Predictor Effects Graphics Gallery

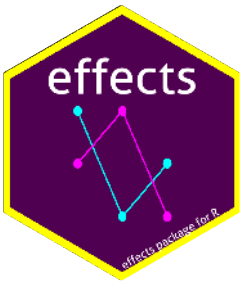
John Fox and Sanford Weisberg

2018-12-19, minor revisions 2023-02-20

Contents

Abstract

Predictor effect displays visualize the response surface of complex regression models by averaging and conditioning, producing a sequence of 2D line graphs, one graph or set of graphs for each predictor in the regression problem (Fox and Weisberg, 2019, 2018). In this vignette, we give examples of effect plots produced by the **effects** package, and in the process systematically illustrate the optional arguments to functions in the package, which can be used to customize predictor effect plots.



1 Introduction

Predictor effect plots (Fox and Weisberg, 2018) provide graphical summaries for fitted regression models with linear predictors, including linear models, generalized linear models, linear and generalized linear mixed models, and many others. These graphs are an alternative to tables of fitted coefficients, which can be much harder to interpret than predictor effect plots. Predictor effect plots are implemented in R in the **effects** package, documented in Fox and Weisberg (2019). This vignette provides many examples of variations on the graphical displays that can be obtained with the **effects** package. Many of the details, and more complete descriptions of the data sets used as examples, are provided in the references cited at the end of the vignette.

1.1 Effects and Predictor Effect Plots

We begin with an example of a multiple linear regression, using the `Prestige` data set in the **carData** package:

```
R> library("car") # also loads the carData package
R> Prestige$type <- factor(Prestige$type, levels=c("bc", "wc", "prof"))
R> lm1 <- lm(prestige ~ education + poly(women, 2) +
+                          log(income)*type, data=Prestige)
```

The data, collected circa 1970, pertain to 102 Canadian occupations. The model `lm1` is a linear model with response `prestige`, continuous predictors `income`, `education`, and `women`, and the factor predictor `type`, which has three levels. Before fitting the model, we reorder the levels of `type` as `"bc"` (blue-collar), `"wc"` (white-collar), and `"prof"` (professional and managerial). The predictor `education` represents itself in the linear model, and so it is both a predictor and a *regressor*, as defined in Fox and Weisberg (2019, Sec. 4.1). The predictor `income` is represented by the regressor log(`income`). The variable `women`, a percentage between 0 and 100, is represented by regressors that define a polynomial of degree 2 using `poly()`'s default orthogonal polynomials. The variable `type` is a factor with three levels, so it is represented by two dummy regressors defined by the default contrast-generating function in R, `contr.treatment()`. Finally, the formula includes an interaction between `income` and `type`, defined by multiplying the regressor for `income` (log(`income`)) by each of the regressors that represent `type`.

The usual numeric summary of the fit of `lm1` is a table of estimated coefficients, which we obtain via the `S()` function in the **car** package that is similar to, but somewhat more flexible than, the standard R `summary()` function:

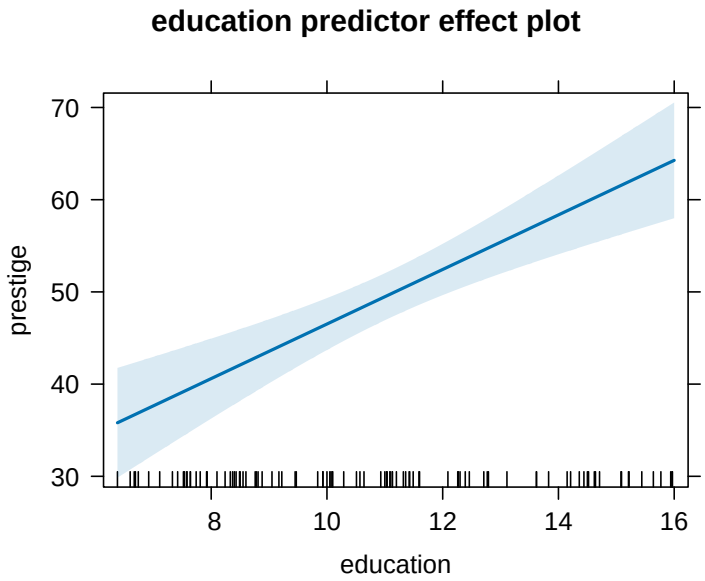
```
R> S(lm1)

Call: lm(formula = prestige ~ education + poly(women, 2) + log(income) * type,
         data = Prestige)

Coefficients:
                     Estimate Std. Error t value Pr(>|t|)
(Intercept)          -137.500     23.522   -5.85  8.2e-08
education               2.959      0.582    5.09  2.0e-06
poly(women, 2)1        28.339     10.190    2.78   0.0066
poly(women, 2)2        12.566      7.095    1.77   0.0800
log(income)            17.514      2.916    6.01  4.1e-08
typewc                  0.969     39.495    0.02   0.9805
typeprof               74.276     30.736    2.42   0.0177
log(income):typewc     -0.466      4.620   -0.10   0.9199
log(income):typeprof   -7.698      3.451   -2.23   0.0282

Residual standard deviation: 6.2 on 89 degrees of freedom
  (4 observations deleted due to missingness)
Multiple R-squared: 0.879
F-statistic: 81.1 on 8 and 89 DF,  p-value: <2e-16
   AIC    BIC
646.26 672.11
```

- Interpretation of the regression coefficients is straightforward only for the predictor `education`, where an increase of one year of `education`, holding other predictors fixed, corresponds to an estimated expected increase in the response of 2.959 units.
- Even ignoring the interaction, the log transformation complicates the interpretation of the effect of `income`.
- The predictor `women` is represented by two regressors, so the effect of `women` requires examining two coefficient estimates that are interpretable only by those knowledgeable about polynomial regression analysis. Even if raw rather than orthogonal polynomial regressors were used, via `poly(women, 2, raw=TRUE)` in place of `poly(women, 2)`, interpretation of the effect of `women` is complicated.
- Understanding the coefficients for the main effect of `type` depends on the contrasts used to define the effect. The contrasts can be changed by the user, and the default contrasts in R are different from the default contrasts used by SAS or other programs, so the coefficients cannot be reliably interpreted without information not present in the regression summary.
- Finally, the interaction further complicates the interpretation of the effect of either `income` or `type`, because the interaction coefficients need to be interpreted jointly with the main effect coefficients.

Summarization of the effects of predictors using tables of coefficient estimates is often incomplete. Effects, and particularly plots of effects, can in many instances reveal the relationship of the response to the predictors more clearly. This conclusion is especially true for models with linear predictors that include interactions and multiple-coefficient terms such as regression splines and polynomials, as illustrated in this vignette.

A predictor effect plot summarizes the role of a selected *focal* predictor in a fitted regression model. The `predictorEffect()` function is used to compute the appropriate summary of the regression, and then the `plot()` function may be used to graph the resulting object, as in the following example:

```
R> library("effects")
R> e1.lm1 <- predictorEffect("education", lm1)
R> plot(e1.lm1)
```



This graph visualizes the partial slope for `education`, that for each year increase in `education`, the fitted `prestige` increases by 2.959 points, when the other predictors are held fixed. The intercept of the line, which is outside the range of `education` on the graph, affects only the height of the line, and is determined by the choices made for averaging over the fixed predictors, but for any choice of averaging method, the slope of the line would be the same. The shaded area is a pointwise confidence band for the fitted values, based on standard errors computed from the covariance matrix of the fitted regression coefficients. The rug plot at the bottom of the graph shows the location of the `education` values.

The information that is needed to draw the plot is computed by the `predictorEffect()` function. The minimal arguments for `predictorEffect()` are the quoted name of a predictor in the model followed by the fitted model object. The essential purpose of this function is to compute fitted values from the model with `education` varying and all other predictors fixed at typical values (Fox and Weisberg, 2019, Sec. 4.3). The command below displays the values of the regressors for which fitted values are computed, including a column of 1s for the intercept:

```
R> brief(e1.lm1$model.matrix)

50 x 9 matrix (45 rows and 5 columns omitted)
   (Intercept) education . . . log(income):typewc log(income):typeprof
1            1      6.38                  2.0758                2.7979
2            1      6.58                  2.0758                2.7979
3            1      6.77                  2.0758                2.7979
. . .
49           1     15.80                  2.0758                2.7979
50           1     16.00                  2.0758                2.7979
```

The focal predictor `education` was evaluated by default at 50 points covering the observed range of values of `education`. We use the `brief()` function in the **car** package to show only a few of the 50 rows of the matrix. For each value of `education` the remaining regressors have the same fixed values for each fitted value. The fixed value for log(`income`) is the logarithm of the sample mean `income`, the fixed values for the regressors for `women` are computed at the mean of `women` in the data, and the fixed values for the regressors for `type` effectively take a weighted average of the fitted values

at the three levels of `type`, with weights proportional to the number of cases in each level of the factor. Differences in the fitted values are due to `education` alone because all the other predictors, and their corresponding regressors, are fixed. Thus the output gives the partial effect of `education` with all other predictors fixed.

The computed fitted values can be viewed by printing the `"eff"` object returned by `predictorEffect()`, by summarizing the object, or by converting it to a data frame. To make the printouts more compact, we recompute the predictor effect of `education` with fewer values of the focal predictor by specifying the `focal.levels` argument (see Section 2.1):

```
R> e1a.lm1 <- predictorEffect("education", lm1, focal.levels=5)
R> e1a.lm1


 education predictor effect

 education effect
education
   6.4    8.8     11     14     16
35.864 42.965 49.474 58.351 64.268

R> summary(e1a.lm1)


 education effect
education
   6.4    8.8     11     14     16
35.864 42.965 49.474 58.351 64.268

 Lower 95 Percent Confidence Limits
education
   6.4    8.8     11     14     16
29.930 39.334 46.923 54.079 57.989

 Upper 95 Percent Confidence Limits
education
   6.4    8.8     11     14     16
41.798 46.596 52.026 62.623 70.548

R> as.data.frame(e1a.lm1)

  education    fit     se  lower  upper
1       6.4 35.864 2.9865 29.930 41.798
2       8.8 42.965 1.8275 39.334 46.596
3      11.0 49.474 1.2842 46.923 52.026
4      14.0 58.351 2.1500 54.079 62.623
5      16.0 64.268 3.1604 57.989 70.548
```

The values in the column `education` are the values the focal predictor. The remaining columns are the fitted values, their standard errors, and lower and upper end points of 95% confidence intervals for the fitted values. The *predictor effect plot* is simply a graph of the fitted values on the vertical axis versus the focal predictor on the horizontal axis. For a continuous focal predictor such as `education`, a line, in this case, a straight line, is drawn connecting the fitted values.

We turn next to the predictor effect plot for `income`. According to the regression model, the effect of `income` may depend on `type` due to the interaction between the two predictors, so simply averaging over `type` would be misleading. Rather, we should allow both `income` and `type` to vary, fixing the other predictors at their means or other typical values. By default, this computation would

require evaluating the model at $50 \times 3 = 150$ combinations of the predictors, but to save space we will only evaluate `income` at five values, again using the `focal.levels` argument, thus computing only $5 \times 3 = 15$ fitted values:

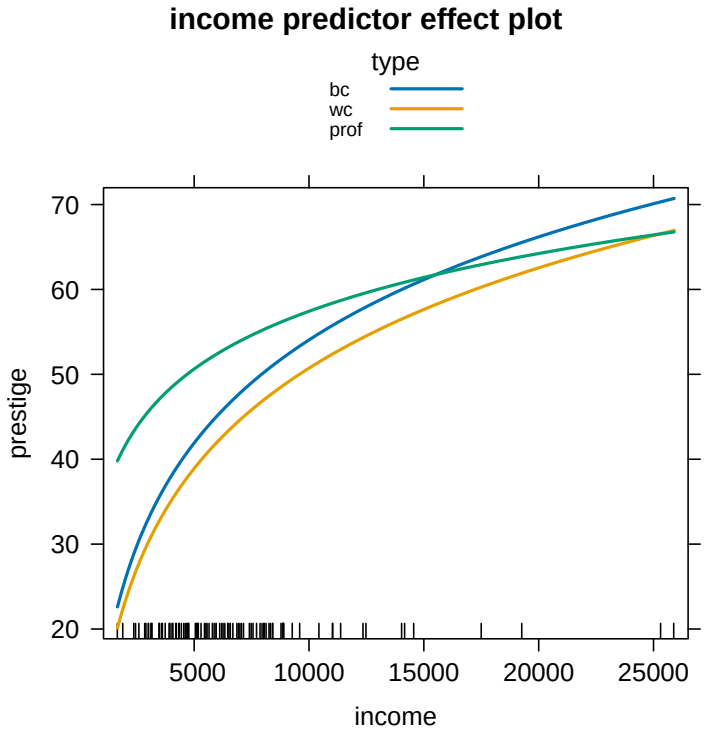
```
R> e2.lm1 <- predictorEffect("income", lm1, focal.levels=5)
R> as.data.frame(e2.lm1)
```

```
   income type    fit     se  lower  upper
1    2000   bc 25.863 3.3037 19.299 32.428
2    8000   bc 50.142 2.3737 45.426 54.859
3   10000   bc 54.050 2.7996 48.487 59.613
4   20000   bc 66.190 4.4814 57.285 75.094
5   30000   bc 73.291 5.5708 62.222 84.360
6    2000   wc 23.290 4.5674 14.214 32.365
7    8000   wc 46.922 2.3106 42.331 51.513
8   10000   wc 50.726 3.0575 44.651 56.802
9   20000   wc 62.543 5.7716 51.075 74.011
10  30000   wc 69.455 7.4432 54.665 84.244
11   2000 prof 41.630 4.4812 32.726 50.534
12   8000 prof 55.237 2.3316 50.605 59.870
13  10000 prof 57.428 2.4552 52.549 62.306
14  20000 prof 64.231 3.6170 57.045 71.418
15  30000 prof 68.211 4.5680 59.135 77.288
```

