Package 'rtemis'

March 26, 2026

Version 1.0.0

Title Machine Learning and Visualization

Date 2026-03-14

Description Machine learning and visualization package with an 'S7' backend featuring comprehensive type checking and validation, paired with an efficient functional user-facing API. train(), cluster(), and decomp() provide one-call access to supervised and unsupervised learning. All configuration steps are performed using setup functions and validated. A single call to train() handles preprocessing, hyperparameter tuning, and testing with nested resampling. Supports 'data.frame', 'data.table', and 'tibble' inputs, parallel execution, and interactive visualizations. The package first appeared in E.D. Gennatas (2017) <https://repository.upenn.edu/entities/publication/d81892ea-3087-4b71-a6f5-739c58626d64>.

License GPL (>= 3)

URL https://www.rtemis.org, https://docs.rtemis.org/r/, https://docs.rtemis.org/r-api/

BugReports https://github.com/rtemis-org/rtemis/issues

ByteCompile yes

Depends R (>= 4.1.0)

Imports grDevices, graphics, stats, methods, utils, S7, data.table, future, htmltools, cli

Suggests arrow, bit64, car, colorspace, DBI, dbscan, dendextend (>= 0.18.0), duckdb, e1071, farff, fastICA, flexclust, future.apply, future.mirai, futurize, geosphere, ggplot2, glmnet, geojsonio, glue, grid, gsubfn, haven, heatmaply, htmlwidgets, igraph, jsonlite, leaflet, leaps, lightAUC, lightgbm, matrixStats, mgcv, mice, mirai, missRanger, networkD3, NMF, openxlsx, parallelly, partykit, plotly, pROC, progressr, psych, pvclust, ranger, reactable, readxl, reticulate, ROCR, rpart, Rtsne, seqinr, sf, shapr, survival, tabnet, threejs, testthat (>= 3.0.0), tibble, timeDate, toml, torch, uwot, vegan, vroom, withr

Encoding UTF-8

Config/testthat/edition 3

RoxygenNote 7.3.3

LazyData true

NeedsCompilation no

Author E.D. Gennatas [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0001-9280-3609>)

Maintainer E.D. Gennatas <gennatas@gmail.com>

Repository CRAN

Date/Publication 2026-03-26 10:00:02 UTC

Contents

`rtemis-package` **rtemis***: Advanced Machine Learning and Visualization*

Description

Advanced Machine Learning & Visualization made efficient, accessible, reproducible

Online Documentation and Vignettes

https://docs.rtemis.org/r/

System Setup

There are some options you can define in your .Rprofile (usually found in your home directory), so you do not have to define each time you execute a function.

rtemis_theme General plotting theme; set to e.g. "whiteigrid" or "darkgraygrid"

rtemis_font Font family to use in plots.

rtemis_palette Name of default palette to use in plots. See options by running `get_palette()`

Visualization

Graphics are handled using the `draw` family, which produces interactive plots primarily using `plotly` and other packages.

Supervised Learning

By convention, the last column of the data is the outcome variable, and all other columns are predictors. Convenience function set_outcome can be used to move a specified column to the end of the data. Regression and Classification is performed using `train()`. This function allows you to preprocess, train, tune, and test models on multiple resamples. Use available_supervised to get a list of available algorithms

Classification

For training of binary classification models, the outcome should be provided as a factor, with the *second* level of the factor being the 'positive' class.

Clustering

Clustering is performed using `cluster()`. Use available_clustering to get a list of available algorithms.

Decomposition

Decomposition is performed using `decomp()`. Use available_decomposition to get a list of available algorithms.

Type Documentation

Function documentation includes input type (e.g. "Character", "Integer", "Float"/"Numeric", etc). When applicable, value ranges are provided in interval notation. For example, Float: [0, 1) means floats between 0 and 1 including 0, but excluding 1. Categorical variables may include set of allowed values using curly braces. For example, Character: {"future", "mirai", "none"}.

Tabular Data

rtemis internally uses methods for efficient handling of tabular data, with support for `data.frame`, `data.table`, and `tibble`. If a function is documented as accepting "tabular data", it should work with any of these data structures. If a function is documented as accepting only one of these, then it should only be used with that structure. For example, some optimized `data.table` operations that perform in-place modifications only work with `data.table` objects.

Author(s)

Maintainer: E.D. Gennatas <gennatas@gmail.com> (ORCID) [copyright holder]

See Also

Useful links:

- https://www.rtemis.org
- https://docs.rtemis.org/r/
- https://docs.rtemis.org/r-api/
- Report bugs at https://github.com/rtemis-org/rtemis/issues

available_draw *Available Draw Functions*

Description

Print available draw functions for visualization.

Usage

```
available_draw()
```

Value

NULL, invisibly.

Author(s)

EDG

Examples

```
available_draw()
```

available_supervised *Available Algorithms*

Description

Print available algorithms for supervised learning, clustering, and decomposition.

Usage

```
available_supervised()

available_clustering()

available_decomposition()
```

Value

Called for its side effect of printing available algorithms.

Author(s)

EDG

Examples

```
available_supervised()
available_clustering()
available_decomposition()
```

`available_themes` *Print available* **rtemis** *themes*

Description

Print available **rtemis** themes

Usage

```
available_themes()
```

Value

Called for its side effect of printing available themes.

Author(s)

EDG

Examples

```
available_themes()
```

`calibrate` *Calibrate* `Classification` *&* `ClassificationRes` *Models*

Description

Generic function to calibrate binary classification models.

Usage

```
calibrate(
  x,
  algorithm = "isotonic",
  hyperparameters = NULL,
  verbosity = 1L,
  ...
)
```

Arguments

x	Classification or ClassificationRes object to calibrate.
algorithm	Character: Algorithm to use to train calibration model.
hyperparameters	Hyperparameters object: Setup using one of setup_* functions.
verbosity	Integer: Verbosity level.
...	Additional arguments passed to specific methods.

Details

The goal of calibration is to adjust the predicted probabilities of a binary classification model so that they better reflect the true probabilities (i.e. empirical risk) of the positive class.

Value

Calibrated model object.

Method-specific parameters

For Classification **objects:**

- predicted_probabilities: Numeric vector of predicted probabilities
- true_labels: Factor of true class labels

For ClassificationRes **objects:**

- resampler_config: ResamplerConfig object for calibration training
- train_verbosity: Integer controlling calibration model training output

Author(s)

EDG

Examples

```
# --- Calibrate Classification ---
dat <- iris[51:150, ]
res <- resample(dat)
dat$Species <- factor(dat$Species)
dat_train <- dat[res[[1]], ]
dat_test <- dat[-res[[1]], ]

# Train GLM on a training/test split
mod_c_glm <- train(
  x = dat_train,
  dat_test = dat_test,
  algorithm = "glm"
)

# Calibrate the `Classification` by defining `predicted_probabilities` and `true_labels`,
```

```
# in this case using the training data, but it could be a separate calibration dataset.
mod_c_glm_cal <- calibrate(
  mod_c_glm,
  predicted_probabilities = mod_c_glm$predicted_prob_training,
  true_labels = mod_c_glm$y_training
)
mod_c_glm_cal

# --- Calibrate ClassificationRes ---

# Train GLM with cross-validation
resmod_c_glm <- train(
  x = dat,
  algorithm = "glm",
  outer_resampling_config = setup_Resampler(n_resamples = 3L, type = "KFold")
)

# Calibrate the `ClassificationRes` using the same resampling configuration as used for training.
resmod_c_glm_cal <- calibrate(resmod_c_glm)
resmod_c_glm_cal
```

---

check_data *Check Data*

Description

Check Data

Usage

```
check_data(
  x,
  name = NULL,
  get_duplicates = TRUE,
  get_na_case_pct = FALSE,
  get_na_feature_pct = FALSE
)
```

Arguments

x	tabular data: Input to be checked.
name	Character: Name of dataset.
get_duplicates	Logical: If TRUE, check for duplicate cases.
get_na_case_pct	Logical: If TRUE, calculate percent of NA values per case.
get_na_feature_pct	Logical: If TRUE, calculate percent of NA values per feature.

Value

CheckData object.

Author(s)

EDG

Examples

```
n <- 1000
x <- rnormmat(n, 50, return_df = TRUE)
x$char1 <- sample(letters, n, TRUE)
x$char2 <- sample(letters, n, TRUE)
x$fct <- factor(sample(letters, n, TRUE))
x <- rbind(x, x[1, ])
x$const <- 99L
x[sample(nrow(x), 20), 3] <- NA
x[sample(nrow(x), 20), 10] <- NA
x$fct[30:35] <- NA
check_data(x)
```

---

choose_theme *Select an rtemis theme*

Description

Select an rtemis theme

Usage

```
choose_theme(
  x = c("white", "whitegrid", "whiteigrid", "black", "blackgrid", "blackigrid",
   "darkgray", "darkgraygrid", "darkgrayigrid", "lightgraygrid", "mediumgraygrid"),
  override = NULL
)
```

Arguments

x	Character: Name of theme to select. If not defined, will use getOption("rtemis_theme", "whitegrid").
override	Optional List: Theme parameters to override defaults.

Details

If x is not defined, choose_theme() will use getOption("rtemis_theme", "whitegrid") to select the theme. This allows users to set a default theme for all rtemis plots by setting options(rtemis_theme = "theme_name") at any point.

Value

Theme object.

Author(s)

EDG

Examples

```
# Get default theme set by options(rtemis_theme = "theme_name").
# If not set, defaults to "whitegrid":
choose_theme()
# Get darkgraygrid theme. Same as `theme_darkgraygrid()`:
choose_theme("darkgraygrid")
# This will use the default theme, and override the foreground color to red:
choose_theme(override = list(fg = "#ff0000"))
```

---

classification_metrics

Classification Metrics

Description

Classification Metrics

Usage

```
classification_metrics(
  true_labels,
  predicted_labels,
  predicted_prob = NULL,
  binclasspos = 2L,
  calc_auc = TRUE,
  calc_brier = TRUE,
  auc_method = "lightAUC",
  sample = character(),
  verbosity = 0L
)
```

Arguments

true_labels	Factor: True labels.
predicted_labels	Factor: predicted values.
predicted_prob	Numeric vector: predicted probabilities.
binclasspos	Integer: Factor level position of the positive class in binary classification.

calc_auc	Logical: If TRUE, calculate AUC. May be slow in very large datasets.
calc_brier	Logical: If TRUE, calculate Brier_Score.
auc_method	Character: "lightAUC", "pROC", "ROCR".
sample	Character: Sample name.
verbosity	Integer: Verbosity level.

Details

Note that auc_method = "pROC" is the only one that will output an AUC even if one or more predicted probabilities are NA.

Value

ClassificationMetrics object.

Author(s)

EDG

Examples

```
# Assume positive class is "b"
true_labels <- factor(c("a", "a", "a", "b", "b", "b", "b", "b", "b", "b"))
predicted_labels <- factor(c("a", "b", "a", "b", "b", "a", "b", "b", "b", "a"))
predicted_prob <- c(0.3, 0.55, 0.45, 0.75, 0.57, 0.3, 0.8, 0.63, 0.62, 0.39)

classification_metrics(true_labels, predicted_labels, predicted_prob)
classification_metrics(true_labels, predicted_labels, 1 - predicted_prob, binclasspos = 1L)
```

---

class_imbalance *Class Imbalance*

Description

Calculate class imbalance as given by:

$$I = K \cdot \sum_{i=1}^{K} (n_i/N - 1/K)^2$$

where K is the number of classes, and n_i is the number of instances of class i

Usage

```
class_imbalance(x)
```

Arguments

x	Vector, factor: Outcome.

Value

Numeric.

Author(s)

EDG

Examples

```
# iris is perfectly balanced
class_imbalance(iris[["Species"]])
# Simulate imbalanced outcome
x <- factor(sample(c("A", "B"), size = 500L, replace = TRUE, prob = c(0.9, 0.1)))
class_imbalance(x)
```

---

clean_colnames *Clean column names*

Description

Clean column names by replacing all spaces and punctuation with a single underscore

Usage

```
clean_colnames(x, lowercase = FALSE, uppercase = FALSE, titlecase = FALSE)
```

Arguments

x	Character vector OR any object with `colnames()` method, like matrix, data.frame, data.table, tibble, etc.
lowercase	Logical: If TRUE, convert to lowercase.
uppercase	Logical: If TRUE, convert to uppercase.
titlecase	Logical: If TRUE, convert to Title Case.

Value

Character vector with cleaned names.

Author(s)

EDG

Examples

```
clean_colnames(iris, lowercase = FALSE, uppercase = FALSE, titlecase = FALSE)
```

clean_names *Clean names*

Description

Clean character vector by replacing all symbols and sequences of symbols with single underscores, ensuring no name begins or ends with a symbol

Usage

```
clean_names(x, sep = "_", prefix_digits = "V_")
```

Arguments

x	Character vector.
sep	Character: Separator to replace symbols with.
prefix_digits	Character: prefix to add to names beginning with a digit. Set to NA to skip.

Value

Character vector.

Author(s)

EDG

Examples

```
x <- c("Patient ID", "_Date-of-Birth", "SBP (mmHg)")
x
clean_names(x)
clean_names(x, sep = " ")
```

cluster *Perform Clustering*

Description

Perform clustering on the rows (usually cases) of a dataset.

Usage

```
cluster(x, algorithm = "KMeans", config = NULL, verbosity = 1L)
```

Arguments

x	Matrix or data.frame: Data to cluster. Rows are cases to be clustered.
algorithm	Character: Clustering algorithm.
config	List: Algorithm-specific config.
verbosity	Integer: Verbosity level.

Details

See docs.rtemis.org/r for detailed documentation.

Value

Clustering object.

Author(s)

EDG

Examples

```
iris_km <- cluster(exc(iris, "Species"), algorithm = "KMeans")
```

---

col2grayscale *Color to Grayscale*

Description

Convert a color to grayscale

Usage

```
col2grayscale(x, what = c("color", "decimal"))
```

Arguments

x	Color to convert to grayscale
what	Character: "color" returns a hexadecimal color, "decimal" returns a decimal between 0 and 1

Details

Uses the NTSC grayscale conversion: 0.299 * R + 0.587 * G + 0.114 * B

Value

Character: color hex code.

Author(s)

EDG

Examples

```
col2grayscale("red")
col2grayscale("red", "dec")
```

`color_adjust` *Adjust HSV Color*

Description

Modify alpha, hue, saturation and value (HSV) of a color

Usage

```
color_adjust(color, alpha = NULL, hue = 0, sat = 0, val = 0)
```

Arguments

`color`	Input color. Any format that grDevices::col2rgb() recognizes
`alpha`	Numeric: Scale alpha by this amount. Future: replace with absolute setting
`hue`	Float: How much hue to add to `color`
`sat`	Float: How much saturation to add to `color`
`val`	Float: How much to increase value of `color` by

Value

Adjusted color

Author(s)

EDG

Examples

```
previewcolor(c(teal = "#00ffff", teal50 = color_adjust("#00ffff", alpha = 0.5)))
```

ddb_collect *Collect a lazy-read duckdb table*

Description

Collect a table read with `ddb_data(x, collect = FALSE)`

Usage

```
ddb_collect(sql, progress = TRUE, returnobj = c("data.frame", "data.table"))
```

Arguments

`sql`	Character: DuckDB SQL query, usually output of ddb_data with `collect = FALSE`
`progress`	Logical: If TRUE, show progress bar
`returnobj`	Character: data.frame or data.table: class of object to return

Value

`data.frame` or `data.table`.

Author(s)

EDG

Examples

```
## Not run:
  # Requires local CSV file; replace with your own path
  sql <- ddb_data("/Data/iris.csv", collect = FALSE)
  ir <- ddb_collect(sql)

## End(Not run)
```

ddb_data *Read CSV using DuckDB*

Description

Lazy-read a CSV file, optionally: filter rows, remove duplicates, clean column names, convert character to factor, collect.

Usage

```
ddb_data(
  filename,
  datadir = NULL,
  sep = ",",
  header = TRUE,
  quotechar = "",
  ignore_errors = TRUE,
  make_unique = TRUE,
  select_columns = NULL,
  filter_column = NULL,
  filter_vals = NULL,
  character2factor = FALSE,
  collect = TRUE,
  progress = TRUE,
  returnobj = c("data.table", "data.frame"),
  data.table.key = NULL,
  clean_colnames = TRUE,
  verbosity = 1L
)
```

Arguments

`filename`	Character: file name; either full path or just the file name, if `datadir` is also provided.
`datadir`	Character: Optional path if `filename` is not full path.
`sep`	Character: Field delimiter/separator.
`header`	Logical: If TRUE, first line will be read as column names.
`quotechar`	Character: Quote character.
`ignore_errors`	Logical: If TRUE, ignore parsing errors (sometimes it's either this or no data, so).
`make_unique`	Logical: If TRUE, keep only unique rows.
`select_columns`	Character vector: Column names to select.
`filter_column`	Character: Name of column to filter on, e.g. "ID".
`filter_vals`	Numeric or Character vector: Values in `filter_column` to keep. `filter_column` to keep.
`character2factor`	Logical: If TRUE, convert character columns to factors.
`collect`	Logical: If TRUE, collect data and return structure class as defined by `returnobj`.
`progress`	Logical: If TRUE, print progress (no indication this works).
`returnobj`	Character: "data.frame" or "data.table" object class to return. If "data.table", data.frame object returned from `DBI::dbGetQuery` is passed to `data.table::setDT`; will add to execution time if very large, but then that's when you need a data.table.

`data.table.key`	Character: If set, this corresponds to a column name in the dataset. This column will be set as key in the data.table output.
`clean_colnames`	Logical: If TRUE, clean colnames with clean_colnames.
`verbosity`	Integer: Verbosity level.

Value

data.frame or data.table if `collect` is TRUE, otherwise a character with the SQL query

Author(s)

EDG

Examples

```
## Not run:
  # Requires local CSV file; replace with your own path
  ir <- ddb_data("/Data/massive_dataset.csv",
    filter_column = "ID",
    filter_vals = 8001:9999
  )

## End(Not run)
```

---

ddSci *Format Numbers for Printing*

Description

2 Decimal places, otherwise scientific notation

Usage

```
ddSci(x, decimal_places = 2, hi = 1e+06, as_numeric = FALSE)
```

Arguments

`x`	Vector of numbers
`decimal_places`	Integer: Return this many decimal places.
`hi`	Float: Threshold at or above which scientific notation is used.
`as_numeric`	Logical: If TRUE, convert to numeric before returning. This will not force all numbers to print 2 decimal places. For example: 1.2035 becomes "1.20" if `as_numeric = FALSE`, but 1.2 otherwise This can be helpful if you want to be able to use the output as numbers / not just for printing.

Details

Numbers will be formatted to 2 decimal places, unless this results in 0.00 (e.g. if input was .0032), in which case they will be converted to scientific notation with 2 significant figures. `ddSci` will return `0.00` if the input is exactly zero. This function can be used to format numbers in plots, on the console, in logs, etc.

Value

Formatted number

Author(s)

EDG

Examples

```
x <- .34876549
ddSci(x)
# "0.35"
x <- .00000000457823
ddSci(x)
# "4.6e-09"
```

`decomp` *Perform Data Decomposition*

Description

Perform linear or non-linear decomposition of numeric data.

Usage

```
decomp(x, algorithm = "ICA", config = NULL, verbosity = 1L)
```

Arguments

`x`	Matrix or data frame: Input data.
`algorithm`	Character: Decomposition algorithm.
`config`	DecompositionConfig: Algorithm-specific config.
`verbosity`	Integer: Verbosity level.

Details

See docs.rtemis.org/r for detailed documentation.

Value

`Decomposition` object.

Author(s)

EDG

Examples

```
iris_pca <- decomp(exc(iris, "Species"), algorithm = "PCA")
```

`describe` *Describe rtemis object*

Description

This generic is used to provide a description of an rtemis object in plain language.

Usage

```
describe(x, ...)
```

Arguments

`x`	`Supervised` or `SupervisedRes` object or list of such objects.
`...`	Not used.

Value

A character string describing the object.

Author(s)

EDG

Examples

```
species_lightrf <- train(iris, algorithm = "lightrf")
describe(species_lightrf)
```

describe.factor *Describe factor*

Description

Outputs a single character with names and counts of each level of the input factor.

Arguments

x	factor.
max_n	Integer: Return counts for up to this many levels.
return_ordered	Logical: If TRUE, return levels ordered by count, otherwise return in level order.

Value

Character with level counts.

Author(s)

EDG

Examples

```
# Small number of levels
describe(iris[["Species"]])

# Large number of levels: show top n by count
x <- factor(sample(letters, 1000, TRUE))
describe(x)
describe(x, 3)
describe(x, 3, return_ordered = FALSE)
```

df_movecolumn *Move data frame column*

Description

Move data frame column

Usage

```
df_movecolumn(x, colname, to = ncol(x))
```

Arguments

x	data.frame.
colname	Character: Name of column you want to move.
to	Integer: Which column position to move the vector to. Default = ncol(x) i.e. the last column.

Value

data.frame

Author(s)

EDG

Examples

```
ir <- df_movecolumn(iris, colname = "Species", to = 1L)
```

---

df_nunique_perfeat *Unique values per feature*

Description

Get number of unique values per features

Usage

```
df_nunique_perfeat(x, excludeNA = FALSE)
```

Arguments

x	matrix or data frame input
excludeNA	Logical: If TRUE, exclude NA values from unique count.

Value

Vector, integer of length NCOL(x) with number of unique values per column/feature

Author(s)

EDG

Examples

```
df_nunique_perfeat(iris)
```

draw_3Dscatter *Interactive 3D Scatter Plots*

Description

Draw interactive 3D scatter plots using `plotly`.

Usage

```
draw_3Dscatter(
  x,
  y = NULL,
  z = NULL,
  fit = NULL,
  cluster = NULL,
  cluster_config = NULL,
  group = NULL,
  formula = NULL,
  rsq = TRUE,
  mode = "markers",
  order_on_x = NULL,
  main = NULL,
  xlab = NULL,
  ylab = NULL,
  zlab = NULL,
  alpha = 0.8,
  bg = NULL,
  plot_bg = NULL,
  theme = choose_theme(getOption("rtemis_theme")),
  palette = get_palette(getOption("rtemis_palette")),
  axes_square = FALSE,
  group_names = NULL,
  font_size = 16,
  marker_col = NULL,
  marker_size = 8,
  fit_col = NULL,
  fit_alpha = 0.7,
  fit_lwd = 2.5,
  tick_font_size = 12,
  spike_col = NULL,
  legend = NULL,
  legend_xy = c(0, 1),
  legend_xanchor = "left",
  legend_yanchor = "auto",
  legend_orientation = "v",
  legend_col = NULL,
  legend_bg = "#FFFFFF00",
```

```
  legend_border_col = "#FFFFFF00",
  legend_borderwidth = 0,
  legend_group_gap = 0,
  margin = list(t = 30, b = 0, l = 0, r = 0),
  fit_params = NULL,
  width = NULL,
  height = NULL,
  padding = 0,
  displayModeBar = TRUE,
  modeBar_file_format = "svg",
  verbosity = 0L,
  filename = NULL,
  file_width = 500,
  file_height = 500,
  file_scale = 1
)
```

Arguments

x	Numeric, vector/data.frame/list: x-axis data.
y	Numeric, vector/data.frame/list: y-axis data.
z	Numeric, vector/data.frame/list: z-axis data.
fit	Character: Fit method.
cluster	Character: Clustering method.
cluster_config	List: Config for clustering.
group	Factor: Grouping variable.
formula	Formula: Formula for non-linear least squares fit.
rsq	Logical: If TRUE, print R-squared values in legend if `fit` is set.
mode	Character, vector: "markers", "lines", "markers+lines".
order_on_x	Logical: If TRUE, order `x` and `y` on `x`.
main	Character: Main title.
xlab	Character: x-axis label.
ylab	Character: y-axis label.
zlab	Character: z-axis label.
alpha	Numeric: Alpha for markers.
bg	Background color.
plot_bg	Plot background color.
theme	`Theme` object.
palette	Character vector: Colors to use.
axes_square	Logical: If TRUE, draw a square plot.
group_names	Character: Names for groups.
font_size	Numeric: Font size.

marker_col	Color for markers.
marker_size	Numeric: Marker size.
fit_col	Color for fit line.
fit_alpha	Numeric: Alpha for fit line.
fit_lwd	Numeric: Line width for fit line.
tick_font_size	Numeric: Tick font size.
spike_col	Spike lines color.
legend	Logical: If TRUE, draw legend.
legend_xy	Numeric: Position of legend.
legend_xanchor	Character: X anchor for legend.
legend_yanchor	Character: Y anchor for legend.
legend_orientation	Character: Orientation of legend.
legend_col	Color for legend text.
legend_bg	Color for legend background.
legend_border_col	Color for legend border.
legend_borderwidth	Numeric: Border width for legend.
legend_group_gap	Numeric: Gap between legend groups.
margin	Numeric, named list: Margins for top, bottom, left, right.
fit_params	Hyperparameters for fit.
width	Numeric: Width of plot.
height	Numeric: Height of plot.
padding	Numeric: Graph padding.
displayModeBar	Logical: If TRUE, display mode bar.
modeBar_file_format	Character: File format for mode bar.
verbosity	Integer: Verbosity level.
filename	Character: Filename to save plot.
file_width	Numeric: Width of saved file.
file_height	Numeric: Height of saved file.
file_scale	Numeric: Scale of saved file.

Details

See docs.rtemis.org/r for detailed documentation.

Note that draw_3Dscatter uses the theme's plot_bg as grid_col.

Value

A plotly object.

Author(s)

EDG

Examples

```
draw_3Dscatter(iris, group = iris$Species, theme = theme_darkgraygrid())
```

---

draw_bar *Interactive Barplots*

Description

Draw interactive barplots using plotly

Usage

```
draw_bar(
  x,
  main = NULL,
  xlab = NULL,
  ylab = NULL,
  alpha = 1,
  horizontal = FALSE,
  theme = choose_theme(getOption("rtemis_theme")),
  palette = get_palette(getOption("rtemis_palette")),
  barmode = c("group", "relative", "stack", "overlay"),
  group_names = NULL,
  order_by_val = FALSE,
  ylim = NULL,
  hovernames = NULL,
  feature_names = NULL,
  font_size = 16,
  annotate = FALSE,
  annotate_col = theme[["labs_col"]],
  legend = NULL,
  legend_col = NULL,
  legend_xy = c(1, 1),
  legend_orientation = "v",
  legend_xanchor = "left",
  legend_yanchor = "auto",
  hline = NULL,
  hline_col = NULL,
```

```
  hline_width = 1,
  hline_dash = "solid",
  hline_annotate = NULL,
  hline_annotation_x = 1,
  margin = list(b = 65, l = 65, t = 50, r = 10, pad = 0),
  automargin_x = TRUE,
  automargin_y = TRUE,
  padding = 0,
  displayModeBar = TRUE,
  modeBar_file_format = "svg",
  filename = NULL,
  file_width = 500,
  file_height = 500,
  file_scale = 1,
  verbosity = 0L
)
```

Arguments

x	vector (possibly named), matrix, or data.frame: If matrix or data.frame, rows are groups (can be 1 row), columns are features
main	Character: Main plot title.
xlab	Character: x-axis label.
ylab	Character: y-axis label.
alpha	Float (0, 1]: Transparency for bar colors.
horizontal	Logical: If TRUE, plot bars horizontally
theme	Theme object.
palette	Character vector: Colors to use.
barmode	Character: Type of bar plot to make: "group", "relative", "stack", "overlay". Default = "group". Use "relative" for stacked bars, wich handles negative values correctly, unlike "stack", as of writing.
group_names	Character, vector, length = NROW(x): Group names. Default = NULL, which uses rownames(x)
order_by_val	Logical: If TRUE, order bars by increasing value. Only use for single group data.
ylim	Float, vector, length 2: y-axis limits.
hovernames	Character, vector: Optional character vector to show on hover over each bar.
feature_names	Character, vector, length = NCOL(x): Feature names. Default = NULL, which uses colnames(x)
font_size	Float: Font size for all labels.
annotate	Logical: If TRUE, annotate stacked bars
annotate_col	Color for annotations
legend	Logical: If TRUE, draw legend. Default = NULL, and will be turned on if there is more than one feature present

legend_col	Color: Legend text color. Default = NULL, determined by theme
legend_xy	Numeric, vector, length 2: x and y for plotly's legend
legend_orientation	"v" or "h" for vertical or horizontal
legend_xanchor	Character: Legend's x anchor: "left", "center", "right", "auto"
legend_yanchor	Character: Legend's y anchor: "top", "middle", "bottom", "auto"
hline	Float: If defined, draw a horizontal line at this y value.
hline_col	Color for hline.
hline_width	Float: Width for hline.
hline_dash	Character: Type of line to draw: "solid", "dot", "dash", "longdash", "dashdot", or "longdashdot"
hline_annotate	Character: Text of horizontal line annotation if hline is set
hline_annotation_x	Numeric: x position to place annotation with paper as reference. 0: to the left of the plot area; 1: to the right of the plot area
margin	Named list: plot margins.
automargin_x	Logical: If TRUE, automatically set x-axis margins
automargin_y	Logical: If TRUE, automatically set y-axis margins
padding	Integer: N pixels to pad plot.
displayModeBar	Logical: If TRUE, show plotly's modebar
modeBar_file_format	Character: "svg", "png", "jpeg", "pdf" / any output file type supported by plotly and your system
filename	Character: Path to file to save static plot.
file_width	Integer: File width in pixels for when filename is set.
file_height	Integer: File height in pixels for when filename is set.
file_scale	Numeric: If saving to file, scale plot by this number
verbosity	Integer: Verbosity level.

