Package: modelfitter (via r-universe)

October 23, 2024

Title Bootstrapping models and model fits

Version 0.0.0.9031

Description This is a very early work in progress.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

Imports dplyr, emmeans, ggplot2, geepack, lmtest, logbin, magrittr, rlang, stats, broom, car, mice, performance, purrr, sandwich, scales, stringr, tibble, tidyr, tidyselect, digest, fs, utils, patchwork, ggstance, huxtable, systemfonts, rappdirs, survival, lifecycle, MASS, cli, elrm, forcats, glue

URL https://github.com/bristol-vaccine-centre/modelfitter

Repository https://bristol-vaccine-centre.r-universe.dev

RemoteUrl https://github.com/bristol-vaccine-centre/modelfitter

RemoteRef HEAD

RemoteSha 0b74981b564b4cde944fff9b14620dffe683a14b

Contents

`as_vars` *Reuse tidy-select syntax outside of a tidy-select function*

Description

Reuse tidy-select syntax outside of a tidy-select function

Usage

```
as_vars(tidyselect, data = NULL)
```

Arguments

`tidyselect`	a tidyselect syntax which will be evaluated in context by looking for a call in the call stack that includes a dataframe as the first argument
`data`	(optional) a specific dataframe with which to evaluate the tidyselect

Value

a list of symbols resulting from the evaluation of the tidyselect in the context of the current call stack (or a provided data frame)

bootstrap_provider *Provide a access to bootstrap resamples of a dataset*

Description

Provide a access to bootstrap resamples of a dataset

Usage

```
bootstrap_provider(data, max_n, formulae = ~.)
```

Arguments

data	the data frame
max_n	the maximum number of different bootstraps
formulae	a list of formulae with all the columns used

Value

a function that returns a dataset for inputs between 1:max_n

Examples

```
bp = iris %>% bootstrap_provider(10)
bp(1) %>% head()
tmp = bp(1)
tmp2 = bp(1)
identical(tmp,tmp2)
```

configure_model *Configure a single model fit using the* modelfitter *framework*

Description

Configure a single model fit using the modelfitter framework

Usage

```
configure_model(model_name, model_formula, dataset, model_function)
```

Arguments

model_name	a label for the model
model_formula	the model formula you want to fit
dataset	the dataset
model_function	the model function to fit, e.g. logistic_regression

Value

a configuration dataframe and data provider to execute the fit with

Examples

```
# Simple example
tmp = configure_model(
  "Iris model",
  I(Species == "versicolor") ~ .,
  iris,
  modelfitter::logistic_regression
)
tmp2 = tmp %>% execute_configuration()
summfit = tmp2 %>% summarise_fits()
```

---

configure_models *Configure a set of model fits from a specification of different combinations*

Description

Configure a set of model fits from a specification of different combinations

Usage

```
configure_models(
  model_formula,
  dataset,
  model_function,
  data_subset = data_subset_provider(default = TRUE),
  formula_update = formula_provider(default = . ~ .)
)
```

Arguments

model_formula	the model formula or formulae you want to fit as a formula_provider(...).
dataset	the dataset imputed or bootstrapped if need be as a bootstrap_provider(...) or imputation_provider(...)
model_function	the model functions to fit as a model_function_provider(...)
data_subset	the data subsets to fit as a data_subset_provider(...)
formula_update	and model formula updates to apply before fitting as a formula_provider(...).

Value

a configuration dataframe and data provider to run

Examples

```
form = ~ bmi + age + chl

fp = formula_provider(
  `Univariate` = modelfitter::univariate_from_multivariate(form),
  `Adj Model 1` = form,
  `Adj Model 2` = update(form, . ~ . - chl),
)

mfp = model_function_provider(
   `Logistic` = modelfitter::logistic_regression,
   `Poisson` = modelfitter::quasi_poisson
)

ip = imputation_provider(
    mice::nhanes2 %>% dplyr::mutate(death=TRUE), 10)

fup = formula_provider(
  hypertension = hyp ~ .,
  death = death ~ .
)

dsp = data_subset_provider(
  all = TRUE,
  hypertensives = hyp == "yes",
  nonhypertensives = hyp == "no"
)

tmp = configure_models(fp, ip, mfp, dsp, fup) %>% dplyr::glimpse()

# the number of models to fit will be this:
sum(tmp$n_boots)
```

`cox_model` *Fit standard cox model.*

Description

Fit standard cox model.

Usage

```
cox_model(data, formula, ...)
```

Arguments

`data`	a data frame
`formula`	a formula of the form `survival::Surv(time, event) ~ obs1 + obs2 + ...`
`...`	not used

Value

a model object

`data_subset_provider` *Provide a mechanism for subsetting data*

Description

A data subset may be used for a sensitivity analysis. This provides a mechanism to filter the data within the pipeline.

