# Package 'Colossus'

July 21, 2025

**Type** Package

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

**Version** 1.3.0

**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 <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, callr, stringr, processx, dplyr, tibble, lubridate

**LinkingTo** Rcpp, RcppEigen, testthat

**SystemRequirements** make

**RoxygenNote** 7.3.2

**Encoding** UTF-8

**Suggests** knitr, rmarkdown, testthat, xml2, ggplot2, pandoc, spelling, survival, 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],
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],
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** 2025-06-05 18:50:02 UTC

# Contents

---

`Check_Dupe_Columns` *checks for duplicated column names*

---

## Description

`Check_Dupe_Columns` checks for duplicated columns, columns with the same values, and columns with single value. Currently not updated for multi-terms

## Usage

```
Check_Dupe_Columns(df, cols, term_n, verbose = 0, factor_check = FALSE)
```

### Arguments

| | |
|---|---|
| df | a data.table containing the columns of interest |
| cols | columns to check |
| term_n | term numbers for each element of the model |
| 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. |
| factor_check | a boolean used to skip comparing columns of the form ?_? with the same initial string, which is used for factored columns |

### Value

returns the usable columns

### See Also

Other Data Cleaning Functions: Check_Trunc(), Check_Verbose(), Convert_Model_Eq(), Correct_Formula_Order(), Date_Shift(), Def_Control(), Def_Control_Guess(), Def_model_control(), Def_modelform_fix(), Event_Count_Gen(), Event_Time_Gen(), Joint_Multiple_Events(), Replace_Missing(), Time_Since(), factorize(), factorize_par(), gen_time_dep(), interact_them()

### 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)
cols <- c("a", "b", "c")
term_n <- c(0, 0, 1)
unique_cols <- Check_Dupe_Columns(df, cols, term_n)
```

---

## Check_Trunc *Applies time duration truncation limits to create columns for Cox model*

---

### Description

Check_Trunc creates columns to use for truncation

### Usage

```
Check_Trunc(df, ce, verbose = 0)
```

### Arguments

| | |
|---|---|
| df | a data.table containing the columns of interest |
| ce | columns to check for truncation, (t0, t1, event) |
| 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 the updated data and time period columns

### See Also

Other Data Cleaning Functions: Check_Dupe_Columns(), Check_Verbose(), Convert_Model_Eq(), Correct_Formula_Order(), Date_Shift(), Def_Control(), Def_Control_Guess(), Def_model_control(), Def_modelform_fix(), Event_Count_Gen(), Event_Time_Gen(), Joint_Multiple_Events(), Replace_Missing(), Time_Since(), factorize(), factorize_par(), gen_time_dep(), interact_them()

### 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)
)
# For the interval case
time1 <- "Starting_Age"
time2 <- "Ending_Age"
ce <- c("%trunc%", "Ending_Age")
val <- Check_Trunc(df, ce)
df <- val$df
ce <- val$ce
```

---

## Check_Verbose *General purpose verbosity check*

---

### Description

Check_Verbose checks and assigns verbosity values

### Usage

```
Check_Verbose(verbose)
```

### Arguments

| | |
|---|---|
| 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 correct verbose value

### See Also

Other Data Cleaning Functions: Check_Dupe_Columns(), Check_Trunc(), Convert_Model_Eq(), Correct_Formula_Order(), Date_Shift(), Def_Control(), Def_Control_Guess(), Def_model_control(), Def_modelform_fix(), Event_Count_Gen(), Event_Time_Gen(), Joint_Multiple_Events(), Replace_Missing(), Time_Since(), factorize(), factorize_par(), gen_time_dep(), interact_them()

---

## Convert_Model_Eq *Converts a string equation to regression model inputs*

---

### Description

Convert_Model_Eq Converts a string expression of a risk model into the vectors used by different Colossus regression functions

### Usage

```
Convert_Model_Eq(Model_Eq, df)
```

### Arguments

| | |
|---|---|
| Model_Eq | String representation of a survival model. Left-hand side details the model (cox, poisson, cox_strata, poisson_strata), time columns, event, and strata when used. The right-hand side details the subterm effects. The 'Unified Equation Representation' vignette provides more details. |
| df | a data.table containing the columns of interest |

### Value

returns a list of regression inputs

## See Also

Other Data Cleaning Functions: Check_Dupe_Columns(), Check_Trunc(), Check_Verbose(), Correct_Formula_Order(), Date_Shift(), Def_Control(), Def_Control_Guess(), Def_model_control(), Def_modelform_fix(), Event_Count_Gen(), Event_Time_Gen(), Joint_Multiple_Events(), Replace_Missing(), Time_Since(), factorize(), factorize_par(), gen_time_dep(), interact_them()

## 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)
table <- data.table::data.table(
  "a" = a, "b" = b, "c" = c,
  "d" = d, "e" = e
)
Model_Eq <- "cox(a,b, c) ~ loglinear(d, factor(e), 0) + multiplicative()"
e <- Convert_Model_Eq(Model_Eq, table)
```

---

# Correct_Formula_Order *Corrects the order of terms/formula/etc*

---

## Description

Correct_Formula_Order checks the order of formulas given and corrects any ordering issues, orders alphabetically, by term number, etc.

## Usage

```
Correct_Formula_Order(
  term_n,
  tform,
  keep_constant,
  a_n,
  names,
  cons_mat = matrix(c(0)),
  cons_vec = c(0),
  verbose = FALSE,
  model_control = list()
)
```

## Arguments

| | |
|---|---|
| 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. |
| names | columns for elements of the model, used to identify data columns |
| 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 |
| 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. |
| model_control | controls which alternative model options are used, see Def_model_control() for options and vignette("Control_Options") for further details |

## Value

returns the corrected lists

## See Also

Other Data Cleaning Functions: Check_Dupe_Columns(), Check_Trunc(), Check_Verbose(), Convert_Model_Eq(), Date_Shift(), Def_Control(), Def_Control_Guess(), Def_model_control(), Def_modelform_fix(), Event_Count_Gen(), Event_Time_Gen(), Joint_Multiple_Events(), Replace_Missing(), Time_Since(), factorize(), factorize_par(), gen_time_dep(), interact_them()

## Examples

```
library(data.table)
## basic example code reproduced from the starting-description vignette
term_n <- c(0, 1, 1, 0, 0)
tform <- c("loglin", "quad_slope", "lin", "lin_int", "lin_slope")
keep_constant <- c(0, 0, 0, 1, 0)
a_n <- c(1, 2, 3, 4, 5)
names <- c("a", "a", "a", "a", "a")
val <- Correct_Formula_Order(term_n, tform, keep_constant,
  a_n, names,
  cons_mat = matrix(c(0)),
  cons_vec = c(0)
)
term_n <- val$term_n
```

```
tform <- val$tform
keep_constant <- val$keep_constant
a_n <- val$a_n
names <- val$names
```

---

CoxCurveSolver *Calculates the likelihood curve for a cox model directly*

---

## Description

CoxCurveSolver solves the confidence interval for a cox model, starting at the optimum point and iteratively optimizing end-points of intervals. Intervals updated using the bisection method.

## Usage

```
CoxCurveSolver(
  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 Def_Control() for options or vignette("Control_Options") |
| 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 Def_model_control() for options and vignette("Control_Options") 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: RunCaseControlRegression_Omnibus(), RunCoxNull(), RunCoxRegression(), RunCoxRegression_Basic(), RunCoxRegression_CR(), RunCoxRegression_Guesses_CPP(), RunCoxRegression_Omnib
RunCoxRegression_Omnibus_Multidose(), RunCoxRegression_Single(), RunCoxRegression_Strata(), RunCoxRegression_Tier_Guesses()

---

# Cox_Relative_Risk *Calculates hazard ratios for a reference vector*

---

## Description

RunCoxRegression uses user provided data, vectors specifying the model, and options to calculate relative risk for every row in the provided data

## Usage

```
Cox_Relative_Risk(
  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(),
  model_control = list()
)
```

