Package 'adas.utils'

May 7, 2026

Title Design of Experiments and Factorial Plans Utilities

Version 1.4.0

Description
A number of functions to create and analyze factorial plans according to the Design of Experiments (DoE) approach, with the addition of some utility function to perform some statistical analyses. DoE approach follows the approach in ``Design and Analysis of Experiments" by Douglas C. Montgomery (2019, ISBN:978-1-119-49244-3). The package also provides utilities used in the course ``Analysis of Data and Statistics" at the University of Trento, Italy.

Depends R (>= 4.1.0)

License CC BY 4.0

Encoding UTF-8

RoxygenNote 7.3.3

Imports dplyr, gghalfnorm, ggplot2, glue, grDevices, lubridate, magrittr, purrr, readr, rlang, scales, stats, stringr, tibble, tidyr, utils, vctrs

LazyData true

Suggests knitr, rmarkdown, testthat (>= 3.0.0), tidyverse

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Paolo Bosetti [aut, cre]

Maintainer Paolo Bosetti <paolo.bosetti@unitn.it>

Repository CRAN

Date/Publication 2026-02-27 15:52:12 UTC

Contents

add_predictions *Add predictions to a data frame*

Description

Add predictions to a data frame

Usage

```
add_predictions(data, model, var = "pred", type = NULL, ...)

spread_predictions(data, ..., type = NULL)

gather_predictions(data, ..., .pred = "pred", .model = "model", type = NULL)
```

Arguments

data	A data frame used to generate the predictions.
model	add_predictions takes a single model;
var	The name of the output column, default value is pred
type	Prediction type, passed on to stats::predict(). Consult predict() documentation for given model to determine valid values.
...	gather_predictions and spread_predictions take multiple models. The name will be taken from either the argument name of the name of the model. add_predictions passes further arguments to the underlying predict generic method; this allows, for example, to also get confidence or prediction bands (when available).
.pred, .model	The variable names used by gather_predictions.

Value

A data frame. add_prediction adds a single new column, with default name pred, to the input data. spread_predictions adds one column for each model. gather_predictions adds two columns .model and .pred, and repeats the input rows for each model.

Examples

```
df <- tibble::tibble(
  x = sort(runif(100)),
  y = 5 * x + 0.5 * x ^ 2 + 3 + rnorm(length(x))
)
plot(df)

m1 <- lm(y ~ x, data = df)
grid <- data.frame(x = seq(0, 1, length = 10))
grid %>% add_predictions(m1)
```

```
# To also get confidence bands:
grid %>% add_predictions(m1, interval="confidence", level=0.99)

m2 <- lm(y ~ poly(x, 2), data = df)
grid %>% spread_predictions(m1, m2)
grid %>% gather_predictions(m1, m2)
```

`add_residuals` *Add residuals to a data frame*

Description

Add residuals to a data frame

Usage

```
add_residuals(data, model, var = "resid")

spread_residuals(data, ...)

gather_residuals(data, ..., .resid = "resid", .model = "model")
```

Arguments

`data`	A data frame used to generate the residuals
`model`, `var`	`add_residuals` takes a single `model`; the output column will be called `resid`
`...`	`gather_residuals` and `spread_residuals` take multiple models. The name will be taken from either the argument name of the name of the model.
`.resid`, `.model`	The variable names used by `gather_residuals`.

Value

A data frame. `add_residuals` adds a single new column, `.resid`, to the input `data`. `spread_residuals` adds one column for each model. `gather_predictions` adds two columns `.model` and `.resid`, and repeats the input rows for each model.

Examples

```
df <- tibble::tibble(
  x = sort(runif(100)),
  y = 5 * x + 0.5 * x ^ 2 + 3 + rnorm(length(x))
)
plot(df)

m1 <- lm(y ~ x, data = df)
df %>% add_residuals(m1)

m2 <- lm(y ~ poly(x, 2), data = df)
df %>% spread_residuals(m1, m2)
df %>% gather_residuals(m1, m2)
```

as_tibble.alias.matrix *Convert an alias matrix to a tibble*

Description

Given an alias matrix, this function returns a tidy tibble of the alias structures, with the added `generator` column containing the generator (i.e. right-hand side) of the defining relationship that generates each alias.

Usage

```
## S3 method for class 'alias.matrix'
as_tibble(x, ..., compact = TRUE)
```

Arguments

`x`	the alias matrix object.
`...`	additional arguments to `as_tibble`.
`compact`	a logical: if TRUE, it reports all possible effects combinations, even those with no alias.

