

Package ‘ggeffects’

January 15, 2018

Type Package

Encoding UTF-8

Title Create Tidy Data Frames of Marginal Effects for 'ggplot' from Model Outputs

Version 0.3.1

Date 2018-01-16

Author Daniel Lüdecke <d.luedecke@uke.de>

Maintainer Daniel Lüdecke <d.luedecke@uke.de>

Description Compute marginal effects at the mean or average marginal effects from statistical models and returns the result as tidy data frames. These data frames are ready to use with the 'ggplot2'-package. Marginal effects can be calculated for many different models. Interaction terms, splines and polynomial terms are also supported. The two main functions are `ggpredict()` and `ggaverage()`, however, there are some convenient wrapper-functions especially for polynomials or interactions. There is a generic `plot()`-method to plot the results using 'ggplot2'.

License GPL-3

Depends R (>= 3.2), stats, utils

Imports dplyr (>= 0.7.1), effects (>= 4.0-0), ggplot2 (>= 2.2.1), magrittr, prediction (>= 0.2.0), purrr, rlang, scales, sjlabelled (>= 1.0.6), sjmisc (>= 2.6.3), sjstats (>= 0.14.0), tibble (>= 1.3.3), tidyr (>= 0.7.0), tidyselect

Suggests knitr, MASS, rmarkdown, rstantools

URL <https://github.com/strengejacked/ggeffects>

BugReports <https://github.com/strengejacked/ggeffects/issues>

RoxygenNote 6.0.1

VignetteBuilder knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2018-01-15 14:36:01 UTC

R topics documented:

efc	2
emm	3
get_title	4
ggalleffects	5
ggaverage	6
ggeffect	11
gginteraction	12
ggpoly	14
plot	16
Index	19

 efc

Sample dataset from the EUROFAMCARE project

Description

A SPSS sample data set, imported with the [read_spss](#) function.

Examples

```
# Attach EFC-data
data(efc)

# Show structure
str(efc)

# show first rows
head(efc)

# show variables
## Not run:
library(sjmisc)
library(sjPlot)
view_df(efc)

# show variable labels
get_label(efc)

# plot efc-data frame summary
sjt.df(efc, alternateRowColor = TRUE)
## End(Not run)
```

emm

*Get marginal effects for model response***Description**

`emm()` is a convenient shortcut to compute the estimated marginal mean, resp. the marginal effect of the model's response variable, with all independent variables held constant (at their [typical_value](#)).

Usage

```
emm(model, ci.lvl = 0.95, type = c("fe", "re"), typical = "mean", ...)
```

Arguments

<code>model</code>	A fitted model object, or a list of model objects. Any model that supports common methods like <code>predict()</code> , <code>family()</code> or <code>model.frame()</code> should work.
<code>ci.lvl</code>	Numeric, the level of the confidence intervals. For <code>ggpredict()</code> , use <code>ci.lvl = NA</code> , if confidence intervals should not be calculated (for instance, due to computation time).
<code>type</code>	Character, only applies for mixed effects models. Indicates whether predicted values should be conditioned on random effects (<code>type = "re"</code>) or fixed effects only (<code>type = "fe"</code> , the default).
<code>typical</code>	Character vector, naming the function to be applied to the covariates over which the effect is "averaged". The default is "mean". See typical_value for options.
<code>...</code>	Further arguments passed down to <code>predict()</code> .

Details

For linear models, the predicted value is the estimated marginal mean. Else, the predicted value is on the scale of the inverse of link function.

Value

A tibble with the marginal effect of the response (`predicted`) and the confidence intervals `conf.low` and `conf.high`. For `polr`-objects, the marginal effect for each level of the response variable is returned.

Examples

```
data(efc)
fit <- lm(barthtot ~ c12hour + neg_c_7 + c161sex + c172code, data = efc)
emm(fit)

# Example from ?MASS::polr
library(MASS)
options(contrasts = c("contr.treatment", "contr.poly"))
house.plr <- polr(Sat ~ Infl + Type + Cont, weights = Freq, data = housing)
```

```
emm(house.plr)
```

get_title	<i>Get titles and labels from data</i>
-----------	--

Description

Get variable and value labels from ggeffects-objects. Functions like `ggpredict()` or `gginteraction()` save information on variable names and value labels as additional attributes in the returned data frame. This is especially helpful for labelled data (see [sjlabelled](#)), since these labels can be used to set axis labels and titles.

