c, 0) + plinear(d, 0) + multiplicative()
res <- CoxRun(formula, df,
  a_n = list(c(1.1, -0.1, 0.2, 0.5), c(1.6, -0.12, 0.3, 0.4)),
  control = control
)
```

`CoxRunMulti` *Fully runs a cox or fine-gray regression model with multiple column realizations, returning the model and results*

Description

`CoxRunMulti` uses a formula, data.table, and list of controls to prepare and run a Colossus cox or fine-gray regression function

Usage

```
CoxRunMulti(
  model,
  df,
  a_n = list(c(0)),
  keep_constant = c(0),
 realization_columns = matrix(c("temp00", "temp01", "temp10", "temp11"), nrow = 2),
  realization_index = c("temp0", "temp1"),
  control = list(),
  gradient_control = list(),
  single = FALSE,
  observed_info = FALSE,
  fma = FALSE,
  mcml = FALSE,
```

```
  cons_mat = as.matrix(c(0)),
  cons_vec = c(0),
  ...
)
```

Arguments

model	either a formula written for the get_form function, or the model result from the get_form function.
df	a data.table containing the columns of interest
a_n	list of initial parameter values, used to determine the number of parameters. May be either a list of vectors or a single vector.
keep_constant	binary values to denote which parameters to change
realization_columns	used for multi-realization regressions. Matrix of column names with rows for each column with realizations, columns for each realization
realization_index	used for multi-realization regressions. Vector of column names, one for each column with realizations. Each name should be used in the "names" variable in the equation definition
control	list of parameters controlling the convergence, see the Control_Options vignette for details
gradient_control	a list of control options for the gradient descent algorithm. If any value is given, a gradient descent algorithm is used instead of Newton-Raphson. See the Control_Options vignette for details
single	a boolean to denote that only the log-likelihood should be calculated and returned, no derivatives or iterations
observed_info	a boolean to denote that the observed information matrix should be used to calculate the standard error for parameters, not the expected information matrix
fma	a boolean to denote that the Frequentist Model Averaging method should be used
mcml	a boolean to denote that the Monte Carlo Maximum Likelihood method should be used
cons_mat	Matrix containing coefficients for a system of linear constraints, formatted as matrix
cons_vec	Vector containing constants for a system of linear constraints, formatted as vector
...	can include the named entries for the control list parameter

Value

returns a class fully describing the model and the regression results

See Also

Other Cox Wrapper Functions: `CoxRun()`, `LikelihoodBound.coxres()`, `RunCoxRegression_Omnibus()`

Examples

```
library(data.table)
df <- data.table::data.table(
  "UserID" = c(112, 114, 213, 214, 115, 116, 117),
  "t0" = c(18, 20, 18, 19, 21, 20, 18),
  "t1" = c(30, 45, 57, 47, 36, 60, 55),
  "lung" = c(0, 0, 1, 0, 1, 0, 0),
  "dose" = c(0, 1, 1, 0, 1, 0, 1)
)
set.seed(3742)
df$rand <- floor(runif(nrow(df), min = 0, max = 5))
df$rand0 <- floor(runif(nrow(df), min = 0, max = 5))
df$rand1 <- floor(runif(nrow(df), min = 0, max = 5))
df$rand2 <- floor(runif(nrow(df), min = 0, max = 5))
names <- c("dose", "rand")
realization_columns <- matrix(c("rand0", "rand1", "rand2"), nrow = 1)
realization_index <- c("rand")
control <- list(
  "ncores" = 2, "lr" = 0.75, "maxiter" = 1,
  "halfmax" = 2, "epsilon" = 1e-6,
  "deriv_epsilon" = 1e-6, "step_max" = 1.0,
  "thres_step_max" = 100.0,
  "verbose" = 0, "ties" = "breslow", "double_step" = 1
)
formula <- Cox(t0, t1, lung) ~ loglinear(dose, rand, 0) + multiplicative()
res <- CoxRun(formula, df, control = control)
```

Date_Shift *Automates creating a date difference column*

Description

`Date_Shift` generates a new dataframe with a column containing time difference in a given unit

Usage

```
Date_Shift(df, dcol0, dcol1, col_name, units = "days")
```

Arguments

`df`	a data.table containing the columns of interest
`dcol0`	list of starting month, day, and year
`dcol1`	list of ending month, day, and year
`col_name`	vector of new column names
`units`	time unit to use

Value

returns the updated dataframe

See Also

Other Data Cleaning Functions: `Check_Iters()`, `Event_Count_Gen()`, `Event_Time_Gen()`, `Joint_Multiple_Events()`, `Replace_Missing()`, `Time_Since()`, `apply_norm()`, `factorize()`, `gen_time_dep()`

Examples

```
library(data.table)
m0 <- c(1, 1, 2, 2)
m1 <- c(2, 2, 3, 3)
d0 <- c(1, 2, 3, 4)
d1 <- c(6, 7, 8, 9)
y0 <- c(1990, 1991, 1997, 1998)
y1 <- c(2001, 2003, 2005, 2006)
df <- data.table::data.table("m0" = m0, "m1" = m1, "d0" = d0, "d1" = d1, "y0" = y0, "y1" = y1)
df <- Date_Shift(df, c("m0", "d0", "y0"), c("m1", "d1", "y1"), "date_since")
```

EventAssignment *Generic background/excess event calculation function*

Description

`EventAssignment` Generic background/excess event calculation function

Usage

```
EventAssignment(x, df, ...)
```

Arguments

`x`	result object from a regression, class poisres
`df`	a data.table containing the columns of interest
`...`	extended for other necessary parameters

EventAssignment.default

Predicts how many events are due to baseline vs excess

Description

EventAssignment Generic background/excess event calculation function, by default nothing happens

Usage

```
## Default S3 method:
EventAssignment(x, df, ...)
```

Arguments

x	result object from a regression, class poisres
df	a data.table containing the columns of interest
...	extended for other necessary parameters

EventAssignment.poisres

Predicts how many events are due to baseline vs excess for a completed poisson model

Description

EventAssignment.poisres uses user provided data, person-year/event columns, vectors specifying the model, and options to calculate background and excess events for a solved Poisson regression

Usage

```
## S3 method for class 'poisres'
EventAssignment(
  x,
  df,
  assign_control = list(),
  control = list(),
  a_n = c(),
  ...
)
```

Arguments

x	result object from a regression, class poisres
df	a data.table containing the columns of interest
assign_control	control list for bounds calculated
control	list of parameters controlling the convergence, see the Control_Options vignette for details
a_n	list of initial parameter values, used to determine the number of parameters. May be either a list of vectors or a single vector.
...	can include the named entries for the assign_control list parameter

Value

returns a list of the final results

See Also

Other Poisson Wrapper Functions: LikelihoodBound.poisres(), PoisRun(), PoisRunJoint(), PoisRunMulti(), Residual.poisres(), RunPoissonRegression_Omnibus()

Event_Count_Gen *uses a table, list of categories, and list of event summaries to generate person-count tables*

Description

Event_Count_Gen generates event-count tables

Usage

```
Event_Count_Gen(table, categ, events, verbose = FALSE)
```

Arguments

table	dataframe with every category/event column needed
categ	list with category columns and methods, methods can be either strings or lists of boundaries
events	list of columns to summarize, supports counts and means and renaming the summary column
verbose	integer valued 0-4 controlling what information is printed to the terminal. Each level includes the lower levels. 0: silent, 1: errors printed, 2: warnings printed, 3: notes printed, 4: debug information printed. Errors are situations that stop the regression, warnings are situations that assume default values that the user might not have intended, notes provide information on regression progress, and debug prints out C++ progress and intermediate results. The default level is 2 and True/False is converted to 3/0.

Value

returns a grouped table and a list of category boundaries used

See Also

Other Data Cleaning Functions: Check_Iters(), Date_Shift(), Event_Time_Gen(), Joint_Multiple_Events(), Replace_Missing(), Time_Since(), apply_norm(), factorize(), gen_time_dep()

Examples

