Package 'linear.tools'

July 6, 2016

Type Package

Title Manipulate Formulas and Evaluate Marginal Effects

Version 1.3.0

Author Fan Yang

Maintainer Fan Yang <yfno1@msn.com>

Description Provides tools to manipulate formulas, such as getting x, y or contrasts from the model/formula, and functions to evaluate and check the marginal effects of a linear model.

License GPL-3

LazyData True

RoxygenNote 5.0.1

Suggests knitr, rmarkdown

VignetteBuilder knitr

Imports ggplot2, plyr, pryr, stringr, stats, utils, magrittr, scales

NeedsCompilation no

Repository CRAN

Date/Publication 2016-07-06 09:50:47

R topics documented:

`deleting_wrongeffect` *check monotonicity of marginal impacts and re-estimate the model.*

Description

check monotonicity of marginal impacts and re-estimate the model (optional) until we get correct marginal impacts.

Usage

```
deleting_wrongeffect(model, focus_var_raw = NULL, focus_var_model = NULL,
  Monoton_to_Match = 1, family = NULL, re_estimate = TRUE, data,
  STOP = FALSE, PRINT = TRUE, PLOT = TRUE, ...)
```

Arguments

`model,`	an output of lm or glm
`focus_var_raw`	see `effects`.
`focus_var_model`	see `effects`.
`Monoton_to_Match`	1 or -1. 1 means you want monotonic increasing as the correct marginal effect, -1 means negative
`family`	family of glm, for example, can be gaussian `"(link = 'identity')"` or `"(link = 'logit')"`. If NULL, we will use the default family of the model
`re_estimate`	a boolean with default as TRUE. This is to decide if the marginal impacts are found to be incorrect, then whether to delete a model var that potentially cause the wrong marginal impacts and re-estimate the model
`data`	optional, a new dataset to show the marginal impacts and re-estimate the model. If NULL, then use the data used in model itself.
`STOP`	a boolean. When find a model with incorrect marginal impacts, whether to stop there and wait to continue (call the `Enter_to_Continue`)
`PRINT`	a boolean, whether to print messages and to plot.
`PLOT`	a boolean, whether to plot.
`...`	additional arguments going to `effect`

Details

This function first calls function `effects` and then checks the monotonicity of marginal impacts. If the direction of marginal impacts are incorrect, it can delete a model var that potentially causes the wrong marginal impacts and then re-estimate the model. We will keep doing this until the correct marginal impacts are found

Details of evaluating the marginal impacts `effects`

Value

a model (`lm` or `glm`).

- If re_estimate == TRUE, then return will be an re-estimated model with correct marginal impacts given we can find one.
- If re_estimate == FALSE, original model will be returned.

Examples

```
##
set.seed(413)
traing_data = ggplot2::diamonds[runif(nrow(ggplot2::diamonds))<0.05,]
nrow(traing_data)

diamond_lm3 = lm(formula = price ~ carat + I(carat^2) + I(carat^3) + cut +
                   I(carat * depth) , data = traing_data)


test = deleting_wrongeffect(model = diamond_lm3,
                            focus_var_raw = 'carat',
                            focus_var_model = c("I(carat^3)","I(carat*depth)",
                                                "I(carat^2)","I(carat)"),
                            focus_value = list(carat=seq(0.5,6,0.1)),
                            data = traing_data,
                            PRINT = TRUE,STOP = FALSE,
                            Reverse = FALSE)


## two focus on vars
test =
  deleting_wrongeffect(model = diamond_lm3 ,
                       focus_var_raw = c('carat',"cut"),
                       focus_var_model = c("I(carat*depth)","I(carat^3)"),
                       focus_value = list(carat=seq(0.5,6,0.1)),
                       data = traing_data,PRINT = TRUE,STOP =FALSE)

diamond_lm3 = lm(formula = price ~ cut + depth +
                   I(carat * depth) , data = ggplot2::diamonds)
##  negative signs
deleting_wrongeffect(model = diamond_lm3 ,
                     focus_var_raw = c('depth',"cut"),
                     focus_var_model = c("depth"),Monoton_to_Match = -1,
```

```
                    data = ggplot2::diamonds,PRINT = TRUE,STOP =FALSE)

## wrong variables names
deleting_wrongeffect(diamond_lm3, focus_var_raw = 'carat',
                    focus_var_model = c("I(cara79t^3)"),
                    data = ggplot2::diamonds,PRINT = TRUE)

deleting_wrongeffect(diamond_lm3, focus_var_raw = 'carat890',
                    focus_var_model = c("I(carat^3)"),
                    data = ggplot2::diamonds, PRINT = TRUE)
```

