Package: Colossus (via r-universe)

October 7, 2026

Type Package

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

Version 1.6.3

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

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

Description Performs risk 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 <https://en.wikipedia.org/wiki/Proportional_hazards_model>, Poisson <https://en.wikipedia.org/wiki/Poisson_regression>, and Fine-Gray competing risks <https://www.publichealth.columbia.edu/research/population-health-methods/competing-risk-analysis> 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, pracma, withr

LinkingTo Rcpp, RcppEigen, testthat

SystemRequirements make, C++17

Encoding UTF-8

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

Depends R (>= 4.1.0)

Config/testthat/edition 3

VignetteBuilder knitr

Language en-US

Biarch TRUE

NeedsCompilation yes

Author Eric King-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 King-Giunta <egiunta@ksu.edu>

Config/roxygen2/version 8.1.0

Repository https://cran.r-universe.dev

Date/Publication 2026-10-07 01:39:33 UTC

RemoteUrl https://github.com/cran/Colossus

RemoteRef HEAD

RemoteSha f841c4537e8ccb19fc0c49af719a969e464be63b

Contents

CaseControlRun 3
CoxRun 5
CoxRunMulti 7
Date_Shift 9
Event_Count_Gen 10
Event_Time_Gen 11
EventAssignment 13
EventAssignment.poisres 14
EventAssignment.poisresbound 15
factorize 16
gen_time_dep 17
get_form 18
get_form_joint 19
Joint_Multiple_Events 20
Likelihood_Ratio_Test 21

LikelihoodBound . . . 22
LikelihoodBound.coxres . . . 23
LikelihoodBound.logitres . . . 24
LikelihoodBound.poisres . . . 25
Linked_Dose_Formula . . . 25
Linked_Lin_Exp_Para . . . 26
LogisticRun . . . 27
LogisticRunMulti . . . 29
LogisticRunMultiOut . . . 31
OMP_Check . . . 33
plot.coxres . . . 33
plotMartingale . . . 34
plotMartingale.coxres . . . 35
plotRisk . . . 36
plotRisk.coxres . . . 36
plotSchoenfeld . . . 37
plotSchoenfeld.coxres . . . 38
plotSurvival . . . 38
plotSurvival.coxres . . . 39
PoisRun . . . 40
PoisRunJoint . . . 42
PoisRunMulti . . . 44
PoisRunMultiOut . . . 46
RelativeRisk . . . 48
RelativeRisk.coxmodel . . . 48
RelativeRisk.coxres . . . 49
RelativeRisk.formula . . . 51
Replace_Missing . . . 52
Residual . . . 53
Residual.logitmodel . . . 53
Residual.logitres . . . 54
Residual.poismodel . . . 55
Residual.poisres . . . 56
System_Version . . . 57
Time_Since . . . 58

Index **59**

`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(0),
  keep_constant = 0,
  control = list(),
  conditional_threshold = 50,
  gradient_control = list(),
  single = FALSE,
  observed_info = FALSE,
  cons_mat = as.matrix(0),
  cons_vec = 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

Examples

```
library(data.table)
df <- 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 = 1, 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
)
```

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(0),
  keep_constant = 0,
  control = list(),
  gradient_control = list(),
  single = FALSE,
  observed_info = FALSE,
  cons_mat = as.matrix(0),
```

```
  cons_vec = 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()`

Examples

```
library(data.table)
df <- 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 = 1, 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(0),
  keep_constant = 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 = TRUE,
  mcml = FALSE,
  cons_mat = as.matrix(0),
  cons_vec = 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()`

Examples

```
library(data.table)
df <- 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))
realization_columns <- matrix(c("rand0", "rand1", "rand2"), nrow = 1)
realization_index <- c("rand")
control <- list(
  ncores = 1, lr = 0.75, maxiters = c(1, 1),
  halfmax = 1
)
formula <- Cox(t0, t1, lung) ~ loglinear(dose, rand, 0) + multiplicative()
res <- CoxRunMulti(formula, df,
  control = control,
  realization_columns = realization_columns,
  realization_index = realization_index
)
```

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: Joint_Multiple_Events(), Replace_Missing(), Time_Since(), 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(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")
```

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 = list(), events = list())
```

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

Value

returns a grouped table and a list of category boundaries used

See Also

Other Table Generation Functions: Event_Time_Gen()

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(
  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)
```

`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 = list(),
  time_scale = list(),
  categ = list(),
  summaries = list(),
  events = NULL,
  fcount = FALSE,
  lcount = FALSE,
  studyid = "studyID",
  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
time_scale	list with the time scale information, either a calendar category or an age category
categ	list with category columns and methods, methods can be either strings or lists of boundaries
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
fcount	boolean if the `first at risk` count should be returned in the data. Returns the number of observations starting in each combination of categories.
lcount	boolean if the `last at risk` count should be returned in the data. Returns the number of observations ending in each combination of categories.
studyid	id used to determine distinct subjects used for first and last at risk intervals.
verbose	boolean if updates should be printed to the console.