Value

a tibble representation of the alias matrix

Examples

```
tibble::as_tibble(fp_alias_matrix(~A*B*C, ~B*C*D))
```

battery *Battery experiment data*

Description

Battery life in hour of a factorial experiment with 2 factors and 3 levels each. Factors are:

Usage

```
battery
```

Format

A data frame with 36 rows and 6 columns

Details

- `Temperature`: of the battery during the discharge experiment
- `Material`: Plate material for the battery

Other columns are:

- `StandardOrder`: Yate's standard order
- `RunOrder`: randomized order, in which tests have been executed
- `Repeat`: repeat number
- `Response`: battery life in hours

References

Douglas C. Montgomery, "Design and Analysis of Experiments", 8th edition, Wiley, 2019

---

ccd_experiment_yield *Central Composite Design Experiment Yields*

Description

Yield data for a two factor CCD experiment

Usage

```
ccd_experiment_yield
```

Format

A list with three vectors:

- `base`: the yield for a 2^2 factorial design, replicated 3 times
- `center`: the yield for the center points, replicated 4 times
- `axial`: the yield for the axial points, replicated 2 times

chauvenet *Chauvenet's criterion*

Description

Applies the Chauvenet's criterion to a sample, identifying a possible outlier.

Usage

```
chauvenet(x, threshold = 0.5)
```

Arguments

x	the sample vector.
threshold	the threshold for the frequency of the suspect outlier.

Value

an object of class `chauvenet` with the following components:

`sample` the name of the sample

`s0` the maximum difference

`index` the index of the suspect outlier

`value` the value of the suspect outlier

`expected` the expected frequency of the suspect outlier

`reject` a logical value indicating whether the suspect outlier should be rejected

Examples

```
x <- rnorm(100)
chauvenet(x)
chauvenet(x, threshold=0.1)
```

cotton *Cotton yarn experiment data*

Description

Yarn tensile strength in a completely randomized experiment with 5 different levels of cotton fiber.

Usage

```
cotton
```

Format

A data frame with 25 rows and 3 columns. Columns represent:

- `Run`: run order
- `Cotton`: cotton content in mass percentage
- `Strength`: yarn tensile strength in N

References

Douglas C. Montgomery, "Design and Analysis of Experiments", 8th edition, Wiley, 2019

`daniel_plot_hn` *Daniel's plot (half-normal)*

Description

Given a non-replicated model of a factorial plan, this function provides a half-normal plot of the effects of the model, labeling the main n effects.

Usage

```
daniel_plot_hn(model, label_n = 6, line.p = c(0, 0.4), ...)
```

Arguments

`model`	a linear model
`label_n`	plot the labels of the highest n values
`line.p`	vector of quantiles to use when fitting the Q-Q line, defaults defaults to 0.25 that corresponds to c(0.25, 0.75) on the full set of quantiles. If you pass a vector of two elements (first MUST be 0), then the line is fitted through the origin and up to 1 minus the second element quantile
`...`	further arguments to `gghalfnorm::gghalfnorm()`

Value

a half-normal plot (GGPlot2 object) with the effects of the model

See Also

`gghalfnorm::gghalfnorm()`

Examples

```
daniel_plot_hn(lm(Y~A*B*C*D, data=filtration))
```

daniel_plot_qq *Daniel's plot (quantile-quantile)*

Description

Given a non-replicated model of a factorial plan, this function provides a QQ plot of the effects of the model, labeling all the effects.

Usage

```
daniel_plot_qq(model, alpha = 0.5, xlim = c(-3, 3))
```

Arguments

model	a linear model
alpha	the transparency of the horizontal lines
xlim	the limits of the x-axis

Value

a QQ plot (GGPlot2 object) with the effects of the model

Examples

```
daniel_plot_qq(lm(Y~A*B*C*D, data=filtration))
```

examples_url *Examples URL*

Description

Provides the URL for the desired example data, so that it can be more easily downloaded.

Usage

```
examples_url(example)
```

Arguments

example	data file name

Value

the full URL for the desired example

Examples

```
examples_url("battery.dat") |> read.table(header=TRUE)
```

expand_formula *Expand a formula*

Description

Expand a formula

Usage

```
expand_formula(f)
```

Arguments

f	a formula

Value

a formula after expansion, e.g. `Y ~ A + B` becomes `Y ~ A + B + A:B`

Examples

```
expand_formula(Y ~ (A + B)^3)
```

filtration *Filtration data*

Description

Non-replicated factorial plan for a slurry filtration process.