To draw the predictor effects plot we recalculate the fitted values using the default `focal.levels=50` to get more accurately plotted regression curves:

```
R> plot(predictorEffect("income", lm1),
+         lines=list(multiline=TRUE))
```

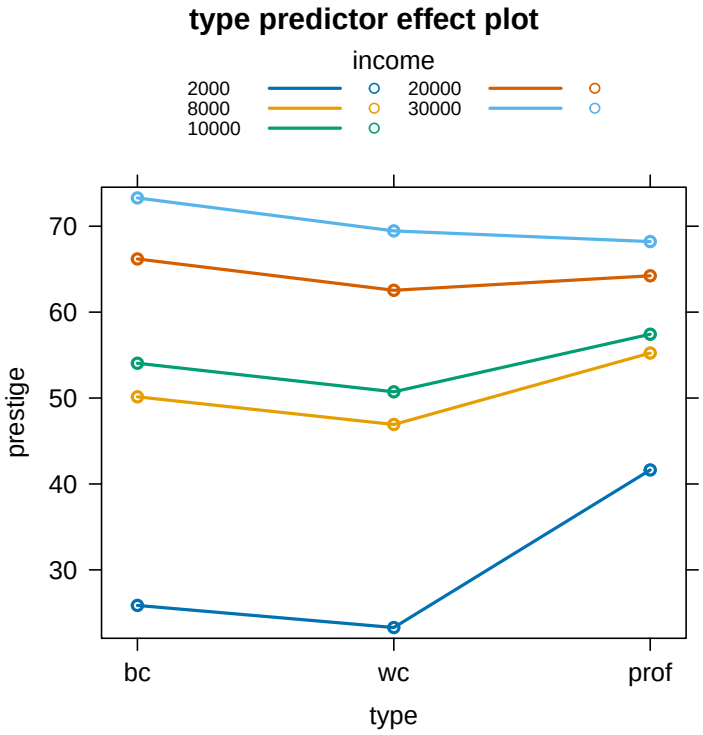
Here we use both the `predictorEffect()` and `plot()` functions in the same command.



The focal predictor `income` is displayed on the horizontal axis. There is a separate line shown for the fitted values at each level of `type`. The lines are curved rather than straight because `income` appears in the model in log-scale but is displayed in the predictor effect plot in arithmetic (i.e., dollar) scale. The lines in the graph are not parallel because of the interaction between log(`income`) and `type`. For `type` = `"prof"`, the fitted values of `prestige` are relatively high for lower values of `income`, and are relatively less affected by increasing values of `income`.

The predictor effect plot for `type` uses essentially the same fitted values as the plot for `income`, but we now get five lines, one for each of the five (not 50) values of `income` selected by the `predictorEffect()` function in this context:

```
R> plot(predictorEffect("type", lm1), lines=list(multiline=TRUE))
```



Because the horizontal axis is now a factor, the fitted values are displayed explicitly as points, and the lines that join the points are merely a visual aid representing *profiles* of fitted values. Fitted `prestige` increases with `income` for all levels of `type`, but, as we found before, when `type` = `"prof"`, fitted `prestige` is relatively high for lower `income`.

These initial examples use only default arguments for `predictorEffect()` and `plot()`, apart from the `multiline` argument to `plot()` to put all the fitted lines in the same graph. We explain how to customize predictor effect plots in subsequent sections of this vignette.

1.2 General Outline for Constructing Predictor Effect Plots

Using the **effects** package to draw plots usually entails the following steps:

1. Fit a regression model with a linear predictor. The package supports models created by `lm()`, `glm()`, `lmer()` and `glmer()` in the **lme4** package, `lme()` in the **nlme** package, and many other regression-modeling functions (see `?Effect`).

2. The regression model created in the first step is then used as input to either `predictorEffect()`, to get the effects for one predictor, or `predictorEffects`, to get effects for one or more predictors. These functions do the averaging needed to get fitted values that will ultimately be plotted. There are many arguments for customizing the computation of the effects.

The two predictor effect functions call the more basic `Effect()` function, and almost all of the material in this vignette applies to `Effect()` as well.

3. Use the generic `plot()` function to draw a graph or graphs based on the object created in Step 2.

1.3 How `predictorEffect()` Chooses Conditioning Predictors

Suppose that you select a *focal predictor* for which you want to draw a predictor effect plot. The `predictorEffect()` function divides the predictors in a model formula into three groups:

1. The focal predictor.
2. The *conditioning group*, consisting of all predictors with at least one interaction in common with the focal predictor.
3. The *fixed group*, consisting of all other predictors, that is, those with no interactions in common with the focal predictor.

For simplicity, let's assume for the moment that all of the fixed predictors are numeric. The predictors in the fixed group are all evaluated at *typical values*, usually their means, effectively averaging out the influence of these predictors on the fitted value. Fitted values are computed for all combinations of levels of the focal predictor and the predictors in the conditioning group, with each numeric predictor in the conditioning group replaced by a few discrete values spanning the range of the predictor, for example, replacing years of `education` by a discrete variable with the values 8, 12, and 16 years.

Suppose that we fit a model with R formula

```
y ~ x1 + x2 + x3 + x4 + x2:x3 + x2:x4
```
(1)

or, equivalently,

```
y ~ x1 + x2*x3 + x2*x4
```

There are four predictor effect plots for this model, one for each predictor selected in turn as the focal predictor:

Focal Predictor	Conditioning Group	Fixed Group
`x1`	none	`x2`, `x3`, `x4`
`x2`	`x3`, `x4`	`x1`
`x3`	`x2`	`x1`, `x4`
`x4`	`x2`	`x1` `x3`

The predictor `x1` does not interact with any of the other predictors, so its conditioning set is empty and all the remaining predictors are averaged over; `x2` interacts with both `x3` and `x4`; `x3` interacts only with `x2`; and `x4` interacts with `x2`.

1.4 The `Effect()` Function

Until recently, the primary function in **effects** for computing and displaying effects was the `Effect()` function.[1] Whereas the `predictorEffect()` function automatically determines the conditioning group and the fixed group of predictors, the `Effect()` function puts that burden on the user. The

[1]The **effects** package also includes the older `allEffects()` function, which computes effects for each high-order term in a model with a linear predictor. As we explain in Fox and Weisberg (2018), we prefer predictor effects to high-order term effects, and so, although its use is similar to `predictorEffects()`, we won't describe `allEffects()` in this vignette. There is also an older `effect()` function (with a lowercase "e"), which is a less flexible version of `Effect()`, and which calls `Effect()` to perform computations; `effect()` is retained only for backwards comparability.

`Effect()` function doesn't distinguish between between a focal predictor and conditioning predictors, but rather only between varying (that is, focal *and* conditioning) and fixed predictors.

Each call to `predictorEffect()` is equivalent to a specific call to the `Effect()` function as follows. Suppose that `m` is the fitted model produced by the formula in (1); then, except for the ways in which the default levels for predictors are determined:

`predictorEffect("x1", m)` is equivalent to `Effect("x1", m)`;

`predictorEffect("x2", m)` is equivalent to `Effect(c("x2", "x3", "x4"), m)`;

`predictorEffect("x3", m)` is equivalent to `Effect(c("x3", "x2"), m)`; and

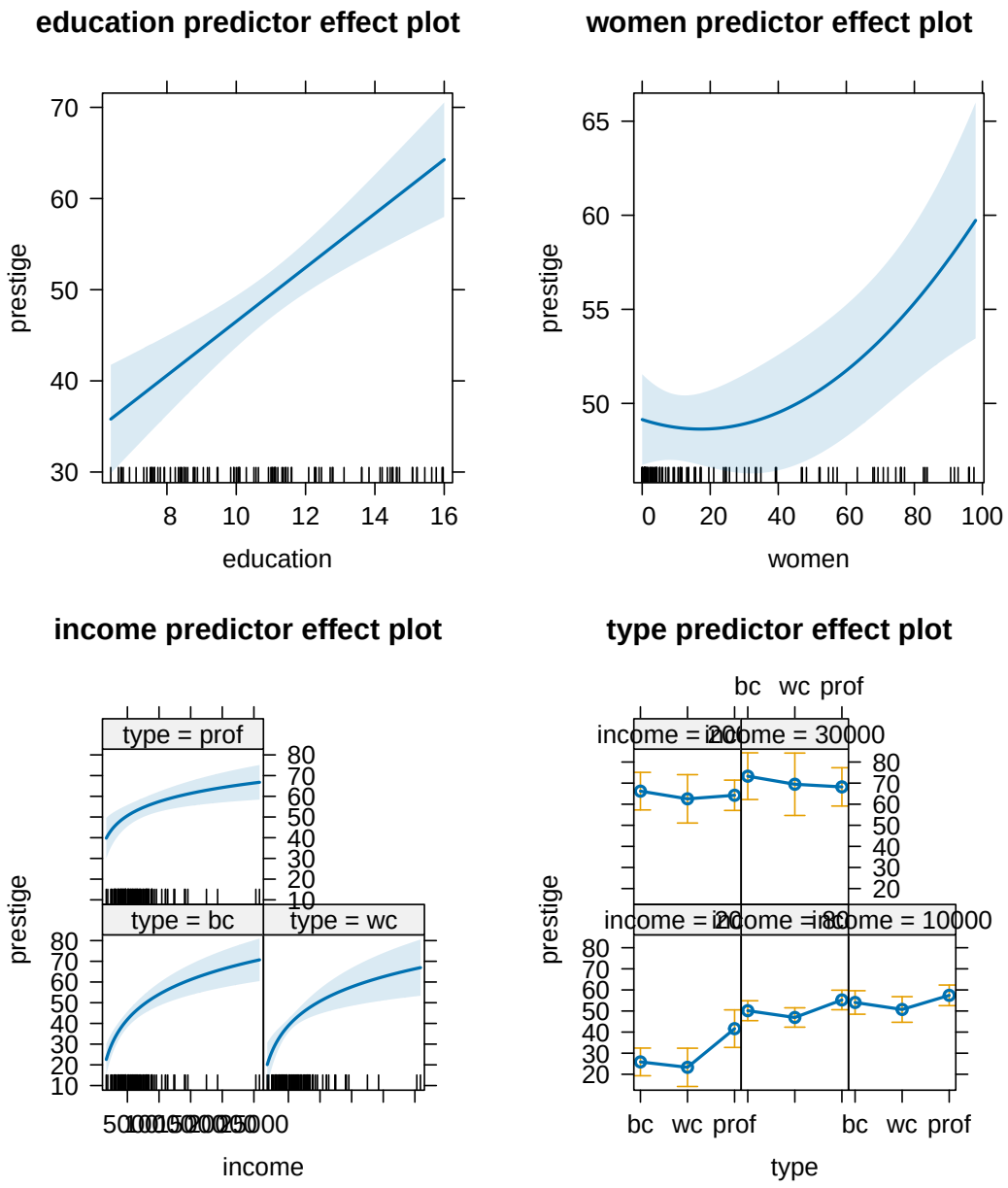
`predictorEffect("x4", m)` is equivalent to `Effect(c("x4", "x2"), m)`.

The `predictorEffect()` function determines the correct call to `Effect()` based on the choice of focal predictor and on the structure of main effects and interactions in the linear predictor for the model. It then uses the `Effect()` function to do the computing. As a result, most of the arguments to `predictorEffect()` are documented in `help("Effect")` rather than in `help("predictorEffect")`.

1.5 The `predictorEffects()` Function

This function, whose name ends with the plural "`effects`", computes the values needed for one or more predictor effect plots, and by default for *all* of the predictors in the model. For example, the following command produces all of the predictor effect plots for the model we fit to the `Prestige` data:

```
R> eall.lm1 <- predictorEffects(lm1)
R> plot(eall.lm1)
```



The predictor effect plots for this model are displayed in an array of graphs. The plots for `income` and `type` have a separate panel for each level of the conditioning variable because the default argument `lines=list(multiline=FALSE)` was implicitly used. Confidence bounds are shown by default when `multiline=FALSE`.

The resulting object `eall.lm1` is a list with four elements, where `eall.lm1[[1]]` is the summary for the first predictor effect plot, `eall.lm1[[2]]` for the second plot, and so on. The following equivalent commands draw the same array of predictor effect plots:

```
R> plot(eall.lm1)
R> plot(predictorEffects(lm1))
R> plot(predictorEffects(lm1, ~ income + education + women + type))
```

If you want only the predictor effect plots for `type` and `education`, in that order, you could enter

```
R> plot(predictorEffects(lm1, ~ type + education))
```

Similarly, the commands

```
R> plot(predictorEffects(lm1, ~ women))
R> plot(predictorEffects(lm1)[[2]])
R> plot(predictorEffect("women", lm1))
```

all produce the same graph, the predictor effect plot for `women`.