Value

returns a grouped table and a list of category boundaries used

See Also

Other Table Generation Functions: `Event_Count_Gen()`

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(
  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"
)
calendar_categ <- list(
  type = "calendar",
  day = c(1, 1, 1),
  month = c(1, 1, 1),
  year = c(1901, 1904, 1908)
)
time_scale <- list(time = calendar_categ)
summary <- list(
  c = "count AS cases"
)
events <- list("c")
pyr <- list(
  entry = list(year = "f"),
  exit = list(year = "i"),
  unit = "years"
)
e <- Event_Time_Gen(table, pyr, time_scale, categ, summary, events)
```

`EventAssignment` *Generic background/excess event calculation function*

Description

`EventAssignment` Generic background/excess event calculation function for standard poisson regressions and likelihood based confidence interval results.

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

Value

returns matrices of events assigned to background and excess

See Also

Other Poisson Event Assignment: `EventAssignment.poisres()`, `EventAssignment.poisresbound()`

`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 = NULL,
  ...
)
```

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 matrices of events assigned to background and excess

See Also

Other Poisson Event Assignment: `EventAssignment()`, `EventAssignment.poisresbound()`

EventAssignment.poisresbound *Predicts how many events are due to baseline vs excess for a completed poisson likelihood boundary regression*

Description

EventAssignment.poisresbound 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 'poisresbound'
EventAssignment(
  x,
  df,
  assign_control = list(),
  control = list(),
  a_n = NULL,
  ...
)
```

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 matrices of events assigned to background and excess

See Also

Other Poisson Event Assignment: EventAssignment(), EventAssignment.poisres()

`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: `Date_Shift()`, `Joint_Multiple_Events()`, `Replace_Missing()`, `Time_Since()`, `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(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: Date_Shift(), Joint_Multiple_Events(), Replace_Missing(), Time_Since(), 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(a = a, b = b, c = c)
time1 <- "%trunc%"
time2 <- "a"
event <- "c"
grt_f <- function(df, time_col) {
  return((df[, "b"] * df[, get(time_col)])[[1]])
}
func_form <- "lin"
df_new <- gen_time_dep(
  df, time1, time2, event, TRUE, 0.01, c("grt"), NULL,
  c(grt_f), paste0(tempfile(), "test_new.csv"), func_form, 1
)
```

---

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, nthreads = as.numeric(detectCores())/2)
```

Arguments

formula	a formula object, written in Colossus notation. See the unified equation expression vignette vignette for details.
df	a data.table containing the columns of interest
nthreads	number of threads to use, do not use more threads than available on your machine

Value

returns a class fully describing the model and the updated data

See Also

Other Formula Interpretation: `get_form_joint()`

Examples

```
library(data.table)
## basic example code reproduced from the starting-description vignette
df <- 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, 1)
```

---

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, nthreads = as.numeric(detectCores())/2)
```

Arguments

`formula_list`	a list of formula objects, each written in Colossus notation. See the unified equation expression vignette 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
`nthreads`	number of threads to use, do not use more threads than available on your machine