Details

See docs.rtemis.org/r for detailed documentation.

Value

plotly object.

Author(s)

EDG

Examples

```
draw_bar(VADeaths, legend_xy = c(0, 1))
draw_bar(VADeaths, legend_xy = c(1, 1), legend_xanchor = "left")
# simple individual bars
a <- c(4, 7, 2)
draw_bar(a)
# if input is a data.frame, each row is a group and each column is a feature
b <- data.frame(x = c(3, 5, 7), y = c(2, 1, 8), z = c(4, 5, 2))
rownames(b) <- c("Jen", "Ben", "Ren")
draw_bar(b)
# stacked
draw_bar(b, barmode = "stack")
```

---

draw_box *Interactive Boxplots & Violin plots*

Description

Draw interactive boxplots or violin plots using **plotly**

Usage

```
draw_box(
  x,
  time = NULL,
  time_bin = c("year", "quarter", "month", "day"),
  type = c("box", "violin"),
  group = NULL,
  x_transform = c("none", "scale", "minmax"),
  main = NULL,
  xlab = "",
  ylab = NULL,
  alpha = 0.6,
  bg = NULL,
  plot_bg = NULL,
  theme = choose_theme(getOption("rtemis_theme")),
  palette = get_palette(getOption("rtemis_palette")),
  boxpoints = "outliers",
  quartilemethod = "linear",
  xlim = NULL,
  ylim = NULL,
  violin_box = TRUE,
  orientation = "v",
  annotate_n = FALSE,
  annotate_n_y = 1,
  annotate_mean = FALSE,
```

```
  annotate_meansd = FALSE,
  annotate_meansd_y = 1,
  annotate_col = theme[["labs_col"]],
  xnames = NULL,
  group_lines = FALSE,
  group_lines_dash = "dot",
  group_lines_col = NULL,
  group_lines_alpha = 0.5,
  labelify = TRUE,
  order_by_fn = NULL,
  font_size = 16,
  ylab_standoff = 18,
  legend = NULL,
  legend_col = NULL,
  legend_xy = NULL,
  legend_orientation = "v",
  legend_xanchor = "auto",
  legend_yanchor = "auto",
  xaxis_type = "category",
  cataxis_tickangle = "auto",
  margin = list(b = 65, l = 65, t = 50, r = 12, pad = 0),
  automargin_x = TRUE,
  automargin_y = TRUE,
  boxgroupgap = NULL,
  hovertext = NULL,
  show_n = FALSE,
  pvals = NULL,
  htest = "none",
  htest_compare = 0,
  htest_y = NULL,
  htest_annotate = TRUE,
  htest_annotate_x = 0,
  htest_annotate_y = -0.065,
  htest_star_col = theme[["labs_col"]],
  htest_bracket_col = theme[["labs_col"]],
  starbracket_pad = c(0.04, 0.05, 0.09),
  use_plotly_group = FALSE,
  width = NULL,
  height = NULL,
  displayModeBar = TRUE,
  modeBar_file_format = "svg",
  filename = NULL,
  file_width = 500,
  file_height = 500,
  file_scale = 1,
  mathjax = NULL
)
```

Arguments

x	Vector or List of vectors: Input
time	Date or date-time vector
time_bin	Character: "year", "quarter", "month", or "day". Period to bin by
type	Character: "box" or "violin"
group	Factor to group by
x_transform	Character: "none", "scale", or "minmax" to use raw values, scaled and centered values or min-max normalized to 0-1, respectively. Transform is applied to each variable before grouping, so that groups are comparable
main	Character: Plot title.
xlab	Character: x-axis label.
ylab	Character: y-axis label.
alpha	Float (0, 1]: Transparency for box colors.
bg	Color: Background color.
plot_bg	Color: Background color for plot area.
theme	Theme object.
palette	Character vector: Colors to use.
boxpoints	Character or FALSE: "all", "suspectedoutliers", "outliers" See https://plotly.com/r/box-plots/#choosing-the-algorithm-for-computing-quartiles
quartilemethod	Character: "linear", "exclusive", "inclusive"
xlim	Numeric vector: x-axis limits
ylim	Numeric vector: y-axis limits
violin_box	Logical: If TRUE and type is "violin" show box within violin plot
orientation	Character: "v" or "h" for vertical, horizontal
annotate_n	Logical: If TRUE, annotate with N in each box
annotate_n_y	Numeric: y position for annotate_n
annotate_mean	Logical: If TRUE, annotate with mean of each box
annotate_meansd	Logical: If TRUE, annotate with mean (SD) of each box
annotate_meansd_y	Numeric: y position for annotate_meansd
annotate_col	Color for annotations
xnames	Character, vector, length = NROW(x): x-axis names. Default = NULL, which tries to set names automatically.
group_lines	Logical: If TRUE, add separating lines between groups of boxplots
group_lines_dash	Character: "solid", "dot", "dash", "longdash", "dashdot", or "longdashdot"
group_lines_col	Color for group_lines

group_lines_alpha
: Numeric: transparency for group_lines_col

labelify
: Logical: If TRUE, labelify x names

order_by_fn
: Function: If defined, order boxes by increasing value of this function (e.g. median).

font_size
: Float: Font size for all labels.

ylab_standoff
: Numeric: Standoff for y-axis label

legend
: Logical: If TRUE, draw legend.

legend_col
: Color: Legend text color. Default = NULL, determined by the theme.

legend_xy
: Float, vector, length 2: Relative x, y position for legend.

legend_orientation
: "v" or "h" for vertical, horizontal

legend_xanchor
: Character: Legend's x anchor: "left", "center", "right", "auto"

legend_yanchor
: Character: Legend's y anchor: "top", "middle", "bottom", "auto"

xaxis_type
: Character: "linear", "log", "date", "category", "multicategory"

cataxis_tickangle
: Numeric: Angle for categorical axis tick labels

margin
: Named list: plot margins.

automargin_x
: Logical: If TRUE, automatically set x-axis margins

automargin_y
: Logical: If TRUE, automatically set y-axis margins

boxgroupgap
: Numeric: Sets the gap (in plot fraction) between boxes of the same location coordinate

hovertext
: Character vector: Text to show on hover for each data point

show_n
: Logical: If TRUE, show N in each box

pvals
: Numeric vector: Precomputed p-values. Should correspond to each box. Bypasses htest and htest_compare. Requires group to be set

htest
: Character: e.g. "t.test", "wilcox.test" to compare each box to the *first* box. If grouped, compare within each group to the first box. If p-value of test is less than htest.thresh, add asterisk above/ to the side of each box

htest_compare
: Integer: 0: Compare all distributions against the first one; 2: Compare every second box to the one before it. Requires group to be set

htest_y
: Numeric: y coordinate for htest annotation

htest_annotate
: Logical: if TRUE, include htest annotation

htest_annotate_x
: Numeric: x-axis paper coordinate for htest annotation

htest_annotate_y
: Numeric: y-axis paper coordinate for htest annotation

htest_star_col
: Color for htest annotation stars

htest_bracket_col
: Color for htest annotation brackets

`starbracket_pad`
: Numeric: Padding for htest annotation brackets

`use_plotly_group`
: If TRUE, use plotly's `group` arg to group boxes.

`width`
: Numeric: Force plot size to this width. Default = NULL, i.e. fill available space

`height`
: Numeric: Force plot size to this height. Default = NULL, i.e. fill available space

`displayModeBar`
: Logical: If TRUE, show plotly's modebar

`modeBar_file_format`
: Character: "svg", "png", "jpeg", "pdf"

`filename`
: Character: Path to file to save static plot.

`file_width`
: Integer: File width in pixels for when `filename` is set.

`file_height`
: Integer: File height in pixels for when `filename` is set.

`file_scale`
: Numeric: If saving to file, scale plot by this number

`mathjax`
: Optional Character {"local", "cdn"}: Whether to use local or CDN version of MathJax for rendering mathematical annotations.

Details

See docs.rtemis.org/r for detailed documentation.

For multiple box plots, the recommendation is:

- `x=dat[, columnindex]` for multiple variables of a data.frame
- `x=list(a=..., b=..., etc.)` for multiple variables of potentially different length
- `x=split(var, group)` for one variable with multiple groups: group names appear below boxplots
- `x=dat[, columnindex], group = factor` for grouping multiple variables: group names appear in legend

If `orientation == "h"`, `xlab` is applied to y-axis and vice versa. Similarly, `x.axist.type` applies to y-axis - this defaults to "category" and would not normally need changing.

Value

`plotly` object.

Author(s)

EDG

Examples

```
# A.1 Box plot of 4 variables
draw_box(iris[, 1:4])
# A.2 Grouped Box plot
draw_box(iris[, 1:4], group = iris[["Species"]])
draw_box(iris[, 1:4], group = iris[["Species"]], annotate_n = TRUE)
# B. Boxplot binned by time periods
```

```
# Synthetic data with an instantenous shift in distributions
set.seed(2021)
dat1 <- data.frame(alpha = rnorm(200, 0), beta = rnorm(200, 2), gamma = rnorm(200, 3))
dat2 <- data.frame(alpha = rnorm(200, 5), beta = rnorm(200, 8), gamma = rnorm(200, -3))
x <- rbind(dat1, dat2)
startDate <- as.Date("2019-12-04")
endDate <- as.Date("2021-03-31")
time <- seq(startDate, endDate, length.out = 400)
draw_box(x[, 1], time, "year", ylab = "alpha")
draw_box(x, time, "year", legend.xy = c(0, 1))
draw_box(x, time, "quarter", legend.xy = c(0, 1))
draw_box(x, time, "month",
  legend.orientation = "h",
  legend.xy = c(0, 1),
  legend.yanchor = "bottom"
)
# (Note how the boxplots widen when the period includes data from both dat1 and dat2)
```

`draw_calibration` *Draw calibration plot*

Description

Draw calibration plot

Usage

```
draw_calibration(
  true_labels,
  predicted_prob,
  n_bins = 10L,
  bin_method = c("quantile", "equidistant"),
  binclasspos = 2L,
  main = NULL,
  subtitle = NULL,
  xlab = "Mean predicted probability",
  ylab = "Empirical risk",
  show_marginal_x = TRUE,
  marginal_x_y = -0.02,
  marginal_col = NULL,
  marginal_size = 10,
  mode = "markers+lines",
  show_brier = TRUE,
  theme = choose_theme(getOption("rtemis_theme")),
  filename = NULL,
  ...
)
```

Arguments

true_labels	Factor or list of factors with true class labels
predicted_prob	Numeric vector or list of numeric vectors with predicted probabilities
n_bins	Integer: Number of windows to split the data into
bin_method	Character: "quantile" or "equidistant": Method to bin the estimated probabilities.
binclasspos	Integer: Index of the positive class. The convention used in the package is the second level is the positive class.
main	Character: Main title
subtitle	Character: Subtitle, placed bottom right of plot
xlab	Character: x-axis label
ylab	Character: y-axis label
show_marginal_x	Logical: Add marginal plot of distribution of estimated probabilities
marginal_x_y	Numeric: y position of marginal plot
marginal_col	Character: Color of marginal plot
marginal_size	Numeric: Size of marginal plot
mode	Character: "lines", "markers", "lines+markers": How to plot.
show_brier	Logical: If TRUE, add Brier scores to trace names.
theme	Theme object.
filename	Character: Path to save output.
...	Additional arguments passed to draw_scatter

Value

plotly object.

Author(s)

EDG

Examples

```
# Synthetic data with n cases
n <- 500L
true_labels <- factor(sample(c("A", "B"), n, replace = TRUE))
# Synthetic probabilities where A has mean 0.25 and B has mean 0.75
predicted_prob <- ifelse(true_labels == "A",
  rbeta(n, 2, 6),
  rbeta(n, 6, 2)
)
draw_calibration(true_labels, predicted_prob)
```

`draw_confusion` *Plot confusion matrix*

Description

Plot confusion matrix

Usage

```
draw_confusion(
  x,
  xlab = "Predicted",
  ylab = "Reference",
  true_col = "#43A4AC",
  false_col = "#FA9860",
  font_size = 18,
  main = NULL,
  main_y = 1,
  main_yanchor = "bottom",
  theme = choose_theme(getOption("rtemis_theme")),
  margin = list(l = 20, r = 5, b = 5, t = 20),
  filename = NULL,
  file_width = 500,
  file_height = 500,
  file_scale = 1
)
```

Arguments

`x`	`ClassificationMetrics` object produced by classification_metrics or confusion matrix where rows are the reference and columns are the estimated classes. For binary classification, the first row and column are the positive class.
`xlab`	Character: x-axis label. Default is "Predicted".
`ylab`	Character: y-axis label. Default is "Reference".
`true_col`	Color for true positives & true negatives.
`false_col`	Color for false positives & false negatives.
`font_size`	Integer: font size.
`main`	Character: plot title.
`main_y`	Numeric: y position of the title.
`main_yanchor`	Character: y anchor of the title.
`theme`	`Theme` object.
`margin`	List: Plot margins.
`filename`	Character: file name to save the plot. Default is NULL.

file_width	Numeric: width of the file. Default is 500.
file_height	Numeric: height of the file. Default is 500.
file_scale	Numeric: scale of the file. Default is 1.

Value

plotly object.

Author(s)

EDG

Examples

```
# Assume positive class is "b"
true_labels <- factor(c("a", "a", "a", "b", "b", "b", "b", "b", "b", "b"))
predicted_labels <- factor(c("a", "b", "a", "b", "b", "a", "b", "b", "b", "a"))
predicted_prob <- c(0.3, 0.55, 0.45, 0.75, 0.57, 0.3, 0.8, 0.63, 0.62, 0.39)
metrics <- classification_metrics(true_labels, predicted_labels, predicted_prob)
draw_confusion(metrics)
```

---

draw_dist *Draw Distributions using Histograms and Density Plots*

Description

Draw Distributions using Histograms and Density Plots using plotly.

Usage

```
draw_dist(
  x,
  type = c("density", "histogram"),
  mode = c("overlap", "ridge"),
  group = NULL,
  main = NULL,
  xlab = NULL,
  ylab = NULL,
  col = NULL,
  alpha = 0.75,
  plot_bg = NULL,
  theme = choose_theme(getOption("rtemis_theme")),
  palette = getOption("rtemis_palette", "rtms"),
  axes_square = FALSE,
  group_names = NULL,
  font_size = 16,
  font_alpha = 0.8,
```

```
  legend = NULL,
  legend_xy = c(0, 1),
  legend_col = NULL,
  legend_bg = "#FFFFFF00",
  legend_border_col = "#FFFFFF00",
  bargap = 0.05,
  vline = NULL,
  vline_col = theme[["fg"]],
  vline_width = 1,
  vline_dash = "dot",
  text = NULL,
  text_x = 1,
  text_xref = "paper",
  text_xanchor = "left",
  text_y = 1,
  text_yref = "paper",
  text_yanchor = "top",
  text_col = theme[["fg"]],
  margin = list(b = 65, l = 65, t = 50, r = 10, pad = 0),
  automargin_x = TRUE,
  automargin_y = TRUE,
  zerolines = FALSE,
  density_kernel = "gaussian",
  density_bw = "SJ",
  histnorm = c("", "density", "percent", "probability", "probability density"),
  histfunc = c("count", "sum", "avg", "min", "max"),
  hist_n_bins = 20,
  barmode = "overlay",
  ridge_sharex = TRUE,
  ridge_y_labs = FALSE,
  ridge_order_on_mean = TRUE,
  displayModeBar = TRUE,
  modeBar_file_format = "svg",
  width = NULL,
  height = NULL,
  filename = NULL,
  file_width = 500,
  file_height = 500,
  file_scale = 1
)
```

Arguments

x	Numeric vector / data.frame / list: Input. If not a vector, each column / each element is drawn.
type	Character: "density" or "histogram".
mode	Character: "overlap", "ridge". How to plot different groups; on the same axes ("overlap"), or on separate plots with the same x-axis ("ridge").

group	Vector: Will be converted to factor; levels define group members.
main	Character: Main title for the plot.
xlab	Character: Label for the x-axis.
ylab	Character: Label for the y-axis.
col	Color: Colors for the plot.
alpha	Numeric: Alpha transparency for plot elements.
plot_bg	Color: Background color for plot area.
theme	Theme object.
palette	Character: Color palette to use.
axes_square	Logical: If TRUE, draw a square plot to fill the graphic device. Default = FALSE.
group_names	Character: Names for the groups.
font_size	Numeric: Font size for plot text.
font_alpha	Numeric: Alpha transparency for font.
legend	Logical: If TRUE, draw legend. Default = NULL, which will be set to TRUE if x is a list of more than 1 element.
legend_xy	Numeric, vector, length 2: Relative x, y position for legend. Default = c(0, 1).
legend_col	Color: Color for the legend text.
legend_bg	Color: Background color for legend.
legend_border_col	Color: Border color for legend.
bargap	Numeric: The gap between adjacent histogram bars in plot fraction.
vline	Numeric, vector: If defined, draw a vertical line at this x value(s).
vline_col	Color: Color for vline.
vline_width	Numeric: Width for vline.
vline_dash	Character: Type of line to draw: "solid", "dot", "dash", "longdash", "dashdot", or "longdashdot".
text	Character: If defined, add this text over the plot.
text_x	Numeric: x-coordinate for text.
text_xref	Character: "x": text_x refers to plot's x-axis; "paper": text_x refers to plotting area from 0-1.
text_xanchor	Character: "auto", "left", "center", "right".
text_y	Numeric: y-coordinate for text.
text_yref	Character: "y": text_y refers to plot's y-axis; "paper": text_y refers to plotting area from 0-1.
text_yanchor	Character: "auto", "top", "middle", "bottom".
text_col	Color: Color for text.
margin	List: Margins for the plot.
automargin_x	Logical: If TRUE, automatically adjust x-axis margins.

automargin_y	Logical: If TRUE, automatically adjust y-axis margins.
zerolines	Logical: If TRUE, draw lines at y = 0.
density_kernel	Character: Kernel to use for density estimation.
density_bw	Character: Bandwidth to use for density estimation.
histnorm	Character: NULL, "percent", "probability", "density", "probability density".
histfunc	Character: "count", "sum", "avg", "min", "max".
hist_n_bins	Integer: Number of bins to use if type = "histogram".
barmode	Character: Barmode for histogram. One of "overlay", "stack", "relative", "group".
ridge_sharex	Logical: If TRUE, draw single x-axis when mode = "ridge".
ridge_y_labs	Logical: If TRUE, show individual y labels when mode = "ridge".
ridge_order_on_mean	Logical: If TRUE, order groups by mean value when mode = "ridge".
displayModeBar	Logical: If TRUE, display the mode bar.
modeBar_file_format	Character: File format for mode bar. Default = "svg".
width	Numeric: Force plot size to this width. Default = NULL, i.e. fill available space.
height	Numeric: Force plot size to this height. Default = NULL, i.e. fill available space.
filename	Character: Path to file to save static plot.
file_width	Integer: File width in pixels for when filename is set.
file_height	Integer: File height in pixels for when filename is set.
file_scale	Numeric: If saving to file, scale plot by this number.

Details

See docs.rtemis.org/r for detailed documentation.

If input is data.frame, non-numeric variables will be removed.

Value

plotly object.

Author(s)

EDG

Examples

```
# Will automatically use only numeric columns
draw_dist(iris)
draw_dist(iris[["Sepal.Length"]], group = iris[["Species"]])
```

draw_fit *True vs. Predicted Plot*

Description

A draw_scatter wrapper for plotting true vs. predicted values

Usage

```
draw_fit(
  x,
  y,
  xlab = "True",
  ylab = "Predicted",
  fit = "glm",
  se_fit = TRUE,
  axes_square = TRUE,
  axes_equal = TRUE,
  diagonal = TRUE,
  ...
)
```

Arguments

x	Numeric, vector/data.frame/list: True values. If y is NULL and NCOL(x) > 1, first two columns used as x and y, respectively
y	Numeric, vector/data.frame/list: Predicted values
xlab	Character: x-axis label.
ylab	Character: y-axis label.
fit	Character: Fit method.
se_fit	Logical: If TRUE, include standard error of the fit.
axes_square	Logical: If TRUE, draw a square plot.
axes_equal	Logical: If TRUE, set equal scaling for axes.
diagonal	Logical: If TRUE, add diagonal line.
...	Additional arguments passed to draw_scatter

Value

plotly object.

Author(s)

EDG

Examples

```
x <- rnorm(500)
y <- x + rnorm(500)
draw_fit(x, y)
```

`draw_graphD3` *Plot graph using* **networkD3**

Description

Plot graph using **networkD3**

Usage

```
draw_graphD3(
  net,
  groups = NULL,
  color_scale = NULL,
  edge_col = NULL,
  node_col = NULL,
  node_alpha = 0.5,
  edge_alpha = 0.33,
  zoom = TRUE,
  legend = FALSE,
  palette = get_palette(getOption("rtemis_palette")),
  theme = choose_theme(getOption("rtemis_theme")),
  ...
)
```

Arguments

`net`	**igraph** network.
`groups`	Vector, length n nodes indicating group/cluster/community membership of nodes in `net`.
`color_scale`	D3 colorscale (e.g. `networkD3::JS("d3.scaleOrdinal(d3.schemeCategory20b);")`).
`edge_col`	Color for edges.
`node_col`	Color for nodes.
`node_alpha`	Float [0, 1]: Node opacity.
`edge_alpha`	Float [0, 1]: Edge opacity.
`zoom`	Logical: If TRUE, graph is zoomable.
`legend`	Logical: If TRUE, display legend for groups.
`palette`	Character vector: Colors to use.
`theme`	`Theme` object.
`...`	Additional arguments to pass to `networkD3`.

Value

forceNetwork object.

Author(s)

EDG

Examples

```
library(igraph)
g <- make_ring(10)
draw_graphD3(g)
```

---

draw_graphjs *Plot network using* **threejs::graphjs**

Description

Interactive plotting of an **igraph** net using **threejs**.

Usage

```
draw_graphjs(
  net,
  vertex_size = 1,
  vertex_col = NULL,
  vertex_label_col = NULL,
  vertex_label_alpha = 0.66,
  vertex_frame_col = NA,
  vertex_label = NULL,
  vertex_shape = "circle",
  edge_col = NULL,
  edge_alpha = 0.5,
  edge_curved = 0.35,
  edge_width = 2,
 layout = c("fr", "dh", "drl", "gem", "graphopt", "kk", "lgl", "mds", "sugiyama"),
  coords = NULL,
  layout_args = list(),
  cluster = NULL,
  groups = NULL,
  cluster_config = list(),
  cluster_mark_groups = TRUE,
  cluster_color_vertices = FALSE,
  main = "",
  theme = choose_theme(getOption("rtemis_theme")),
  palette = getOption("rtemis_palette", "rtms"),
```

```
  mar = rep(0, 4),
  filename = NULL,
  verbosity = 1L,
  ...
)
```

Arguments

net	**igraph** network.
vertex_size	Numeric: Vertex size.
vertex_col	Color for vertices.
vertex_label_col	Color for vertex labels.
vertex_label_alpha	Numeric: Transparency for vertex_label_col.
vertex_frame_col	Color for vertex border (frame).
vertex_label	Character vector: Vertex labels. Default = NULL, which will keep existing names in net if any. Set to NA to avoid printing vertex labels.
vertex_shape	Character, vector, length 1 or N nodes: Vertex shape. See graphjs("vertex_shape").
edge_col	Color for edges.
edge_alpha	Numeric: Transparency for edges.
edge_curved	Numeric: Curvature of edges.
edge_width	Numeric: Edge thickness.
layout	Character: one of: "fr", "dh", "drl", "gem", "graphopt", "kk", "lgl", "mds", "sugiyama", corresponding to all the available layouts in **igraph**.
coords	Output of precomputed **igraph** layout. If provided, layout is ignored.
layout_args	List of arguments to pass to layout function.
cluster	Character: one of: "edge_betweenness", "fast_greedy", "infomap", "label_prop", "leading_eigen", "louvain", "optimal", "spinglass", "walktrap", corresponding to all the available **igraph** clustering functions.
groups	Output of precomputed **igraph** clustering. If provided, cluster is ignored.
cluster_config	List of arguments to pass to cluster function.
cluster_mark_groups	Logical: If TRUE, draw polygons to indicate clusters, if groups or cluster are defined.
cluster_color_vertices	Logical: If TRUE, color vertices by cluster membership.
main	Character: Main title.
theme	Theme object.
palette	Color vector or name of rtemis palette.
mar	Numeric vector, length 4: par's margin argument.
filename	Character: If provided, save plot to this filepath.
verbosity	Integer: Verbosity level.
...	Extra arguments to pass to igraph::plot.igraph().

Value

scatterplotThree object.

Author(s)

EDG

Examples

```
library(igraph)
g <- make_ring(10)
draw_graphjs(g)
```

---

draw_heatmap *Interactive Heatmaps*

Description

Draw interactive heatmaps using heatmaply.

Usage

```
draw_heatmap(
  x,
  Rowv = TRUE,
  Colv = TRUE,
  cluster = FALSE,
  symm = FALSE,
  cellnote = NULL,
  colorgrad_n = 101,
  colors = NULL,
  space = "rgb",
  lo = "#18A3AC",
  lomid = NULL,
  mid = NULL,
  midhi = NULL,
  hi = "#F48024",
  k_row = 1,
  k_col = 1,
  grid_gap = 0,
  limits = NULL,
  margins = NULL,
  main = NULL,
  xlab = NULL,
  ylab = NULL,
  key_title = NULL,
```

```
  showticklabels = NULL,
  colorbar_len = 0.7,
  plot_method = "plotly",
  theme = choose_theme(getOption("rtemis_theme")),
  row_side_colors = NULL,
  row_side_palette = NULL,
  col_side_colors = NULL,
  col_side_palette = NULL,
  font_size = NULL,
  padding = 0,
  displayModeBar = TRUE,
  modeBar_file_format = "svg",
  filename = NULL,
  file_width = 500,
  file_height = 500,
  file_scale = 1,
  ...
)
```

Arguments

x	Input matrix.
Rowv	Logical or dendrogram. If Logical: Compute dendrogram and reorder rows. Defaults to FALSE. If dendrogram: use as is, without reordering. See more at `heatmaply::heatmaply("Rowv")`.
Colv	Logical or dendrogram. If Logical: Compute dendrogram and reorder columns. Defaults to FALSE. If dendrogram: use as is, without reordering. See more at `heatmaply::heatmaply("Colv")`.
cluster	Logical: If TRUE, set Rowv and Colv to TRUE.
symm	Logical: If TRUE, treat x symmetrically - x must be a square matrix.
cellnote	Matrix with values to be displayed on hover. Defaults to `ddSci(x)`.
colorgrad_n	Integer: Number of colors in gradient. Default = 101.
colors	Character vector: Colors to use in gradient.
space	Character: Color space to use. Default = "rgb".
lo	Character: Color for low values. Default = "#18A3AC".
lomid	Character: Color for low-mid values.
mid	Character: Color for mid values.
midhi	Character: Color for mid-high values.
hi	Character: Color for high values. Default = "#F48024".
k_row	Integer: Number of desired number of groups by which to color dendrogram branches in the rows. Default = 1.
k_col	Integer: Number of desired number of groups by which to color dendrogram branches in the columns. Default = 1.
grid_gap	Integer: Space between cells. Default = 0 (no space).

limits	Float, length 2: Determine color range. Default = NULL, which automatically centers values around 0.
margins	Float, length 4: Heatmap margins.
main	Character: Main title.
xlab	Character: x-axis label.
ylab	Character: y-axis label.
key_title	Character: Title for the color key.
showticklabels	Logical: If TRUE, show tick labels.
colorbar_len	Numeric: Length of the colorbar.
plot_method	Character: Plot method to use. Default = "plotly".
theme	Theme object.
row_side_colors	Data frame: Column names will be label names, cells should be label colors. See heatmaply::heatmaply("row_side_colors").
row_side_palette	Color palette function. See heatmaply::heatmaply("row_side_palette").
col_side_colors	Data frame: Column names will be label names, cells should be label colors. See heatmaply::heatmaply("col_side_colors").
col_side_palette	Color palette function. See heatmaply::heatmaply("col_side_palette").
font_size	Numeric: Font size.
padding	Numeric: Padding between cells.
displayModeBar	Logical: If TRUE, display the plotly mode bar.
modeBar_file_format	Character: File format for image exports from the mode bar.
filename	Character: File name to save the plot.
file_width	Numeric: Width of exported image.
file_height	Numeric: Height of exported image.
file_scale	Numeric: Scale of exported image.
...	Additional arguments to be passed to heatmaply::heatmaply.