Usage

```
data_subset_provider(...)
```

Arguments

`...`	a set of named data filter criteria. In most cases you will want to include a `default = TRUE` option in this list. (this is the default value)

Value

a data subset provider which can fiter data based on a named set of subset criteria

Examples

```
dsp = data_subset_provider(one = TRUE, two = Species == "versicolor", three = Sepal.Width < 2.6)
dsp()
f = dsp("three")
f(iris)

dsp2 = data_subset_provider()
dsp2("default")(mtcars) %>% head(10)
```

`exact_logistic_regression`
Fit exact logistic regression

Description

Fit exact logistic regression

Usage

```
exact_logistic_regression(
  data,
  formula,
  positive = c("yes", "true", "present", "confirmed"),
  ...
)
```

Arguments

`data`	a data frame
`formula`	a formula of the form `binary_outcome ~ obs1 + obs2 + ...`
`positive`	test strings to interpret as true if outcome is not a factor or logical.
`...`	not used

Value

a model object

`execute_configuration` *Fit statistical models defined in a* `modelfitter` *configuration*

Description

Fit statistical models defined in a `modelfitter` configuration

Usage

```
execute_configuration(
  cfg,
  retain_fit = sum(cfg$n_boots) < 50,
  performance = TRUE,
  ...,
  cache = FALSE
)
```

Arguments

`cfg`	a 4 column dataframe with • model_name - minumum • model_fn • data_subset_fn • model_formula
`retain_fit`	keep the model fit object? If the config results in less than 50 models then the default of this is TRUE. Otherwise we summarise the models as we go saving memory.

`performance` calculate performance metrics for all the models. These will be summarised over bootstrap replicates. Setting this to false may be a good idea if you have lots of models.

`...` Arguments passed on to `model_labels`

`model` a model or list of models.

`label_fn` a function that allows a predictor label to be renamed. This should expect a vector and return a vector of the same length. Levels will be terms in the model function and may be column names, or combinations thereof

`subgroup_label_fn` a function that allows a subgroup label to be renamed. This should expect a vector and return a vector of the same length. The input to this function will be either a factor level name or a combination of them or whatever else the model decides to name it's coefficients.

`cache` cache model output

Value

a 5 column dataframe with additional nested fitted results in `model_fit`. This nested dataframe will have the columns:

- `boot` - an id for the data bootstrap or imputation;
- `fit` (optional) - the model fit itself;
- `coef` the model coefficients;
- `global_p` the global p values for this model (see global_p_value());
- `performance` (optional) the performance metrics for this model;

Examples

```
tmp = configure_model(
  "Iris", I(Species == "versicolor") ~ ., iris, logistic_regression)
tmp2 = tmp %>% execute_configuration()

tmp3  = configure_models(
   formula_provider(
      "<F" = I(color < "F") ~ cut + carat + clarity + price,
      "<H" = I(color < "H") ~ cut + carat + clarity + price
   ),
   bootstrap_provider(ggplot2::diamonds, max_n = 10),
   model_function_provider(
     "Log reg" = modelfitter::logistic_regression,
     "Poisson" = modelfitter::quasi_poisson
   )
)

tmp4 = tmp3 %>% execute_configuration(cache = TRUE)
```

format_ci *Format a confidence interval*

Description

Format a confidence interval

Usage

```
format_ci(
  median,
  lower,
  upper,
  is_reference_value = FALSE,
  fmt.ci = "%s [%s - %s]",
  na.ci = "-",
  fn = format_ratio,
  ...
)
```

Arguments

median	the median value
lower	the lower value
upper	the upper quantile
is_reference_value	is the value a reference value
fmt.ci	the layout of the 3 elements as a sprintf using %s for each element
na.ci	the value to show if the median ci is NA
fn	a function to format each number
...	Arguments passed on to format_ratio x a vector of numbers fmt.ratio a sprintf format string max.ratio a max ratio after which to display as e.g. >50 or <0.02 na.ratio a symbol in case the value is NA.

Value

a formatted CI string

Examples

```
format_ci(
  median = 2^(-5:5),
  lower = 2^(-5:5-1),
  upper = 2^(-5:5+1),
  fmt.ratio = "%1.3g"
)
```

`format_model` *Generate a table for a single model output*

Description

The `modelfitter` configuration, execution, summary, format pipeline is for the complex use cases with multiple models. This is the simple version when we have a single fitted model and we want to print it in a table. This is more or less what `gtsummary` does.