```
library(data.table)
a <- c(0, 1, 2, 3, 4, 5, 6)
b <- c(1, 2, 3, 4, 5, 6, 7)
c <- c(0, 1, 0, 0, 0, 1, 0)
table <- data.table::data.table(
  "a" = a,
  "b" = b,
  "c" = c
)
categ <- list(
  "a" = "0/3/5]7",
  "b" = list(
    lower = c(-1, 3, 6),
    upper = c(3, 6, 10),
    name = c("low", "medium", "high")
  )
)
event <- list(
  "c" = "count AS cases",
  "a" = "mean", "b" = "mean"
)
e <- Event_Count_Gen(table, categ, event, T)
```

Event_Time_Gen *uses a table, list of categories, list of summaries, list of events, and person-year information to generate person-time tables*

Description

Event_Time_Gen generates event-time tables

Usage

```
Event_Time_Gen(table, pyr, categ, summaries, events, verbose = FALSE)
```

Arguments

table	dataframe with every category/event column needed
pyr	list with entry and exit lists, containing day/month/year columns in the table
categ	list with category columns and methods, methods can be either strings or lists of boundaries, includes a time category or entry/exit are both required for the pyr list
summaries	list of columns to summarize, supports counts, means, and weighted means by person-year and renaming the summary column
events	list of events or interests, checks if events are within each time interval
verbose	integer valued 0-4 controlling what information is printed to the terminal. Each level includes the lower levels. 0: silent, 1: errors printed, 2: warnings printed, 3: notes printed, 4: debug information printed. Errors are situations that stop the regression, warnings are situations that assume default values that the user might not have intended, notes provide information on regression progress, and debug prints out C++ progress and intermediate results. The default level is 2 and True/False is converted to 3/0.

Value

returns a grouped table and a list of category boundaries used

See Also

Other Data Cleaning Functions: Check_Iters(), Date_Shift(), Event_Count_Gen(), Joint_Multiple_Events(), Replace_Missing(), Time_Since(), apply_norm(), factorize(), gen_time_dep()

Examples

```
library(data.table)
a <- c(0, 1, 2, 3, 4, 5, 6)
b <- c(1, 2, 3, 4, 5, 6, 7)
c <- c(0, 1, 0, 0, 0, 1, 0)
d <- c(1, 2, 3, 4, 5, 6, 7)
e <- c(2, 3, 4, 5, 6, 7, 8)
f <- c(
  1900, 1900, 1900, 1900,
  1900, 1900, 1900
)
g <- c(1, 2, 3, 4, 5, 6, 7)
h <- c(2, 3, 4, 5, 6, 7, 8)
i <- c(
  1901, 1902, 1903, 1904,
  1905, 1906, 1907
)
table <- data.table::data.table(
  "a" = a, "b" = b, "c" = c,
  "d" = d, "e" = e, "f" = f,
  "g" = g, "h" = h, "i" = i
)
```

```
categ <- list(
  "a" = "-1/3/5]7",
  "b" = list(
    lower = c(-1, 3, 6), upper = c(3, 6, 10),
    name = c("low", "medium", "high")
  ),
  "time AS time" = list(
    "day" = c(1, 1, 1, 1, 1),
    "month" = c(1, 1, 1, 1, 1),
    "year" = c(1899, 1903, 1910)
  )
)
summary <- list(
  "c" = "count AS cases",
  "a" = "mean",
  "b" = "weighted_mean"
)
events <- list("c")
pyr <- list(
  entry = list(year = "f", month = "e", day = "d"),
  exit = list(year = "i", month = "h", day = "g"),
  unit = "years"
)
e <- Event_Time_Gen(table, pyr, categ, summary, events, T)
```

`factorize` *Splits a parameter into factors*

Description

`factorize` uses user provided list of columns to define new parameter for each unique value and update the data.table. Not for interaction terms

Usage

```
factorize(df, col_list, verbose = 0)
```

Arguments

`df`	a data.table containing the columns of interest
`col_list`	an array of column names that should have factor terms defined
`verbose`	integer valued 0-4 controlling what information is printed to the terminal. Each level includes the lower levels. 0: silent, 1: errors printed, 2: warnings printed, 3: notes printed, 4: debug information printed. Errors are situations that stop the regression, warnings are situations that assume default values that the user might not have intended, notes provide information on regression progress, and debug prints out C++ progress and intermediate results. The default level is 2 and True/False is converted to 3/0.

Value

returns a list with two named fields. df for the updated dataframe, and cols for the new column names

See Also

Other Data Cleaning Functions: Check_Iters(), Date_Shift(), Event_Count_Gen(), Event_Time_Gen(), Joint_Multiple_Events(), Replace_Missing(), Time_Since(), apply_norm(), gen_time_dep()

Examples

```
library(data.table)
a <- c(0, 1, 2, 3, 4, 5, 6)
b <- c(1, 2, 3, 4, 5, 6, 7)
c <- c(0, 1, 2, 1, 0, 1, 0)
df <- data.table::data.table("a" = a, "b" = b, "c" = c)
col_list <- c("c")
val <- factorize(df, col_list)
df <- val$df
new_col <- val$cols
```

---

gen_time_dep *Applies time dependence to parameters*

Description

gen_time_dep generates a new dataframe with time dependent covariates by applying a grid in time

Usage

```
gen_time_dep(
  df,
  time1,
  time2,
  event0,
  iscox,
  dt,
  new_names,
  dep_cols,
  func_form,
  fname,
  tform,
  nthreads = as.numeric(detectCores())
)
```

Arguments

df	a data.table containing the columns of interest
time1	column used for time period starts
time2	column used for time period end
event0	column used for event status
iscox	boolean if rows not at event times should not be kept, rows are removed if true. a Cox proportional hazards model does not use rows with intervals not containing event times
dt	spacing in time for new rows
new_names	list of new names to use instead of default, default used if entry is ""
dep_cols	columns that are not needed in the new dataframe
func_form	vector of functions to apply to each time-dependent covariate. Of the form func(df, time) returning a vector of the new column value
fname	filename used for new dataframe
tform	list of string function identifiers, used for linear/step
nthreads	number of threads to use, do not use more threads than available on your machine

Value

returns the updated dataframe

See Also

Other Data Cleaning Functions: Check_Iters(), Date_Shift(), Event_Count_Gen(), Event_Time_Gen(), Joint_Multiple_Events(), Replace_Missing(), Time_Since(), apply_norm(), factorize()

Examples

```
library(data.table)
# Adapted from the tests
a <- c(20, 20, 5, 10, 15)
b <- c(1, 2, 1, 1, 2)
c <- c(0, 0, 1, 1, 1)
df <- data.table::data.table("a" = a, "b" = b, "c" = c)
time1 <- "%trunc%"
time2 <- "a"
event <- "c"
control <- list(
  "lr" = 0.75, "maxiter" = -1, "halfmax" = 5, "epsilon" = 1e-9,
  "deriv_epsilon" = 1e-9, "step_max" = 1.0,
  "thres_step_max" = 100.0,
  "verbose" = FALSE, "ties" = "breslow", "double_step" = 1
)
grt_f <- function(df, time_col) {
  return((df[, "b"] * df[, get(time_col)])[[1]])
}
```

```
func_form <- c("lin")
df_new <- gen_time_dep(
  df, time1, time2, event, TRUE, 0.01, c("grt"), c(),
  c(grt_f), paste("test", "_new.csv", sep = ""), func_form, 2
)
file.remove("test_new.csv")
```

`get_form` *Interprets a Colossus formula and makes necessary changes to data*

Description

`get_form` uses a formula and data.table, to fully describe the model for a Colossus regression function.

Usage

```
get_form(formula, df)
```

Arguments

`formula`	a formula object, written in Colossus notation. See the Unified Equation Representation vignette for details.
`df`	a data.table containing the columns of interest

Value

returns a class fully describing the model and the updated data

See Also

Other Formula Interpretation: `ColossusCoxSurv()`, `ColossusLogitSurv()`, `ColossusPoisSurv()`, `get_form_joint()`

Examples

```
library(data.table)
## basic example code reproduced from the starting-description vignette
df <- data.table::data.table(
  "UserID" = c(112, 114, 213, 214, 115, 116, 117),
  "Starting_Age" = c(18, 20, 18, 19, 21, 20, 18),
  "Ending_Age" = c(30, 45, 57, 47, 36, 60, 55),
  "Cancer_Status" = c(0, 0, 1, 0, 1, 0, 0),
  "a" = c(0, 1, 1, 0, 1, 0, 1),
  "b" = c(1, 1.1, 2.1, 2, 0.1, 1, 0.2),
  "c" = c(10, 11, 10, 11, 12, 9, 11),
  "d" = c(0, 0, 0, 1, 1, 1, 1),
  "e" = c(0, 0, 1, 0, 0, 0, 1)
```