Details

See docs.rtemis.org/r for detailed documentation. 'heatmaply' unfortunately forces loading of the 'colorspace' namespace.

Value

plotly object.'

Author(s)

EDG

Examples

```
x <- rnormmat(200, 20)
xcor <- cor(x)
draw_heatmap(xcor)
```

`draw_leaflet` *Plot interactive choropleth map using* **leaflet**

Description

Plot interactive choropleth map using **leaflet**

Usage

```
draw_leaflet(
  fips,
  values,
  names = NULL,
  fillOpacity = 1,
  color_mapping = c("Numeric", "Bin"),
  col_lo = "#0290EE",
  col_hi = "#FE4AA3",
  col_na = "#303030",
  col_highlight = "#FE8A4F",
  col_interpolate = c("linear", "spline"),
  col_bins = 21,
  domain = NULL,
  weight = 0.5,
  color = "black",
  alpha = 1,
  bg_tile_provider = leaflet::providers[["CartoDB.Positron"]],
  bg_tile_alpha = 0.67,
  fg_tile_provider = leaflet::providers[["CartoDB.PositronOnlyLabels"]],
  legend_position = c("topright", "bottomright", "bottomleft", "topleft"),
  legend_alpha = 0.8,
  legend_title = NULL,
  init_lng = -98.5418083333333,
  init_lat = 39.2074138888889,
  init_zoom = 3,
  stroke = TRUE
)
```

Arguments

`fips`	Character vector: FIPS codes. (If numeric, it will be appropriately zero-padded).
`values`	Values to map to `fips`.

names	Character vector: Optional county names to appear on hover along values.
fillOpacity	Float: Opacity for fill colors.
color_mapping	Character: "Numeric" or "Bin".
col_lo	Overlay color mapped to lowest value.
col_hi	Overlay color mapped to highest value.
col_na	Color mapped to NA values.
col_highlight	Hover border color.
col_interpolate	Character: "linear" or "spline".
col_bins	Integer: Number of color bins to create if color_mapping = "Bin".
domain	Limits for mapping colors to values. Default = NULL and set to range.
weight	Float: Weight of county border lines.
color	Color of county border lines.
alpha	Float: Overlay transparency.
bg_tile_provider	Background tile (below overlay colors), one of leaflet::providers.
bg_tile_alpha	Float: Background tile transparency.
fg_tile_provider	Foreground tile (above overlay colors), one of leaflet::providers.
legend_position	Character: One of: "topright", "bottomright", "bottomleft", "topleft".
legend_alpha	Float: Legend box transparency.
legend_title	Character: Defaults to name of values variable.
init_lng	Float: Center map around this longitude (in decimal form). Default = -98.54180833333334 (US geographic center).
init_lat	Float: Center map around this latitude (in decimal form). Default = 39.207413888888894 (US geographic center).
init_zoom	Integer: Initial zoom level (depends on device, i.e. window, size).
stroke	Logical: If TRUE, draw polygon borders.

Value

leaflet object.

Author(s)

EDG

Examples

```
fips <- c(06075, 42101)
population <- c(874961, 1579000)
names <- c("SF", "Philly")
draw_leaflet(fips, population, names)
```

draw_pie *Interactive Pie Chart*

Description

Draw interactive pie charts using `plotly`.

Usage

```
draw_pie(
  x,
  main = NULL,
  xlab = NULL,
  ylab = NULL,
  alpha = 0.8,
  bg = NULL,
  plot_bg = NULL,
  theme = choose_theme(getOption("rtemis_theme")),
  palette = get_palette(getOption("rtemis_palette")),
  category_names = NULL,
  textinfo = "label+percent",
  font_size = 16,
  labs_col = NULL,
  legend = TRUE,
  legend_col = NULL,
  sep_col = NULL,
  margin = list(b = 50, l = 50, t = 50, r = 20),
  padding = 0,
  displayModeBar = TRUE,
  modeBar_file_format = "svg",
  filename = NULL,
  file_width = 500,
  file_height = 500,
  file_scale = 1
)
```

Arguments

x	data.frame: Input: Either a) 1 numeric column with categories defined by rownames, or b) two columns, the first is category names, the second numeric or c) a numeric vector with categories defined using the `category.names` argument.
main	Character: Plot title. Default = NULL, which results in `colnames(x)[1]`.
xlab	Character: x-axis label.
ylab	Character: y-axis label.
alpha	Numeric: Alpha for the pie slices.

bg	Character: Background color.
plot_bg	Character: Plot background color.
theme	Theme object.
palette	Character vector: Colors to use.
category_names	Character, vector, length = NROW(x): Category names. Default = NULL, which uses either rownames(x), or the first column of x if ncol(x) = 2.
textinfo	Character: Info to show over each slice: "label", "percent", "label+percent".
font_size	Integer: Font size for labels.
labs_col	Character: Color of labels.
legend	Logical: If TRUE, show legend.
legend_col	Character: Color for legend.
sep_col	Character: Separator color.
margin	List: Margin settings.
padding	Numeric: Padding between cells.
displayModeBar	Logical: If TRUE, display the plotly mode bar.
modeBar_file_format	Character: File format for image exports from the mode bar.
filename	Character: File name to save plot.
file_width	Integer: Width for saved file.
file_height	Integer: Height for saved file.
file_scale	Numeric: Scale for saved file.

Value

plotly object.

Author(s)

EDG

Examples

```
draw_pie(VADeaths[, 1, drop = FALSE])
```

draw_protein *Plot an amino acid sequence with annotations*

Description

Plot an amino acid sequence with multiple site and/or region annotations.

Usage

```
draw_protein(
  x,
  site = NULL,
  region = NULL,
  ptm = NULL,
  cleavage_site = NULL,
  variant = NULL,
  disease_variants = NULL,
  n_per_row = NULL,
  main = NULL,
  main_xy = c(0.055, 0.975),
  main_xref = "paper",
  main_yref = "paper",
  main_xanchor = "middle",
  main_yanchor = "top",
  layout = c("simple", "grid", "1curve", "2curve"),
  show_markers = TRUE,
  show_labels = TRUE,
  font_size = 18,
  label_col = NULL,
  scatter_mode = "markers+lines",
  marker_size = 28,
  marker_col = NULL,
  marker_alpha = 1,
  marker_symbol = "circle",
  line_col = NULL,
  line_alpha = 1,
  line_width = 2,
  show_full_names = TRUE,
  region_scatter_mode = "markers+lines",
  region_style = 3,
  region_marker_size = marker_size,
  region_marker_alpha = 0.6,
  region_marker_symbol = "circle",
  region_line_dash = "solid",
  region_line_shape = "line",
  region_line_smoothing = 1,
  region_line_width = 1,
```

```
region_line_alpha = 0.6,
theme = choose_theme(getOption("rtemis_theme")),
region_palette = getOption("rtemis_palette", "rtms"),
region_outline_only = FALSE,
region_outline_pad = 2,
region_pad = 0.35,
region_fill_alpha = 0.1666666,
region_fill_shape = "line",
region_fill_smoothing = 1,
bpadcx = 0.5,
bpadcy = 0.5,
site_marker_size = marker_size,
site_marker_symbol = marker_symbol,
site_marker_alpha = 1,
site_border_width = 1.5,
site_palette = getOption("rtemis_palette", "rtms"),
variant_col = "#FA6E1E",
disease_variant_col = "#E266AE",
showlegend_ptm = TRUE,
ptm_col = NULL,
ptm_symbol = "circle",
ptm_offset = 0.12,
ptm_pad = 0.35,
ptm_marker_size = marker_size/4.5,
clv_col = NULL,
clv_symbol = "triangle-down",
clv_offset = 0.12,
clv_pad = 0.35,
clv_marker_size = marker_size/4,
annotate_position_every = 10,
annotate_position_alpha = 0.5,
annotate_position_ay = -0.4 * marker_size,
position_font_size = font_size - 6,
legend_xy = c(0.97, 0.954),
legend_xanchor = "left",
legend_yanchor = "top",
legend_orientation = "v",
legend_col = NULL,
legend_bg = "#FFFFFF00",
legend_border_col = "#FFFFFF00",
legend_borderwidth = 0,
legend_group_gap = 0,
margin = list(b = 0, l = 0, t = 0, r = 0, pad = 0),
showgrid_x = FALSE,
showgrid_y = FALSE,
automargin_x = TRUE,
automargin_y = TRUE,
xaxis_autorange = TRUE,
```

```
  yaxis_autorange = "reversed",
  scaleanchor_y = "x",
  scaleratio_y = 1,
  hoverlabel_align = "left",
  displayModeBar = TRUE,
  modeBar_file_format = "svg",
  scrollZoom = TRUE,
  filename = NULL,
  file_width = 1320,
  file_height = 990,
  file_scale = 1,
  width = NULL,
  height = NULL,
  verbosity = 1L
)
```

Arguments

x	Character vector: amino acid sequence (1-letter abbreviations) OR a3 object OR Character: path to JSON file OR Character: UniProt accession number.
site	Named list of lists with indices of sites. These will be highlighted by coloring the border of markers.
region	Named list of lists with indices of regions. These will be highlighted by coloring the markers and lines of regions using the palette colors.
ptm	List of post-translational modifications.
cleavage_site	List of cleavage sites.
variant	List of variant information.
disease_variants	List of disease variant information.
n_per_row	Integer: Number of amino acids to show per row.
main	Character: Main title.
main_xy	Numeric vector, length 2: x and y coordinates for title. e.g. if main_xref and main_yref are "paper": c(0.055, .975) is top left, c(.5, .975) is top and middle.
main_xref	Character: xref for title.
main_yref	Character: yref for title.
main_xanchor	Character: xanchor for title.
main_yanchor	Character: yanchor for title.
layout	Character: "1curve", "grid": type of layout to use.
show_markers	Logical: If TRUE, show amino acid markers.
show_labels	Logical: If TRUE, annotate amino acids with elements.
font_size	Integer: Font size for labels.
label_col	Color for labels.

`scatter_mode` Character: Mode for scatter plot.

`marker_size` Integer: Size of markers.

`marker_col` Color for markers.

`marker_alpha` Numeric: Alpha for markers.

`marker_symbol` Character: Symbol for markers.

`line_col` Color for lines.

`line_alpha` Numeric: Alpha for lines.

`line_width` Numeric: Width for lines.

`show_full_names`
Logical: If TRUE, show full names of amino acids.

`region_scatter_mode`
Character: Mode for scatter plot.

`region_style` Integer: Style for regions.

`region_marker_size`
Integer: Size of region markers.

`region_marker_alpha`
Numeric: Alpha for region markers.

`region_marker_symbol`
Character: Symbol for region markers.

`region_line_dash`
Character: Dash for region lines.

`region_line_shape`
Character: Shape for region lines.

`region_line_smoothing`
Numeric: Smoothing for region lines.

`region_line_width`
Numeric: Width for region lines.

`region_line_alpha`
Numeric: Alpha for region lines.

`theme` `Theme` object.

`region_palette` Named list of colors for regions.

`region_outline_only`
Logical: If TRUE, only show outline of regions.

`region_outline_pad`
Numeric: Padding for region outline.

`region_pad` Numeric: Padding for region.

`region_fill_alpha`
Numeric: Alpha for region fill.

`region_fill_shape`
Character: Shape for region fill.

`region_fill_smoothing`
Numeric: Smoothing for region fill.

bpadcx Numeric: Padding for region border.

bpadcy Numeric: Padding for region border.

site_marker_size
Integer: Size of site markers.

site_marker_symbol
Character: Symbol for site markers.

site_marker_alpha
Numeric: Alpha for site markers.

site_border_width
Numeric: Width for site borders.

site_palette Named list of colors for sites.

variant_col Color for variants.

disease_variant_col
Color for disease variants.

showlegend_ptm Logical: If TRUE, show legend for PTMs.

ptm_col Named list of colors for PTMs.

ptm_symbol Character: Symbol for PTMs.

ptm_offset Numeric: Offset for PTMs.

ptm_pad Numeric: Padding for PTMs.

ptm_marker_size
Integer: Size of PTM markers.

clv_col Color for cleavage site annotations.

clv_symbol Character: Symbol for cleavage site annotations.

clv_offset Numeric: Offset for cleavage site annotations.

clv_pad Numeric: Padding for cleavage site annotations.

clv_marker_size
Integer: Size of cleavage site annotation markers.

annotate_position_every
Integer: Annotate every nth position.

annotate_position_alpha
Numeric: Alpha for position annotations.

annotate_position_ay
Numeric: Y offset for position annotations.

position_font_size
Integer: Font size for position annotations.

legend_xy Numeric vector, length 2: x and y coordinates for legend.

legend_xanchor Character: xanchor for legend.

legend_yanchor Character: yanchor for legend.

legend_orientation
Character: Orientation for legend.

legend_col Color for legend.

`legend_bg` Color for legend background.

`legend_border_col` Color for legend border.

`legend_borderwidth` Numeric: Width for legend border.

`legend_group_gap` Numeric: Gap between legend groups.

`margin` List: Margin settings.

`showgrid_x` Logical: If TRUE, show x grid.

`showgrid_y` Logical: If TRUE, show y grid.

`automargin_x` Logical: If TRUE, use automatic margin for x axis.

`automargin_y` Logical: If TRUE, use automatic margin for y axis.

`xaxis_autorange` Logical: If TRUE, use automatic range for x axis.

`yaxis_autorange` Character: If TRUE, use automatic range for y axis.

`scaleanchor_y` Character: Scale anchor for y axis.

`scaleratio_y` Numeric: Scale ratio for y axis.

`hoverlabel_align` Character: Alignment for hover label.

`displayModeBar` Logical: If TRUE, display mode bar.

`modeBar_file_format` Character: File format for mode bar.

`scrollZoom` Logical: If TRUE, enable scroll zoom.

`filename` Character: File name to save plot.

`file_width` Integer: Width for saved file.

`file_height` Integer: Height for saved file.

`file_scale` Numeric: Scale for saved file.

`width` Integer: Width for plot.

`height` Integer: Height for plot.

`verbosity` Integer: Verbosity level.

Value

`plotly` object.

Author(s)

EDG

Examples

```
## Not run:
  # Reads sequence from UniProt server
  tau <- seqinr::read.fasta("https://rest.uniprot.org/uniprotkb/P10636.fasta",
    seqtype = "AA"
  )
  draw_protein(as.character(tau[[1]]))

  # or directly using the UniProt accession number:
  draw_protein("P10636")

## End(Not run)
```

`draw_pvals` *Barplot p-values using draw_bar*

Description

Plot 1 - p-values as a barplot

Usage

```
draw_pvals(
  x,
  xnames = NULL,
  yname = NULL,
  p_adjust_method = "none",
  pval_hline = 0.05,
  hline_col = rt_red,
  hline_dash = "dash",
  ...
)
```

Arguments

`x`	Float, vector: p-values.
`xnames`	Character, vector: feature names.
`yname`	Character: outcome name.
`p_adjust_method`	Character: method for p.adjust.
`pval_hline`	Float: Significance level at which to plot horizontal line.
`hline_col`	Color for `pval_hline`.
`hline_dash`	Character: type of line to draw.
`...`	Additional arguments passed to draw_bar.

Value

plotly object.

Author(s)

EDG

Examples

```
draw_pvals(c(0.01, 0.02, 0.03), xnames = c("Feature1", "Feature2", "Feature3"))
```

---

draw_roc *Draw ROC curve*

Description

Draw ROC curve

Usage

```
draw_roc(
  true_labels,
  predicted_prob,
  multiclass_fill_labels = TRUE,
  main = NULL,
  theme = choose_theme(getOption("rtemis_theme")),
  palette = get_palette(getOption("rtemis_palette")),
  legend = TRUE,
  legend_title = "Group (AUC)",
  legend_xy = c(1, 0),
  legend_xanchor = "right",
  legend_yanchor = "bottom",
  auc_dp = 3L,
  xlim = c(-0.05, 1.05),
  ylim = c(-0.05, 1.05),
  diagonal = TRUE,
  diagonal_col = NULL,
  axes_square = TRUE,
  filename = NULL,
  ...
)
```

Arguments

true_labels	Factor: True outcome labels.
predicted_prob	Numeric vector [0, 1]: Predicted probabilities for the positive class (i.e. second level of outcome). Or, for multiclass, a matrix of predicted probabilities with one column per class. Or, a list of such vectors/matrices to draw multiple ROC curves on the same plot.
multiclass_fill_labels	Logical: If TRUE, fill in labels for multiclass ROC curves. If FALSE, column names of predicted_prob must match levels of true_labels.
main	Character: Main title for the plot.
theme	Theme object.
palette	Character vector: Colors to use.
legend	Logical: If TRUE, draw legend.
legend_title	Character: Title for the legend.
legend_xy	Numeric vector: Position of the legend in the form c(x, y).
legend_xanchor	Character: X anchor for the legend.
legend_yanchor	Character: Y anchor for the legend.
auc_dp	Integer: Number of decimal places for AUC values.
xlim	Numeric vector: Limits for the x-axis.
ylim	Numeric vector: Limits for the y-axis.
diagonal	Logical: If TRUE, draw diagonal line.
diagonal_col	Character: Color for the diagonal line.
axes_square	Logical: If TRUE, make axes square.
filename	Character: If provided, save the plot to this file.
...	Additional arguments passed to draw_scatter.

Value

plotly object.

Author(s)

EDG

Examples

```
# Binary classification
true_labels <- factor(c("A", "B", "A", "A", "B", "A", "B", "B", "A", "B"))
predicted_prob <- c(0.1, 0.4, 0.35, 0.8, 0.65, 0.2, 0.9, 0.55, 0.3, 0.7)
draw_roc(true_labels, predicted_prob)
```

draw_scatter *Interactive Scatter Plots*

Description

Draw interactive scatter plots using `plotly`.

Usage

```
draw_scatter(
  x,
  y = NULL,
  fit = NULL,
  se_fit = FALSE,
  se_times = 1.96,
  include_fit_name = TRUE,
  cluster = NULL,
  cluster_config = list(k = 2),
  group = NULL,
  rsq = TRUE,
  mode = "markers",
  order_on_x = NULL,
  main = NULL,
  subtitle = NULL,
  xlab = NULL,
  ylab = NULL,
  alpha = NULL,
  theme = choose_theme(getOption("rtemis_theme")),
  palette = get_palette(getOption("rtemis_palette")),
  axes_square = FALSE,
  group_names = NULL,
  font_size = 16,
  marker_col = NULL,
  marker_size = 8,
  symbol = "circle",
  fit_col = NULL,
  fit_alpha = 0.8,
  fit_lwd = 2.5,
  line_shape = "linear",
  se_col = NULL,
  se_alpha = 0.4,
  scatter_type = "scatter",
  show_marginal_x = FALSE,
  show_marginal_y = FALSE,
  marginal_x = x,
  marginal_y = y,
  marginal_x_y = NULL,
```

```
    marginal_y_x = NULL,
    marginal_col = NULL,
    marginal_alpha = 0.333,
    marginal_size = 10,
    legend = NULL,
    legend_title = NULL,
    legend_trace = TRUE,
    legend_xy = c(0, 0.98),
    legend_xanchor = "left",
    legend_yanchor = "auto",
    legend_orientation = "v",
    legend_col = NULL,
    legend_bg = "#FFFFFF00",
    legend_border_col = "#FFFFFF00",
    legend_borderwidth = 0,
    legend_group_gap = 0,
    x_showspikes = FALSE,
    y_showspikes = FALSE,
    spikedash = "solid",
    spikemode = "across",
    spikesnap = "hovered data",
    spikecolor = NULL,
    spikethickness = 1,
    margin = list(b = 65, l = 65, t = 50, r = 10, pad = 0),
    main_y = 1.01,
    main_yanchor = "bottom",
    subtitle_x = 0.02,
    subtitle_y = 0.99,
    subtitle_xref = "paper",
    subtitle_yref = "paper",
    subtitle_xanchor = "left",
    subtitle_yanchor = "top",
    automargin_x = TRUE,
    automargin_y = TRUE,
    xlim = NULL,
    ylim = NULL,
    axes_equal = FALSE,
    diagonal = FALSE,
    diagonal_col = NULL,
    diagonal_dash = "dot",
    diagonal_alpha = 0.66,
    fit_params = NULL,
    vline = NULL,
    vline_col = theme[["fg"]],
    vline_width = 1,
    vline_dash = "dot",
    hline = NULL,
    hline_col = theme[["fg"]],
```

```
  hline_width = 1,
  hline_dash = "dot",
  hovertext = NULL,
  width = NULL,
  height = NULL,
  displayModeBar = TRUE,
  modeBar_file_format = "svg",
  scrollZoom = TRUE,
  filename = NULL,
  file_width = 500,
  file_height = 500,
  file_scale = 1,
  verbosity = 0L
)
```

Arguments

x	Numeric, vector/data.frame/list: x-axis data. If y is NULL and NCOL(x) > 1, first two columns used as x and y, respectively.
y	Numeric, vector/data.frame/list: y-axis data.
fit	Character: Fit method.
se_fit	Logical: If TRUE, include standard error of the fit.
se_times	Numeric: Multiplier for standard error.
include_fit_name	Logical: If TRUE, include fit name in legend.
cluster	Character: Clustering method.
cluster_config	List: Config for clustering.
group	Factor: Grouping variable.
rsq	Logical: If TRUE, print R-squared values in legend if fit is set.
mode	Character, vector: "markers", "lines", "markers+lines".
order_on_x	Logical: If TRUE, order x and y on x.
main	Character: Main title.
subtitle	Character: Subtitle.
xlab	Character: x-axis label.
ylab	Character: y-axis label.
alpha	Numeric: Alpha for markers.
theme	Theme object.
palette	Character vector: Colors to use.
axes_square	Logical: If TRUE, draw a square plot.
group_names	Character: Names for groups.
font_size	Numeric: Font size.
marker_col	Color for markers.

`marker_size` Numeric: Marker size.

`symbol` Character: Marker symbol.

`fit_col` Color for fit line.

`fit_alpha` Numeric: Alpha for fit line.

`fit_lwd` Numeric: Line width for fit line.

`line_shape` Character: Line shape for line plots. Options: "linear", "hv", "vh", "hvh", "vhv".

`se_col` Color for standard error band.

`se_alpha` Numeric: Alpha for standard error band.

`scatter_type` Character: Scatter plot type.

`show_marginal_x`
Logical: If TRUE, add marginal distribution line markers on x-axis.

`show_marginal_y`
Logical: If TRUE, add marginal distribution line markers on y-axis.

`marginal_x` Numeric: Data for marginal distribution on x-axis.

`marginal_y` Numeric: Data for marginal distribution on y-axis.

`marginal_x_y` Numeric: Y position of marginal markers on x-axis.

`marginal_y_x` Numeric: X position of marginal markers on y-axis.

`marginal_col` Color for marginal markers.

`marginal_alpha` Numeric: Alpha for marginal markers.

`marginal_size` Numeric: Size of marginal markers.

`legend` Logical: If TRUE, draw legend.

`legend_title` Character: Title for legend.

`legend_trace` Logical: If TRUE, draw legend trace. (For when you have `fit` and don't want a trace for the markers.)

`legend_xy` Numeric: Position of legend.

`legend_xanchor` Character: X anchor for legend.

`legend_yanchor` Character: Y anchor for legend.

`legend_orientation`
Character: Orientation of legend.

`legend_col` Color for legend text.

`legend_bg` Color for legend background.

`legend_border_col`
Color for legend border.

`legend_borderwidth`
Numeric: Border width for legend.

`legend_group_gap`
Numeric: Gap between legend groups.

`x_showspikes` Logical: If TRUE, show spikes on x-axis.

`y_showspikes` Logical: If TRUE, show spikes on y-axis.

spikedash Character: Dash type for spikes.

spikemode Character: Spike mode.

spikesnap Character: Spike snap mode.

spikecolor Color for spikes.

spikethickness Numeric: Thickness of spikes.

margin List: Plot margins.

main_y Numeric: Y position of main title.

main_yanchor Character: Y anchor for main title.

subtitle_x Numeric: X position of subtitle.

subtitle_y Numeric: Y position of subtitle.

subtitle_xref Character: X reference for subtitle.

subtitle_yref Character: Y reference for subtitle.

subtitle_xanchor Character: X anchor for subtitle.

subtitle_yanchor Character: Y anchor for subtitle.

automargin_x Logical: If TRUE, automatically adjust x-axis margins.

automargin_y Logical: If TRUE, automatically adjust y-axis margins.

xlim Numeric: Limits for x-axis.

ylim Numeric: Limits for y-axis.

axes_equal Logical: If TRUE, set equal scaling for axes.

diagonal Logical: If TRUE, add diagonal line.

diagonal_col Color for diagonal line.

diagonal_dash Character: "solid", "dash", "dot", "dashdot", "longdash", "longdashdot". Dash type for diagonal line.

diagonal_alpha Numeric: Alpha for diagonal line.

fit_params Hyperparameters for fit.

vline Numeric: X position for vertical line.

vline_col Color for vertical line.

vline_width Numeric: Width for vertical line.

vline_dash Character: Dash type for vertical line.

hline Numeric: Y position for horizontal line.

hline_col Color for horizontal line.

hline_width Numeric: Width for horizontal line.

hline_dash Character: Dash type for horizontal line.

hovertext List: Hover text for markers.

width Numeric: Width of plot.

height Numeric: Height of plot.

displayModeBar	Logical: If TRUE, display mode bar.
modeBar_file_format	Character: File format for mode bar.
scrollZoom	Logical: If TRUE, enable scroll zoom.
filename	Character: Filename to save plot.
file_width	Numeric: Width of saved file.
file_height	Numeric: Height of saved file.
file_scale	Numeric: Scale of saved file.
verbosity	Integer: Verbosity level.

Value

plotly object.