Value

returns a class fully describing the model and the updated data

See Also

Other Formula Interpretation: `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: Date_Shift(), Replace_Missing(), Time_Since(), 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 <- 0
term_n_e1 <- 0
term_n_shared <- c(0, 0)
tform_e0 <- "loglin"
tform_e1 <- "loglin"
tform_shared <- c("quad_slope", "loglin_top")
keep_constant_e0 <- 0
keep_constant_e1 <- 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
)
```

`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
a <- c(0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6)
b <- c(1, 2, 3, 4, 5, 6, 7, 1, 2, 3, 4, 5, 6, 7)
c <- c(1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0)
d <- c(3, 4, 5, 6, 7, 8, 9, 1, 2, 1, 1, 2, 1, 2)
e <- c(1, 2, 0, 0, 1, 2, 0, 0, 1, 2, 0, 0, 1, 2)
df <- data.table(a = a, b = b, c = c, d = d, e = e)
keep_constant <- 0
a_n <- c(-0.1, 0.1, 0.1, 0.2)
control <- list(ncores = 1, maxiter = 10, verbose = 0)
model <- Cox(a, b, c) ~ plinear(d * d, 0) + loglinear(factor(e))
alternative_model <- CoxRun(model, df,
  control = control,
  a_n = a_n, keep_constant = c(0, 1, 0)
)
null_model <- CoxRun(Cox(a, b, c) ~ null(), df, control = control)
score <- Likelihood_Ratio_Test(alternative_model, null_model)
```

`LikelihoodBound` *Generic likelihood boundary calculation function*

Description

`LikelihoodBound` Generic likelihood boundary calculation function. Used by the coxres, poisres, and logitres objects to calculate likelihood based confidence intervals.

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

Value

returns a summary of the likelihood boundary

See Also

Other Likelihood Boundaries: `LikelihoodBound.coxres()`, `LikelihoodBound.logitres()`, `LikelihoodBound.poisres`

`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 summary of the likelihood boundary

See Also

Other Likelihood Boundaries: `LikelihoodBound()`, `LikelihoodBound.logitres()`, `LikelihoodBound.poisres()`

`LikelihoodBound.logitres`

Calculates the likelihood boundary for a completed Logistic model

---

Description

`LikelihoodBound.logitres` 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 'logitres'
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 summary of the likelihood boundary

See Also

Other Likelihood Boundaries: `LikelihoodBound()`, `LikelihoodBound.coxres()`, `LikelihoodBound.poisres()`

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 summary of the likelihood boundary

See Also

Other Likelihood Boundaries: LikelihoodBound(), LikelihoodBound.coxres(), LikelihoodBound.logitres()

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)
```

Arguments

tforms	list of formula types
paras	list of formula parameters

Value

returns list of full parameters

See Also

Other Piece-wise Assistance Functions: Linked_Lin_Exp_Para()

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)
```

Arguments

y	point formula switch
a0	linear slope
a1_goal	exponential maximum desired

Value

returns parameter used by Colossus

See Also

Other Piece-wise Assistance Functions: Linked_Dose_Formula()

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(0),
  keep_constant = 0,
  control = list(),
  gradient_control = list(),
  link = "odds",
  single = FALSE,
  observed_info = FALSE,
  cons_mat = as.matrix(0),
  cons_vec = 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: LogisticRunMulti(), LogisticRunMultiOut()

Examples

```
library(data.table)
df <- 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 = 1, 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)
```

LogisticRunMulti *Fully runs a logistic regression model with multiple column realizations, returning the model and results*

Description

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

Usage

```
LogisticRunMulti(
  model,
  df,
  a_n = list(0),
  keep_constant = 0,
 realization_columns = matrix(c("temp00", "temp01", "temp10", "temp11"), nrow = 2),
  realization_index = c("temp0", "temp1"),
  control = list(),
  gradient_control = list(),
  link = "odds",
  single = FALSE,
  observed_info = FALSE,
  fma = TRUE,
  mcml = FALSE,
  cons_mat = as.matrix(0),
  cons_vec = 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
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
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 Logistic Wrapper Functions: LogisticRun(), LogisticRunMultiOut()

Examples

```
library(data.table)
df <- 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))
realization_columns <- matrix(c("rand0", "rand1", "rand2"), nrow = 1)
realization_index <- c("rand")
```

```
control <- list(
  ncores = 1, lr = 0.75, maxiters = c(1, 1),
  halfmax = 1
)
formula <- logit(lung) ~ loglinear(CONST, dose, rand, 0) + multiplicative()
res <- LogisticRunMulti(formula, df,
  control = control,
  realization_columns = realization_columns,
  realization_index = realization_index
)
```

`LogisticRunMultiOut` *Fully runs a logistic regression model with multiple event realizations, returning the model and results*

Description

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

Usage

```
LogisticRunMultiOut(
  model,
  df,
  a_n = list(0),
  keep_constant = 0,
  realization_columns = c("event0", "event1"),
  control = list(),
  gradient_control = list(),
  link = "odds",
  single = FALSE,
  observed_info = FALSE,
  cons_mat = as.matrix(0),
  cons_vec = 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