Usage

```
filtration
```

Format

Factors are:

- A: Temperature
- B: Pressure
- C: Concentration of solid phase
- D: Agitation speed

The yield is in column Y and represents the filtration speed

fp_add_names *Add factor names to a design matrix*

Description

Store factor names in the `factorial.plan` object, as a list within the `factor.names` attribute.

Usage

```
fp_add_names(dm, ...)
```

Arguments

`dm`	the design matrix.
`...`	a set of factors to name, with their respective names, e.g. `A="Temperature", B="Pressure"`. If the factor is not in the design matrix factors list, a warning is printed and the factor is skipped.

Value

the design matrix with the named factors.

Examples

```
fp_design_matrix(3, rep=2) %>%
  fp_add_names(A="Temperature", B="Pressure")
```

fp_add_scale *Scale factors levels*

Description

This function allows to add columns to a design matrix with scaled factor, i.e. factors reported in real units rather in coded units (e.g. -1, 1).

Usage

```
fp_add_scale(dm, ..., suffix = "_s")
```

Arguments

`dm`	the design matrix to scale.
`...`	a set of factors to scale, with their respective ranges, e.g. `A=c(10, 30), B=c(0, 1)`.if the range is not a two-number vector or the factor is not numeric, a warning is printed and the factor is skipped.
`suffix`	the suffix to add to the scaled factor name in creating new columns. If the suffix is the empty string, factors are replaced.

Value

the design matrix with the scaled factors.

Examples

```
fp_design_matrix(3, rep=2) %>%
  fp_add_scale(A=c(10, 30), B=c(0, 1), suffix=".scaled")
```

`fp_alias` *Build an alias table from a formula*

Description

Given a Defining relationship or a number of factors, this function builds a matrix of aliases.

Usage

```
fp_alias(arg)
```

Arguments

`arg`	A formula or a number of factors.

Details

Defining relationships are represented as one side formulas, e.g. $I=ABC$ becomes `~A*B*C`.

Value

A matrix of logical values.

Examples

```
fp_alias(~A*B*C*D)
fp_alias(3)
```

fp_alias_list *List All Alias for a Fractional Factorial Plan*

Description

Given a defining relationship, as a one side firmula,, this function lists all the aliases for a fractional factorial plan.

Usage

```
fp_alias_list(arg)
```

Arguments

arg	A formula for the defining relationship, or the number of factors.

Details

Defining relationships are represented as one side formulas, e.g. $I=ABC$ becomes `~A*B*C`.

Value

a list of aliases (as formulas).

Examples

```
fp_alias_list(~A*B*C*D)
```

fp_alias_matrix *Build the alias matrix*

Description

Given a list of formulas (defining relationships), this function returns a matrix of all possible aliases.

Usage

```
fp_alias_matrix(...)
```

Arguments

...	one or more formulas, or a single list of formulas, or a fractional factorial plan.

Details

It is also possible to pass a fractional factorial plan, in which case the defining relationships will be extracted from it.

Value

a square matrix: each cell is `0` if there is no alias, or an integer representing the index of the generator that produced that alias in the list of generators.

See Also

`fp_fraction()`

Examples

```
# with formulas:
fp_alias_matrix(~A*B*C, ~B*C*D)

# with a fractional factorial plan:
fp_design_matrix(5) %>%
  fp_fraction(~A*B*C*D) %>%
  fp_fraction(~B*C*D*E) %>%
  fp_alias_matrix() %>%
  plot()
```

---

fp_all_drs *Return a list of all defining relationships*

Description

Given two or more independent refining relationships, represented as one side formulas,, this function returns a list of all possible defining relationships, including the dependent ones.

Usage

```
fp_all_drs(...)
```

Arguments

`...`	formulas, or a single list of formulas.

Details

Defining relationships are represented as one side formulas, e.g. $I=ABC$ becomes `~A*B*C`.

Value

a list of formulas.

Examples

```
fp_all_drs(~A*B*C, ~B*C*D)
```

fp_augment_axial *Augment to a central composite design*

Description

Adds the axial points to a 2^n centered factorial plan.

Usage

```
fp_augment_axial(dm, rep = 1)
```

Arguments

dm	A factorial plan table, with central points.
rep	The number of replications.

Value

A central composite design (a `factorial.plan` object).