## 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 Def_Control() for options or vignette("Control_Options") |
| model_control | controls which alternative model options are used, see Def_model_control() for options and vignette("Control_Options") for further details |

## Value

returns a list of the final results

## See Also

Other Plotting Wrapper Functions: RunCoxPlots()

## 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)
)
```

```
# For the interval case
time1 <- "Starting_Age"
time2 <- "Ending_Age"
event <- "Cancer_Status"
names <- c("a", "b", "c", "d")
term_n <- c(0, 1, 1, 2)
tform <- c("loglin", "lin", "lin", "plin")
modelform <- "M"
a_n <- c(1.1, 0.1, 0.2, 0.5) # used to test at a specific point
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, "abs_max" = 1.0,
  "dose_abs_max" = 100.0, "verbose" = FALSE, "ties" = "breslow",
  "double_step" = 1
)
e <- Cox_Relative_Risk(
  df, time1, time2, event, names, term_n, tform,
  keep_constant, a_n, modelform, 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_Dupe_Columns(), Check_Trunc(), Check_Verbose(), Convert_Model_Eq(), Correct_Formula_Order(), Def_Control(), Def_Control_Guess(), Def_model_control(), Def_modelform_fix(), Event_Count_Gen(), Event_Time_Gen(), Joint_Multiple_Events(), Replace_Missing(), Time_Since(), factorize(), factorize_par(), gen_time_dep(), interact_them()

## 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")
```

---

# Def_Control *Automatically assigns missing control values*

---

## Description

Def_Control checks and assigns default values

## Usage

```
Def_Control(control)
```

## Arguments

| | |
|---|---|
| control | list of parameters controlling the convergence, see Def_Control() for options or vignette("Control_Options") |

## Value

returns a filled list

## See Also

Other Data Cleaning Functions: Check_Dupe_Columns(), Check_Trunc(), Check_Verbose(), Convert_Model_Eq(), Correct_Formula_Order(), Date_Shift(), Def_Control_Guess(), Def_model_control(), Def_modelform_fix(), Event_Count_Gen(), Event_Time_Gen(), Joint_Multiple_Events(), Replace_Missing(), Time_Since(), factorize(), factorize_par(), gen_time_dep(), interact_them()

## Examples

```
library(data.table)
control <- list(
  "ncores" = 2, "lr" = 0.75, "maxiter" = 5,
  "ties" = "breslow", "double_step" = 1
)
control <- Def_Control(control)
```

---

`Def_Control_Guess` *Automatically assigns missing guessing control values*

---

## Description

`Def_Control_Guess` checks and assigns default values

## Usage

```
Def_Control_Guess(guesses_control, a_n)
```

## Arguments

| | |
|---|---|
| `guesses_control` | list of parameters to control how the guessing works, see Def_Control_Guess() for options or vignette("Control_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. |

## Value

returns a filled list

## See Also

Other Data Cleaning Functions: `Check_Dupe_Columns()`, `Check_Trunc()`, `Check_Verbose()`, `Convert_Model_Eq()`, `Correct_Formula_Order()`, `Date_Shift()`, `Def_Control()`, `Def_model_control()`, `Def_modelform_fix()`, `Event_Count_Gen()`, `Event_Time_Gen()`, `Joint_Multiple_Events()`, `Replace_Missing()`, `Time_Since()`, `factorize()`, `factorize_par()`, `gen_time_dep()`, `interact_them()`

## Examples

```
library(data.table)
guesses_control <- list(
  "maxiter" = 10, "guesses" = 10,
  "loglin_min" = -1, "loglin_max" = 1, "loglin_method" = "uniform"
)
a_n <- c(0.1, 2, 1.3)
guesses_control <- Def_Control_Guess(guesses_control, a_n)
```

`Def_modelform_fix` *Automatically assigns geometric-mixture values and checks that a valid modelform is used*

## Description

`Def_modelform_fix` checks and assigns default values for modelform options

## Usage

```
Def_modelform_fix(control, model_control, modelform, term_n)
```

## Arguments

| | |
|---|---|
| `control` | list of parameters controlling the convergence, see Def_Control() for options or vignette("Control_Options") |
| `model_control` | controls which alternative model options are used, see Def_model_control() for options and vignette("Control_Options") for further details |
| `modelform` | string specifying the model type: M, ME, A, PA, PAE, GMIX, GMIX-R, GMIX-E |
| `term_n` | term numbers for each element of the model |

## Value

returns a filled list

## See Also

Other Data Cleaning Functions: `Check_Dupe_Columns()`, `Check_Trunc()`, `Check_Verbose()`, `Convert_Model_Eq()`, `Correct_Formula_Order()`, `Date_Shift()`, `Def_Control()`, `Def_Control_Guess()`, `Def_model_control()`, `Event_Count_Gen()`, `Event_Time_Gen()`, `Joint_Multiple_Events()`, `Replace_Missing()`, `Time_Since()`, `factorize()`, `factorize_par()`, `gen_time_dep()`, `interact_them()`

## Examples

```
library(data.table)
control <- list(
  "ncores" = 2, "lr" = 0.75, "maxiter" = 5,
  "ties" = "breslow", "double_step" = 1
)
control <- Def_Control(control)
model_control <- list("single" = TRUE)
model_control <- Def_model_control(model_control)
term_n <- c(0, 1, 1)
modelform <- "a"
val <- Def_modelform_fix(control, model_control, modelform, term_n)
model_control <- val$model_control
modelform <- val$modelform
```

## Def_model_control *Automatically assigns missing model control values*

### Description

Def_model_control checks and assigns default values

### Usage

```
Def_model_control(control)
```

### Arguments

| | |
|---|---|
| control | list of parameters controlling the convergence, see Def_Control() for options or vignette("Control_Options") |

### Value

returns a filled list

### See Also

Other Data Cleaning Functions: Check_Dupe_Columns(), Check_Trunc(), Check_Verbose(), Convert_Model_Eq(), Correct_Formula_Order(), Date_Shift(), Def_Control(), Def_Control_Guess(), Def_modelform_fix(), Event_Count_Gen(), Event_Time_Gen(), Joint_Multiple_Events(), Replace_Missing(), Time_Since(), factorize(), factorize_par(), gen_time_dep(), interact_them()

### Examples

```
library(data.table)
control <- list("single" = TRUE)
control <- Def_model_control(control)
```

## 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_Dupe_Columns(), Check_Trunc(), Check_Verbose(), Convert_Model_Eq(), Correct_Formula_Order(), Date_Shift(), Def_Control(), Def_Control_Guess(), Def_model_control(), Def_modelform_fix(), Event_Time_Gen(), Joint_Multiple_Events(), Replace_Missing(), Time_Since(), factorize(), factorize_par(), gen_time_dep(), interact_them()

## 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_Dupe_Columns()`, `Check_Trunc()`, `Check_Verbose()`, `Convert_Model_Eq()`, `Correct_Formula_Order()`, `Date_Shift()`, `Def_Control()`, `Def_Control_Guess()`, `Def_model_control()`, `Def_modelform_fix()`, `Event_Count_Gen()`, `Joint_Multiple_Events()`, `Replace_Missing()`, `Time_Since()`, `factorize()`, `factorize_par()`, `gen_time_dep()`, `interact_them()`

## 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_Dupe_Columns()`, `Check_Trunc()`, `Check_Verbose()`, `Convert_Model_Eq()`, `Correct_Formula_Order()`, `Date_Shift()`, `Def_Control()`, `Def_Control_Guess()`, `Def_model_control()`, `Def_modelform_fix()`, `Event_Count_Gen()`, `Event_Time_Gen()`, `Joint_Multiple_Events()`, `Replace_Missing()`, `Time_Since()`, `factorize_par()`, `gen_time_dep()`, `interact_them()`

## 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
```

factorize_par *Splits a parameter into factors in parallel*

## Description

factorize_par 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_par(df, col_list, verbose = 0, nthreads = as.numeric(detectCores()))
```

## 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. |
| nthreads | number of threads to use, do not use more threads than available on your machine |

## 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_Dupe_Columns(), Check_Trunc(), Check_Verbose(), Convert_Model_Eq(), Correct_Formula_Order(), Date_Shift(), Def_Control(), Def_Control_Guess(), Def_model_control(), Def_modelform_fix(), Event_Count_Gen(), Event_Time_Gen(), Joint_Multiple_Events(), Replace_Missing(), Time_Since(), factorize(), gen_time_dep(), interact_them()

## 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_par(df, col_list, FALSE, 2)
```

```
df <- val$df
new_col <- val$cols
```

