Package: midr (via r-universe)

June 3, 2026

Type Package

Title Learning from Black-Box Models by Maximum Interpretation Decomposition

Version 0.6.1.900

Description The goal of 'midr' is to provide a model-agnostic method for interpreting and explaining black-box predictive models by creating a globally interpretable surrogate model. The package implements 'Maximum Interpretation Decomposition' (MID), a functional decomposition technique that finds an optimal additive approximation of the original model. This approximation is achieved by minimizing the squared error between the predictions of the black-box model and the surrogate model. The theoretical foundations of MID are described in Iwasawa & Matsumori (2025) [Forthcoming], and the package itself is detailed in Asashiba et al. (2025) <doi:10.48550/arXiv.2506.08338>.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Imports graphics, grDevices, Rcpp, RcppEigen, rlang, stats, utils

Suggests datasets, ggplot2, khroma, knitr, RColorBrewer, rmarkdown, scales, testthat, viridisLite

Config/testthat/edition 3

RoxygenNote 7.3.2

URL https://github.com/ryo-asashi/midr, https://ryo-asashi.github.io/midr/

BugReports https://github.com/ryo-asashi/midr/issues

LinkingTo Rcpp

Repository https://ryo-asashi.r-universe.dev

Date/Publication 2026-05-04 15:13:58 UTC

RemoteUrl https://github.com/ryo-asashi/midr

RemoteRef HEAD

RemoteSha 510310eef88ae6d4932d3e74de73fcc12167ddcc

Contents

`color.theme` *Color Themes for Graphics*

Description

The `color.theme()` function is the main interface for working with "color.theme" objects. It acts as a dispatcher that, depending on the class of `object`, can retrieve a pre-defined theme by name (see the "Theme Name Syntax" section), create a new theme from a vector of colors or a color-generating function, and modify an existing "color.theme" object.

Usage

```
color.theme(
  object,
  kernel.args = list(),
  options = list(),
  name = NULL,
  source = NULL,
  type = NULL,
  reverse = FALSE,
  env = color.theme.env(),
  ...
)
```

Arguments

`object`	a character string to retrieve a pre-defined theme, a color kernel (i.e., a vector of colors or a color generating function) to create a new theme, or a "color.theme" object to be modified. See the "Details" section.
`kernel.args`	a list of arguments to be passed to the color kernel.
`options`	a list of option values to control the color theme's behavior.
`name`	a character string for the color theme name.
`source`	a character string for the source name of the color theme.
`type`	a character string specifying the type of the color theme. One of "sequential", "diverging", or "qualitative".
`reverse`	logical. If `TRUE`, the order of colors is reversed.
`env`	an environment where the color themes are registered.
`...`	optional named arguments used to modify the color theme. Any argument passed here will override the corresponding settings in `kernel.args` or `options`.
`kernel`	a color vector, a palette function, or a ramp function that serves as the basis for generating colors.

Details

The "color.theme" object is a special environment that provides two color-generating functions: `...$palette()` and `...$ramp()`.

`...$palette()` takes an integer `n` and returns a vector of `n` discrete colors. It is primarily intended for qualitative themes, where distinct colors are used to represent categorical data.

`...$ramp()` takes a numeric vector `x` with values in the [0, 1] interval, and returns a vector of corresponding colors. It maps numeric values onto a continuous color gradient, making it suitable for sequential and diverging themes.

This function, `color.theme()`, is a versatile dispatcher that behaves differently depending on the class of the `object` argument. If `object` is a character string (e.g., "Viridis", "grDevices/RdBu_r@q?alpha=.5"), the string is parsed according to the theme name syntax, and the corresponding pre-defined theme is loaded (see the "Theme Name Syntax" section for details). If `object` is a color kernel (i.e., a character vector of colors, a palette function, or a ramp function), a new color theme is created from the kernel. If `object` is a "color.theme" object, the function returns a modified version of the theme, applying any other arguments to update its settings.

Value

`color.theme()` returns a "color.theme" object, which is an environment with the special class attribute, containing the `...$palette()` and `...$ramp` functions, along with other metadata about the theme.

Theme Name Syntax

When retrieving a theme using a character string, you can use a special syntax to specify the source and apply modifications:

`"[(source)/](name)[_r][@(type)][?(query)]"`

- source: (optional) the source package or collection of the theme (e.g., "grDevices").
- name: the name of the theme (e.g., "RdBu").
- "_r": (optional) a suffix to reverse the color order.
- type: (optional) the desired theme type, which will be matched with "sequential", "diverging" or "qualitative" (i.e., "s", "d", and "q" are sufficient, but longer strings such as "seq", "div", "qual" are also possible).
- query: (optional) a query string to overwrite the color theme's metadata including specific theme options or kernel arguments. Pairs are in `key=value` format and separated by `;` or `&` (e.g., "...?alpha=0.5;na.color='gray50'"). Possible keys include "name", "source", "type", "reverse" and any item of the theme's `options` and `kernel.args`.

See Also

`scale_color_theme`, `set.color.theme`, `color.theme.info`

Examples

```
# Retrieve a pre-defined theme
ct <- color.theme("Mako")
ct$palette(5L)
ct$ramp(seq.int(0, 1, length.out = 5))

# Use special syntax to get a reversed, qualitative theme with alpha value
ct <- color.theme("grDevices/Zissou 1_r@qual?alpha=0.75")
ct$palette(5L)
ct$ramp(seq.int(0, 1, length.out = 5))

# Create a new theme from a vector of colors
ct <- color.theme(c("#003f5c", "#7a5195", "#ef5675", "#ffa600"))
ct$palette(5L)
ct$ramp(seq.int(0, 1, length.out = 5))

# Create a new theme from a palette function
ct <- color.theme(grDevices::rainbow)
ct$palette(5L)
ct$ramp(seq.int(0, 1, length.out = 5))

# Modify an existing theme
ct <- color.theme(ct, type = "qualitative", kernel.args = list(v = 0.5))
ct$palette(5L)
ct$ramp(seq.int(0, 1, length.out = 5))
```

`color.theme.info` *Retrieve Color Theme Information*

Description

`color.theme.info()` returns a data frame listing all available color themes.

`color.theme.env()` provides direct access to the environment where the color themes are registered.

Usage

```
color.theme.info(env = color.theme.env())

color.theme.env()
```

Arguments

`env`	an environment where the color themes are registered.

Details

These functions provide tools for inspecting the color themes available in the current R session.

`color.theme.info()` is the primary user-facing function for discovering themes by name, source, and type.

`color.theme.env()` is an advanced function that returns the environment currently used as the theme registry. It first checks for a user-specified environment via `getOption("midr.color.theme.env")`. If this option is `NULL` (the default), the function returns the package's internal environment where the default themes are stored.

Value

`color.theme.info()` returns a data frame with columns "name", "source", and "type".

`color.theme.env()` returns the environment currently used as the default theme registry.

See Also

`color.theme`, `set.color.theme`

Examples

```
# Get a data frame of all available themes
head(color.theme.info())

# Get the environment where color themes are stored
theme_env <- color.theme.env()
names(theme_env)[1:5]
```

---

extract.midlist *Subset MID Objects*

Description

S3 methods to extract parts of a "midlist" or "midrib" collection object.

Usage

```
## S3 method for class 'midlist'
x[i, drop = if (missing(i)) TRUE else length(i) == 1L]

## S3 method for class 'midrib'
x[i, drop = if (missing(i)) TRUE else length(i) == 1L]

## S3 method for class 'midrib'
x[[i, exact = TRUE]]
```

Arguments

x	a collection object of class "midlist" or "midrib".
i	indices specifying elements to extract. Can be numeric, character, or logical vectors.
drop	logical. If TRUE the result is coerced to the lowest possible dimension. For a collection (e.g., "mids"), extracting a single element drops it to a base object (e.g., "mid").
exact	logical. If TRUE, exact matching is used for character strings.

Details

A "midlist" or "midrib" object stores multiple objects of the same single base class: "mid", "midimp", "midcon", or "midbrk".

When extracting items using [, it returns a subsetted "midlist" or "midrib" object, preserving its collection class (e.g., "mids", "midimps"). By default, if a single base object is extracted (length(i) == 1L) and drop = TRUE, the object is simplified to a single base object (e.g., "mid", "midimp"). [[always extracts a single base object.

Value

[returns a subsetted collection object or a single base object if drop = TRUE.

[[returns a single base object.

See Also

midlist, labels.midlist

Examples

```
# Fit a multivariate linear model
fit <- lm(cbind(y1, y2, y3) ~ x1 + I(x1^2), data = anscombe)

# Interpret the linear models
collection <- interpret(cbind(y1, y2, y3) ~ x1, data = anscombe, model = fit)

# Check the default labels
labels(collection)

# Rename the models in the collection
labels(collection) <- letters[1L:3L]
labels(collection)

# Extract a single base "mid" object by its new name using [[
mid <- collection[["a"]]
class(mid)

# Subset the collection to keep only the first two models using [
sub <- collection[1:2]
class(sub) # Maintains the collection class (e.g., "mids"-"midrib")
```

`factor.encoder` *Encoder for Qualitative Variables*

Description

`factor.encoder()` creates an encoder function for a qualitative (factor or character) variable. This encoder converts the variable into a one-hot encoded (dummy) design matrix.

`factor.frame()` is a helper function to create a "factor.frame" object that defines the encoding scheme.

Usage

```
factor.encoder(
  x,
  k = NULL,
  lump = c("none", "auto", "rank", "order"),
  others = "others",
  sep = ">",
  weights = NULL,
  frame = NULL,
  tag = "x"
)

factor.frame(levels, others = NULL, map = NULL, original = NULL, tag = "x")
```

Arguments

`x`	a vector to be encoded as a qualitative variable.
`k`	an integer specifying the maximum number of distinct levels to retain (including the catch-all level). If not positive, all unique values of `x` are used.
`lump`	a character string specifying the lumping strategy: `"none"`, no lumping is performed; `"rank"`, lumps levels based on frequency rank; `"order"` merges adjacent levels based on cumulative frequency to preserve order; and `"auto"` automatically selects `"order"` for ordered factors and `"rank"` for others.
`others`	a character string for the catch-all level (used when `lump = "rank"`).
`sep`	a character string used to separate the start and end levels when merging ordered factors (e.g., "Level1..Level3").
`weights`	an optional numeric vector of sample weights for `x`.
`frame`	a "factor.frame" object or a character vector that explicitly defines the levels of the variable.
`tag`	the name of the variable.
`levels`	a vector to be used as the levels of the variable.
`map`	a named vector that maps original levels to lumped levels.
`original`	a character vector to be used as the original levels for expanding the frame. Defaults to `NULL`.

Details

This function is designed to handle qualitative data for use in the MID model's linear system formulation.

The primary mechanism is one-hot encoding. Each unique level of the input variable becomes a column in the output matrix. For a given observation, the column corresponding to its level is assigned a `1`, and all other columns are assigned `0`.

When a variable has many unique levels (high cardinality), you can use the `lump` and `k` arguments to reduce dimensionality. This is crucial for preventing MID models from becoming overly complex.

Value

`factor.encoder()` returns an object of class "encoder". This is a list containing the following components:

`frame`	a "factor.frame" object containing the encoding information (levels).
`n`	the number of encoding levels (i.e., columns in the design matrix).
`type`	a character string describing the encoding type: "factor" or "null".
`envir`	an environment for the `transform` and `encode` functions.
`transform`	a function `transform(x, lumped = TRUE, ...)` that converts a vector into a factor with the encoded levels.
`encode`	a function `encode(x, ...)` that converts a vector into the one-hot encoded matrix.

`factor.frame()` returns a "factor.frame" object containing the encoding information.

See Also

`numeric.encoder`

Examples

```
# Create an encoder for a qualitative variable
data(iris, package = "datasets")
enc <- factor.encoder(x = iris$Species, lump = "none", tag = "Species")
enc

# Encode a vector with NA
enc$encode(iris$Species[c(50, 100, 150)])

# Lumping by rank (retain top k - 1 levels and others)
enc <- factor.encoder(x = iris$Species, k = 2, lump = "rank")
enc$encode(iris$Species[c(50, 100, 150)])

# Lumping by order (merge adjacent levels)
enc <- factor.encoder(x = iris$Species, k = 2, lump = "order")
enc$encode(iris$Species[c(50, 100, 150)])
```

get.link *Extended Parametric Link Functions*

Description

`get.link()` creates a link function object (inheriting from "link-glm") capable of handling parametric transformations such as Box-Cox, Yeo-Johnson, and shifted logarithms. This function serves as a wrapper and extension to `make.link()`.

Usage

```
get.link(link, ..., simplify = TRUE)
```

Arguments

`link`	a character string naming the link function: "log1p", "shifted.log", "shifted.identity", "robit", "ashinh", "scobit", "box-cox", "box-cox2", or "yeo-johnson". Standard links (e.g., "logit", "probit", "log") are passed to `stats::make.link()`.
`...`	named arguments passed to the specific link generation logic. See Details for available parameters and defaults.
`simplify`	logical. If `TRUE` (default), the function returns a standard link object or recursively calls `get.link` when parameters equate to a simpler and more computationally efficient form (e.g., `box-cox` with `lambda=0` becomes `log`).