Usage

```
get_title(x, case = NULL)
get_x_title(x, case = NULL)
get_y_title(x, case = NULL)
get_legend_title(x, case = NULL)
get_legend_labels(x, case = NULL)
get_x_labels(x, case = NULL)
get_complete_df(x, case = NULL)
```

Arguments

x	An object of class <code>ggeffects</code> , as returned by any <code>ggeffects</code> -function; for <code>get_complete_df()</code> , must be a list of <code>ggeffects</code> -objects.
case	Desired target case. Labels will automatically converted into the specified character case. See convert_case for more details on this argument.

Value

The titles or labels as character string, or NULL, if variables had no labels; `get_complete_df()` returns the input list `x` as single data frame, where the grouping variable indicates the marginal effects for each term.

Examples

```
data(efc)
efc$c172code <- sjmisc::to_factor(efc$c172code)
fit <- lm(barthtot ~ c12hour + neg_c_7 + c161sex + c172code, data = efc)
```

```

mydf <- ggpredict(fit, terms = c("c12hour", "c161sex", "c172code"))

library(ggplot2)
ggplot(mydf, aes(x = x, y = predicted, colour = group)) +
  stat_smooth(method = "lm") +
  facet_wrap(~facet, ncol = 2) +
  labs(
    x = get_x_title(mydf),
    y = get_y_title(mydf),
    colour = get_legend_title(mydf)
  )

# get marginal effects, a list of tibbles (one tibble per term)
eff <- ggalleffects(fit)
eff
get_complete_df(eff)

# get marginal effects for education only, and get x-axis-labels
mydat <- eff[["c172code"]]
ggplot(mydat, aes(x = x, y = predicted, group = group)) +
  stat_summary(fun.y = sum, geom = "line") +
  scale_x_discrete(labels = get_x_labels(mydat))

```

ggalleffects

Get marginal effects for all model predictors

Description

ggalleffects() computes marginal effects of model terms. It internally calls [allEffects](#) and puts the result into tidy data frames.

Usage

```
ggalleffects(model, terms = NULL, ci.lvl = 0.95, ...)
```

Arguments

model	A fitted model object, or a list of model objects. Any model that is supported by the effects -package should work.
terms	Character vector with term names of selected variables from model, which should be used to compute marginal effects. If terms = NULL, marginal effects for all model terms are returned.
ci.lvl	Numeric, the level of the confidence intervals. For ggpredict(), use ci.lvl = NA, if confidence intervals should not be calculated (for instance, due to computation time).
...	Further arguments passed down to allEffects .

Value

A list of tibbles (with `ggeffects` class attribute) with consistent data columns. The list contains one tibble per model term. Columns are:

`x` the values of the model predictor to which the effect pertains, used as x-position in plots.

`predicted` the predicted values, used as y-position in plots.

`conf.low` the lower bound of the confidence interval for the predicted values.

`conf.high` the upper bound of the confidence interval for the predicted values.

Note

Interaction effects are not included in the return value, i.e. `ggalleffects()` does not compute marginal effects for interaction terms. Use [gginteraction](#) to create tidy data frames especially for interaction terms.

Examples

```
data(efc)
fit <- lm(barthtot ~ c12hour + neg_c_7 + c161sex + c172code, data = efc)
ggalleffects(fit)

library(ggplot2)
fit <- lm(barthtot ~ c12hour + neg_c_7 + c160age, data = efc)
mydf <- get_complete_df(ggalleffects(fit))

ggplot(mydf, aes(x, predicted)) +
  geom_line() +
  facet_wrap(~group, scale = "free_x", ncol = 1)
```

 ggaverage

Get marginal effects from model terms

Description

`ggpredict()` computes predicted (fitted) values for the response, at the margin of specific values from certain model terms, where additional model terms indicate the grouping structure. `ggaverage()` computes the average predicted values. The result is returned as tidy data frame.