Usage

```
format_model(model_name, fit, statistic, ...)
```

Arguments

`model_name`	a title for the model.
`fit`	the model fit.
`statistic`	the type of stastic this model outputs (e.g. OR, HR, RR). This mostly defines the label in the table.
`...`	Arguments passed on to `format_summary` `summfit` A set of fitted, and summarised configured models `global.p` present the global p value (anova III) rather than the line by line values. `inv_link` the inverse of the link function employed in the models. This is almost always the inverse `exp(...)` unless we are dealing with a linear model. `col_header` a glue spec using columns from the summfit data table to label the model columns. `model_name` and `n_obs_summary` should be defined as a minimum. Other bits of metadata will be present if the table has been configured using `configure_models(...)` including `model_type_name`, `data_subset_name`, `model_base_name`, `model_update_name`, `n_boots`. `row_design` a glue spec for presenting the statistic. valid columns are `reference` - the referent status, `group.type`, `beta.lower`, `beta.median`, `beta.upper`, `value.lower`, `value.median`, `value.upper`, `p.value.mixture`, `global.p.mixture`, `global.p.method`. The helper functions `format_ratio(x,fmt.ratio = "%1.3g")` and `format_ci(med,low,hi,ref)` may be useful in this glue string

`p_format` a function (or lambda) converting a number into a p-value string

`font_size` (optional) the font size for the table in points

`font` (optional) the font family for the table (which will be matched to closest on your system)

`footer_text` any text that needs to be added at the end of the table, setting this to FALSE dsables the whole footer (as does `options("tableone.hide_footer"=TRUE)`).

`summarise_fn` in the event that we want to present multiple models in the same column of a table it is possible that there are multiple entries for each variable. This function will combine them (at a text level) so they can be placed in a table. Examples could be `dplyr::first` or `~ paste0(.x,collapse="\n")`

Value

a huxtable formatted model summary table

Examples

```
coxfit = cox_model(survival::flchain, survival::Surv(futime, death) ~ sex * age)
format_model("FLChain", coxfit, "HR")
```

`format_ratio` *Format a ratio, truncating at a set level above and below 1 on log scale.*

Description

Format a ratio, truncating at a set level above and below 1 on log scale.

Usage

```
format_ratio(x, fmt.ratio = "%1.2f", max.ratio = 50, na.ratio = "Unk")
```

Arguments

`x`	a vector of numbers
`fmt.ratio`	a sprintf format string
`max.ratio`	a max ratio after which to display as e.g. `>50` or `<0.02`
`na.ratio`	a symbol in case the value is `NA`.

Value

a string of formatted ratios

Examples

```
format_ratio(2^(-6:6))
```

format_summary *Format a summary of multiple fits into a table.*

Description

Format a summary of multiple fits into a table.

Usage

```
format_summary(
  summfit,
  ...,
  statistic = "OR",
  global.p = getOption("modelfitter.global_p_values", TRUE),
  inv_link = exp,
  col_header = "{model_name} (N={sprintf('%d',max(n_obs_summary))})",
  row_design =
   "{format_ci(value.median,value.lower,value.upper,reference,fmt.ratio = '%1.2g')}",
  p_format = NULL,
  font_size = getOption("modelfitter.font_size", 8),
  font = getOption("modelfitter.font", "Arial"),
  footer_text = NULL,
  summarise_fn = NULL
)
```

Arguments

summfit	A set of fitted, and summarised configured models
...	Arguments passed on to model_labels
	model a model or list of models.
	label_fn a function that allows a predictor label to be renamed. This should expect a vector and return a vector of the same length. Levels will be terms in the model function and may be column names, or combinations thereof
	subgroup_label_fn a function that allows a subgroup label to be renamed. This should expect a vector and return a vector of the same length. The input to this function will be either a factor level name or a combination of them or whatever else the model decides to name it's coefficients.
statistic	what model output is this table presenting? Is it an OR, a RR or a HR? or something else?
global.p	present the global p value (anova III) rather than the line by line values.
inv_link	the inverse of the link function employed in the models. This is almost always the inverse exp(...) unless we are dealing with a linear model.
col_header	a glue spec using columns from the summfit data table to label the model columns. model_name and n_obs_summary should be defined as a minimum. Other bits of metadata will be present if the table has been configured using configure_models(...)

	including model_type_name, data_subset_name, model_base_name, model_update_name, n_boots.
row_design	a glue spec for presenting the statistic. valid columns are reference - the referent status, group.type, beta.lower, beta.median, beta.upper, value.lower, value.median, value.upper, p.value.mixture, global.p.mixture, global.p.method. The helper functions format_ratio(x,fmt.ratio = "%1.3g") and format_ci(med,low,hi,ref) may be useful in this glue string
p_format	a function (or lambda) converting a number into a p-value string
font_size	(optional) the font size for the table in points
font	(optional) the font family for the table (which will be matched to closest on your system)
footer_text	any text that needs to be added at the end of the table, setting this to FALSE dsables the whole footer (as does options("tableone.hide_footer"=TRUE)).
summarise_fn	in the event that we want to present multiple models in the same column of a table it is possible that there are multiple entries for each variable. This function will combine them (at a text level) so they can be placed in a table. Examples could be dplyr::first or ~ paste0(.x,collapse="\n")