Predictor effect plots in an array can be a useful shortcut for drawing many graphs quickly, but can lead to problems with the displayed graphs. For example, the horizontal axis labels for the plot for `income` are overprinted, and the labels at the top of the panels for `type` with conditioning variable `income` are larger than the available space. These problems can often be fixed using optional arguments described later in this vignette or by plotting predictor effects individually.

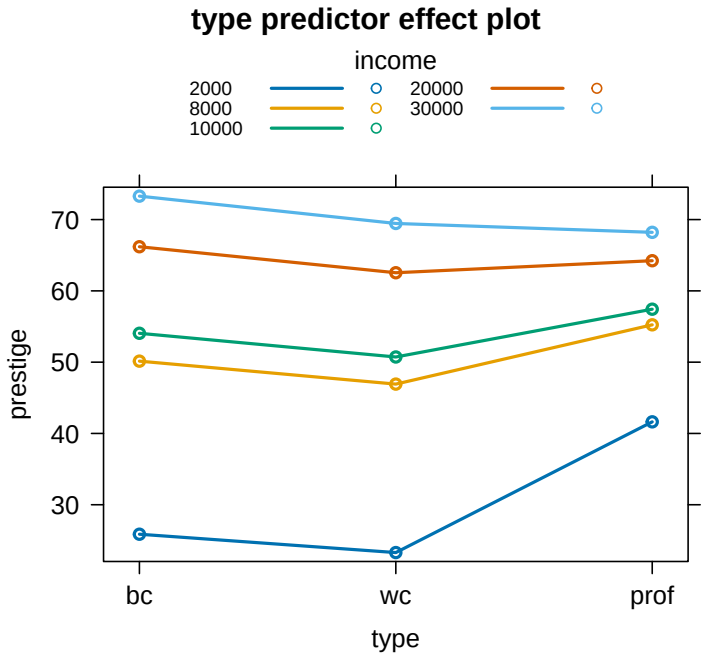
2 Optional Arguments for the `predictorEffect()` and `Effect()` Functions

This section comprises a catalog of the arguments available to modify the behavior of the `predictorEffect()` and `Effect()` functions. These arguments may also be specified to the `predictorEffects()` function. The information provided by `help("Effect")` is somewhat more comprehensive, if terser, explaining for example exceptions applying to `"svyglm"` objects or for plotting residuals.

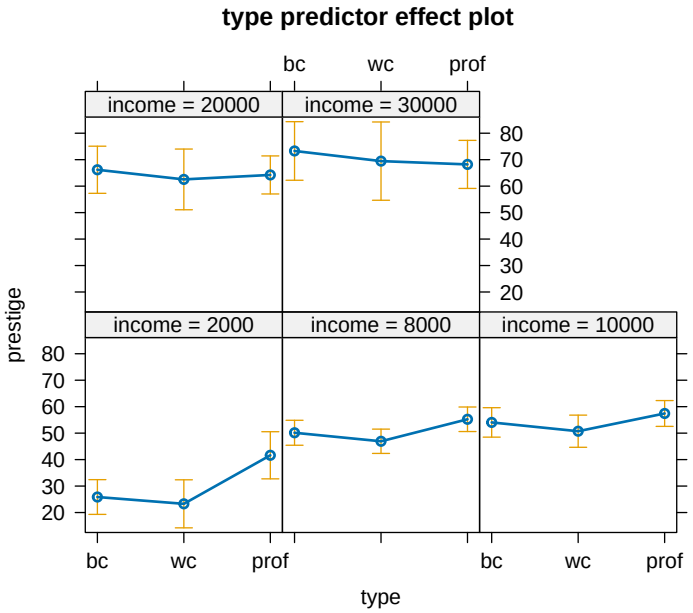
2.1 `focal.levels` and `xlevels`: Options for the Values of the Focal Predictor and Predictors in the Conditioning Group

Numeric predictors in the conditioning group need to be discretized to draw a predictor effect plot. For example the predictor effect plot for `type` in model `lm1` consists of a separate line, or a separate panel, for each discrete value of `income`:

```
R> e3.lm1 <- predictorEffect("type", lm1)
R> plot(e3.lm1, lines=list(multiline=TRUE))
```



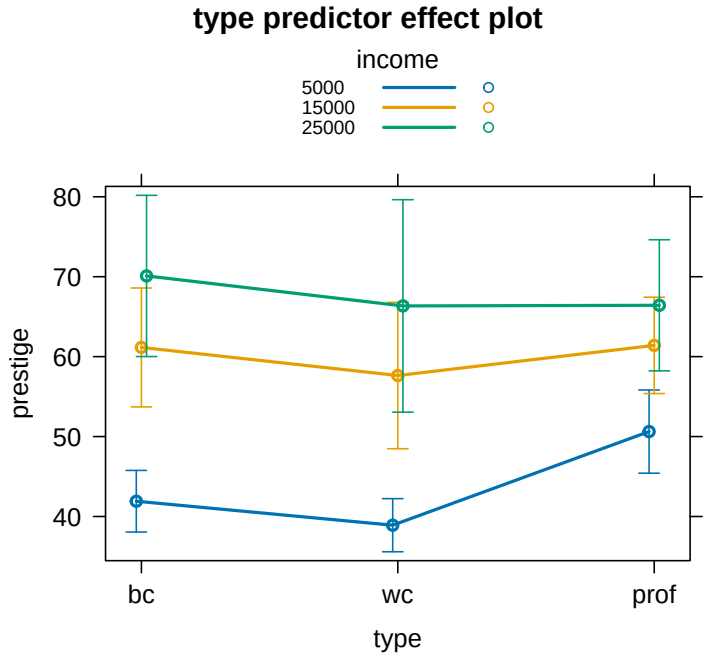

```
R> plot(e3.lm1, lines=list(multiline=FALSE)) # the default
```



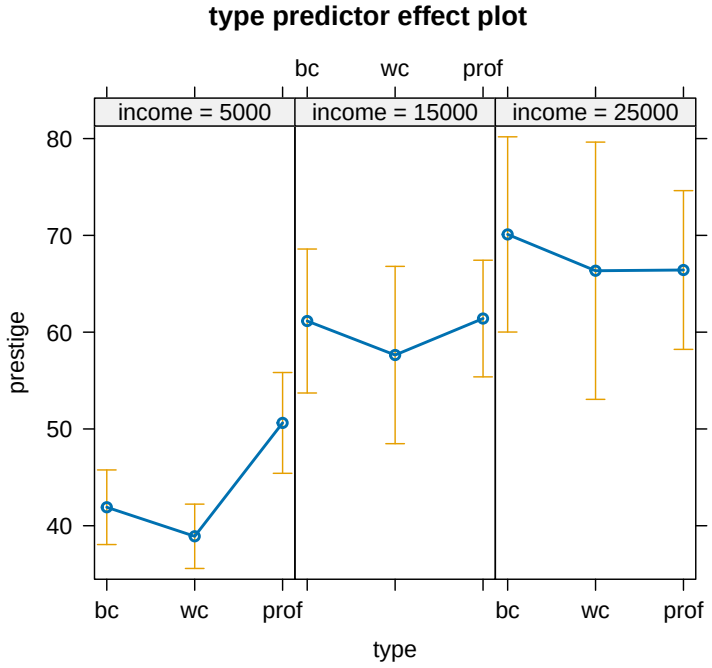
The numeric conditioning predictor `income` is evaluated by default at five equally spaced values, when are then rounded to "nice" numbers.

Using the three values of 5000, 15000, 25000 for the conditioning predictor `income` in this example produces a simpler graph:

```
R> e3.lm1 <- predictorEffect("type", lm1,
+                            xlevels=list(income=c(5000, 15000, 25000)))
R> plot(e3.lm1, lines=list(multiline=TRUE),
+         confint=list(style="bars"))
```




```
R> plot(e3.lm1,
+         lines=list(multiline=FALSE), # the default
+         lattice=list(layout=c(3, 1)))
```



The argument `xlevels` is a list of sub-arguments that control how numeric predictors are discretized when used in the conditioning group. For example, `xlevels=list(x1=c(2, 4, 7), x2=6)` would use the values 2, 4, and 7 for the levels of the predictor `x1`, use 6 equally spaced values for the predictor `x2`, and use the default of 5 values for any other numeric conditioning predictors. Numeric predictors in the *fixed* group are not affected by the `xlevels` argument. We use the `layout` sub-argument of the `lattice` argument group to arrange the panels of the second graph in 3 columns and 1 row (see Section 3.4.2). See `help("plot.eff")` for information on the `quantiles` argument, which provides an alternative method of setting `xlevels` when partial residuals are displayed, as discussed in Section 4.

The points at which a numeric focal predictor is evaluated is controlled by the `focal.levels` argument. The default of `focal.levels=50` is recommended for drawing graphs, but if the goal is to produce a table of fitted values a smaller value such as `focal.levels=5` produces more compact output. The focal predictor can also be set to a vector of particular values, as in `focal.levels=c(30, 50, 70)`. Used with the `predictorEffects` function, the `focal.levels` argument can be set separately for each focal predictor, similarly to the `xlevels` argument; see `help("predictorEffects")`.

2.2 `fixed.predictors`: Options for Predictors in the Fixed Group

Predictors in the fixed group are replaced by "typical" values of the predictors. Fitted values are then computed using these typical values for the fixed group, varying the values of predictors in the conditioning group and of the focal predictor. The user can control how the fixed values are determined by specifying the `fixed.predictors` argument. This argument takes a list of sub-arguments that allow for controlling each predictor in the fixed group individually, with different rules for factors and numeric predictors.

2.2.1 Factor Predictors

Imagine computing the fitted values evaluating a fixed factor at each of its levels. The fitted value that is used in the predictor effects plot is a weighed average of these within-level fitted values, with weights proportional to the number of observations at each level of the factor. This is the default approach, and is an appropriate notion of "typical" for a factor if the data at hand are viewed as a random sample from a population, and so the sample fraction at each level estimates the population

fraction.

A second approach is to average the level-specific fitted values with equal weights at each level. This may be appropriate, for example, in designed experiments in which the levels of a factor are assigned by an investigator. The latter method is invoked by setting `fixed.predictors=list(given.values="equal")`.

You can construct other weighting schemes for averaging over the levels of a factor, as described on the help page for the `Effect()` function.

2.2.2 Numeric Predictors

For a numeric predictor in the fixed group the default method of selecting a typical value is to apply the `mean()` function to the data for the predictor. The specification `fixed.predictors=list(typical=median)` would instead use the `median()` function; in general, `typical` can be any function that takes a numeric vector as its argument and returns a single number.

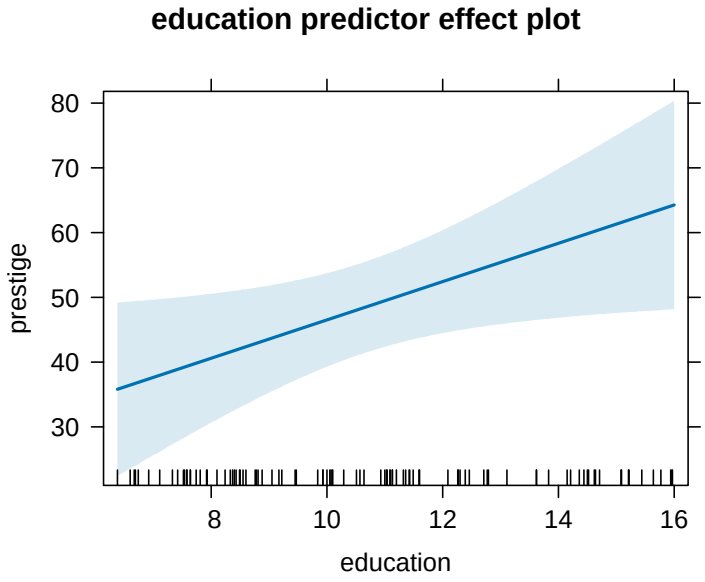
Other sub-arguments to `fixed.predictors` apply to the use of offsets, and to the **survey** package; see the help page for the `Effect()` function.

2.3 `se` and `vcov.`: Standard Errors and Confidence Intervals

Standard errors and confidence intervals for fitted values are computed by default, which corresponds to setting the argument `se=list(compute=TRUE, type="pointwise", level=.95)`. Setting `se=FALSE` omits standard errors, `type="scheffe"` uses wider Scheffé intervals that adjust for simultaneous inference, and `level=.8`, for example, produces 80% intervals.

Standard errors are based by default on the "usual" sample covariance matrix of the estimated regression coefficients. You can replace the default coefficient covariance matrix with some other estimate, such as one obtained from the bootstrap or a sandwich coefficient covariance matrix estimator, by setting the `vcov.` argument either to a function that returns a coefficient covariance matrix, such as `hccm()` in the **car** package for linear models, or to a matrix of the correct size; for example:

```
R> e4.lm1 <- predictorEffect("education", lm1,
+                            se=list(type="scheffe", level=.99),
+                            vcov.=hccm)
R> plot(e4.lm1)
```



This plot displays 99% Scheffé intervals based on a robust coefficient covariance matrix computed by the sandwich method; see `help("hccm")`.

2.4 `residuals`: Computing Residuals for Partial Residual Plots

The argument `residuals=TRUE` computes and saves residuals, providing the basis for adding partial residuals to subsequent effect plots, a topic that we discuss in Section 4.