Author(s)

EDG

Examples

```
draw_scatter(iris$Sepal.Length, iris$Petal.Length,
  fit = "gam", se_fit = TRUE, group = iris$Species
)
```

---

draw_spectrogram *Interactive Spectrogram*

Description

Draw interactive spectrograms using plotly

Usage

```
draw_spectrogram(
  x,
  y,
  z,
  colorgrad_n = 101,
  colors = NULL,
  xlab = "Time",
  ylab = "Frequency",
  zlab = "Power",
  hover_xlab = xlab,
  hover_ylab = ylab,
  hover_zlab = zlab,
```

```
  zmin = NULL,
  zmax = NULL,
  zauto = TRUE,
  hoverlabel_align = "right",
  colorscale = "Jet",
  colorbar_y = 0.5,
  colorbar_yanchor = "middle",
  colorbar_xpad = 0,
  colorbar_ypad = 0,
  colorbar_len = 0.75,
  colorbar_title_side = "bottom",
  showgrid = FALSE,
  space = "rgb",
  lo = "#18A3AC",
  lomid = NULL,
  mid = NULL,
  midhi = NULL,
  hi = "#F48024",
  grid_gap = 0,
  limits = NULL,
  main = NULL,
  key_title = NULL,
  showticklabels = NULL,
  theme = choose_theme(getOption("rtemis_theme")),
  font_size = NULL,
  padding = 0,
  displayModeBar = TRUE,
  modeBar_file_format = "svg",
  filename = NULL,
  file_width = 500,
  file_height = 500,
  file_scale = 1,
  ...
)
```

Arguments

x	Numeric: Time.
y	Numeric: Frequency.
z	Numeric: Power.
colorgrad_n	Integer: Number of colors in the gradient.
colors	Character: Custom colors for the gradient.
xlab	Character: x-axis label.
ylab	Character: y-axis label.
zlab	Character: z-axis label.
hover_xlab	Character: x-axis label for hover.

hover_ylab	Character: y-axis label for hover.
hover_zlab	Character: z-axis label for hover.
zmin	Numeric: Minimum value for color scale.
zmax	Numeric: Maximum value for color scale.
zauto	Logical: If TRUE, automatically set zmin and zmax.
hoverlabel_align	Character: Alignment of hover labels.
colorscale	Character: Color scale.
colorbar_y	Numeric: Y position of colorbar.
colorbar_yanchor	Character: Y anchor of colorbar.
colorbar_xpad	Numeric: X padding of colorbar.
colorbar_ypad	Numeric: Y padding of colorbar.
colorbar_len	Numeric: Length of colorbar.
colorbar_title_side	Character: Side of colorbar title.
showgrid	Logical: If TRUE, show grid.
space	Character: Color space for gradient.
lo	Character: Low color for gradient.
lomid	Character: Low-mid color for gradient.
mid	Character: Mid color for gradient.
midhi	Character: Mid-high color for gradient.
hi	Character: High color for gradient.
grid_gap	Integer: Space between cells.
limits	Numeric, length 2: Determine color range. Default = NULL, which automatically centers values around 0.
main	Character: Main title.
key_title	Character: Title of the key.
showticklabels	Logical: If TRUE, show tick labels.
theme	Theme object.
font_size	Numeric: Font size.
padding	Numeric: Padding between cells.
displayModeBar	Logical: If TRUE, display the plotly mode bar.
modeBar_file_format	Character: File format for image exports from the mode bar.
filename	Character: Filename to save the plot. Default is NULL.
file_width	Numeric: Width of exported image.
file_height	Numeric: Height of exported image.
file_scale	Numeric: Scale of exported image.
...	Additional arguments to be passed to heatmaply::heatmaply.

Details

To set custom colors, use a minimum of `lo` and `hi`, optionally also `lomid`, `mid`, `midhi` colors and set `colorscale = NULL`.

Value

`plotly` object.

Author(s)

EDG

Examples

```
# Example data
time <- seq(0, 10, length.out = 100)
freq <- seq(1, 100, length.out = 100)
power <- outer(time, freq, function(t, f) sin(t) * cos(f))
draw_spectrogram(
  x = time,
  y = freq,
  z = power
)
```

`draw_survfit` *Draw a survfit object*

Description

Draw a `survfit` object using draw_scatter.

Usage

```
draw_survfit(
  x,
  mode = "lines",
  symbol = "cross",
  line_shape = "hv",
  xlim = NULL,
  ylim = NULL,
  xlab = "Time",
  ylab = "Survival",
  main = NULL,
  legend_xy = c(1, 1),
  legend_xanchor = "right",
  legend_yanchor = "top",
  theme = choose_theme(getOption("rtemis_theme")),
```

```
  nrisk_table = FALSE,
  filename = NULL,
  ...
)
```

Arguments

x	survfit object created by survival::survfit.
mode	Character, vector: "markers", "lines", "markers+lines".
symbol	Character: Symbol to use for the points.
line_shape	Character: Line shape for line plots. Options: "linear", "hv", "vh", "hvh", "vhv".
xlim	Numeric vector of length 2: x-axis limits.
ylim	Numeric vector of length 2: y-axis limits.
xlab	Character: x-axis label.
ylab	Character: y-axis label.
main	Character: Main title.
legend_xy	Numeric: Position of legend.
legend_xanchor	Character: X anchor for legend.
legend_yanchor	Character: Y anchor for legend.
theme	Theme object.
nrisk_table	Logical: If TRUE, subplot a table of the number at risk at each time point.
filename	Character: Filename to save plot.
...	Additional arguments passed to draw_scatter.

Value

plotly object.

Author(s)

EDG

Examples

```
# Get the lung dataset
data(cancer, package = "survival")
sf1 <- survival::survfit(survival::Surv(time, status) ~ 1, data = lung)
draw_survfit(sf1)
sf2 <- survival::survfit(survival::Surv(time, status) ~ sex, data = lung)
draw_survfit(sf2)
# with N at risk table
draw_survfit(sf2)
```

`draw_table` *Simple HTML table*

Description

Draw an html table using `plotly`

Usage

```
draw_table(
  x,
  .ddSci = TRUE,
  main = NULL,
  main_col = "black",
  main_x = 0,
  main_xanchor = "auto",
  fill_col = "#18A3AC",
  table_bg = "white",
  bg = "white",
  line_col = "white",
  lwd = 1,
  header_font_col = "white",
  table_font_col = "gray20",
  font_size = 14,
  font_family = "Helvetica Neue",
  margin = list(l = 0, r = 5, t = 30, b = 0, pad = 0)
)
```

Arguments

`x`	data.frame: Table to draw
`.ddSci`	Logical: If TRUE, apply ddSci to numeric columns.
`main`	Character: Table tile.
`main_col`	Color: Title color.
`main_x`	Float [0, 1]: Align title: 0: left, .5: center, 1: right.
`main_xanchor`	Character: "auto", "left", "right": plotly's layout xanchor for title.
`fill_col`	Color: Used to fill header with column names and first column with row names.
`table_bg`	Color: Table background.
`bg`	Color: Background.
`line_col`	Color: Line color.
`lwd`	Float: Line width.
`header_font_col`	Color: Header font color.

table_font_col	Color: Table font color.
font_size	Integer: Font size.
font_family	Character: Font family.
margin	List: plotly's margins.

Value

plotly object.

Author(s)

EDG

Examples

```
 df <- data.frame(
   Name = c("Alice", "Bob", "Charlie"),
   Age = c(25, 30, 35),
   Score = c(90.5, 85.0, 88.0)
)
p <- draw_table(
  df,
  main = "Sample Table",
  main_col = "#00b2b2"
)
```

---

draw_ts *Interactive Timeseries Plots*

Description

Draw interactive timeseries plots using plotly

Usage

```
draw_ts(
  x,
  time,
  window = 7L,
  group = NULL,
  roll_fn = c("mean", "median", "max", "none"),
  roll_col = NULL,
  roll_alpha = 1,
  roll_lwd = 2,
  roll_name = NULL,
  alpha = NULL,
  align = "center",
```

```
  group_names = NULL,
  xlab = "Time",
  n_xticks = 12,
  scatter_type = "scatter",
  legend = TRUE,
  x_showspikes = TRUE,
  y_showspikes = FALSE,
  spikedash = "solid",
  spikemode = "across",
  spikesnap = "hovered data",
  spikecolor = NULL,
  spikethickness = 1,
  displayModeBar = TRUE,
  modeBar_file_format = "svg",
  theme = choose_theme(getOption("rtemis_theme")),
  palette = getOption("rtemis_palette", "rtms"),
  filename = NULL,
  file_width = 500,
  file_height = 500,
  file_scale = 1,
  ...
)
```

Arguments

x	Numeric vector of values to plot or list of vectors
time	Numeric or Date vector of time corresponding to values of x
window	Integer: apply roll_fn over this many units of time
group	Factor defining groups
roll_fn	Character: "mean", "median", "max", or "sum": Function to apply on rolling windows of x
roll_col	Color for rolling line
roll_alpha	Numeric: transparency for rolling line
roll_lwd	Numeric: width of rolling line
roll_name	Rolling function name (for annotation)
alpha	Numeric [0, 1]: Transparency
align	Character: "center", "right", or "left"
group_names	Character vector of group names
xlab	Character: x-axis label
n_xticks	Integer: number of x-axis ticks to use (approximately)
scatter_type	Character: "scatter" or "lines"
legend	Logical: If TRUE, show legend
x_showspikes	Logical: If TRUE, show x-axis spikes on hover
y_showspikes	Logical: If TRUE, show y-axis spikes on hover

spikedash	Character: dash type string ("solid", "dot", "dash", "longdash", "dashdot", or "longdashdot") or a dash length list in px (eg "5px,10px,2px,2px")
spikemode	Character: If "toaxis", spike line is drawn from the data point to the axis the series is plotted on. If "across", the line is drawn across the entire plot area, and supercedes "toaxis". If "marker", then a marker dot is drawn on the axis the series is plotted on
spikesnap	Character: "data", "cursor", "hovered data". Determines whether spikelines are stuck to the cursor or to the closest datapoints.
spikecolor	Color for spike lines
spikethickness	Numeric: spike line thickness
displayModeBar	Logical: If TRUE, display plotly's modebar
modeBar_file_format	Character: modeBar image export file format
theme	Theme object.
palette	Character: palette name, or list of colors
filename	Character: Path to filename to save plot
file_width	Numeric: image export width
file_height	Numeric: image export height
file_scale	Numeric: image export scale
...	Additional arguments to be passed to draw_scatter

Value

plotly object.

Author(s)

EDG

Examples

```
time <- sample(seq(as.Date("2020-03-01"), as.Date("2020-09-23"), length.out = 140))
x1 <- rnorm(140)
x2 <- rnorm(140, 1, 1.2)
# Single timeseries
draw_ts(x1, time)
# Multiple timeseries input as list
draw_ts(list(Alpha = x1, Beta = x2), time)
# Multiple timeseries grouped by group, different lengths
time1 <- sample(seq(as.Date("2020-03-01"), as.Date("2020-07-23"), length.out = 100))
time2 <- sample(seq(as.Date("2020-05-01"), as.Date("2020-09-23"), length.out = 140))
time <- c(time1, time2)
x <- c(rnorm(100), rnorm(140, 1, 1.5))
group <- c(rep("Alpha", 100), rep("Beta", 140))
draw_ts(x, time, 7, group)
```

`draw_varimp` *Interactive Variable Importance Plot*

Description

Plot variable importance using `plotly`

Usage

```
draw_varimp(
  x,
  names = NULL,
  main = NULL,
  type = c("bar", "line"),
  xlab = NULL,
  ylab = NULL,
  plot_top = 1,
  orientation = "v",
  line_width = 12,
  labelify = TRUE,
  alpha = 1,
  palette = get_palette(getOption("rtemis_palette")),
  mar = NULL,
  font_size = 16,
  axis_font_size = 14,
  theme = choose_theme(getOption("rtemis_theme")),
  showlegend = TRUE,
  filename = NULL,
  file_width = 500,
  file_height = 500,
  file_scale = 1
)
```

Arguments

`x`	Numeric vector: Input.
`names`	Vector, string: Names of features.
`main`	Character: Main title.
`type`	Character: "bar" or "line".
`xlab`	Character: x-axis label.
`ylab`	Character: y-axis label.
`plot_top`	Integer: Plot this many top features.
`orientation`	Character: "h" or "v".
`line_width`	Numeric: Line width.

labelify	Logical: If TRUE, labelify feature names.
alpha	Numeric: Transparency.
palette	Character vector: Colors to use.
mar	Vector, numeric, length 4: Plot margins in pixels (NOT inches).
font_size	Integer: Overall font size to use (essentially for the title at this point).
axis_font_size	Integer: Font size to use for axis labels and tick labels.
theme	Theme object.
showlegend	Logical: If TRUE, show legend.
filename	Character: Path to save the plot image.
file_width	Numeric: Width of the saved plot image.
file_height	Numeric: Height of the saved plot image.
file_scale	Numeric: Scale of the saved plot image.

Details

A simple plotly wrapper to plot horizontal barplots, sorted by value, which can be used to visualize variable importance, model coefficients, etc.

Value

plotly object.

Author(s)

EDG

Examples

```
# synthetic data
x <- rnorm(10)
names(x) <- paste0("Feature_", seq(x))
draw_varimp(x)
draw_varimp(x, orientation = "h")
```

---

draw_volcano *Volcano Plot*

Description

Volcano Plot

Usage

```
draw_volcano(
  x,
  pvals,
  xnames = NULL,
  group = NULL,
  x_thresh = 0,
  p_thresh = 0.05,
 p_adjust_method = c("holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr",
    "none"),
  p_transform = function(x) -log10(x),
  legend = NULL,
  legend_lo = NULL,
  legend_hi = NULL,
  label_lo = "Low",
  label_hi = "High",
  main = NULL,
  xlab = NULL,
  ylab = NULL,
  margin = list(b = 65, l = 65, t = 50, r = 10, pad = 0),
  xlim = NULL,
  ylim = NULL,
  alpha = NULL,
  hline = NULL,
  hline_col = NULL,
  hline_width = 1,
  hline_dash = "solid",
  hline_annotate = NULL,
  hline_annotation_x = 1,
  theme = choose_theme(getOption("rtemis_theme")),
  annotate = TRUE,
  annotate_col = theme[["labs_col"]],
  font_size = 16,
  palette = NULL,
  legend_x_lo = NULL,
  legend_x_hi = NULL,
  legend_y = 0.97,
  annotate_n = 7L,
  ax_lo = NULL,
  ay_lo = NULL,
  ax_hi = NULL,
  ay_hi = NULL,
  annotate_alpha = 0.7,
  hovertext = NULL,
  displayModeBar = "hover",
  filename = NULL,
  file_width = 500,
  file_height = 500,
```

```
  file_scale = 1,
  verbosity = 1L,
  ...
)
```

Arguments

x	Numeric vector: Input values, e.g. log2 fold change, coefficients, etc.
pvals	Numeric vector: p-values.
xnames	Character vector: x names.
group	Optional factor: Used to color code points. If NULL, significant points below x_thresh, non-significant points, and significant points above x_thresh will be plotted with the first, second and third color of palette.
x_thresh	Numeric x-axis threshold separating low from high.
p_thresh	Numeric: p-value threshold of significance.
p_adjust_method	Character: p-value adjustment method. "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none". Default = "holm". Use "none" for raw p-values.
p_transform	function.
legend	Logical: If TRUE, show legend. Will default to FALSE, if group = NULL, otherwise to TRUE.
legend_lo	Character: Legend to annotate significant points below the x_thresh.
legend_hi	Character: Legend to annotate significant points above the x_thresh.
label_lo	Character: label for low values.
label_hi	Character: label for high values.
main	Character: Main title.
xlab	Character: x-axis label.
ylab	Character: y-axis label.
margin	Named list of plot margins. Default = list(b = 65, l = 65, t = 50, r = 10, pad = 0).
xlim	Numeric vector, length 2: x-axis limits.
ylim	Numeric vector, length 2: y-axis limits.
alpha	Numeric: point transparency.
hline	Numeric: If defined, draw a horizontal line at this y value.
hline_col	Color for hline.
hline_width	Numeric: Width for hline.
hline_dash	Character: Type of line to draw: "solid", "dot", "dash", "longdash", "dashdot", or "longdashdot".
hline_annotate	Character: Text of horizontal line annotation if hline is set.

`hline_annotation_x`
: Numeric: x position to place annotation with paper as reference. 0: to the left of the plot area; 1: to the right of the plot area.

`theme`
: `Theme` object.

`annotate`
: Logical: If TRUE, annotate significant points.

`annotate_col`
: Color for annotations.

`font_size`
: Integer: Font size.

`palette`
: Character vector: Colors to use. If `group` is NULL, the first, second and third colors will be used for significant points with negative coefficients, non-significant points, and significant points with positive coefficients, respectively. If `group` is not NULL, colors will be assigned to groups, in order of appearance.

`legend_x_lo`
: Numeric: x position of `legend_lo`.

`legend_x_hi`
: Numeric: x position of `legend_hi`.

`legend_y`
: Numeric: y position for `legend_lo` and `legend_hi`.

`annotate_n`
: Integer: Number of significant points to annotate.

`ax_lo`
: Numeric: Sets the x component of the arrow tail about the arrow head for significant points below `x_thresh`.

`ay_lo`
: Numeric: Sets the y component of the arrow tail about the arrow head for significant points below `x_thresh`.

`ax_hi`
: Numeric: Sets the x component of the arrow tail about the arrow head for significant points above `x_thresh`.

`ay_hi`
: Numeric: Sets the y component of the arrow tail about the arrow head for significant points above `x_thresh`.

`annotate_alpha`
: Numeric: Transparency for annotations.

`hovertext`
: Character vector: Text to display on hover.

`displayModeBar`
: Logical: If TRUE, display plotly mode bar.

`filename`
: Character: Path to save the plot image.

`file_width`
: Numeric: Width of the saved plot image.

`file_height`
: Numeric: Height of the saved plot image.

`file_scale`
: Numeric: Scale of the saved plot image.

`verbosity`
: Integer: Verbosity level.

`...`
: Additional arguments passed to draw_scatter.

Value

`plotly` object.

Author(s)

EDG

Examples

```
set.seed(2019)
y <- rnormmat(500, 500, return_df = TRUE)
x <- data.frame(x = y[, 3] + y[, 5] - y[, 9] + y[, 15] + rnorm(500))
mod <- massGLM(x, y)
draw_volcano(summary(mod)[["Coefficient_x"]], summary(mod)[["p_value_x"]])
```

---

draw_xt *Plot timeseries data*

Description

Plot timeseries data

Usage

```
draw_xt(
  x,
  y,
  x2 = NULL,
  y2 = NULL,
  which_xy = NULL,
  which_xy2 = NULL,
  shade_bin = NULL,
  shade_interval = NULL,
  shade_col = NULL,
  shade_x = NULL,
  shade_name = "",
  shade_showlegend = FALSE,
  ynames = NULL,
  y2names = NULL,
  xlab = NULL,
  ylab = NULL,
  y2lab = NULL,
  xunits = NULL,
  yunits = NULL,
  y2units = NULL,
  yunits_col = NULL,
  y2units_col = NULL,
  zt = NULL,
  show_zt = TRUE,
  show_zt_every = NULL,
  zt_nticks = 18L,
  main = NULL,
  main_y = 1,
  main_yanchor = "bottom",
```

```
  x_nticks = 0,
  y_nticks = 0,
  show_rangeslider = NULL,
  slider_start = NULL,
  slider_end = NULL,
  theme = choose_theme(getOption("rtemis_theme")),
  palette = get_palette(getOption("rtemis_palette")),
  font_size = 16,
  yfill = "none",
  y2fill = "none",
  fill_alpha = 0.2,
  yline_width = 2,
  y2line_width = 2,
  x_showspikes = TRUE,
  spike_dash = "solid",
  spike_col = NULL,
  x_spike_thickness = -2,
  tickfont_size = 16,
  x_tickmode = "auto",
  x_tickvals = NULL,
  x_ticktext = NULL,
  x_tickangle = NULL,
  legend_x = 0,
  legend_y = 1.1,
  legend_xanchor = "left",
  legend_yanchor = "top",
  legend_orientation = "h",
  margin = list(l = 75, r = 75, b = 75, t = 75),
  x_standoff = 20L,
  y_standoff = 20L,
  y2_standoff = 20L,
  hovermode = "x",
  displayModeBar = TRUE,
  modeBar_file_format = "svg",
  scrollZoom = TRUE,
  filename = NULL,
  file_width = 960,
  file_height = 500,
  file_scale = 1
)
```

Arguments

x	Datetime vector or list of vectors.
y	Numeric vector or named list of vectors: y-axis data.
x2	Datetime vector or list of vectors, optional: must be provided if y2 does not correspond to values in x. A single x-axis will be drawn for all values in x and x2.

y2	Numeric vector, optional: If provided, a second y-axis will be added to the right side of the plot.
which_xy	Integer vector: Indices of x and y to plot. If not provided, will select up to the first two x-y traces.
which_xy2	Integer vector: Indices of x2 and y2 to plot. If not provided, will select up to the first two x2-y2 traces.
shade_bin	Integer vector {0, 1}: Time points in x to shade on the plot. For example, if there are 10 time points in x, and you want to shade time points 3 to 7, shade_bin = c(0, 0, 1, 1, 1, 1, 1, 0, 0, 0). Only set shade_bin or shade_interval, not both.
shade_interval	List of numeric vectors: Intervals to shade on the plot. Only set shade_bin or shade_interval, not both.
shade_col	Color: Color to shade intervals.
shade_x	Numeric vector: x-values to use for shading.
shade_name	Character: Name for shaded intervals.
shade_showlegend	Logical: If TRUE, show legend for shaded intervals.
ynames	Character vector, optional: Names for each vector in y.
y2names	Character vector, optional: Names for each vector in y2.
xlab	Character: x-axis label.
ylab	Character: y-axis label.
y2lab	Character: y2-axis label.
xunits	Character: x-axis units.
yunits	Character: y-axis units.
y2units	Character: y2-axis units.
yunits_col	Color for y-axis units.
y2units_col	Color for y2-axis units.
zt	Numeric vector: Zeitgeber time. If provided, will be shown on the x-axis instead of x. To be used only with a single x vector and no x2.
show_zt	Logical: If TRUE, show zt on x-axis, if zt is provided.
show_zt_every	Optional integer: Show zt every show_zt_every ticks. If NULL, will be calculated to be x_nticks +/- 1 if x_nticks is not 0, otherwise 12 +/- 1.
zt_nticks	Integer: Number of zt ticks to show. Only used if show_zt_every is NULL. The actual number of ticks shown will depend on the periodicity of zt, so that zt = 0 is always included.
main	Character: Main title.
main_y	Numeric: Y position of main title.
main_yanchor	Character: "top", "middle", "bottom".
x_nticks	Integer: Number of ticks on x-axis.
y_nticks	Integer: Number of ticks on y-axis.

show_rangeslider
: Logical: If TRUE, show a range slider.

slider_start
: Numeric: Start of range slider.

slider_end
: Numeric: End of range slider.

theme
: Theme object.

palette
: Character vector: Colors to be used to draw each vector in y and y2, in order.

font_size
: Numeric: Font size for text.

yfill
: Character: Fill type for y-axis: "none", "tozeroy", "tonexty".

y2fill
: Character: Fill type for y2-axis: "none", "tozeroy", "tonexty".

fill_alpha
: Numeric: Fill opacity for y-axis.

yline_width
: Numeric: Line width for y-axis lines.

y2line_width
: Numeric: Line width for y2-axis lines.

x_showspikes
: Logical: If TRUE, show spikes on x-axis.

spike_dash
: Character: Dash type for spikes: "solid", "dot", "dash", "longdash", "dashdot", "longdashdot".

spike_col
: Color for spikes.

x_spike_thickness
: Numeric: Thickness of spikes. -2 avoids drawing border around spikes.

tickfont_size
: Numeric: Font size for tick labels.

x_tickmode
: Character: "auto", "linear", "array".

x_tickvals
: Numeric vector: Tick positions.

x_ticktext
: Character vector: Tick labels.

x_tickangle
: Numeric: Angle of tick labels.

legend_x
: Numeric: X position of legend.

legend_y
: Numeric: Y position of legend.

legend_xanchor
: Character: "left", "center", "right".

legend_yanchor
: Character: "top", "middle", "bottom".

legend_orientation
: Character: "v" for vertical, "h" for horizontal.

margin
: Named list with 4 numeric values: "l", "r", "t", "b" for left, right, top, bottom margins.

x_standoff
: Numeric: Distance from x-axis to x-axis label.

y_standoff
: Numeric: Distance from y-axis to y-axis label.

y2_standoff
: Numeric: Distance from y2-axis to y2-axis label.

hovermode
: Character: "closest", "x", "x unified".

displayModeBar
: Logical: If TRUE, display plotly mode bar.

modeBar_file_format
: Character: "png", "svg", "jpeg", "webp", "pdf": file format for mode bar image export.

scrollZoom	Logical: If TRUE, enable zooming by scrolling.
filename	Character: Path to save the plot image.
file_width	Numeric: Width of the saved plot image.
file_height	Numeric: Height of the saved plot image.
file_scale	Numeric: Scale of the saved plot image.

Value

plotly object.

Author(s)

EDG

Examples

```
datetime <- seq(
  as.POSIXct("2020-01-01 00:00"),
  as.POSIXct("2020-01-02 00:00"),
  by = "hour"
)
df <- data.frame(
  datetime = datetime,
  value1 = rnorm(length(datetime)),
  value2 = rnorm(length(datetime))
)
draw_xt(df, x = df[, 1], y = df[, 2:3])
```

---

dt_describe *Describe data.table*

Description

Describe data.table

Usage

```
dt_describe(x, verbosity = 1L)
```

Arguments

x	data.table: Input data.table.
verbosity	Integer: If > 0, print output to console.

Value

List with three data.tables: Numeric, Categorical, and Date.

Author(s)

EDG

Examples

```
library(data.table)
origin <- as.POSIXct("2022-01-01 00:00:00", tz = "America/Los_Angeles")
x <- data.table(
  ID = paste0("ID", 1:10),
  V1 = rnorm(10),
  V2 = rnorm(10, 20, 3),
  V1_datetime = as.POSIXct(
    seq(
      1, 1e7,
      length.out = 10
    ),
    origin = origin
  ),
  V2_datetime = as.POSIXct(
    seq(
      1, 1e7,
      length.out = 10
    ),
    origin = origin
  ),
  C1 = sample(c("alpha", "beta", "gamma"), 10, TRUE),
  F1 = factor(sample(c("delta", "epsilon", "zeta"), 10, TRUE))
)
```

---

dt_inspect_types *Inspect column types*

Description

Will attempt to identify columns that should be numeric but are either character or factor by running inspect_type on each column.

Usage

```
dt_inspect_types(x, cols = NULL, verbosity = 1L)
```

Arguments

x	data.table: Input data.table.
cols	Character vector: columns to inspect.
verbosity	Integer: Verbosity level.

Value

Character vector.

Author(s)

EDG

Examples

```
library(data.table)
x <- data.table(
  id = 8001:8006,
  a = c("3", "5", "undefined", "21", "4", NA),
  b = c("mango", "banana", "tangerine", NA, "apple", "kiwi"),
  c = c(1, 2, 3, 4, 5, 6)
)
dt_inspect_types(x)
```

---

dt_keybin_reshape *Long to wide key-value reshaping*

Description

Reshape a long format `data.table` using key-value pairs with `data.table::dcast`

Usage

```
dt_keybin_reshape(
  x,
  id_name,
  key_name,
  positive = 1,
  negative = 0,
  xname = NULL,
  verbosity = 1L
)
```

Arguments

`x`	`data.table` object.
`id_name`	Character: Name of column in `x` that defines the IDs identifying individual rows.
`key_name`	Character: Name of column in `x` that holds the key.
`positive`	Numeric or Character: Used to fill id ~ key combination present in the long format input `x`.
`negative`	Numeric or Character: Used to fill id ~ key combination NOT present in the long format input `x`.
`xname`	Character: Name of `x` to be used in messages.
`verbosity`	Integer: Verbosity level.

Value

`data.table` in wide format.

Author(s)

EDG

Examples

```
library(data.table)
x <- data.table(
  ID = rep(1:3, each = 2),
  Dx = c("A", "C", "B", "C", "D", "A")
)
dt_keybin_reshape(x, id_name = "ID", key_name = "Dx")
```

---

dt_merge *Merge data.tables*

Description

Merge data.tables

Usage

```
dt_merge(
  left,
  right,
  on = NULL,
  left_on = NULL,
  right_on = NULL,
  how = "left",
  left_name = NULL,
  right_name = NULL,
  left_suffix = NULL,
  right_suffix = NULL,
  verbosity = 1L,
  ...
)
```

Arguments

`left`	data.table
`right`	data.table
`on`	Character: Name of column to join on.
`left_on`	Character: Name of column on left table.

right_on	Character: Name of column on right table.
how	Character: Type of join: "inner", "left", "right", "outer".
left_name	Character: Name of left table.
right_name	Character: Name of right table.
left_suffix	Character: If provided, add this suffix to all left column names, excluding on/left_on.
right_suffix	Character: If provided, add this suffix to all right column names, excluding on/right_on.
verbosity	Integer: Verbosity level.
...	Additional arguments to be passed to data.table::merge.

Value

Merged data.table.

Author(s)

EDG

Examples

```
library(data.table)
xleft <- data.table(ID = 1:5, Alpha = letters[1:5])
xright <- data.table(ID = c(3, 4, 5, 6), Beta = LETTERS[3:6])
xlr_inner <- dt_merge(xleft, xright, on = "ID", how = "inner")
```

---

dt_names_by_attr *List column names by attribute*

Description

List column names by attribute

Usage

```
dt_names_by_attr(x, attribute, exact = TRUE, sorted = TRUE)
```

Arguments

x	data.table: Input data.table.
attribute	Character: name of attribute.
exact	Logical: If TRUE, use exact matching.
sorted	Logical: If TRUE, sort the output.

Value

Character vector.