```
)
formula <- Cox(Starting_Age, Ending_Age, Cancer_Status) ~
  loglinear(a, b, c, 0) + plinear(d, 0) + multiplicative()
model <- get_form(formula, df)
```

`get_form_joint` *Interprets a Poisson joint formula and makes necessary changes to data*

Description

`get_form_joint` uses two event formula, a shared formula, and data.table, to fully describe the model for a joint Poisson model.

Usage

```
get_form_joint(formula_list, df)
```

Arguments

`formula_list`	a list of formula objects, each written in Colossus notation. See the Unified Equation Representation vignette for details. Each formula should include the elements specific to the specified event column. The list can include an entry named "shared" to denote shared terms. The person-year and strata columns should be the same.
`df`	a data.table containing the columns of interest

Value

returns a class fully describing the model and the updated data

See Also

Other Formula Interpretation: `ColossusCoxSurv()`, `ColossusLogitSurv()`, `ColossusPoisSurv()`, `get_form()`

Joint_Multiple_Events *Automates creating data for a joint competing risks analysis*

Description

Joint_Multiple_Events generates input for a regression with multiple non-independent events and models

Usage

```
Joint_Multiple_Events(
  df,
  events,
  name_list,
  term_n_list = list(),
  tform_list = list(),
  keep_constant_list = list(),
  a_n_list = list()
)
```

Arguments

df	a data.table containing the columns of interest
events	vector of event column names
name_list	list of vectors for columns for event specific or shared model elements, required
term_n_list	list of vectors for term numbers for event specific or shared model elements, defaults to term 0
tform_list	list of vectors for subterm types for event specific or shared model elements, defaults to loglinear
keep_constant_list	list of vectors for constant elements for event specific or shared model elements, defaults to free (0)
a_n_list	list of vectors for parameter values for event specific or shared model elements, defaults to term 0

Value

returns the updated dataframe and model inputs

See Also

Other Data Cleaning Functions: Check_Iters(), Date_Shift(), Event_Count_Gen(), Event_Time_Gen(), Replace_Missing(), Time_Since(), apply_norm(), factorize(), gen_time_dep()

Examples

```
library(data.table)
a <- c(0, 0, 0, 1, 1, 1)
b <- c(1, 1, 1, 2, 2, 2)
c <- c(0, 1, 2, 2, 1, 0)
d <- c(1, 1, 0, 0, 1, 1)
e <- c(0, 1, 1, 1, 0, 0)
df <- data.table("t0" = a, "t1" = b, "e0" = c, "e1" = d, "fac" = e)
time1 <- "t0"
time2 <- "t1"
df$pyr <- df$t1 - df$t0
pyr <- "pyr"
events <- c("e0", "e1")
names_e0 <- c("fac")
names_e1 <- c("fac")
names_shared <- c("t0", "t0")
term_n_e0 <- c(0)
term_n_e1 <- c(0)
term_n_shared <- c(0, 0)
tform_e0 <- c("loglin")
tform_e1 <- c("loglin")
tform_shared <- c("quad_slope", "loglin_top")
keep_constant_e0 <- c(0)
keep_constant_e1 <- c(0)
keep_constant_shared <- c(0, 0)
a_n_e0 <- c(-0.1)
a_n_e1 <- c(0.1)
a_n_shared <- c(0.001, -0.02)
name_list <- list("shared" = names_shared, "e0" = names_e0, "e1" = names_e1)
term_n_list <- list("shared" = term_n_shared, "e0" = term_n_e0, "e1" = term_n_e1)
tform_list <- list("shared" = tform_shared, "e0" = tform_e0, "e1" = tform_e1)
keep_constant_list <- list(
  "shared" = keep_constant_shared,
  "e0" = keep_constant_e0, "e1" = keep_constant_e1
)
a_n_list <- list("shared" = a_n_shared, "e0" = a_n_e0, "e1" = a_n_e1)
val <- Joint_Multiple_Events(
  df, events, name_list, term_n_list,
  tform_list, keep_constant_list, a_n_list
)
```

`LikelihoodBound` *Generic likelihood boundary calculation function*

Description

`LikelihoodBound` Generic likelihood boundary calculation function

Usage

```
LikelihoodBound(x, df, curve_control = list(), control = list(), ...)
```

Arguments

x	result object from a regression, class coxres or poisres
df	a data.table containing the columns of interest
curve_control	a list of control options for the likelihood boundary regression. See the Control_Options vignette for details.
control	list of parameters controlling the convergence, see the Control_Options vignette for details
...	extended for other necessary parameters

LikelihoodBound.coxres

Calculates the likelihood boundary for a completed cox model

Description

`LikelihoodBound.coxres` solves the confidence interval for a cox model, starting at the optimum point and iteratively optimizing end-points of intervals.

Usage

```
## S3 method for class 'coxres'
LikelihoodBound(x, df, curve_control = list(), control = list(), ...)
```

Arguments

x	result object from a regression, class coxres
df	a data.table containing the columns of interest
curve_control	a list of control options for the likelihood boundary regression. See the Control_Options vignette for details.
control	list of parameters controlling the convergence, see the Control_Options vignette for details
...	can include the named entries for the curve_control list parameter

Value

returns a list of the final results

See Also

Other Cox Wrapper Functions: `CoxRun()`, `CoxRunMulti()`, `RunCoxRegression_Omnibus()`

`LikelihoodBound.default`

Generic likelihood boundary calculation function, default option

---

Description

`LikelihoodBound` Generic likelihood boundary calculation function, by default nothing happens

Usage

```
## Default S3 method:
LikelihoodBound(x, df, curve_control = list(), control = list(), ...)
```

Arguments

`x`	result object from a regression, class coxres or poisres
`df`	a data.table containing the columns of interest
`curve_control`	a list of control options for the likelihood boundary regression. See the Control_Options vignette for details.
`control`	list of parameters controlling the convergence, see the Control_Options vignette for details
`...`	extended for other necessary parameters

`LikelihoodBound.poisres`

Calculates the likelihood boundary for a completed Poisson model

---

Description

`LikelihoodBound.poisres` solves the confidence interval for a Poisson model, starting at the optimum point and iteratively optimizing end-points of intervals.

Usage

```
## S3 method for class 'poisres'
LikelihoodBound(x, df, curve_control = list(), control = list(), ...)
```

Arguments

`x`	result object from a regression, class poisres
`df`	a data.table containing the columns of interest
`curve_control`	a list of control options for the likelihood boundary regression. See the Control_Options vignette for details.
`control`	list of parameters controlling the convergence, see the Control_Options vignette for details
`...`	can include the named entries for the curve_control list parameter

Value

returns a list of the final results

See Also

Other Poisson Wrapper Functions: EventAssignment.poisres(), PoisRun(), PoisRunJoint(), PoisRunMulti(), Residual.poisres(), RunPoissonRegression_Omnibus()

Likelihood_Ratio_Test *Defines the likelihood ratio test*

Description

Likelihood_Ratio_Test uses two models and calculates the ratio

Usage

```
Likelihood_Ratio_Test(alternative_model, null_model)
```

Arguments

alternative_model	the new model of interest in list form, output from a Poisson regression
null_model	a model to compare against, in list form

Value

returns the score statistic

Examples

```
library(data.table)
# In an actual example, one would run two seperate RunCoxRegression regressions,
#    assigning the results to e0 and e1
e0 <- list("name" = "First Model", "LogLik" = -120)
e1 <- list("name" = "New Model", "LogLik" = -100)
score <- Likelihood_Ratio_Test(e1, e0)
```

Linked_Dose_Formula *Calculates Full Parameter list for Special Dose Formula*

Description

Linked_Dose_Formula Calculates all parameters for linear-quadratic and linear-exponential linked formulas

Usage

```
Linked_Dose_Formula(tforms, paras, verbose = 0)
```

Arguments

tforms	list of formula types
paras	list of formula parameters
verbose	integer valued 0-4 controlling what information is printed to the terminal. Each level includes the lower levels. 0: silent, 1: errors printed, 2: warnings printed, 3: notes printed, 4: debug information printed. Errors are situations that stop the regression, warnings are situations that assume default values that the user might not have intended, notes provide information on regression progress, and debug prints out C++ progress and intermediate results. The default level is 2 and True/False is converted to 3/0.

Value

returns list of full parameters

Examples