---

`Gather_Guesses_CPP` *Performs checks to gather a list of guesses and iterations*

---

## Description

`Gather_Guesses_CPP` called from within R, uses a list of options and the model definition to generate a list of parameters and iterations that do not produce errors

## Usage

```
Gather_Guesses_CPP(
  df,
  dfc,
  names,
  term_n,
  tform,
  keep_constant,
  a_n,
  x_all,
  a_n_default,
  modelform,
  control,
  guesses_control,
  model_control = list()
)
```

## Arguments

| | |
|---|---|
| `df` | a data.table containing the columns of interest |
| `dfc` | vector matching subterm number to matrix column |
| `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. |
| `x_all` | covariate matrix |
| `a_n_default` | center of parameter distribution guessing scope |
| `modelform` | string specifying the model type: M, ME, A, PA, PAE, GMIX, GMIX-R, GMIX-E |

| | |
|---|---|
| control | list of parameters controlling the convergence, see Def_Control() for options or vignette("Control_Options") |
| guesses_control | list of parameters to control how the guessing works, see Def_Control_Guess() for options or vignette("Control_Options") |
| model_control | controls which alternative model options are used, see Def_model_control() for options and vignette("Control_Options") for further details |

## Value

returns a list of the final results

## 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(3, 4, 5, 6, 7, 8, 9)
df <- data.table::data.table("a" = a, "b" = b, "c" = c, "d" = d)
time1 <- "a"
time2 <- "b"
event <- "c"
names <- c("d")
term_n <- c(0)
tform <- c("loglin")
keep_constant <- c(0)
a_n <- c(-0.1)
a_n_default <- a_n
modelform <- "M"
control <- list(
  "ncores" = 2, "lr" = 0.75, "maxiter" = -1,
  "halfmax" = 5, "epsilon" = 1e-9,
  "deriv_epsilon" = 1e-9, "abs_max" = 1.0,
  "dose_abs_max" = 100.0, "verbose" = FALSE, "ties" = "breslow",
  "double_step" = 1
)
guesses_control <- list()
model_control <- list()
all_names <- unique(names(df))
dfc <- match(names, all_names)
term_tot <- max(term_n) + 1
x_all <- as.matrix(df[, all_names, with = FALSE])
control <- Def_Control(control)
guesses_control <- Def_Control_Guess(guesses_control, a_n)
model_control <- Def_model_control(model_control)
options(warn = -1)
Gather_Guesses_CPP(
  df, dfc, names, term_n, tform, keep_constant,
  a_n, x_all, a_n_default,
  modelform, control, guesses_control
)
```

## gcc_version *Checks default c++ compiler*

### Description

gcc_version Checks default c++ compiler, part of configuration script

### Usage

```
gcc_version()
```

### Value

returns a string representation of gcc, clang, or c++ output

## 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_Dupe_Columns(), Check_Trunc(), Check_Verbose(), Convert_Model_Eq(), Correct_Formula_Order(), Date_Shift(), Def_Control(), Def_Control_Guess(), Def_model_control(), Def_modelform_fix(), Event_Count_Gen(), Event_Time_Gen(), Joint_Multiple_Events(), Replace_Missing(), Time_Since(), factorize(), factorize_par(), interact_them()

## 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, "abs_max" = 1.0,
  "dose_abs_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")
```

---

`GetCensWeight` *Calculates and returns data for time by hazard and survival to estimate censoring rate*

---

## Description

`GetCensWeight` uses user provided data, time/event columns, vectors specifying the model, and options generate an estimate of the censoring rate, plots, and returns the data

## Usage

```
GetCensWeight(
  df,
  time1,
  time2,
  event0,
  names,
  term_n,
  tform,
  keep_constant,
  a_n,
  modelform,
  control,
  plot_options,
  model_control = list(),
  strat_col = "e"
)
```

## 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 Def_Control() for options or vignette("Control_Options") |
| plot_options | list of parameters controlling the plot options, see RunCoxPlots() for different options |
| model_control | controls which alternative model options are used, see Def_model_control() for options and vignette("Control_Options") for further details |
| strat_col | column to stratify by if needed |

## Value

saves the plots in the current directory and returns a data.table of time and corresponding hazard, cumulative hazard, and survival

## 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)
)
# For the interval case
time1 <- "Starting_Age"
time2 <- "Ending_Age"
event <- "Cancer_Status"
names <- c("a", "b", "c", "d")
term_n <- c(0, 1, 1, 2)
tform <- c("loglin", "lin", "lin", "plin")
modelform <- "M"
a_n <- c(0.1, 0.1, 0.1, 0.1)
keep_constant <- c(0, 0, 0, 0)
df$censor <- (df$Cancer_Status == 0)
event <- "censor"
control <- list(
  "ncores" = 2, "lr" = 0.75, "maxiter" = 20, "halfmax" = 5,
  "epsilon" = 1e-6, "deriv_epsilon" = 1e-6,
```

```
  "abs_max" = 1.0, "dose_abs_max" = 100.0, "verbose" = FALSE,
  "ties" = "breslow", "double_step" = 1
)
plot_options <- list(
  "name" = paste(tempfile(), "run_06", sep = ""), "verbose" = FALSE,
  "studyID" = "studyID", "age_unit" = "years"
)
dft <- GetCensWeight(
  df, time1, time2, event, names, term_n, tform,
  keep_constant, a_n, modelform, control, plot_options
)
t_ref <- dft$t
surv_ref <- dft$surv
t_c <- df$t1
cens_weight <- approx(t_ref, surv_ref, t_c, rule = 2)$y
```

---

`get_os` *Checks system OS*

---

## Description

`get_os` checks the system OS, part of configuration script

## Usage

```
get_os()
```

## Value

returns a string representation of OS

---

`interact_them` *Defines Interactions*

---

## Description

`interact_them` uses user provided interactions define interaction terms and update the data.table. assumes interaction is "+" or "*" and applies basic anti-aliasing to avoid duplicates

## Usage

```
interact_them(df, interactions, new_names, verbose = 0)
```

### Arguments

| | |
|---|---|
| df | a data.table containing the columns of interest |
| interactions | array of strings, each one is of form term1?*?term2" for term1 interaction of type * or + with term2, "?" dlimits |
| new_names | list of new names to use instead of default, default used if entry is ”" |
| 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_Dupe_Columns(), Check_Trunc(), Check_Verbose(), Convert_Model_Eq(), Correct_Formula_Order(), Date_Shift(), Def_Control(), Def_Control_Guess(), Def_model_control(), Def_modelform_fix(), Event_Count_Gen(), Event_Time_Gen(), Joint_Multiple_Events(), Replace_Missing(), Time_Since(), factorize(), factorize_par(), 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)
interactions <- c("a?+?b", "a?*?c")
new_names <- c("ab", "ac")
vals <- interact_them(df, interactions, new_names)
df <- vals$df
new_col <- vals$cols
```

---

## Interpret_Output *Prints a regression output clearly*

---

### Description

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

## Usage

```
Interpret_Output(out_list, digits = 2)
```

## Arguments

| | |
|---|---|
| out_list | list output from a regression, used to build results table and pull out convergence values |
| digits | digits used for printing results |

## Value

return nothing, prints the results to console

## See Also

Other Output and Information Functions: System_Version()