Author(s)

EDG

Examples

```
library(data.table)
x <- data.table(
  id = 1:5,
  sbp = rnorm(5, 120, 15),
  dbp = rnorm(5, 80, 10),
  paO2 = rnorm(5, 90, 10),
  paCO2 = rnorm(5, 40, 5)
)
setattr(x[["id"]], "source", "demographics")
setattr(x[["sbp"]], "source", "outpatient")
setattr(x[["dbp"]], "source", "outpatient")
setattr(x[["paO2"]], "source", "icu")
setattr(x[["paCO2"]], "source", "icu")

dt_names_by_attr(x, "source", "outpatient")
```

---

dt_nunique_perfeat *Number of unique values per feature*

Description

Number of unique values per feature

Usage

```
dt_nunique_perfeat(x, excludeNA = FALSE, limit = 20L, verbosity = 1L)
```

Arguments

x	data.table: Input data.table.
excludeNA	Logical: If TRUE, exclude NA values.
limit	Integer: Print up to this many features. Set to -1L to print all.
verbosity	Integer: If > 0, print output to console.

Value

Named integer vector of length `NCOL(x)` with number of unique values per column/feature, invisibly.

Author(s)

EDG

Examples

```
library(data.table)
ir <- as.data.table(iris)
dt_nunique_perfeat(ir)
```

`dt_pctmatch` *Get N and percent match of values between two columns of two data.tables*

Description

Get N and percent match of values between two columns of two data.tables

Usage

```
dt_pctmatch(x, y, on = NULL, left_on = NULL, right_on = NULL, verbosity = 1L)
```

Arguments

`x`	data.table: First input data.table.
`y`	data.table: Second input data.table.
`on`	Integer or character: column to read in `x` and `y`, if it is the same
`left_on`	Integer or character: column to read in `x`
`right_on`	Integer or character: column to read in `y`
`verbosity`	Integer: Verbosity level.

Value

list.

Author(s)

EDG

Examples

```
library(data.table)
x <- data.table(ID = 1:5, Alpha = letters[1:5])
y <- data.table(ID = c(3, 4, 5, 6), Beta = LETTERS[3:6])
dt_pctmatch(x, y, on = "ID")
```

dt_pctmissing *Get percent of missing values from every column*

Description

Get percent of missing values from every column

Usage

```
dt_pctmissing(x, verbosity = 1L)
```

Arguments

x	data.frame or data.table
verbosity	Integer: Verbosity level.

Value

list

Author(s)

EDG

Examples

```
library(data.table)
x <- data.table(a = c(1, 2, NA, 4), b = c(NA, NA, 3, 4), c = c("A", "B", "C", NA))
dt_pctmissing(x)
```

dt_set_autotypes *Set column types automatically*

Description

This function inspects a data.table and attempts to identify columns that should be numeric but have been read in as character, and fixes their type **in-place**. This can happen when one or more fields contain non-numeric characters, for example.

Usage

```
dt_set_autotypes(x, cols = NULL, verbosity = 1L)
```

Arguments

x	data.table: Input data.table. Will be modified **in-place**, if needed.
cols	Character vector: columns to work on. If not defined, will work on all columns
verbosity	Integer: Verbosity level.

Value

data.table, invisibly.

Author(s)

EDG

Examples

```
library(data.table)
x <- data.table(
  id = 8001:8006,
  a = c("3", "5", "undefined", "21", "4", NA),
  b = c("mango", "banana", "tangerine", NA, "apple", "kiwi"),
  c = c(1, 2, 3, 4, 5, 6)
)
str(x)
# ***in-place*** operation means no assignment is needed
dt_set_autotypes(x)
str(x)

# Try excluding column 'a' from autotyping
x <- data.table(
  id = 8001:8006,
  a = c("3", "5", "undefined", "21", "4", NA),
  b = c("mango", "banana", "tangerine", NA, "apple", "kiwi"),
  c = c(1, 2, 3, 4, 5, 6)
)
str(x)
# exclude column 'a' from autotyping
dt_set_autotypes(x, cols = setdiff(names(x), "a"))
str(x)
```

---

dt_set_cleanfactorlevels

Clean factor levels of data.table **in-place**

Description

Finds all factors in a data.table and cleans factor levels to include only underscore symbols

Usage

```
dt_set_cleanfactorlevels(x, prefix_digits = NA)
```

Arguments

x	data.table: Input data.table. Will be modified **in-place**.
prefix_digits	Character: If not NA, add this prefix to all factor levels that are numbers

Value

Nothing, modifies x **in-place**.

Author(s)

EDG

Examples

```
library(data.table)
x <- as.data.table(iris)
levels(x[["Species"]]) <- c("setosa:iris", "versicolor$iris", "virginica iris")
levels(x[["Species"]])
dt_set_cleanfactorlevels(x)
levels(x[["Species"]])
```

---

dt_set_clean_all *Clean column names and factor levels* **in-place**

Description

Clean column names and factor levels **in-place**

Usage

```
dt_set_clean_all(x, prefix_digits = NA)
```

Arguments

x	data.table: Input data.table. Will be modified **in-place**, if needed.
prefix_digits	Character: prefix to add to names beginning with a digit. Set to NA to skip

Value

Nothing, modifies x **in-place**.

Author(s)

EDG

Examples

```
library(data.table)
x <- as.data.table(iris)
levels(x[["Species"]]) <- c("setosa:iris", "versicolor$iris", "virginica iris")
names(x)
levels(x[["Species"]])
# ***in-place*** operation means no assignment is needed
dt_set_clean_all(x)
names(x)
levels(x[["Species"]])
```

---

dt_set_logical2factor *Convert data.table logical columns to factors*

Description

Convert data.table logical columns to factors with custom labels **in-place**

Usage

```
dt_set_logical2factor(
  x,
  cols = NULL,
  labels = c("False", "True"),
  maintain_attributes = TRUE,
  fillNA = NULL
)
```

Arguments

x	data.table: Input data.table. Will be modified **in-place**.
cols	Optional Integer or character: columns to convert. If NULL, operates on all logical columns.
labels	Character: labels for factor levels.
maintain_attributes	Logical: If TRUE, maintain column attributes.
fillNA	Optional Character: If not NULL, fill NA values with this constant.

Value

data.table, invisibly.

Author(s)

EDG

Examples

```
library(data.table)
x <- data.table(a = 1:5, b = c(TRUE, FALSE, FALSE, FALSE, TRUE))
x
dt_set_logical2factor(x)
x
z <- data.table(
  alpha = 1:5,
  beta = c(TRUE, FALSE, TRUE, NA, TRUE),
  gamma = c(FALSE, FALSE, TRUE, FALSE, NA)
)
# You can usee fillNA to fill NA values with a constant
dt_set_logical2factor(z, cols = "beta", labels = c("No", "Yes"), fillNA = "No")
z
w <- data.table(mango = 1:5, banana = c(FALSE, FALSE, TRUE, TRUE, FALSE))
w
dt_set_logical2factor(w, cols = 2, labels = c("Ugh", "Huh"))
w
# Column attributes are maintained by default:
z <- data.table(
  alpha = 1:5,
  beta = c(TRUE, FALSE, TRUE, NA, TRUE),
  gamma = c(FALSE, FALSE, TRUE, FALSE, NA)
)
for (i in seq_along(z)) setattr(z[[i]], "source", "Guava")
str(z)
dt_set_logical2factor(z, cols = "beta", labels = c("No", "Yes"))
str(z)
```

---

dt_set_one_hot *Convert data.table's factor to one-hot encoding* **in-place**

Description

Convert data.table's factor to one-hot encoding **in-place**

Usage

```
dt_set_one_hot(x, xname = NULL, verbosity = 1L)
```

Arguments

x	data.table: Input data.table. Will be modified **in-place**.
xname	Character, optional: Dataset name.
verbosity	Integer: Verbosity level.

Value

The input, invisibly, after it has been modified **in-place**.

Author(s)

EDG

Examples

```
ir <- data.table::as.data.table(iris)
# dt_set_one_hot operates ***in-place***; therefore no assignment is used:
dt_set_one_hot(ir)
ir
```

---

exc *Exclude columns by character or numeric vector.*

Description

Exclude columns by character or numeric vector.

Usage

```
exc(x, idx)
```

Arguments

x	tabular data.
idx	Character or numeric vector: Column names or indices to exclude.

Value

data.frame, tibble, or data.table.

Author(s)

EDG

Examples

```
exc(iris, "Species") |> head()
exc(iris, c(1, 3)) |> head()
```

features *Get features from tabular data*

Description

Returns all columns except the last one.

Usage

```
features(x)
```

Arguments

x	tabular data: Input data to get features from.

Details

This can be applied to tabular datasets used for supervised learning in **rtemis**, where, by convention, the last column is the outcome variable and all other columns are features.

Value

Object of the same class as the input, after removing the last column.

Author(s)

EDG

Examples

```
features(iris) |> head()
```

feature_matrix *Convert tabular data to feature matrix*

Description

Convert a tabular dataset to a matrix, one-hot encoding factors, if present.

Usage

```
feature_matrix(x)
```

Arguments

x	tabular data: Input data to convert to a feature matrix.

Details

This is a convenience function that uses `features()`, `preprocess()`, `as.matrix()`.

Value

Matrix with features. Factors are one-hot encoded, if present.

Author(s)

EDG

Examples

```
# reorder columns so that we have a categorical feature
x <- set_outcome(iris, "Sepal.Length")
feature_matrix(x) |> head()
```

---

feature_names *Get feature names*

Description

Returns all column names except the last one

Usage

```
feature_names(x)
```

Arguments

x	tabular data.

Details

This applied to tabular datasets used for supervised learning in rtemis, where, by convention, the last column is the outcome variable and all other columns are features.

Value

Character vector of feature names.

Author(s)

EDG

Examples

```
feature_names(iris)
```

getnames *Get names by string matching or class*

Description

Get names by string matching or class

Usage

```
getnames(
  x,
  pattern = NULL,
  starts_with = NULL,
  ends_with = NULL,
  ignore_case = TRUE
)

getfactornames(x)

getnumericnames(x)

getlogicalnames(x)

getcharacternames(x)

getdatenames(x)
```

Arguments

x	object with `names()` method.
pattern	Character: pattern to match anywhere in names of x.
starts_with	Character: pattern to match in the beginning of names of x.
ends_with	Character: pattern to match at the end of names of x.
ignore_case	Logical: If TRUE, well, ignore case.

Details

For `getnames()` only: `pattern`, `starts_with`, and `ends_with` are applied sequentially. If more than one is provided, the result will be the intersection of all matches.

Value

Character vector of matched names.

Author(s)

EDG

Examples

```
getnames(iris, starts_with = "Sepal")
getnames(iris, ends_with = "Width")
getfactornames(iris)
getnumericnames(iris)
```

`getnamesandtypes` *Get data.frame names and types*

Description

Get data.frame names and types

Usage

```
getnamesandtypes(x)
```

Arguments

x	data.frame / data.table or similar

Value

character vector of column names with attribute "type" holding the class of each column

Author(s)

EDG

Examples

```
getnamesandtypes(iris)
```

`get_factor_names` *Get factor names*

Description

Get factor names

Usage

```
get_factor_names(x)
```

Arguments

x	tabular data.

Details

This applied to tabular datasets used for supervised learning in rtemis, where, by convention, the last column is the outcome variable and all other columns are features.

Value

Character vector of factor names.

Author(s)

EDG

Examples

```
get_factor_names(iris)
```

---

get_mode *Get the mode of a factor or integer*

Description

Returns the mode of a factor or integer

Usage

```
get_mode(x, na.rm = TRUE, getlast = TRUE, retain_class = TRUE)
```

Arguments

x	Vector, factor or integer: Input data.
na.rm	Logical: If TRUE, exclude NAs (using na.exclude(x)).
getlast	Logical: If TRUE, get the last value in case of ties.
retain_class	Logical: If TRUE, output is always same class as input.

Value

The mode of x

Author(s)

EDG

Examples

```
x <- c(9, 3, 4, 4, 0, 2, 2, NA)
get_mode(x)
x <- c(9, 3, 2, 2, 0, 4, 4, NA)
get_mode(x)
get_mode(x, getlast = FALSE)
```

get_palette *Get Color Palette*

Description

`get_palette()` returns a color palette (character vector of colors). Without arguments, prints names of available color palettes. Each palette is a named list of hexadecimal color definitions which can be used with any graphics function.

Usage

```
get_palette(palette = NULL, verbosity = 1L)
```

Arguments

`palette`	Character: Name of palette to return. Default = NULL: available palette names are printed and no palette is returned.
`verbosity`	Integer: Verbosity level.

Value

Character vector of colors for the specified palette, or invisibly returns list of available palettes if `palette = NULL`.

Author(s)

EDG

Examples

```
# Print available palettes
get_palette()
# Get the Imperial palette
get_palette("imperial")
```

inc *Select (include) columns by character or numeric vector.*

Description

Select (include) columns by character or numeric vector.

Usage

```
inc(x, idx)
```

Arguments

x	tabular data.
idx	Character or numeric vector: Column names or indices to include.

Value

data.frame, tibble, or data.table.

Author(s)

EDG

Examples

```
inc(iris, c(3, 4)) |> head()
inc(iris, c("Sepal.Length", "Species")) |> head()
```

---

index_col_by_attr *Index columns by attribute name & value*

Description

Index columns by attribute name & value

Usage

```
index_col_by_attr(x, name, value, exact = TRUE)
```

Arguments

x	tabular data.
name	Character: Name of attribute.
value	Character: Value of attribute.
exact	Logical: Passed to `attr` when retrieving attribute value. If `TRUE`, attribute name must match `name` exactly, otherwise, partial match is allowed.

Value

Integer vector.

Author(s)

EDG

Examples

```
library(data.table)
x <- data.table(
  id = 1:5,
  sbp = rnorm(5, 120, 15),
  dbp = rnorm(5, 80, 10),
  paO2 = rnorm(5, 90, 10),
  paCO2 = rnorm(5, 40, 5)
)
setattr(x[["sbp"]], "source", "outpatient")
setattr(x[["dbp"]], "source", "outpatient")
setattr(x[["paO2"]], "source", "icu")
setattr(x[["paCO2"]], "source", "icu")
index_col_by_attr(x, "source", "icu")
```

---

init_project_dir *Initialize Project Directory*

Description

Initializes Directory Structure: "R", "Data", "Results"

Usage

```
init_project_dir(path, output_dir = "Out", verbosity = 1L)
```

Arguments

path	Character: Path to initialize project directory in.
output_dir	Character: Name of output directory to create.
verbosity	Integer: Verbosity level.

Value

Character: the path where the project directory was initialized, invisibly.

Author(s)

EDG

Examples

```
## Not run:
# Will create "my_project" directory with
init_project_dir("my_project")

## End(Not run)
```

inspect *Inspect rtemis object*

Description

Inspect rtemis object

Usage

```
inspect(x)
```

Arguments

x	R object to inspect.

Value

Called for side effect of printing information to console; returns character string invisibly.

Author(s)

EDG

Examples

```
inspect(iris)
```

inspect_type *Inspect character and factor vector*

Description

Checks character or factor vector to determine whether it might be best to convert to numeric.

Usage

```
inspect_type(x, xname = NULL, verbosity = 1L, thresh = 0.5, na.omit = TRUE)
```

Arguments

x	Character or factor vector.
xname	Character: Name of input vector x.
verbosity	Integer: Verbosity level.
thresh	Numeric: Threshold for determining whether to convert to numeric.
na.omit	Logical: If TRUE, remove NA values before checking.

Details

All data can be represented as a character string. A numeric variable may be read as a character variable if there are non-numeric characters in the data. It is important to be able to automatically detect such variables and convert them, which would mean introducing NA values.

Value

Character.

Author(s)

EDG

Examples

```
x <- c("3", "5", "undefined", "21", "4", NA)
inspect_type(x)
z <- c("mango", "banana", "tangerine", NA)
inspect_type(z)
```

---

is_constant *Check if vector is constant*

Description

Check if vector is constant

Usage

```
is_constant(x, skip_missing = FALSE)
```

Arguments

x	Vector: Input
skip_missing	Logical: If TRUE, skip NA values before test

Value

Logical.

Author(s)

EDG

Examples

```
x <- rep(9, 1000000)
is_constant(x)
x[10] <- NA
is_constant(x)
is_constant(x, skip_missing = TRUE)
```

`labelify` *Format text for label printing*

Description

Format text for label printing

Usage

```
labelify(
  x,
  underscores_to_spaces = TRUE,
  dotsToSpaces = TRUE,
  toLower = FALSE,
  toTitleCase = TRUE,
  capitalize_strings = c("id"),
  stringsToSpaces = c("\\$", "`")
)
```

Arguments

`x`	Character: Input
`underscores_to_spaces`	Logical: If TRUE, convert underscores to spaces.
`dotsToSpaces`	Logical: If TRUE, convert dots to spaces.
`toLower`	Logical: If TRUE, convert to lowercase (precedes `toTitleCase`). Default = FALSE (Good for getting all-caps words converted to title case, bad for abbreviations you want to keep all-caps)
`toTitleCase`	Logical: If TRUE, convert to Title Case. Default = TRUE (This does not change all-caps words, set `toLower` to TRUE if desired)
`capitalize_strings`	Character, vector: Always capitalize these strings, if present. Default = `"id"`
`stringsToSpaces`	Character, vector: Replace these strings with spaces. Escape as needed for `gsub`. Default = `"\\$"`, which formats common input of the type `data.frame$variable`

Value

Character vector.

Author(s)

EDG

Examples

```
x <- c("county_name", "total.cost$", "age", "weight.kg")
labelify(x)
```

---

massGLM *Mass-univariate GLM Analysis*

Description

Mass-univariate GLM Analysis

Usage

```
massGLM(x, y, scale_y = NULL, center_y = NULL, verbosity = 1L)
```

Arguments

x	tabular data: Predictor variables. Usually a small number of covariates.
y	data.frame or similar: Each column is a different outcome. The function will train one GLM for each column of y. Usually a large number of features.
scale_y	Logical: If TRUE, scale each column of y to have mean 0 and sd 1. If NULL, defaults to TRUE if y is numeric, FALSE otherwise.
center_y	Logical: If TRUE, center each column of y to have mean 0. If NULL, defaults to TRUE if scale_y is TRUE, FALSE otherwise.
verbosity	Integer: Verbosity level.

Value

MassGLM object.

Author(s)

EDG

Examples

```
set.seed(2022)
y <- rnormmat(500, 40, return_df = TRUE)
x <- data.frame(
  x1 = y[[3]] - y[[5]] + y[[14]] + rnorm(500),
  x2 = y[[21]] + rnorm(500)
)
massmod <- massGLM(x, y)
# Print table of coefficients, p-values, etc. for all models
summary(massmod)
```

matchcases *Match cases by covariates*

Description

Find one or more cases from a pool data.frame that match cases in a target data.frame. Match exactly and/or by distance (sum of squared distances).

Usage

```
matchcases(
  target,
  pool,
  n_matches = 1,
  target_id = NULL,
  pool_id = NULL,
  exactmatch_factors = TRUE,
  exactmatch_cols = NULL,
  distmatch_cols = NULL,
  norepeats = TRUE,
  ignore_na = FALSE,
  verbosity = 1L
)
```

Arguments

target	data.frame you are matching against.
pool	data.frame you are looking for matches from.
n_matches	Integer: Number of matches to return.
target_id	Character: Column name in target that holds unique cases IDs. Default = NULL, in which case integer case numbers will be used.
pool_id	Character: Same as target_id for pool.
exactmatch_factors	Logical: If TRUE, selected cases will have to exactly match factors available in target.
exactmatch_cols	Character: Names of columns that should be matched exactly.
distmatch_cols	Character: Names of columns that should be distance-matched.
norepeats	Logical: If TRUE, cases in pool can only be chosen once.
ignore_na	Logical: If TRUE, ignore NA values during exact matching.
verbosity	Integer: Verbosity level.

Value

data.frame

Author(s)

EDG

Examples

```
set.seed(2021)
cases <- data.frame(
  PID = paste0("PID", seq(4)),
  Sex = factor(c(1, 1, 0, 0)),
  Handedness = factor(c(1, 1, 0, 1)),
  Age = c(21, 27, 39, 24),
  Var = c(.7, .8, .9, .6),
  Varx = rnorm(4)
)
controls <- data.frame(
  CID = paste0("CID", seq(50)),
  Sex = factor(sample(c(0, 1), 50, TRUE)),
  Handedness = factor(sample(c(0, 1), 50, TRUE, c(.1, .9))),
  Age = sample(16:42, 50, TRUE),
  Var = rnorm(50),
  Vary = rnorm(50)
)

mc <- matchcases(cases, controls, 2, "PID", "CID")
```

---

mgetnames *Get names by string matching multiple patterns*

Description

Get names by string matching multiple patterns

Usage

```
mgetnames(
  x,
  pattern = NULL,
  starts_with = NULL,
  ends_with = NULL,
  ignore_case = TRUE,
  return_index = FALSE
)
```

Arguments

x	Character vector or object with `names()` method.
pattern	Character vector: pattern(s) to match anywhere in names of x.
starts_with	Character: pattern to match in the beginning of names of x.

ends_with	Character: pattern to match at the end of names of x.
ignore_case	Logical: If TRUE, well, ignore case.
return_index	Logical: If TRUE, return integer index of matches instead of names.

Details

pattern, starts_with, and ends_with are applied and the union of all matches is returned. pattern can be a character vector of multiple patterns to match.

Value

Character vector of matched names or integer index.

Author(s)

EDG

Examples

```
mgetnames(iris, pattern = c("Sepal", "Petal"))
mgetnames(iris, starts_with = "Sepal")
mgetnames(iris, ends_with = "Width")
```

---

names_by_class *List column names by class*

Description

List column names by class

Usage

```
names_by_class(x, sorted = TRUE, item_format = highlight, maxlength = 24)
```

Arguments

x	tabular data.
sorted	Logical: If TRUE, sort the output
item_format	Function: Function to format each item
maxlength	Integer: Maximum number of items to print

Value

NULL, invisibly.

Author(s)

EDG

Examples

```
names_by_class(iris)
```

`one_hot2factor` *Convert one-hot encoded matrix to factor*

Description

Convert one-hot encoded matrix to factor

Usage

```
one_hot2factor(x, labels = colnames(x))
```

Arguments

`x`	one-hot encoded matrix or data.frame.
`labels`	Character vector of level names.

Details

If input has a single column, it will be converted to factor and returned

Value

A factor.

Author(s)

EDG

Examples

```
x <- data.frame(matrix(FALSE, 10, 3))
colnames(x) <- c("Dx1", "Dx2", "Dx3")
x$Dx1[1:3] <- x$Dx2[4:6] <- x$Dx3[7:10] <- TRUE
one_hot2factor(x)
```

outcome *Get the outcome as a vector*

Description

Returns the last column of x, which is by convention the outcome variable.

Usage

```
outcome(x)
```

Arguments

x	tabular data.

Details

This applied to tabular datasets used for supervised learning in rtemis, where, by convention, the last column is the outcome variable and all other columns are features.

Value

Vector containing the last column of x.

Author(s)

EDG

Examples

```
outcome(iris)
```

outcome_name *Get the name of the last column*

Description

Get the name of the last column

Usage

```
outcome_name(x)
```

Arguments

x	tabular data.

Details

This applied to tabular datasets used for supervised learning in rtemis, where, by convention, the last column is the outcome variable and all other columns are features.

Value

Name of the last column.

Author(s)

EDG

Examples

```
outcome_name(iris)
```

`plot.MassGLM` *Plot MassGLM using volcano plot*

Description

Plot MassGLM using volcano plot

Usage

```
## S3 method for class 'MassGLM'
plot(
  x,
  coefname = NULL,
  p_adjust_method = "holm",
  p_transform = function(x) -log10(x),
  xlab = "Coefficient",
  ylab = NULL,
  theme = choose_theme(getOption("rtemis_theme")),
  verbosity = 1L,
  ...
)
```

Arguments

`x`	MassGLM object trained using massGLM.
`coefname`	Character: Name of coefficient to plot. If `NULL`, the first coefficient is used.
`p_adjust_method`	Character: "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none" - p-value adjustment method.
`p_transform`	Function to transform p-values for plotting. Default is `function(x) -log10(x)`.

xlab	Character: x-axis label.
ylab	Character: y-axis label.
theme	Theme object. Create using one of the theme_ functions, e.g. theme_whitegrid().
verbosity	Integer: Verbosity level.
...	Additional arguments passed to draw_volcano.

Value

plotly object with volcano plot.

Author(s)

EDG

Examples

```
set.seed(2019)
y <- rnormmat(500, 500, return_df = TRUE)
x <- data.frame(x = y[, 3] + y[, 5] - y[, 9] + y[, 15] + rnorm(500))
mod <- massGLM(x, y)
plot(mod)
```

---

plot_manhattan *Manhattan plot*

Description

Draw a Manhattan plot for MassGLM objects created with massGLM.

Usage

```
plot_manhattan(x, ...)

plot_manhattan.MassGLM(
  x,
  coefname = NULL,
 p_adjust_method = c("holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr",
    "none"),
  p_transform = function(x) -log10(x),
  ylab = NULL,
  theme = choose_theme(getOption("rtemis_theme")),
  col_pos = "#43A4AC",
  col_neg = "#FA9860",
  alpha = 0.8,
  ...
)
```

Arguments

x	MassGLM object.
...	Additional arguments passed to draw_bar.
coefname	Character: Name of coefficient to plot. If NULL, the first coefficient is used.
p_adjust_method	Character: "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none" - p-value adjustment method.
p_transform	Function to transform p-values for plotting. Default is function(x) -log10(x).
ylab	Character: y-axis label.
theme	Theme object.
col_pos	Character: Color for positive significant coefficients.
col_neg	Character: Color for negative significant coefficients.
alpha	Numeric: Transparency level for the bars.

Value

plotly object.

Author(s)

EDG

Examples

```
# x: outcome of interest as first column, optional covariates in the other columns
# y: features whose association with x we want to study
set.seed(2022)
y <- data.table(rnormmat(500, 40))
x <- data.table(
  x1 = y[[3]] - y[[5]] + y[[14]] + rnorm(500),
  x2 = y[[21]] + rnorm(500)
)
massmod <- massGLM(x, y)
plot_manhattan(massmod)
```

---

plot_roc *Plot ROC curve*

Description

This generic is used to plot the ROC curve for a model.

Usage

```
plot_roc(x, ...)
```

Arguments

x	Classification or ClassificationRes object.
...	Additional arguments passed to the plotting function.

Value

A plotly object containing the ROC curve.

Author(s)

EDG

Examples

```
ir <- iris[51:150, ]
ir[["Species"]] <- factor(ir[["Species"]])
species_glm <- train(ir, algorithm = "GLM")
plot_roc(species_glm)
```

---

plot_true_pred *Plot True vs. Predicted Values*

Description

Plot True vs. Predicted Values for Supervised objects. For classification, it plots a confusion matrix. For regression, it plots a scatter plot of true vs. predicted values.

Usage

```
plot_true_pred(x, ...)
```

Arguments

x	Supervised or SupervisedRes object.
...	Additional arguments passed to methods.

Value

plotly object.

Author(s)

EDG

Examples

```
x <- set_outcome(iris, "Sepal.Length")
sepallength_glm <- train(x, algorithm = "GLM")
plot_true_pred(sepallength_glm)
```

plot_varimp *Plot Variable Importance*

Description

Plot Variable Importance for Supervised objects.

Usage

```
plot_varimp(x, ...)
```

Arguments

x	Supervised or SupervisedRes object.
...	Additional arguments passed to methods.

Details

This method calls draw_varimp internally. If you pass an integer to the plot_top argument, the method will plot this many top features. If you pass a number between 0 and 1 to the plot_top argument, the method will plot this fraction of top features.

Value

plotly object or invisible NULL if no variable importance is available.

Author(s)

EDG

See Also

draw_varimp, which is called by this method

Examples

```
ir <- set_outcome(iris, "Sepal.Length")
seplen_cart <- train(ir, algorithm = "CART")
plot_varimp(seplen_cart)
# Plot horizontally
plot_varimp(seplen_cart, orientation = "h")
plot_varimp(seplen_cart, orientation = "h", plot_top = 3L)
plot_varimp(seplen_cart, orientation = "h", plot_top = 0.5)
```

preprocess *Preprocess Data*

Description

Preprocess data for analysis and visualization.