Details

The available links and their parameters are:

- "log1p": shifted log link $\eta = \log(\mu + 1)$.
- "shifted.log": shifted log link $\eta = \log(\mu + h)$ with a shift parameter `h` (default 1).
- "shifted.identity": shifted identity link $\eta = \mu + h$ with a shift parameter `h` (default 1).
- "robit": robit link using the Student's t-distribution CDF. $\eta = F_t^{-1}(\mu, \nu)$ with a degrees of freedom parameter `df` (or alias `nu`, default 7).
- "asinh": inverse hyperbolic sine transformation $\eta = \text{asinh}(\lambda\mu)$ with a scale parameter `lambda` (default 1).
- "scobit": skewed logit link $\eta = \text{logit}(\mu^{1/\alpha})$. The parameter `alpha` (default 1) controls the asymmetry of the tails. `alpha=1` corresponds to standard logistic regression.
- "box-cox": Box-Cox transformation $\eta = (\mu^\lambda - 1)/\lambda$ with a power parameter `lambda` (default 0).
- "box-cox2": two-parameter Box-Cox transformation $\eta = ((\mu+\lambda_2)^{\lambda_1} - 1)/\lambda_1$ with parameters `lambda1` (power, default 0) and `lambda2` (shift, default 1).
- "yeo-johnson": Yeo-Johnson transformation with a parameter `lambda` (default 0). Handles negative values.

Value

`get.link()` returns an object of class "link-glm" and "parametric.link" containing:

`linkfun`	link function $g(\mu)$.
`linkinv`	inverse link function $g^{-1}(\eta)$.
`mu.eta`	derivative $d\mu/d\eta$.
`valideta`	a function checking validity of linear predictors.
`name`	name of the link.

See Also

`make.link`

Examples

```
# Standard Box-Cox with lambda = 0.5 (Square root-like)
lk <- get.link("box-cox", lambda = 0.5)
plot(x <- seq(1, 100, length.out = 50), lk$linkfun(x), type = "l")

# Yeo-Johnson with lambda = 1.5 (Handles negative values)
lk <- get.link("yeo-johnson", lambda = 1.5)
plot(x <- seq(-100, 100, length.out = 50), lk$linkfun(x), type = "l")

# Robit link with df=2 (Heavier tails than probit)
lk <- get.link("robit", df = 2)
print(lk)
plot(x <- seq(-5, 5, length.out = 50), lk$linkinv(x), type = "l")
lk <- get.link("robit", df = 1)
cat(lk$name) # cauchit
points(x, lk$linkinv(x), type = "l", lty = 2L)
lk <- get.link("robit", df = Inf)
cat(lk$name) # probit
points(x, lk$linkinv(x), type = "l", lty = 3L)

# Scobit link with alpha=0.5 (Skewed)
lk <- get.link("scobit", alpha = 0.5)
plot(x <- seq(-5, 5, length.out = 50), lk$linkinv(x), type = "l")

# Inverse Hyperbolic Sine (Alternative to log for zero-inflated data)
lk <- get.link("asinh", lambda = 10)
plot(x <- seq(0, 5, length.out = 50), lk$linkfun(x), type = "l")
lk <- get.link("log1p")
points(x, lk$linkfun(x), type = "l", lty = 2L)

# Shifted log simplifies to log1p when h=1
get.link("shifted.log", h = 1) # Returns link="log1p"

# Box-Cox with lambda=1 simplifies to shifted identity
get.link("box-cox", lambda = 1)
```

get.yhat *Wrapper Prediction Function*

Description

get.yhat() is a generic function that provides a unified interface for obtaining predictions from various fitted model objects.

Usage

```
get.yhat(object, newdata, ..., target)

## Default S3 method:
get.yhat(object, newdata, ..., target = -1L)

## S3 method for class 'lm'
get.yhat(object, newdata, ...)

## S3 method for class 'glm'
get.yhat(object, newdata, ...)

## S3 method for class 'gam'
get.yhat(object, newdata, ...)

## S3 method for class 'mid'
get.yhat(object, newdata, ...)

## S3 method for class 'mids'
get.yhat(object, newdata, ..., target = -1L)

## S3 method for class 'rpart'
get.yhat(object, newdata, ..., target = -1L)

## S3 method for class 'randomForest'
get.yhat(object, newdata, ..., target = -1L)

## S3 method for class 'ranger'
get.yhat(object, newdata, ..., target = -1L)

## S3 method for class 'rfsrc'
get.yhat(object, newdata, ..., target = -1L)

## S3 method for class 'ObliqueForest'
get.yhat(object, newdata, ..., target = -1L)

## S3 method for class 'svm'
get.yhat(object, newdata, ..., target = -1L)
```

```
## S3 method for class 'ksvm'
get.yhat(object, newdata, ..., target = -1L)

## S3 method for class 'AccurateGLM'
get.yhat(object, newdata, ..., target = -1L)

## S3 method for class 'glmnet'
get.yhat(object, newdata, ..., target = -1L)

## S3 method for class 'model_fit'
get.yhat(object, newdata, ..., target = -1L)

## S3 method for class 'workflow'
get.yhat(object, newdata, ..., target = -1L)

## S3 method for class 'rpf'
get.yhat(object, newdata, ..., target = -1L)

## S3 method for class 'coxph'
get.yhat(object, newdata, ...)

## S3 method for class 'flexsurvreg'
get.yhat(object, newdata, ...)

## S3 method for class 'mboost'
get.yhat(object, newdata, ..., target = -1L)

## S3 method for class 'fitlist'
get.yhat(object, newdata, ..., target = -1L)
```

Arguments

`object`	a fitted model object.
`newdata`	a data.frame or matrix.
`...`	optional named arguments passed on to the underlying `predict()` method for the `object`'s class.
`target`	an integer or character vector specifying the target levels used for the classification models that return a matrix or data frame of class probabilities. The default, `-1`, represents the probability of not being the base level.

Details

While many predictive models have a `stats::predict()` method, the structure and type of their outputs are not uniform. For example, some return a numeric vector, others a matrix of class probabilities, and some a list. This function, `get.yhat()`, abstracts away this complexity.

For regression models, it returns the numeric prediction in the original scale of the response variable. For classification models, it returns the sum of class probabilities for the classes specified by the

target argument.

Furthermore, get.yhat() provides more consistent handling of missing values. While some stats::predict() methods may return a shorter vector by omitting NAs, get.yhat() is designed to return a vector of the same length as newdata, preserving NAs in their original positions.

The design of get.yhat() is strongly influenced by DALEX::yhat().

Value

get.yhat() returns a numeric vector of model predictions for newdata.

See Also

predict.mid

Examples

```
data(trees, package = "datasets")
model <- glm(Volume ~ ., trees, family = Gamma(log))

# The output of stats::predict() might not be in the scale of the response variable
predict(model, trees[1:5, ])

# get.yhat() returns a numeric vector in the original scale of the response variable
get.yhat(model, trees[1:5, ])
predict(model, trees[1:5, ], type = "response")
```

---

ggmid *Plot MID Component Function with ggplot2*

Description

ggmid() is an S3 generic function for creating various visualizations from MID-related objects using **ggplot2**. For "mid" objects (i.e., fitted MID models), it visualizes a single component function specified by the term argument.

Usage

```
ggmid(object, ...)

## S3 method for class 'mid'
ggmid(
  object,
  term,
  type = c("effect", "data", "compound"),
  theme = NULL,
  intercept = FALSE,
  main.effects = FALSE,
  data = NULL,
```

```
  limits = c(NA, NA),
  jitter = NULL,
  resolution = c(100L, 100L),
  lumped = TRUE,
  ...
)

## S3 method for class 'mid'
autoplot(object, ...)
```

Arguments

object	a "mid" object to be visualized.
...	optional parameters passed to the main plotting layer.
term	a character string specifying the component function to be plotted.
type	the plotting style. One of "effect", "data" or "compound".
theme	a character string or object defining the color theme. See color.theme for details.
intercept	logical. If TRUE, the intercept is added to the MID values.
main.effects	logical. If TRUE, main effects are included in the interaction plot.
data	a data frame to be plotted with the corresponding MID values. If not provided, data is automatically extracted based on the function call.
limits	a numeric vector of length two specifying the limits of the plotting scale. NA values are replaced by the minimum and/or maximum MID values.
jitter	a numeric value specifying the amount of jitter for the data points.
resolution	an integer or vector of two integers specifying the resolution of the raster plot for interactions.
lumped	logical. If TRUE, uses the lumped factor levels; if FALSE, uses the original levels from the data. Always FALSE when main.effects = TRUE.

Details

For "mid" objects, ggmid() creates a "ggplot" object that visualizes a component function of the fitted MID model.

The type argument controls the visualization style. The default, type = "effect", plots the component function itself. In this style, the plotting method is automatically selected based on the effect's type: a line plot for quantitative main effects; a bar plot for qualitative main effects; and a raster plot for interactions. The type = "data" option creates a scatter plot of data, colored by the values of the component function. The type = "compound" option combines both approaches, plotting the component function alongside the data points.

Value

ggmid.mid() returns a "ggplot" object.

See Also

interpret, ggmid.midimp, ggmid.midcon, ggmid.midbrk, plot.mid

Examples

```
data(diamonds, package = "ggplot2")
set.seed(42)
idx <- sample(nrow(diamonds), 1e4)
mid <- interpret(price ~ (carat + cut + color + clarity)^2, diamonds[idx, ])

# Plot a quantitative main effect
ggmid(mid, "carat")

# Plot a qualitative main effect
ggmid(mid, "clarity")

# Plot an interaction effect with data points and a raster layer
ggmid(mid, "carat:clarity", type = "compound", data = diamonds[idx, ])

# Use a different color theme
ggmid(mid, "clarity:color", theme = "RdBu")
```

---

ggmid.midbrk *Plot MID Breakdown with ggplot2*

Description

For "midbrk" objects, ggmid() visualizes the breakdown of a prediction by component functions.

Usage

```
## S3 method for class 'midbrk'
ggmid(
  object,
  type = c("waterfall", "barplot", "dotchart"),
  theme = NULL,
  terms = NULL,
  max.nterms = 15L,
  vline = TRUE,
  others = "others",
  pattern = c("%t=%v", "%t:%t"),
  format.args = list(),
  ...
)

## S3 method for class 'midbrk'
autoplot(object, ...)
```

Arguments

object	a "midbrk" object to be visualized.
type	the plotting style. One of "waterfall", "barplot" or "dotchart".
theme	a character string or object defining the color theme. See color.theme for details.
terms	an optional character vector specifying which terms to display.
max.nterms	the maximum number of terms to display in the plot. Less important terms will be grouped into a "catchall" category.
vline	logical. If TRUE, a vertical line is drawn at the zero or intercept line.
others	a character string for the catchall label.
pattern	a character vector of length one or two specifying the format of the axis labels. The first element is used for main effects (default "%t = %v"), and the second is for interactions (default "%t:%t"). Use "%t" for the term name and "%v" for its value.
format.args	a named list of additional arguments passed to format for formatting the values. Common arguments include digits, nsmall, and big.mark.
...	optional parameters passed on to the main layer.

Details

This is an S3 method for the ggmid() generic that creates a breakdown plot from a "midbrk" object, visualizing the contribution of each component function to a single prediction.

The type argument controls the visualization style. The default, type = "waterfall", creates a waterfall plot that shows how the prediction is built up from the intercept, with each term's contribution sequentially added or subtracted. The type = "barplot" option creates a standard bar plot where the length of each bar represents the magnitude of the term's contribution. The type = "dotchart" option creates a dot plot showing the contribution of each term as a point connected to a zero baseline.

Value

ggmid.midbrk() returns a "ggplot" object.

See Also

mid.breakdown, ggmid, plot.midbrk

Examples

```
data(diamonds, package = "ggplot2")
set.seed(42)
idx <- sample(nrow(diamonds), 1e4)
mid <- interpret(price ~ (carat + cut + color + clarity)^2, diamonds[idx, ])
mbd <- mid.breakdown(mid, diamonds[1L, ])

# Create a waterfall plot
```

```
ggmid(mbd, type = "waterfall")

# Create a bar plot with a different theme
ggmid(mbd, type = "barplot", theme = "highlight")

# Create a dot chart
ggmid(mbd, type = "dotchart", size = 3)
```

`ggmid.midbrks` *Compare MID Breakdowns with ggplot2*

Description

For "midbrks" collection objects, `ggmid()` visualizes and compares the breakdown of a prediction by component functions.

Usage

```
## S3 method for class 'midbrks'
ggmid(
  object,
  type = c("barplot", "dotchart", "series"),
  theme = NULL,
  terms = NULL,
  max.nterms = 15L,
  vline = TRUE,
  others = "others",
  pattern = c("%t=%v", "%t:%t"),
  format.args = list(),
  labels = NULL,
  ...
)

## S3 method for class 'midbrks'
autoplot(object, ...)
```

Arguments

`object`	a "midbrks" collection object to be visualized.
`type`	the plotting style. One of "barplot", "dotchart", or "series".
`theme`	a character string or object defining the color theme. See `color.theme` for details.
`terms`	an optional character vector specifying which terms to display. If `NULL`, terms are automatically extracted from the object.
`max.nterms`	the maximum number of terms to display. Defaults to 15.
`vline`	logical. If `TRUE`, a vertical line is drawn at the zero or intercept line.

others	a character string for the catchall label. Defaults to "others".
pattern	a character vector of length one or two specifying the format of the axis labels. The first element is used for main effects (default "%t = %v"), and the second is for interactions (default "%t:%t"). Use "%t" for the term name and "%v" for its value.
format.args	a named list of additional arguments passed to format for formatting the values. Common arguments include digits, nsmall, and big.mark.
labels	an optional numeric or character vector to specify the model labels or x-axis coordinates. Defaults to the labels found in the object.
...	optional parameters passed on to the main layer (e.g., geom_col).