Examples

```
fp_design_matrix(3) %>%
  fp_augment_center(rep=4) %>%
  fp_augment_axial()
```

fp_augment_center *Augment to a centered design*

Description

Add the central points to an existing 2^n factorial plan.

Usage

```
fp_augment_center(dm, rep = 5)
```

Arguments

dm	A factorial plan table.
rep	The number of replications.

Value

A central composite design (a `factorial.plan` object).

Examples

```
fp_design_matrix(3) %>%
  fp_augment_center()
```

`fp_defrel` *Factorial Plan Defining Relationship*

Description

Builds a formula from a number of factors

Usage

```
fp_defrel(arg)
```

Arguments

`arg`	If it is a formula, it is returned verbatim. If it is a number, a formula is built with the number of factors. If it is neither a formula nor a number, an error is thrown.

Value

A formula.

Examples

```
# Defining relationships with three factors
fp_defrel(3)

# Defining relationship I=ABC
fp_defrel(~A*B*C)
```

`fp_design_matrix` *Factorial Plan Design Matrix*

Description

Builds a design matrix from a one side formula or a number of factors.

Usage

```
fp_design_matrix(arg, rep = 1, levels = c(-1, 1))
```

Arguments

arg	Either a formula or a number of factors. If it is a formula, the factors are extracted from it. If it is a number, the factors are the first n capital letters.
rep	Number of replications.
levels	Levels of the factors.

Details

Defining relationships are represented as one side formulas, e.g. $I=ABC$ becomes ~A*B*C.

Value

A design matrix: a subclass of a tibble of class factorial.plan. The class has the following attributes:

def.rel The defining relationship (a formula).

generators The list of generators (formulas) if the factorial plan is fractional.

fraction The list of fractions (character vectors) if the factorial plan is fractional.

levels The levels of the factors (all equal), in coded units.

scales A list: for each factor, a vector of two values corresponding to the extreme values in coded units.

Examples

```
fp_design_matrix(3, rep=2, levels=c("-", "+"))
```

---

fp_effect_names *Factorial Plan effect names from a formula*

Description

Returns the effect names from a formula, according to Yates' convention.

Usage

```
fp_effect_names(arg)
```

Arguments

arg	A formula.

Details

Defining relationships are represented as one side formulas, e.g. $I=ABC$ becomes ~A*B*C.

Value

An ordered factor with the effect names.

Examples

```
fp_effect_names(~A*B*C)
```

`fp_fraction` *Reduce a Factorial Plan by 1/2 Fraction*

Description

Reduce a Factorial Plan by 1/2 Fraction

Usage

```
fp_fraction(dm, formula, remove = TRUE)
```

Arguments

`dm`	A factorial plan table.
`formula`	A formula for the defining relationship.
`remove`	A logical value indicating if the removed columns should be removed. This setting is sticky: if it is FALSE and you pipe the result of this function to another `fp_fraction()` call, the columns will be kept by default.

Value

A reduced factorial plan table (a `factorial.plan` object).

See Also

`fp_design_matrix()`

Examples

```
# build a 2^5-2 fractional factorial plan with defining relationships
#   I=ABCD and I=BCDE
fp_design_matrix(5) %>%
  fp_fraction(~A*B*C*D) %>%
  fp_fraction(~B*C*D*E)
```

fp_gen2alias *Given a generator, find the alias*

Description

Given a generator and an effect, this function returns the alias.

Usage

```
fp_gen2alias(generator, effect)
```

Arguments

generator	a generator, in the form of ABCD....
effect	an effect, in the form of BD....

Details

Generators and aliases are strings of capital letters.

Value

An effect (string).

Examples

```
fp_gen2alias("ABCD", "BD")
```

fp_info *Factorial plan info*

Description

Print information about the factorial plan.

Usage

```
fp_info(x, file = "", comment = "")
```

Arguments

x	the factorial plan.
file	the file to write the information to. Use console if empty.
comment	a comment mark to add before each line of the information.

Value

No return value, just prints the fp information.

Examples

```
fp_design_matrix(3, rep=2) %>%
  fp_info()
```

`fp_merge_drs` *Return a merged defining relationship*

Description

This function, given one or more independent refining relationships, returns the most complete relationship, i.e. that which includes all the factors.

Usage

```
fp_merge_drs(f1, ...)
```

Arguments

`f1`	a formula.
`...`	other formulas.

Details

Defining relationships are represented as one side formulas, e.g. $I=ABC$ becomes `~A*B*C`.