`mem()` is an alias for `ggpredict()` (marginal effects at the mean), `ame()` is an alias for `ggaverage()` (average marginal effects).

Usage

```
ggaverage(model, terms, ci.lvl = 0.95, type = c("fe", "re"),
  typical = "mean", ppd = FALSE, ...)

ame(model, terms, ci.lvl = 0.95, type = c("fe", "re"), typical = "mean",
  ppd = FALSE, ...)

ggpredict(model, terms, ci.lvl = 0.95, type = c("fe", "re"),
  full.data = FALSE, typical = "mean", ppd = FALSE, ...)

mem(model, terms, ci.lvl = 0.95, type = c("fe", "re"), full.data = FALSE,
  typical = "mean", ppd = FALSE, ...)
```

Arguments

model	A fitted model object, or a list of model objects. Any model that supports common methods like <code>predict()</code> , <code>family()</code> or <code>model.frame()</code> should work.
terms	Character vector with the names of those terms from <code>model</code> , for which marginal effects should be displayed. At least one term is required to calculate effects, maximum length is three terms, where the second and third term indicate the groups, i.e. predictions of first term are grouped by the levels of the second (and third) term. Indicating levels in square brackets allows for selecting only specific groups. Term name and levels in brackets must be separated by a whitespace character, e.g. <code>terms = c("age", "education [1,3]")</code> . See 'Examples'. All remaining covariates that are not specified in <code>terms</code> are held constant (if <code>full.data = FALSE</code> , the default) or are set to the values from the observations (i.e. are kept as they happen to be; see 'Details').
ci.lvl	Numeric, the level of the confidence intervals. For <code>ggpredict()</code> , use <code>ci.lvl = NA</code> , if confidence intervals should not be calculated (for instance, due to computation time).
type	Character, only applies for mixed effects models. Indicates whether predicted values should be conditioned on random effects (<code>type = "re"</code>) or fixed effects only (<code>type = "fe"</code> , the default).
typical	Character vector, naming the function to be applied to the covariates over which the effect is "averaged". The default is "mean". See typical_value for options.
ppd	Logical, if TRUE, predictions for Stan-models are based on the posterior predictive distribution (posterior_predict). If FALSE (the default), predictions are based on posterior draws of the linear predictor (posterior_linpred).
...	Further arguments passed down to <code>predict()</code> .
full.data	Logical, if TRUE, the returned data frame contains predictions for all observations. This data frame also has columns for residuals and observed values, and can also be used to plot a scatter plot of all data points or fitted values. If FALSE (the default), the returned data frame only contains predictions for all combinations of unique values of the model's predictors. Residuals and observed values are set to NA. Usually, this argument is only used internally by <code>ggaverage()</code> .

Details

Currently supported model-objects are: `lm`, `glm`, `glm.nb`, `lme`, `lmer`, `glmer`, `glmer.nb`, `nlmer`, `glmmTMB`, `ga`. Other models not listed here are passed to a generic `predict`-function and might work as well, or maybe with `ggeffect()`, which effectively does the same as `ggpredict()`. The main difference between `ggpredict()` and `ggeffect()` is how factors are held constant: `ggpredict()` uses the reference level, while `ggeffect()` computes a kind of "average" value, which represents the proportions of each factor's category.

For `ggpredict()`, if `full.data = FALSE`, `expand.grid()` is called on all unique combinations of `model.frame(model)[, terms]` and used as `newdata`-argument for `predict()`. In this case, all remaining covariates that are not specified in `terms` are held constant. Numeric values are set to the mean (unless changed with the `typical`-argument), factors are set to their reference level and character vectors to their mode (most common element).

`ggaverage()` computes the average predicted values, by calling `ggpredict()` with `full.data = TRUE`, where argument `newdata = model.frame(model)` is used in `predict()`. Hence, predictions are made on the model data. In this case, all remaining covariates that are not specified in `terms` are *not* held constant, but vary between observations (and are kept as they happen to be). The predicted values are then averaged for each group (if any).