Value

a huxtable tabular output of the model(s)

Examples

```
cfg  = configure_models(
   formula_provider(
      "<F" = I(color < "F") ~ cut + carat + clarity + price,
      "<H" = I(color < "H") ~ cut + carat + clarity + price
   ),
   bootstrap_provider(ggplot2::diamonds, max_n = 10),
   model_function_provider(
     "Log reg" = modelfitter::logistic_regression,
     "Poisson" = modelfitter::quasi_poisson
   )
)

exectn = cfg %>% execute_configuration(cache = TRUE)
summfit = exectn %>% summarise_fits()
hux = summfit %>% format_summary()
hux
```

---

formula_provider *Construct a configuration for multiple formulae*

Description

multiple formulae for a parameterised model fitting pipeline. Different models may be a key part of the analysis or a sensitivity. A single name maybe associated with one or many models - This is because it is frequent that we want to group together models such as univariate comparisons into the same format table as another single multivarible model. Another use case for multiple models is a single set of predictors

Usage

```
formula_provider(...)
```

Arguments

...	a named list of formulae or formulae lists. In the simplest use case this is a single formula, see examples for more possibilities.

Examples

```
form = is_coloured ~ cut + carat + clarity * price
fp = formula_provider(
  `Univariate` = modelfitter::univariate_from_multivariate(form),
  `Adj Model 1` = form,
  `Adj Model 2` = update(form, . ~ . - clarity * price),
)


predictors = ~ x + y + z
fp2 = formula_provider(
  `Death` = death ~ .,
  `ICU admission` = icu ~ .,
  `Delayed discharge` = delayed_discharge ~ .,
)
sapply(fp2(), fp2) %>% sapply(update, predictors)

# the simplest configuration
fp3 = formula_provider(outcome ~ x + y + z)
fp3()

fp4 = formula_provider(outcome1 ~ x + y + z, outcome2 ~ x + y + z)
fp4()
try(fp4("blah blah"))

# must define at least one forumula
try(formula_provider())

formula_provider(default = . ~ .)
```

f_lhs_vars *Get all used variables on LHS of a formula*

Description

Get all used variables on LHS of a formula

Usage

```
f_lhs_vars(formula)
```

Arguments

formula	a formula or list of formulae

Value

the variables on the RHS of the formulae as a character vector

Examples

```
f_lhs_vars(survival::Surv(time,event) ~ x + z + y)
```

f_rhs_vars *Get all used variables on LHS of a formula*

Description

Get all used variables on LHS of a formula

Usage

```
f_rhs_vars(formula)
```

Arguments

formula	a formula or list of formulae

Value

the variables on the LHS of the formulae as a character vector

Examples

```
f_rhs_vars(survival::Surv(time,event) ~ x + z +y)
f_rhs_vars(survival::Surv(time,event) ~ pspline(x) + z + y)
```

`global_p_value` *Calculate a global P-value for a model fit for each of the predictors.*

Description

In the summary output of a model there is a p-value for each level of the factors whereas usually we want to know what the overall contribution of the individual predictor to the model is.

Usage

```
global_p_value(x, ...)
```

Arguments

`x`	a model
`...`	not used.

Details

This function usually calculates a Type 2 Anova for the predictors within the models but sometimes falls back to different methods if the model is not supported by the `car::Anova` or `stats::anova` packages. The method used is reported in the output. This is designed as an automated process and may not work in all situations, or produce appropriate output for all model types. Interaction terms may be particularly problematic.

Value

a dataframe of the predictors of the model (but not the levels) in one column and a global p value in another. The global.p will be given as zero or one despite this being not really possible. It si assumed that it will be formatted to truncate very low or very high values.