Usage

```
preprocess(x, config, ...)

preprocess.class_tabular.PreprocessorConfig(
  x,
  config,
  dat_validation = NULL,
  dat_test = NULL,
  verbosity = 1L
)

preprocess.class_tabular.Preprocessor(x, config, verbosity = 1L)
```

Arguments

x	data.frame, data.table, tbl_df (tabular data): Data to be preprocessed.
config	PreprocessorConfig: Setup using setup_Preprocessor OR Preprocessor object: Output of previous run of preprocess. This allows, for example, applying preprocessing to a validation or test set using the same parameters as were used for the training set. In particular, the same scale centers and coefficients will be applied to the new data.
...	Not used.
dat_validation	tabular data: Validation set data.
dat_test	tabular data: Test set data.
verbosity	Integer: Verbosity level.

Details

Methods are provided for preprocessing training set data, which accepts a PreprocessorConfig object, and for preprocessing validation and test set data, which accept a Preprocessor object.

Value

Preprocessor object.

Author(s)

EDG

Examples

```
# Setup a `Preprocessor`: this outputs a `PreprocessorConfig` object.
prp <- setup_Preprocessor(remove_duplicates = TRUE, scale = TRUE, center = TRUE)

# Includes a long list of parameters
prp

# Resample iris to get train and test data
res <- resample(iris, setup_Resampler(seed = 2026))
iris_train <- iris[res[[1]], ]
iris_test <- iris[-res[[1]], ]

# Preprocess training data
iris_pre <- preprocess(iris_train, prp)

# Access preprocessd training data with `preprocessed()`
preprocessed(iris_pre)

# Apply the same preprocessing to test data
# In this case, the scale and center values from training data will be used.
# Note how `preprocess()` accepts either a `PreprocessorConfig` or `Preprocessor` object for
# this reason.
iris_test_pre <- preprocess(iris_test, iris_pre)

# Access preprocessed test data
preprocessed(iris_test_pre)
```

---

preprocessed *Get preprocessed data from* Preprocessor.

Description

Returns the preprocessed data from a Preprocessor object.

Usage

```
preprocessed(x)
```

Arguments

x	Preprocessor: A Preprocessor object.

Value

data.frame: The preprocessed data.

Examples

```
prp <- preprocess(iris, setup_Preprocessor(scale = TRUE, center = TRUE))
preprocessed(prp)
```

present *Present rtemis object*

Description

This generic is used to present an rtemis object by printing to console and drawing plots.

Usage

```
present(x, ...)
```

Arguments

x	Supervised or SupervisedRes object or list of such objects.
...	Additional arguments passed to the plotting function.

Value

A plotly object.

Author(s)

EDG

Examples

```
ir <- set_outcome(iris, "Sepal.Length")
seplen_lightrf <- train(ir, algorithm = "lightrf")
present(seplen_lightrf)
```

present.list *Present list of Supervised or SupervisedRes objects*

Description

Plot training and testing performance boxplots of multiple Supervised or SupervisedRes objects

Usage

```
present.list(
  x,
  metric = NULL,
  model_names = NULL,
  ylim = NULL,
  theme = choose_theme(getOption("rtemis_theme")),
  boxpoints = "all",
  filename = NULL,
  file_width = 800,
  file_height = 600,
  file_scale = 1,
  verbosity = 1L
)
```

Arguments

x	List of Supervised or SupervisedRes objects.
metric	Character: Metric to plot.
model_names	Character: Names of models being plotted.
ylim	Numeric vector of length 2: y-axis limits for the boxplots.
theme	Theme object.
boxpoints	Character: "all", "outliers", or "suspectedoutliers". Determines how points are displayed in the boxplot.
filename	Character: Filename to save the plot to.
file_width	Numeric: Width of the exported image in pixels.
file_height	Numeric: Height of the exported image in pixels.
file_scale	Numeric: Scale factor for the exported image.
verbosity	Integer: Verbosity level.

Value

plotly object

Author(s)

EDG

Examples

```
## Not run:
iris_lightrf <- train(
  iris,
  algorithm = "lightrf",
  outer_resampling_config = setup_Resampler(seed = 2026)
)
```

```
iris_rsvm <- train(
  iris,
  algorithm = "radialsvm",
  outer_resampling_config = setup_Resampler(seed = 2026)
)
present(list(iris_lightrf, iris_rsvm), metric = "Balanced_Accuracy")

## End(Not run)
```

`previewcolor` *Preview color*

Description

Preview one or multiple colors using little rhombi with their little labels up top

Usage

```
previewcolor(
  x,
  main = NULL,
  bg = "#333333",
  main_col = "#b3b3b3",
  main_x = 0.7,
  main_y = 0.2,
  main_adj = 0,
  main_cex = 0.9,
  main_font = 2,
  width = NULL,
  xlim = NULL,
  ylim = c(0, 2.2),
  asp = 1,
  labels_y = 1.55,
  label_cex = NULL,
  mar = c(0, 0, 0, 1),
  filename = NULL,
  pdf_width = 8,
  pdf_height = 2.5
)
```

Arguments

`x`	Color, vector: One or more colors that R understands
`main`	Character: Title. Default = NULL, which results in `deparse(substitute(x))`
`bg`	Background color.
`main_col`	Color: Title color
`main_x`	Float: x coordinate for `main`.

main_y	Float: y coordinate for main.
main_adj	Float: adj argument to mtext for main.
main_cex	Float: character expansion factor for main.
main_font	Integer, 1 or 2: Weight of main 1: regular, 2: bold.
width	Float: Plot width. Default = NULL, i.e. set automatically
xlim	Vector, length 2: x-axis limits. Default = NULL, i.e. set automatically
ylim	Vector, length 2: y-axis limits.
asp	Float: Plot aspect ratio.
labels_y	Float: y coord for labels. Default = 1.55 (rhombi are fixed and range y .5 - 1.5)
label_cex	Float: Character expansion for labels. Default = NULL, and is calculated automatically based on length of x
mar	Numeric vector, length 4: margin size.
filename	Character: Path to save plot as PDF.
pdf_width	Numeric: Width of PDF in inches.
pdf_height	Numeric: Height of PDF in inches.

Value

Nothing, prints plot.

Author(s)

EDG

Examples

```
previewcolor(get_palette("rtms"))
```

---

read *Read tabular data from a variety of formats*

Description

Read data and optionally clean column names, keep unique rows, and convert characters to factors

Usage

```
read(
  filename,
  datadir = NULL,
  make_unique = FALSE,
  character2factor = FALSE,
  clean_colnames = TRUE,
  delim_reader = c("data.table", "vroom", "duckdb", "arrow"),
```

```
  xlsx_sheet = 1,
  sep = NULL,
  quote = "\"",
  na_strings = c(""),
  output = c("data.table", "tibble", "data.frame"),
  attr = NULL,
  value = NULL,
  verbosity = 1L,
  fread_verbosity = 0L,
  timed = verbosity > 0L,
  ...
)
```

Arguments

`filename`	Character: filename or full path if `datadir = NULL`.
`datadir`	Character: Optional path to directory where `filename` is located. If not specified, `filename` must be the full path.
`make_unique`	Logical: If TRUE, keep unique rows only.
`character2factor`	Logical: If TRUE, convert character variables to factors.
`clean_colnames`	Logical: If TRUE, clean columns names using clean_colnames.
`delim_reader`	Character: package to use for reading delimited data.
`xlsx_sheet`	Integer or character: Name or number of XLSX sheet to read.
`sep`	Single character: field separator. If `delim_reader = "fread"` and `sep = NULL`, this defaults to "auto", otherwise defaults to ",".
`quote`	Single character: quote character.
`na_strings`	Character vector: Strings to be interpreted as NA values. For `delim_reader = "duckdb"`, this must be a single string.
`output`	Character: "default" or "data.table", If default, return the delim_reader's default data structure, otherwise convert to data.table.
`attr`	Character: Attribute to set (Optional).
`value`	Character: Value to set (if `attr` is not NULL).
`verbosity`	Integer: Verbosity level.
`fread_verbosity`	Integer: Verbosity level. Passed to `data.table::fread`
`timed`	Logical: If TRUE, time the process and print to console
`...`	Additional arguments to pass to `data.table::fread`, `arrow::read_delim_arrow()`, `vroom::vroom()`, or `readxl::read_excel()`.

Details

`read` is a convenience function to read:

- **Delimited** files using `data.table:fread()`, `arrow:read_delim_arrow()`, `vroom::vroom()`, or `duckdb::duckdb_read_csv()`
- **ARFF** files using `farff::readARFF()`
- **Parquet** files using `arrow::read_parquet()`
- **XLSX** files using `readxl::read_excel()`
- **DTA** files from Stata using `haven::read_dta()`
- **FASTA** files using `seqinr::read.fasta()`
- **RDS** files using `readRDS()`

Value

data.frame, data.table, or tibble.

Author(s)

EDG

Examples

```
## Not run:
# Replace with your own data directory and filename
datadir <- "/Data"
dat <- read("iris.csv", datadir)

## End(Not run)
```

---

read_config *Read* SuperConfig *from TOML file*

Description

Read `SuperConfig` object from TOML file that was written with `write_toml()`.

Usage

```
read_config(file)
```

Arguments

`file`	Character: Path to input TOML file.

Value

`SuperConfig` object.

Author(s)

EDG

Examples

```
# Create a SuperConfig object
x <- setup_SuperConfig(
  dat_training_path = "~/Data/iris.csv",
  algorithm = "LightRF",
  hyperparameters = setup_LightRF()
)
# Write TOML file
tmpdir <- tempdir()
tmpfile <- file.path(tmpdir, "rtemis_test.toml")
write_toml(x, tmpfile)
# Read config from TOML file
x_read <- read_config(tmpfile)
```

---

regression_metrics *Regression Metrics*

Description

Regression Metrics

Usage

```
regression_metrics(true, predicted, na.rm = TRUE, sample = NULL)
```

Arguments

true	Numeric vector: True values.
predicted	Numeric vector: Predicted values.
na.rm	Logical: If TRUE, remove NA values before computation.
sample	Character: Sample name (e.g. "training", "test").

Value

RegressionMetrics object.

Author(s)

EDG

Examples

```
true <- rnorm(100)
predicted <- true + rnorm(100, sd = 0.5)
regression_metrics(true, predicted)
```

`resample` *Resample data*

Description

Create resamples of your data, e.g. for model building or validation. "KFold" creates stratified folds, , "StratSub" creates stratified subsamples, "Bootstrap" gives the standard bootstrap, i.e. random sampling with replacement, while "StratBoot" uses StratSub and then randomly duplicates some of the training cases to reach original length of input (default) or length defined by `target_length`.

Usage

```
resample(x, config = setup_Resampler(), verbosity = 1L)
```

Arguments

`x`	Vector or data.frame: Usually the outcome; `NROW(x)` defines the sample size.
`config`	Resampler object created by setup_Resampler.
`verbosity`	Integer: Verbosity level.

Details

Note that option 'KFold' may result in resamples of slightly different length. Avoid all operations which rely on equal-length vectors. For example, you can't place resamples in a data.frame, but must use a list instead.

Value

`Resampler` object.

Author(s)

EDG

Examples

```
y <- rnorm(200)
# 10-fold (stratified)
y_10fold <- resample(y, setup_Resampler(10L, "kfold"))
y_10fold
# 25 stratified subsamples
y_25strat <- resample(y, setup_Resampler(25L, "stratsub"))
y_25strat
# 100 stratified bootstraps
y_100strat <- resample(y, setup_Resampler(100L, "stratboot"))
y_100strat
# LOOCV
y_loocv <- resample(y, setup_Resampler(type = "LOOCV"))
y_loocv
```

`rnormmat` *Random Normal Matrix*

Description

Create a matrix or data frame of defined dimensions, whose columns are random normal vectors

Usage

```
rnormmat(
  nrow = 10,
  ncol = 10,
  mean = 0,
  sd = 1,
  return_df = FALSE,
  seed = NULL
)
```

Arguments

`nrow`	Integer: Number of rows.
`ncol`	Integer: Number of columns.
`mean`	Float: Mean.
`sd`	Float: Standard deviation.
`return_df`	Logical: If TRUE, return data.frame, otherwise matrix.
`seed`	Integer: Set seed for `rnorm`.

Value

`matrix` or `data.frame`.

Author(s)

EDG

Examples

```
x <- rnormmat(20, 5, mean = 12, sd = 6, return_df = TRUE, seed = 2026)
x
```

`rtemis_colors` *rtemis Color System*

Description

A named list of colors used consistently across all packages in the rtemis ecosystem.

Usage

```
rtemis_colors
```

Format

A named list with the following elements:

red "kaimana red"

blue "kaimana light blue"

green "kaimana medium green"

orange "coastside orange"

teal "rtemis teal"

purple "rtemis purple"

magenta "rtemis magenta"

highlight_col "highlight color"

object "rtemis teal"

info "lmd burgundy"

outer "kaimana red"

tuner "coastside orange"

Details

Colors are provided as hex strings.

Author(s)

EDG

Examples

```
rtemis_colors[["orange"]]

rtemis_colors[["teal"]]
```

rtversion *Get rtemis version and system info*

Description

Get rtemis version and system info

Usage

```
rtversion()
```

Value

List: rtemis version and system info, invisibly.

Author(s)

EDG

Examples

```
rtversion()
```

runifmat *Random Uniform Matrix*

Description

Create a matrix or data frame of defined dimensions, whose columns are random uniform vectors

Usage

```
runifmat(
  nrow = 10,
  ncol = 10,
  min = 0,
  max = 1,
  return_df = FALSE,
  seed = NULL
)
```

Arguments

nrow	Integer: Number of rows.
ncol	Integer: Number of columns.
min	Float: Min.
max	Float: Max.
return_df	Logical: If TRUE, return data.frame, otherwise matrix.
seed	Integer: Set seed for rnorm.

Value

matrix or data.frame.

Author(s)

EDG

Examples

```
x <- runifmat(20, 5, min = 12, max = 18, return_df = TRUE, seed = 2026)
x
```

---

setdiffsym *Symmetric Set Difference*

Description

Symmetric Set Difference

Usage

```
setdiffsym(x, y)
```

Arguments

x	vector
y	vector of same type as x

Value

Vector.

Author(s)

EDG

Examples

```
setdiff(1:10, 1:5)
setdiff(1:5, 1:10)
setdiffsym(1:10, 1:5)
setdiffsym(1:5, 1:10)
```

---

setup_CART *Setup CART Hyperparameters*

Description

Setup hyperparameters for CART training.

Usage

```
setup_CART(
  cp = 0.01,
  maxdepth = 20L,
  minsplit = 2L,
  minbucket = 1L,
  prune_cp = NULL,
  method = "auto",
  model = TRUE,
  maxcompete = 4L,
  maxsurrogate = 5L,
  usesurrogate = 2L,
  surrogatestyle = 0L,
  xval = 0L,
  cost = NULL,
  ifw = FALSE
)
```

Arguments

cp	(Tunable) Numeric: Complexity parameter.
maxdepth	(Tunable) Integer: Maximum depth of tree.
minsplit	(Tunable) Integer: Minimum number of observations in a node to split.
minbucket	(Tunable) Integer: Minimum number of observations in a terminal node.
prune_cp	(Tunable) Numeric: Complexity for cost-complexity pruning after tree is built
method	String: Splitting method.
model	Logical: If TRUE, return a model.
maxcompete	Integer: Maximum number of competitive splits.
maxsurrogate	Integer: Maximum number of surrogate splits.
usesurrogate	Integer: Number of surrogate splits to use.

surrogatestyle	Integer: Type of surrogate splits.
xval	Integer: Number of cross-validation folds.
cost	Numeric (>=0): One for each feature.
ifw	Logical: If TRUE, use Inverse Frequency Weighting in classification.

Details

Get more information from rpart::rpart and rpart::rpart.control.

Value

CARTHyperparameters object.

Author(s)

EDG

Examples

```
cart_hyperparams <- setup_CART(cp = 0.01, maxdepth = 10L, ifw = TRUE)
cart_hyperparams
```

---

setup_CMeans *Setup CMeansConfig*

Description

Setup CMeansConfig

Usage

```
setup_CMeans(
  k = 2L,
  max_iter = 100L,
  dist = c("euclidean", "manhattan"),
  method = c("cmeans", "ufcl"),
  m = 2,
  rate_par = NULL,
  weights = 1,
  control = list()
)
```

Arguments

k	Integer: Number of clusters.
max_iter	Integer: Maximum number of iterations.
dist	Character: Distance measure to use: 'euclidean' or 'manhattan'.
method	Character: "cmeans" - fuzzy c-means clustering; "ufcl": on-line update.
m	Float (>1): Degree of fuzzification.
rate_par	Float (0, 1): Learning rate for the online variant.
weights	Float (>0): Case weights.
control	List: Control config for clustering algorithm.

Value

CMeansConfig object.

Author(s)

EDG

Examples

```
cmeans_config <- setup_CMeans(k = 4L, dist = "euclidean")
cmeans_config
```

---

setup_DBSCAN *Setup DBSCANConfig*

Description

Setup DBSCANConfig

Usage

```
setup_DBSCAN(
  eps = 0.5,
  min_points = 5L,
  weights = NULL,
  border_points = TRUE,
  search = c("kdtree", "linear", "dist"),
  bucket_size = 100L,
  split_rule = c("SUGGEST", "STD", "MIDPT", "FAIR", "SL_MIDPT", "SL_FAIR"),
  approx = FALSE
)
```

Arguments

eps	Float: Radius of neighborhood.
min_points	Integer: Minimum number of points in a neighborhood to form a cluster.
weights	Numeric vector: Weights for data points.
border_points	Logical: If TRUE, assign border points to clusters.
search	Character: Nearest neighbor search strategy: "kdtree", "linear", or "dist".
bucket_size	Integer: Size of buckets for k-dtree search.
split_rule	Character: Rule for splitting clusters: "SUGGEST", "STD", "MIDPT", "FAIR", "SL_MIDPT", "SL_FAIR".
approx	Logical: If TRUE, use approximate nearest neighbor search.

Value

DBSCANConfig object.

Author(s)

EDG

Examples

```
dbscan_config <- setup_DBSCAN(eps = 0.5, min_points = 5L)
dbscan_config
```

---

setup_ExecutionConfig *Setup Execution Configuration*

Description

Setup Execution Configuration

Usage

```
setup_ExecutionConfig(
  backend = c("future", "mirai", "none"),
  n_workers = NULL,
  future_plan = NULL
)
```

Arguments

backend	Character: Execution backend: "future", "mirai", or "none".
n_workers	Integer: Number of workers for parallel execution. Only used if backend is "future" or "mirai". Do not rely on the default value, set to an appropriate number depending on your system.
future_plan	Character: Future plan to use if backend is "future".

Value

ExecutionConfig object.

Author(s)

EDG

Examples

```
setup_ExecutionConfig(backend = "future", n_workers = 4L, future_plan = "multisession")
```

---

setup_GAM *Setup GAM Hyperparameters*

Description

Setup hyperparameters for GAM training.

Usage

```
setup_GAM(k = 5L, ifw = FALSE)
```

Arguments

k	(Tunable) Integer: Number of knots.
ifw	(Tunable) Logical: If TRUE, use Inverse Frequency Weighting in classification.

Details

Get more information from mgcv::gam.

Value

GAMHyperparameters object.

Author(s)

EDG

Examples

```
gam_hyperparams <- setup_GAM(k = 5L, ifw = FALSE)
gam_hyperparams
```

setup_GLM *Setup GLM Hyperparameters*

Description

Setup hyperparameters for GLM training.

Usage

```
setup_GLM(ifw = FALSE)
```

Arguments

ifw	(Tunable) Logical: If TRUE, use Inverse Frequency Weighting in classification.

Value

GLMHyperparameters object.

Author(s)

EDG

Examples

```
glm_hyperparams <- setup_GLM(ifw = TRUE)
glm_hyperparams
```

setup_GLMNET *Setup GLMNET Hyperparameters*

Description

Setup hyperparameters for GLMNET training.

Usage

```
setup_GLMNET(
  alpha = 1,
  family = NULL,
  offset = NULL,
  which_lambda_cv = "lambda.1se",
  nlambda = 100L,
  lambda = NULL,
  penalty_factor = NULL,
  standardize = TRUE,
```

```
  intercept = TRUE,
  ifw = TRUE
)
```

Arguments

alpha	(Tunable) Numeric: Mixing parameter.
family	Character: Family for GLMNET.
offset	Numeric: Offset for GLMNET.
which_lambda_cv	Character: Which lambda to use for prediction: "lambda.1se" or "lambda.min"
nlambda	Positive integer: Number of lambda values.
lambda	Numeric: Lambda values.
penalty_factor	Numeric: Penalty factor for each feature.
standardize	Logical: If TRUE, standardize features.
intercept	Logical: If TRUE, include intercept.
ifw	Logical: If TRUE, use Inverse Frequency Weighting in classification.

Details

Get more information from glmnet::glmnet.

Value

GLMNETHyperparameters object.

Author(s)

EDG

Examples

```
glm_hyperparams <- setup_GLMNET(alpha = 1, ifw = TRUE)
glm_hyperparams
```

---

setup_GridSearch *Setup Grid Search Config*

Description

Create a `GridSearchConfig` object that can be passed to train.

Usage

```
setup_GridSearch(
  resampler_config = setup_Resampler(n_resamples = 5L, type = "KFold"),
  search_type = "exhaustive",
  randomize_p = NULL,
  metrics_aggregate_fn = "mean",
  metric = NULL,
  maximize = NULL
)
```

Arguments

resampler_config	ResamplerConfig set by setup_Resampler.
search_type	Character: "exhaustive" or "randomized". Type of grid search to use. Exhaustive search will try all combinations of config. Randomized will try a random sample of size randomize_p * N of total combinations
randomize_p	Float (0, 1): For search_type == "randomized", randomly test this proportion of combinations.
metrics_aggregate_fn	Character: Name of function to use to aggregate error metrics.
metric	Character: Metric to minimize or maximize.
maximize	Logical: If TRUE, maximize metric, otherwise minimize it.

Value

A GridSearchConfig object.

Author(s)

EDG

Examples

```
gridsearch_config <- setup_GridSearch(
  resampler_config = setup_Resampler(n_resamples = 5L, type = "KFold"),
  search_type = "exhaustive"
)
gridsearch_config
```

setup_HardCL *Setup HardCLConfig*

Description

Setup HardCLConfig

Usage

```
setup_HardCL(k = 3L, dist = c("euclidean", "manhattan"))
```

Arguments

k	Number of clusters.
dist	Character: Distance measure to use: 'euclidean' or 'manhattan'.

Value

HardCLConfig object.

Author(s)

EDG

Examples

```
hardcl_config <- setup_HardCL(k = 4L, dist = "euclidean")
hardcl_config
```

setup_ICA *setup_ICA*

Description

Setup ICA config.

Usage

```
setup_ICA(
  k = 3L,
  type = c("parallel", "deflation"),
  fun = c("logcosh", "exp"),
  alpha = 1,
  row_norm = TRUE,
  maxit = 100L,
  tol = 1e-04
)
```

Arguments

k	Integer: Number of components.
type	Character: Type of ICA: "parallel" or "deflation".
fun	Character: ICA function: "logcosh", "exp".
alpha	Numeric [1, 2]: Used in approximation to neg-entropy with fun = "logcosh".
row_norm	Logical: If TRUE, normalize rows of x before ICA.
maxit	Integer: Maximum number of iterations.
tol	Numeric: Tolerance.

Value

ICAConfig object.

Author(s)

EDG

Examples

```
ica_config <- setup_ICA(k = 3L)
ica_config
```

---

setup_Isomap *Setup Isomap config.*

Description

Setup Isomap config.

Usage

```
setup_Isomap(
  k = 2L,
  dist_method = c("euclidean", "manhattan"),
  nsd = 0L,
  path = c("shortest", "extended")
)
```

Arguments

k	Integer: Number of components.
dist_method	Character: Distance method.
nsd	Integer: Number of shortest dissimilarities retained.
path	Character: Path argument for vegan::isomap.

Value

IsomapConfig object.

Author(s)

EDG

Examples

```
isomap_config <- setup_Isomap(k = 3L)
isomap_config
```

`setup_Isotonic` *Setup Isotonic Hyperparameters*

Description

Setup hyperparameters for Isotonic Regression.

Usage

```
setup_Isotonic(ifw = FALSE)
```

Arguments

`ifw`	Logical: If TRUE, use Inverse Frequency Weighting in classification.

Details

There are not hyperparameters for this algorithm at this moment.

Value

IsotonicHyperparameters object.

Author(s)

EDG

Examples

```
isotonic_hyperparams <- setup_Isotonic(ifw = TRUE)
isotonic_hyperparams
```

setup_KMeans *Setup KMeansConfig*

Description

Setup KMeansConfig

Usage

```
setup_KMeans(k = 3L, dist = c("euclidean", "manhattan"))
```

Arguments

k	Number of clusters.
dist	Character: Distance measure to use: 'euclidean' or 'manhattan'.

Value

KMeansConfig object.

Author(s)

EDG

Examples

```
kmeans_config <- setup_KMeans(k = 4L, dist = "euclidean")
kmeans_config
```

setup_LightCART *Setup LightCART Hyperparameters*

Description

Setup hyperparameters for LightCART training.

Usage

```
setup_LightCART(
  num_leaves = 32L,
  max_depth = -1L,
  lambda_l1 = 0,
  lambda_l2 = 0,
  min_data_in_leaf = 20L,
  max_cat_threshold = 32L,
  min_data_per_group = 100L,
```

```
  linear_tree = FALSE,
  objective = NULL,
  ifw = FALSE
)
```

Arguments

num_leaves	(Tunable) Positive integer: Maximum number of leaves in one tree.
max_depth	(Tunable) Integer: Maximum depth of trees.
lambda_l1	(Tunable) Numeric: L1 regularization.
lambda_l2	(Tunable) Numeric: L2 regularization.
min_data_in_leaf	(Tunable) Positive integer: Minimum number of data in a leaf.
max_cat_threshold	(Tunable) Positive integer: Maximum number of categories for categorical features.
min_data_per_group	(Tunable) Positive integer: Minimum number of observations per categorical group.
linear_tree	(Tunable) Logical: If TRUE, use linear trees.
objective	Character: Objective function.
ifw	Logical: If TRUE, use Inverse Frequency Weighting in classification.

Details

Get more information from lightgbm::lgb.train.

Value

LightCARTHyperparameters object.

Author(s)

EDG

Examples

```
lightcart_hyperparams <- setup_LightCART(num_leaves = 32L, ifw = FALSE)
lightcart_hyperparams
```

`setup_LightGBM` *Setup LightGBM Hyperparameters*

Description

Setup hyperparameters for LightGBM training.

Usage

```
setup_LightGBM(
  max_nrounds = 1000L,
  force_nrounds = NULL,
  early_stopping_rounds = 10L,
  num_leaves = 8L,
  max_depth = -1L,
  learning_rate = 0.01,
  feature_fraction = 1,
  subsample = 1,
  subsample_freq = 1L,
  lambda_l1 = 0,
  lambda_l2 = 0,
  max_cat_threshold = 32L,
  min_data_per_group = 32L,
  linear_tree = FALSE,
  ifw = FALSE,
  objective = NULL,
  device_type = "cpu",
  tree_learner = "serial",
  force_col_wise = TRUE
)
```

Arguments

`max_nrounds`	Positive integer: Maximum number of boosting rounds.
`force_nrounds`	Positive integer: Use this many boosting rounds. Disable search for nrounds.
`early_stopping_rounds`	Positive integer: Number of rounds without improvement to stop training.
`num_leaves`	(Tunable) Positive integer: Maximum number of leaves in one tree.
`max_depth`	(Tunable) Integer: Maximum depth of trees.
`learning_rate`	(Tunable) Numeric: Learning rate.
`feature_fraction`	(Tunable) Numeric: Fraction of features to use.
`subsample`	(Tunable) Numeric: Fraction of data to use.
`subsample_freq`	(Tunable) Positive integer: Frequency of subsample.

`lambda_l1`	(Tunable) Numeric: L1 regularization.
`lambda_l2`	(Tunable) Numeric: L2 regularization.
`max_cat_threshold`	(Tunable) Positive integer: Maximum number of categories for categorical features.
`min_data_per_group`	(Tunable) Positive integer: Minimum number of observations per categorical group.
`linear_tree`	Logical: If TRUE, use linear trees.
`ifw`	Logical: If TRUE, use Inverse Frequency Weighting in classification.
`objective`	Character: Objective function.
`device_type`	Character: "cpu" or "gpu".
`tree_learner`	Character: "serial", "feature", "data", or "voting".
`force_col_wise`	Logical: Use only with CPU - If TRUE, force col-wise histogram building.

Details

Get more information from lightgbm::lgb.train.

Value

LightGBMHyperparameters object.