## 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,
  "abs_max" = 1.0, "dose_abs_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
  )
)
Interpret_Output(e)
```

---

`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_Dupe_Columns(), Check_Trunc(), Check_Verbose(), Convert_Model_Eq(), Correct_Formula_Order(), Date_Shift(), Def_Control(), Def_Control_Guess(), Def_model_control(), Def_modelform_fix(), Event_Count_Gen(), Event_Time_Gen(), Replace_Missing(), Time_Since(), factorize(), factorize_par(), gen_time_dep(), interact_them()

## 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
)
```

## 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)
```

---

# OMP_Check *Checks the OMP flag*

---

## Description

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

## Usage

```
OMP_Check()
```

## Value

boolean: True for OMP allowed

---

# PoissonCurveSolver *Calculates the likelihood curve for a poisson model directly*

---

## Description

PoissonCurveSolver solves the confidence interval for a poisson model, starting at the optimum point and iteratively optimizing each point to using the bisection method

## Usage

```
PoissonCurveSolver(
  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 Def_Control() for options or vignette("Control_Options") |
| strat_col | column to stratify by if needed |
| model_control | controls which alternative model options are used, see Def_model_control() for options and vignette("Control_Options") 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: RunPoissonEventAssignment(), RunPoissonEventAssignment_bound(), RunPoissonRegression(), RunPoissonRegression_Guesses_CPP(), RunPoissonRegression_Joint_Omnibus(), RunPoissonRegression_Omnibus(), RunPoissonRegression_Residual(), RunPoissonRegression_Single(), RunPoissonRegression_Strata(), RunPoissonRegression_Tier_Guesses()

## Rcomp_version *Checks how R was compiled*

### Description

Rcomp_version Checks how R was compiled, part of configuration script

### Usage

```
Rcomp_version()
```

### Value

returns a string representation of gcc, clang, or R CMD config CC output

## Rcpp_version *Checks default R c++ compiler*

### Description

Rcpp_version checks ~/.R/Makevars script for default compilers set, part of configuration script

### Usage

```
Rcpp_version()
```

### Value

returns a string representation of gcc, clang, or head ~/.R/Makevars

## 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_Dupe_Columns(), Check_Trunc(), Check_Verbose(), Convert_Model_Eq(), Correct_Formula_Order(), Date_Shift(), Def_Control(), Def_Control_Guess(), Def_model_control(), Def_modelform_fix(), Event_Count_Gen(), Event_Time_Gen(), Joint_Multiple_Events(), Time_Since(), factorize(), factorize_par(), gen_time_dep(), interact_them()

### 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)
```

---

## 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 Def_Control() for options or vignette("Control_Options") |
| 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 Def_model_control() for options and vignette("Control_Options") 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: CoxCurveSolver(), RunCoxNull(), RunCoxRegression(), RunCoxRegression_Basic(), RunCoxRegression_CR(), RunCoxRegression_Guesses_CPP(), RunCoxRegression_Omnibus(), RunCoxRegression_Omnibus_Multidose(), RunCoxRegression_Single(), RunCoxRegression_Strata(), RunCoxRegression_Tier_Guesses()

## 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,
  "abs_max" = 1.0, "dose_abs_max" = 100.0,
  "verbose" = FALSE,
  "ties" = "breslow", "double_step" = 1, "guesses" = 2
)
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
  )
)
```

RunCoxNull *Performs basic Cox Proportional Hazards regression with the null model*

## Description

RunCoxRegression uses user provided data and time/event columns to calculate the log-likelihood with constant hazard ratio

## Usage

```
RunCoxNull(
  df,
  time1 = "%trunc%",
  time2 = "%trunc%",
  event0 = "event",
  control = list()
)
```

## 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 |
| control | list of parameters controlling the convergence, see Def_Control() for options or vignette("Control_Options") |

## Value

returns a list of the final results

## See Also

Other Cox Wrapper Functions: CoxCurveSolver(), RunCaseControlRegression_Omnibus(), RunCoxRegression(), RunCoxRegression_Basic(), RunCoxRegression_CR(), RunCoxRegression_Guesses_CPP(), RunCoxRegression_Omnibus(), RunCoxRegression_Omnibus_Multidose(), RunCoxRegression_Single(), RunCoxRegression_Strata(), RunCoxRegression_Tier_Guesses()

## 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)
)
# For the interval case
time1 <- "Starting_Age"
time2 <- "Ending_Age"
event <- "Cancer_Status"
control <- list(
  "ncores" = 2, "verbose" = FALSE, "ties" = "breslow",
  "double_step" = 1
)
e <- RunCoxNull(df, time1, time2, event, control)
```

---

RunCoxPlots *Performs Cox Proportional Hazard model plots*

---

## Description

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

## Usage

```
RunCoxPlots(
  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(),
  plot_options = list(),
  model_control = list()
)
```

## 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 Def_Control() for options or vignette("Control_Options") |
| plot_options | list of parameters controlling the plot options, see RunCoxPlots() for different options |
| model_control | controls which alternative model options are used, see Def_model_control() for options and vignette("Control_Options") for further details |

## Value

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

## See Also

Other Plotting Wrapper Functions: Cox_Relative_Risk()

## 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)
)
# For the interval case
time1 <- "Starting_Age"
time2 <- "Ending_Age"
event <- "Cancer_Status"
names <- c("a", "b", "c", "d")
term_n <- c(0, 1, 1, 2)
tform <- c("loglin", "lin", "lin", "plin")
modelform <- "M"
a_n <- c(-0.1, 0.5, 1.1, -0.3)
keep_constant <- c(0, 0, 0, 0)
control <- list(
  "ncores" = 2, "lr" = 0.75, "maxiter" = -1, "halfmax" = 5,
  "epsilon" = 1e-3, "deriv_epsilon" = 1e-3,
  "abs_max" = 1.0, "dose_abs_max" = 100.0,
  "verbose" = FALSE, "ties" = "breslow", "double_step" = 1
```

```
)
# setting maxiter below 0 forces the function to calculate the score
# and return
plot_options <- list(
  "type" = c("surv", paste(tempfile(),
    "run",
    sep = ""
  )), "studyid" = "UserID",
  "verbose" = FALSE
)
RunCoxPlots(
  df, time1, time2, event, names, term_n, tform, keep_constant,
  a_n, modelform, control, plot_options
)
```

---

RunCoxRegression *Performs basic Cox Proportional Hazards regression without special options*

---

## Description

RunCoxRegression uses user provided data, time/event columns, vectors specifying the model, and options to control the convergence and starting position

## Usage

```
RunCoxRegression(
  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()
)
```

## 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 Def_Control() for options or vignette("Control_Options") |

## Value

returns a list of the final results

## See Also

Other Cox Wrapper Functions: CoxCurveSolver(), RunCaseControlRegression_Omnibus(), RunCoxNull(), RunCoxRegression_Basic(), RunCoxRegression_CR(), RunCoxRegression_Guesses_CPP(), RunCoxRegression_Omnibus(), RunCoxRegression_Omnibus_Multidose(), RunCoxRegression_Single(), RunCoxRegression_Strata(), RunCoxRegression_Tier_Guesses()

## 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)
)
# For the interval case
time1 <- "Starting_Age"
time2 <- "Ending_Age"
event <- "Cancer_Status"
names <- c("a", "b", "c", "d")
term_n <- c(0, 1, 1, 2)
tform <- c("loglin", "lin", "lin", "plin")
modelform <- "M"
a_n <- c(0.1, 0.1, 0.1, 0.1)
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,
```

```
  "abs_max" = 1.0, "dose_abs_max" = 100.0,
  "verbose" = FALSE, "ties" = "breslow", "double_step" = 1
)
e <- RunCoxRegression(
  df, time1, time2, event, names, term_n, tform,
  keep_constant, a_n, modelform, control
)
```

---

RunCoxRegression_Basic

*Performs basic Cox Proportional Hazards regression with a multiplicative log-linear model*

---

## Description

RunCoxRegression_Basic uses user provided data, time/event columns, vectors specifying the model, and options to control the convergence and starting positions

## Usage