Examples

```
diamonds2 = ggplot2::diamonds %>% dplyr::mutate(
   is_coloured = color <= "F",
   dplyr::across(dplyr::where(is.ordered), ~ factor(.x,ordered=FALSE))
) %>% dplyr::select(-color)

lmodel = stats::lm(Petal.Width ~ ., iris)
glmodel = logistic_regression(diamonds2, is_coloured ~ cut + carat + clarity * price)
coxfit = cox_model(survival::flchain, survival::Surv(futime, death) ~ sex * age)

# Error conditions
try(anova(stats::lm( cut ~ carat + clarity * price, diamonds2)))
global_p_value(stats::lm( cut ~ carat + clarity * price, diamonds2))


# cox model
global_p_value(coxfit)
```

```
# linear model
global_p_value(lmodel)

# logistic regression (OR)
global_p_value(
  stats::glm(is_coloured ~ cut + carat + clarity + price, diamonds2,  family="binomial"))
global_p_value(
  logistic_regression(diamonds2, is_coloured ~ cut + carat + clarity * price))

# poisson regression (RR)
global_p_value(
  quasi_poisson(diamonds2, is_coloured ~ cut + carat + clarity * price))
global_p_value(
  robust_poisson(diamonds2, is_coloured ~ cut + carat + clarity * price))
global_p_value(
  robust_poisson_2(diamonds2, is_coloured ~ cut + carat + clarity * price))

# log binomial regression (RR)
# TODO: this does not work at the moment as an example as the log_binomial starting value is
# a problem
global_p_value(
  log_binomial(diamonds2, is_coloured ~ cut + carat + clarity + price))

# global_p_value(
#  log_binomial_2(diamonds2, is_coloured ~ cut + carat + clarity + price))
```

`imputation_provider` *Provide access to missing data imputations*

Description

Provide access to missing data imputations

Usage

```
imputation_provider(data, max_n, formulae = ~., ...)
```

Arguments

`data`	the data frame with missing rows
`max_n`	the maximum number of different imputations to
`formulae`	a lit of formulae with all the columns used (defaults to everything)
`...`	cache control

Value

a function that returns a dataset for inputs between `1:max_n`

Examples

```
ip = imputation_provider(mice::nhanes2, 10, list(~ hyp + bmi, ~ age + chl))
ip(1) %>% head(10)
```

`logistic_regression` *Fit standard logistic regression with unordered factors.*

Description

This makes an effort to cast the result column into a logical from a factor or other data type. It also will convert ordered predictors into unordered before running.

Usage

```
logistic_regression(
  data,
  formula,
  positive = c("yes", "true", "present", "confirmed"),
  ...
)
```

Arguments

`data`	a data frame
`formula`	a formula of the form `binary_outcome ~ obs1 + obs2 + ...`
`positive`	test strings to interpret as true if outcome is not a factor or logical.
`...`	not used

Value

a model object

Examples

```
diamonds3 = ggplot2::diamonds %>% dplyr::mutate(
  is_coloured = color <= "F",
  cut = factor(cut,ordered=FALSE),
  price_cat = tableone::cut_integer(price, c(500,1000,2000,4000))
) %>% dplyr::select(-color)

model5 = diamonds3 %>%
  logistic_regression(is_coloured ~ cut + carat + clarity * price)
model6 = ggplot2::diamonds %>%
  logistic_regression(I(color <= "F") ~ cut + carat + clarity * price)

# this wont work as the factors are converted to unordered
```

```
# model6 = ggplot2::diamonds %>%
#   logistic_regression(I(color <= "F") ~ I(cut<"Very Good") + carat + clarity * price)
```

`logistic_regression_ordered` *Fit standard logistic regression.*

Description

This makes an effort to cast the result column into a logical from a factor or other data type. It also will model ordered factors as contrasts rather than as a polynomial expansion, allowing an interpretable output from ordered factors.

Usage

```
logistic_regression_ordered(
  data,
  formula,
  positive = c("yes", "true", "present", "confirmed"),
  ...
)
```

Arguments

`data`	a data frame
`formula`	a formula of the form `binary_outcome ~ obs1 + obs2 + ...`
`positive`	test strings to interpret as true if outcome is not a factor or logical.
`...`	not used

Value

a model object

Examples

```
diamonds3 = ggplot2::diamonds %>% dplyr::mutate(
  is_coloured = color <= "F",
  cut = factor(cut,ordered=FALSE),
  price_cat = tableone::cut_integer(price, c(500,1000,2000,4000))
) %>% dplyr::select(-color)

model5 = diamonds3 %>%
  logistic_regression_ordered(is_coloured ~ cut + carat + clarity * price)
```

log_binomial *Fit standard log binomial regression using* stats::glm

Description

The log binomial model using standard glm is less that satisfactory and hard to make converge. The algorithm needs starting values to have any hope and these do make a difference to the outcome. This needs more investigation before being used.