Details

This is an S3 method for the ggmid() generic that creates a comparative importance plot from a "midbrks" collection object. It visualizes the contribution of each component function to a single prediction across multiple models, allowing for easy comparison across different models.

The type argument controls the visualization style: The default, type = "barplot", creates a grouped bar plot where the bars for each term are placed side-by-side across the models. The type = "dotchart" option creates a grouped dot plot, offering a cleaner comparison across models. The type = "series" option plots the contribution trend over the models for each component term.

Value

ggmid.midbrks() returns a "ggplot" object.

See Also

ggmid.midbrk, plot.midbrks

Examples

```
data(mtcars, package = "datasets")

# Fit two different models for comparison
mid1 <- interpret(mpg ~ wt + hp + cyl, data = mtcars)
mid2 <- interpret(mpg ~ (wt + hp + cyl)^2, data = mtcars)

# Calculate importance for both models and combine them
brks <- midlist(
  "Main Effects" = mid.breakdown(mid1, data = mtcars[1, ]),
  "Interactions" = mid.breakdown(mid2, data = mtcars[1, ])
)

# Create a comparative grouped bar plot (default)
ggmid(brks)

# Create a comparative dot chart with a specific theme
ggmid(rev(brks), type = "dotchart", theme = "R4")
```

```
# Create a series plot to observe trends across models
ggmid(brks, type = "series")
```

`ggmid.midcon` *Plot MID Conditional Expectation with ggplot2*

Description

For "midcon" objects, `ggmid()` visualizes Individual Conditional Expectation (ICE) curves derived from a fitted MID model.

Usage

```
## S3 method for class 'midcon'
ggmid(
  object,
  type = c("iceplot", "centered"),
  theme = NULL,
  term = NULL,
  var.alpha = NULL,
  var.color = NULL,
  var.linetype = NULL,
  var.linewidth = NULL,
  reference = 1L,
  points = TRUE,
  sample = NULL,
  ...
)

## S3 method for class 'midcon'
autoplot(object, ...)
```

Arguments

`object`	a "midcon" object to be visualized.
`type`	the plotting style. One of "iceplot" or "centered".
`theme`	a character string or object defining the color theme. See `color.theme` for details.
`term`	an optional character string specifying an interaction term. If passed, the ICE curve for the specified term is plotted.
`var.alpha`	a variable name or expression to map to the alpha aesthetic.
`var.color`	a variable name or expression to map to the color aesthetic.
`var.linetype`	a variable name or expression to map to the linetype aesthetic.
`var.linewidth`	a variable name or expression to map to the linewidth aesthetic.

reference	an integer specifying the index of the evaluation point to use as the reference for centering the c-ICE plot.
points	logical. If TRUE, points representing the actual predictions for each observation are plotted.
sample	an optional vector specifying the names of observations to be plotted.
...	optional parameters passed on to the main layer.

Details

This is an S3 method for the ggmid() generic that produces ICE curves from a "midcon" object. ICE plots are a model-agnostic tool for visualizing how a model's prediction for a single observation changes as one feature varies. This function plots one line for each observation in the data.

The type argument controls the visualization style: The default, type = "iceplot", plots the raw ICE curves. The type = "centered" option creates the centered ICE (c-ICE) plot, where each curve is shifted to start at zero, making it easier to compare the slopes of the curves.

The var.color, var.alpha, etc., arguments allow you to map aesthetics to other variables in your data using (possibly) unquoted expressions.

Value

ggmid.midcon() returns a "ggplot" object.

See Also

mid.conditional, ggmid, plot.midcon

Examples

```
data(airquality, package = "datasets")
mid <- interpret(Ozone ~ .^2, airquality, lambda = 0.1)
ice <- mid.conditional(mid, "Temp", data = airquality)

# Create an ICE plot, coloring lines by 'Wind'
ggmid(ice, var.color = "Wind")

# Create a centered ICE plot, mapping color and linetype to other variables
ggmid(ice, type = "centered", theme = "Purple-Yellow",
      var.color = factor(Month), var.linetype = Wind > 10)
```

---

ggmid.midcons *Compare MID Conditional Expectations with ggplot2*

Description

For "midcons" collection objects, ggmid() visualizes and compares Individual Conditional Expectation (ICE) curves derived from multiple fitted MID models.

Usage

```
## S3 method for class 'midcons'
ggmid(
  object,
  type = c("iceplot", "centered", "series"),
  theme = NULL,
  var.alpha = NULL,
  var.linetype = NULL,
  var.linewidth = NULL,
  reference = 1L,
  sample = NULL,
  labels = NULL,
  ...
)

## S3 method for class 'midcons'
autoplot(object, ...)
```

Arguments

object	a "midcons" collection object to be visualized.
type	the plotting style. One of "iceplot", "centered", or "series".
theme	a character string or object defining the color theme. See color.theme for details.
var.alpha	a variable name or expression to map to the alpha aesthetic.
var.linetype	a variable name or expression to map to the linetype aesthetic.
var.linewidth	a variable name or expression to map to the linewidth aesthetic.
reference	an integer specifying the index of the evaluation point to use as the reference for centering the c-ICE plot.
sample	an optional vector specifying the names of observations to be plotted.
labels	an optional numeric or character vector to specify the model labels. Defaults to the labels found in the object.
...	optional parameters passed on to the main layer.

Details

This is an S3 method for the ggmid() generic that produces comparative ICE curves from a "midcons" object. It plots one line for each observation in the data per model.

For type = "iceplot" and "centered", lines are colored by the model label. For type = "series", lines are colored by the feature value and plotted across models.

The var.alpha, var.linetype, and var.linewidth arguments allow you to map aesthetics to other variables in your data using (possibly) unquoted expressions.

Value

ggmid.midcons() returns a "ggplot" object.

See Also

ggmid.midcon, plot.midcons

Examples

```
data(mtcars, package = "datasets")

# Fit two different models for comparison
mid1 <- interpret(mpg ~ wt + hp + cyl, data = mtcars)
mid2 <- interpret(mpg ~ (wt + hp + cyl)^2, data = mtcars)

# Calculate conditional expectations for both models
cons <- midlist(
  "Main Effects" = mid.conditional(mid1, "wt", data = mtcars[3:5, ]),
  "Interactions" = mid.conditional(mid2, "wt", data = mtcars[3:5, ])
)

# Create an ICE plot (default)
ggmid(cons)

# Create a centered-ICE plot
ggmid(cons, type = "centered")

# Create a series plot to observe trends across models
ggmid(cons, type = "series", var.linetype = ".id")
```

---

ggmid.midimp *Plot MID Importance with ggplot2*

Description

For "midimp" objects, ggmid() visualizes the importance of component functions of the fitted MID model.

Usage

```
## S3 method for class 'midimp'
ggmid(
  object,
  type = c("barplot", "dotchart", "heatmap", "boxplot"),
  theme = NULL,
  terms = NULL,
  max.nterms = 30L,
  ...
)

## S3 method for class 'midimp'
autoplot(object, ...)
```

Arguments

object	a "midimp" object to be visualized.
type	the plotting style. One of "barplot", "dotchart", "heatmap", or "boxplot".
theme	a character string or object defining the color theme. See color.theme for details.
terms	an optional character vector specifying which terms to display.
max.nterms	the maximum number of terms to display. Defaults to 30 for bar, dot and box plots.
...	optional parameters passed on to the main layer.

Details

This is an S3 method for the ggmid() generic that creates an importance plot from a "midimp" object, visualizing the average contribution of component functions to the fitted MID model.

The type argument controls the visualization style. The default, type = "barplot", creates a standard bar plot where the length of each bar represents the overall importance of the term. The type = "dotchart" option creates a dot plot, offering a clean alternative to the bar plot for visualizing term importance. The type = "heatmap" option creates a matrix-shaped heat map where the color of each cell represents the importance of the interaction between a pair of variables, or the main effect on the diagonal. The type = "boxplot" option creates a box plot where each box shows the distribution of a term's contributions across all observations, providing insight into the variability of each term's effect.

Value

ggmid.midimp() returns a "ggplot" object.

See Also

mid.importance, ggmid, plot.midimp

Examples

```
data(diamonds, package = "ggplot2")
set.seed(42)
idx <- sample(nrow(diamonds), 1e4)
mid <- interpret(price ~ (carat + cut + color + clarity)^2, diamonds[idx, ])
imp <- mid.importance(mid)

# Create a bar plot (default)
ggmid(imp)

# Create a dot chart
ggmid(imp, type = "dotchart", theme = "Okabe-Ito", size = 3)

# Create a heatmap
ggmid(imp, type = "heatmap")
```

```
# Create a boxplot to see the distribution of effects
ggmid(imp, type = "boxplot")
```

`ggmid.midimps` *Compare MID Importances with ggplot2*

Description

For "midimps" collection objects, `ggmid()` visualizes and compares the importance of component functions across multiple fitted MID models.

Usage

```
## S3 method for class 'midimps'
ggmid(
  object,
  type = c("barplot", "dotchart", "series"),
  theme = NULL,
  terms = NULL,
  max.nterms = 15L,
  labels = NULL,
  ...
)

## S3 method for class 'midimps'
autoplot(object, ...)
```

Arguments

`object`	a "midimps" collection object to be visualized.
`type`	the plotting style. One of "barplot", "dotchart", or "series".
`theme`	a character string or object defining the color theme. See `color.theme` for details.
`terms`	an optional character vector specifying which terms to display. If `NULL`, terms are automatically extracted from the object.
`max.nterms`	the maximum number of terms to display. Defaults to 15.
`labels`	an optional numeric or character vector to specify the model labels. Defaults to the labels found in the object.
`...`	optional parameters passed on to the main layer (e.g., `geom_col`).

Details

This is an S3 method for the ggmid() generic that creates a comparative importance plot from a "midimps" collection object. It visualizes the average contribution of component functions to the fitted MID models, allowing for easy comparison across different models.

The type argument controls the visualization style: The default, type = "barplot", creates a standard grouped bar plot where the length of each bar represents the overall importance of the term, positioned side-by-side by model label. The type = "dotchart" option creates a grouped dot plot, offering a clean alternative to the bar plot for visualizing and comparing term importance across models. The type = "series" option plots the importance trend over the models for each component function.

Value

ggmid.midimps() returns a "ggplot" object.

See Also

ggmid.midimp, plot.midimps

Examples

```
data(mtcars, package = "datasets")

# Fit two different models for comparison
mid1 <- interpret(mpg ~ wt + hp + cyl, data = mtcars)
mid2 <- interpret(mpg ~ (wt + hp + cyl)^2, data = mtcars)

# Calculate importance for both models and combine them
imps <- midlist(
  "Main Effects" = mid.importance(mid1),
  "Interactions" = mid.importance(mid2)
)

# Create a comparative grouped bar plot (default)
ggmid(imps)

# Create a comparative dot chart with a specific theme
ggmid(rev(imps), type = "dotchart", theme = "Okabe-Ito")

# Create a series plot to observe trends across models
ggmid(imps, type = "series")
```

---

ggmid.mids *Compare MID Component Functions with ggplot2*

Description

For "mids" collection objects, ggmid() visualizes and compares a single main effect across multiple models.

Usage

```
## S3 method for class 'mids'
ggmid(
  object,
  term,
  type = c("effect", "series"),
  theme = NULL,
  intercept = FALSE,
  limits = c(NA, NA),
  resolution = NULL,
  labels = base::labels(object),
  ...
)

## S3 method for class 'mids'
autoplot(object, ...)
```

Arguments

object	a "mids" collection object to be visualized.
term	a character string specifying the main effect to evaluate.
type	the plotting style: "effect" plots the effect curve per model, while "series" plots the effect trend over models per feature value.
theme	a character string or object defining the color theme. See color.theme for details.
intercept	logical. If TRUE, the model intercept is added to the component effect.
limits	a numeric vector of length two specifying the limits of the plotting scale. NA values are replaced by the minimum and/or maximum MID values.
resolution	an integer specifying the number of evaluation points for continuous variables.
labels	an optional numeric or character vector to specify the model labels. Defaults to labels(object). The function attempts to parse these labels into numeric values where possible.
...	optional parameters passed to the main layer (e.g., linewidth, alpha).

Details

This is an S3 method for the ggmid() generic that evaluates the specified term over a grid of values and compares the results across all models in the collection.

The type argument controls the visualization style. The default, type = "effect", plots the component functions of the specified term for each model individually. The type = "series" option transposes the view to plot the effect trend over the models for each feature value.

Note: Comparative plotting for interaction terms (2D surfaces) is not supported for collection objects.

Value

ggmid.mids() returns a "ggplot" object.