```
RunCoxRegression_Basic(
  df,
  time1 = "%trunc%",
  time2 = "%trunc%",
  event0 = "event",
  names = c("CONST"),
  keep_constant = c(0),
  a_n = c(0),
  control = list()
)
```

## 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 |
| 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. |
| control | list of parameters controlling the convergence, see Def_Control() for options or vignette("Control_Options") |

## Value

returns a list of the final results

## See Also

Other Cox Wrapper Functions: CoxCurveSolver(), RunCaseControlRegression_Omnibus(), RunCoxNull(), RunCoxRegression(), RunCoxRegression_CR(), RunCoxRegression_Guesses_CPP(), RunCoxRegression_Omnibus(), RunCoxRegression_Omnibus_Multidose(), RunCoxRegression_Single(), RunCoxRegression_Strata(), RunCoxRegression_Tier_Guesses()

## 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)
)
# For the interval case
time1 <- "Starting_Age"
time2 <- "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
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, "abs_max" = 1.0,
  "dose_abs_max" = 100.0, "verbose" = FALSE,
  "ties" = "breslow", "double_step" = 1
)
e <- RunCoxRegression_Basic(
  df, time1, time2, event, names, keep_constant,
  a_n, control
)
```

---

RunCoxRegression_CR *Performs basic Cox Proportional Hazards regression with competing risks*

---

## Description

RunCoxRegression_CR uses user provided data, time/event columns, vectors specifying the model, and options to control the convergence, starting positions, and censoring adjustment

## Usage

```
RunCoxRegression_CR(
  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(),
  cens_weight = "null"
)
```

## 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 Def_Control() for options or vignette("Control_Options") |
| cens_weight | column containing the row weights |

## Value

returns a list of the final results

## See Also

Other Cox Wrapper Functions: CoxCurveSolver(), RunCaseControlRegression_Omnibus(), RunCoxNull(), RunCoxRegression(), RunCoxRegression_Basic(), RunCoxRegression_Guesses_CPP(), RunCoxRegression_Omnibus(), RunCoxRegression_Omnibus_Multidose(), RunCoxRegression_Single(), RunCoxRegression_Strata(), RunCoxRegression_Tier_Guesses()

## 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, 2, 1, 2, 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)
)
# For the interval case
time1 <- "Starting_Age"
time2 <- "Ending_Age"
event <- "Cancer_Status"
names <- c("a", "b", "c", "d")
term_n <- c(0, 1, 1, 2)
tform <- c("loglin", "lin", "lin", "plin")
modelform <- "M"
a_n <- c(0.1, 0.1, 0.1, 0.1)
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, "abs_max" = 1.0,
  "dose_abs_max" = 100.0, "verbose" = FALSE,
  "ties" = "breslow", "double_step" = 1
)
# weights the probability that a row would continue to extend without censoring,
#    for risk group calculation
df$cens_weight <- c(0.83, 0.37, 0.26, 0.34, 0.55, 0.23, 0.27)
# censoring weight is generated by the survival library finegray function, or by hand.
# The ratio of weight at event end point to weight at row endpoint is used.
e <- RunCoxRegression_CR(
  df, time1, time2, event, names, term_n, tform,
  keep_constant, a_n, modelform, control, "cens_weight"
)
```

---

`RunCoxRegression_Guesses_CPP`

*Performs basic Cox Proportional Hazards regression, Generates multiple starting guesses on c++ side*

---

## Description

`RunCoxRegression_Guesses_CPP` 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

```
RunCoxRegression_Guesses_CPP(
  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(),
  guesses_control = list(),
  strat_col = "null",
  model_control = list(),
  cens_weight = "null"
)
```

## 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 Def_Control() for options or vignette("Control_Options") |
| guesses_control | list of parameters to control how the guessing works, see Def_Control_Guess() for options or vignette("Control_Options") |
| strat_col | column to stratify by if needed |
| model_control | controls which alternative model options are used, see Def_model_control() for options and vignette("Control_Options") for further details |
| cens_weight | column containing the row weights |

## Value

returns a list of the final results

## See Also

Other Cox Wrapper Functions: CoxCurveSolver(), RunCaseControlRegression_Omnibus(), RunCoxNull(), RunCoxRegression(), RunCoxRegression_Basic(), RunCoxRegression_CR(), RunCoxRegression_Omnibus(), RunCoxRegression_Omnibus_Multidose(), RunCoxRegression_Single(), RunCoxRegression_Strata(), RunCoxRegression_Tier_Guesses()

## 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 <- 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, "abs_max" = 1.0,
  "dose_abs_max" = 100.0, "verbose" = FALSE, "ties" = "breslow",
  "double_step" = 1
)
guesses_control <- list(
  "maxiter" = 10, "guesses" = 10,
  "lin_min" = 0.001, "lin_max" = 1,
  "loglin_min" = -1, "loglin_max" = 1,
  "lin_method" = "uniform",
  "loglin_method" = "uniform", strata = FALSE
)
strat_col <- "e"
options(warn = -1)
e <- RunCoxRegression_Guesses_CPP(
```

```
  df, time1, time2, event, names, term_n,
  tform, keep_constant, a_n, modelform,
  control, guesses_control, strat_col
)
```

---

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 Def_Control() for options or vignette("Control_Options") |
| 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 Def_model_control() for options and vignette("Control_Options") 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: CoxCurveSolver(), RunCaseControlRegression_Omnibus(), RunCoxNull(), RunCoxRegression(), RunCoxRegression_Basic(), RunCoxRegression_CR(), RunCoxRegression_Guesses_CPP(), RunCoxRegression_Omnibus_Multidose(), RunCoxRegression_Single(), RunCoxRegression_Strata(), RunCoxRegression_Tier_Guesses()

## 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,
  "abs_max" = 1.0, "dose_abs_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
  )
)
```

---

RunCoxRegression_Omnibus_Multidose *Performs Cox Proportional Hazards regression using the omnibus function with multiple column realizations*

---

## Description

RunCoxRegression_Omnibus_Multidose uses user provided data, time/event columns, vectors specifying the model, and options to control the convergence and starting positions. Used for 2DMC column uncertainty methods. Returns optimized parameters, log-likelihood, and standard deviation for each realization. Has additional options for using stratification, multiplicative loglinear 1-term, competing risks, and calculation without derivatives

## Usage

```
RunCoxRegression_Omnibus_Multidose(
  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",
 realization_columns = matrix(c("temp00", "temp01", "temp10", "temp11"), nrow = 2),
```

```
  realization_index = c("temp0", "temp1"),
  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 |
| 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 Def_Control() for options or vignette("Control_Options") |
| 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 Def_model_control() for options and vignette("Control_Options") 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 for each realization

## See Also

Other Cox Wrapper Functions: CoxCurveSolver(), RunCaseControlRegression_Omnibus(), RunCoxNull(), RunCoxRegression(), RunCoxRegression_Basic(), RunCoxRegression_CR(), RunCoxRegression_Guesses_CPP(), RunCoxRegression_Omnibus(), RunCoxRegression_Single(), RunCoxRegression_Strata(), RunCoxRegression_Tier_Guesses()

## 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),
  "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))
time1 <- "t0"
time2 <- "t1"
names <- c("dose", "rand")
term_n <- c(0, 0)
tform <- c("loglin", "loglin")
realization_columns <- matrix(c("rand0", "rand1", "rand2"), nrow = 1)
realization_index <- c("rand")
keep_constant <- c(1, 0)
a_n <- c(0, 0)
modelform <- "M"
cens_weight <- c(0)
event <- "lung"
a_n <- c(-0.1, -0.1)
keep_constant <- c(0, 0)
control <- list(
  "ncores" = 2, "lr" = 0.75, "maxiter" = 1,
  "halfmax" = 2, "epsilon" = 1e-6,
  "deriv_epsilon" = 1e-6, "abs_max" = 1.0,
  "dose_abs_max" = 100.0,
  "verbose" = 0, "ties" = "breslow", "double_step" = 1
)
e <- RunCoxRegression_Omnibus_Multidose(df, time1, time2, event,
  names,
  term_n = term_n, tform = tform,
  keep_constant = keep_constant, a_n = a_n,
  modelform = modelform,
  realization_columns = realization_columns,
  realization_index = realization_index,
  control = control, strat_col = "fac",
  model_control = list(), cens_weight = "null"
```