`effect` *evaluate the marginal effects of the selected raw variable on the dependent.*

Description

evaluate the marginal effects of the selected raw variable on the dependent.

Usage

```
effect(model, data = NULL, focus_var_raw, focus_var_coeff = NULL,
  focus_var_model = NULL, focus_value = NULL, nonfocus_value = NULL,
  transform_y = NULL, PRINT = TRUE, PLOT = TRUE, Reverse = FALSE,
  bar_plot = NULL, intolerance_on_wrong_names = FALSE)
```

Arguments

`model`	an output of lm or glm
`data`	NULL (default) or a data.frame, a new dataset to evaluate the categorical variables. If NULL, then use the data used in model itself.
`focus_var_raw`	NULL or a character vector with maximum length of 2, in which you can choose `raw vars` you want to focus. See `get_x` for the meaning of `raw var`. • If there is only one raw var in the vector `focus_var_raw`, then we will check the marginal impact of that raw var. • If there is only two raw vars in the vector `focus_var_raw`, then we will check the marginal impact of the FIRST raw var (`focus_var_raw[1]`) under different values of SECOND raw var (`focus_var_raw[2]`). See the example code for details.
`focus_var_coeff`	NULL or a character vector. Must be `coeff vars` containing `focus_var_raw[1]`. See `get_x` for the meaning of `coeff var`. After you set up the `focus_var_raw`, you can also choose to focus on effects of `focus_var_raw[1]` through only certain coeff vars, then all other unspecified coeff vars related `focus_var_raw[1]` will have coeff 0 by default, focus_var_coeff is null, which means we will check effect of `focus_var_raw[1]` on all coeff vars. See the example code for details.

`focus_var_model`
: NULL or a character vector. Must be model vars containing `focus_var_raw[1]`. See `get_x` for the meaning of `model var`. Similar use as argument `focus_var_coeff`, except here you can specify which model vars you want to focus.

  See the example code for details.

`focus_value`
: NULL or a list; each element of the list must have names in focus_var_raw. By default, we will check marginal effects of `focus_var_raw[1]` through `seq(0.05,0.95,by = 0.05)` quantiles of its values in the modelling data. But you can also specify the values you want to check here. See the sample code.

`nonfocus_value`
: NULL or a list; each element of the list must have names in non-focused raw vars (not show up in `focus_var_raw`) The meaning of non-focus var is: When we check the marginal effect of focus var on dependent, we let the focus var vary and fix the non-focus vars. By default, for non-focused raw vars, we assume their values are fixed at mean (if numeric) or mode (if factor or character) in the modelling data. But you can also specify the fixed values you want. See the sample code.

`transform_y`
: NULL or a function, used only for plot. Used as a function to recalculate y (a function on y (ex. log(y))).

`PRINT`
: a boolean, whether to print messages AND to plot.

`PLOT`
: a bookean, whether to plot

`Reverse`
: a boolean, whether to use reverse order in x-axis when plot. Default is FALSE.

`bar_plot`
: NULL or a boolean, choose bar plot or line plot. If NULL, we will choose automatically.

`intolerance_on_wrong_names`
: a boolean. If a name is wrong, either in focus_var_raw, focus_var_model, focus_var_coeff, focus_value or nonfocus_value, whether we delete the wrong names and go on (default), or report an error.

Details

This function will evaluate marginal impacts and show the monotonicity of marginal impacts of a selected variable on the dependent.

Note that the marginal impacts is not simply the sign of coeff: In a model like `y~ x + x^2 + p + q`, marginal impacts of `x` on `y` requires an evaluation of both `x` and `x^2` at the same time.