Value

a formula.

Examples

```
fp_merge_drs(~A*B*C, ~B*C*D)
```

fp_name *Get the full name for a factor*

Description

This is a utility function mostly for internal use.

Usage

```
fp_name(fp, fct)
```

Arguments

fp	The factorial plan object
fct	The standard name of the factor as a string (e.g. "A", or "B", etc.)

Value

A string with the factor full name, if available (must have been added with fp_add_names). If not, it returns fct ditto.

See Also

fp_add_names()

Examples

```
df <- fp_design_matrix(2) %>%
  fp_add_names(A="Temperature", B="Pressure")
fp_name(df, "A")
```

fp_read_csv *Load a design matrix from a CSV file*

Description

Load from a CSV file the design matrix that has previously been saved with fp_write_csv(). It is an error if the loaded data frame has different dimensions or column names than the original design matrix.

Usage

```
fp_read_csv(dm, file, type = c(1, 2), yield = "Y", comment = "#")
```

Arguments

`dm`	the design matrix.
`file`	the file to read the design matrix from.
`type`	the CSV version (1 or 2).
`yield`	the yield column name.
`comment`	the comment mark.

Details

Note that the design matrix is sorted by the `StdOrder` column after loading.

Value

the design matrix with the new values.

See Also

`fp_write_csv()`

`fp_scale` *Retrieve the scale of a factor*

Description

This is a utility function mostly for internal use.

Usage

```
fp_scale(fp, fct)
```

Arguments

`fp`	The factorial plan object
`fct`	The standard name of the factor as a string (e.g. `"A"`, or `"B"`, etc.)

Value

A vector representing the factor range in scaled units; scales must have been added with `fp_add_scale`. If scales are not available, the range in coded units is returned (i.e. `c(-1,1)`)

See Also

`fp_add_scale()`

Examples

```
df <- fp_design_matrix(2) %>%
  fp_add_names(A="Temperature", B="Pressure") %>%
  fp_add_scale(A=c(20,30), B=c(3, 7))
fp_scale(df, "A")
```

`fp_scaled_axis` *Create a secondary GGPlot2 axis with scaled units and factor names*

Description

This is useful when creating contour plots for the FP response surface or factors interaction plots. It allows to add secondary axes that use the scaled factor units rather than the coded units (in range -1, 1).

Usage

```
fp_scaled_axis(fp, fct)
```

Arguments

`fp`	The factorial plan object
`fct`	The standard name of the factor as a string (e.g. `"A"`, or `"B"`, etc.)

Value

A `ggplot2::sec_axis` object

See Also

`fp_name()` `fp_scale()`

Examples

```
library(tidyverse)
fp <- fp_design_matrix(2, rep=3) %>%
  fp_add_names(A="Temperature (°C)", B="Pressure (bar)") %>%
  fp_add_scale(A=c(20,30), B=c(2, 3))
fp$Y <- ccd_experiment_yield$base
fp.lm <- lm(Y~A*B, data=fp)

# Interaction plot:
fp %>%
  mutate(
    pred = predict(fp.lm),
    B=factor(B)
  ) %>%
  ggplot(aes(x=A, y=pred, color=B)) +
```

```
  geom_line() +
  scale_x_continuous(sec.axis = fp_scaled_axis(fp, "A"))+
  scale_color_discrete(labels = fp_scale(fp, "B")) +
  labs(color=fp_name(fp, "B"), y="Yield")

expand.grid(
  A = seq(-1, 1, length.out=100),
  B = seq(-1, 1, length.out=100)
) %>% {
  mutate(., Y=predict(fp.lm, newdata=.))
} %>%
  ggplot(aes(x=A, y=B, z=Y)) +
  geom_contour_filled() +
  scale_x_continuous(sec.axis = fp_scaled_axis(fp, "A")) +
  scale_y_continuous(sec.axis = fp_scaled_axis(fp, "B")) +
  labs(fill="Yield")
```

`fp_treatments` *Factorial Plan List of Treatments*

Description

Builds a list of treatments from a formula, or from a number of factors.

Usage

```
fp_treatments(arg)
```

Arguments

`arg`	Either a formula or a number of factors. If it is a formula, the factors are extracted from it. If it is a number, the factors are the first `n` capital letters.

Details

Defining relationships are represented as one side formulas, e.g. $I=ABC$ becomes `~A*B*C`.

Value

A list of treatments (character vector).