See Also

ggmid, plot.mids

Examples

```
# Use a lightweight dataset for fast execution
data(mtcars, package = "datasets")

# Fit two models with different complexities
fit1 <- lm(mpg ~ wt, data = mtcars)
mid1 <- interpret(mpg ~ wt, data = mtcars, model = fit1)
fit2 <- lm(mpg ~ wt + hp, data = mtcars)
mid2 <- interpret(mpg ~ wt + hp, data = mtcars, model = fit2)

# Combine them into a "midlist" collection (which inherits from "mids")
mids <- midlist("wt" = mid1, "wt + hp" = mid2)

# Compare the main effect of 'wt' across both models
ggmid(mids, term = "wt")

# Compare the effect of 'wt' as a series plot across the models
ggmid(mids, term = "wt", type = "series")
```

---

interpret *Fit MID Models*

Description

interpret() is used to fit a Maximum Interpretation Decomposition (MID) model. MID models are additive, highly interpretable models composed of functions, each with up to two variables.

Usage

```
interpret(object, ...)

## Default S3 method:
interpret(
  object,
  x,
  y = NULL,
  weights = NULL,
  pred.fun = get.yhat,
  link = NULL,
  k = c(NA, NA),
```

```
  type = c(1L, 1L),
  interactions = FALSE,
  terms = NULL,
  singular.ok = FALSE,
  mode = 1L,
  method = NULL,
  lambda = 0,
  kappa = 1e+06,
  na.action = getOption("na.action"),
  verbosity = 1L,
  frames = list(),
  split = "quantile",
  digits = NULL,
  lump = "none",
  others = "others",
  sep = ">",
  max.nelements = 1000000000L,
  nil = 1e-07,
  tol = 1e-07,
  pred.args = list(),
  ...
)

## S3 method for class 'formula'
interpret(
  formula,
  data = NULL,
  model = NULL,
  pred.fun = get.yhat,
  weights = NULL,
  subset = NULL,
  na.action = getOption("na.action"),
  verbosity = 1L,
  mode = 1L,
  drop.unused.levels = FALSE,
  pred.args = list(),
  ...
)
```

Arguments

object	a fitted model object to be interpreted.
...	optional arguments. For interpret.formula(), arguments to be passed on to interpret.default(). For interpret.default(), ... can include convenient aliases (e.g., "ok" for singular.ok, "ie" for interactions) as well as several advanced fitting options (see the "Advanced Fitting Options" section for details).
x	a matrix or data.frame of predictor variables to be used in the fitting process.

	The response variable should not be included.
y	an optional vector or matrix of the model predictions or the response variables.
weights	a numeric vector of sample weights for each observation in x.
pred.fun	a function to obtain predictions from a fitted model, where the first argument is for the fitted model and the second argument is for new data. The default is get.yhat().
link	a character string specifying the link function. This can be one of the links from make.link() (e.g., "log", "logit", "probit", "cauchit"), one of the links from get.link() (e.g., "log1p", "robit", "scobit", "box-cox"), or a custom object containing at least the linkfun() and linkinv() functions.
k	an integer or a vector of two integers specifying the maximum number of sample points for main effects (k[1]) and interactions (k[2]). If a single integer is provided, it is used for main effects while the value for interactions is automatically determined. Any NA value will also trigger this automatic determination. With non-positive values, all unique data points are used as sample points.
type	a character string, an integer, or a vector of length two specifying the encoding type. Can be integer (1 for linear, 0 for step) or character ("linear", "constant"). If a vector is passed, type[1L] is used for main effects and type[2L] is used for interactions.
interactions	logical. If TRUE and if terms and formula are not supplied, all interactions for each pair of variables are modeled and calculated.
terms	a character vector of term labels or formula, specifying the set of component functions to be modeled. If not passed, terms includes all main effects, and all second-order interactions if interactions is TRUE.
singular.ok	logical. If FALSE, a singular fit is an error.
mode	an integer specifying the method of calculation. If mode is 1, the centering constraints are treated as penalties for the least squares problem. If mode is 2, the constraints are used to reduce the number of free parameters.
method	an integer or a character string specifying the algorithm to solve the core least squares problem. Built-in options include 0 or "qr" (column-pivoted QR), 1 or "unpivoted.qr", 2 or "llt" (LLT Cholesky), 3 or "ldlt" (LDLT Cholesky), 4 or "svd" (singular value decomposition), and 5 or "eigen" (eigenvalue-eigenvector decomposition). For multi-response targets (matrix y), the computation automatically utilizes base R equivalents or safely falls back to "qr" with stats::.lm.fit(). External custom solvers can also be injected by setting options(midr.solver.<method_name> = function(x, y) ...).
lambda	the penalty factor for pseudo smoothing. The default is 0.
kappa	the penalty factor for centering constraints. Used only when mode is 1. The default is 1e+6.
na.action	a function or character string specifying the method of NA handling. The default is "na.omit".
verbosity	the level of verbosity. 0: fatal, 1: warning (default), 2: info or 3: debug.

`frames`	a named list of encoding frames ("numeric.frame" or "factor.frame" objects). The encoding frames are used to encode the variable of the corresponding name. If the name begins with "|" or ":", the encoding frame is used only for main effects or interactions, respectively.
`split`	a character string specifying the splitting strategy for numeric variables: `"quantile"` or `"uniform"`.
`digits`	an integer. The rounding digits for encoding numeric variables. Used only when `type` is `1` or `"linear"`.
`lump`	a character string specifying the lumping strategy for factor variables: `"none"`, `"rank"`, `"order"`, or `"auto"`.
`others`	a character string specifying the others level.
`sep`	a character string used to separate levels when merging ordered factors or creating interaction terms.
`max.nelements`	an integer specifying the maximum number of elements of the design matrix. Defaults to `1e9`.
`nil`	a threshold for the intercept and coefficients to be treated as zero. The default is `1e-7`.
`tol`	a tolerance for the singular value decomposition. The default is `1e-7`.
`pred.args`	optional parameters other than the fitted model and new data to be passed to `pred.fun()`.
`formula`	a symbolic description of the MID model to be fit.
`data`	a data.frame, list or environment containing the variables in `formula`. If not found in data, the variables are taken from `environment(formula)`.
`model`	a fitted model object to be interpreted.
`subset`	an optional vector specifying a subset of observations to be used in the fitting process.
`drop.unused.levels`	logical. If `TRUE`, unused levels of factors will be dropped.

Details

Maximum Interpretation Decomposition (MID) is a functional decomposition framework designed to serve as a faithful surrogate for complex, black-box models. It deconstructs a target prediction function $f(\mathbf{X})$ into a set of highly interpretable components:

$$f(\mathbf{X}) = g_\emptyset + \sum_j g_j(X_j) + \sum_{j<k} g_{jk}(X_j, X_k) + g_D(\mathbf{X})$$

where $g_\emptyset$ is the intercept, $g_j(X_j)$ represents the main effect of feature j, $g_{jk}(X_j, X_k)$ represents the second-order interaction between features j and k, and $g_D(\mathbf{X})$ is the residual.

The components g_j and g_{jk} are modeled as a linear expansion of basis functions, resulting in piecewise linear or piecewise constant functions. The estimation is performed by minimizing a penalized squared residual objective over a representative dataset:

$$\text{minimize } \mathbf{E}[g_D(\mathbf{X})^2] + \lambda R(g; \mathbf{X})$$

where $\lambda \geq 0$ is a regularization parameter that controls the smoothness of the components by penalizing the second-order differences of adjacent coefficients (a discrete roughness penalty).

To ensure the uniqueness and identifiability of the decomposition, MID imposes the centering constraints: for any feature j, $\mathbf{E}[g_j(X_j)] = 0$; and for any feature pair (j, k), $\mathbf{E}[g_{jk}(X_j, X_k) \mid X_j = x_j] = 0$ for all x_j and $\mathbf{E}[g_{jk}(X_j, X_k) \mid X_k = x_k] = 0$ for all x_k. In cases where the least-squares solution is still not unique due to collinearity, an additional probability-weighted minimum-norm constraint is applied to the coefficients to ensure a stable and unique solution.

Value

`interpret()` returns an object of class "mid". This is a list with the following components:

`weights`	a numeric vector of the sample weights.
`call`	the matched call.
`terms`	the `terms.object` used.
`link`	a "link-glm" or "link-midr" object containing the link function.
`intercept`	the intercept.
`encoders`	a list of variable encoders.
`main.effects`	a list of data frames representing the main effects.
`interactions`	a list of data frames representing the interactions.
`ratio`	the ratio of the sum of squared error between the target model predictions and the fitted MID values, to the sum of squared deviations of the target model predictions.
`linear.predictors`	a numeric vector of the linear predictors.
`fitted.values`	a numeric vector of the fitted values.
`residuals`	a numeric vector of the working residuals.
`na.action`	information about the special handling of `NA`s.

If a matrix is provided for `y`, `interpret()` returns a "midrib" and "mids" object.

Advanced Fitting Options

The `...` argument can be used to pass several advanced fitting options:

fit.intercept logical. If `TRUE`, the intercept term is fitted as part of the least squares problem. If `FALSE` (default), it is calculated as the weighted mean of the response.

interpolation a character string specifying the method for interpolating inestimable coefficients (betas) that arise from sparse data regions. Can be "iterative" for an iterative smoothing process, "direct" for solving a linear system, or "none" to disable interpolation.

max.niterations an integer specifying the maximum number of iterations for the "iterative" interpolation method.

save.memory an integer (0, 1, or 2) specifying the memory-saving level. Higher values reduce memory usage at the cost of increased computation time.

weighted.norm logical. If `TRUE`, the columns of the design matrix are normalized by the square root of their weighted sum. This is required to ensure the minimum-norm least squares solution obtained by appropriate methods (i.e., `4` or `5`) of `fastLmPure()` is the minimum-norm solution in a *weighted* sense.

weighted.encoding logical. If `TRUE`, sample weights are used during the encoding process (e.g., for calculating quantiles to determine knots).

References

Asashiba R, Kozuma R, Iwasawa H (2025). "midr: Learning from Black-Box Models by Maximum Interpretation Decomposition." 2506.08338, https://arxiv.org/abs/2506.08338.

See Also

`print.mid`, `summary.mid`, `predict.mid`, `plot.mid`, `ggmid`, `mid.plots`, `mid.effect`, `mid.terms`, `mid.importance`, `mid.conditional`, `mid.breakdown`

Examples

```
# Fit a MID model as a surrogate for another model
data(cars, package = "datasets")
model <- lm(dist ~ I(speed^2) + speed, cars)
mid <- interpret(dist ~ speed, cars, model)
plot(mid, "speed", intercept = TRUE)
points(cars)

# Fit a MID model as a standalone predictive model
data(airquality, package = "datasets")
mid <- interpret(Ozone ~ .^2, data = airquality, lambda = .5)
plot(mid, "Wind")
plot(mid, "Temp")
plot(mid, "Wind:Temp", main.effects = TRUE)

data(Nile, package = "datasets")
nile <- data.frame(time = 1:length(Nile), flow = as.numeric(Nile))

# A flexible fit with many knots
mid <- interpret(flow ~ time, data = nile, k = 100L)
plot(mid, "time", intercept = TRUE, limits = c(600L, 1300L))
points(x = 1L:100L, y = Nile)

# A smoother fit with fewer knots
mid <- interpret(flow ~ time, data = nile, k = 10L)
plot(mid, "time", intercept = TRUE, limits = c(600L, 1300L))
points(x = 1L:100L, y = Nile)

# A pseudo-smoothed fit using a penalty
mid <- interpret(flow ~ time, data = nile, k = 100L, lambda = 100L)
plot(mid, "time", intercept = TRUE, limits = c(600L, 1300L))
```

```
points(x = 1L:100L, y = Nile)
```

`labels.midlist` *Label MID Objects*

Description

S3 methods to get or set the labels (names) of a "midrib" or "midlist" object.

Usage

```
## S3 method for class 'midlist'
labels(object, ...)

## S3 method for class 'midrib'
labels(object, ...)

labels(object) <- value

## S3 replacement method for class 'midlist'
labels(object) <- value

## S3 replacement method for class 'midrib'
labels(object) <- value
```

Arguments

`object`	a collection object of class "midlist" or "midrib".
`...`	optional parameters passed to other methods.
`value`	a character vector of the same length as the number of base objects in the collection object.

Details

While a "midlist" object is a standard R list containing only one of a single base class, a "midrib" object stores multiple MID models in an optimized struct-of-arrays format.

Because of the internal struct-of-arrays ("AsIs") structure, using `names()` on a "midrib" object returns internal component names (e.g., "intercept", "main.effects"). To safely access or modify the names of the models, always use `labels()` and `labels<-`.

Value

`labels()` returns a character vector of labels of the stored base objects.

`labels<-` returns the updated collection object with new labels.