Usage

```
log_binomial(
  data,
  formula,
  positive = c("yes", "true", "present", "confirmed"),
  ...
)
```

Arguments

data	a data frame
formula	a formula of the form binary_outcome ~ obs1 + obs2 + ...
positive	test strings to interpret as true if outcome is not a factor or logical.
...	not used

Value

a model object

Examples

```
diamonds3 = ggplot2::diamonds %>% dplyr::mutate(
  is_coloured = color <= "F",
  cut = factor(cut,ordered=FALSE),
  price_cat = tableone::cut_integer(price, c(500,1000,2000,4000))
) %>% dplyr::select(-color)

model5 = diamonds3 %>% log_binomial(is_coloured ~ cut + carat + clarity * price)
global_p_value(model5)
```

log_binomial_2 *Fit standard log binomial regression using* logbin::logbin

Description

This can be very slow for simple models, and will not handle interaction terms It does not seem to report std error. This is very much a work in progress,

Usage

```
log_binomial_2(
  data,
  formula,
  positive = c("yes", "true", "present", "confirmed"),
  ...
)
```

Arguments

data	a data frame
formula	a formula of the form binary_outcome ~ obs1 + obs2 + ...
positive	test strings to interpret as true if outcome is not a factor or logical.
...	not used

Value

a model object

Examples

```
diamonds3 = ggplot2::diamonds %>% dplyr::mutate(
  is_coloured = color <= "F",
  cut = factor(cut,ordered=FALSE),
  price_cat = tableone::cut_integer(price, c(500,1000,2000,4000))
) %>% dplyr::select(-color)

model5 = diamonds3 %>% log_binomial_2(is_coloured ~ cut + carat + clarity + price)
global_p_value(model5)
```

model_function_provider *Construct a configuration for multiple statistical models*

Description

multiple statistical models for a parameterised model fitting pipeline.

Usage

```
model_function_provider(...)
```

Arguments

`...`	a named list of functions. In the simplest use case this is a single function, see examples for more possibilities. the functions must accept `data` as first parameter and `formula` as second.

Examples

```
# in the simplest version the name is pulled from the input
mfp = model_function_provider(logistic_regression)
mfp
mfp()

mfp = model_function_provider(logistic_regression, quasi_poisson)
mfp()

mfp2 = model_function_provider(
   Logistic = modelfitter::logistic_regression,
   Poisson = modelfitter::quasi_poisson
)
mfp2()

# the functions are named the other way round to normal model functions
# this was by design to fit into a tidy pipeline:
mfp = model_function_provider(linear = ~ stats::lm(.y,.x))
linear_model = mfp("linear")
iris %>% linear_model(Petal.Length ~ Petal.Width)
```

model_labels *Creates a table or plot row template for a set of models*

Description

This produces a dataframe which can be used to arrange the coefficients and p-values from one statistical model, or set of models. The rows are a superset of all the coefficients of the models and is designed to be used to left_join the outputs of `broom::tidy` (by `term`) or `global_p_value` (by `predictor`) to construct a tabular output.

Usage

```
model_labels(model, label_fn, subgroup_label_fn, ...)
```

Arguments

model	a model or list of models.
label_fn	a function that allows a predictor label to be renamed. This should expect a vector and return a vector of the same length. Levels will be terms in the model function and may be column names, or combinations thereof
subgroup_label_fn	a function that allows a subgroup label to be renamed. This should expect a vector and return a vector of the same length. The input to this function will be either a factor level name or a combination of them or whatever else the model decides to name it's coefficients.
...	not used

Details

It is expect that use cases such as multiple univariate models + a few fully adjusted models are passed to this function and the result is to be displayed in a single plot or figure.

Value

a data frame with `model.order`, `group.order`, `subgroup.order`, `characteristic`, `subgroup`, dplaye columns and `predictor` and `term` key columns to link to the output of `broom::tidy` or `global_p_value` (i.e. Anova II/III outputs)

Examples

```
diamonds3 = ggplot2::diamonds %>% dplyr::mutate(
  is_coloured = color <= "F",
  cut = factor(cut,ordered=FALSE),
  price_cat = tableone::cut_integer(price, c(500,1000,2000,4000))
) %>% dplyr::select(-color)

model = stats::glm(is_coloured ~ cut + carat + clarity * price, diamonds3, family="binomial")
model_labels(model, toupper, tolower)


model2 = logistic_regression(diamonds3, is_coloured ~ I(cut=="Good") + carat + clarity * price)
model_labels(model2, toupper, tolower)

model3 = stats::glm(is_coloured ~ carat + cut * clarity +  price, diamonds3, family="binomial")
model_labels(model3, toupper, tolower)


model4 = stats::glm(
  is_coloured ~ cut + carat + clarity + price,
  diamonds3,
  family="binomial",
```

```
  contrasts=list(clarity=MASS::contr.sdif))

coef(model4)
model_labels(model4, toupper, tolower)

# tmp = .ordered_contrasts(diamonds3, )
# model5 = stats::glm(
#   is_coloured ~ cut + carat + clarity * price,
#   diamonds3,
#   family="binomial", contrasts=tmp)
#
# model_labels(model5, toupper, tolower)

model6 = stats::glm(
  is_coloured ~ cut + carat + clarity + price_cat,
  diamonds3,
  family="binomial",
  contrasts=list(clarity=MASS::contr.sdif, price_cat=MASS::contr.sdif)
)

model_labels(model6, toupper, tolower)

model7 = stats::glm(
  is_coloured ~ cut + carat + clarity + splines::ns(price,df=2),
  diamonds3,
  family="binomial",
  contrasts=list(clarity=MASS::contr.sdif))

model_labels(model7, toupper)
```