```
)
```

---

`RunCoxRegression_Single` *Performs basic Cox Proportional Hazards calculation with no derivative*

---

## Description

`RunCoxRegression_Single` uses user provided data, time/event columns, vectors specifying the model, and options and returns the log-likelihood

## Usage

```
RunCoxRegression_Single(
  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()
)
```

## 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 Def_Control() for options or vignette("Control_Options") |

## Value

returns a list of the final results

## See Also

Other Cox Wrapper Functions: CoxCurveSolver(), RunCaseControlRegression_Omnibus(), RunCoxNull(), RunCoxRegression(), RunCoxRegression_Basic(), RunCoxRegression_CR(), RunCoxRegression_Guesses_CPP(), RunCoxRegression_Omnibus(), RunCoxRegression_Omnibus_Multidose(), RunCoxRegression_Strata(), RunCoxRegression_Tier_Guesses()

## 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)
)
# For the interval case
time1 <- "Starting_Age"
time2 <- "Ending_Age"
event <- "Cancer_Status"
names <- c("a", "b", "c", "d")
term_n <- c(0, 1, 1, 2)
tform <- c("loglin", "lin", "lin", "plin")
modelform <- "M"
a_n <- c(1.1, -0.1, 0.2, 0.5) # used to test at a specific point
keep_constant <- c(0, 0, 0, 0)
control <- list(
  "ncores" = 2, "verbose" = FALSE,
  "ties" = "breslow", "double_step" = 1
)
e <- RunCoxRegression_Single(
  df, time1, time2, event, names, term_n, tform,
  keep_constant, a_n, modelform, control
)
```

---

RunCoxRegression_Strata *Performs basic Cox Proportional Hazards regression with strata effect*

---

## Description

RunCoxRegression_Strata uses user provided data, time/event columns, vectors specifying the model, and options to control the convergence and starting positions

## Usage

```
RunCoxRegression_Strata(
  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"
)
```

## 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 Def_Control() for options or vignette("Control_Options") |
| strat_col | column to stratify by if needed |

## Value

returns a list of the final results

## See Also

Other Cox Wrapper Functions: CoxCurveSolver(), RunCaseControlRegression_Omnibus(), RunCoxNull(), RunCoxRegression(), RunCoxRegression_Basic(), RunCoxRegression_CR(), RunCoxRegression_Guesses_CPP(), RunCoxRegression_Omnibus(), RunCoxRegression_Omnibus_Multidose(), RunCoxRegression_Single(), RunCoxRegression_Tier_Guesses()

## 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, 0, 0, 1, 0, 1)
)
# For the interval case
time1 <- "Starting_Age"
time2 <- "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,
  "abs_max" = 1.0, "dose_abs_max" = 100.0,
  "verbose" = FALSE, "ties" = "breslow", "double_step" = 1
)
strat_col <- "e"
e <- RunCoxRegression_Strata(
  df, time1, time2, event, names, term_n,
  tform, keep_constant, a_n, modelform,
  control, strat_col
)
```

---

RunCoxRegression_Tier_Guesses *Performs basic cox regression, with multiple guesses, starts with solving for a single term*

---

## Description

RunCoxRegression_Tier_Guesses uses user provided data, time/event columns, vectors specifying the model, and options to control the convergence and starting positions, with additional guesses

## Usage

```
RunCoxRegression_Tier_Guesses(
  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(),
  guesses_control = list(),
  strat_col = "null",
  model_control = list(),
  cens_weight = "null"
)
```

## 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 Def_Control() for options or vignette("Control_Options") |
| guesses_control | list of parameters to control how the guessing works, see Def_Control_Guess() for options or vignette("Control_Options") |
| strat_col | column to stratify by if needed |

| | |
|---|---|
| model_control | controls which alternative model options are used, see Def_model_control() for options and vignette("Control_Options") for further details |
| cens_weight | column containing the row weights |

## Value

returns a list of the final results

## See Also

Other Cox Wrapper Functions: CoxCurveSolver(), RunCaseControlRegression_Omnibus(), RunCoxNull(), RunCoxRegression(), RunCoxRegression_Basic(), RunCoxRegression_CR(), RunCoxRegression_Guesses_CPP(), RunCoxRegression_Omnibus(), RunCoxRegression_Omnibus_Multidose(), RunCoxRegression_Single(), RunCoxRegression_Strata()

## 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, 0, 0, 1, 0, 1)
)
# For the interval case
time1 <- "Starting_Age"
time2 <- "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,
  "abs_max" = 1.0, "dose_abs_max" = 100.0,
  "verbose" = FALSE, "ties" = "breslow", "double_step" = 1
)
guesses_control <- list(
  "iterations" = 10, "guesses" = 10, "lin_min" = 0.001,
  "lin_max" = 1, "loglin_min" = -1, "loglin_max" = 1, "lin_method" = "uniform",
  "loglin_method" = "uniform", strata = TRUE, term_initial = c(0, 1)
)
strat_col <- "e"
```

```
options(warn = -1)
e <- RunCoxRegression_Tier_Guesses(
  df, time1, time2, event, names,
  term_n, tform, keep_constant,
  a_n, modelform,
  control, guesses_control,
  strat_col
)
```

---

`RunPoissonEventAssignment` *Predicts how many events are due to baseline vs excess*

---

## Description

`RunPoissonEventAssignment` uses user provided data, person-year/event columns, vectors specifying the model, and options to calculate background and excess events

## Usage

```
RunPoissonEventAssignment(
  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()
)
```

## 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 Def_Control() for options or vignette("Control_Options") |
| strat_col | column to stratify by if needed |
| model_control | controls which alternative model options are used, see Def_model_control() for options and vignette("Control_Options") for further details |

## Value

returns a list of the final results

## See Also

Other Poisson Wrapper Functions: PoissonCurveSolver(), RunPoissonEventAssignment_bound(), RunPoissonRegression(), RunPoissonRegression_Guesses_CPP(), RunPoissonRegression_Joint_Omnibus(), RunPoissonRegression_Omnibus(), RunPoissonRegression_Residual(), RunPoissonRegression_Single(), RunPoissonRegression_Strata(), RunPoissonRegression_Tier_Guesses()

## 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)
)
# For the interval case
df$pyr <- df$Ending_Age - df$Starting_Age
pyr <- "pyr"
event <- "Cancer_Status"
names <- c("a", "b", "c", "d")
term_n <- c(0, 1, 1, 2)
tform <- c("loglin", "lin", "lin", "plin")
modelform <- "M"
a_n <- c(0.1, 0.1, 0.1, 0.1)
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, "abs_max" = 1.0,
  "dose_abs_max" = 100.0, "verbose" = FALSE, "double_step" = 1
```

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

---

RunPoissonEventAssignment_bound *Predicts how many events are due to baseline vs excess at the confidence bounds of a single parameter*

---

## Description

RunPoissonEventAssignment_bound uses user provided data, the results of a poisson regression, and options to calculate background and excess events

## Usage