See Also

midlist, extract.midlist

Examples

```
# Fit a multivariate linear model
fit <- lm(cbind(y1, y2, y3) ~ x1 + I(x1^2), data = anscombe)

# Interpret the linear models
collection <- interpret(cbind(y1, y2, y3) ~ x1, data = anscombe, model = fit)

# Check the default labels
labels(collection)

# Rename the models in the collection
labels(collection) <- letters[1L:3L]
labels(collection)

# Extract a single base "mid" object by its new name using [[
mid <- collection[["a"]]
class(mid)

# Subset the collection to keep only the first two models using [
sub <- collection[1:2]
class(sub) # Maintains the collection class (e.g., "mids"-"midrib")
```

---

mid.breakdown *Calculate MID Breakdown*

Description

`mid.breakdown()` calculates the contribution of each component function of a fitted MID model to a single prediction. It breaks down the total prediction into the effects of the intercept, main effects, and interactions.

Usage

```
mid.breakdown(object, data = NULL, row = NULL, sort = TRUE)
```

Arguments

`object`	a "mid" object.
`data`	a data frame containing one or more observations for which to calculate the MID breakdown. If not provided, data is automatically extracted based on the function call.
`row`	an optional numeric value or character string specifying the row of `data` to be used for the breakdown. If `NULL`, and the `data` contains two or more observations, only the first observation is used.
`sort`	logical. If `TRUE`, the output data frame is sorted by the absolute contribution of each effect.

Details

This function provides local interpretability for a specific observation by decomposing its prediction into the individual contributions of the MID components. For a target observation $\mathbf{x}$, the total prediction is represented as the sum of all estimated terms:

$$g(\mathbf{x}) = g_\emptyset + \sum_j g_j(x_j) + \sum_{j<k} g_{jk}(x_j, x_k)$$

The output data frame itemizes the numerical value of each main effect $g_j(x_j)$ and interaction effect $g_{jk}(x_j, x_k)$, along with the intercept $g_\emptyset$. This decomposition makes the model's decision for a single instance fully transparent and easy to attribute to specific features or their combinations.

Value

`mid.breakdown()` returns an object of class "midbrk". This is a list with the following components:

`breakdown`	a data frame containing the breakdown of the prediction.
`data`	the data frame containing the predictor variable values used for the prediction.
`intercept`	the intercept of the MID model.
`prediction`	the predicted value from the MID model.

For a "mids" collection object, `mid.breakdown()` returns a collection object of class "midbrks"-"midlist".

See Also

`interpret`, `plot.midbrk`, `ggmid.midbrk`

Examples

```
data(airquality, package = "datasets")
mid <- interpret(Ozone ~ .^2, data = airquality, lambda = 1)

# Calculate the breakdown for the first observation in the data
brk <- mid.breakdown(mid, data = airquality, row = 1)
print(brk)

# Calculate the breakdown for the third observation in the data
brk <- mid.breakdown(mid, data = airquality, row = 3)
print(brk)
```

`mid.conditional` *Calculate MID Conditional Expectation*

Description

`mid.conditional()` calculates the data required to draw Individual Conditional Expectation (ICE) curves from a fitted MID model. ICE curves visualize how a single observation's prediction changes as a specified variable's value varies, while all other variable are held constant.

Usage

```
mid.conditional(
  object,
  variable,
  data = NULL,
  resolution = 100L,
  max.nsamples = 500L,
  seed = NULL,
  type = c("response", "link"),
  keep.effects = TRUE
)
```

Arguments

`object`	a "mid" object.
`variable`	a character string or expression specifying the single predictor variable for which to calculate ICE curves.
`data`	a data frame containing the observations to be used for the ICE calculations. If not provided, data is automatically extracted based on the function call.
`resolution`	an integer specifying the number of evaluation points for the `variable`'s range.
`max.nsamples`	an integer specifying the maximum number of samples. If the number of observations exceeds this limit, the `data` is randomly sampled.
`seed`	an integer seed for random sampling. Default is `NULL`.
`type`	the type of prediction to return. "response" (default) for the original scale or "link" for the scale of the linear predictor.
`keep.effects`	logical. If `TRUE`, the effects of individual component functions are stored in the output object.

Details

This function generates Individual Conditional Expectation (ICE) data by evaluating the MID model over a range of values for a specific variable. For a given observation $\mathbf{x}_i$, the ICE value at $X_j = x'$ is computed by replacing the value $x_{i,j}$ with x' while keeping all other features $\mathbf{x}_{i,\setminus j}$ fixed:

$$f_{\text{ICE}}(x') = g(x', \mathbf{x}_{i,\setminus j})$$

The function creates a set of hypothetical observations across a grid of evaluation points for the specified `variable`. The resulting object can be plotted to visualize how the prediction changes for individuals as a specific feature varies, revealing both global trends and local departures (heterogeneity).

Value

`mid.conditional()` returns an object of class "midcon". This is a list with the following components:

`observed`	a data frame of the original observations used, along with their predictions.
`conditional`	a data frame of the hypothetical observations and their corresponding predictions.
`variable`	name of the target variable.
`values`	a vector of the sample points for the `variable` used in the ICE calculation.

For a "mids" collection object, `mid.conditional()` returns a collection object of class "midcons"-"midlist".

See Also

`interpret`, `plot.midcon`, `ggmid.midcon`

Examples

```
data(airquality, package = "datasets")
mid <- interpret(Ozone ~ .^2, data = airquality, lambda = 1)

# Calculate the ICE values for a fitted MID model
con <- mid.conditional(mid, variable = "Wind", data = airquality)
print(con)
```

`mid.effect` *Evaluate Single MID Component Functions*

Description

`mid.effect()` calculates the contribution of a single component function of a fitted MID model. It serves as a low-level helper function for making predictions or for direct analysis of a term's effect.

`mid.f()` is a convenient shorthand for `mid.effect()`.

Usage

```
mid.effect(object, term, x, y = NULL)

mid.f(object, term, x, y = NULL)
```

Arguments

object	a "mid" object or a collection of models ("mids").
term	a character string specifying the component function (term) to evaluate.
x	a vector of values for the first variable in the term. If a matrix or data frame is provided, values of the related variables are automatically extracted from it.
y	a vector of values for the second variable in an interaction term. Ignored if x is a data frame containing both variables.

Details

`mid.effect()` is a low-level function designed to calculate the contribution of a single component function. Unlike `predict.mid()`, which is designed to return total model predictions, `mid.effect()` is more flexible. It accepts vectors, as well as matrices or data frames, as input for `x` and `y`. If `x` is a data frame, the necessary columns are automatically extracted. This makes it particularly useful for visualizing a component's effect in combination with standard plotting functions, such as `graphics::curve()`.

For a main effect, the function evaluates the component function $f_j(x_j)$ for a vector of values x_j. For an interaction, it evaluates $f_{jk}(x_j, x_k)$ using vectors x_j and x_k.

Value

`mid.effect()` returns a numeric vector of the calculated term contributions, with the same length as `x`.

For a collection of models ("mids"), `mid.effect()` returns a numeric matrix where each column corresponds to a model.

See Also

`interpret`, `predict.mid`

Examples

```
data(airquality, package = "datasets")
mid <- interpret(Ozone ~ .^2, data = airquality, lambda = 1)

# Visualize the main effect of "Wind"
curve(mid.effect(mid, term = "Wind", x), from = 0, to = 25)

# Visualize the interaction of "Wind" and "Temp"
curve(mid.f(mid, "Wind:Temp", x, 50), 0, 25)
curve(mid.f(mid, "Wind:Temp", x, 60), 0, 25, add = TRUE, lty = 2)
curve(mid.f(mid, "Wind:Temp", x, 70), 0, 25, add = TRUE, lty = 3)
```

`mid.importance` *Calculate MID Importance*

Description

`mid.importance()` calculates the MID importance of a fitted MID model. This is a measure of feature importance that quantifies the average contribution of each component function across a dataset.

Usage

```
mid.importance(
  object,
  data = NULL,
  weights = NULL,
  sort = TRUE,
  measure = 1L,
  max.nsamples = 10000L,
  seed = NULL
)
```

Arguments

`object`	a "mid" object.
`data`	a data frame containing the observations to calculate the importance. If not provided, data is automatically extracted based on the function call.
`weights`	an optional numeric vector of sample weights.
`sort`	logical. If `TRUE`, the output data frame is sorted by importance in descending order.
`measure`	an integer specifying the measure of importance. Possible alternatives are `1` for the mean absolute effect, `2` for the root mean square effect, and `3` for the median absolute effect.
`max.nsamples`	an integer specifying the maximum number of samples to retain in the `predictions` component of the returned object. If the number of observations exceeds this value, a weighted random sample is taken.
`seed`	an integer seed for random sampling. Default is `NULL`.

Details

The MID importance of a component function g_S, where S represents a single feature $\{j\}$ or a feature pair $\{j, k\}$, is defined as the mean absolute effect on the predictions within the given data:

$$\mathbf{I}(g_S) = \frac{1}{n} \sum_{i=1}^{n} |g_S(\mathbf{x}_{i,S})|$$

Terms with higher importance values have a larger average impact on the model's overall predictions. Because all components (main effects and interactions) are measured on the same scale as the response variable, these values provide a direct and comparable measure of each term's contribution to the model.

Value

`mid.importance()` returns an object of class "midimp". This is a list containing the following components:

`importance`	a data frame with the calculated importance values, sorted by default.
`predictions`	the matrix of the fitted or predicted MID values. If the number of observations exceeds `max.nsamples`, this matrix contains a sampled subset.
`measure`	a character string describing the type of the importance measure used.

For a "mids" collection object, `mid.importance()` returns a collection object of class "midimps"-"midlist".

See Also

`interpret`, `plot.midimp`, `ggmid.midimp`

Examples

```
data(airquality, package = "datasets")
mid <- interpret(Ozone ~ .^2, data = airquality, lambda = 1)

# Calculate MID importance using median absolute contribution
imp <- mid.importance(mid, data = airquality)
print(imp)

# Calculate MID importance using root mean square contribution
imp <- mid.importance(mid, measure = 2)
print(imp)
```

`mid.plots` *Plot Multiple MID Component Functions*

Description

`mid.plots()` is a convenience function for applying `ggmid()` or `plot()` to multiple component functions of a "mid" object at once. It can automatically determine common plotting scales and manage the layout.

Usage

```
mid.plots(
  object,
  terms = mid.terms(object, interactions = FALSE),
  limits = c(NA, NA),
  intercept = FALSE,
  main.effects = FALSE,
  max.nplots = NULL,
  engine = c("ggplot2", "graphics"),
  ...
)
```

Arguments

object	a "mid" object.
terms	a character vector of the terms to be visualized. By default, only the main effect terms are used.
limits	a numeric vector of length two specifying the mid value limits. NA values are replaced by the minimum and/or maximum of the plotted MID values. If intercept = TRUE is set, the intercept is also included in the limit calculation.
intercept	logical. If TRUE, the intercept is added to the MID values and the plotting scale is shifted accordingly.
main.effects	logical. If TRUE, main effects are added to the interaction plots to show conditional effects. This argument disables automatic limit calculations.
max.nplots	the maximum number of plots to generate.
engine	the plotting engine to use, either "ggplot2" or "graphics".
...	optional parameters passed on to plot.mid() or ggmid().

Value

If engine is "ggplot2", mid.plots() returns a list of "ggplot" objects. Otherwise (i.e., if engine is "graphics"), mid.plots() produces plots as side-effects and returns NULL invisibly.

See Also

interpret, plot.mid, ggmid

Examples

```
data(diamonds, package = "ggplot2")
set.seed(42)
idx <- sample(nrow(diamonds), 1e4L)
mid <- interpret(price ~ (carat + cut + color + clarity) ^ 2, diamonds[idx, ])

# Plot selected main effects and interaction using the ggplot2 engine
mid.plots(mid, mid.terms(mid, require = "color", remove = "cut"), limits = NULL)
```

`mid.terms` *Extract Terms from MID Models*

Description

`mid.terms()` extracts term labels from a fitted MID model or derivative objects. Its primary strength is the ability to filter terms based on their type (main effects vs. interactions) or their associated variable names.

Usage

```
mid.terms(
  object,
  main.effects = TRUE,
  interactions = TRUE,
  require = NULL,
  remove = NULL,
  ...
)
```

Arguments

`object`	a "mid" object or another object that contains model terms. Can be a "mid.importance", "mid.conditional", or "mid.breakdown" object.
`main.effects`	logical. If `FALSE`, the main effect terms are excluded.
`interactions`	logical. If `FALSE`, the interactions terms are excluded.
`require`	a character vector of variable names. Only terms related to at least one of these variables are returned.
`remove`	a character vector of variable names. Terms related to any of these variables are excluded.
`...`	aliases are supported for convenience: "me" for `main.effects` and "ie" for `interactions`.

Details

A "term" in a MID model refers to either a main effect (e.g., "Wind") or an interaction effect (e.g., "Wind:Temp"). This function provides a flexible way to select a subset of these terms, which is useful for plotting, summarizing, or other downstream analyses.

Value

`mid.terms()` returns a character vector of the selected term labels.

Note

This function provides the common underlying logic for the `stats::terms()` S3 methods for "mid", "mid.importance", "mid.conditional", and "mid.breakdown" objects.