```
library(data.table)
tforms <- list("cov_0" = "quad", "cov_1" = "exp")
paras <- list("cov_0" = c(1, 3.45), "cov_1" = c(1.2, 4.5, 0.1))
full_paras <- Linked_Dose_Formula(tforms, paras)
```

Linked_Lin_Exp_Para *Calculates The Additional Parameter For a linear-exponential formula with known maximum*

Description

Linked_Lin_Exp_Para Calculates what the additional parameter would be for a desired maximum

Usage

```
Linked_Lin_Exp_Para(y, a0, a1_goal, verbose = 0)
```

Arguments

y	point formula switch
a0	linear slope
a1_goal	exponential maximum desired
verbose	integer valued 0-4 controlling what information is printed to the terminal. Each level includes the lower levels. 0: silent, 1: errors printed, 2: warnings printed, 3: notes printed, 4: debug information printed. Errors are situations that stop the regression, warnings are situations that assume default values that the user might not have intended, notes provide information on regression progress, and debug prints out C++ progress and intermediate results. The default level is 2 and True/False is converted to 3/0.

Value

returns parameter used by Colossus

Examples

```
library(data.table)
y <- 7.6
a0 <- 1.2
a1_goal <- 15
full_paras <- Linked_Lin_Exp_Para(y, a0, a1_goal)
```

LogisticRun *Fully runs a logistic regression model, returning the model and results*

Description

LogisticRun uses a formula, data.table, and list of controls to prepare and run a Colossus logistic regression function

Usage

```
LogisticRun(
  model,
  df,
  a_n = list(c(0)),
  keep_constant = c(0),
  control = list(),
  gradient_control = list(),
  link = "odds",
```

```
  single = FALSE,
  observed_info = FALSE,
  cons_mat = as.matrix(c(0)),
  cons_vec = c(0),
  norm = "null",
  ...
)
```

Arguments

model	either a formula written for the get_form function, or the model result from the get_form function.
df	a data.table containing the columns of interest
a_n	list of initial parameter values, used to determine the number of parameters. May be either a list of vectors or a single vector.
keep_constant	binary values to denote which parameters to change
control	list of parameters controlling the convergence, see the Control_Options vignette for details
gradient_control	a list of control options for the gradient descent algorithm. If any value is given, a gradient descent algorithm is used instead of Newton-Raphson. See the Control_Options vignette for details
link	Used in logistic regression, the linking function relating the input model and event probability. Current options are "odds", "ident", and "loglink" for the odds ratio, identity, and complimentary loglink options.
single	a boolean to denote that only the log-likelihood should be calculated and returned, no derivatives or iterations
observed_info	a boolean to denote that the observed information matrix should be used to calculate the standard error for parameters, not the expected information matrix
cons_mat	Matrix containing coefficients for a system of linear constraints, formatted as matrix
cons_vec	Vector containing constants for a system of linear constraints, formatted as vector
norm	methods used to normalize the covariates. Default is 'null' for no normalization. Other options include 'max' to normalize by the absolute maximum and 'mean' to normalize by the mean
...	can include the named entries for the control list parameter

Value

returns a class fully describing the model and the regression results

See Also

Other Logistic Wrapper Functions: RunLogisticRegression_Omnibus()

Examples

```
library(data.table)
df <- data.table::data.table(
  "UserID" = c(112, 114, 213, 214, 115, 116, 117),
  "Starting_Age" = c(18, 20, 18, 19, 21, 20, 18),
  "Ending_Age" = c(30, 45, 57, 47, 36, 60, 55),
  "Cancer_Status" = c(0, 0, 1, 0, 1, 0, 0),
  "a" = c(0, 1, 1, 0, 1, 0, 1),
  "b" = c(1, 1.1, 2.1, 2, 0.1, 1, 0.2),
  "c" = c(10, 11, 10, 11, 12, 9, 11),
  "d" = c(0, 0, 0, 1, 1, 1, 1),
  "e" = c(0, 0, 1, 0, 0, 0, 1)
)
control <- list(
  "ncores" = 2, "lr" = 0.75, "maxiters" = c(1, 1),
  "halfmax" = 1
)
formula <- logit(Cancer_Status) ~
  loglinear(a, b, c, 0) + plinear(d, 0) + multiplicative()
res <- LogisticRun(formula, df, a_n = c(1.1, -0.1, 0.2, 0.5), control = control)
```

OMP_Check *Checks the OMP flag*

Description

OMP_Check Called directly from R, checks the omp flag and returns true if omp is enabled

Usage

```
OMP_Check()
```

Value

boolean: True for OMP allowed

---

plot.coxres *Performs Cox Proportional Hazard model plots*

Description

plot.coxres uses user provided data, time/event columns, vectors specifying the model, and options to choose and save plots

Usage

```
## S3 method for class 'coxres'
plot(x, df, plot_options, a_n = c(), ...)
```

Arguments

x	result object from a regression, class coxres
df	a data.table containing the columns of interest
plot_options	list of parameters controlling the plot options, see RunCoxPlots() for different options
a_n	list of initial parameter values, used to determine the number of parameters. May be either a list of vectors or a single vector.
...	can include the named entries for the plot_options parameter

Value

saves the plots in the current directory and returns the data used for plots

Examples

```
library(data.table)
## basic example code reproduced from the starting-description vignette
df <- data.table::data.table(
  "UserID" = c(112, 114, 213, 214, 115, 116, 117),
  "Starting_Age" = c(18, 20, 18, 19, 21, 20, 18),
  "Ending_Age" = c(30, 45, 57, 47, 36, 60, 55),
  "Cancer_Status" = c(0, 0, 1, 0, 1, 0, 0),
  "a" = c(0, 1, 1, 0, 1, 0, 1),
  "b" = c(1, 1.1, 2.1, 2, 0.1, 1, 0.2),
  "c" = c(10, 11, 10, 11, 12, 9, 11),
  "d" = c(0, 0, 0, 1, 1, 1, 1)
)
control <- list(
  "ncores" = 2, "lr" = 0.75, "maxiters" = c(1, 1),
  "halfmax" = 1
)
formula <- Cox(Starting_Age, Ending_Age, Cancer_Status) ~
  loglinear(a, b, c, 0) + plinear(d, 0) + multiplicative()
res <- CoxRun(formula, df,
  control = control,
  a_n = list(c(1.1, -0.1, 0.2, 0.5), c(1.6, -0.12, 0.3, 0.4))
)
plot_options <- list(
  "type" = c("surv", paste(tempfile(),
    "run",
    sep = ""
  )), "studyid" = "UserID",
  "verbose" = FALSE
)
plot(res, df, plot_options)
```

PoisRun *Fully runs a poisson regression model, returning the model and results*

Description

PoisRun uses a formula, data.table, and list of controls to prepare and run a Colossus poisson regression function

Usage

```
PoisRun(
  model,
  df,
  a_n = list(c(0)),
  keep_constant = c(0),
  control = list(),
  gradient_control = list(),
  single = FALSE,
  observed_info = FALSE,
  cons_mat = as.matrix(c(0)),
  cons_vec = c(0),
  norm = "null",
  ...
)
```

Arguments

model	either a formula written for the get_form function, or the model result from the get_form function.
df	a data.table containing the columns of interest
a_n	list of initial parameter values, used to determine the number of parameters. May be either a list of vectors or a single vector.
keep_constant	binary values to denote which parameters to change
control	list of parameters controlling the convergence, see the Control_Options vignette for details
gradient_control	a list of control options for the gradient descent algorithm. If any value is given, a gradient descent algorithm is used instead of Newton-Raphson. See the Control_Options vignette for details
single	a boolean to denote that only the log-likelihood should be calculated and returned, no derivatives or iterations
observed_info	a boolean to denote that the observed information matrix should be used to calculate the standard error for parameters, not the expected information matrix
cons_mat	Matrix containing coefficients for a system of linear constraints, formatted as matrix

cons_vec	Vector containing constants for a system of linear constraints, formatted as vector
norm	methods used to normalize the covariates. Default is 'null' for no normalization. Other options include 'max' to normalize by the absolute maximum and 'mean' to normalize by the mean
...	can include the named entries for the control list parameter

Value

returns a class fully describing the model and the regression results

See Also

Other Poisson Wrapper Functions: EventAssignment.poisres(), LikelihoodBound.poisres(), PoisRunJoint(), PoisRunMulti(), Residual.poisres(), RunPoissonRegression_Omnibus()

Examples