`pmixnorm` *The cumulative density function of a mixture of normal distributions*

Description

The cumulative density function of a mixture of normal distributions

Usage

```
pmixnorm(q, means, sds, weights = rep(1, length(means)), na.rm = FALSE)
```

Arguments

`q`	vector of quantiles.
`means`	a vector of normal distribution means
`sds`	a vector of normal distribution sds
`weights`	a vector of weights
`na.rm`	remove distributions which have NA for mean or sd

Value

the pdf of the mixture distribution.

Examples

```
pmixnorm(q=c(2,20), means=c(10,13,14), sds=c(1,1,2), weights=c(2,2,3))
```

`qmixnorm` *A quantile function for a mixture of normal distributions*

Description

A quantile function for a mixture of normal distributions

Usage

```
qmixnorm(p, means, sds, weights = rep(1, length(means)), na.rm = FALSE)
```

Arguments

`p`	vector of probabilities.
`means`	a vector of normal distribution means
`sds`	a vector of normal distribution sds
`weights`	a vector of weights
`na.rm`	remove distributions with NA values for mean or sd

Value

the value of the yth quantile

Examples

```
qmixnorm(p=c(0.025,0.5,0.975), means=c(10,13,14), sds=c(1,1,2))
```

quasi_poisson *Fit standard poisson regression*

Description

This is roughly equivalent to a log_binomial model but converges well. Its output can be interpreted at a RR. It is using a stats::glm with a log link function and a family of quasipoisson. The binary outcome is interpreted as a count where true is 1 and false is 0.

Usage

```
quasi_poisson(
  data,
  formula,
  positive = c("yes", "true", "present", "confirmed"),
  ...
)
```

Arguments

data	a data frame
formula	a formula of the form binary_outcome ~ obs1 + obs2 + ...
positive	test strings to interpret as true if outcome is not a factor or logical.
...	not used

Value

a model object

Examples

```
diamonds3 = ggplot2::diamonds %>% dplyr::mutate(
  is_coloured = color <= "F",
  cut = factor(cut,ordered=FALSE),
  price_cat = tableone::cut_integer(price, c(500,1000,2000,4000))
) %>% dplyr::select(-color)

model5 = diamonds3 %>% quasi_poisson(is_coloured ~ cut + carat + clarity + price)
global_p_value(model5)
```

quasi_poisson_ordered *Fit standard poisson regression with ordered contrasts*

Description

This is roughly equivalent to a log_binomial model but converges well. Its output can be interpreted at a RR. It is using a stats::glm with a log link function and a family of quasipoisson. The binary outcome is interpreted as a count where true is 1 and false is 0. Ordered variables are represented as contrasts between adjacent levels

Usage

```
quasi_poisson_ordered(
  data,
  formula,
  positive = c("yes", "true", "present", "confirmed"),
  ...
)
```

Arguments

data	a data frame
formula	a formula of the form binary_outcome ~ obs1 + obs2 + ...
positive	test strings to interpret as true if outcome is not a factor or logical.
...	not used

Value

a model object

Examples

```
diamonds3 = ggplot2::diamonds %>% dplyr::mutate(
  is_coloured = color <= "F",
  cut = factor(cut,ordered=FALSE),
  price_cat = tableone::cut_integer(price, c(500,1000,2000,4000))
) %>% dplyr::select(-color)

model5 = diamonds3 %>% quasi_poisson_ordered(is_coloured ~ cut + carat + clarity + price)
summary(model5)
```

raw_data_provider *Provide a access to a dataset*

Description

Provide a access to a dataset

Usage

```
raw_data_provider(data, formulae = ~.)
```

Arguments

data	the data frame
formulae	a list of formulae with all the columns used

Value

a function that returns a dataset

robust_poisson *Fit robust poisson regression using sandwich estimators*

Description

This is roughly equivalent to a log_binomial model but converges well. Its output can be interpreted at a RR. It is using a glm with a log link function and a family of poisson with robust sandwich estimators. The binary outcome is interpreted as a count where true is 1 and false is 0.