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

...
: 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: `LogisticRun()`, `LogisticRunMulti()`

Examples

```
library(data.table)
df <- 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),
  event = c(0, 0, 1, 0, 1, 0, 0),
  dose = c(0, 1, 1, 0, 1, 0, 1)
)
set.seed(3742)
df$event0 <- rbinom(nrow(df), size = 1, prob = 0.3)
df$event1 <- rbinom(nrow(df), size = 1, prob = 0.3)
df$event2 <- rbinom(nrow(df), size = 1, prob = 0.3)
realization_columns <- c("event0", "event1", "event2")
control <- list(
  ncores = 1, lr = 0.75, maxiter = 10,
  verbose = 0
)
```

```
formula <- logit(event) ~ loglinear(dose, 0) + multiplicative()
res <- LogisticRunMultiOut(formula, df,
  realization_columns = realization_columns,
  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

See Also

Other Output and Information Functions: `System_Version()`

`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. The 'plot_options' variable can be used to select the types of plots run.

Usage

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

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 the plotting vignette 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

returns the data used for plots

See Also

Other Plotting Wrapper Functions: plotMartingale.coxres(), plotRisk.coxres(), plotSchoenfeld.coxres(), plotSurvival.coxres()

Examples

```
library(data.table)
## basic example code reproduced from the starting-description vignette
df <- 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 = 1, 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", paste0(
    tempfile(),
    "run"
  )), studyid = "UserID",
  verbose = FALSE
)
res_plot <- plot(res, df, plot_options)
```

---

plotMartingale *Generic Martingale Residual Plotting function*

Description

plotMartingale Generic martingale residual plotting, used by the coxres object to plot martingale residuals. Currently only returns data that could be plotted.

Usage

```
plotMartingale(x, df, plot_options, a_n = NULL, ...)
```

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 the plotting vignette 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

returns the data used for plots

---

plotMartingale.coxres *Performs Cox Proportional Hazard model martingale residual plots*

Description

plotMartingale.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'
plotMartingale(x, df, plot_options, a_n = NULL, ...)
```

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 the plotting vignette 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

returns the data used for plots

See Also

Other Plotting Wrapper Functions: `plot.coxres()`, `plotRisk.coxres()`, `plotSchoenfeld.coxres()`, `plotSurvival.coxres()`

`plotRisk` *Generic Risk Plotting function*

Description

`plotRisk` Generic risk plotting function. Used by the coxres object to plot the hazard ratios at different levels of individual covariates. Currently only returns data that could be plotted.

Usage

```
plotRisk(x, df, plot_options, a_n = NULL, ...)
```

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 the plotting vignette 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

returns the data used for plots

`plotRisk.coxres` *Performs Cox Proportional Hazard model hazard ratio plots*

Description

`plotRisk.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'
plotRisk(x, df, plot_options, a_n = NULL, ...)
```

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 the plotting vignette 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

returns the data used for plots

See Also

Other Plotting Wrapper Functions: `plot.coxres()`, `plotMartingale.coxres()`, `plotSchoenfeld.coxres()`, `plotSurvival.coxres()`

---

plotSchoenfeld *Generic Schoenfeld Residual Plotting function*

Description

`plotSchoenfeld` Generic Schoenfeld residual plotting function, used by the coxres object to calculate Schoenfeld residuals. Currently only returns data that could be plotted.

Usage

```
plotSchoenfeld(x, df, plot_options, a_n = NULL, ...)
```

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 the plotting vignette 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

returns the data used for plots

plotSchoenfeld.coxres *Performs Cox Proportional Hazard model schoenfeld residual plots*

Description

plotSchoenfeld.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'
plotSchoenfeld(x, df, plot_options, a_n = NULL, ...)
```

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 the plotting vignette 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

returns the data used for plots

See Also

Other Plotting Wrapper Functions: plot.coxres(), plotMartingale.coxres(), plotRisk.coxres(), plotSurvival.coxres()

plotSurvival *Generic Survival Plotting function*

Description

plotSurvival Generic survival plotting, used by the coxres object. Calculates the baseline hazard, cumulative hazard, and survival curve for the list of event times. Currently only returns data that could be plotted.