```
library(data.table)
df <- data.table::data.table(
  "UserID" = c(112, 114, 213, 214, 115, 116, 117),
  "Starting_Age" = c(18, 20, 18, 19, 21, 20, 18),
  "Ending_Age" = c(30, 45, 57, 47, 36, 60, 55),
  "Cancer_Status" = c(0, 0, 1, 0, 1, 0, 0),
  "a" = c(0, 1, 1, 0, 1, 0, 1),
  "b" = c(1, 1.1, 2.1, 2, 0.1, 1, 0.2),
  "c" = c(10, 11, 10, 11, 12, 9, 11),
  "d" = c(0, 0, 0, 1, 1, 1, 1),
  "e" = c(0, 0, 1, 0, 0, 0, 1)
)
control <- list(
  "ncores" = 2, "lr" = 0.75, "maxiters" = c(1, 1),
  "halfmax" = 1
)
formula <- Pois(Ending_Age, Cancer_Status) ~
  loglinear(a, b, c, 0) + plinear(d, 0) + multiplicative()
res <- PoisRun(formula, df, a_n = c(1.1, -0.1, 0.2, 0.5), control = control)
```

PoisRunJoint *Fully runs a joint poisson regression model, returning the model and results*

Description

PoisRunJoint uses a list of formula, data.table, and list of controls to prepare and run a Colossus poisson regression function on a joint dataset

Usage

```
PoisRunJoint(
  model,
  df,
  a_n = list(c(0)),
  keep_constant = c(0),
  control = list(),
  gradient_control = list(),
  single = FALSE,
  observed_info = FALSE,
  cons_mat = as.matrix(c(0)),
  cons_vec = c(0),
  norm = "null",
  ...
)
```

Arguments

model	either a formula written for the get_form function, or the model result from the get_form function.
df	a data.table containing the columns of interest
a_n	list of initial parameter values, used to determine the number of parameters. May be either a list of vectors or a single vector.
keep_constant	binary values to denote which parameters to change
control	list of parameters controlling the convergence, see the Control_Options vignette for details
gradient_control	a list of control options for the gradient descent algorithm. If any value is given, a gradient descent algorithm is used instead of Newton-Raphson. See the Control_Options vignette for details
single	a boolean to denote that only the log-likelihood should be calculated and returned, no derivatives or iterations
observed_info	a boolean to denote that the observed information matrix should be used to calculate the standard error for parameters, not the expected information matrix
cons_mat	Matrix containing coefficients for a system of linear constraints, formatted as matrix
cons_vec	Vector containing constants for a system of linear constraints, formatted as vector
norm	methods used to normalize the covariates. Default is 'null' for no normalization. Other options include 'max' to normalize by the absolute maximum and 'mean' to normalize by the mean
...	can include the named entries for the control list parameter

Value

returns a class fully describing the model and the regression results

See Also

Other Poisson Wrapper Functions: EventAssignment.poisres(), LikelihoodBound.poisres(), PoisRun(), PoisRunMulti(), Residual.poisres(), RunPoissonRegression_Omnibus()

Examples

```
library(data.table)
df <- data.table::data.table(
  "UserID" = c(112, 114, 213, 214, 115, 116, 117),
  "Starting_Age" = c(18, 20, 18, 19, 21, 20, 18),
  "Ending_Age" = c(30, 45, 57, 47, 36, 60, 55),
  "Cancer_Status" = c(0, 0, 1, 0, 1, 0, 0),
  "Flu_Status" = c(0, 1, 0, 0, 1, 0, 1),
  "a" = c(0, 1, 1, 0, 1, 0, 1),
  "b" = c(1, 1.1, 2.1, 2, 0.1, 1, 0.2),
  "c" = c(10, 11, 10, 11, 12, 9, 11),
  "d" = c(0, 0, 0, 1, 1, 1, 1),
  "e" = c(0, 0, 1, 0, 0, 0, 1)
)
control <- list(
  "ncores" = 2, "lr" = 0.75, "maxiters" = c(1, 1),
  "halfmax" = 1
)
formula_list <- list(Pois(Ending_Age, Cancer_Status) ~ plinear(d, 0),
  Pois(Ending_Age, Flu_Status) ~ loglinear(d, 0),
  "shared" = Pois(Ending_Age) ~ loglinear(a, b, c, 0)
)
res <- PoisRunJoint(formula_list, df, control = control)
```

PoisRunMulti *Fully runs a poisson regression model with multiple column realizations, returning the model and results*

Description

PoisRunMulti uses a formula, data.table, and list of controls to prepare and run a Colossus poisson regression function

Usage

```
PoisRunMulti(
  model,
  df,
  a_n = list(c(0)),
  keep_constant = c(0),
 realization_columns = matrix(c("temp00", "temp01", "temp10", "temp11"), nrow = 2),
  realization_index = c("temp0", "temp1"),
  control = list(),
```

```
  gradient_control = list(),
  single = FALSE,
  observed_info = FALSE,
  fma = FALSE,
  mcml = FALSE,
  cons_mat = as.matrix(c(0)),
  cons_vec = c(0),
  ...
)
```

Arguments

model	either a formula written for the get_form function, or the model result from the get_form function.
df	a data.table containing the columns of interest
a_n	list of initial parameter values, used to determine the number of parameters. May be either a list of vectors or a single vector.
keep_constant	binary values to denote which parameters to change
realization_columns	used for multi-realization regressions. Matrix of column names with rows for each column with realizations, columns for each realization
realization_index	used for multi-realization regressions. Vector of column names, one for each column with realizations. Each name should be used in the "names" variable in the equation definition
control	list of parameters controlling the convergence, see the Control_Options vignette for details
gradient_control	a list of control options for the gradient descent algorithm. If any value is given, a gradient descent algorithm is used instead of Newton-Raphson. See the Control_Options vignette for details
single	a boolean to denote that only the log-likelihood should be calculated and returned, no derivatives or iterations
observed_info	a boolean to denote that the observed information matrix should be used to calculate the standard error for parameters, not the expected information matrix
fma	a boolean to denote that the Frequentist Model Averaging method should be used
mcml	a boolean to denote that the Monte Carlo Maximum Likelihood method should be used
cons_mat	Matrix containing coefficients for a system of linear constraints, formatted as matrix
cons_vec	Vector containing constants for a system of linear constraints, formatted as vector
...	can include the named entries for the control list parameter

Value

returns a class fully describing the model and the regression results

See Also

Other Poisson Wrapper Functions: EventAssignment.poisres(), LikelihoodBound.poisres(), PoisRun(), PoisRunJoint(), Residual.poisres(), RunPoissonRegression_Omnibus()

Examples

```
library(data.table)
df <- data.table::data.table(
  "UserID" = c(112, 114, 213, 214, 115, 116, 117),
  "t0" = c(18, 20, 18, 19, 21, 20, 18),
  "t1" = c(30, 45, 57, 47, 36, 60, 55),
  "lung" = c(0, 0, 1, 0, 1, 0, 0),
  "dose" = c(0, 1, 1, 0, 1, 0, 1)
)
set.seed(3742)
df$rand <- floor(runif(nrow(df), min = 0, max = 5))
df$rand0 <- floor(runif(nrow(df), min = 0, max = 5))
df$rand1 <- floor(runif(nrow(df), min = 0, max = 5))
df$rand2 <- floor(runif(nrow(df), min = 0, max = 5))
names <- c("dose", "rand")
realization_columns <- matrix(c("rand0", "rand1", "rand2"), nrow = 1)
realization_index <- c("rand")
control <- list(
  "ncores" = 2, "lr" = 0.75, "maxiter" = 1,
  "halfmax" = 2, "epsilon" = 1e-6,
  "deriv_epsilon" = 1e-6, "step_max" = 1.0,
  "thres_step_max" = 100.0,
  "verbose" = 0, "ties" = "breslow", "double_step" = 1
)
formula <- Pois(t1, lung) ~ loglinear(CONST, dose, rand, 0) + multiplicative()
res <- PoisRun(formula, df, control = control)
```

---

print.caseconres *Prints a case-control regression output clearly*

Description

print.caseconres uses the list output from a regression, prints off a table of results and summarizes the score and convergence.

Usage