```
RunPoissonEventAssignment_bound(
  df,
  pyr0 = "pyr",
  event0 = "event",
  alternative_model = list(),
  keep_constant = c(0),
  modelform = "M",
  check_num = 1,
  z = 2,
  control = list(),
  strat_col = "null",
  model_control = list()
)
```

## Arguments

| | |
|---|---|
| df | a data.table containing the columns of interest |
| pyr0 | column used for person-years per row |
| event0 | column used for event status |
| alternative_model | the new model of interest in list form, output from a Poisson regression |
| keep_constant | binary values to denote which parameters to change |
| modelform | string specifying the model type: M, ME, A, PA, PAE, GMIX, GMIX-R, GMIX-E |
| check_num | the parameter number to check at the bounds of, indexed from 1 using the order returned by Colossus |
| z | Z score to use for confidence interval |

| | |
|---|---|
| control | list of parameters controlling the convergence, see Def_Control() for options or vignette("Control_Options") |
| strat_col | column to stratify by if needed |
| model_control | controls which alternative model options are used, see Def_model_control() for options and vignette("Control_Options") for further details |

## Value

returns a list of the final results

## See Also

Other Poisson Wrapper Functions: PoissonCurveSolver(), RunPoissonEventAssignment(), RunPoissonRegression(), RunPoissonRegression_Guesses_CPP(), RunPoissonRegression_Joint_Omnibus(), RunPoissonRegression_Omnibus(), RunPoissonRegression_Residual(), RunPoissonRegression_Single(), RunPoissonRegression_Strata(), RunPoissonRegression_Tier_Guesses()

## 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, "abs_max" = 1.0,
  "dose_abs_max" = 100.0, "verbose" = FALSE, "ties" = "breslow",
  "double_step" = 1
)
guesses_control <- list(
  "maxiter" = 10, "guesses" = 10, "lin_min" = 0.001,
  "lin_max" = 1, "loglin_min" = -1, "loglin_max" = 1, "lin_method" = "uniform",
  "loglin_method" = "uniform", strata = FALSE
```

```
)
strat_col <- "e"
e0 <- RunPoissonRegression_Omnibus(
  df, pyr, event, names, term_n, tform,
  keep_constant,
  a_n, modelform,
  control, strat_col
)
e <- RunPoissonEventAssignment_bound(
  df, pyr, event, e0, keep_constant,
  modelform, 4, 2, control
)
```

---

`RunPoissonRegression` *Performs basic poisson regression*

---

## Description

`RunPoissonRegression` uses user provided data, person-year/event columns, vectors specifying the model, and options to control the convergence and starting positions with no special options

## Usage

```
RunPoissonRegression(
  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()
)
```

## 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 Def_Control() for options or vignette("Control_Options") |

## Value

returns a list of the final results

## See Also

Other Poisson Wrapper Functions: PoissonCurveSolver(), RunPoissonEventAssignment(), RunPoissonEventAssignment_bound(), RunPoissonRegression_Guesses_CPP(), RunPoissonRegression_Joint_Omni RunPoissonRegression_Omnibus(), RunPoissonRegression_Residual(), RunPoissonRegression_Single(), RunPoissonRegression_Strata(), RunPoissonRegression_Tier_Guesses()

## 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)
)
# For the interval case
df$pyr <- df$Ending_Age - df$Starting_Age
pyr <- "pyr"
event <- "Cancer_Status"
names <- c("a", "b", "c", "d")
term_n <- c(0, 1, 1, 2)
tform <- c("loglin", "lin", "lin", "plin")
modelform <- "M"
a_n <- c(0.1, 0.1, 0.1, 0.1)
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, "abs_max" = 1.0,
  "dose_abs_max" = 100.0, "verbose" = FALSE, "double_step" = 1
)
e <- RunPoissonRegression(
  df, pyr, event, names, term_n, tform,
  keep_constant,
```

```
  a_n, modelform, control
)
```

---

RunPoissonRegression_Guesses_CPP *Performs basic Poisson regression, generates multiple starting guesses on c++ side*

---

## Description

RunPoissonRegression_Guesses_CPP 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_Guesses_CPP(
  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(),
  guesses_control = list(),
  strat_col = "null",
  model_control = list()
)
```

## 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 Def_Control() for options or vignette("Control_Options") |
| guesses_control | list of parameters to control how the guessing works, see Def_Control_Guess() for options or vignette("Control_Options") |
| strat_col | column to stratify by if needed |
| model_control | controls which alternative model options are used, see Def_model_control() for options and vignette("Control_Options") for further details |

## Value

returns a list of the final results

## See Also

Other Poisson Wrapper Functions: PoissonCurveSolver(), RunPoissonEventAssignment(), RunPoissonEventAssignment_bound(), RunPoissonRegression(), RunPoissonRegression_Joint_Omnibus(), RunPoissonRegression_Omnibus(), RunPoissonRegression_Residual(), RunPoissonRegression_Single(), RunPoissonRegression_Strata(), RunPoissonRegression_Tier_Guesses()

## 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, "abs_max" = 1.0,
  "dose_abs_max" = 100.0, "verbose" = FALSE, "ties" = "breslow",
  "double_step" = 1
)
```

```
guesses_control <- list(
  "maxiter" = 10, "guesses" = 10,
  "lin_min" = 0.001, "lin_max" = 1,
  "loglin_min" = -1, "loglin_max" = 1, "lin_method" = "uniform",
  "loglin_method" = "uniform", strata = FALSE
)
strat_col <- "e"
options(warn = -1)
e <- RunPoissonRegression_Guesses_CPP(
  df, pyr, event, names, term_n,
  tform, keep_constant, a_n, modelform,
  control, guesses_control, strat_col
)
```

---

RunPoissonRegression_Joint_Omnibus

*Performs joint Poisson regression using the omnibus function*

---

## Description

RunPoissonRegression_Joint_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, uses joint competing risks equation

## Usage

```
RunPoissonRegression_Joint_Omnibus(
  df,
  pyr0,
  events,
  name_list,
  term_n_list = list(),
  tform_list = list(),
  keep_constant_list = list(),
  a_n_list = list(),
  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 |
| 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 |
| modelform | string specifying the model type: M, ME, A, PA, PAE, GMIX, GMIX-R, GMIX-E |
| control | list of parameters controlling the convergence, see Def_Control() for options or vignette("Control_Options") |
| strat_col | column to stratify by if needed |
| model_control | controls which alternative model options are used, see Def_model_control() for options and vignette("Control_Options") 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: PoissonCurveSolver(), RunPoissonEventAssignment(), RunPoissonEventAssignment_bound(), RunPoissonRegression(), RunPoissonRegression_Guesses_CPP(), RunPoissonRegression_Omnibus(), RunPoissonRegression_Residual(), RunPoissonRegression_Single(), RunPoissonRegression_Strata(), RunPoissonRegression_Tier_Guesses()

## Examples

```
library(data.table)
## basic example code reproduced from the starting-description vignette
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)
f <- c(0, 1, 0, 0, 1, 1)
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)
modelform <- "M"
control <- list(
  "ncores" = 2, "lr" = 0.75, "maxiter" = 5,
  "halfmax" = 5, "epsilon" = 1e-3,
  "deriv_epsilon" = 1e-3, "abs_max" = 1.0,
  "dose_abs_max" = 100.0, "verbose" = FALSE,
  "ties" = "breslow", "double_step" = 1
)
guesses_control <- list(
  "maxiter" = 10, "guesses" = 10,
  "lin_min" = 0.001, "lin_max" = 1,
  "loglin_min" = -1, "loglin_max" = 1, "lin_method" = "uniform",
  "loglin_method" = "uniform", strata = FALSE
)
strat_col <- "f"
e <- RunPoissonRegression_Joint_Omnibus(
  df, pyr, events, name_list,
  term_n_list,
  tform_list, keep_constant_list,
  a_n_list,
  modelform,
  control, strat_col
)
```

`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 Def_Control() for options or vignette("Control_Options") |
| `strat_col` | column to stratify by if needed |

| | |
|---|---|
| model_control | controls which alternative model options are used, see Def_model_control() for options and vignette("Control_Options") 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: PoissonCurveSolver(), RunPoissonEventAssignment(), RunPoissonEventAssignment_bound(), RunPoissonRegression(), RunPoissonRegression_Guesses_CPP(), RunPoissonRegression_Joint_Omnibus(), RunPoissonRegression_Residual(), RunPoissonRegression_Single(), RunPoissonRegression_Strata(), RunPoissonRegression_Tier_Guesses()

## 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, "abs_max" = 1.0,
  "dose_abs_max" = 100.0, "verbose" = FALSE, "ties" = "breslow",
  "double_step" = 1
)
guesses_control <- list(
  "maxiter" = 10, "guesses" = 10, "lin_min" = 0.001,
```