Usage

```
plotSurvival(x, df, plot_options, a_n = NULL, ...)
```

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 the plotting vignette 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

returns the data used for plots

---

plotSurvival.coxres *Performs Cox Proportional Hazard model survival plots*

Description

plotSurvival.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'
plotSurvival(x, df, plot_options, a_n = NULL, ...)
```

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 the plotting vignette 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

returns the data used for plots

See Also

Other Plotting Wrapper Functions: plot.coxres(), plotMartingale.coxres(), plotRisk.coxres(), plotSchoenfeld.coxres()

Examples

```
library(data.table)
## basic example code reproduced from the starting-description vignette
df <- 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 = 1, 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(
  fname = paste0(
    tempfile(),
    "run"
  ), studyid = "UserID",
  verbose = FALSE
)
res_plot <- plotSurvival(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(0),
  keep_constant = 0,
  control = list(),
  gradient_control = list(),
```

```
  single = FALSE,
  observed_info = FALSE,
  cons_mat = as.matrix(0),
  cons_vec = 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: PoisRunJoint(), PoisRunMulti(), PoisRunMultiOut()

Examples

```
library(data.table)
df <- 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 = 1, 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(0),
  keep_constant = 0,
  control = list(),
  gradient_control = list(),
  single = FALSE,
  observed_info = FALSE,
  cons_mat = as.matrix(0),
  cons_vec = 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: PoisRun(), PoisRunMulti(), PoisRunMultiOut()

Examples

```
library(data.table)
df <- 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 = 1, 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(0),
  keep_constant = 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 = TRUE,
  mcml = FALSE,
  cons_mat = as.matrix(0),
  cons_vec = 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: PoisRun(), PoisRunJoint(), PoisRunMultiOut()

Examples

```
library(data.table)
df <- 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))
realization_columns <- matrix(c("rand0", "rand1", "rand2"), nrow = 1)
realization_index <- c("rand")
control <- list(
  ncores = 1, lr = 0.75, maxiters = c(1, 1),
  halfmax = 1
)
formula <- Pois(t1, lung) ~ loglinear(CONST, dose, rand, 0) + multiplicative()
res <- PoisRunMulti(formula, df,
  control = control,
  realization_columns = realization_columns,
  realization_index = realization_index
)
```

`PoisRunMultiOut` *Fully runs a poisson regression model with multiple event realizations, returning the model and results*

Description

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

Usage

```
PoisRunMultiOut(
  model,
  df,
  a_n = list(0),
  keep_constant = 0,
  realization_columns = c("event0", "event1"),
  control = list(),
  gradient_control = list(),
  single = FALSE,
  observed_info = FALSE,
  cons_mat = as.matrix(0),
  cons_vec = 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
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
...	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: PoisRun(), PoisRunJoint(), PoisRunMulti()

Examples

```
library(data.table)
df <- 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$rand <- c(1, floor(runif(nrow(df) - 1, min = 0, max = 1)))
df$lung0 <- c(1, floor(runif(nrow(df) - 1, min = 0, max = 1)))
df$lung1 <- c(1, floor(runif(nrow(df) - 1, min = 0, max = 1)))
df$lung2 <- c(1, floor(runif(nrow(df) - 1, min = 0, max = 1)))
realization_columns <- c("lung", "lung1", "lung2")
control <- list(
```

```
  ncores = 1, lr = 0.75, maxiters = c(1, 1),
  halfmax = 1
)
formula <- Pois(t1, lung) ~ loglinear(CONST, dose, rand, 0) + multiplicative()
res <- PoisRunMultiOut(formula, df,
  control = control,
  realization_columns = realization_columns
)
```

`RelativeRisk` *Generic relative risk calculation function*

Description

`RelativeRisk` Generic relative risk calculation function, currently only used by the coxres object, to calculate relative risks for either the last parameter or a custom point.

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

Value

returns matrices of the relatives risks

See Also

Other Predicted Risk and Rate: `RelativeRisk.coxmodel()`, `RelativeRisk.coxres()`, `RelativeRisk.formula()`

`RelativeRisk.coxmodel` *Calculates hazard ratios for a reference vector and model*

Description

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

Usage

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

Arguments

x	cox model object, which could be fed to a regression
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

See Also

Other Predicted Risk and Rate: RelativeRisk(), RelativeRisk.coxres(), RelativeRisk.formula()

Examples

```
library(data.table)
df <- 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 = 1, 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()
model <- get_form(formula, df)$model
res_risk <- RelativeRisk(model, df, a_n = c(1.1, -0.1, 0.2, 0.5))
```

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 = NULL, ...)
```