```
## S3 method for class 'caseconres'
print(x, ...)
```

Arguments

x	result object from a regression, class caseconres
...	can include the number of digits, named digit, or an unnamed integer entry assumed to be digits

Value

return nothing, prints the results to console

See Also

Other Output and Information Functions: System_Version(), print.coxres(), print.coxresbound(), print.logitres(), print.poisres(), print.poisresbound()

---

print.coxres *Prints a cox regression output clearly*

Description

print.coxres uses the list output from a regression, prints off a table of results and summarizes the score and convergence.

Usage

```
## S3 method for class 'coxres'
print(x, ...)
```

Arguments

x	result object from a regression, class coxres
...	can include the number of digits, named digit, or an unnamed integer entry assumed to be digits

Value

return nothing, prints the results to console

See Also

Other Output and Information Functions: System_Version(), print.caseconres(), print.coxresbound(), print.logitres(), print.poisres(), print.poisresbound()

print.coxresbound *Prints a cox likelihood boundary regression output clearly*

Description

print.coxresbound uses the list output from a regression, prints off a table of results and summarizes the score and convergence.

Usage

```
## S3 method for class 'coxresbound'
print(x, ...)
```

Arguments

x	result object from a regression, class coxresbound
...	can include the number of digits, named digit, or an unnamed integer entry assumed to be digits

Value

return nothing, prints the results to console

See Also

Other Output and Information Functions: System_Version(), print.caseconres(), print.coxres(), print.logitres(), print.poisres(), print.poisresbound()

print.logitres *Prints a logistic regression output clearly*

Description

print.logitres uses the list output from a regression, prints off a table of results and summarizes the score and convergence.

Usage

```
## S3 method for class 'logitres'
print(x, ...)
```

Arguments

x	result object from a regression, class logitres
...	can include the number of digits, named digit, or an unnamed integer entry assumed to be digits

Value

return nothing, prints the results to console

See Also

Other Output and Information Functions: System_Version(), print.caseconres(), print.coxres(), print.coxresbound(), print.poisres(), print.poisresbound()

`print.poisres` *Prints a poisson regression output clearly*

Description

`print.poisres` uses the list output from a regression, prints off a table of results and summarizes the score and convergence.

Usage

```
## S3 method for class 'poisres'
print(x, ...)
```

Arguments

`x`	result object from a regression, class poisres
`...`	can include the number of digits, named digit, or an unnamed integer entry assumed to be digits

Value

return nothing, prints the results to console

See Also

Other Output and Information Functions: System_Version(), print.caseconres(), print.coxres(), print.coxresbound(), print.logitres(), print.poisresbound()

print.poisresbound *Prints a poisson likelihood boundary regression output clearly*

Description

print.poisresbound uses the list output from a regression, prints off a table of results and summarizes the score and convergence.

Usage

```
## S3 method for class 'poisresbound'
print(x, ...)
```

Arguments

x	result object from a regression, class poisresbound
...	can include the number of digits, named digit, or an unnamed integer entry assumed to be digits

Value

return nothing, prints the results to console

See Also

Other Output and Information Functions: System_Version(), print.caseconres(), print.coxres(), print.coxresbound(), print.logitres(), print.poisres()

RelativeRisk *Generic relative risk calculation function*

Description

RelativeRisk Generic relative risk calculation function

Usage

```
RelativeRisk(x, df, ...)
```

Arguments

x	result object from a regression, class coxres
df	a data.table containing the columns of interest
...	extended for other necessary parameters

`RelativeRisk.coxres` *Calculates hazard ratios for a reference vector*

Description

`coxres.RelativeRisk` uses a cox result object and data, to evaluate relative risk in the data using the risk model from the result

Usage

```
## S3 method for class 'coxres'
RelativeRisk(x, df, a_n = c(), ...)
```

Arguments

`x`	result object from a regression, class coxres
`df`	a data.table containing the columns of interest
`a_n`	list of initial parameter values, used to determine the number of parameters. May be either a list of vectors or a single vector.
`...`	extended to match any future parameters needed

Value

returns a class fully describing the model and the regression results

Examples

```
library(data.table)
df <- data.table::data.table(
  "UserID" = c(112, 114, 213, 214, 115, 116, 117),
  "Starting_Age" = c(18, 20, 18, 19, 21, 20, 18),
  "Ending_Age" = c(30, 45, 57, 47, 36, 60, 55),
  "Cancer_Status" = c(0, 0, 1, 0, 1, 0, 0),
  "a" = c(0, 1, 1, 0, 1, 0, 1),
  "b" = c(1, 1.1, 2.1, 2, 0.1, 1, 0.2),
  "c" = c(10, 11, 10, 11, 12, 9, 11),
  "d" = c(0, 0, 0, 1, 1, 1, 1),
  "e" = c(0, 0, 1, 0, 0, 0, 1)
)
control <- list(
  "ncores" = 2, "lr" = 0.75, "maxiters" = c(1, 1),
  "halfmax" = 1
)
formula <- Cox(Starting_Age, Ending_Age, Cancer_Status) ~
  loglinear(a, b, c, 0) + plinear(d, 0) + multiplicative()
res <- CoxRun(formula, df,
  a_n = list(c(1.1, -0.1, 0.2, 0.5), c(1.6, -0.12, 0.3, 0.4)),
  control = control
```

```
)
RelativeRisk(res, df)
```

`RelativeRisk.default` *Generic relative risk calculation function, default option*

Description

`RelativeRisk.default` Generic relative risk calculation function, by default nothing happens

Usage

```
## Default S3 method:
RelativeRisk(x, df, ...)
```

Arguments

`x`	result object from a regression, class coxres
`df`	a data.table containing the columns of interest
`...`	extended for other necessary parameters

`Replace_Missing` *Automatically assigns missing values in listed columns*

Description

`Replace_Missing` checks each column and fills in NA values

Usage

```
Replace_Missing(df, name_list, msv, verbose = FALSE)
```

Arguments

`df`	a data.table containing the columns of interest
`name_list`	vector of string column names to check
`msv`	value to replace na with, same used for every column used
`verbose`	integer valued 0-4 controlling what information is printed to the terminal. Each level includes the lower levels. 0: silent, 1: errors printed, 2: warnings printed, 3: notes printed, 4: debug information printed. Errors are situations that stop the regression, warnings are situations that assume default values that the user might not have intended, notes provide information on regression progress, and debug prints out C++ progress and intermediate results. The default level is 2 and True/False is converted to 3/0.

Value

returns a filled datatable

See Also

Other Data Cleaning Functions: `Check_Iters()`, `Date_Shift()`, `Event_Count_Gen()`, `Event_Time_Gen()`, `Joint_Multiple_Events()`, `Time_Since()`, `apply_norm()`, `factorize()`, `gen_time_dep()`

Examples

```
library(data.table)
## basic example code reproduced from the starting-description vignette
df <- data.table::data.table(
  "UserID" = c(112, 114, 213, 214, 115, 116, 117),
  "Starting_Age" = c(18, 20, 18, 19, 21, 20, 18),
  "Ending_Age" = c(30, 45, NA, 47, 36, NA, 55),
  "Cancer_Status" = c(0, 0, 1, 0, 1, 0, 0)
)
df <- Replace_Missing(df, c("Starting_Age", "Ending_Age"), 70)
```

`Residual` *Generic Residual calculation function*

Description

`Residual` Generic Residual calculation function

Usage

```
Residual(x, df, ...)
```

Arguments

`x`	result object from a regression, class coxres or poisres
`df`	a data.table containing the columns of interest
`...`	extended for other necessary parameters

Residual.default *Generic Residual calculation function, default option*

Description

`Residual.default` Generic Residual calculation function, by default nothing happens

Usage

```
## Default S3 method:
Residual(x, df, ...)
```

Arguments

`x`	result object from a regression, class coxres or poisres
`df`	a data.table containing the columns of interest
`...`	extended for other necessary parameters

Residual.poisres *Calculates the Residuals for a completed poisson model*

Description

`Residual.poisres` uses user provided data, person-year/event columns, vectors specifying the model, and options to calculate residuals for a solved Poisson regression

Usage