Thus, `ggpredict()` can be considered as calculating marginal effects at the mean, while `ggaverage()` computes average marginal effects.

`ggpredict()` also works with **Stan**-models from the **rstanarm** or **brms**-package. The predicted values are the median value of all drawn posterior samples. The confidence intervals for Stan-models are actually high density intervals, computed by `hdi`, unless `ppd = TRUE`. If `ppd = TRUE`, predictions are based on draws of the posterior predictive distribution and the uncertainty interval is computed using `predictive_interval`. By default (i.e. `ppd = FALSE`), the predictions are based on `posterior_linpred` and hence have some limitations: the uncertainty of the error term is not taken into account. The recommendation is to use the posterior predictive distribution (`posterior_predict`). Note that for binomial models, the `newdata`-argument used in `posterior_predict()` must also contain the vector with the number of trials. In this case, a dummy-vector is used, where all values for the response are set to 1.

Value

A tibble (with `ggeffects` class attribute) with consistent data columns:

`x` the values of the first term in `terms`, used as x-position in plots.

`predicted` the predicted values of the response, used as y-position in plots.

`conf.low` the lower bound of the confidence interval for the predicted values.

`conf.high` the upper bound of the confidence interval for the predicted values.

`observed` if `full.data = TRUE`, this columns contains the observed values (the response vector).

`residuals` if `full.data = TRUE`, this columns contains residuals.

`group` the grouping level from the second term in `terms`, used as grouping-aesthetics in plots.

`facet` the grouping level from the third term in `terms`, used to indicate facets in plots.

For proportional odds logistic regression (see `polr`) resp. cumulative link models (e.g., see `clm`), an additional column `response.level` is returned, which indicates the grouping of predictions based on the level of the model's response.

Note

Since data for `ggaverage()` comes from the model frame, not all possible combinations of values in terms might be present in the data, thus lines or confidence bands from `plot()` might not span over the complete x-axis-range.

There are some limitations for certain model objects. For example, it is currently only possible to compute predicted risk scores for `coxph`-models, but not expected number of events nor survival probabilities.

`polr`- or `clm`-models have an additional column `response.level`, which indicates with which level of the response variable the predicted values are associated.