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

See Also

Other Predicted Risk and Rate: RelativeRisk(), RelativeRisk.coxmodel(), RelativeRisk.formula()

Examples

```
library(data.table)
df <- 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 = 1, 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
)
res_risk <- RelativeRisk(res, df)
```

`RelativeRisk.formula` *Calculates hazard ratios for a reference vector and model*

Description

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

Usage

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

Arguments

`x`	formula object, which could be fed to a regression
`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 matrices of the relatives risks

See Also

Other Predicted Risk and Rate: `RelativeRisk()`, `RelativeRisk.coxmodel()`, `RelativeRisk.coxres()`

Examples

```
library(data.table)
df <- 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 = 1, 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_risk <- RelativeRisk(formula, df, a_n = c(1.1, -0.1, 0.2, 0.5))
```

`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: `Date_Shift()`, `Joint_Multiple_Events()`, `Time_Since()`, `factorize()`, `gen_time_dep()`

Examples

```
library(data.table)
## basic example code reproduced from the starting-description vignette
df <- 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 for regression results. Note that the Cox residuals are calculated using the various plotting based functions.

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

Value

returns matrices of residuals

See Also

Other Residuals: `Residual.logitmodel()`, `Residual.logitres()`, `Residual.poismodel()`, `Residual.poisres()`

Residual.logitmodel *Calculates the Residuals for a logistic model*

Description

`Residual.logitmodel` 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 'logitmodel'
Residual(
  x,
  df,
  control = list(),
  a_n = NULL,
  link = "odds",
  pearson = FALSE,
  deviance = FALSE,
  ...
)
```

Arguments

x	result model for a regression, class logitmodel
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.
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.
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 matrices of residuals

See Also

Other Residuals: `Residual()`, `Residual.logitres()`, `Residual.poismodel()`, `Residual.poisres()`

`Residual.logitres` *Calculates the Residuals for a completed logistic model*

Description

`Residual.logitres` 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 'logitres'
Residual(
  x,
  df,
  control = list(),
  a_n = NULL,
  pearson = FALSE,
  deviance = FALSE,
  ...
)
```

Arguments

x	result object from a regression, class logitres
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 matrices of residuals

See Also

Other Residuals: `Residual()`, `Residual.logitmodel()`, `Residual.poismodel()`, `Residual.poisres()`

`Residual.poismodel` *Calculates the Residuals for a poisson formula*

Description

`Residual.poismodel` 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 'poismodel'
Residual(
  x,
  df,
  control = list(),
  a_n = NULL,
  pearson = FALSE,
  deviance = FALSE,
  ...
)
```

Arguments

x	result object from a poisson model, class poismodel
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 matrices of residuals

See Also

Other Residuals: Residual(), Residual.logitmodel(), Residual.logitres(), Residual.poisres()

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 = NULL,
  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 matrices of residuals

See Also

Other Residuals: Residual(), Residual.logitmodel(), Residual.logitres(), Residual.poismodel()

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: OMP_Check()

Examples

```
System_Version()
```

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: Date_Shift(), Joint_Multiple_Events(), Replace_Missing(), 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(
  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

∗ **Case Control Wrapper Functions**
CaseControlRun, 3
∗ **Cox Wrapper Functions**
CoxRun, 5
CoxRunMulti, 7
∗ **Data Cleaning Functions**
Date_Shift, 9
factorize, 16
gen_time_dep, 17
Joint_Multiple_Events, 20
Replace_Missing, 52
Time_Since, 58
∗ **Formula Interpretation**
get_form, 18
get_form_joint, 19
∗ **Likelihood Boundaries**
LikelihoodBound, 22
LikelihoodBound.coxres, 23
LikelihoodBound.logitres, 24
LikelihoodBound.poisres, 25
∗ **Logistic Wrapper Functions**
LogisticRun, 27
LogisticRunMulti, 29
LogisticRunMultiOut, 31
∗ **Output and Information Functions**
OMP_Check, 33
System_Version, 57
∗ **Piece-wise Assistance Functions**
Linked_Dose_Formula, 25
Linked_Lin_Exp_Para, 26
∗ **Plotting Wrapper Functions**
plot.coxres, 33
plotMartingale.coxres, 35
plotRisk.coxres, 36
plotSchoenfeld.coxres, 38
plotSurvival.coxres, 39
∗ **Poisson Event Assignment**
EventAssignment, 13
EventAssignment.poisres, 14
EventAssignment.poisresbound, 15
∗ **Poisson Wrapper Functions**
PoisRun, 40
PoisRunJoint, 42
PoisRunMulti, 44
PoisRunMultiOut, 46
∗ **Predicted Risk and Rate**
RelativeRisk, 48
RelativeRisk.coxmodel, 48
RelativeRisk.coxres, 49
RelativeRisk.formula, 51
∗ **Residuals**
Residual, 53
Residual.logitmodel, 53
Residual.logitres, 54
Residual.poismodel, 55
Residual.poisres, 56
∗ **Table Generation Functions**
Event_Count_Gen, 10
Event_Time_Gen, 11