3 Arguments for Plotting Predictor Effects

The arguments described in Section 2 are for the `predictorEffect()` function or the `Effect()` function. Those arguments modify the computations that are performed, such as methods for averaging and fixing predictors, and for computing standard errors. Arguments to the `plot()` methods for the predictor effect and effect objects produced by the `predictorEffect()` and `Effect()` functions are described in this section, and these change the appearance of a predictor effect plot or modify the quantities that are plotted. These optional arguments are described in more detail in `help("plot.eff")`.

In 2018, we reorganized the `plot()` method for effect objects by combining arguments into five major groups of related sub-arguments, with the goal of simplifying the specification of effect plots. For example, the `lines` argument group is a list of sub-arguments for determining line type, color, and width, whether or not multiple lines should be drawn on the same graph, and whether plotted lines should be smoothed. The defaults for these sub-arguments are the choices we generally find the most useful, but they will not be the best choices in all circumstances. The cost of reorganizing the arguments in this manner is the necessity of specifying arguments as lists, some of whose elements are themselves lists, requiring the user to make sure that parentheses specifying the possibly nested lists are properly balanced.

In addition to the five argument groups that we describe below, the `plot()` method for effect objects accepts the arguments `main` for the main title of the graph and `id` for identifying points in effect plots that include residuals, as discussed in Section 4.

Finally, the `plot()` method for effect objects retains a number of "legacy" arguments shown in `help("plot.eff")`. These arguments have been kept so existing scripts using the **effects** package would not break, but they are all duplicated as sub-arguments of the five argument groups. The legacy arguments work but they may not be supported forever, so we encourage you to use the newer argument groups and sub-arguments.

3.1 The `axes` Group: Specify Axis Characteristics

The `axes` argument group has two major sub-arguments, `x` for the horizontal axis, `y` for the vertical axis, and two minor sub-arguments, the `grid` argument, which adds a background grid to the plot, and the `alternating` argument, for changing the placement of axis-tick labels in multi-panel plots.

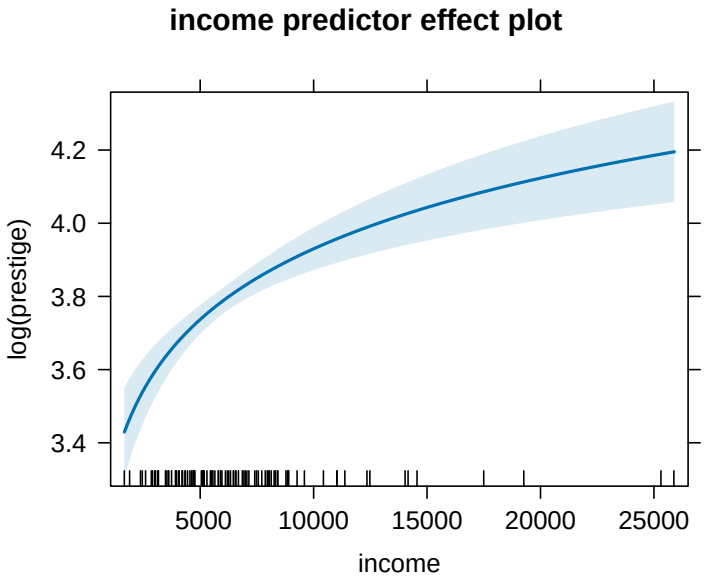
3.1.1 `x`: Horizontal Axis Specification

We introduce another linear model fit to the `Prestige` data set to serve as an example:

```
R> lm2 <- lm(log(prestige) ~ log(income) + education + type, Prestige)
```

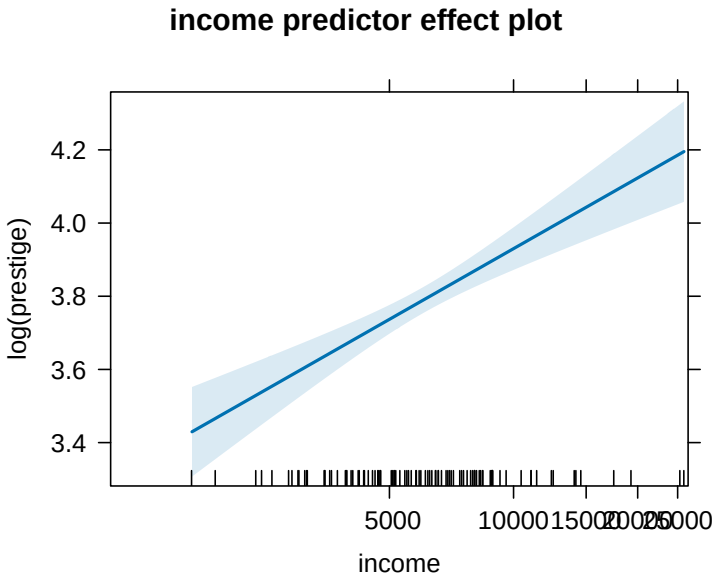
The default predictor effect plot for `income` is

```
R> plot(predictorEffects(lm2, ~ income))
```



The plot is curved because the predictor `income` is represented by its logarithm in the model formula, but the default predictor effect plot uses the predictor `income`, not the regressor log(`income`), on the horizontal axis. The `x` sub-argument can be used transform the horizontal axis, for example to replace `income` by log(`income`):

```
R> plot(predictorEffects(lm2, ~ income),
+     axes=list(
+       x=list(income=list(transform=list(trans=log, inverse=exp)))
+       ))
```

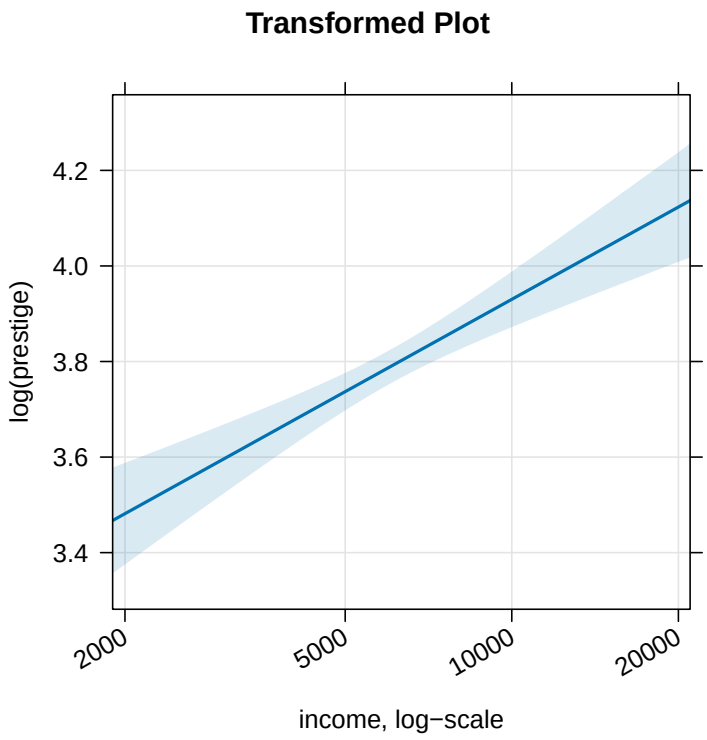


The transformation changes the scale on the horizontal axis to log-scale, but leaves the tick labels in arithmetic scale, and the graph is now a straight line because of the change to log-scale. This plot has several obviously undesirable features with regard to the range of the horizontal axis and over-printing of tick marks. We show next that additional arguments to `plot()` can correct these defects.

A more elaborate version of the graph illustrates all the sub-arguments to `x` in `axis` argument

group:

```
R> plot(predictorEffects(lm2, ~ income),
+     main="Transformed Plot",
+     axes=list(
+        grid=TRUE,
+        x=list(rotate=30,
+               rug=FALSE,
+               income=list(transform=list(trans=log, inverse=exp),
+                           lab="income, log-scale",
+                           ticks=list(at=c(2000, 5000, 10000, 20000)),
+                           lim=c(1900, 21000))
+        )))
```



We use the top-level argument `main="Transformed Plot"` to set the title of the plot. The `axes` argument is a list with two sub-arguments, `grid` to turn on the background grid, and `x` to modify the horizontal axis.

The `x` sub-argument is itself a list with three elements: The sub-arguments `rotate` and `rug` set the rotation angle for the tick labels and suppress the rug plot, respectively. The additional sub-argument is a list called `income`, the name of the focal predictor. If you were drawing many predictor effect plots you would supply one list named for each of the focal predictors. All of the sub-arguments for `income` are displayed in the example code above. The sub-argument `transform=list(trans=log, inverse=exp)` specifies how to transform the x-axis. The `ticks` and `lim` sub-arguments set the tick marks and range for the horizontal axis.

This is admittedly a complex command, but it allows you to fine-tune the graph to look the way you want. In specifying nested argument lists, you may encounter problems getting the parentheses in the right places. Be careful, indent your code to clarify the structure of the command, and be patient!

3.1.2 x: Horizontal Axis Specification for Date Variables

The functions in the **effects** package, such as `Effect()` and `predictorEffect()`, support models with numeric, factor, character, and logical predictors. Date predictors must be converted to numeric for these functions to work.

We supply the generic function `levels2dates()`, with methods for `"eff"` and `"effpoly"` objects, which can be used to properly label the horizontal axes of effect and predictor effect plots by translating numeric dates back to dates for the axis tick-mark labels. `levels2dates()` takes several arguments:

`effect` An `"eff"` or `"effpoly"` object, created, e.g., by `Effect()` or `predictorEffect()`.

`predictor` The quoted name of the numeric version of the date predictor.

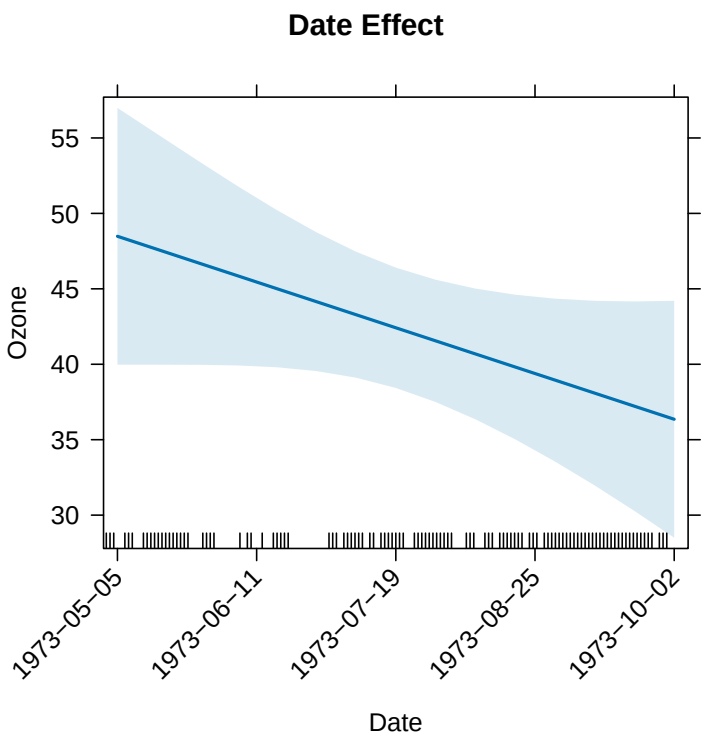
`origin` A quoted string giving the origin date (see the examples below).

`evenly.spaced` If `TRUE` (the default), the tick marks on the horizontal axis are evenly spaced; if `FALSE` the tick marks are taken from the levels of the numeric date predictor in the `"eff"` or `"effpoly"` object.

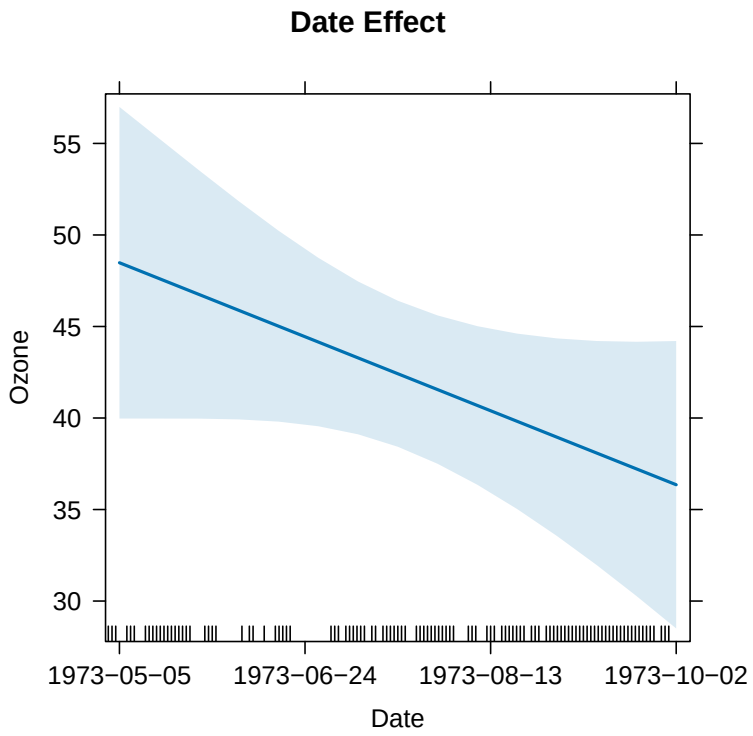
`n` The number of tick marks; if missing, the number of levels of the predictor in the `"eff"` or `"effpoly"` object.