Author(s)

EDG

Examples

```
lightgbm_hyperparams <- setup_LightGBM(
  max_nrounds = 500L,
  learning_rate = c(0.001, 0.01, 0.05), ifw = TRUE
)
lightgbm_hyperparams
```

---

setup_LightRF *Setup LightRF Hyperparameters*

Description

Setup hyperparameters for LightRF training.

Usage

```
setup_LightRF(
  nrounds = 500L,
  num_leaves = 4096L,
  max_depth = -1L,
  feature_fraction = 0.7,
  subsample = 0.623,
  lambda_l1 = 0,
  lambda_l2 = 0,
  max_cat_threshold = 32L,
  min_data_per_group = 32L,
  linear_tree = FALSE,
  ifw = FALSE,
  objective = NULL,
  device_type = "cpu",
  tree_learner = "serial",
  force_col_wise = TRUE
)
```

Arguments

nrounds	(Tunable) Positive integer: Number of boosting rounds.
num_leaves	(Tunable) Positive integer: Maximum number of leaves in one tree.
max_depth	(Tunable) Integer: Maximum depth of trees.
feature_fraction	(Tunable) Numeric: Fraction of features to use.
subsample	(Tunable) Numeric: Fraction of data to use.
lambda_l1	(Tunable) Numeric: L1 regularization.
lambda_l2	(Tunable) Numeric: L2 regularization.
max_cat_threshold	(Tunable) Positive integer: Maximum number of categories for categorical features.
min_data_per_group	(Tunable) Positive integer: Minimum number of observations per categorical group.
linear_tree	Logical: If TRUE, use linear trees.
ifw	Logical: If TRUE, use Inverse Frequency Weighting in classification.
objective	Character: Objective function.
device_type	Character: "cpu" or "gpu".
tree_learner	Character: "serial", "feature", "data", or "voting".
force_col_wise	Logical: Use only with CPU - If TRUE, force col-wise histogram building.

Details

Get more information from lightgbm::lgb.train. Note that hyperparameters subsample_freq and early_stopping_rounds are fixed, and cannot be set because they are what makes `lightgbm` train a random forest. These can all be set when training gradient boosting with LightGBM.

Value

LightRFHyperparameters object.

Author(s)

EDG

Examples

```
lightrf_hyperparams <- setup_LightRF(nrounds = 1000L, ifw = FALSE)
lightrf_hyperparams
```

---

setup_LightRuleFit *Setup LightRuleFit Hyperparameters*

Description

Setup hyperparameters for LightRuleFit training.

Usage

```
setup_LightRuleFit(
  nrounds = 200L,
  num_leaves = 32L,
  max_depth = 4L,
  learning_rate = 0.1,
  subsample = 0.666,
  subsample_freq = 1L,
  lambda_l1 = 0,
  lambda_l2 = 0,
  objective = NULL,
  ifw_lightgbm = FALSE,
  alpha = 1,
  lambda = NULL,
  ifw_glmnet = FALSE,
  ifw = FALSE
)
```

Arguments

nrounds	(Tunable) Positive integer: Number of boosting rounds.
num_leaves	(Tunable) Positive integer: Maximum number of leaves in one tree.
max_depth	(Tunable) Integer: Maximum depth of trees.
learning_rate	(Tunable) Numeric: Learning rate.
subsample	(Tunable) Numeric: Fraction of data to use.
subsample_freq	(Tunable) Positive integer: Frequency of subsample.

lambda_l1	(Tunable) Numeric: L1 regularization.
lambda_l2	(Tunable) Numeric: L2 regularization.
objective	Character: Objective function.
ifw_lightgbm	(Tunable) Logical: If TRUE, use Inverse Frequency Weighting in the LightGBM step.
alpha	(Tunable) Numeric: Alpha for GLMNET.
lambda	Numeric: Lambda for GLMNET.
ifw_glmnet	(Tunable) Logical: If TRUE, use Inverse Frequency Weighting in the GLMNET step.
ifw	Logical: If TRUE, use Inverse Frequency Weighting in classification. This applies IFW to both LightGBM and GLMNET.

Details

Get more information from lightgbm::lgb.train.

Value

LightRuleFitHyperparameters object.

Author(s)

EDG

Examples

```
lightrulefit_hyperparams <- setup_LightRuleFit(nrounds = 300L, max_depth = 3L)
lightrulefit_hyperparams
```

---

setup_LinearSVM *Setup LinearSVM Hyperparameters*

Description

Setup hyperparameters for LinearSVM training.

Usage

```
setup_LinearSVM(cost = 1, ifw = FALSE)
```

Arguments

cost	(Tunable) Numeric: Cost of constraints violation.
ifw	Logical: If TRUE, use Inverse Frequency Weighting in classification.

Details

Get more information from e1071::svm.

Value

LinearSVMHyperparameters object.

Author(s)

EDG

Examples

```
linear_svm_hyperparams <- setup_LinearSVM(cost = 0.5, ifw = TRUE)
linear_svm_hyperparams
```

---

setup_NeuralGas *Setup NeuralGasConfig*

Description

Setup NeuralGasConfig

Usage

```
setup_NeuralGas(k = 3L, dist = c("euclidean", "manhattan"))
```

Arguments

k	Number of clusters.
dist	Character: Distance measure to use: 'euclidean' or 'manhattan'.

Value

NeuralGasConfig object.

Author(s)

EDG

Examples

```
neuralgas_config <- setup_NeuralGas(k = 4L, dist = "euclidean")
neuralgas_config
```

setup_NMF *Setup NMF config.*

Description

Setup NMF config.

Usage

```
setup_NMF(k = 2L, method = "brunet", nrun = if (length(k) > 1L) 30L else 1L)
```

Arguments

k	Integer: Number of components.
method	Character: NMF method. See NMF::nmf.
nrun	Integer: Number of runs to perform.

Value

NMFConfig object.

Author(s)

EDG

Examples

```
nmf_config <- setup_NMF(k = 3L)
nmf_config
```

setup_PCA *Setup PCA config.*

Description

Setup PCA config.

Usage

```
setup_PCA(k = 3L, center = TRUE, scale = TRUE, tol = NULL)
```

Arguments

k	Integer: Number of components. (passed to prcomp rank.)
center	Logical: If TRUE, center the data.
scale	Logical: If TRUE, scale the data.
tol	Numeric: Tolerance.

Value

PCAConfig object.

Author(s)

EDG

Examples

```
pca_config <- setup_PCA(k = 3L)
pca_config
```

`setup_Preprocessor` *Setup Preprocessor*

Description

Creates a `PreprocessorConfig` object, which can be used in preprocess.

Usage

```
setup_Preprocessor(
  complete_cases = FALSE,
  remove_features_thres = NULL,
  remove_cases_thres = NULL,
  missingness = FALSE,
  impute = FALSE,
  impute_type = c("missRanger", "micePMM", "meanMode"),
  impute_missRanger_params = list(pmm.k = 3, maxiter = 10, num.trees = 500),
  impute_discrete = "get_mode",
  impute_continuous = "mean",
  integer2factor = FALSE,
  integer2numeric = FALSE,
  logical2factor = FALSE,
  logical2numeric = FALSE,
  numeric2factor = FALSE,
  numeric2factor_levels = NULL,
  numeric_cut_n = 0,
  numeric_cut_labels = FALSE,
  numeric_quant_n = 0,
  numeric_quant_NAonly = FALSE,
  unique_len2factor = 0,
  character2factor = FALSE,
  factorNA2missing = FALSE,
  factorNA2missing_level = "missing",
  factor2integer = FALSE,
  factor2integer_startat0 = TRUE,
```

```
  scale = FALSE,
  center = scale,
  scale_centers = NULL,
  scale_coefficients = NULL,
  remove_constants = FALSE,
  remove_constants_skip_missing = TRUE,
  remove_features = NULL,
  remove_duplicates = FALSE,
  one_hot = FALSE,
  one_hot_levels = NULL,
  add_date_features = FALSE,
  date_features = c("weekday", "month", "year"),
  add_holidays = FALSE,
  exclude = NULL
)
```

Arguments

`complete_cases`	Logical: If TRUE, only retain complete cases (no missing data).
`remove_features_thres`	Float (0, 1): Remove features with missing values in >= to this fraction of cases.
`remove_cases_thres`	Float (0, 1): Remove cases with >= to this fraction of missing features.
`missingness`	Logical: If TRUE, generate new boolean columns for each feature with missing values, indicating which cases were missing data.
`impute`	Logical: If TRUE, impute missing cases. See `impute_discrete` and `impute_continuous`.
`impute_type`	Character: Package to use for imputation.
`impute_missRanger_params`	Named list with elements "pmm.k" and "maxiter", which are passed to `missRanger::missRanger`. `pmm.k` greater than 0 results in predictive mean matching. Default `pmm.k = 3` `maxiter = 10` `num.trees = 500`. Reduce `num.trees` for faster imputation especially in large datasets. Set `pmm.k = 0` to disable predictive mean matching.
`impute_discrete`	Character: Name of function that returns single value: How to impute discrete variables for `impute_type = "meanMode"`.
`impute_continuous`	Character: Name of function that returns single value: How to impute continuous variables for `impute_type = "meanMode"`.
`integer2factor`	Logical: If TRUE, convert all integers to factors. This includes `bit64::integer64` columns.
`integer2numeric`	Logical: If TRUE, convert all integers to numeric (will only work if `integer2factor = FALSE`). This includes `bit64::integer64` columns.
`logical2factor`	Logical: If TRUE, convert all logical variables to factors.
`logical2numeric`	Logical: If TRUE, convert all logical variables to numeric.

`numeric2factor` Logical: If TRUE, convert all numeric variables to factors.

`numeric2factor_levels`
: Character vector: Optional - will be passed to `levels` arg of `factor()` if `numeric2factor = TRUE`. For advanced/ specific use cases; need to know unique values of numeric vector(s) and given all numeric vars have same unique values.

`numeric_cut_n` Integer: If > 0, convert all numeric variables to factors by binning using `base::cut` with `breaks` equal to this number.

`numeric_cut_labels`
: Logical: The `labels` argument of base::cut.

`numeric_quant_n`
: Integer: If > 0, convert all numeric variables to factors by binning using `base::cut` with `breaks` equal to this number of quantiles. produced using `stats::quantile`.

`numeric_quant_NAonly`
: Logical: If TRUE, only bin numeric variables with missing values.

`unique_len2factor`
: Integer (>=2): Convert all variables with less than or equal to this number of unique values to factors. For example, if binary variables are encoded with 1, 2, you could use `unique_len2factor = 2` to convert them to factors.

`character2factor`
: Logical: If TRUE, convert all character variables to factors.

`factorNA2missing`
: Logical: If TRUE, make NA values in factors be of level `factorNA2missing_level`. In many cases this is the preferred way to handle missing data in categorical variables. Note that since this step is performed before imputation, you can use this option to handle missing data in categorical variables and impute numeric variables in the same `preprocess` call.

`factorNA2missing_level`
: Character: Name of level if `factorNA2missing = TRUE`.

`factor2integer` Logical: If TRUE, convert all factors to integers.

`factor2integer_startat0`
: Logical: If TRUE, start integer coding at 0.

`scale` Logical: If TRUE, scale columns of `x`.

`center` Logical: If TRUE, center columns of `x`. Note that by default it is the same as `scale`.

`scale_centers` Named vector: Centering values for each feature.

`scale_coefficients`
: Named vector: Scaling values for each feature.

`remove_constants`
: Logical: If TRUE, remove constant columns.

`remove_constants_skip_missing`
: Logical: If TRUE, skip missing values, before checking if feature is constant.

`remove_features`
: Character vector: Features to remove.

`remove_duplicates`
: Logical: If TRUE, remove duplicate cases.

one_hot	Logical: If TRUE, convert all factors using one-hot encoding.
one_hot_levels	List: Named list of the form "feature_name" = "levels". Used when applying one-hot encoding to validation or test data using Preprocessor.
add_date_features	Logical: If TRUE, extract date features from date columns.
date_features	Character vector: Features to extract from dates.
add_holidays	Logical: If TRUE, extract holidays from date columns.
exclude	Integer, vector: Exclude these columns from preprocessing.

Value

PreprocessorConfig object.

Order of Operations

- keep complete cases only
- remove constants
- remove duplicates
- remove cases by missingness threshold
- remove features by missingness threshold
- integer to factor
- integer to numeric
- logical to factor
- logical to numeric
- numeric to factor
- cut numeric to n bins
- cut numeric to n quantiles
- numeric with less than N unique values to factor
- character to factor
- factor NA to named level
- add missingness column
- impute
- scale and/or center
- one-hot encoding

Author(s)

EDG

Examples

```
preproc_config <- setup_Preprocessor(factorNA2missing = TRUE)
preproc_config
```

setup_RadialSVM *Setup RadialSVM Hyperparameters*

Description

Setup hyperparameters for RadialSVM training.

Usage

```
setup_RadialSVM(cost = 1, gamma = 0.01, ifw = FALSE)
```

Arguments

cost	(Tunable) Numeric: Cost of constraints violation.
gamma	(Tunable) Numeric: Kernel coefficient.
ifw	Logical: If TRUE, use Inverse Frequency Weighting in classification.

Details

Get more information from e1071::svm.

Value

RadialSVMHyperparameters object.

Author(s)

EDG

Examples

```
radial_svm_hyperparams <- setup_RadialSVM(cost = 10, gamma = 0.1, ifw = TRUE)
radial_svm_hyperparams
```

setup_Ranger *Setup Ranger Hyperparameters*

Description

Setup hyperparameters for Ranger Random Forest training.

Usage

```
setup_Ranger(
  num_trees = 500,
  mtry = NULL,
  importance = "impurity",
  write_forest = TRUE,
  probability = FALSE,
  min_node_size = NULL,
  min_bucket = NULL,
  max_depth = NULL,
  replace = TRUE,
  sample_fraction = ifelse(replace, 1, 0.632),
  case_weights = NULL,
  class_weights = NULL,
  splitrule = NULL,
  num_random_splits = 1,
  alpha = 0.5,
  minprop = 0.1,
  poisson_tau = 1,
  split_select_weights = NULL,
  always_split_variables = NULL,
  respect_unordered_factors = NULL,
  scale_permutation_importance = FALSE,
  local_importance = FALSE,
  regularization_factor = 1,
  regularization_usedepth = FALSE,
  keep_inbag = FALSE,
  inbag = NULL,
  holdout = FALSE,
  quantreg = FALSE,
  time_interest = NULL,
  oob_error = TRUE,
  save_memory = FALSE,
  verbose = TRUE,
  node_stats = FALSE,
  seed = NULL,
  na_action = "na.learn",
  ifw = FALSE
)
```

Arguments

num_trees	(Tunable) Positive integer: Number of trees.
mtry	(Tunable) Positive integer: Number of features to consider at each split.
importance	Character: Variable importance mode. "none", "impurity", "impurity_corrected", "permutation". The "impurity" measure is the Gini index for classification, the variance of the responses for regression.

`write_forest` Logical: Save ranger.forest object, required for prediction. Set to FALSE to reduce memory usage if no prediction intended.

`probability` Logical: Grow a probability forest as in Malley et al. (2012). For classification only.

`min_node_size` (Tunable) Positive integer: Minimal node size. Default 1 for classification, 5 for regression, 3 for survival, and 10 for probability.

`min_bucket` Positive integer: Minimal number of samples in a terminal node. Only for survival. Deprecated in favor of min_node_size.

`max_depth` (Tunable) Positive integer: Maximal tree depth. A value of NULL or 0 (the default) corresponds to unlimited depth, 1 to tree stumps (1 split per tree).

`replace` Logical: Sample with replacement.

`sample_fraction`
(Tunable) Numeric: Fraction of observations to sample. Default is 1 for sampling with replacement and 0.632 for sampling without replacement.

`case_weights` Numeric vector: Weights for sampling of training observations. Observations with larger weights will be selected with higher probability in the bootstrap (or subsampled) samples for the trees.

`class_weights` Numeric vector: Weights for the outcome classes for classification. Vector of the same length as the number of classes, with names corresponding to the class labels.

`splitrule` (Tunable) Character: Splitting rule. For classification: "gini", "extratrees", "hellinger". For regression: "variance", "extratrees", "maxstat", "beta". For survival: "logrank", "extratrees", "C", "maxstat".

`num_random_splits`
(Tunable) Positive integer: For "extratrees" splitrule: Number of random splits to consider for each candidate splitting variable.

`alpha` (Tunable) Numeric: For "maxstat" splitrule: significance threshold to allow splitting.

`minprop` (Tunable) Numeric: For "maxstat" splitrule: lower quantile of covariate distribution to be considered for splitting.

`poisson_tau` Numeric: For "poisson" regression splitrule: tau parameter for Poisson regression.

`split_select_weights`
Numeric vector: Numeric vector with weights between 0 and 1, representing the probability to select variables for splitting. Alternatively, a list of size num_trees, with one weight vector per tree.

`always_split_variables`
Character vector: Character vector with variable names to be always selected in addition to the mtry variables tried for splitting.

`respect_unordered_factors`
Character or logical: Handling of unordered factor covariates. For "partition" all 2^(k-1)-1 possible partitions are considered for splitting, where k is the number of factor levels. For "ignore", all factor levels are ordered by their first occurrence in the data. For "order", all factor levels are ordered by their average response. TRUE corresponds to "partition" for the randomForest package compatibility.

`scale_permutation_importance`
: Logical: Scale permutation importance by standard error as in (Breiman 2001). Only applicable if permutation variable importance mode selected.

`local_importance`
: Logical: For permutation variable importance, use local importance as in Breiman (2001) and Liaw & Wiener (2002).

`regularization_factor`
: (Tunable) Numeric: Regularization factor. Penalize variables with many split points. Requires splitrule = "variance".

`regularization_usedepth`
: Logical: Use regularization factor with node depth. Requires regularization_factor.

`keep_inbag`
: Logical: Save how often observations are in-bag in each tree. These will be used for (local) variable importance if inbag.counts in predict() is NULL.

`inbag`
: List: Manually set observations per tree. List of size num_trees, containing inbag counts for each observation. Can be used for stratified sampling.

`holdout`
: Logical: Hold-out mode. Hold-out all samples with case weight 0 and use these for variable importance and prediction error.

`quantreg`
: Logical: Prepare quantile prediction as in quantile regression forests (Meinshausen 2006). For regression only. Set keep_inbag = TRUE to prepare out-of-bag quantile prediction.

`time_interest`
: Numeric: For GWAS data: SNP with this number will be used as time variable. Only for survival. Deprecated, use time.var in formula instead.

`oob_error`
: Logical: Compute OOB prediction error. Set to FALSE to save computation time if only the forest is needed.

`save_memory`
: Logical: Use memory saving (but slower) splitting mode. No effect for survival and GWAS data. Warning: This option slows down the tree growing, use only if you encounter memory problems.

`verbose`
: Logical: Show computation status and estimated runtime.

`node_stats`
: Logical: Save additional node statistics. Only terminal nodes for now.

`seed`
: Positive integer: Random seed. Default is NULL, which generates the seed from R. Set to 0 to ignore the R seed.

`na_action`
: Character: Action to take if the data contains missing values. "na.learn" uses observations with missing values in splitting, treating missing values as a separate category.

`ifw`
: Logical: Inverse Frequency Weighting for classification. If TRUE, class weights are set inversely proportional to the class frequencies.

Details

Get more information from ranger::ranger.

Value

RangerHyperparameters object.

Author(s)

EDG

Examples

```
ranger_hyperparams <- setup_Ranger(num_trees = 1000L, ifw = FALSE)
ranger_hyperparams
```

---

setup_Resampler *Setup Resampler*

Description

Setup Resampler

Usage

```
setup_Resampler(
  n_resamples = 10L,
  type = c("KFold", "StratSub", "StratBoot", "Bootstrap", "LOOCV"),
  stratify_var = NULL,
  train_p = 0.75,
  strat_n_bins = 4L,
  target_length = NULL,
  id_strat = NULL,
  seed = NULL,
  verbosity = 1L
)
```

Arguments

n_resamples	Integer: Number of resamples to make.
type	Character: Type of resampler: "KFold", "StratSub", "StratBoot", "Bootstrap", "LOOCV"
stratify_var	Character: Variable to stratify by.
train_p	Float: Training set percentage.
strat_n_bins	Integer: Number of bins to stratify by.
target_length	Integer: Target length for stratified bootstraps.
id_strat	Integer: Vector of indices to stratify by. These may be, for example, case IDs if your dataset contains repeated measurements. By specifying this vector, you can ensure that each case can only be present in the training or test set, but not both.
seed	Integer: Random seed.
verbosity	Integer: Verbosity level.

Value

ResamplerConfig object.

Author(s)

EDG

Examples

```
tenfold_resampler <- setup_Resampler(n_resamples = 10L, type = "KFold", seed = 2026L)
tenfold_resampler
```

`setup_SuperConfig` *Setup SuperConfig*

Description

Setup `SuperConfig` object.

Usage

```
setup_SuperConfig(
  dat_training_path,
  dat_validation_path = NULL,
  dat_test_path = NULL,
  weights = NULL,
  preprocessor_config = NULL,
  algorithm = NULL,
  hyperparameters = NULL,
  tuner_config = NULL,
  outer_resampling_config = NULL,
  execution_config = setup_ExecutionConfig(),
  question = NULL,
  outdir = "results/",
  verbosity = 1L
)
```

Arguments

`dat_training_path`	Character: Path to training data file.
`dat_validation_path`	Character: Path to validation data file.
`dat_test_path`	Character: Path to test data file.
`weights`	Optional Character: Column name in training data to use as observation weights. If NULL, no weights are used.

preprocessor_config
: PreprocessorConfig object: Configuration for data preprocessing.

algorithm
: Character: Algorithm to use for training.

hyperparameters
: Hyperparameters object: Configuration for model hyperparameters.

tuner_config
: TunerConfig object: Configuration for hyperparameter tuning.

outer_resampling_config
: ResamplerConfig object: Configuration for outer res resampling during model training.

execution_config
: ExecutionConfig object: Configuration for execution settings. Setup with setup_ExecutionConfig.

question
: Character: Question to answer with the supervised learning analysis.

outdir
: Character: Output directory for results.

verbosity
: Integer: Verbosity level.

Value

SuperConfig object.

Author(s)

EDG

Examples

```
sc <- setup_SuperConfig(
  dat_training_path = "train.csv",
  preprocessor_config = setup_Preprocessor(remove_duplicates = TRUE),
  algorithm = "LightRF",
  hyperparameters = setup_LightRF(),
  tuner_config = setup_GridSearch(),
  outer_resampling_config = setup_Resampler(),
  execution_config = setup_ExecutionConfig(),
  question = "Can we tell iris species apart given their measurements?",
  outdir = "models/"
)
```

---

setup_TabNet *Setup TabNet Hyperparameters*

Description

Setup hyperparameters for TabNet training.

Usage

```
setup_TabNet(
  batch_size = 1024^2,
  penalty = 0.001,
  clip_value = NULL,
  loss = "auto",
  epochs = 50L,
  drop_last = FALSE,
  decision_width = NULL,
  attention_width = NULL,
  num_steps = 3L,
  feature_reusage = 1.3,
  mask_type = "sparsemax",
  virtual_batch_size = 256^2,
  valid_split = 0,
  learn_rate = 0.02,
  optimizer = "adam",
  lr_scheduler = NULL,
  lr_decay = 0.1,
  step_size = 30,
  checkpoint_epochs = 10L,
  cat_emb_dim = 1L,
  num_independent = 2L,
  num_shared = 2L,
  num_independent_decoder = 1L,
  num_shared_decoder = 1L,
  momentum = 0.02,
  pretraining_ratio = 0.5,
  device = "auto",
  importance_sample_size = NULL,
  early_stopping_monitor = "auto",
  early_stopping_tolerance = 0,
  early_stopping_patience = 0,
  num_workers = 0L,
  skip_importance = FALSE,
  ifw = FALSE
)
```

Arguments

batch_size	(Tunable) Positive integer: Batch size.
penalty	(Tunable) Numeric: Regularization penalty.
clip_value	Numeric: Clip value.
loss	Character: Loss function.
epochs	(Tunable) Positive integer: Number of epochs.
drop_last	Logical: If TRUE, drop last batch.
decision_width	(Tunable) Positive integer: Decision width.

`attention_width`
: (Tunable) Positive integer: Attention width.

`num_steps`
: (Tunable) Positive integer: Number of steps.

`feature_reusage`
: (Tunable) Numeric: Feature reusage.

`mask_type`
: Character: Mask type.

`virtual_batch_size`
: (Tunable) Positive integer: Virtual batch size.

`valid_split`
: Numeric: Validation split.

`learn_rate`
: (Tunable) Numeric: Learning rate.

`optimizer`
: Character or torch function: Optimizer.

`lr_scheduler`
: Character or torch function: "step", "reduce_on_plateau".

`lr_decay`
: Numeric: Learning rate decay.

`step_size`
: Positive integer: Step size.

`checkpoint_epochs`
: (Tunable) Positive integer: Checkpoint epochs.

`cat_emb_dim`
: (Tunable) Positive integer: Categorical embedding dimension.

`num_independent`
: (Tunable) Positive integer: Number of independent Gated Linear Units (GLU) at each step of the encoder.

`num_shared`
: (Tunable) Positive integer: Number of shared Gated Linear Units (GLU) at each step of the encoder.

`num_independent_decoder`
: (Tunable) Positive integer: Number of independent GLU layers for pretraining.

`num_shared_decoder`
: (Tunable) Positive integer: Number of shared GLU layers for pretraining.

`momentum`
: (Tunable) Numeric: Momentum.

`pretraining_ratio`
: (Tunable) Numeric: Pretraining ratio.

`device`
: Character: Device "cpu" or "cuda".

`importance_sample_size`
: Positive integer: Importance sample size.

`early_stopping_monitor`
: Character: Early stopping monitor. "valid_loss", "train_loss", "auto".

`early_stopping_tolerance`
: Numeric: Minimum relative improvement to reset the patience counter.

`early_stopping_patience`
: Positive integer: Number of epochs without improving before stopping.

`num_workers`
: Positive integer: Number of subprocesses for data loacding.

`skip_importance`
: Logical: If TRUE, skip importance calculation.

`ifw`
: Logical: If TRUE, use Inverse Frequency Weighting in classification.

Value

TabNetHyperparameters object.

Author(s)

EDG

Examples

```
tabnet_hyperparams <- setup_TabNet(epochs = 100L, learn_rate = 0.01)
tabnet_hyperparams
```

`setup_tSNE` *Setup tSNE config.*

Description

Setup tSNE config.

Usage

```
setup_tSNE(
  k = 2L,
  initial_dims = 50L,
  perplexity = 30,
  theta = 0.5,
  check_duplicates = TRUE,
  pca = TRUE,
  partial_pca = FALSE,
  max_iter = 1000L,
  verbose = getOption("verbose", FALSE),
  is_distance = FALSE,
  Y_init = NULL,
  pca_center = TRUE,
  pca_scale = FALSE,
  normalize = TRUE,
  stop_lying_iter = ifelse(is.null(Y_init), 250L, 0L),
  mom_switch_iter = ifelse(is.null(Y_init), 250L, 0L),
  momentum = 0.5,
  final_momentum = 0.8,
  eta = 200,
  exaggeration_factor = 12,
  num_threads = 1L
)
```

Arguments

k	Integer: Number of components.
initial_dims	Integer: Initial dimensions.
perplexity	Integer: Perplexity.
theta	Float: Theta.
check_duplicates	Logical: If TRUE, check for duplicates.
pca	Logical: If TRUE, perform PCA.
partial_pca	Logical: If TRUE, perform partial PCA.
max_iter	Integer: Maximum number of iterations.
verbose	Logical: If TRUE, print messages.
is_distance	Logical: If TRUE, x is a distance matrix.
Y_init	Matrix: Initial Y matrix.
pca_center	Logical: If TRUE, center PCA.
pca_scale	Logical: If TRUE, scale PCA.
normalize	Logical: If TRUE, normalize.
stop_lying_iter	Integer: Stop lying iterations.
mom_switch_iter	Integer: Momentum switch iterations.
momentum	Float: Momentum.
final_momentum	Float: Final momentum.
eta	Float: Eta.
exaggeration_factor	Float: Exaggeration factor.
num_threads	Integer: Number of threads.

Details

Get more information on the config by running `?Rtsne::Rtsne`.

Value

tSNEConfig object.

Author(s)

EDG

Examples

```
tSNE_config <- setup_tSNE(k = 3L)
tSNE_config
```

setup_UMAP *Setup UMAP config.*

Description

Setup UMAP config.