See Also

interpret

Examples

```
data(airquality, package = "datasets")
mid <- interpret(Ozone ~ .^2, airquality, lambda = 1)

# Get only main effect terms
mid.terms(mid, interactions = FALSE)

# Get terms related to "Wind" or "Temp"
mid.terms(mid, require = c("Wind", "Temp"))

# Get terms related to "Wind" or "Temp", but exclude any with "Day"
mid.terms(mid, require = c("Wind", "Temp"), remove = "Day")

# Get the predicted contributions of only the terms associated with "Wind"
terms_wind <- mid.terms(mid, require = "Wind")
predict(mid, airquality[1:3,], terms = terms_wind, type = "terms")
```

---

midlist *Combine MID Objects*

Description

Combines multiple MID models ("mid") or their interpretation results ("midimp", "midcon", "midbrk") into a unified collection object. This is useful for grouping models and their explanations together for seamless comparison, summary, and visualization.

Usage

```
midlist(...)

as.midlist(x)
```

Arguments

...	objects to be combined, possibly named. All inputs must inherit from exactly one of the supported base classes: "mid", "midimp", "midcon", or "midbrk". Collection classes (e.g., "mids"-"midrib", "midimps"-"midlist") are also accepted and will be flattened appropriately.
x	object to be coerced or tested.

Details

The `midlist()` function acts as a polymorphic constructor for collection objects. Depending on the class of the input objects, it automatically assigns the appropriate classes (e.g., "mid" objects become a "mids"-"midlist" collection; "midimp" objects become a "midimps"-"midlist" collection). All objects provided in `...` must belong to the same base class.

If a single "midrib" object is provided, it is returned as-is, preserving its optimized struct-of-arrays format. However, if a "midrib" object is combined with other objects via `...`, it is automatically coerced into a pure list (array of structures) to ensure structural consistency before concatenation.

Value

`midlist()` returns a list-based collection object inheriting from "midlist" and the appropriate collection class (e.g., "midcons"-"midlist"). If a single "midrib" object is provided, the original object is returned as-is.

`as.midlist()` returns a "midlist" object with a type-class "mids", "midimps", "midbrks", or "midcons".

See Also

`extract.midlist`, `labels.midlist`

Examples

```
# Fit models using the built-in anscombe dataset
fit1 <- lm(cbind(y1, y2, y3) ~ x1, data = anscombe)
fit2 <- lm(y4 ~ x4, data = anscombe)

# Create interpretation objects
# mid1 is a "midrib" collection containing 3 models
mid1 <- interpret(cbind(y1, y2, y3) ~ x1, data = anscombe, model = fit1)
class(mid1)
# mid2 is a single "mid" object
mid2 <- interpret(y4 ~ x4, data = anscombe, model = fit2)

# Combine a "midrib" and a "mid" into a single "midlist" collection.
collection <- midlist(mid1, y4 = mid2)

# Check the labels of the combined collection
labels(collection)

# The resulting object is a flat list of models
class(collection)
```

numeric.encoder *Encoder for Quantitative Variables*

Description

`numeric.encoder()` creates an encoder function for a quantitative variable. This encoder can then be used to convert a numeric vector into a design matrix using either piecewise linear or one-hot interval encoding, which are core components for modeling effects in a MID model.

`numeric.frame()` is a helper function to create a "numeric.frame" object that defines the encoding scheme.

Usage

```
numeric.encoder(
  x,
  k,
  type = c("linear", "constant", "null"),
  split = c("quantile", "uniform"),
  digits = NULL,
  weights = NULL,
  frame = NULL,
  tag = "x"
)

numeric.frame(
  reps = NULL,
  breaks = NULL,
  type = NULL,
  digits = NULL,
  tag = "x"
)
```

Arguments

`x`	a numeric vector to be encoded.
`k`	an integer specifying the coarseness of the encoding. If not positive, all unique values of `x` are used as knots or bins.
`type`	a character string or an integer specifying the encoding method: `"linear"` / `1` (default) or `"constant"` / `0`.
`split`	a character string specifying the splitting strategy: `"quantile"` (default) creates bins/knots based on data density; `"uniform"` creates equally spaced bins/knots over the data range.
`digits`	an integer specifying the rounding digits for the piecewise linear encoding (`type = "linear"`).
`weights`	an optional numeric vector of sample weights for `x`.

frame	a "numeric.frame" object or a numeric vector that explicitly defines the knots or breaks for the encoding.
tag	the name of the variable.
reps	a numeric vector to be used as the representative values (knots).
breaks	a numeric vector to be used as the binning breaks.

Details

The primary purpose of the encoder is to transform a single numeric variable into a design matrix for the MID model's linear system formulation. The output of the encoder depends on the type argument.

When type = 1, the variable's effect is modeled as a piecewise linear function with k knots including both ends. For each value, the encoder finds the two nearest knots and assigns a weight to each, based on its relative position. This results in a design matrix where each row has at most two non-zero values that sum to 1. This approach creates a smooth, continuous representation of the effect.

When type = 0, the variable's effect is modeled as a step function by dividing its range into k intervals (bins). The encoder determines which interval each value falls into and assigns a 1 to the corresponding column in the design matrix, with all other columns being 0. This results in a standard one-hot encoded matrix and creates a discrete, bin-based representation of the effect.

Value

numeric.encoder() returns an object of class "encoder". This is a list containing the following components:

frame	a "numeric.frame" object containing the encoding information.
n	the number of encoding levels (i.e., columns in the design matrix).
type	a character string describing the encoding type: "linear", "constant", or "null".
envir	an environment for the transform and encode functions.
transform	a function transform(x, ...) (identity by default).
encode	a function encode(x, ...) that converts a numeric vector into a dummy matrix.

numeric.frame() returns a "numeric.frame" object containing the encoding information.

See Also

factor.encoder

Examples

```
# Create an encoder for a quantitative variable
data(iris, package = "datasets")
enc <- numeric.encoder(x = iris$Sepal.Length, k = 5L, tag = "Sepal.Length")
enc

# Encode a numeric vector with NA and Inf
enc$encode(x = c(4:8, NA, Inf))
```

```
# Create an encoder with a pre-defined encoding frame
frm <- numeric.frame(breaks = c(3, 5, 7, 9), type = 0L)
enc <- numeric.encoder(x = iris$Sepal.Length, frame = frm)
enc$encode(x = c(4:8, NA, Inf))

# Create an encoder with a numeric vector specifying the knots
enc <- numeric.encoder(x = iris$Sepal.Length, frame = c(3, 5, 7, 9))
enc$encode(x = c(4:8, NA, Inf))
```

`plot.mid` *Plot MID Component Function*

Description

For "mid" objects (i.e., fitted MID models), `plot()` visualizes a single component function specified by the `term` argument.

Usage

```
## S3 method for class 'mid'
plot(
  x,
  term,
  type = c("effect", "data", "compound"),
  theme = NULL,
  intercept = FALSE,
  main.effects = FALSE,
  data = NULL,
  limits = NULL,
  jitter = NULL,
  resolution = c(100L, 100L),
  lumped = TRUE,
  ...
)
```

Arguments

`x`	a "mid" object to be visualized.
`term`	a character string specifying the component function to be plotted.
`type`	the plotting style. One of "effect", "data" or "compound".
`theme`	a character string or object defining the color theme. See `color.theme` for details.
`intercept`	logical. If `TRUE`, the intercept is added to the MID values.
`main.effects`	logical. If `TRUE`, main effects are included in the interaction plot.

data	a data frame to be plotted with the corresponding MID values. If not provided, data is automatically extracted from the function call.
limits	a numeric vector of length two specifying the limits of the plotting scale.
jitter	a numeric value specifying the amount of jitter for the data points.
resolution	an integer or vector of two integers specifying the resolution of the raster plot for interactions.
lumped	logical. If TRUE, uses the lumped factor levels; if FALSE, uses the original levels from the data. Always FALSE when main.effects = TRUE.
...	optional parameters to be passed to the graphing function. Possible arguments are "col", "fill", "pch", "cex", "lty", "lwd" and aliases of them.

Details

This is an S3 method for the plot() generic that produces a plot from a "mid" object, visualizing a component function of the fitted MID model.

The type argument controls the visualization style. The default, type = "effect", plots the component function itself. In this style, the plotting method is automatically selected based on the effect's type: a line plot for quantitative main effects; a bar plot for qualitative main effects; and a filled contour (level) plot for interactions. The type = "data" option creates a scatter plot of data, colored by the values of the component function. The type = "compound" option combines both approaches, plotting the component function alongside the data points.

Value

plot.mid() produces a plot as a side-effect and returns NULL invisibly.

See Also

interpret, ggmid

Examples

```
data(diamonds, package = "ggplot2")
set.seed(42)
idx <- sample(nrow(diamonds), 1e4)
mid <- interpret(price ~ (carat + cut + color + clarity)^2, diamonds[idx, ])

# Plot a quantitative main effect
plot(mid, "carat")

# Plot a qualitative main effect
plot(mid, "clarity")

# Plot an interaction effect with data points and a raster layer
plot(mid, "carat:clarity", type = "compound", data = diamonds[idx, ])

# Use a different color theme
plot(mid, "clarity:color", theme = "RdBu")
```

`plot.midbrk` *Plot MID Breakdown*

Description

For "midbrk" objects, `plot()` visualizes the breakdown of a prediction by component functions.

Usage

```
## S3 method for class 'midbrk'
plot(
  x,
  type = c("waterfall", "barplot", "dotchart"),
  theme = NULL,
  terms = NULL,
  max.nterms = 15L,
  vline = TRUE,
  others = "others",
  pattern = c("%t=%v", "%t:%t"),
  format.args = list(),
  ...
)
```

Arguments

`x`	a "midbrk" object to be visualized.
`type`	the plotting style. One of "waterfall", "barplot" or "dotchart".
`theme`	a character string or object defining the color theme. See `color.theme` for details.
`terms`	an optional character vector specifying which terms to display.
`max.nterms`	the maximum number of terms to display in the plot. Less important terms will be grouped into a "catchall" category.
`vline`	logical. If `TRUE`, a vertical line is drawn at the zero or intercept line.
`others`	a character string for the catchall label.
`pattern`	a character vector of length one or two specifying the format of the axis labels. The first element is used for main effects (default `"%t = %v"`), and the second is for interactions (default `"%t:%t"`). Use `"%t"` for the term name and `"%v"` for its value.
`format.args`	a named list of additional arguments passed to `format` for formatting the values. Common arguments include `digits`, `nsmall`, and `big.mark`.
`...`	optional parameters passed on to the graphing function. Possible arguments are "col", "fill", "pch", "cex", "lty", "lwd" and aliases of them.

Details

This is an S3 method for the `plot()` generic that produces a breakdown plot from a "midbrk" object, visualizing the contribution of each component function to a single prediction.

The `type` argument controls the visualization style. The default, `type = "waterfall"`, creates a waterfall plot that shows how the prediction builds from the intercept, with each term's contribution sequentially added or subtracted. The `type = "barplot"` option creates a standard bar plot where the length of each bar represents the magnitude of the term's contribution. The `type = "dotchart"` option creates a dot plot showing the contribution of each term as a point connected to a zero baseline.

Value

`plot.midbrk()` produces a plot as a side effect and returns `NULL` invisibly.

See Also

`mid.breakdown`, `ggmid.midbrk`

Examples

```
data(diamonds, package = "ggplot2")
set.seed(42)
idx <- sample(nrow(diamonds), 1e4)
mid <- interpret(price ~ (carat + cut + color + clarity)^2, diamonds[idx, ])
mbd <- mid.breakdown(mid, diamonds[1L, ])

# Create a waterfall plot
plot(mbd, type = "waterfall")

# Create a bar plot with a different theme
plot(mbd, type = "barplot", theme = "highlight")

# Create a dot chart
plot(mbd, type = "dotchart", size = 1.5)
```

---

plot.midbrks *Compare MID Breakdowns*

Description

For "midbrks" collection objects, `plot()` visualizes and compares the breakdown of a prediction by component functions across multiple models using base R graphics.

Usage

```
## S3 method for class 'midbrks'
plot(
  x,
  type = c("barplot", "dotchart", "series"),
  theme = NULL,
  terms = NULL,
  max.nterms = 15L,
  vline = TRUE,
  others = "others",
  pattern = c("%t=%v", "%t:%t"),
  format.args = list(),
  labels = NULL,
  ...
)
```

Arguments

x	a "midbrks" collection object to be visualized.
type	the plotting style. One of "barplot", "dotchart", or "series".
theme	a character string or object defining the color theme. See color.theme for details.
terms	an optional character vector specifying which terms to display. If NULL, terms are automatically extracted from the object.
max.nterms	the maximum number of terms to display. Defaults to 15.
vline	logical. If TRUE, a vertical line is drawn at the zero or intercept line.
others	a character string for the catchall label. Defaults to "others".
pattern	a character vector of length one or two specifying the format of the axis labels. The first element is used for main effects (default "%t = %v"), and the second is for interactions (default "%t:%t"). Use "%t" for the term name and "%v" for its value.
format.args	a named list of additional arguments passed to format for formatting the values. Common arguments include digits, nsmall, and big.mark.
labels	an optional numeric or character vector to specify the model labels. Defaults to the labels found in the object.
...	optional parameters passed on to the main layer (e.g., geom_col).

Details

This is an S3 method for the plot() generic that evaluates the component contributions to a single prediction and compares the results across all models in the collection.

The type argument controls the visualization style: The default, type = "barplot", creates a grouped bar plot where the bars for each term are placed side-by-side across the models. The type = "dotchart" option creates a grouped dot plot, offering a cleaner comparison across models. The type = "series" option plots the contribution trend over the models for each component term.