Here the `focus_var_raw` is `x`, `focus_var_coeff` are `x` and `x^2` `nonfocus_value` is `p` and `q`

Also the monotonicity of marginal impacts of `x` will be different for different range of `x`'s values.

Another interesting case is when `x` is interacting with other variables, then its marginal impacts will also be dependent on the values of those interacted variables.

Level of marginal impacts: To make the level of marginal impacts of `x` realistic, by default we fixed all other right-hand-side variables fixed at their mean (numeric) or mode (character or factor). You can also provide fixed values for them. Also by default we let the interested variable (focused raw var) `x` to vary between its `seq(0.05,0.95,by = 0.05)` quantiles.

This function will take care those cases above and make evaluating marginal impacts easier.

Value

a list:

- Focus_values: show the values of focus_var_raw we used to evaluate the marginal effects.
- data_and_predict: full dataset used to evaluate the marginal effects.
- summmary_glm: a summary of lm or glm model.
- Monoton_Increase: whether the marginal impact is Monotonic Increase.
- Monoton_Decrease: whether the marginal impact is Monotonic Decrease.

Examples

```
##___ unit test ____

# ___________________  One Dimension: the most basic case _____________________



set.seed(413)
traing_data = ggplot2::diamonds[runif(nrow(ggplot2::diamonds))<0.05,]
nrow(traing_data)

diamond_lm3 = lm(price~ cut + carat + I(carat^2) +
                   I(carat^3) + I(carat  * depth) + cut:depth, traing_data) # a GLM

# more carats, higher price.
effect(model = diamond_lm3,
       data = traing_data,
       focus_var_raw = c('carat'),
       Reverse = TRUE) # value in x-axis is reverse

# focus on only 'I(carat^3)', which means we will make all other coeff,
# including 'carat' and 'I(carat^2)' into 0
effect(model = diamond_lm3,
       data =traing_data,
       focus_var_raw =c('carat'),
       focus_var_coeff = 'I(carat^3)')
# ___________________  One Dimension: Categorical _____________________

# selected model-var to focus: here not focus on cut:depth, only focus on cut
suppressWarnings(
  effect(model = diamond_lm3,
         data = traing_data,
         focus_var_raw = c('cut'),
         focus_var_model = 'cut'
         )
  )

# ___________________  Double Dimensions _____________________

# here focus_var_raw has two values: "carat" and "cut"
```

```
# that means we will evaluate impact of "carat" on "price" through different value of "cut"

effect(model = diamond_lm3,data = traing_data, focus_var_raw=c('carat',"cut"))

# ___________________  Provide Values to Focused vars  _____________________

# when evaluating impacts,
# we can provide the range of values for key variables

effect(model = diamond_lm3,data = traing_data,
       focus_var_raw = c('carat',"cut"),
       focus_value = list(carat=seq(0.5,6,0.1)))
```

Enter_to_Continue *Enter_to_Continue: wait your response to continue*

Description

wait your response to continue

Usage

```
Enter_to_Continue(df_input_output = NULL)
```

Arguments

`df_input_output`
: data.frame. df_input_output shall be either NULL or a two column data.frame with characters as values, with first column as what you want to type, and second column as what you want to return. If it is NULL, then it will return ' Press [enter] to continue; Type [s] to stop'. See the sample code for the df case.

Value

Type through keyboard to continue in console.

Examples

```
Enter_to_Continue(rbind(c('small','small data'),c('n','normal'),c('w','weird curve')))
```

`focusing_var_coeff` *focusing on selected variables in the model, and eliminating impacts from other variables.*

Description

focusing on selected variables in the model, and eliminating impacts from other variables.

Usage

```
focusing_var_coeff(model, focus_var_coeff = NULL, focus_var_raw = NULL,
  intercept_include = TRUE, data = NULL)
```

Arguments

`model`	an output of lm or glm
`focus_var_coeff`	NULL or a character vector, choose coeff vars you want to focus. The unselected vars will have coeff values as 0. Default is NULL, which means to choosing nothing.
`focus_var_raw`	NULL or a character vector, choose raw vars you want to focus. The unselected vars will have coeff values as 0. Default is NULL, which means to choosing nothing.
`intercept_include`	a boolean, whether to include the intercept (default is TRUE).
`data`	optional, a new dataset to evaluate the categorical variables. If NULL, then use the data used in model itself.