Examples

```
data(efc)
fit <- lm(barthtot ~ c12hour + neg_c_7 + c161sex + c172code, data = efc)

ggpredict(fit, terms = "c12hour")
ggpredict(fit, terms = "c12hour", full.data = TRUE)
ggpredict(fit, terms = c("c12hour", "c172code"))
ggpredict(fit, terms = c("c12hour", "c172code", "c161sex"))

# to plot ggeffects-objects, you can use the 'plot()' -function.
# the following examples show how to build your ggplot by hand.

# plot predicted values, remaining covariates held constant
library(ggplot2)
mydf <- ggpredict(fit, terms = "c12hour")
ggplot(mydf, aes(x, predicted)) +
  geom_line() +
  geom_ribbon(aes(ymin = conf.low, ymax = conf.high), alpha = .1)

# with "full.data = TRUE", remaining covariates vary between
# observations, so fitted values can be plotted
mydf <- ggpredict(fit, terms = "c12hour", full.data = TRUE)
ggplot(mydf, aes(x, predicted)) + geom_point()

# you can add a smoothing-geom to show the linear trend of fitted values
ggplot(mydf, aes(x, predicted)) +
  geom_smooth(method = "lm", se = FALSE) +
  geom_point()

# three variables, so we can use facets and groups
mydf <- ggpredict(
  fit,
  terms = c("c12hour", "c161sex", "c172code"),
  full.data = TRUE
```

```

)
ggplot(mydf, aes(x = x, y = predicted, colour = group)) +
  stat_smooth(method = "lm", se = FALSE) +
  facet_wrap(~facet, ncol = 2)

# average marginal effects
mydf <- ggaverage(fit, terms = c("c12hour", "c172code"))
ggplot(mydf, aes(x = x, y = predicted, colour = group)) +
  stat_smooth(method = "lm", se = FALSE)

# select specific levels for grouping terms
mydf <- ggpredict(fit, terms = c("c12hour", "c172code [1,3]", "c161sex"))
ggplot(mydf, aes(x = x, y = predicted, colour = group)) +
  stat_smooth(method = "lm", se = FALSE) +
  facet_wrap(~facet) +
  labs(
    y = get_y_title(mydf),
    x = get_x_title(mydf),
    colour = get_legend_title(mydf)
  )
)

# level indication also works for factors with non-numeric levels
# and in combination with numeric levels for other variables
library(sjlabelled)
data(efc)
efc$c172code <- as_label(efc$c172code)
fit <- lm(barthtot ~ c12hour + neg_c_7 + c161sex + c172code, data = efc)
ggpredict(fit, terms = c("c12hour",
  "c172code [low level of education, high level of education]",
  "c161sex [1]"))

# use categorical value on x-axis, use axis-labels, add error bars
dat <- ggpredict(fit, terms = c("c172code", "c161sex"))
ggplot(dat, aes(x, predicted, colour = group)) +
  geom_point(position = position_dodge(.1)) +
  geom_errorbar(
    aes(ymin = conf.low, ymax = conf.high),
    position = position_dodge(.1)
  ) +
  scale_x_continuous(breaks = 1:3, labels = get_x_labels(dat))

# 3-way-interaction with 2 continuous variables
data(efc)
# make categorical
efc$c161sex <- as_factor(efc$c161sex)
fit <- lm(neg_c_7 ~ c12hour * barthtot * c161sex, data = efc)
# select only levels 30, 50 and 70 from continuous variable Barthel-Index
dat <- ggpredict(fit, terms = c("c12hour", "barthtot [30,50,70]", "c161sex"))
ggplot(dat, aes(x = x, y = predicted, colour = group)) +
  stat_smooth(method = "lm", se = FALSE, fullrange = TRUE) +
  facet_wrap(~facet) +
  labs(
    colour = get_legend_title(dat),

```

```

    x = get_x_title(dat),
    y = get_y_title(dat),
    title = get_title(dat)
  )

# or with ggeffects' plot-method
## Not run:
plot(dat, ci = FALSE)
## End(Not run)

```

ggeffect

Get marginal effects from model terms

Description

`ggeffect()` computes marginal effects of model terms. It internally calls [Effect](#) and puts the result into tidy data frames. `eff()` is an alias for `ggeffect()`

Usage

```

ggeffect(model, terms, ci.lvl = 0.95, ...)

eff(model, terms, ci.lvl = 0.95, ...)

```

Arguments

<code>model</code>	A fitted model object, or a list of model objects. Any model that is supported by the effects -package should work.
<code>terms</code>	Character vector with the names of those terms from <code>model</code> , for which marginal effects should be displayed. At least one term is required to calculate effects, maximum length is three terms, where the second and third term indicate the groups, i.e. predictions of first term are grouped by the levels of the second (and third) term. Indicating levels in square brackets allows for selecting only specific groups. Term name and levels in brackets must be separated by a whitespace character, e.g. <code>terms = c("age", "education [1,3]")</code> . See 'Examples'. All remaining covariates that are not specified in <code>terms</code> are held constant (if <code>full.data = FALSE</code> , the default) or are set to the values from the observations (i.e. are kept as they happen to be; see 'Details').
<code>ci.lvl</code>	Numeric, the level of the confidence intervals. For <code>ggpredict()</code> , use <code>ci.lvl = NA</code> , if confidence intervals should not be calculated (for instance, due to computation time).
<code>...</code>	Further arguments passed down to Effect .

Value

A tibble (with `ggeffects` class attribute) with consistent data columns:

`x` the values of the model predictor to which the effect pertains, used as x-position in plots.

`predicted` the predicted values, used as y-position in plots.

`conf.low` the lower bound of the confidence interval for the predicted values.

`conf.high` the upper bound of the confidence interval for the predicted values.

`group` the grouping level from the second term in terms, used as grouping-aesthetics in plots.