Value

`plot.midbrks()` produces a plot as a side effect and returns `NULL` invisibly.

See Also

`plot.midbrk`, `ggmid.midbrks`

Examples

```
data(mtcars, package = "datasets")

# Fit two different models for comparison
mid1 <- interpret(mpg ~ wt + hp + cyl, data = mtcars)
mid2 <- interpret(mpg ~ (wt + hp + cyl)^2, data = mtcars)

# Calculate importance for both models and combine them
brks <- midlist(
  "Main Effects" = mid.breakdown(mid1, data = mtcars[1, ]),
  "Interactions" = mid.breakdown(mid2, data = mtcars[1, ])
)

# Create a comparative grouped bar plot (default)
plot(brks)

# Create a comparative dot chart with a specific theme
plot(rev(brks), type = "dotchart", theme = "R4")

# Create a series plot to observe trends across models
plot(brks, type = "series")
```

---

plot.midcon *Plot MID Conditional Expectation*

Description

For "midcon" objects, `plot()` visualizes Individual Conditional Expectation (ICE) curves derived from a fitted MID model.

Usage

```
## S3 method for class 'midcon'
plot(
  x,
  type = c("iceplot", "centered"),
  theme = NULL,
  term = NULL,
  var.alpha = NULL,
  var.color = NULL,
```

```
  var.linetype = NULL,
  var.linewidth = NULL,
  reference = 1L,
  points = TRUE,
  sample = NULL,
  ...
)
```

Arguments

x	a "midcon" object to be visualized.
type	the plotting style. One of "iceplot" or "centered".
theme	a character string or object defining the color theme. See color.theme for details.
term	an optional character string specifying an interaction term. If passed, the ICE curve for the specified term is plotted.
var.alpha	a variable name or expression to map to the alpha aesthetic.
var.color	a variable name or expression to map to the color aesthetic.
var.linetype	a variable name or expression to map to the linetype aesthetic.
var.linewidth	a variable name or expression to map to the linewidth aesthetic.
reference	an integer specifying the index of the evaluation point to use as the reference for centering the c-ICE plot.
points	logical. If TRUE, points representing the actual predictions for each observation are plotted.
sample	an optional vector specifying the names of observations to be plotted.
...	optional parameters passed on to the graphing functions.

Details

This is an S3 method for the plot() generic that produces ICE curves from a "midcon" object. ICE plots are a model-agnostic tool for visualizing how a model's prediction for a single observation changes as one feature varies. This function plots one line for each observation in the data.

The type argument controls the visualization style: The default, type = "iceplot", plots the raw ICE curves. The type = "centered" option creates the centered ICE (c-ICE) plot, where each curve is shifted to start at zero, making it easier to compare the slopes of the curves.

The var.color, var.alpha, etc., arguments allow you to map aesthetics to other variables in your data using (possibly) unquoted expressions.

Value

plot.midcon() produces an ICE plot as a side-effect and returns NULL invisibly.

See Also

mid.conditional, ggmid.midcon

Examples

```
data(airquality, package = "datasets")
library(midr)
mid <- interpret(Ozone ~ .^2, data = airquality, lambda = 0.1)
ice <- mid.conditional(mid, "Temp", data = airquality)

# Create an ICE plot, coloring lines by 'Wind'
plot(ice, var.color = "Wind")

# Create a centered ICE plot, mapping color and linetype to other variables
plot(ice, type = "centered", theme = "Purple-Yellow",
     var.color = factor(Month), var.linetype = Wind > 10)
```

---

plot.midcons *Compare MID Conditional Expectations*

Description

For "midcons" collection objects, `plot()` visualizes and compares Individual Conditional Expectation (ICE) curves derived from multiple fitted MID models.

Usage

```
## S3 method for class 'midcons'
plot(
  x,
  type = c("iceplot", "centered", "series"),
  theme = NULL,
  var.alpha = NULL,
  var.linetype = NULL,
  var.linewidth = NULL,
  reference = 1L,
  sample = NULL,
  labels = NULL,
  ...
)
```

Arguments

`x`	a "midcons" collection object to be visualized.
`type`	the plotting style. One of "iceplot", "centered", or "series".
`theme`	a character string or object defining the color theme. See `color.theme` for details.
`var.alpha`	a variable name or expression to map to the alpha aesthetic.
`var.linetype`	a variable name or expression to map to the linetype aesthetic.
`var.linewidth`	a variable name or expression to map to the linewidth aesthetic.

reference	an integer specifying the index of the evaluation point to use as the reference for centering the c-ICE plot.
sample	an optional vector specifying the names of observations to be plotted.
labels	an optional numeric or character vector to specify the model labels. Defaults to the labels found in the object.
...	optional parameters passed on to the graphing functions (e.g., col, lty, lwd).

Details

This is an S3 method for the plot() generic that produces comparative ICE curves from a "midcons" object. It plots one line for each observation in the data per model.

For type = "iceplot" and "centered", lines are colored by the model label. For type = "series", lines are colored by the feature value and plotted across models.

The var.alpha, var.linetype, and var.linewidth arguments allow you to map aesthetics to other variables in your data using (possibly) unquoted expressions.

Value

plot.midcons() produces a plot as a side effect and returns NULL invisibly.

See Also

plot.midcon, ggmid.midcons

Examples

```
data(mtcars, package = "datasets")

# Fit two different models for comparison
mid1 <- interpret(mpg ~ wt + hp + cyl, data = mtcars)
mid2 <- interpret(mpg ~ (wt + hp + cyl)^2, data = mtcars)

# Calculate conditional expectations for both models
cons <- midlist(
  "Main Effects" = mid.conditional(mid1, "wt", data = mtcars[3:5, ]),
  "Interactions" = mid.conditional(mid2, "wt", data = mtcars[3:5, ])
)

# Create an ICE plot (default)
plot(cons)

# Create a centered-ICE plot
plot(cons, type = "centered")

# Create a series plot to observe trends across models
plot(cons, type = "series", var.linetype = ".id")
```

`plot.midimp` *Plot MID Importance*

Description

For "midimp" objects, `plot()` visualizes the importance of component functions of the fitted MID model.

Usage

```
## S3 method for class 'midimp'
plot(
  x,
  type = c("barplot", "dotchart", "heatmap", "boxplot"),
  theme = NULL,
  terms = NULL,
  max.nterms = 30L,
  ...
)
```

Arguments

`x`	a "midimp" object to be visualized.
`type`	the plotting style. One of "barplot", "dotchart", "heatmap", or "boxplot".
`theme`	a character string or object defining the color theme. See `color.theme` for details.
`terms`	an optional character vector specifying which terms to display.
`max.nterms`	the maximum number of terms to display. Defaults to 30 for bar, dot and box plots.
`...`	optional parameters passed on to the graphing functions. Possible arguments are "col", "fill", "pch", "cex", "lty", "lwd" and aliases of them.

Details

This is an S3 method for the `plot()` generic that produces an importance plot from a "midimp" object, visualizing the average contribution of component functions to the fitted MID model.

The `type` argument controls the visualization style. The default, `type = "barplot"`, creates a standard bar plot where the length of each bar represents the overall importance of the term. The `type = "dotchart"` option creates a dot plot, offering a clean alternative to the bar plot for visualizing term importance. The `type = "heatmap"` option creates a matrix-shaped heat map where the color of each cell represents the importance of the interaction between a pair of variables, or the main effect on the diagonal. The `type = "boxplot"` option creates a box plot where each box shows the distribution of a term's contributions across all observations, providing insight into the variability of each term's effect.

Value

`plot.midimp()` produces a plot as a side effect and returns `NULL` invisibly.

See Also

`mid.importance`, `ggmid.midimp`

Examples

```
data(diamonds, package = "ggplot2")
set.seed(42)
idx <- sample(nrow(diamonds), 1e4)
mid <- interpret(price ~ (carat + cut + color + clarity)^2, diamonds[idx, ])
imp <- mid.importance(mid)

# Create a bar plot (default)
plot(imp)

# Create a dot chart
plot(imp, type = "dotchart", theme = "Okabe-Ito", cex = 1.5)

# Create a heatmap
plot(imp, type = "heatmap")

# Create a boxplot to see the distribution of effects
plot(imp, type = "boxplot")
```

---

plot.midimps *Compare MID Importances*

Description

For "midimps" collection objects, `plot()` visualizes and compares the importance of component functions across multiple fitted MID models.

Usage

```
## S3 method for class 'midimps'
plot(
  x,
  type = c("barplot", "dotchart", "series"),
  theme = NULL,
  terms = NULL,
  max.nterms = 30L,
  labels = NULL,
  ...
)
```

Arguments

x	a "midimps" collection object to be visualized.
type	the plotting style. One of "barplot", "dotchart", or "series".
theme	a character string or object defining the color theme. See color.theme for details.
terms	an optional character vector specifying which terms to display. If NULL, terms are automatically extracted from the object.
max.nterms	the maximum number of terms to display. Defaults to 30.
labels	an optional numeric or character vector to specify the model labels. Defaults to the labels found in the object.
...	optional parameters passed on to the graphing functions. Possible arguments are "col", "fill", "pch", "cex", "lty", "lwd" and aliases of them.

Details

This is an S3 method for the plot() generic that creates a comparative importance plot from a "midimps" collection object. It visualizes the average contribution of component functions to the fitted MID models, allowing for easy comparison across different models.

The type argument controls the visualization style: The default, type = "barplot", creates a standard grouped bar plot where the length of each bar represents the overall importance of the term, positioned side-by-side by model label. The type = "dotchart" option creates a grouped dot plot, offering a clean alternative to the bar plot for visualizing and comparing term importance across models. The type = "series" option plots the importance trend over the models for each component function.

Value

plot.midimps() produces a plot as a side effect and returns NULL invisibly.

See Also

plot.midimp, ggmid.midimps

Examples

```
data(mtcars, package = "datasets")

# Fit two different models for comparison
mid1 <- interpret(mpg ~ wt + hp + cyl, data = mtcars)
mid2 <- interpret(mpg ~ (wt + hp + cyl)^2, data = mtcars)

# Calculate importance for both models and combine them
imps <- midlist(
  "Main Effects" = mid.importance(mid1),
  "Interactions" = mid.importance(mid2)
)

# Create a comparative grouped bar plot (default)
```

```
plot(imps)

# Create a comparative dot chart with a specific theme
plot(rev(imps), type = "dotchart", theme = "Okabe-Ito")

# Create a series plot to observe trends across models
plot(imps, type = "series")
```

`plot.mids` *Compare MID Component Functions*

Description

For "mids" collection objects, `plot()` visualizes and compares a single main effect across multiple models.

Usage

```
## S3 method for class 'mids'
plot(
  x,
  term,
  type = c("effect", "series"),
  theme = NULL,
  intercept = FALSE,
  limits = NULL,
  resolution = NULL,
  labels = base::labels(x),
  ...
)
```

Arguments

`x`	a "mids" collection object to be visualized.
`term`	a character string specifying the main effect to evaluate.
`type`	the plotting style: "effect" plots the effect curve per model, while "series" plots the effect trend over models per feature value.
`theme`	a character string or object defining the color theme. See `color.theme` for details.
`intercept`	logical. If `TRUE`, the model intercept is added to the component effect.
`limits`	a numeric vector of length two specifying the limits of the plotting scale.
`resolution`	an integer specifying the number of evaluation points for continuous variables.
`labels`	an optional numeric or character vector to specify the model labels. Defaults to `labels(object)`. The function attempts to parse these labels into numeric values where possible.
`...`	optional parameters passed to the main layer (e.g., `linewidth`, `alpha`).

Details

This is an S3 method for the `plot()` generic that evaluates the specified `term` over a grid of values and compares the results across all models in the collection.

The `type` argument controls the visualization style. The default, `type = "effect"`, plots the component functions of the specified `term` for each model individually. The `type = "series"` option transposes the view to plot the effect trend over the models for each feature value.

Note: Comparative plotting for interaction terms (2D surfaces) is not supported for collection objects.

Value

`plot.mids()` produces a plot as a side-effect and returns `NULL` invisibly.

See Also

`plot.mid`, `ggmid.mids`

Examples

```
# Use a lightweight dataset for fast execution
data(mtcars, package = "datasets")

# Fit two models with different complexities
fit1 <- lm(mpg ~ wt, data = mtcars)
mid1 <- interpret(mpg ~ wt, data = mtcars, model = fit1)
fit2 <- lm(mpg ~ wt + hp, data = mtcars)
mid2 <- interpret(mpg ~ wt + hp, data = mtcars, model = fit2)

# Combine them into a "midlist" collection (which inherits from "mids")
mids <- midlist("wt" = mid1, "wt + hp" = mid2)

# Compare the main effect of 'wt' across both models
plot(mids, term = "wt")

# Compare the effect of 'wt' as a series plot across the models
plot(mids, term = "wt", type = "series")
```

`predict.mid` *Predict Method for fitted MID Models*

Description

`predict()` methods for obtaining predictions from a fitted MID model ("mid") or a collection of MID models ("mids"). It can be used to predict on new data or to retrieve the fitted values from the original data.