Details

In a model `y ~ a + b`. Sometimes you want to fix value of `a` and see the variations of `b` in `y`. The most straightforward way to code this, as we did in this function, is to make `a`'s coefficients as 0, and then use the predict().

Value

a new model with only focused vars having coeff unchanged, and all other vars having coeff as 0.

Examples

```
focus_var_raw  = 'carat'

model = lm(price~ cut + carat + I(carat^2) + I(carat^3) +
             I(carat  * depth) + depth,ggplot2::diamonds)
# all coeffs except carat's will be 0
focusing_var_coeff(model, focus_var_coeff = 'carat')
```

```
# all coeffs except cut.L's will be 0
focusing_var_coeff(model, focus_var_coeff = 'cut.L')
# all coeffs without raw vars cut or carat will be 0
focusing_var_coeff(model, focus_var_raw = c('cut','carat'))

# if you didn't specify anything, then all vars' coeff will become 0 except intercept
focusing_var_coeff(model)


# if cannot find the focus_var_coeff or focus_var_raw in the model
tryCatch(focusing_var_coeff(model, focus_var_coeff = 'caratdsd'),
         error = function(err) warning(err))
tryCatch(focusing_var_coeff(model, focus_var_raw = '3213'),
         error = function(err) warning(err))
```

`get_contrast` *get contrast of categorical variables in a model*

Description

get contrast of categorical variables in a model

Usage

```
get_contrast(model, data = NULL, PRINT = TRUE, return_method = FALSE,
  delete.minus.var = TRUE)
```

Arguments

`model`	a model, either `lm` or `glm`.
`data`	dataframe, to provide new data to evaluate the model. If NULL (default), then we use the default data in the model.
`PRINT`	a boolean, whether to print messages. Default is TRUE.
`return_method`	a boolean, whether to return the method of contrast, rather than the contrast itself. Default is FALSE.
`delete.minus.var`	a boolean. whether to delete x2 in y ~ x1 - x2. Default is TRUE.

Details

When R put categorical vars in the linear model, R will transform them into set of 'contrast' using certain contrast encoding schedule. See example code and the reference link below for details.

Value

contrasts of the categorical vars in the model, or the contrast method if `return_method` is TRUE.

References

http://www.ats.ucla.edu/stat/r/library/contrast_coding.htm

Examples

```
get_contrast(lm(price ~ carat + I(carat^2) + cut:carat +
                  color,ggplot2::diamonds))
get_contrast(lm(price ~ carat + I(carat^2) + cut:carat +
                  color,ggplot2::diamonds),return_method = TRUE)

# dirty formulas: all categorical vars are with minus sign
# no categorical vars, thus no contast
get_contrast(lm(price ~ carat + I(carat^2) ,ggplot2::diamonds))

model_dirty = lm(price ~ carat + I(carat^2) - cut:carat - color,
                 ggplot2::diamonds)
get_contrast(model = model_dirty )

diamond_lm3 = lm(price~ I(cut) + depth,ggplot2::diamonds) # a GLM
get_contrast(model = diamond_lm3 )
```

---

get_model_pair *get a list of model vars with their corresponding coeff vars or raw vars.*

Description

get a list of model vars with their corresponding coeff vars or raw vars.

Usage

```
get_model_pair(model, data = NULL, pair_with = c("coeff", "raw"))
```

Arguments

model	a lm or glm output
data	NULL (default) or data.frame, a new dataset to evaluate the categorical variables. If NULL, then use the data used in model itself.
pair_with	either 'raw' (default) or 'coeff', to decide the elements of list are raw vars or coeff vars. See get_x for the meaning of model var, coeff var and raw var.

Details

get a list of model vars with their corresponding coeff vars or raw vars. See get_x for the meaning of model var, coeff var and raw var.