Examples

```
fp_treatments(3)
```

fp_write_csv *Save a design matrix to a CSV file*

Description

Writes the design matrix to a CSV file, with a timestamp and comment lines.

Usage

```
fp_write_csv(dm, file, comment = "# ", timestamp = TRUE, type = c(1, 2), ...)
```

Arguments

dm	the design matrix.
file	the file to write the design matrix to.
comment	a comment mark to add before each line of the information.
timestamp	whether to add a timestamp to the file.
type	the CSV version (1 or 2).
...	other parameters passed to write_csv().

Details

Note that the design matrix is saved in the same order of the RunOrder column, i.e. random.

Value

Invisibly return the design matrix, unchanged, for further piping.

geom_pareto_bars *Pareto bars*

Description

geom_pareto_bars() draws a Pareto-style bar layer: values are sorted by decreasing absolute y, bars are drawn with absolute heights, and a computed fill indicates the original sign ("positive" or "negative").

Usage

```
geom_pareto_bars(
  mapping = NULL,
  data = NULL,
  stat = "pareto_bars",
  position = "identity",
  ...,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

stat_pareto_bars(
  mapping = NULL,
  data = NULL,
  geom = "col",
  position = "identity",
  ...,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
stat	The statistical transformation to use on the layer.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter".

- For more information and other ways to specify the position, see the layer position documentation.

`...` Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the `position` argument, or aesthetics that are required can *not* be passed through `....` Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `....` This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`na.rm` If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.

`show.legend` logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted.

`inherit.aes` If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `annotation_borders()`.

`geom` The geometric object to use display the data.

`geom_pareto_line` *Pareto cumulative line*

Description

`geom_pareto_line()` computes cumulative totals from absolute values sorted by decreasing magnitude and draws the resulting path.

`geom_pareto_point()` computes cumulative totals from absolute values sorted by decreasing magnitude and draws the resulting points.

Usage

```
geom_pareto_line(
  mapping = NULL,
  data = NULL,
  stat = "pareto_line",
  position = "identity",
  ...,
  cumulative = c("raw", "proportion"),
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_pareto_point(
  mapping = NULL,
  data = NULL,
  stat = "pareto_line",
  position = "identity",
  ...,
  cumulative = c("raw", "proportion"),
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

stat_pareto_line(
  mapping = NULL,
  data = NULL,
  geom = "path",
  position = "identity",
  ...,
  cumulative = c("raw", "proportion"),
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created.

A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).

stat — The statistical transformation to use on the layer.

position — A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following:

- The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter".
- For more information and other ways to specify the position, see the layer position documentation.

... — Other arguments passed on to layer()'s params argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can *not* be passed through .... Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, colour = "red" or linewidth = 3. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the params. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a stat_*() function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is stat_density(geom = "area", outline.type = "both"). The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a geom_*() function, the ... argument can be used to pass on parameters to the stat part of the layer. An example of this is geom_area(stat = "density", adjust = 0.5). The stat's documentation lists which parameters it can accept.
- The key_glyph argument of layer() may also be passed on through .... This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

cumulative — "raw" for cumulative absolute sums; "proportion" for cumulative percentages in [0, 1].

na.rm — If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.

show.legend — logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.

inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. annotation_borders().
geom	The geometric object to use display the data.

---

geom_qqhn *Half-normal QQ points*

Description

geom_qqhn() draws a half-normal QQ plot using absolute sample values. Theoretical quantiles are from the half-normal distribution, i.e. qnorm((p + 1) / 2).

Usage

```
geom_qqhn(
  mapping = NULL,
  data = NULL,
  stat = "qqhn",
  position = "identity",
  ...,
  abs_sample = TRUE,
  label_n = 0,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

stat_qqhn(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  ...,
  abs_sample = TRUE,
  label_n = 0,
  labels_var = NULL,
  output = c("all", "points", "labels"),
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.

`data` The data to be displayed in this layer. There are three options:

If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.

A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.

A `function` will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).

`stat` The statistical transformation to use on the layer.

`position` A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The `position` argument accepts the following:

- The result of calling a position function, such as `position_jitter()`. This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`.
- For more information and other ways to specify the position, see the layer position documentation.

`...` Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the `position` argument, or aesthetics that are required can *not* be passed through `....` Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `....` This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`abs_sample` Should the sample be converted to absolute values. Defaults to `TRUE` for half-normal plots.

label_n	Number of largest points to label. These labeled points are removed from the point layer and drawn as text labels. Label text is taken from the labels aesthetic when mapped, otherwise from sample values.
na.rm	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. annotation_borders().
geom	The geometric object to use display the data.
labels_var	Internal parameter used to forward label-column mappings.
output	Internal parameter controlling whether stat_qqhn() returns all transformed rows, only point rows, or only labeled rows.

---

geom_qqhn_band *Half-normal QQ confidence band*

Description

geom_qqhn_band() adds a confidence envelope around the half-normal QQ reference line. The envelope is built from order-statistic probabilities mapped through half-normal quantiles and then transformed by the fitted QQHN line.

Usage