Here are some examples:

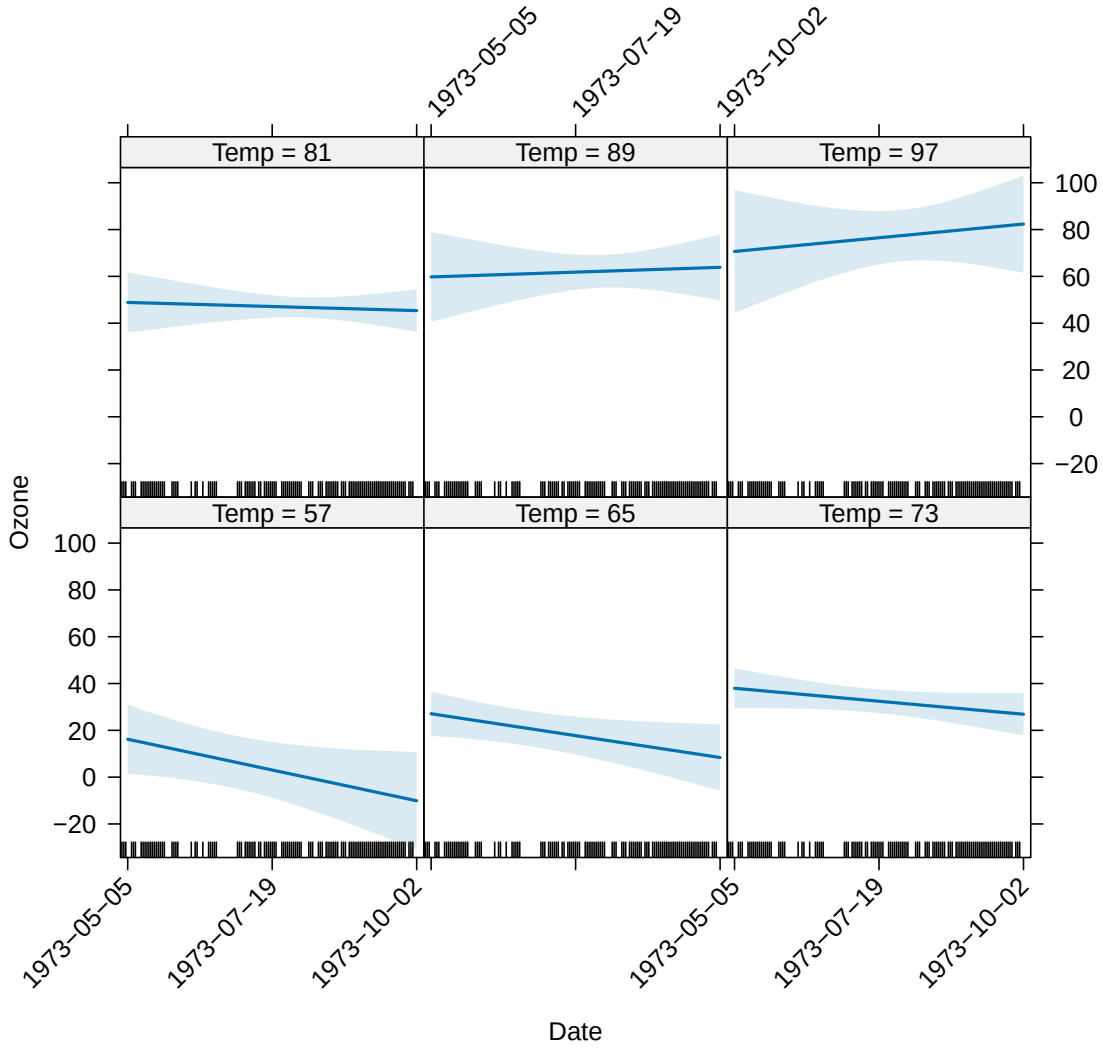
```
R> data("airquality", package="datasets")
R> airquality$Date <- with(airquality, as.Date(paste("1973", Month, Day, sep="-"),
+        format="%Y-%m-%d"))
R> airquality$Date.num <- as.numeric(airquality$Date)
R> m1.date <- lm(Ozone ~ Date.num + Solar.R + Wind + Temp, data=airquality)
R> eff.date.1 <- Effect("Date.num", m1.date)
R> plot(eff.date.1, axes=list(x=list(Date.num=list(lab="Date",
+        ticks=list(at=levels2dates(eff.date.1, "Date.num", "1970-01-01"))),
+        rotate=45)), main="Date Effect")
```



```
R> plot(eff.date.1, axes=list(x=list(Date.num=list(lab="Date",
+        ticks=list(at=levels2dates(eff.date.1, "Date.num", "1970-01-01", n=4))))),
+        main="Date Effect")
```




```
R> eff.date.df <- as.data.frame(eff.date.1)
R> eff.date.df$Date <- as.Date(eff.date.df$Date.num, origin="1970-01-01")
R> eff.date.df

  Date.num    fit     se  lower  upper       Date
1     1220 48.482 4.2933 39.970 56.994 1973-05-05
2     1250 46.057 3.0874 39.936 52.178 1973-06-04
3     1290 42.822 2.0396 38.779 46.866 1973-07-14
4     1330 39.588 2.4922 34.647 44.529 1973-08-23
5     1370 36.354 3.9606 28.502 44.206 1973-10-02

R> m2.date <- lm(Ozone ~ Date.num*Temp + Solar.R + Wind, data=airquality)
R> eff.date.2 <- Effect(c("Date.num", "Temp"), m2.date, xlevels=6)
R> plot(eff.date.2, axes=list(x=list(Date.num=list(lab="Date",
+      ticks=list(at=levels2dates(eff.date.2, "Date.num", "1970-01-01", n=3))),
+      rotate=45)), main="Date Effect by Temperature")
```



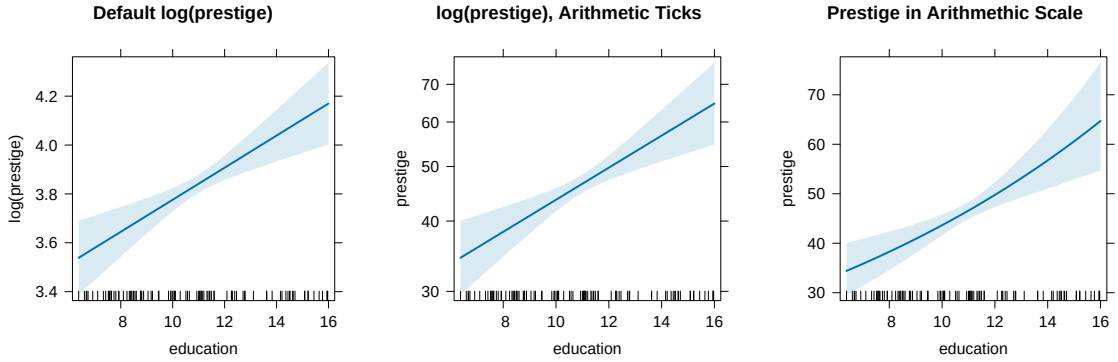
3.1.3 y: Vertical Axis Specification for Linear Models

The model `lm2` has a transformed response log(`prestige`), and "untransforming" the response to arithmetic scale may be desirable. This can be accomplished with the `y` sub-argument, which has two sub-arguments named `transform` and `type` that together control the scale and labeling of the vertical axis.

There are three options for drawing the predictor effect plot for a numeric response like log(`prestige`):

```
R> # default:
R> plot(predictorEffects(lm2, ~ education),
+         main="Default log(prestige)")

R> # Change only tick-mark labels to arithmetic scale:
R> plot(predictorEffects(lm2, ~ education),
+         main="log(prestige), Arithmetic Ticks",
+         axes=list(y=list(transform=list(trans=log, inverse=exp),
+                          lab="prestige", type="rescale")))
```

```
R> # Replace log(presige) by prestige:
R> plot(predictorEffects(lm2, ~ education),
+         main="Prestige in Arithmethic Scale",
+         axes=list(y=list(transform=exp, lab="prestige")))
```



The first plot is the default, with a log-response. In the second plot, the `transform` sub-argument specifies the transformation of the response and its inverse, and the sub-argument `type="rescale"` changes the tick marks on the vertical axis to arithmetic scale. In the third version, with `transform=exp, lab="prestige"`, the vertical axis now is in arithmetic scale, not log scale, although that may not be completely obvious in the example because $\log(x)$ is nearly linear: Look closely to see that the axis ticks marks in the second graph are unequally spaced, while those in the third graph are equally spaced and the plotted line in the latter is slightly curved. The help page `?plot.eff` provides a somewhat more detailed explanation of these options.

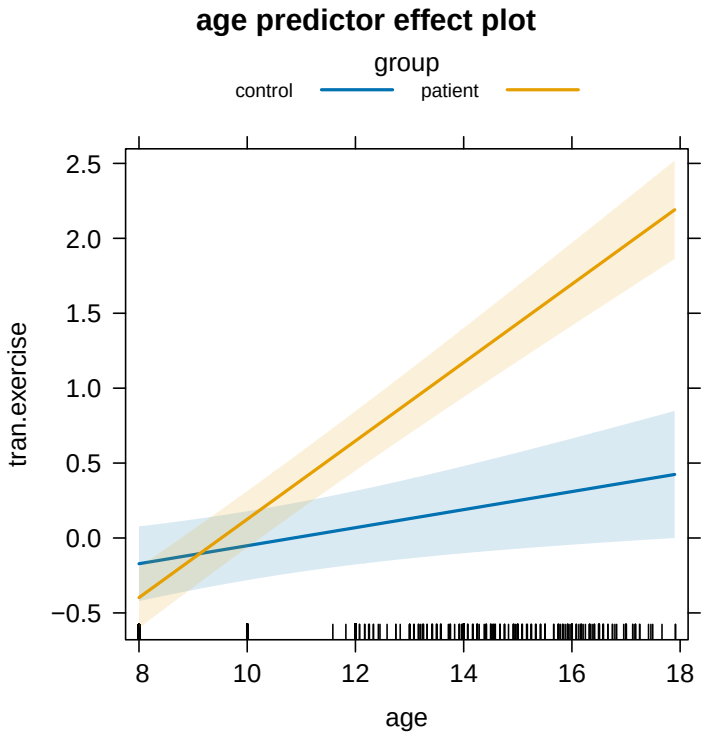
As a second example we will reconstruct Figure 7.10 in Fox and Weisberg (2019, Sec. 7.2). In that section, we fit a linear mixed-effects model to data from the `Blackmore` data frame in the **carData** package. `Blackmore` includes longitudinal data on amount of exercise for girls hospitalized for eating disorders and for similar control subjects who were not hospitalized. We transformed the response variable in the model, hours of `exercise`, using a transformation in a modified Box-Cox power family that allows zero or negative responses, explained briefly by Fox and Weisberg (2019, Sec. 3.4) and more thoroughly by Hawkins and Weisberg (2017). The fitted model is

```
R> library("lme4") # for lmer()

Loading required package: Matrix

R> Blackmore$tran.exercise <- bcnPower(Blackmore$exercise,
+                                      lambda=0.25, gamma=0.1)
R> mm1 <- lmer(tran.exercise ~ I(age - 8)*group +
+                  (I(age - 8) | subject), data=Blackmore)
```

This model, with numeric predictor `age` and factor predictor `group`, is a linear mixed model with random intercepts and slopes for `age` that vary by `subject`. The response variable is a transformation of `exercise` similar to the fourth root with adjustment for zero values; see `help("bcnPower")`.