Value

a list with names as model vars and elements as their corresponding coeff/raw vars

Examples

```
# return coeff
get_model_pair(model = price~  I(carat^2) + cut  + carat*table, data = ggplot2::diamonds)
# return raw vars
get_model_pair(price~  I(carat^2) + cut  + carat*table, data= ggplot2::diamonds, pair_with = 'raw')

# correctly deal with irregular formulas
model_dirty = lm(price~  I(carat^   2) + cut  - carat:table - cut ,ggplot2::diamonds)
get_model_pair(model_dirty,pair_with = 'raw')
```

---

get_model_with_coeff *get a list of model variables with their corresponding coeff vars.*

Description

a wrap up function of get_model_pair

Usage

```
get_model_with_coeff(model, data = NULL)
```

Arguments

model	See get_model_pair
data	See get_model_pair

Details

See get_model_pair

Value

a list with names as model vars and elements as their corresponding coeff

Examples

```
get_model_with_coeff(price~  I(carat^   2) + cut  + carat*table, data= ggplot2::diamonds)
```

get_model_with_raw *get a list of model vars with their corresponding raw vars.*

Description

a warp up function of get_model_pair

Usage

```
get_model_with_raw(model, data = NULL)
```

Arguments

model,	See get_model_pair
data,	See get_model_pair

Details

See get_model_pair

Value

a list with names as model vars and elements as their raw coeff

Examples

```
get_model_with_raw(price~  I(carat^   2) + cut  + carat*table, data= ggplot2::diamonds)
```

get_valid_rows *identify missing rows for model/formula.*

Description

identify missing rows for model/formula.

Usage

```
get_valid_rows(model, data)
```

Arguments

model	a formula or an output of lm or glm
data	the data.frame supposed to be used in modelling

Details

Data often contains missing values and `lm()` or `glm()` often skip those rows. This function is to identify which rows that `lm()` or `glm()` skips.

Value

a boolean vector with same length as the number of rows of data, with TRUE if a row has full data for the modelling and FALSE if not.

Examples

```
model = lm(price ~ carat, head(ggplot2::diamonds,1000))
data = head(ggplot2::diamonds,10)

# so observation 1, 4, 7 will be not valid rows
data[c(1,4,7),"price"] = NA
data
get_valid_rows(model,data)

# error message as no "price" is found in the data
data[,"price"] = NULL
tryCatch(get_valid_rows(model,data),
         error = function(x){
           print(x)
         })
```

---

get_x *get x (left hand of var) from model or formula*

Description

get x (left hand of var) from model or formula

Usage

```
get_x(model, method = c("raw", "model", "coeff"), data = NULL)
```

Arguments

`model`	a formula or a model.
`method`	either `'raw'`, `'model'`, or `'coeff'`, to decide what kind variables to show. Default is 'raw'. See section Detials below.
`data`	a dataframe, to provide new data to evaluate the model. If NULL (default), then we use the default data in the model.

Details

What do 'raw' variable, 'model' variable, and 'coeff' variable mean?

- raw var is the underlying variable without any calculation or transformation.
- model var is the underlying variable with calculations or transformation.
- coeff var is the coefficient variable in the model output. So only evaluated model has coeff vars. Most of the time one categorical variable will have several coeff vars according to their contrast encoding. see `get_contrast`

Example:

In the model, `log(price) ~ cut + I(carat^2)` in `diamonds` data, we have:

- 'raw' variables of x: `carat` and `cut`.
- 'model' variables of x: `I(carat^2)` and `cut`.
- 'coeff' variables of x: `cut.L`,`"cut.Q"`,`"cut.C"`,`"cut^4"` and `I(carat^2)`.

See the sample code below for more examples.