```
geom_qqhn_band(
  mapping = NULL,
  data = NULL,
  stat = "qqhn_band",
  position = "identity",
  ...,
  line.p = 0.25,
  abs_sample = TRUE,
  conf.level = 0.95,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

stat_qqhn_band(
  mapping = NULL,
```

```
  data = NULL,
  geom = "ribbon",
  position = "identity",
  ...,
  line.p = 0.25,
  abs_sample = TRUE,
  conf.level = 0.95,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
stat	The statistical transformation to use on the layer.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter". • For more information and other ways to specify the position, see the layer position documentation.
...	Other arguments passed on to layer()'s params argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can *not* be passed through .... Unknown arguments that are not part of the 4 categories below are ignored. • Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, colour = "red" or linewidth = 3. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the params. Please note that while passing unmapped aesthetics as vectors is

technically possible, the order and required length is not guaranteed to be parallel to the input data.

- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `....` This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`line.p` A single probability `p` in `(0, 0.5)` used with its symmetric counterpart `1 - p` to compute the reference line, or a 2-vector `c(0, p)` with `p` in `(0, 1)` to force the line through the origin.

`abs_sample` Should the sample be converted to absolute values. Defaults to `TRUE` for half-normal plots.

`conf.level` Confidence level for the envelope.

`na.rm` If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.

`show.legend` logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted.

`inherit.aes` If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `annotation_borders()`.

`geom` The geometric object to use display the data.

Details

Use `conf.level` to control envelope coverage (for example `0.95`); `alpha` only controls ribbon transparency.

`geom_qqhn_line` *Half-normal QQ reference line*

Description

`geom_qqhn_line()` adds a reference line for a half-normal QQ plot. It mirrors `geom_qq_line()` but uses half-normal theoretical quantiles.

Usage

```
geom_qqhn_line(
  mapping = NULL,
  data = NULL,
  stat = "qqhn_line",
  position = "identity",
  ...,
  line.p = 0.25,
  abs_sample = TRUE,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

stat_qqhn_line(
  mapping = NULL,
  data = NULL,
  geom = "path",
  position = "identity",
  ...,
  line.p = 0.25,
  abs_sample = TRUE,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
stat	The statistical transformation to use on the layer.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position.

- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`.
- For more information and other ways to specify the position, see the layer position documentation.

`...` Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the `position` argument, or aesthetics that are required can *not* be passed through `....` Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `....` This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`line.p` A single probability `p` in `(0, 0.5)` used with its symmetric counterpart `1 - p` to compute the reference line, or a 2-vector `c(0, p)` with `p` in `(0, 1)` to force the line through the origin.

`abs_sample` Should the sample be converted to absolute values. Defaults to `TRUE` for half-normal plots.

`na.rm` If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.

`show.legend` logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted.

`inherit.aes` If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `annotation_borders()`.

`geom` The geometric object to use display the data.

ggTukey *ggTukey*

Description

This is a generic function for plotting Tukey's HSD test via GGplot2.

Usage

```
ggTukey(obj, ...)
```

Arguments

obj	an object to plot: either a TukeyHSD object or a data frame
...	Other parameters passed to the specialized functions

Value

a GGPlot2 object

ggTukey.data.frame *ggTukey for data.frame*

Description

ggTukey for data.frame

Usage

```
## S3 method for class 'data.frame'
ggTukey(obj, formula, which = 1, splt = NULL, ...)
```

Arguments

obj	a data frame
formula	a formula to be used in the aov call
which	the index of the comparison. Used when the formula in the undelying aov call has more than one term.
splt	a formula to split the data frame
...	further parameters passed to TukeyHSD (e.g. conf.level)

Value

a GGPlot2 object

Examples

```
library(tidyverse)
battery %>%
  ggTukey(Response~Material, splt=~Temperature, conf.level=0.99)
```

---

ggTukey.TukeyHSD *ggTukey for TukeyHSD*

Description

Plot Tukey's HSD test via GGplot2.

Usage