```
## S3 method for class 'poisres'
Residual(
  x,
  df,
  control = list(),
  a_n = c(),
  pearson = FALSE,
  deviance = FALSE,
  ...
)
```

Arguments

x	result object from a regression, class poisres
df	a data.table containing the columns of interest
control	list of parameters controlling the convergence, see the Control_Options vignette for details
a_n	list of initial parameter values, used to determine the number of parameters. May be either a list of vectors or a single vector.
pearson	boolean to calculate pearson residuals
deviance	boolean to calculate deviance residuals
...	can include the named entries for the assign_control list parameter

Value

returns a list of the final results

See Also

Other Poisson Wrapper Functions: EventAssignment.poisres(), LikelihoodBound.poisres(), PoisRun(), PoisRunJoint(), PoisRunMulti(), RunPoissonRegression_Omnibus()

RunCaseControlRegression_Omnibus

Performs Matched Case-Control Conditional Logistic Regression

---

Description

RunCaseControlRegression_Omnibus uses user provided data, time/event columns, vectors specifying the model, and options to control the convergence and starting positions. Has additional options for starting with several initial guesses, using stratification and/or matching by time at risk, and calculation without derivatives

Usage

```
RunCaseControlRegression_Omnibus(
  df,
  time1 = "%trunc%",
  time2 = "%trunc%",
  event0 = "event",
  names = c("CONST"),
  term_n = c(0),
  tform = "loglin",
  keep_constant = c(0),
  a_n = c(0),
  modelform = "M",
  control = list(),
```

```
  strat_col = "null",
  cens_weight = "null",
  model_control = list(),
  cons_mat = as.matrix(c(0)),
  cons_vec = c(0)
)
```

Arguments

df	a data.table containing the columns of interest
time1	column used for time period starts
time2	column used for time period end
event0	column used for event status
names	columns for elements of the model, used to identify data columns
term_n	term numbers for each element of the model
tform	list of string function identifiers, used for linear/step
keep_constant	binary values to denote which parameters to change
a_n	list of initial parameter values, used to determine the number of parameters. May be either a list of vectors or a single vector.
modelform	string specifying the model type: M, ME, A, PA, PAE, GMIX, GMIX-R, GMIX-E
control	list of parameters controlling the convergence, see the Control_Options vignette for details
strat_col	column to stratify by if needed
cens_weight	column containing the row weights
model_control	controls which alternative model options are used, see the Control_Options vignette for further details
cons_mat	Matrix containing coefficients for a system of linear constraints, formatted as matrix
cons_vec	Vector containing constants for a system of linear constraints, formatted as vector

Value

returns a list of the final results

See Also

Other Case Control Wrapper Functions: `CaseControlRun()`

Examples

```
library(data.table)
## basic example code reproduced from the starting-description vignette
df <- data.table::data.table(
  "UserID" = c(112, 114, 213, 214, 115, 116, 117),
  "Starting_Age" = c(18, 20, 18, 19, 21, 20, 18),
  "Ending_Age" = c(30, 45, 57, 47, 36, 60, 55),
  "Cancer_Status" = c(0, 0, 1, 0, 1, 0, 0),
  "a" = c(0, 1, 1, 0, 1, 0, 1),
  "b" = c(1, 1.1, 2.1, 2, 0.1, 1, 0.2),
  "c" = c(10, 11, 10, 11, 12, 9, 11),
  "d" = c(0, 0, 0, 1, 1, 1, 1),
  "e" = c(0, 0, 1, 0, 0, 0, 1)
)
# For the interval case
time1 <- "Starting_Age"
time2 <- "Ending_Age"
event <- "Cancer_Status"
names <- c("a", "b", "c", "d")
a_n <- list(c(1.1, -0.1, 0.2, 0.5), c(1.6, -0.12, 0.3, 0.4))
# used to test at a specific point
term_n <- c(0, 1, 1, 2)
tform <- c("loglin", "lin", "lin", "plin")
modelform <- "M"
keep_constant <- c(0, 0, 0, 0)
control <- list(
  "ncores" = 2, "lr" = 0.75, "maxiters" = c(5, 5, 5),
  "halfmax" = 5, "epsilon" = 1e-3, "deriv_epsilon" = 1e-3,
  "step_max" = 1.0, "thres_step_max" = 100.0,
  "verbose" = FALSE,
  "ties" = "breslow", "double_step" = 1
)
e <- RunCaseControlRegression_Omnibus(df, time1, time2, event,
  names, term_n, tform, keep_constant,
  a_n, modelform, control,
  model_control = list(
    "stata" = FALSE,
    "time_risk" = FALSE
  )
)
```

RunCoxRegression_Omnibus

Performs Cox Proportional Hazards regression using the omnibus function

---

Description

RunCoxRegression_Omnibus uses user provided data, time/event columns, vectors specifying the model, and options to control the convergence and starting positions. Has additional options for

starting with several initial guesses, using stratification, multiplicative loglinear 1-term, competing risks, and calculation without derivatives

Usage

```
RunCoxRegression_Omnibus(
  df,
  time1 = "%trunc%",
  time2 = "%trunc%",
  event0 = "event",
  names = c("CONST"),
  term_n = c(0),
  tform = "loglin",
  keep_constant = c(0),
  a_n = c(0),
  modelform = "M",
  control = list(),
  strat_col = "null",
  cens_weight = "null",
  model_control = list(),
  cons_mat = as.matrix(c(0)),
  cons_vec = c(0)
)
```

Arguments

df	a data.table containing the columns of interest
time1	column used for time period starts
time2	column used for time period end
event0	column used for event status
names	columns for elements of the model, used to identify data columns
term_n	term numbers for each element of the model
tform	list of string function identifiers, used for linear/step
keep_constant	binary values to denote which parameters to change
a_n	list of initial parameter values, used to determine the number of parameters. May be either a list of vectors or a single vector.
modelform	string specifying the model type: M, ME, A, PA, PAE, GMIX, GMIX-R, GMIX-E
control	list of parameters controlling the convergence, see the Control_Options vignette for details
strat_col	column to stratify by if needed
cens_weight	column containing the row weights
model_control	controls which alternative model options are used, see the Control_Options vignette for further details

cons_mat	Matrix containing coefficients for a system of linear constraints, formatted as matrix
cons_vec	Vector containing constants for a system of linear constraints, formatted as vector

Value

returns a list of the final results

See Also

Other Cox Wrapper Functions: CoxRun(), CoxRunMulti(), LikelihoodBound.coxres()

Examples

```
library(data.table)
## basic example code reproduced from the starting-description vignette
df <- data.table::data.table(
  "UserID" = c(112, 114, 213, 214, 115, 116, 117),
  "Starting_Age" = c(18, 20, 18, 19, 21, 20, 18),
  "Ending_Age" = c(30, 45, 57, 47, 36, 60, 55),
  "Cancer_Status" = c(0, 0, 1, 0, 1, 0, 0),
  "a" = c(0, 1, 1, 0, 1, 0, 1),
  "b" = c(1, 1.1, 2.1, 2, 0.1, 1, 0.2),
  "c" = c(10, 11, 10, 11, 12, 9, 11),
  "d" = c(0, 0, 0, 1, 1, 1, 1),
  "e" = c(0, 0, 1, 0, 0, 0, 1)
)
# For the interval case
time1 <- "Starting_Age"
time2 <- "Ending_Age"
event <- "Cancer_Status"
names <- c("a", "b", "c", "d")
a_n <- list(c(1.1, -0.1, 0.2, 0.5), c(1.6, -0.12, 0.3, 0.4))
# used to test at a specific point
term_n <- c(0, 1, 1, 2)
tform <- c("loglin", "lin", "lin", "plin")
modelform <- "M"
keep_constant <- c(0, 0, 0, 0)
control <- list(
  "ncores" = 2, "lr" = 0.75, "maxiters" = c(5, 5, 5),
  "halfmax" = 5, "epsilon" = 1e-3, "deriv_epsilon" = 1e-3,
  "step_max" = 1.0, "thres_step_max" = 100.0,
  "verbose" = FALSE,
  "ties" = "breslow", "double_step" = 1, "guesses" = 2
)
e <- RunCoxRegression_Omnibus(df, time1, time2, event,
  names, term_n, tform, keep_constant,
  a_n, modelform, control,
  model_control = list(
    "single" = FALSE,
    "basic" = FALSE, "cr" = FALSE, "null" = FALSE
```

```
  )
)
```

`RunLogisticRegression_Omnibus` *Performs basic Logistic regression using the omnibus function*

Description

`RunLogisticRegression_Omnibus` uses user provided data, time/event columns, vectors specifying the model, and options to control the convergence and starting positions. Has additional options to starting with several initial guesses

Usage