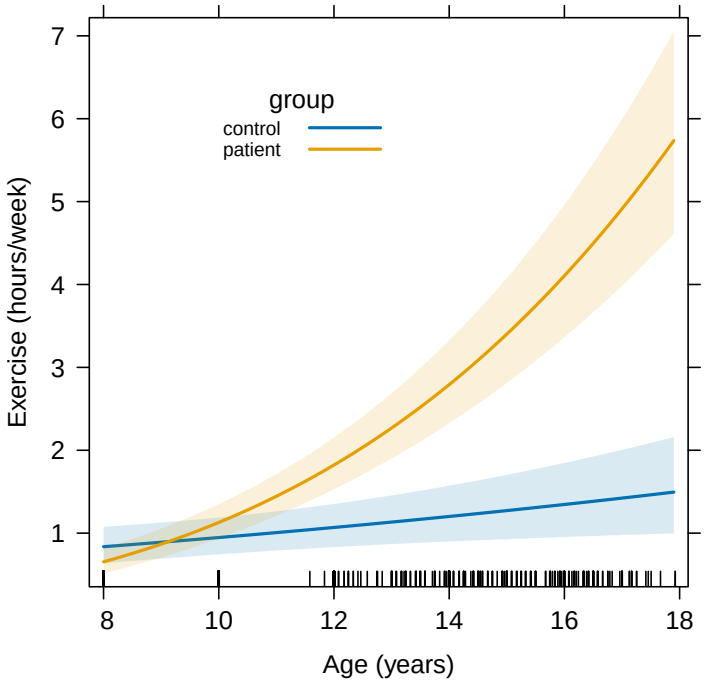
The predictor effect plot for the fixed effect of `age` is

```
R> e1.mm1 <- predictorEffect("age", mm1)
R> plot(e1.mm1, lines=list(multiline=TRUE), confint=list(style="auto"))
```



The plot clearly shows the difference in the average `age` trajectory between the `"control"` and `"patient"` groups, with the fitted response for the latter having a larger slope. The graph is hard to decode, however, because the vertical axis is approximately in the scale of the fourth-root of hours of exercise, so untransforming the response may produce a more informative plot. Because the `bcnPower()` transformation is complex, the **car** package includes the function `bcnPowerInverse()` to reverse the transformation:

```
R> f.trans <- function(x) bcnPower(x, lambda=0.25, gamma=0.1)
R> f.inverse <- function(x) bcnPowerInverse(x, lambda=0.25, gamma=0.1)
R> plot(e1.mm1, lines=list(multiline=TRUE),
+         confint=list(style="auto"),
+         axes=list(x=list(age=list(lab="Age (years)")),
+                   y=list(transform=list(trans=f.trans, inverse=f.inverse),
+                          type="response",
+                          lab="Exercise (hours/week)")),
+         lattice=list(key.args=list(x=.20, y=.75, corner=c(0, 0),
+                                    padding.text=1.25)),
+         main=""
+    )
```



The response scale is now in hours per week, and we see that hours of exercise increase more quickly on average in the patient group for older subjects. We use additional arguments in this plot to match Fox and Weisberg (2019, Fig. 7.10), including moving the key inside of the graph (see Section 3.4.1), changing the axis labels, and removing the main title to the plot.[2]

3.1.4 `y`: Vertical Axis Specification for Generalized Linear Models

Transforming the vertical axis for generalized linear models also uses the `y` sub-argument to the `axes` argument. You typically do not need to specify the `transform` sub-argument because `plot()` obtains the right functions from the regression model's `family` component. The `type` sub-argument has the same three possible values as for linear models, but their interpretation is somewhat different:

1. Predictor effect plots in `type="link"` scale have a predictor on the horizontal axis and the vertical axis is in the scale of the linear predictor. For logistic regression, for example, the vertical axis is in log-odds (logit) scale. For Poisson regression with the log-link, the vertical axis is in log-mean (log-count) scale.

2. Predictor effect plots in `type="response"` or mean scale are obtained by "untransforming" the y axis using the inverse of the link function. For the log-link, this corresponds to transforming the y axis and plotting $\exp(y)$. For logistic regression, $y = \log[p/(1-p)]$ and, solving for p, $p = \exp(y)/[1+\exp(y)] = 1/[1+\exp(-y)]$, so the plot in mean scale uses $1/[1+\exp(-y)]$ on the vertical axis.

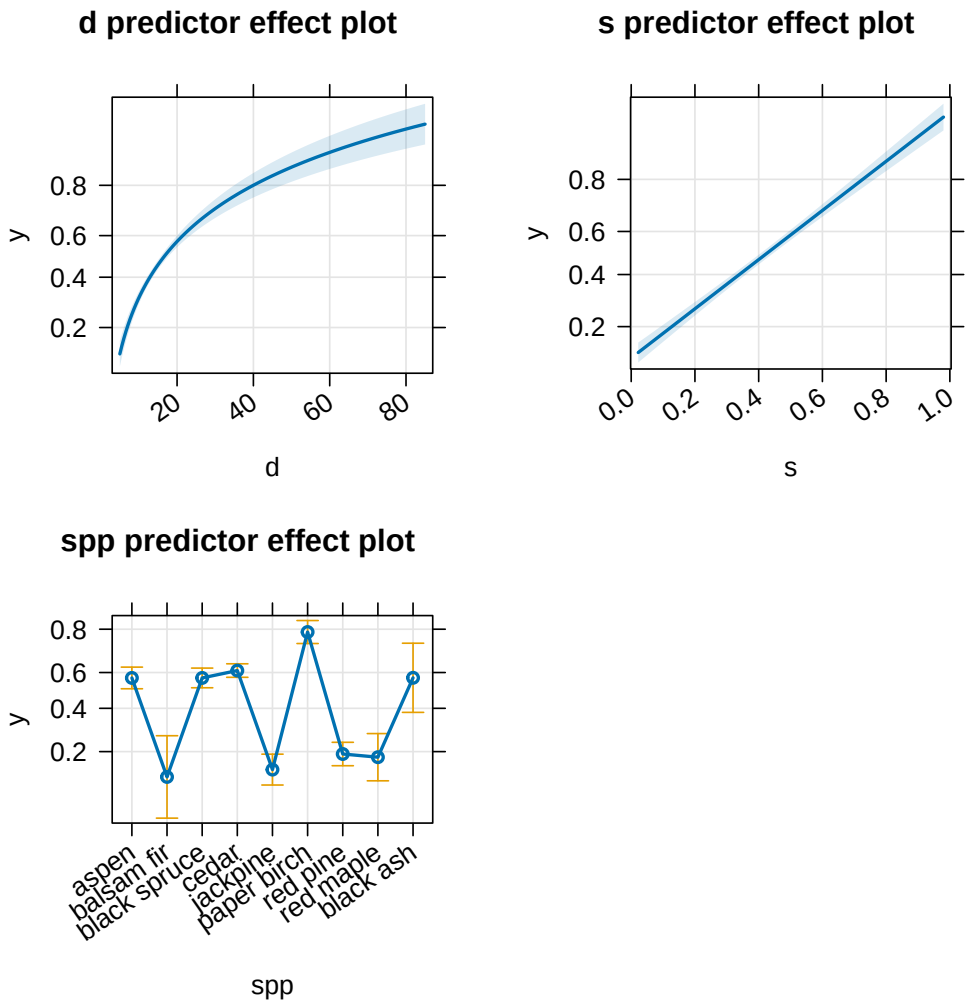
3. We also provide a third option, `type="rescale"`, which plots in linear predictor (e.g., logit) scale, but labels the tick marks on the vertical axis in mean (e.g., probability) scale. This third option, which retains the linear structure of the model but labels the vertical axis on the usually more familiar mean scale, is the default.

We use the `Blowdown` data from the **alr4** package to provide examples. These data concern the probability of *blowdown* `y`, a tree being uprooted as the result of a major straight-line wind storm

[2]The code shown for this graph in Fox and Weisberg (2019) uses "legacy" arguments, and is therefore somewhat different from the code given here. Both commands produce the same plot, however.

in the Boundary Waters Canoe Area Wilderness in 1999, modeled as a function of the diameter `d` of the tree, the local severity `s` of the storm, and the species `spp` of the tree. We fit a main-effects model and then display all three predictor effect plots:

```
R> data("Blowdown", package="alr4")
R> gm1 <- glm(y ~ log(d) + s + spp, family=binomial, data=Blowdown)

R> plot(predictorEffects(gm1),
+         axes=list(grid=TRUE, x=list(rug=FALSE, rotate=35)))
```



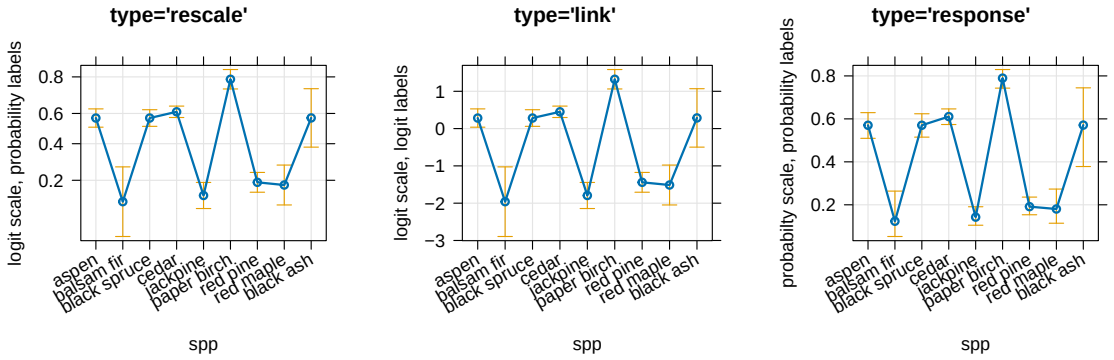
The `rug=FALSE` sub-argument to `x` suppresses the rug plot that appears by default at the bottom of graphs for numeric predictors, and the `grid` sub-argument to `axes` adds background grids. The `rotate` sub-argument prints the horizontal tick labels at an angle to avoid overprinting.

Interpretation of GLM predictor effect plots in link scale is similar to predictor effect plots for linear models, and all the modifications previously described can be used for these plots. Because the default is `type="rescale"`, the vertical axis is in linear predictor scale, which is the log-odds or logit for this logistic regression example, but the vertical axis labels are in mean (probability) scale, so the tick-marks are not equally spaced.

The next three graphs illustrate the possible values of the argument `type`:

```
R> e1.gm1 <- predictorEffect("spp", gm1)
R> plot(e1.gm1, main="type='rescale'",
+         axes=list(y=list(type="rescale",
```

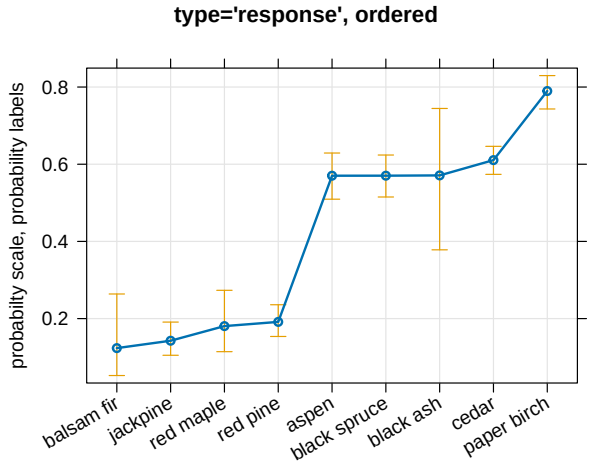
```
+                         lab="logit scale, probability labels"),
+                   x=list(rotate=30),
+                   grid=TRUE))

R> plot(e1.gm1, main="type='link'",
+         axes=list(y=list(type="link",
+                         lab="logit scale, logit labels"),
+                   x=list(rotate=30),
+                   grid=TRUE))

R> plot(e1.gm1, main="type='response'",
+         axes=list(y=list(type="response", grid=TRUE,
+                         lab="probabilty scale, probability labels"),
+                   x=list(rotate=30),
+                   grid=TRUE))
```



The first two graphs show the same plot, but in the first the tick-marks on the vertical axis are unequally spaced and are in probability scale, while in the second the tick-marks are equally spaced and are in log-odds scale. In the third graph, the vertical axis has been transformed to probability scale, and the corresponding tick-marks are now equally spaced.

The predictor effects plot for species would be easier to understand if the levels of the factor were ordered according to the estimated log-odds of blowdown. First, we need to recover the fitted values in link scale, which are log-odds of blowdown for a logistic model. The fitted log-odds are stored in `as.data.frame(e1.gm1)$fit` using the `e1.gm1` object previously computed:

```
R> or <- order(as.data.frame(e1.gm1)$fit) # order smallest to largest
R> Blowdown$spp1 <- factor(Blowdown$spp,  # reorder levels of spp
+                          levels=levels(Blowdown$spp)[or])
R> gm2 <- update(gm1, ~ . - spp + spp1)   # refit model
R> plot(predictorEffects(gm2, ~ spp1), main="type='response', ordered",
+         axes=list(y=list(type="response",
+                         lab="probabilty scale, probability labels"),
+                   x=list(rotate=30, spp=list(lab="Species")),
+                   grid=TRUE))
```



The separation of species into two groups of lower and higher probability species is reasonably clear after ordering, with paper birch more susceptible to blowdown than the other species and possibly in a group by itself.

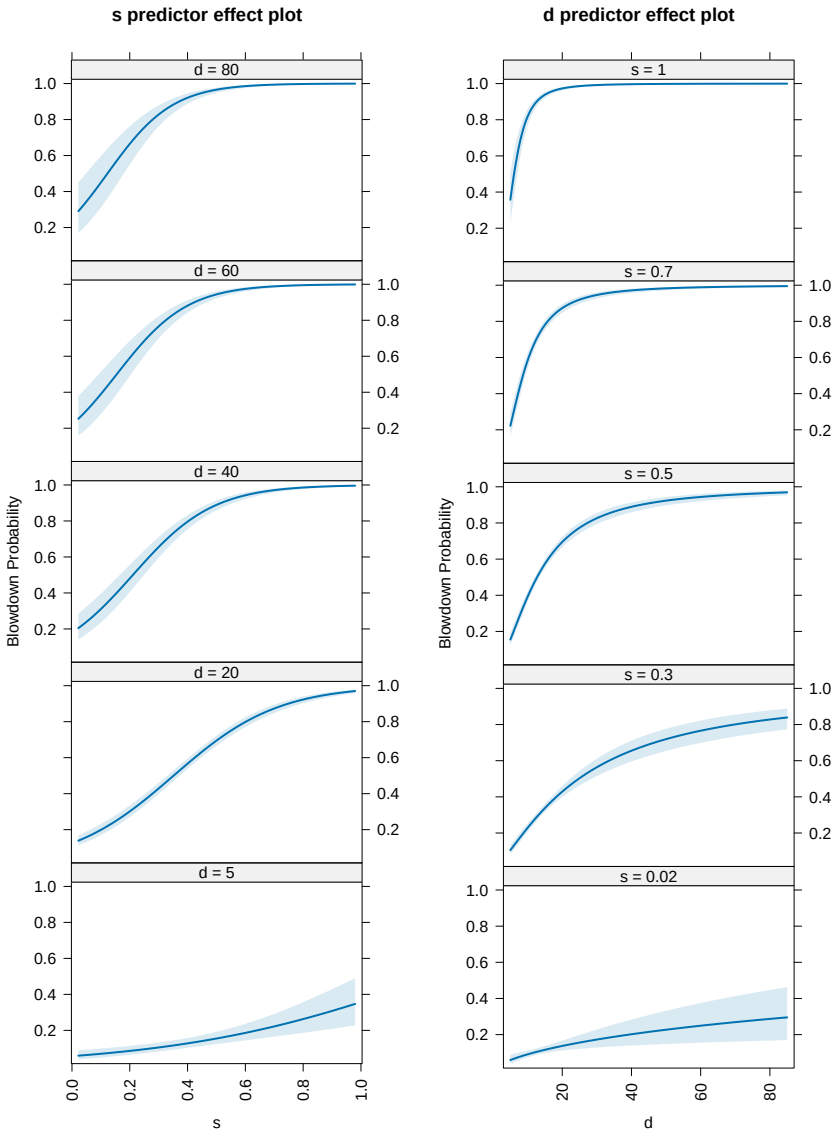
3.2 The `lines` Group: Specifying Plotted Lines

The `lines` argument group allows the user to specify the color, type, thickness, and smoothness of lines. This can be useful, for example, if the colors used by **effects** by default are for some reason unacceptable, such as for publications in which only black or gray-scale lines are permitted. The most common use of this argument group is to allow more than one line to be plotted on the same graph or panel via the `multiline` sub-argument.