`facet` the grouping level from the third term in terms, used to indicate facets in plots.

Note

The results of `ggeffect()` and `ggpredict()` are usually (almost) identical. It's just that `ggpredict()` calls `predict()`, while `ggeffect()` calls `Effect` to compute marginal effects at the mean. However, results may differ when using factors inside the formula: in such cases, `Effect()` takes the "mean" value of factors (i.e. computes a kind of "average" value, which represents the proportions of each factor's category), while `ggpredict()` uses the base (reference) level when holding these predictors at a constant value.

Examples

```
data(efc)
fit <- lm(barthtot ~ c12hour + neg_c_7 + c161sex + c172code, data = efc)
ggeffect(fit, terms = "c12hour")

mydf <- ggeffect(fit, terms = c("c12hour", "c161sex"))
plot(mydf)
```

gginteraction

Get marginal effects for two-way interactions from models

Description

`gginteraction()` computes marginal effects of interaction terms. It internally calls `effect` and puts the result into tidy data frames.

Usage

```
gginteraction(model, mdrt.values = "minmax", swap.pred = FALSE,
  ci.lvl = 0.95, ...)
```

Arguments

model	A fitted model object, or a list of model objects. Any model that is supported by the effects -package should work.
mdrt.values	Indicates which values of the moderator variable should be used to calculate marginal effects of the interaction. "minmax" (default) minimum and maximum values (lower and upper bounds) of the moderator are used to plot the interaction between independent variable and moderator. "meansd" uses the mean value of the moderator as well as one standard deviation below and above mean value to plot the effect of the moderator on the independent variable. "zeromax" is similar to the "minmax" option, however, 0 is always used as minimum value for the moderator. This may be useful for predictors that don't have an empirical zero-value, but absence of moderation should be simulated by using 0 as minimum. "quart" calculates and uses the quartiles (lower, median and upper) of the moderator value. "all" uses all values of the moderator variable. Note that this option only applies to type = "eff", for numeric moderator values.
swap.pred	Logical, if TRUE, the predictor (defining the x-position) and the moderator (defining the groups) in an interaction are swapped. By default, the first interaction term is considered as moderator and the second term is used to define the x-position.
ci.lvl	Numeric, the level of the confidence intervals. For <code>ggpredict()</code> , use <code>ci.lvl = NA</code> , if confidence intervals should not be calculated (for instance, due to computation time).
...	Further arguments passed down to effect .

Value

A tibble (with `ggeffects` class attribute) with consistent data columns:

`x` the values of the model predictor to which the effect pertains, used as x-position in plots.

`predicted` the predicted values, used as y-position in plots.

`conf.low` the lower bound of the confidence interval for the predicted values.

`conf.high` the upper bound of the confidence interval for the predicted values.

`group` the name of x, used as grouping-aesthetics in plots.

Note

`gginteraction()` only computes marginal effects for interaction terms, in particular two-way interactions. Use [ggeffect](#) for marginal effects for simple model terms. Or use [ggpredict](#) for predictions from any model terms, including two- or three-way interactions.

Examples

```

data(efc)
efc$c172code <- sjmisc::to_factor(efc$c172code)
fit <- lm(barthtot ~ c12hour + c161sex + c172code * neg_c_7, data = efc)
gginteraction(fit)

# this would give the same results
ggpredict(fit, terms = c("neg_c_7", "c172code"))

library(ggplot2)
ggplot(gginteraction(fit), aes(x, predicted, colour = group)) +
  geom_line()

dat <- gginteraction(fit)
ggplot(dat, aes(x, predicted, colour = group)) +
  geom_line() +
  labs(
    colour = get_legend_title(dat),
    x = get_x_title(dat),
    y = get_y_title(dat),
    title = get_title(dat)
  ) +
  scale_color_manual(
    values = c("red", "green", "blue"),
    labels = get_legend_labels(dat)
  )

# use continuous term on x-axis, but use values mean +/- sd as groups
dat <- gginteraction(fit, mdrt.values = "meansd", swap.pred = TRUE)
ggplot(dat, aes(x, predicted, colour = group)) + geom_line()

```

 ggpoly

Get marginal effects for polynomial terms

Description

ggpoly() computes marginal effects for polynomial terms. The result is returned as tidy data frame.