```
  "lin_max" = 1, "loglin_min" = -1, "loglin_max" = 1, "lin_method" = "uniform",
  "loglin_method" = "uniform", strata = FALSE
)
strat_col <- "e"
e <- RunPoissonRegression_Omnibus(
  df, pyr, event, names, term_n,
  tform, keep_constant,
  a_n, modelform,
  control, strat_col
)
```

---

`RunPoissonRegression_Residual` *Calculates poisson residuals*

---

## Description

`RunPoissonRegression_Residual` uses user provided data, time/event columns, vectors specifying the model, and options. Calculates residuals or sum of residuals

## Usage

```
RunPoissonRegression_Residual(
  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()
)
```

## 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 Def_Control() for options or vignette("Control_Options") |
| strat_col | column to stratify by if needed |
| model_control | controls which alternative model options are used, see Def_model_control() for options and vignette("Control_Options") for further details |

## Value

returns a list of the final results

## See Also

Other Poisson Wrapper Functions: PoissonCurveSolver(), RunPoissonEventAssignment(), RunPoissonEventAssignment_bound(), RunPoissonRegression(), RunPoissonRegression_Guesses_CPP(), RunPoissonRegression_Joint_Omnibus(), RunPoissonRegression_Omnibus(), RunPoissonRegression_Single(), RunPoissonRegression_Strata(), RunPoissonRegression_Tier_Guesses()

## 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, "abs_max" = 1.0,
  "dose_abs_max" = 100.0, "verbose" = FALSE, "ties" = "breslow",
```

```
  "double_step" = 1
)
guesses_control <- list(
  "maxiter" = 10, "guesses" = 10,
  "lin_min" = 0.001, "lin_max" = 1,
  "loglin_min" = -1, "loglin_max" = 1, "lin_method" = "uniform",
  "loglin_method" = "uniform", strata = FALSE
)
strat_col <- "e"
e <- RunPoissonRegression_Residual(
  df, pyr, event, names, term_n,
  tform, keep_constant,
  a_n, modelform,
  control, strat_col
)
```

---

RunPoissonRegression_Single *Performs poisson regression with no derivative calculations*

---

## Description

RunPoissonRegression_Single uses user provided data, person-year/event columns, vectors specifying the model, and returns the results

## Usage

```
RunPoissonRegression_Single(
  df,
  pyr0 = "pyr",
  event0 = "event",
  names = c("CONST"),
  term_n = c(0),
  tform = "loglin",
  a_n = c(0),
  modelform = "M",
  control = list(),
  keep_constant = rep(0, length(names))
)
```

## 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 |
| 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 Def_Control() for options or vignette("Control_Options") |
| keep_constant | binary values to denote which parameters to change |

## Value

returns a list of the final results

## See Also

Other Poisson Wrapper Functions: PoissonCurveSolver(), RunPoissonEventAssignment(), RunPoissonEventAssignment_bound(), RunPoissonRegression(), RunPoissonRegression_Guesses_CPP(), RunPoissonRegression_Joint_Omnibus(), RunPoissonRegression_Omnibus(), RunPoissonRegression_Residual(), RunPoissonRegression_Strata(), RunPoissonRegression_Tier_Guesses()

## 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)
)
# For the interval case
df$pyr <- df$Ending_Age - df$Starting_Age
pyr <- "pyr"
event <- "Cancer_Status"
names <- c("a", "b", "c", "d")
term_n <- c(0, 1, 1, 2)
tform <- c("loglin", "lin", "lin", "plin")
modelform <- "M"
a_n <- c(0.1, 0.1, 0.1, 0.1)
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,
  "abs_max" = 1.0, "dose_abs_max" = 100.0,
  "verbose" = FALSE, "double_step" = 1
)
```

```
e <- RunPoissonRegression_Single(
  df, pyr, event, names,
  term_n, tform, a_n, modelform,
  control
)
```

---

`RunPoissonRegression_Strata` *Performs poisson regression with strata effect*

---

## Description

`RunPoissonRegression_Strata` uses user provided data, time/event columns, vectors specifying the model, and options to control the convergence and starting positions

## Usage

```
RunPoissonRegression_Strata(
  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"
)
```

## 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 Def_Control() for options or vignette("Control_Options") |
| `strat_col` | column to stratify by if needed |

## Value

returns a list of the final results

## See Also

Other Poisson Wrapper Functions: PoissonCurveSolver(), RunPoissonEventAssignment(), RunPoissonEventAssignment_bound(), RunPoissonRegression(), RunPoissonRegression_Guesses_CPP(), RunPoissonRegression_Joint_Omnibus(), RunPoissonRegression_Omnibus(), RunPoissonRegression_Residual(), RunPoissonRegression_Single(), RunPoissonRegression_Tier_Guesses()

## 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, 0, 0, 1, 0, 1)
)
# For the interval case
df$pyr <- df$Ending_Age - df$Starting_Age
pyr <- "pyr"
event <- "Cancer_Status"
names <- c("a", "b", "c", "d")
term_n <- c(0, 1, 1, 2)
tform <- c("loglin", "lin", "lin", "plin")
modelform <- "M"
a_n <- c(0.1, 0.1, 0.1, 0.1)
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,
  "abs_max" = 1.0, "dose_abs_max" = 100.0,
  "verbose" = FALSE, "double_step" = 1
)
strat_col <- c("e")
e <- RunPoissonRegression_Strata(
  df, pyr, event, names,
  term_n, tform, keep_constant,
  a_n, modelform, control, strat_col
)
```

`RunPoissonRegression_Tier_Guesses` *Performs basic poisson regression, with multiple guesses, starts with a single term*

## Description

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

## Usage

```
RunPoissonRegression_Tier_Guesses(
  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(),
  guesses_control = list(),
  strat_col = "null",
  model_control = list()
)
```

## 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 Def_Control() for options or vignette("Control_Options") |

guesses_control
: list of parameters to control how the guessing works, see Def_Control_Guess() for options or vignette("Control_Options")

strat_col
: column to stratify by if needed

model_control
: controls which alternative model options are used, see Def_model_control() for options and vignette("Control_Options") for further details

## Value

returns a list of the final results

## See Also

Other Poisson Wrapper Functions: PoissonCurveSolver(), RunPoissonEventAssignment(), RunPoissonEventAssignment_bound(), RunPoissonRegression(), RunPoissonRegression_Guesses_CPP(), RunPoissonRegression_Joint_Omnibus(), RunPoissonRegression_Omnibus(), RunPoissonRegression_Residual(), RunPoissonRegression_Single(), RunPoissonRegression_Strata()

## 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, 0, 0, 1, 0, 1)
)
# For the interval case
df$pyr <- df$Ending_Age - df$Starting_Age
pyr <- "pyr"
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, "abs_max" = 1.0,
  "dose_abs_max" = 100.0, "verbose" = FALSE, "double_step" = 1
)
guesses_control <- list(
  "iterations" = 10, "guesses" = 10,
```

```
  "lin_min" = 0.001, "lin_max" = 1,
  "loglin_min" = -1, "loglin_max" = 1, "lin_method" = "uniform",
  "loglin_method" = "uniform", strata = TRUE, term_initial = c(0, 1)
)
strat_col <- c("e")
options(warn = -1)
e <- RunPoissonRegression_Tier_Guesses(
  df, pyr, event, names,
  term_n, tform, keep_constant, a_n, modelform,
  control, guesses_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: `Interpret_Output()`

---

`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_Dupe_Columns(), Check_Trunc(), Check_Verbose(), Convert_Model_Eq(), Correct_Formula_Order(), Date_Shift(), Def_Control(), Def_Control_Guess(), Def_model_control(), Def_modelform_fix(), Event_Count_Gen(), Event_Time_Gen(), Joint_Multiple_Events(), Replace_Missing(), factorize(), factorize_par(), gen_time_dep(), interact_them()

## 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