3.2.1 `multiline` and `z.var`: Multiple Lines in a Plot

Default predictor effect plots with conditioning predictors generate a separate plot for each level of the conditioning variable, or for each combination of levels if there is more than one conditioning variable. For an example, we add the `log(d):s` interaction to the model `gm1`, and generate the predictor effect plots for `s` and for `d`:

```
R> gm3 <- update(gm2, ~ . + s:log(d)) # add an interaction
R> plot(predictorEffects(gm3, ~ s + d),
+         axes=list(x=list(rug=FALSE, rotate=90),
+                   y=list(type="response", lab="Blowdown Probability")),
+         lattice=list(layout=c(1, 5)))
```

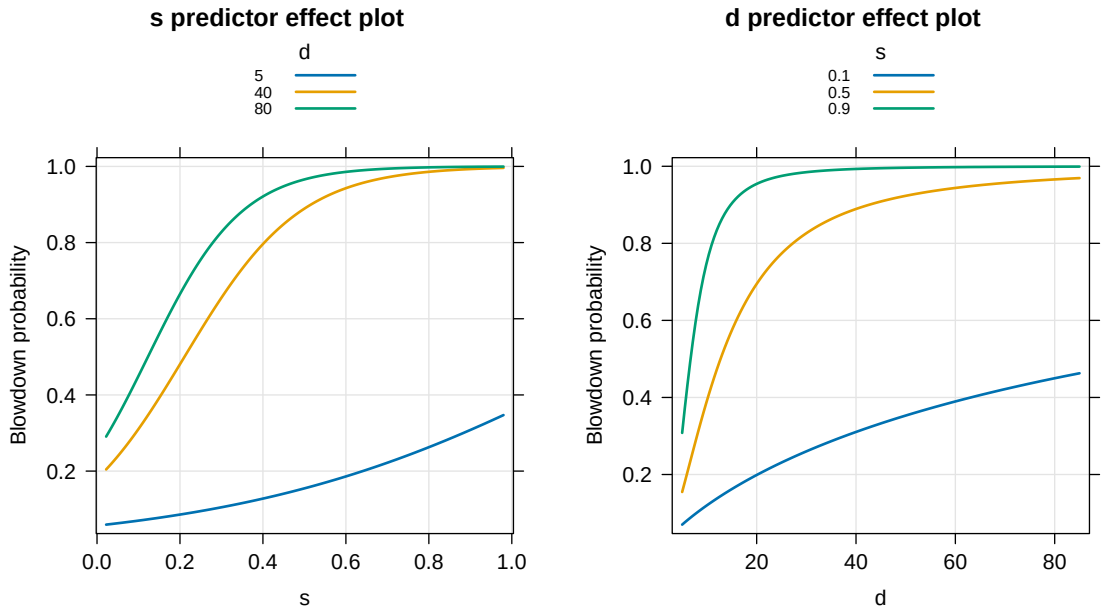


Setting the sub-argument `type="response"` for the `y` axis plots the response on the probability scale. Setting `layout=c(1, 5)` arranges each predictor effect plot in 1 column of 5 rows. See the description of the `lattice` argument in Section 3.4.

The predictor effect plot for `s` conditions on the level of `d`, and displays the plot of the fitted values for `y` versus `s` in a separate panel for each value of `d`. Similarly, the predictor effect plot for `d` displays a separate panel for each conditioning level of `s`. Confidence bands are displayed by default around each fitted line. These two graphs are based on essentially the same fitted values, with the values of the interacting predictors `s` and `d` varying, and fixing the factor predictor `spp` to its distribution in the data, as described in Section 2.2.1. Concentrating on the graph at the right for the focal predictor `d`, when `s` is very small the probability of blowdown is estimated to be in the range of about .05 to .3 for any value of `d`, but for larger values of `s`, the probability of blowdown increases rapidly with `d`. Similar comments can be made concerning the predictor effect plot for `s`.

Setting `multiline=TRUE` superimposes the lines for all the conditioning values in a single graph. In the example below, we reduce the number of levels of the conditioning variable for each predictor effect plot to three explicit values each to produce simpler graphs, although this is not required. The `xlevels` argument changes the number of levels for the conditioning predictors, but does not affect the number of levels for the focal predictor. This latter quantity could be changed with the `focal.levels` argument, but the default value of 50 evaluations is appropriate for graphing effects.

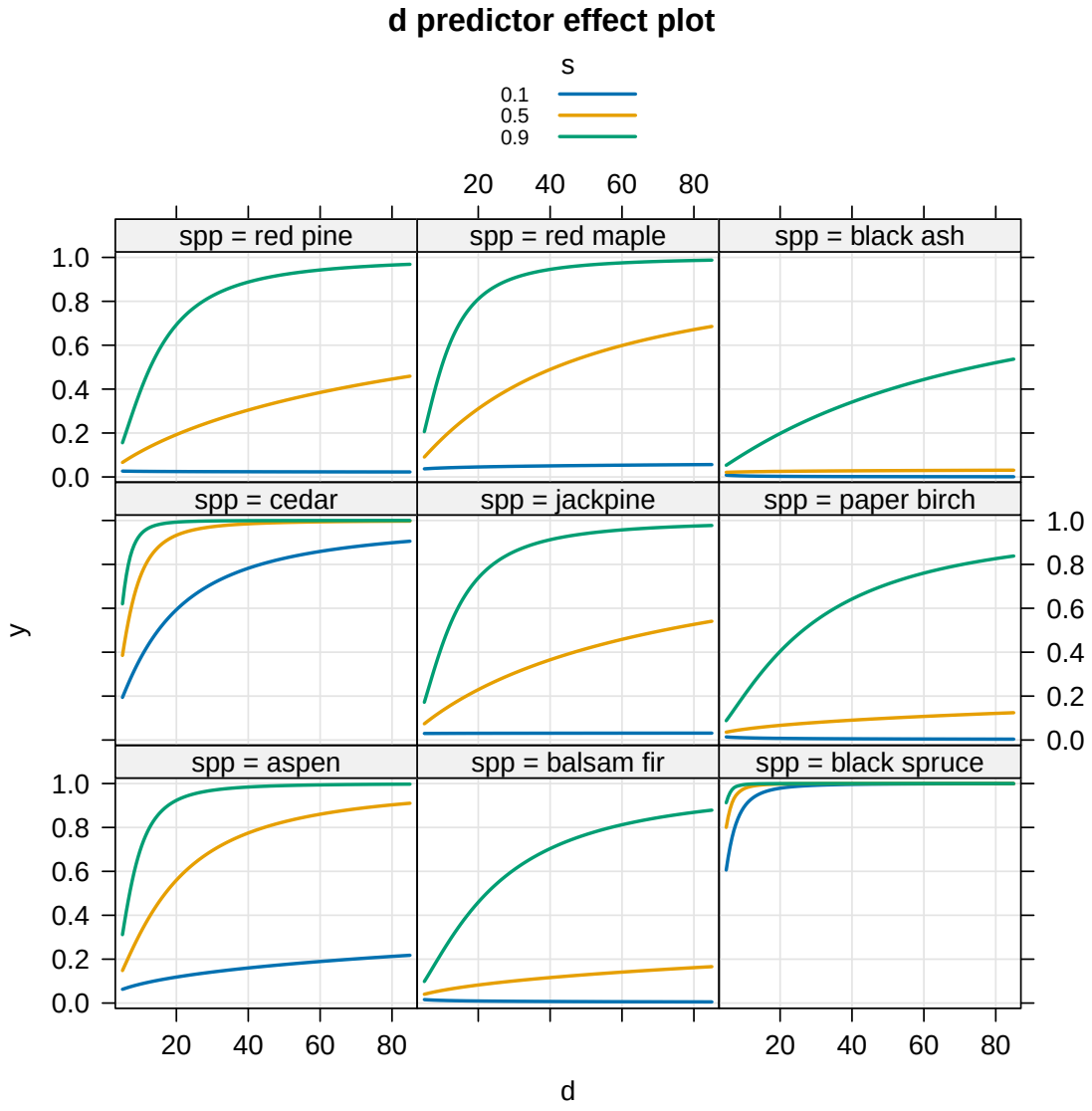
```
R> plot(predictorEffects(gm3, ~ s + d,
+                          xlevels=list(d=c(5, 40, 80), s=c(0.1, 0.5, 0.9))),
+         axes=list(grid=TRUE,
+                   x=list(rug=FALSE),
+                   y=list(type="response", lab="Blowdown probability")),
+         lines=list(multiline=TRUE))
```



In each graph, we kept, more or less, the lowest, middle, and highest values of the conditional predictor for the interaction. We also added a grid to each graph. Multiline plots by default omit confidence bands or intervals, but these can be included using the `confint` argument discussed in Section 3.3. By default, different values of the conditioning predictor are distinguished by color, and a key is provided. The placement and appearance of the key are controlled by the `key.args` sub-argument in the `lattice` group discussed in Section 3.4.1.

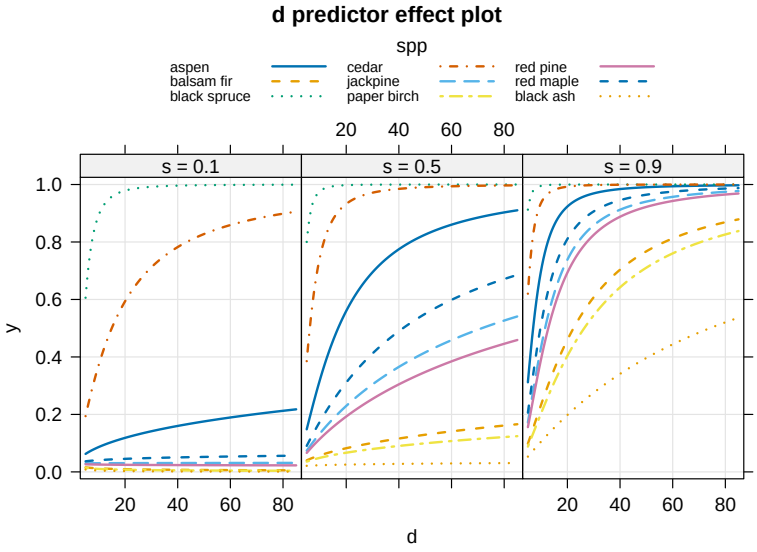
When the conditioning group includes two or more predictors, and certainly when it includes three or more predictors, multiline plots are almost always helpful because otherwise the resulting array of panels becomes too complicated. Suppose that we add the `spp:log(d)` interaction to the illustrative model. The predictor effect plot for `d` now includes both `s` and `spp` in the conditioning set because `d` interacts with both of these predictors:

```
R> gm4 <- update(gm3, ~ . + spp:log(d))
R> plot(predictorEffects(gm4, ~ d, xlevels=list(s=c(0.1, 0.5, 0.9))),
+         axes=list(grid=TRUE,
+                   y=list(type="response"),
+                   x=list(rug=FALSE)),
+         lines=list(multiline=TRUE))
```



This plot now displays the lines for all conditioning values of `s` within the panel for each level of the conditioning factor `spp`. Compare this graph to the much more confusing plot in which different lines are drawn for the nine levels of the conditioning factor `spp`, obtained by using the `z.var` sub-argument in the `lines` group:

```
R> plot(predictorEffects(gm4, ~ d, xlevels=list(s=c(0.1, 0.5, 0.9))),
+         axes=list(grid=TRUE,
+                   y=list(type="response"),
+                   x=list(rug=FALSE)),
+         lines=list(multiline=TRUE, z.var="spp", lty=1:9),
+         lattice=list(layout=c(3, 1)))
```



The `z.var` sub-argument for `lines` selects the predictor that determines the lines within a panel and the remaining predictors, here just `s`, distinguish the panels. The default choice of `z.var` is usually, but not always, appropriate. We also use the `lattice` argument to display the array of panels in 3 columns and 1 row, and differentiate the lines by line type and color using arguments discussed next.