Value

x variables in the formula or model

Examples

```
# use the sample code from get_x_hidden
#
data = ggplot2::diamonds
diamond_lm  =  lm(price~  I(carat^   2) + cut  + carat*table ,ggplot2::diamonds)

#_________ input as model
get_x(model = diamond_lm,method = 'raw')
get_x(diamond_lm,method = 'model')
get_x(diamond_lm,method = 'coeff')

#_______ input as formula
get_x(formula(diamond_lm),method = 'model')
# data is required when input is formula
get_x(formula(diamond_lm), data = ggplot2::diamonds, method = 'coeff')

tryCatch(
  get_x(formula(diamond_lm),method = 'coeff'),
  error =function(err){
    print(err)
  }
)



#________ irregular formulas __________
```

```
model_dirty = model = lm(price~  I(carat^  2) + cut  - carat:table - cut ,ggplot2::diamonds)

# CORRECT for raw vars
get_x(model_dirty)

# correct for model vars
get_x(price~  I(carat^2) + cut  - carat:table - cut,data = ggplot2::diamonds, method ='model')
get_x(model_dirty,method = 'model')
get_x(model_dirty,data = ggplot2::diamonds, method = 'model')
get_x(model_dirty, method = 'model')

# clean method for model vars
# terms((price~  I(carat^2) + cut  - carat:table - cut)) %>% attr(.,"factors") %>% colnames()
# model_dirty %>% terms %>% attr(.,"factors") %>% colnames()
# formula(model_dirty) %>% terms %>% attr(.,"factors") %>% colnames()
```

`get_x_all` *a unique combinations of model vars, coeff vars and raw vars*

Description

a unique combinations of model vars, coeff vars and raw vars

Usage

```
get_x_all(model, data = NULL)
```

Arguments

`model`	`lm` or `glm`
`data`	NULL (default) or data.frame, a new dataset to evaluate the categorical variables. If NULL, then use the data used in model itself.

Details

For the differences between raw var, model var, and coeff var: see `get_x`

Value

a data.frame, a unique combinations of model vars, coeff vars and raw vars See `get_x` for the meaning of `model var`, `coeff var` or `raw var`.

The column `'n_raw_in_model'` is a numeric field showing how many raw variables are in the corresponding model variables. For example, the model variable 'I(carat*table)' contains two raw variables: 'carat' and 'table'. See example code for details.

Examples

```
get_x_all(model = price~  I(carat^   2) + cut  + I(carat*table),data = ggplot2::diamonds)


#________ irregular formulas
model_dirty = lm(price~  I(carat^   2) + cut  - carat:table - cut ,ggplot2::diamonds)
test = get_x_all(model_dirty)

test
test$coeff
# ______  errors _____________

tryCatch(get_x_all(model = price~  I(carat^   2) + cut  + I(carat*table)),
         error = function(x){
           print(x)
         })
```

`get_y` *get y (right hand of var)*

Description

get y (right hand of var)

Usage

```
get_y(Formula, method = c("raw", "model", "coeff"))
```

Arguments

`Formula`	a formula to be paste.
`method`	either `'raw'`, `'model'`, or `'coeff'`, to decide what kind variables to show. Default is 'raw'. See section Detials below.

Details

What do 'raw' variable, 'model' variable, and 'coeff' variable mean?

- raw var is the underlying variable without any calculation or transformation.
- model var is the underlying variable with calculations or transformation.
- coeff var is the coefficient variable in the model output. So only evaluated model has coeff var.

In the formula, `log(y) ~ x1 + x2`, we have: 'raw' variable for `y`: `y` 'model' variable for `y`: `log(y)` 'coeff' variable for `y`: `log(y)`

More examples see the sample code below.

Value

y in formula

Examples

```
get_y(log(price) ~sdfsf + dsa )
get_y(log(price) ~ sdfsf + dsa, method = "model")
get_y(log(price) ~ sdfsf + dsa, method = "coeff") # same as model var in the get_y() case

# can deal with un-regular formula
get_y(log(price) ~sdfsf + dsa ~ dsad)
get_y(log(price) ~ sdfsf + dsa ~ dsad, method = "coeff")
get_y(log(price) ~ sdfsf + dsa ~ dsad, method = "model")

model_dirty = model = lm(price~  I(carat^  2) + cut  - carat:table - cut ,ggplot2::diamonds)
get_y(model_dirty)
```

---

paste_formula *paste a formula as string*

Description

paste a formula as string

Usage

```
paste_formula(Formula, exclude_y = FALSE, clean = FALSE)
```

Arguments

Formula	a formula to be pasted.
exclude_y	a boolean, whether to exclude y when paste. Default is FALSE.
clean	a boolean, whether to clean dirty formula: for example – price ~ cut + carat - cut will be cleaned into price ~ carat. Default is FALSE.