Usage

```
setup_UMAP(
  k = 2L,
  n_neighbors = 15L,
  init = "spectral",
  metric = c("euclidean", "cosine", "manhattan", "hamming", "categorical"),
  n_epochs = NULL,
  learning_rate = 1,
  scale = TRUE
)
```

Arguments

k	Integer: Number of components.
n_neighbors	Integer: Number of keighbors.
init	Character: Initialization type. See uwot::umap "init".
metric	Character: Distance metric to use: "euclidean", "cosine", "manhattan", "hamming", "categorical".
n_epochs	Integer: Number of epochs.
learning_rate	Float: Learning rate.
scale	Logical: If TRUE, scale input data before doing UMAP.

Details

A high n_neighbors value may give error in some systems: "Error in irlba::irlba(L, nv = n, nu = 0, maxit = iters) : function 'as_cholmod_sparse' not provided by package 'Matrix'"

Value

UMAPConfig object.

Author(s)

EDG

Examples

```
umap_config <- setup_UMAP(k = 3L)
umap_config
```

set_outcome *Move outcome to last column*

Description

Move outcome to last column

Usage

```
set_outcome(dat, outcome_column)
```

Arguments

dat	data.frame or similar.
outcome_column	Character: Name of outcome column.

Value

object of same class as data

Author(s)

EDG

Examples

```
ir <- set_outcome(iris, "Sepal.Length")
head(ir)
```

size *Size of object*

Description

Returns the size of an object

Usage

```
size(x, verbosity = 1L)
```

Arguments

x	any object with length() or dim().
verbosity	Integer: Verbosity level. If > 0, print size to console

Details

If `dim(x)` is NULL, returns `length(x)`.

Value

Integer vector with length equal to the number of dimensions of `x`, invisibly.

Author(s)

EDG

Examples

```
x <- rnorm(20)
size(x)
# 20
x <- matrix(rnorm(100), 20, 5)
size(x)
# 20  5
```

`table_column_attr` *Tabulate column attributes*

Description

Tabulate column attributes

Usage

```
table_column_attr(x, attr = "source", useNA = "always")
```

Arguments

`x`	tabular data: Input data set.
`attr`	Character: Attribute to get
`useNA`	Character: Passed to `table`

Value

table.

Author(s)

EDG

Examples

```
library(data.table)
x <- data.table(
  id = 1:5,
  sbp = rnorm(5, 120, 15),
  dbp = rnorm(5, 80, 10),
  paO2 = rnorm(5, 90, 10),
  paCO2 = rnorm(5, 40, 5)
)
setattr(x[["sbp"]], "source", "outpatient")
setattr(x[["dbp"]], "source", "outpatient")
setattr(x[["paO2"]], "source", "icu")
setattr(x[["paCO2"]], "source", "icu")
table_column_attr(x, "source")
```

---

theme_black *Themes for* draw_* *functions*

Description

Themes for draw_* functions

Usage

```
theme_black(
  bg = "#000000",
  plot_bg = "transparent",
  fg = "#ffffff",
  pch = 16,
  cex = 1,
  lwd = 2,
  bty = "n",
  box_col = fg,
  box_alpha = 1,
  box_lty = 1,
  box_lwd = 0.5,
  grid = FALSE,
  grid_nx = NULL,
  grid_ny = NULL,
  grid_col = fg,
  grid_alpha = 0.2,
  grid_lty = 1,
  grid_lwd = 1,
  axes_visible = TRUE,
  axes_col = "transparent",
  tick_col = fg,
  tick_alpha = 0.5,
```

```
  tick_labels_col = fg,
  tck = -0.01,
  tcl = NA,
  x_axis_side = 1,
  y_axis_side = 2,
  labs_col = fg,
  x_axis_line = 0,
  x_axis_las = 0,
  x_axis_padj = -1.1,
  x_axis_hadj = 0.5,
  y_axis_line = 0,
  y_axis_las = 1,
  y_axis_padj = 0.5,
  y_axis_hadj = 0.5,
  xlab_line = 1.4,
  ylab_line = 2,
  zerolines = TRUE,
  zerolines_col = fg,
  zerolines_alpha = 0.5,
  zerolines_lty = 1,
  zerolines_lwd = 1,
  main_line = 0.25,
  main_adj = 0,
  main_font = 2,
  main_col = fg,
  font_family = getOption("rtemis_font", "Helvetica")
)

theme_blackgrid(
  bg = "#000000",
  plot_bg = "transparent",
  fg = "#ffffff",
  pch = 16,
  cex = 1,
  lwd = 2,
  bty = "n",
  box_col = fg,
  box_alpha = 1,
  box_lty = 1,
  box_lwd = 0.5,
  grid = TRUE,
  grid_nx = NULL,
  grid_ny = NULL,
  grid_col = fg,
  grid_alpha = 0.2,
  grid_lty = 1,
  grid_lwd = 1,
  axes_visible = TRUE,
```

```
    axes_col = "transparent",
    tick_col = fg,
    tick_alpha = 1,
    tick_labels_col = fg,
    tck = -0.01,
    tcl = NA,
    x_axis_side = 1,
    y_axis_side = 2,
    labs_col = fg,
    x_axis_line = 0,
    x_axis_las = 0,
    x_axis_padj = -1.1,
    x_axis_hadj = 0.5,
    y_axis_line = 0,
    y_axis_las = 1,
    y_axis_padj = 0.5,
    y_axis_hadj = 0.5,
    xlab_line = 1.4,
    ylab_line = 2,
    zerolines = TRUE,
    zerolines_col = fg,
    zerolines_alpha = 0.5,
    zerolines_lty = 1,
    zerolines_lwd = 1,
    main_line = 0.25,
    main_adj = 0,
    main_font = 2,
    main_col = fg,
    font_family = getOption("rtemis_font", "Helvetica")
  )

  theme_blackigrid(
    bg = "#000000",
    plot_bg = "#1A1A1A",
    fg = "#ffffff",
    pch = 16,
    cex = 1,
    lwd = 2,
    bty = "n",
    box_col = fg,
    box_alpha = 1,
    box_lty = 1,
    box_lwd = 0.5,
    grid = TRUE,
    grid_nx = NULL,
    grid_ny = NULL,
    grid_col = bg,
    grid_alpha = 1,
```

```
  grid_lty = 1,
  grid_lwd = 1,
  axes_visible = TRUE,
  axes_col = "transparent",
  tick_col = fg,
  tick_alpha = 1,
  tick_labels_col = fg,
  tck = -0.01,
  tcl = NA,
  x_axis_side = 1,
  y_axis_side = 2,
  labs_col = fg,
  x_axis_line = 0,
  x_axis_las = 0,
  x_axis_padj = -1.1,
  x_axis_hadj = 0.5,
  y_axis_line = 0,
  y_axis_las = 1,
  y_axis_padj = 0.5,
  y_axis_hadj = 0.5,
  xlab_line = 1.4,
  ylab_line = 2,
  zerolines = TRUE,
  zerolines_col = fg,
  zerolines_alpha = 0.5,
  zerolines_lty = 1,
  zerolines_lwd = 1,
  main_line = 0.25,
  main_adj = 0,
  main_font = 2,
  main_col = fg,
  font_family = getOption("rtemis_font", "Helvetica")
)

theme_darkgray(
  bg = "#121212",
  plot_bg = "transparent",
  fg = "#ffffff",
  pch = 16,
  cex = 1,
  lwd = 2,
  bty = "n",
  box_col = fg,
  box_alpha = 1,
  box_lty = 1,
  box_lwd = 0.5,
  grid = FALSE,
  grid_nx = NULL,
```

```
    grid_ny = NULL,
    grid_col = fg,
    grid_alpha = 0.2,
    grid_lty = 1,
    grid_lwd = 1,
    axes_visible = TRUE,
    axes_col = "transparent",
    tick_col = fg,
    tick_alpha = 0.5,
    tick_labels_col = fg,
    tck = -0.01,
    tcl = NA,
    x_axis_side = 1,
    y_axis_side = 2,
    labs_col = fg,
    x_axis_line = 0,
    x_axis_las = 0,
    x_axis_padj = -1.1,
    x_axis_hadj = 0.5,
    y_axis_line = 0,
    y_axis_las = 1,
    y_axis_padj = 0.5,
    y_axis_hadj = 0.5,
    xlab_line = 1.4,
    ylab_line = 2,
    zerolines = TRUE,
    zerolines_col = fg,
    zerolines_alpha = 0.5,
    zerolines_lty = 1,
    zerolines_lwd = 1,
    main_line = 0.25,
    main_adj = 0,
    main_font = 2,
    main_col = fg,
    font_family = getOption("rtemis_font", "Helvetica")
  )

  theme_darkgraygrid(
    bg = "#121212",
    plot_bg = "transparent",
    fg = "#ffffff",
    pch = 16,
    cex = 1,
    lwd = 2,
    bty = "n",
    box_col = fg,
    box_alpha = 1,
    box_lty = 1,
```

```
  box_lwd = 0.5,
  grid = TRUE,
  grid_nx = NULL,
  grid_ny = NULL,
  grid_col = "#404040",
  grid_alpha = 1,
  grid_lty = 1,
  grid_lwd = 1,
  axes_visible = TRUE,
  axes_col = "transparent",
  tick_col = "#00000000",
  tick_alpha = 1,
  tick_labels_col = fg,
  tck = -0.01,
  tcl = NA,
  x_axis_side = 1,
  y_axis_side = 2,
  labs_col = fg,
  x_axis_line = 0,
  x_axis_las = 0,
  x_axis_padj = -1.1,
  x_axis_hadj = 0.5,
  y_axis_line = 0,
  y_axis_las = 1,
  y_axis_padj = 0.5,
  y_axis_hadj = 0.5,
  xlab_line = 1.4,
  ylab_line = 2,
  zerolines = TRUE,
  zerolines_col = fg,
  zerolines_alpha = 0.5,
  zerolines_lty = 1,
  zerolines_lwd = 1,
  main_line = 0.25,
  main_adj = 0,
  main_font = 2,
  main_col = fg,
  font_family = getOption("rtemis_font", "Helvetica")
)

theme_darkgrayigrid(
  bg = "#121212",
  plot_bg = "#202020",
  fg = "#ffffff",
  pch = 16,
  cex = 1,
  lwd = 2,
  bty = "n",
```

```
  box_col = fg,
  box_alpha = 1,
  box_lty = 1,
  box_lwd = 0.5,
  grid = TRUE,
  grid_nx = NULL,
  grid_ny = NULL,
  grid_col = bg,
  grid_alpha = 1,
  grid_lty = 1,
  grid_lwd = 1,
  axes_visible = TRUE,
  axes_col = "transparent",
  tick_col = "transparent",
  tick_alpha = 1,
  tick_labels_col = fg,
  tck = -0.01,
  tcl = NA,
  x_axis_side = 1,
  y_axis_side = 2,
  labs_col = fg,
  x_axis_line = 0,
  x_axis_las = 0,
  x_axis_padj = -1.1,
  x_axis_hadj = 0.5,
  y_axis_line = 0,
  y_axis_las = 1,
  y_axis_padj = 0.5,
  y_axis_hadj = 0.5,
  xlab_line = 1.4,
  ylab_line = 2,
  zerolines = TRUE,
  zerolines_col = fg,
  zerolines_alpha = 0.5,
  zerolines_lty = 1,
  zerolines_lwd = 1,
  main_line = 0.25,
  main_adj = 0,
  main_font = 2,
  main_col = fg,
  font_family = getOption("rtemis_font", "Helvetica")
)

theme_white(
  bg = "#ffffff",
  plot_bg = "transparent",
  fg = "#000000",
  pch = 16,
```

```
  cex = 1,
  lwd = 2,
  bty = "n",
  box_col = fg,
  box_alpha = 1,
  box_lty = 1,
  box_lwd = 0.5,
  grid = FALSE,
  grid_nx = NULL,
  grid_ny = NULL,
  grid_col = fg,
  grid_alpha = 1,
  grid_lty = 1,
  grid_lwd = 1,
  axes_visible = TRUE,
  axes_col = "transparent",
  tick_col = fg,
  tick_alpha = 0.5,
  tick_labels_col = fg,
  tck = -0.01,
  tcl = NA,
  x_axis_side = 1,
  y_axis_side = 2,
  labs_col = fg,
  x_axis_line = 0,
  x_axis_las = 0,
  x_axis_padj = -1.1,
  x_axis_hadj = 0.5,
  y_axis_line = 0,
  y_axis_las = 1,
  y_axis_padj = 0.5,
  y_axis_hadj = 0.5,
  xlab_line = 1.4,
  ylab_line = 2,
  zerolines = TRUE,
  zerolines_col = fg,
  zerolines_alpha = 0.5,
  zerolines_lty = 1,
  zerolines_lwd = 1,
  main_line = 0.25,
  main_adj = 0,
  main_font = 2,
  main_col = fg,
  font_family = getOption("rtemis_font", "Helvetica")
)

theme_whitegrid(
  bg = "#ffffff",
```

```
    plot_bg = "transparent",
    fg = "#000000",
    pch = 16,
    cex = 1,
    lwd = 2,
    bty = "n",
    box_col = fg,
    box_alpha = 1,
    box_lty = 1,
    box_lwd = 0.5,
    grid = TRUE,
    grid_nx = NULL,
    grid_ny = NULL,
    grid_col = "#c0c0c0",
    grid_alpha = 1,
    grid_lty = 1,
    grid_lwd = 1,
    axes_visible = TRUE,
    axes_col = "transparent",
    tick_col = "#00000000",
    tick_alpha = 1,
    tick_labels_col = fg,
    tck = -0.01,
    tcl = NA,
    x_axis_side = 1,
    y_axis_side = 2,
    labs_col = fg,
    x_axis_line = 0,
    x_axis_las = 0,
    x_axis_padj = -1.1,
    x_axis_hadj = 0.5,
    y_axis_line = 0,
    y_axis_las = 1,
    y_axis_padj = 0.5,
    y_axis_hadj = 0.5,
    xlab_line = 1.4,
    ylab_line = 2,
    zerolines = TRUE,
    zerolines_col = fg,
    zerolines_alpha = 0.5,
    zerolines_lty = 1,
    zerolines_lwd = 1,
    main_line = 0.25,
    main_adj = 0,
    main_font = 2,
    main_col = fg,
    font_family = getOption("rtemis_font", "Helvetica")
  )
```

```
theme_whiteigrid(
  bg = "#ffffff",
  plot_bg = "#E6E6E6",
  fg = "#000000",
  pch = 16,
  cex = 1,
  lwd = 2,
  bty = "n",
  box_col = fg,
  box_alpha = 1,
  box_lty = 1,
  box_lwd = 0.5,
  grid = TRUE,
  grid_nx = NULL,
  grid_ny = NULL,
  grid_col = bg,
  grid_alpha = 1,
  grid_lty = 1,
  grid_lwd = 1,
  axes_visible = TRUE,
  axes_col = "transparent",
  tick_col = "transparent",
  tick_alpha = 1,
  tick_labels_col = fg,
  tck = -0.01,
  tcl = NA,
  x_axis_side = 1,
  y_axis_side = 2,
  labs_col = fg,
  x_axis_line = 0,
  x_axis_las = 0,
  x_axis_padj = -1.1,
  x_axis_hadj = 0.5,
  y_axis_line = 0,
  y_axis_las = 1,
  y_axis_padj = 0.5,
  y_axis_hadj = 0.5,
  xlab_line = 1.4,
  ylab_line = 2,
  zerolines = TRUE,
  zerolines_col = fg,
  zerolines_alpha = 0.5,
  zerolines_lty = 1,
  zerolines_lwd = 1,
  main_line = 0.25,
  main_adj = 0,
  main_font = 2,
```

```
  main_col = fg,
  font_family = getOption("rtemis_font", "Helvetica")
)

theme_lightgraygrid(
  bg = "#dfdfdf",
  plot_bg = "transparent",
  fg = "#000000",
  pch = 16,
  cex = 1,
  lwd = 2,
  bty = "n",
  box_col = fg,
  box_alpha = 1,
  box_lty = 1,
  box_lwd = 0.5,
  grid = TRUE,
  grid_nx = NULL,
  grid_ny = NULL,
  grid_col = "#c0c0c0",
  grid_alpha = 1,
  grid_lty = 1,
  grid_lwd = 1,
  axes_visible = TRUE,
  axes_col = "transparent",
  tick_col = "#00000000",
  tick_alpha = 1,
  tick_labels_col = fg,
  tck = -0.01,
  tcl = NA,
  x_axis_side = 1,
  y_axis_side = 2,
  labs_col = fg,
  x_axis_line = 0,
  x_axis_las = 0,
  x_axis_padj = -1.1,
  x_axis_hadj = 0.5,
  y_axis_line = 0,
  y_axis_las = 1,
  y_axis_padj = 0.5,
  y_axis_hadj = 0.5,
  xlab_line = 1.4,
  ylab_line = 2,
  zerolines = TRUE,
  zerolines_col = fg,
  zerolines_alpha = 0.5,
  zerolines_lty = 1,
  zerolines_lwd = 1,
```

```
  main_line = 0.25,
  main_adj = 0,
  main_font = 2,
  main_col = fg,
  font_family = getOption("rtemis_font", "Helvetica")
)

theme_mediumgraygrid(
  bg = "#b3b3b3",
  plot_bg = "transparent",
  fg = "#000000",
  pch = 16,
  cex = 1,
  lwd = 2,
  bty = "n",
  box_col = fg,
  box_alpha = 1,
  box_lty = 1,
  box_lwd = 0.5,
  grid = TRUE,
  grid_nx = NULL,
  grid_ny = NULL,
  grid_col = "#d0d0d0",
  grid_alpha = 1,
  grid_lty = 1,
  grid_lwd = 1,
  axes_visible = TRUE,
  axes_col = "transparent",
  tick_col = "#00000000",
  tick_alpha = 1,
  tick_labels_col = fg,
  tck = -0.01,
  tcl = NA,
  x_axis_side = 1,
  y_axis_side = 2,
  labs_col = fg,
  x_axis_line = 0,
  x_axis_las = 0,
  x_axis_padj = -1.1,
  x_axis_hadj = 0.5,
  y_axis_line = 0,
  y_axis_las = 1,
  y_axis_padj = 0.5,
  y_axis_hadj = 0.5,
  xlab_line = 1.4,
  ylab_line = 2,
  zerolines = TRUE,
  zerolines_col = fg,
```

```
  zerolines_alpha = 0.5,
  zerolines_lty = 1,
  zerolines_lwd = 1,
  main_line = 0.25,
  main_adj = 0,
  main_font = 2,
  main_col = fg,
  font_family = getOption("rtemis_font", "Helvetica")
)
```

Arguments

bg	Color: Figure background.
plot_bg	Color: Plot region background.
fg	Color: Foreground color used as default for multiple elements like axes and labels, which can be defined separately.
pch	Integer: Point character.
cex	Float: Character expansion factor.
lwd	Float: Line width.
bty	Character: Box type: "o", "l", "7", "c", "u", or "]", or "n".
box_col	Box color if `bty != "n"`.
box_alpha	Float: Box alpha.
box_lty	Integer: Box line type.
box_lwd	Float: Box line width.
grid	Logical: If TRUE, draw grid in plot regions.
grid_nx	Integer: N of vertical grid lines.
grid_ny	Integer: N of horizontal grid lines.
grid_col	Grid color.
grid_alpha	Float: Grid alpha.
grid_lty	Integer: Grid line type.
grid_lwd	Float: Grid line width.
axes_visible	Logical: If TRUE, draw axes.
axes_col	Axes colors.
tick_col	Tick color.
tick_alpha	Float: Tick alpha.
tick_labels_col	Tick labels' color.
tck	`graphics::parr`'s tck argument: Tick length, can be negative.
tcl	`graphics::parr`'s tcl argument.
x_axis_side	Integer: Side to place x-axis.
y_axis_side	Integer: Side to place y-axis.

labs_col	Labels' color.
x_axis_line	Numeric: graphics::axis's line argument for the x-axis.
x_axis_las	Numeric: graphics::axis's las argument for the x-axis.
x_axis_padj	Numeric: x-axis' padj: Adjustment for the x-axis tick labels' position.
x_axis_hadj	Numeric: x-axis' hadj.
y_axis_line	Numeric: graphics::axis's line argument for the y-axis.
y_axis_las	Numeric: graphics::axis's las argument for the y-axis.
y_axis_padj	Numeric: y-axis' padj.
y_axis_hadj	Numeric: y-axis' hadj.
xlab_line	Numeric: Line to place xlab.
ylab_line	Numeric: Line to place ylab.
zerolines	Logical: If TRUE, draw lines on x = 0, y = 0, if within plot limits.
zerolines_col	Zerolines color.
zerolines_alpha	Float: Zerolines alpha.
zerolines_lty	Integer: Zerolines line type.
zerolines_lwd	Float: Zerolines line width.
main_line	Float: How many lines away from the plot region to draw title.
main_adj	Float: How to align title.
main_font	Integer: 1: Regular, 2: Bold.
main_col	Title color.
font_family	Character: Font to be used throughout plot.

Value

Theme object.

Examples

```
theme <- theme_black(font_family = "Geist")
theme
```

`train` *Train Supervised Learning Models*

Description

Preprocess, tune, train, and test supervised learning models using nested resampling in a single call.

Usage

```
train(
  x,
  dat_validation = NULL,
  dat_test = NULL,
  weights = NULL,
  algorithm = NULL,
  preprocessor_config = NULL,
  hyperparameters = NULL,
  tuner_config = NULL,
  outer_resampling_config = NULL,
  execution_config = setup_ExecutionConfig(),
  question = NULL,
  outdir = NULL,
  verbosity = 1L,
  ...
)
```

Arguments

`x`	Tabular data, i.e. data.frame, data.table, or tbl_df (tibble): Training set data.
`dat_validation`	Tabular data: Validation set data.
`dat_test`	Tabular data: Test set data.
`weights`	Optional vector of case weights.
`algorithm`	Character: Algorithm to use. Can be left NULL, if `hyperparameters` is defined.
`preprocessor_config`	Optional PreprocessorConfig object: Setup using setup_Preprocessor.
`hyperparameters`	`Hyperparameters` object: Setup using one of `setup_*` functions.
`tuner_config`	TunerConfig object: Setup using setup_GridSearch.
`outer_resampling_config`	Optional ResamplerConfig object: Setup using setup_Resampler. This defines the outer resampling method, i.e. the splitting into training and test sets for the purpose of assessing model performance. If NULL, no outer resampling is performed, in which case you might want to use a `dat_test` dataset to assess model performance on a single test set.

execution_config
: ExecutionConfig object: Setup using setup_ExecutionConfig. This allows you to set backend ("future", "mirai", or "none"), number of workers, and future plan if using backend = "future".

question
: Optional character string defining the question that the model is trying to answer.

outdir
: Character, optional: String defining the output directory.

verbosity
: Integer: Verbosity level.

...
: Not used.

Details

Online book & documentation

See docs.rtemis.org/r for detailed documentation.

Preprocessing

There are many different stages at which preprocessing could be applied, when running a supervised learning pipeline with nested resampling. Some operations are best done before passing data to train():

- Duplicate rows should be removed before resampling, so that duplicates don't end up in different resamples, e.g. one in training and one in test.
- Constant columns should be removed before resampling. A column may appear constant in a small resample, even if it is not constant in the full dataset. Removing it inconsistently will throw an error during prediction.
- All data-dependent preprocessing steps need to be performed on training data only and applied on validation and test data, e.g. scaling, centering, imputation.

User-defined preprocessing through preprocessor_config is applied on training set data, the learned parameters are stored in the returned Supervised or SupervisedRes object, and the preprocessing is applied on validation and test data.

Binary Classification

For binary classification, the outcome should be a factor where *the 2nd level corresponds to the positive class*.

Resampling

Note that you should not use an outer resampling method with replacement if you will also be using an inner resampling (for tuning). The duplicated cases from the outer resampling may appear both in the training and test sets of the inner resamples, leading to underestimated test error.

Reproducibility

If using **outer resampling**, you can set a seed when defining outer_resampling_config, e.g.

```
outer_resampling_config = setup_Resampler(n_resamples = 10L, type = "KFold", seed = 2026L)
```

If using **tuning with inner resampling**, you can set a seed when defining tuner_config, e.g.

```
tuner_config = setup_GridSearch(
 resampler_config = setup_Resampler(n_resamples = 5L, type = "KFold", seed = 2027L)
)
```

Parallelization

There are three levels of parallelization that may be used during training:

1. Algorithm training (e.g. a parallelized learner like LightGBM)
2. Tuning (inner resampling, where multiple resamples can be processed in parallel)
3. Outer resampling (where multiple outer resamples can be processed in parallel)

The `train()` function will automatically manage parallelization depending on:

- The number of workers specified by the user using `n_workers`
- Whether the training algorithm supports parallelization itself
- Whether hyperparameter tuning is needed

Value

Object of class `Regression(Supervised)`, `RegressionRes(SupervisedRes)`, `Classification(Supervised)`, or `ClassificationRes(SupervisedRes)`.

Author(s)

EDG

Examples

```
iris_c_lightRF <- train(
   iris,
   algorithm = "LightRF",
   outer_resampling_config = setup_Resampler(),
)
```

---

uniprot_get *Get protein sequence from UniProt*

Description

Get protein sequence from UniProt

Usage

```
uniprot_get(
  accession,
  baseURL = "https://rest.uniprot.org/uniprotkb",
  verbosity = 1
)
```

Arguments

accession	Character: UniProt Accession number - e.g. "Q9UMX9"
baseURL	Character: UniProt rest API base URL. Default = "https://rest.uniprot.org/uniprotkb"
verbosity	Integer: Verbosity level.

Value

List with three elements: Identifier, Annotation, and Sequence.

Author(s)

E.D. Gennatas

Examples

```
## Not run:
# This gets the sequence from uniprot.org by default
mapt <- uniprot_get("Q9UMX9")

## End(Not run)
```

---

write_toml *Write to TOML file*

Description

Write to TOML file

Usage

```
write_toml(x, file, overwrite = FALSE, verbosity = 1L)
```

Arguments

x	SuperConfig object.
file	Character: Path to output TOML file.
overwrite	Logical: If TRUE, overwrite existing file.
verbosity	Integer: Verbosity level.

Value

SuperConfig object, invisibly.

Author(s)

EDG

Examples

```
x <- setup_SuperConfig(
  dat_training_path = "~/Data/iris.csv",
  dat_validation_path = NULL,
  dat_test_path = NULL,
  weights = NULL,
  preprocessor_config = setup_Preprocessor(remove_duplicates = TRUE),
  algorithm = "LightRF",
  hyperparameters = setup_LightRF(),
  tuner_config = setup_GridSearch(),
  outer_resampling_config = setup_Resampler(),
  execution_config = setup_ExecutionConfig(),
  question = "Can we tell iris species apart given their measurements?",
  outdir = "models/",
  verbosity = 1L
)
tmpdir <- tempdir()
write_toml(x, file.path(tmpdir, "rtemis.toml"))
```

---

xtdescribe *Describe longitudinal dataset*

Description

This function emulates the `xtdescribe` function in Stata.

Usage

```
xtdescribe(x, id_col = 1, time_col = 2, n_patterns = 9)
```

Arguments

`x`	data.frame: Longitudinal data with ID and time variables.
`id_col`	Integer: The column position of the ID variable.
`time_col`	Integer: The column position of the time variable.
`n_patterns`	Integer: The number of patterns to display.

Value

data.frame: Summary of participation patterns, returned invisibly.

Author(s)

EDG

Examples

```
# Load example longitudinal dataset
data(xt_example)

# Describe the longitudinal structure
xtdescribe(xt_example)
```

`xt_example` *Example longitudinal dataset*

Description

A small synthetic dataset demonstrating various participation patterns in longitudinal data, suitable for examples with `xtdescribe`.

Usage

```
xt_example
```

Format

A data frame with 30 rows and 4 variables:

patient_id Integer: Patient identifier (1-10).

year Integer: Year of measurement (2020-2024).

blood_pressure Numeric: Systolic blood pressure measurement.

treatment Character: Treatment group ("A" or "B").

Details

This dataset includes 10 patients measured at up to 5 time points (years 2020-2024). The dataset demonstrates various participation patterns typical in longitudinal studies:

- Complete participation (all time points)
- Early dropout
- Late entry
- Intermittent participation
- Single time point participation

Examples

```
data(xt_example)
head(xt_example)
summary(xt_example)
```

`%BC%` *Binary matrix times character vector*

Description

Binary matrix times character vector

Usage

```
x %BC% labels
```

Arguments

`x`	A binary matrix or data.frame
`labels`	Character vector length equal to `ncol(x)`

Value

a character vector

Author(s)

EDG

Index