Usage

```
ggpoly(model, poly.term, ci.lvl = 0.95, ...)
```

Arguments

model	A fitted model object, or a list of model objects. Any model that is supported by the effects -package should work.
poly.term	Name of the polynomial term in model, as string.

`ci.lvl` Numeric, the level of the confidence intervals. For `ggpredict()`, use `ci.lvl = NA`, if confidence intervals should not be calculated (for instance, due to computation time).

`...` Further arguments passed down to [effect](#).

Value

A tibble (with `ggeffects` class attribute) with consistent data columns:

`x` the values of the first term in terms, used as x-position in plots.

`predicted` the predicted values, used as y-position in plots.

`conf.low` the lower bound of the confidence interval for the predicted values.

`conf.high` the upper bound of the confidence interval for the predicted values.

`group` the grouping level from the second term in terms, used as grouping-aesthetics in plots.

Note

`ggpoly()` is just an alternative call to `ggpredict()` for polynomial model terms. It may work for certain models that are not yet supported by `ggpredict()`. Otherwise, there should be no difference in the results from `ggpoly()` and `ggpredict()`.

Examples

```
data(efc)
fit <- lm(
  tot_sc_e ~ c12hour + e42dep + e17age + I(e17age^2) + I(e17age^3),
  data = efc
)
dat <- ggpoly(fit, "e17age")

# this would give the same result
ggpredict(fit, "e17age")

library(ggplot2)
ggplot(dat, aes(x, predicted)) +
  stat_smooth(se = FALSE) +
  geom_ribbon(aes(ymin = conf.low, ymax = conf.high), alpha = .15) +
  labs(x = get_x_title(dat), y = get_y_title(dat))

## Not run:
# or:
plot(dat)
## End(Not run)
```

 plot

Plot ggeffects-objects

Description

A generic plot-method for ggeffects-objects.

Usage

```
## S3 method for class 'ggeffects'
plot(x, ci = TRUE, facets, rawdata = FALSE,
     colors = "Set1", alpha = 0.15, dodge = 0.1, use.theme = TRUE,
     dot.alpha = 0.5, jitter = TRUE, case = NULL, show.legend = TRUE, ...)
```

Arguments

x	An object of class ggeffects, as returned by the functions from this package.
ci	Logical, if TRUE, confidence bands (for continuous variables at x-axis) resp. error bars (for factors at x-axis) are plotted. For ggeffects-objects from ggpredict() with argument full.data = TRUE, ci is automatically set to FALSE.
facets	Logical, defaults to TRUE, if x has a column named facet, and defaults to FALSE, if x has no such column. Set facets = TRUE to wrap the plot into facets even for grouping variables (see 'Examples').
rawdata	Logical, if TRUE, a layer with raw data from response by predictor on the x-axis, plotted as point-geoms, is added to the plot.
colors	Character vector with color values in hex-format, valid color value names (see demo("colors") or a name of a color brewer palette. Following options are valid for colors: <ul style="list-style-type: none"> • If not specified, the color brewer palette "Set1" will be used. • If "gs", a greyscale will be used. • If "bw", the plot is black/white and uses different line types to distinguish groups. • If colors is any valid color brewer palette name, the related palette will be used. Use <code>display.brewer.all</code> to view all available palette names. • Else specify own color values or names as vector (e.g. colors = c("#f00000", "#00ff00")).
alpha	Alpha value for the confidence bands.
dodge	Value for offsetting or shifting error bars, to avoid overlapping. Only applies, if a factor is plotted at the x-axis; in such cases, the confidence bands are replaced by error bars.
use.theme	Logical, if TRUE, a slightly tweaked version of ggplot's minimal-theme is applied to the plot. If FALSE, no theme-modifications are applied.
dot.alpha	Alpha value for data points, when rawdata = TRUE.