Details

a pasted formula in string, with all spaces deleted. This function uses get_y and get_x behind the scene.

Value

a pasted formula in string, with all spaces deleted.

Examples

```
paste_formula(price~carat +cut)
paste_formula(price~carat + cut)

paste_formula(price~carat +cut,exclude_y = TRUE)
paste_formula(Formula = price ~ cut + carat, clean = TRUE)

paste_formula(price~carat +cut - cut, clean = TRUE)

# irregular formulas: cross lines
paste_formula(price~carat +
                cut ~ dsad)

paste_formula(price~carat +
                cut ~ dsad,exclude_y = TRUE)
```

`stepwise2` *same as* `step()` *in R, but able to check marginal effects.*

Description

same as `step()` in R, but able to check marginal effects.

Usage

```
stepwise2(model, scope, trace = 1, steps = 1000, k = 2, data,
  family = NULL, IC_method = c("AIC", "BIC"), test_suit = NULL,
  STOP = FALSE)
```

Arguments

`model`	an output of `lm` or `glm`
`scope, trace, steps, k`	see `step()`
`data`	a data.frame used in regression.
`family`	used as the argument for `family` of `glm`, default is NULL, which means we will use the family imbedded in the model.
`IC_method`	either 'AIC' or 'BIC', will overwrite the `k` argument above.
`test_suit`	used to specify the correct marginal effect you want to check. It is a list with names as raw variable and values as arguments of the function `deleting_wrongeffect` If NULL (default), then not check any marginal effect See example code for details.
`STOP`	whether stop and wait your response for each step.

Details

For each step of regression, you can first choose the models with correct marginal effect and then choose the one with highest AIC/BIC within them

Value

a stepwise-selected model. If `test_suit` is specified, then the returned model is the one with highest AIC/BIC within those that get correct marginal impact.

The silde effect is to print a data.frame containing diagnostic informations for each step. The 'correct_effect_ind' column is a boolean vector to show whether the model has correct marginal effect or not.

Examples

```
# starting model:
# can have a dirty formula like below

set.seed(413)
traing_data = ggplot2::diamonds[runif(nrow(ggplot2::diamonds))<0.05,]
nrow(traing_data)

diamond_lm3 = lm(formula = price ~ cut + carat - cut   , data = traing_data)

scope = list(lower = price ~ 1,
             upper = price ~  I(carat^2) + I(carat^3) + I(carat * depth) + depth + carat)

# traditional stepwise regression with no marginal effect check
model1 = stepwise2(model = diamond_lm3, scope = scope,k = 2,
                   trace = TRUE, data = traing_data, STOP = TRUE)
model1
# result is exactly same using the default step() function.
model2 = suppressWarnings(step(diamond_lm3,scope = scope, k = 2))
model2


#__ How to Specify the Correct Marginal Effects in Stepwise Regression  __

# this test_suit means we will check the marginal effect of both 'carat' and 'depth'
# for 'carat', we will only focus on 4 coeff vars :
    # "I(carat^3)","I(carat*depth)","I(carat^2)","carat"
# for 'depth', as we do not specify any particular coeff vars there,
# we will check all coeff var related to 'depth'

test_suit = list(
  carat = list(
    # the list name must be the raw var
    focus_var_raw = "carat",
    # must specify the focus_var_raw (see deleting_wrongeffect() ) as the raw var
    focus_var_coeff = c("I(carat^3)","I(carat*depth)",
                        "I(carat^2)","carat") ,
```

```
    # optional # If not defined, then we to check all coeffs related to the raw var
    focus_value =list(carat = seq(0.5,6,0.1)),
    Monoton_to_Match = 1 # optional. Default is 1
  ),
  depth = list(
    focus_var_raw = "depth",
    Monoton_to_Match = 1
  )
)

model3 =  stepwise2(model = diamond_lm3, scope = scope, trace = TRUE,
                    data = traing_data,
                    STOP = FALSE, test_suit = test_suit)

# see the difference from model1
effect(model3,focus_var_raw =  "carat", focus_value =list(carat = seq(0.5,6,0.1)))
```

Index