CaseControlRun, 3
CoxRun, 5
CoxRun(), *8*
CoxRunMulti, 7
CoxRunMulti(), *6*

Date_Shift, 9
Date_Shift(), *16*, *18*, *21*, *52*, *58*

Event_Count_Gen, 10
Event_Count_Gen(), *12*
Event_Time_Gen, 11
Event_Time_Gen(), *10*
EventAssignment, 13
EventAssignment(), *14, 15*
EventAssignment.poisres, 14
EventAssignment.poisres(), *13*, *15*
EventAssignment.poisresbound, 15
EventAssignment.poisresbound(), *13, 14*

factorize, 16
factorize(), *10*, *18*, *21*, *52*, *58*

gen_time_dep, 17
gen_time_dep(), *10*, *16*, *21*, *52*, *58*
get_form, 18
get_form(), *20*
get_form_joint, 19
get_form_joint(), *19*

Joint_Multiple_Events, 20
Joint_Multiple_Events(), *10*, *16*, *18*, *52*, *58*

Likelihood_Ratio_Test, 21
LikelihoodBound, 22
LikelihoodBound(), *24, 25*
LikelihoodBound.coxres, 23
LikelihoodBound.coxres(), *23–25*
LikelihoodBound.logitres, 24
LikelihoodBound.logitres(), *23–25*
LikelihoodBound.poisres, 25
LikelihoodBound.poisres(), *23, 24*
Linked_Dose_Formula, 25
Linked_Dose_Formula(), *26*
Linked_Lin_Exp_Para, 26
Linked_Lin_Exp_Para(), *26*
LogisticRun, 27
LogisticRun(), *30*, *32*
LogisticRunMulti, 29
LogisticRunMulti(), *28*, *32*
LogisticRunMultiOut, 31
LogisticRunMultiOut(), *28*, *30*

OMP_Check, 33
OMP_Check(), *57*

plot.coxres, 33
plot.coxres(), *36–39*
plotMartingale, 34
plotMartingale.coxres, 35
plotMartingale.coxres(), *34*, *37–39*
plotRisk, 36
plotRisk.coxres, 36
plotRisk.coxres(), *34*, *36*, *38, 39*
plotSchoenfeld, 37
plotSchoenfeld.coxres, 38
plotSchoenfeld.coxres(), *34*, *36, 37*, *39*
plotSurvival, 38
plotSurvival.coxres, 39
plotSurvival.coxres(), *34*, *36–38*
PoisRun, 40
PoisRun(), *43*, *45*, *47*
PoisRunJoint, 42
PoisRunJoint(), *41*, *45*, *47*
PoisRunMulti, 44
PoisRunMulti(), *41*, *43*, *47*
PoisRunMultiOut, 46
PoisRunMultiOut(), *41*, *43*, *45*

RelativeRisk, 48
RelativeRisk(), *49–51*
RelativeRisk.coxmodel, 48
RelativeRisk.coxmodel(), *48*, *50, 51*
RelativeRisk.coxres, 49
RelativeRisk.coxres(), *48, 49*, *51*
RelativeRisk.formula, 51
RelativeRisk.formula(), *48–50*
Replace_Missing, 52
Replace_Missing(), *10*, *16*, *18*, *21*, *58*
Residual, 53
Residual(), *54–57*
Residual.logitmodel, 53
Residual.logitmodel(), *53*, *55–57*
Residual.logitres, 54
Residual.logitres(), *53, 54*, *56, 57*
Residual.poismodel, 55
Residual.poismodel(), *53–55*, *57*
Residual.poisres, 56
Residual.poisres(), *53–56*

System_Version, 57
System_Version(), *33*

Time_Since, 58
Time_Since(), *10*, *16*, *18*, *21*, *52*