<code>jitter</code>	Logical, if TRUE and <code>rawdata = TRUE</code> , adds a small amount of random variation to the location of data points dots, to avoid overplotting. Hence the points don't reflect exact values in the data. For binary outcomes, raw data is never jittered to avoid that data points exceed the axis limits.
<code>case</code>	Desired target case. Labels will automatically converted into the specified character case. See convert_case for more details on this argument.
<code>show.legend</code>	Logical, shows or hides the plot legend.
<code>...</code>	Currently not used.

Details

`ggpredict()` with argument `full.data = FALSE` computes marginal effects at the mean, where covariates are held constant. In this case, the slope between groups does not vary and the standard errors and confidence intervals have the same "trend" as the predicted values. Hence, plotting confidence bands or error bars is possible. However, `ggpredict()` with argument `full.data = TRUE`, covariates and standard errors vary between groups, so plotting confidence bands and error bars would follow a "winding" shape, while the predicted values are smoothed by [geom_smooth](#). Predicted values and confidence bands or error bars would no longer match, thus, `ci` is automatically set to FALSE in such cases. You still may want to plot objects returned by `ggpredict()` with argument `full.data = TRUE` to additionally plot the raw data points, which is automatically done.

For `ggaverage()`, which computes average marginal effects, the same problem with standard errors and confidence bands would apply. However, the standard errors for the average marginal effects are taken from the marginal effects at the mean, and the predicted values from the average marginal effects are used to compute another regression on these values, to get the "smoothened" values that are used to compute standard errors and confidence intervals that match the predicted values of the average marginal effects (maybe, at this point, it is helpful to inspect the code to better understand what is happening...).

For proportional odds logistic regression (see [polr](#)), plots are automatically faceted by `response.level`, which indicates the grouping of predictions based on the level of the model's response.

Value

A `ggplot2`-object.

Examples

```
library(sjlabelled)
data(efc)
efc$c172code <- as_label(efc$c172code)
fit <- lm(barthtot ~ c12hour + neg_c_7 + c161sex + c172code, data = efc)

dat <- ggpredict(fit, terms = "c12hour")
plot(dat)

dat <- ggpredict(fit, terms = "c12hour", full.data = TRUE)
plot(dat)
```

```
dat <- ggaverage(fit, terms = "neg_c_7")
plot(dat)

# facet by group
dat <- ggpredict(fit, terms = c("c12hour", "c172code"))
plot(dat, facet = TRUE)

# don't use facets, b/w figure, w/o confidence bands
dat <- ggaverage(fit, terms = c("c12hour", "c172code"))
plot(dat, colors = "bw", ci = FALSE)

# factor at x axis, plot exact data points and error bars
dat <- ggpredict(fit, terms = c("c172code", "c161sex"))
plot(dat)

# for three variables, automatic facetting
dat <- ggpredict(fit, terms = c("c12hour", "c172code", "c161sex"))
plot(dat)
```

Index

*Topic **data**

efc, [2](#)

allEffects, [5](#)

ame (ggaverage), [6](#)

clm, [9](#)

convert_case, [4](#), [17](#)

display.brewer.all, [16](#)

efc, [2](#)

eff (ggeffect), [11](#)

Effect, [11](#), [12](#)

effect, [12](#), [13](#), [15](#)

emm, [3](#)

geom_smooth, [17](#)

get_complete_df (get_title), [4](#)

get_legend_labels (get_title), [4](#)

get_legend_title (get_title), [4](#)

get_title, [4](#)

get_x_labels (get_title), [4](#)

get_x_title (get_title), [4](#)

get_y_title (get_title), [4](#)

ggalleffects, [5](#)

ggaverage, [6](#)

ggeffect, [11](#), [13](#)

gginteraction, [6](#), [12](#)

ggpoly, [14](#)

ggpredict, [13](#)

ggpredict (ggaverage), [6](#)

hdi, [8](#)

mem (ggaverage), [6](#)

plot, [16](#)

polr, [9](#), [17](#)

posterior_linpred, [7](#), [8](#)

posterior_predict, [7](#), [8](#)

predictive_interval, [8](#)

read_spss, [2](#)

typical_value, [3](#), [7](#)