```
## S3 method for class 'TukeyHSD'
ggTukey(obj, which = 1, ...)
```

Arguments

obj	a TukeyHSD object
which	the index of the comparison. Used when the formula in the undelying aov call has more than one term.
...	further parameters (currently unused)

Value

a GGPlot2 object

See Also

TukeyHSD() ggTukey.data.frame() ggTukey.TukeyHSD()]]

Examples

```
library(tidyverse)
cotton %>%
  aov(Strength~Cotton, data=.) %>%
  TukeyHSD() %>%
  ggTukey()
```

normplot *Normal probability plot*

Description

Normal probability plot

Usage

```
normplot(data, var, breaks = seq(0.1, 0.9, 0.1), linecolor = "red")
```

Arguments

data	a data frame
var	the variable to plot (data column)
breaks	the breaks for the y-axis
linecolor	the color of the normal probability line

Value

a normal probability plot (GGPlot2 object)

Examples

```
library(tibble)
df <- tibble(
  xn = rnorm(100, mean=20, sd=5),
  xu = runif(100, min=0, max=40)
)

df %>% normplot(xn)
df %>% normplot(xu)
```

pareto_chart *Pareto's chart*

Description

This is a generic function for Pareto's chart.

Usage

```
pareto_chart(obj, ...)
```

Arguments

obj	an object
...	further parameters to specialized functions

Value

a Pareto chart of the effects of the model

See Also

pareto_chart.data.frame() pareto_chart.lm()

Examples

```
# For a data frame:
library(tibble)
set.seed(1)
tibble(
  val=rnorm(10, sd=5),
  cat=LETTERS[1:length(val)]
  ) %>%
  pareto_chart(labels=cat, values=val)

# For a linear model:
pareto_chart(lm(Y~A*B*C*D, data=filtration))
```

---

pareto_chart.data.frame

Pareto's chart

Description

Create a Pareto chart for a data frame.

Usage

```
## S3 method for class 'data.frame'
pareto_chart(obj, labels, values, ...)
```

Arguments

obj	a data frame
labels	the column with the labels of the data frame
values	the column with the values of the data frame
...	further parameters (currently unused)

Value

a Pareto chart (GGPlot2 object) of the data frame

Invisibly returns a data frame with the absolute values of the data frame, their sign, and the cumulative value.

Examples

```
library(tibble)
set.seed(1)
tibble(
  val=rnorm(10, sd=5),
  cat=LETTERS[1:length(val)]
  ) %>%
  pareto_chart(labels=cat, values=val)
```

---

pareto_chart.lm *Pareto's chart*

Description

Creates a Pareto chart for the effects of a linear model.

Usage

```
## S3 method for class 'lm'
pareto_chart(obj, ...)
```

Arguments

obj	a linear model
...	further parameters (currently unused)

Value

a Pareto chart (GGPlot2 object) of the effects of the model

Invisibly returns a data frame with the absolute effects of the model, their sign, and the cumulative effect.

Examples

```
pareto_chart(lm(Y~A*B*C*D, data=filtration))
```

plot.alias.matrix *Plot the alias matrix*

Description

Produces a tile plot of the alias matrix.

Usage

```
## S3 method for class 'alias.matrix'
plot(x, ..., compact = TRUE)
```

Arguments

x	an alias matrix.
...	additional arguments to ggplot2::geom_tile().
compact	logical, if TRUE only positive aliases are shown, omitting empty rows and columns.

Value

a ggplot object.

Examples

```
fp_alias_matrix(~A*B*C, ~B*C*D) %>%
  plot()
```

scale_y_pareto *Pareto y-scale with cumulative percentage axis*

Description

scale_y_pareto() configures a primary y-axis for absolute values and a secondary right-side axis that maps cumulative totals to percentages.

Usage

```
scale_y_pareto(
  sum_abs = NULL,
  name = ggplot2::waiver(),
  percent_name = "Cumulative (%)",
  ...
)
```

Arguments

sum_abs	Total absolute value used for percentage conversion, typically sum(abs(y)) from the source data. When NULL (default), the secondary axis is scaled to the current primary-axis maximum.
name	Primary y-axis label.
percent_name	Secondary y-axis label.
...	Additional arguments passed to ggplot2::scale_y_continuous().

Details

Use this with geom_pareto_line(cumulative = "raw") so the cumulative line ends at 100% on the secondary axis.

Index