Usage

```
## S3 method for class 'mid'
predict(
  object,
  newdata = NULL,
  na.action = "na.pass",
  type = c("response", "link", "terms"),
  terms = mid.terms(object),
  ...
)

## S3 method for class 'mids'
predict(object, ...)
```

Arguments

object	a fitted model object of class "mid", or a collection object ("mids") to be used for prediction.
newdata	a data frame of the new observations. If NULL, the original fitted values are extracted and returned.
na.action	a function or character string specifying what should happen when the data contain NA values.
type	the type of prediction required. One of "response", "link", or "terms".
terms	a character vector of term labels, specifying a subset of component functions to use for predictions.
...	further arguments passed to or from other methods.

Details

The type argument allows you to specify the scale of the prediction. By default (type = "response"), the function returns predictions on the original scale of the response variable. Alternatively, you can obtain predictions on the scale of the linear predictor by setting type = "link". For a detailed breakdown, setting type = "terms" returns a matrix where each column represents the contribution of a specific model term on the linear predictor scale.

The terms argument allows for predictions based on a subset of the model's component functions, excluding others.

Value

For a single "mid" object, predict.mid() returns a numeric vector if type is "response" or "link", or a numeric matrix if type = "terms".

For a collection ("mids"), predict.mids() returns a numeric matrix where each column corresponds to a model if type is "response" or "link", or a list of numeric matrices if type = "terms".

See Also

interpret, mid.effect, get.yhat

Examples

```
data(airquality, package = "datasets")
test <- 1:10
mid <- interpret(Ozone ~ .^2, airquality[-test, ], lambda = 1, link = "log")

# Predict on new data
predict(mid, airquality[test, ])

# Get predictions on the link scale
predict(mid, airquality[test, ], type = "link")

# Get the contributions of specific terms
predict(mid, airquality[test, ], terms = c("Temp", "Wind"), type = "terms")
```

---

print.mid *Print MID Models*

Description

`print()` methods for a fitted MID model ("mid") or a collection of models ("mids").

Usage

```
## S3 method for class 'mid'
print(x, digits = max(3L, getOption("digits") - 2L), main.effects = FALSE, ...)

## S3 method for class 'mids'
print(x, max.nmodels = 1L, ...)
```

Arguments

`x`	a "mid" or "mids" object to be printed.
`digits`	an integer specifying the number of significant digits for printing.
`main.effects`	logical. If `TRUE`, the MID values of each main effect are also printed (only applicable for single "mid" objects).
`...`	arguments to be passed to other methods.
`max.nmodels`	an integer specifying the maximum number of models to print for a "midlist" collection.

Details

By default, the `print()` method for "mid" objects provides a quick overview of the model structure by listing the number of main effect and interaction terms. If `main.effects = TRUE` is specified, the method will also print the contribution of each main effect at its sample points, providing a more detailed look at the model's components.

For a collection of models in the structure-of-array format ("midrib"), the method prints a summarized overview. For array-of-structures collections ("midlist"), it prints the first few models up to `max.nmodels`.

Value

`print.mid()` returns the original "mid" object invisibly.

`print.mids()` returns the original "mids" object invisibly.

See Also

`interpret`, `summary.mid`

Examples

```
data(cars, package = "datasets")
mid <- interpret(dist ~ speed, cars)

# Default print provides a concise summary
print(mid)

# Setting main.effects = TRUE prints the contributions of each main effect
print(mid, main.effects = TRUE)
```

`scale_color_theme` *Color Theme Scales for ggplot2 Graphics*

Description

`scale_color_theme()` and its family of functions provide a unified interface to apply custom color themes to the `colour` and `fill` aesthetics of "ggplot" objects.

Usage

```
scale_color_theme(
  theme,
  ...,
  discrete = NULL,
  middle = 0,
  aesthetics = "colour"
)

scale_colour_theme(
  theme,
  ...,
  discrete = NULL,
  middle = 0,
  aesthetics = "colour"
)

scale_fill_theme(theme, ..., discrete = NULL, middle = 0, aesthetics = "fill")
```

Arguments

theme	a color theme name (e.g., "Viridis"), a character vector of color names, or a palette/ramp function. See ?color.theme for more details.
...	optional arguments to be passed to ggplot2::continuous_scale() or ggplot2::discrete_scale().
discrete	logical. If TRUE, a discrete scale is used regardless of the theme type.
middle	a numeric value specifying the middle point for the diverging color themes.
aesthetics	the aesthetic to be scaled. Can be "colour", "color", or "fill".

Details

This function automatically determines the appropriate **ggplot2** scale based on the theme's type. If the theme is "qualitative", a discrete scale is used by default to assign distinct colors to categorical data. The discrete argument is automatically set to TRUE if not specified. If the theme is "sequential" or "diverging", a continuous scale is used by default. The "diverging" themes are handled by scales::rescale_mid() to correctly center the gradient around the middle value.

Value

scale_color_theme() returns a ggplot2 scale object (either a "ScaleContinuous" or "ScaleDiscrete" object) that can be added to a "ggplot" object.

See Also

color.theme

Examples

```
data(txhousing, package = "ggplot2")
cities <- c("Houston", "Fort Worth", "San Antonio", "Dallas", "Austin")
df <- subset(txhousing, city %in% cities)
d <- ggplot2::ggplot(data = df, ggplot2::aes(x = sales, y = median)) +
  ggplot2::geom_point(ggplot2::aes(colour = city))

# Plot with a qualitative theme
d + scale_color_theme("Set 1")

# Use a sequential theme as a discrete scale
d + scale_color_theme("SunsetDark", discrete = TRUE)

data(faithfuld, package = "ggplot2")
v <- ggplot2::ggplot(faithfuld) +
  ggplot2::geom_tile(ggplot2::aes(waiting, eruptions, fill = density))

# Plot with continuous themes
v + scale_fill_theme("Plasma")

# Use a diverging theme with a specified midpoint
v + scale_fill_theme("midr", middle = 0.017)
```

`set.color.theme` *Register Color Themes*

Description

`set.color.theme()` registers a custom color theme in the package's theme registry.

Usage

```
set.color.theme(
  kernel,
  kernel.args = list(),
  options = list(),
  name = "newtheme",
  source = "custom",
  type = NULL,
  env = color.theme.env()
)
```

Arguments

`kernel`	a color vector, a palette function, or a ramp function to be used as a color kernel. It can also be a character vector or a list (see the "Details" section). A "color.theme" object can also be passed.
`kernel.args`	a list of arguments to be passed to the color kernel.
`options`	a list of option values to control the color theme's behavior.
`name`	a character string for the color theme name.
`source`	a character string for the source name of the color theme.
`type`	a character string specifying the type of the color theme. One of "sequential", "diverging", or "qualitative".
`env`	an environment where the color themes are registered.

Details

This function takes a color vector, a color-generating function, or an existing "color.theme" object and registers it under a specified `name` and `source` (default is "custom/newtheme"). The registered color theme can then be easily retrieved using the "Theme Name Syntax" (see `help(color.theme)`).

To keep the registry environment size small, the `kernel` argument supports a form of lazy loading. To use this feature, provide a vector or list containing two character strings. The first is an R expression that returns a color kernel (e.g., "rainbow"), and the second is the namespace in which to evaluate the expression (e.g., "grDevices"). The expression is evaluated only when the color theme is loaded by `color.theme()`.

Value

`set.color.theme()` returns the metadata of the previous theme that was overwritten (or `NULL` if none existed) invisibly.

See Also

`color.theme`, `color.theme.info`

---

shapviz.mid *Calculate MID-Derived Shapley Values*

Description

`shapviz.mid()` is an S3 method for the `shapviz::shapviz()` generic, which calculates MID-derived Shapley values from a fitted MID model. This method is dynamically registered when the **shapviz** package is loaded.

Usage

```
## S3 method for class 'mid'
shapviz(object, data = NULL)
```

Arguments

`object`	a "mid" object.
`data`	a data frame containing the observations for which to calculate MID-derived Shapley values. If not passed, data is automatically extracted based on the function call.

Details

The function calculates MID-derived Shapley values by attributing the contribution of each component function to its respective variables as follows: first, each main effect is fully attributed to its corresponding variable; and then, each second-order interaction effect is split equally between the two variables involved.

Value

`shapviz.mid()` returns an object of class "shapviz".

summary.mid *Summarize MID Models*

Description

summary() methods for a fitted MID model ("mid") or a collection of models ("mids"). It prints a comprehensive summary of the model structure and fit quality.

Usage

```
## S3 method for class 'mid'
summary(
  object,
  diagnose = FALSE,
  digits = max(3L, getOption("digits") - 2L),
  ...
)

## S3 method for class 'mids'
summary(object, max.nmodels = 1L, ...)
```

Arguments

object	a "mid" or "mids" object to be summarized.
diagnose	logical. If TRUE, the diagnosis plot is displayed. Defaults to FALSE.
digits	the number of significant digits for printing numeric values.
...	arguments to be passed to graphics::panel.smooth() for the diagnosis plot.
max.nmodels	an integer specifying the maximum number of models to summarize for a "mids" collection.

Details

The S3 method summary.mid() generates a comprehensive overview of the fitted MID model. The output includes:

- **Call**: the function call used to fit the MID model.
- **Link**: name of the link function used to fit the MID model, if applicable.
- **Uninterpreted Variation Ratio**: proportion of target model variance not explained by MID model.
- **Residuals**: five-number summary of (working) residuals.
- **Encoding**: summary of encoding schemes per variable.
- **Diagnosis**: residuals vs fitted values plot (displayed only when diagnose = TRUE).

Value

summary.mid() returns the original "mid" object invisibly.

summary.mids() returns the original "mids" object invisibly.

See Also

interpret, print.mid

Examples

```
# Summarize a fitted MID model
data(cars, package = "datasets")
mid <- interpret(dist ~ speed, cars)
summary(mid)
```

---

theme_midr *Default Plotting Themes*

Description

theme_midr() returns a complete theme for "ggplot" objects, providing a consistent visual style for **ggplot2** plots.

par.midr() can be used to set graphical parameters for base R graphics.

Usage

```
theme_midr(
  grid_type = c("none", "x", "y", "xy"),
  base_size = 11,
  base_family = "serif",
  base_line_size = base_size/22,
  base_rect_size = base_size/22,
  ...
)

par.midr(...)
```

Arguments

grid_type	the type of grid lines to display, one of "none", "x", "y" or "xy".
base_size	base font size, given in pts.
base_family	base font family.
base_line_size	base size for line elements.
base_rect_size	base size for rect elements.
...	for theme_midr(), other parameters passed on to ggplot2::theme_light(). **ggplot2** >= 4.0.0 accepts ink, paper, and accent. For par.midr(), optional arguments in tag = value form to be passed to graphics::par().

Value

`theme_midr()` provides a **ggplot2** theme customized for the **midr** package.

`par.midr()` returns the previous values of the changed parameters in an invisible named list.

Examples

```
# Use theme_midr() with ggplot2
X <- data.frame(x = 1:10, y = 1:10)
ggplot2::ggplot(X) +
  ggplot2::geom_point(ggplot2::aes(x, y)) +
  theme_midr()
ggplot2::ggplot(X) +
  ggplot2::geom_col(ggplot2::aes(x, y)) +
  theme_midr(grid_type = "y")
ggplot2::ggplot(X) +
  ggplot2::geom_line(ggplot2::aes(x, y)) +
  theme_midr(grid_type = "xy")

# Use par.midr() for base R graphics
old.par <- par.midr()
plot(y ~ x, data = X)
par(old.par)
```

---

weighted.loss *Weighted Loss Function*

Description

`weighted.loss()` computes various loss metrics (e.g., RMSE, MAE) between two numeric vectors, or for the deviations from the weighted mean of a numeric vector.

Usage

```
weighted.loss(
  x,
  y = NULL,
  w = NULL,
  na.rm = FALSE,
  method = c("rmse", "mse", "mae", "medae", "r2")
)
```

Arguments

`x`	a numeric vector.
`y`	an optional numeric vector. If `NULL`, `x` is compared against its weighted mean.
`w`	a numeric vector of sample weights for each value in `x`.

na.rm	logical. If TRUE, any NA and NaNs are removed from all input vectors before the calculation.
method	the loss measure. One of "mse" (mean square error), "rmse" (root mean square error), mae" (mean absolute error), "medae" (median absolute error), or "r2" (R-squared).

Value

weighted.loss() returns a single numeric value.

Examples

```
# Calculate loss metrics between x and y with weights
weighted.loss(x = c(0, 10), y = c(0, 0), w = c(99, 1), method = "rmse")
weighted.loss(x = c(0, 10), y = c(0, 0), w = c(99, 1), method = "mae")
weighted.loss(x = c(0, 10), y = c(0, 0), w = c(99, 1), method = "medae")

# Verify uninterpreted variation ratio of a fitted MID model without weights
mid <- interpret(dist ~ speed, cars)
1 - weighted.loss(cars$dist, predict(mid, cars), method = "r2")
mid$ratio

# Verify uninterpreted variation ratio of a fitted MID model with weights
w <- 1:nrow(cars)
mid <- interpret(dist ~ speed, cars, weights = w)
1 - weighted.loss(cars$dist, predict(mid, cars), w = w, method = "r2")
mid$ratio
```

Index