Usage

```
robust_poisson(
  data,
  formula,
  positive = c("yes", "true", "present", "confirmed"),
  ...
)
```

Arguments

data	a data frame
formula	a formula of the form binary_outcome ~ obs1 + obs2 + ...
positive	test strings to interpret as true if outcome is not a factor or logical.
...	not used

Details

heteroskedasticity-consistent (HC) standard errors using sandwich: https://data.library.virginia.edu/understanding-robust-standard-errors/

Value

a model object

Examples

```
diamonds3 = ggplot2::diamonds %>% dplyr::mutate(
  is_coloured = color <= "F",
  cut = factor(cut,ordered=FALSE),
  price_cat = tableone::cut_integer(price, c(500,1000,2000,4000))
) %>% dplyr::select(-color)

model5 = diamonds3 %>% robust_poisson(is_coloured ~ cut + carat + clarity + price)
global_p_value(model5)
```

---

robust_poisson_2 *Fit robust poisson regression using* geepack::glm

---

Description

equivalent to a log_binomial model

Usage

```
robust_poisson_2(
  data,
  formula,
  positive = c("yes", "true", "present", "confirmed"),
  ...
)
```

Arguments

data	a data frame
formula	a formula of the form binary_outcome ~ obs1 + obs2 + ...
positive	test strings to interpret as true if outcome is not a factor or logical.
...	not used

Value

a model object

Examples

```
diamonds3 = ggplot2::diamonds %>% dplyr::mutate(
  is_coloured = color <= "F",
  cut = factor(cut,ordered=FALSE),
  price_cat = tableone::cut_integer(price, c(500,1000,2000,4000))
) %>% dplyr::select(-color)

model5 = diamonds3 %>% robust_poisson_2(is_coloured ~ cut + carat + clarity + price)
global_p_value(model5)
```

---

spline_term_plot *Spline term marginal effects plot*

Description

Spline term marginal effects plot

Usage

```
spline_term_plot(
  coxmodel,
  var_name,
  xlab = var_name,
  max_y = NULL,
  n_breaks = 7
)
```

Arguments

coxmodel	an output of a coxph model
var_name	a variable that is involved in a spline term
xlab	x axis label
max_y	maximium hazard ratio to display on y axis. Inferred from the central estimates if missing, which will most likely cut off confidence intervals
n_breaks	The number of divisions on the y axis

Value

a ggplot

sprintf_list *Sprintf with a list input*

Description

A variant of sprintf that work well with inputs that are in the format of a list. Good examples of which are the quantile functions

Usage

```
sprintf_list(format, params, na.replace = "-")
```

Arguments

format	the format string
params	the inputs as a list (rather than as a set of individual numbers)
na.replace	a value to replace NA values with.

Value

the formatted string

Examples

```
# generate a mixture confidence interval from a set of distributions
sprintf_list("%1.2f [%1.2f\u2013%1.2f]",
 qmixnorm(p=c(0.5,0.025,0.975),
 means=c(10,13,14), sds=c(1,1,2)))
```

summarise_fits *Summarise a modelfitter execution*

Description

A `modelfitter` execution may consist of multiple models, multiple data sets, and multiple bootstraps of data or imputations of data. This summarises the coefficients, global p values and performance metrics of each model fit over the multiple data bootstraps to get to a single set of coefficients, and metrics for each different model.

Usage

```
summarise_fits(exectn, ...)
```

Arguments

execn	a modelfitter execution of multiple model fits.
...	not used at present

Details

Coefficients are combined assuming normally distributed beta coefficients, prior to link function inversion, as mixtures of normal distributions and 95% confidence intervals calculated. performance metrics and global p values are given as means of the values of bootstrap.

Value

a nested dataframe with a single row per model and summary columns in `coef_summary`, `global_p_summary` and

Examples

```
# Complex example, multiple statsistical models, multiple formulae,
# bootstrapped data.
cfg  = configure_models(
   formula_provider(
      "<F" = I(color < "F") ~ cut + carat + clarity + price,
      "<H" = I(color < "H") ~ cut + carat + clarity + price
   ),
   bootstrap_provider(ggplot2::diamonds, max_n = 10),
   model_function_provider(
     "Log reg" = modelfitter::logistic_regression,
     "Poisson" = modelfitter::quasi_poisson
   )
)

execn = cfg %>% execute_configuration(cache = TRUE)
summfit = execn %>% summarise_fits()

# Simple example
tmp = configure_model(
  "Iris", I(Species == "versicolor") ~ ., iris, logistic_regression)
tmp2 = tmp %>% execute_configuration()
summfit = tmp2 %>% summarise_fits()
```

---

univariate_from_multivariate

Convert multivariate formula to list of univariate formulae

Description

Convert multivariate formula to list of univariate formulae

Usage

```
univariate_from_multivariate(formula)
```

Arguments

formula	a formula of the type y ~ x1 + x2 + x3 + ...

Value

a list of formulae of the type y ~ x1, y ~ x2, y ~ x3, ...

Examples

```
univariate_from_multivariate(y ~ x1 + x2 + x3)
univariate_from_multivariate(~ x1 + x2 + x3)
```

Index