```
RunLogisticRegression_Omnibus(
  df,
  trial0 = "CONST",
  event0 = "event",
  names = c("CONST"),
  term_n = c(0),
  tform = "loglin",
  keep_constant = c(0),
  a_n = c(0),
  modelform = "M",
  control = list(),
  model_control = list(),
  cons_mat = as.matrix(c(0)),
  cons_vec = c(0)
)
```

Arguments

`df`	a data.table containing the columns of interest
`trial0`	column with the number of trials per row, assumed to be 1 if a column not provided
`event0`	column used for event status
`names`	columns for elements of the model, used to identify data columns
`term_n`	term numbers for each element of the model
`tform`	list of string function identifiers, used for linear/step
`keep_constant`	binary values to denote which parameters to change
`a_n`	list of initial parameter values, used to determine the number of parameters. May be either a list of vectors or a single vector.
`modelform`	string specifying the model type: M, ME, A, PA, PAE, GMIX, GMIX-R, GMIX-E

control	list of parameters controlling the convergence, see the Control_Options vignette for details
model_control	controls which alternative model options are used, see the Control_Options vignette for further details
cons_mat	Matrix containing coefficients for a system of linear constraints, formatted as matrix
cons_vec	Vector containing constants for a system of linear constraints, formatted as vector

Value

returns a list of the final results

See Also

Other Logistic Wrapper Functions: LogisticRun()

Examples

```
library(data.table)
## basic example code reproduced from the starting-description vignette
df <- data.table::data.table(
  "Trials" = c(30, 45, 57, 47, 36, 60, 55),
  "Cancer_Status" = c(0, 0, 1, 0, 1, 0, 0),
  "a" = c(0, 1, 1, 0, 1, 0, 1),
  "b" = c(1, 1.1, 2.1, 2, 0.1, 1, 0.2),
  "c" = c(10, 11, 10, 11, 12, 9, 11),
  "d" = c(0, 0, 0, 1, 1, 1, 1),
  "e" = c(0, 0, 1, 0, 0, 0, 1)
)
# For the interval case
trial <- "Trials"
event <- "Cancer_Status"
names <- c("a", "b", "c", "d")
a_n <- c(1.1, -0.1, 0.2, 0.5) # used to test at a specific point
term_n <- c(0, 1, 1, 2)
tform <- c("loglin", "lin", "lin", "plin")
modelform <- "M"
keep_constant <- c(0, 0, 0, 0)
control <- list(
  "ncores" = 2, "lr" = 0.75, "maxiter" = 5,
  "halfmax" = 5, "epsilon" = 1e-3,
  "deriv_epsilon" = 1e-3, "step_max" = 1.0,
  "thres_step_max" = 100.0, "verbose" = FALSE, "ties" = "breslow",
  "double_step" = 1
)
strat_col <- "e"
e <- RunLogisticRegression_Omnibus(
  df, trial, event, names, term_n,
  tform, keep_constant,
  a_n, modelform,
```

```
  control
)
```

`RunPoissonRegression_Omnibus` *Performs basic Poisson regression using the omnibus function*

Description

`RunPoissonRegression_Omnibus` uses user provided data, time/event columns, vectors specifying the model, and options to control the convergence and starting positions. Has additional options to starting with several initial guesses

Usage

```
RunPoissonRegression_Omnibus(
  df,
  pyr0 = "pyr",
  event0 = "event",
  names = c("CONST"),
  term_n = c(0),
  tform = "loglin",
  keep_constant = c(0),
  a_n = c(0),
  modelform = "M",
  control = list(),
  strat_col = "null",
  model_control = list(),
  cons_mat = as.matrix(c(0)),
  cons_vec = c(0)
)
```

Arguments

`df`	a data.table containing the columns of interest
`pyr0`	column used for person-years per row
`event0`	column used for event status
`names`	columns for elements of the model, used to identify data columns
`term_n`	term numbers for each element of the model
`tform`	list of string function identifiers, used for linear/step
`keep_constant`	binary values to denote which parameters to change
`a_n`	list of initial parameter values, used to determine the number of parameters. May be either a list of vectors or a single vector.
`modelform`	string specifying the model type: M, ME, A, PA, PAE, GMIX, GMIX-R, GMIX-E

control	list of parameters controlling the convergence, see the Control_Options vignette for details
strat_col	column to stratify by if needed
model_control	controls which alternative model options are used, see the Control_Options vignette for further details
cons_mat	Matrix containing coefficients for a system of linear constraints, formatted as matrix
cons_vec	Vector containing constants for a system of linear constraints, formatted as vector

Value

returns a list of the final results

See Also

Other Poisson Wrapper Functions: EventAssignment.poisres(), LikelihoodBound.poisres(), PoisRun(), PoisRunJoint(), PoisRunMulti(), Residual.poisres()

Examples

```
library(data.table)
## basic example code reproduced from the starting-description vignette
df <- data.table::data.table(
  "UserID" = c(112, 114, 213, 214, 115, 116, 117),
  "Starting_Age" = c(18, 20, 18, 19, 21, 20, 18),
  "Ending_Age" = c(30, 45, 57, 47, 36, 60, 55),
  "Cancer_Status" = c(0, 0, 1, 0, 1, 0, 0),
  "a" = c(0, 1, 1, 0, 1, 0, 1),
  "b" = c(1, 1.1, 2.1, 2, 0.1, 1, 0.2),
  "c" = c(10, 11, 10, 11, 12, 9, 11),
  "d" = c(0, 0, 0, 1, 1, 1, 1),
  "e" = c(0, 0, 1, 0, 0, 0, 1)
)
# For the interval case
pyr <- "Ending_Age"
event <- "Cancer_Status"
names <- c("a", "b", "c", "d")
a_n <- c(1.1, -0.1, 0.2, 0.5) # used to test at a specific point
term_n <- c(0, 1, 1, 2)
tform <- c("loglin", "lin", "lin", "plin")
modelform <- "M"
keep_constant <- c(0, 0, 0, 0)
control <- list(
  "ncores" = 2, "lr" = 0.75, "maxiter" = 5,
  "halfmax" = 5, "epsilon" = 1e-3,
  "deriv_epsilon" = 1e-3, "step_max" = 1.0,
  "thres_step_max" = 100.0, "verbose" = FALSE, "ties" = "breslow",
  "double_step" = 1
)
```

```
strat_col <- "e"
e <- RunPoissonRegression_Omnibus(
  df, pyr, event, names, term_n,
  tform, keep_constant,
  a_n, modelform,
  control, strat_col
)
```

`System_Version` *Checks OS, compilers, and OMP*

Description

`System_Version` checks OS, default R c++ compiler, and if OMP is enabled

Usage

```
System_Version()
```

Value

returns a list of results

See Also

Other Output and Information Functions: `print.caseconres()`, `print.coxres()`, `print.coxresbound()`, `print.logitres()`, `print.poisres()`, `print.poisresbound()`

`Time_Since` *Automates creating a date since a reference column*

Description

`Time_Since` generates a new dataframe with a column containing time since a reference in a given unit

Usage

```
Time_Since(df, dcol0, tref, col_name, units = "days")
```

Arguments

`df`	a data.table containing the columns of interest
`dcol0`	list of ending month, day, and year
`tref`	reference time in date format
`col_name`	vector of new column names
`units`	time unit to use

Value

returns the updated dataframe

See Also

Other Data Cleaning Functions: Check_Iters(), Date_Shift(), Event_Count_Gen(), Event_Time_Gen(), Joint_Multiple_Events(), Replace_Missing(), apply_norm(), factorize(), gen_time_dep()

Examples

```
library(data.table)
m0 <- c(1, 1, 2, 2)
m1 <- c(2, 2, 3, 3)
d0 <- c(1, 2, 3, 4)
d1 <- c(6, 7, 8, 9)
y0 <- c(1990, 1991, 1997, 1998)
y1 <- c(2001, 2003, 2005, 2006)
df <- data.table::data.table(
  "m0" = m0, "m1" = m1,
  "d0" = d0, "d1" = d1,
  "y0" = y0, "y1" = y1
)
tref <- strptime("3-22-1997", format = "%m-%d-%Y", tz = "UTC")
df <- Time_Since(df, c("m1", "d1", "y1"), tref, "date_since")
```

Index