3.2.2 `col`, `lty`, `lwd`, `spline`: Line Color, Type, Width, Smoothness

Different lines in the same plot are differentiated by default using color. This can be modified by the sub-arguments `lty`, `lwd` and `col` to set line types, widths, and colors, respectively. For example, in the last graph shown you can get all black lines of different line types using `lines=list(multiline=TRUE, col="black", lty=1:9)`, or using a gray scale, `lines=list(multiline=TRUE, col=gray((1:9)/10))`.

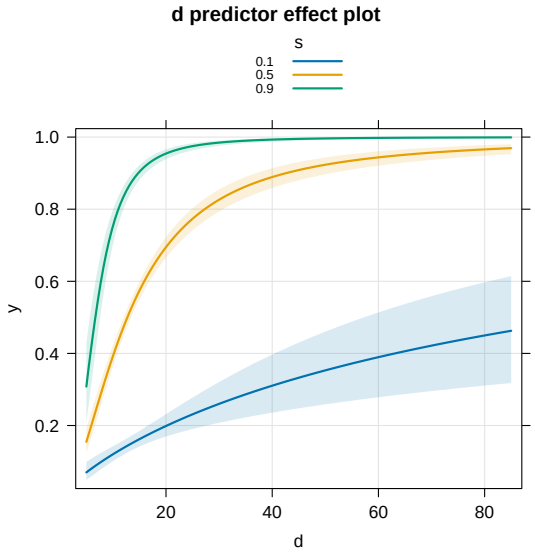
The `plot()` method for effect objects by default uses smoothing splines to interpolate between plotted points. Smoothing can be turned off with `splines=FALSE` in the `lines` argument, but we rarely expect this to be a good idea. The number of values at which the focal predictor is evaluated is set with the `focal.levels` argument, and it defaults to 50. In any case, more than three evaluations, and possibly many more, should be used for a reasonable spline approximation.

3.3 The `confint` Group: Specifying Confidence Interval Inclusion and Style

The `confint` argument group controls the inclusion and appearance of confidence intervals and regions. This argument has three sub-arguments. The `style` sub-argument is either `"bars"`, for confidence bars, typically around the estimated adjusted mean for a factor level; `"bands"`, for shaded confidence bands, typically for numeric focal predictors; `"auto"`, to let the program automatically choose between `"bars"` and `"bands"`; `"lines"`, to draw only the edges of confidence bands with no shading; or `"none"`, to suppress confidence intervals. The default is `"auto"` when `multiline=FALSE` and `"none"` when `multiline=TRUE`. Setting `confint="auto"` produces bars for factors and bands for numeric predictors. For example:

```
R> plot(predictorEffects(gm3, ~ d,
+                         xlevels=list(s=c(0.1, 0.5, 0.9))),
+         axes=list(grid=TRUE,
+                   x=list(rug=FALSE),
+                   y=list(type="response")),
```

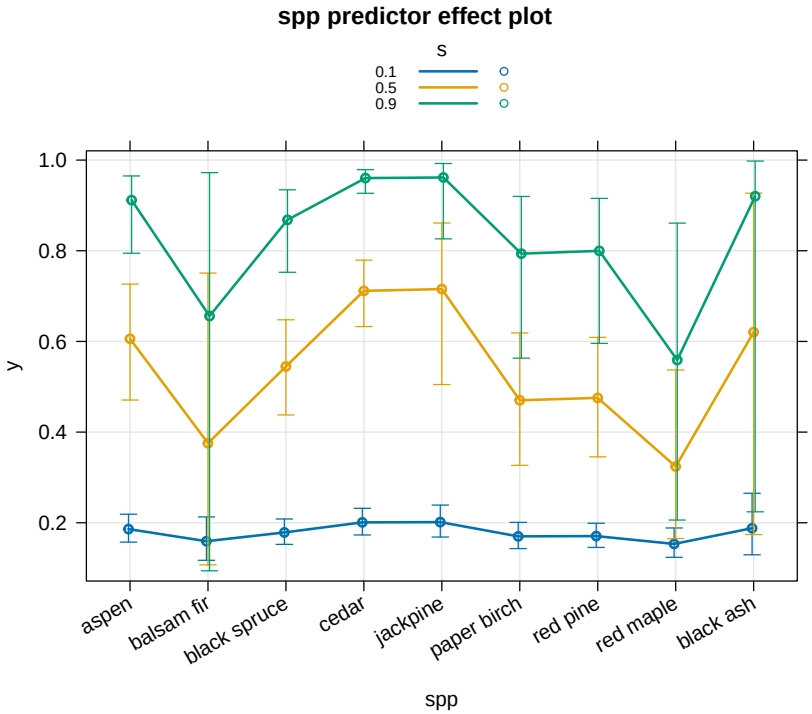
```
+         lines=list(multiline=TRUE),
+         confint=list(style="auto"))
```



In this example the confidence bands are well separated, so including them in a multiline graph isn't problematic; in other cases, overlapping confidence bands produce an artistic but uninterpretable mess.

With a factor focal predictor, we get:

```
R> gm5 <- update(gm2, ~ . + spp:s)
R> plot(predictorEffects(gm5, ~ spp, xlevels=list(s=c(0.1, 0.5, 0.9))),
+         axes=list(grid=TRUE,
+                   y=list(type="response"),
+                   x=list(rug=FALSE, rotate=30)),
+         lines=list(multiline=TRUE),
+         confint=list(style="auto"))
```



The error bars for the various levels of `s` are slightly staggered to reduce over-plotting.

Two additional arguments, `col` and `alpha`, control respectively the color of confidence bars and regions and the transparency of confidence regions. Users are unlikely to need these options. Finally, the type of confidence interval shown, either pointwise or Scheffé corrected for multiple comparisons, is controlled by the `se` argument to the `predictorEffect()` or `Effect()` function (see Section 2.3).

3.4 The `lattice` Group: Specifying Standard lattice Package Arguments

The `plot()` methods defined in the **effects** package use functions in the **lattice** package (Sarkar, 2008), such as `xyplot()`, to draw effect plots, which often comprise rectangular arrays of panels. In particular, the `plot()` method for the `"eff"` objects returned by the `Effect()` function are `"trellis"` objects, which can be manipulated in the normal manner. "Printing" a returned effect-plot object displays the plot in the current R graphics device.

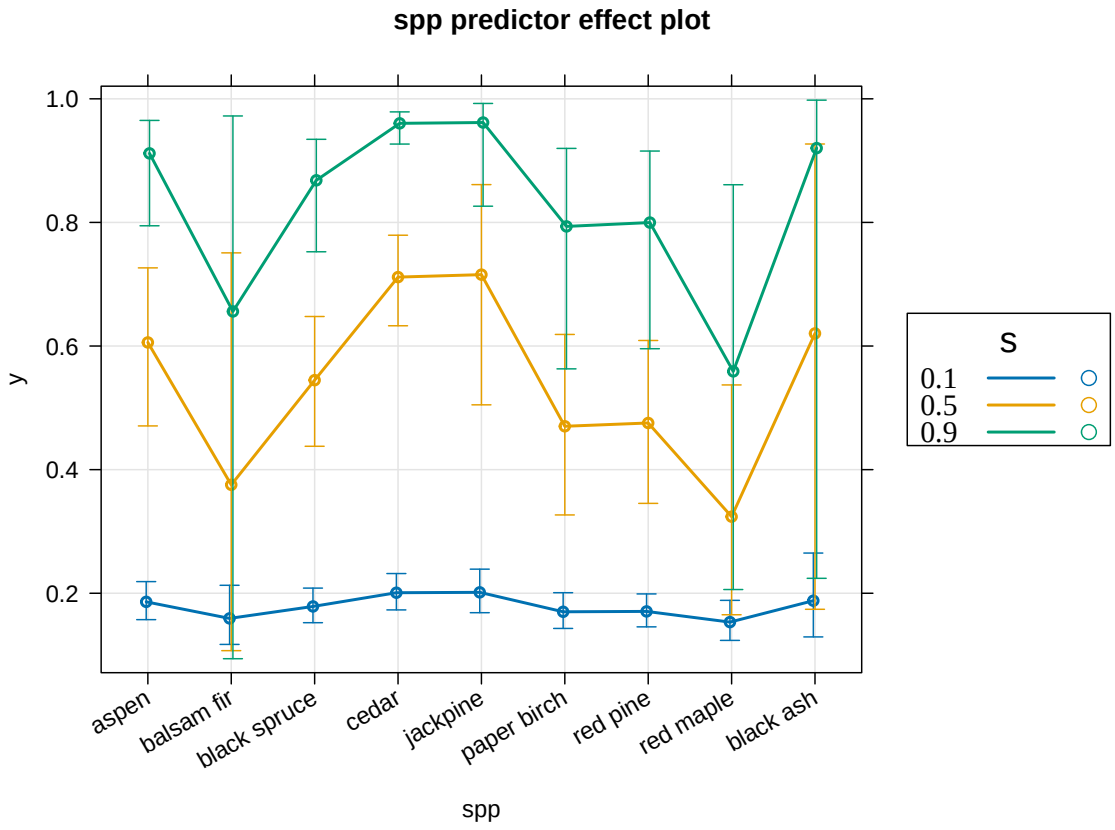
The `lattice` group of arguments to the `plot()` method for effect objects may be used to specify various standard arguments for **lattice** graphics functions such as `xyplot()`. In particular, you can control the number of rows and columns when panels are displayed in an array, modify the key (legend) for the graph, and specify the contents of the "strip" displayed in the shaded region of text above each panel in a **lattice** array. In addition, the `array` sub-argument, for advanced users, controls the layout of multiple predictor effect plots produced by the `predictorEffects()` function.

3.4.1 `key.args`: Modifying the Key

A user can modify the placement and appearance of the key with the `key.args` sub-argument, which is itself a list. For example:

```
R> plot(predictorEffects(gm5, ~ spp, xlevels=list(s=c(0.1, 0.5, 0.9))),
+         rug=FALSE,
+         axes=list(grid=TRUE,
+                   y=list(type="response"),
+                   x=list(rotate=30)),
+         lines=list(multiline=TRUE),
```

```
+         confint=list(style="auto"),
+         lattice=list(key.args=list(space="right",
+                                    columns=1,
+                                    border=TRUE,
+                                    fontfamily="serif",
+                                    cex=1.25,
+                                    cex.title=1.5)))
```

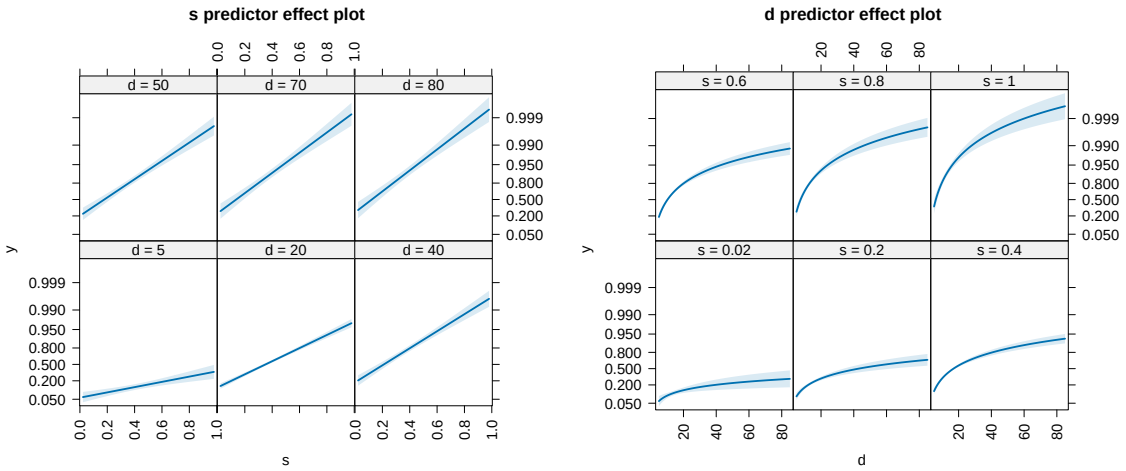


The sub-argument `space="right"` moves the key to the right of the graph, overriding the default `space="top"`. Alternatively the key can be placed inside the graph using the `x`, `y`, and `corner` sub-arguments, as illustrated in the graph on page 23. The choices for `fontfamily` are `"sans"` and `"serif"`, and affect only the key; the rest of the plot uses `"sans"`. The sub-arguments `cex` and `cex.title` control the relative sizes of the key entries and the key title, respectively. Finally, any argument documented in `help("xyplot")` in the `key` section can be set with this argument. If you use the default `space="top"` for placement of the key, you may wish to adjust the number of columns in the key, particularly if the level names are long.

3.4.2 `layout`: Controlling Panel Placement

The `layout` sub-argument to the `lattice` argument allows a user to customize the layout of multiple panels in a predictor effect plot; for example:

```
R> plot(predictorEffects(gm3, ~ s + d, xlevels=list(s=6, d=6)),
+         axes=list(x=list(rug=FALSE, rotate=90),
+                   y=list(ticks=list(at=c(.999, .99, .95, .8, .5, .2, .05)))),
+         lattice=list(layout=c(3, 2)))
```



Here, the `layout` sub-argument specifies an array of 3 columns and 2 rows for each of the predictor effect plots.

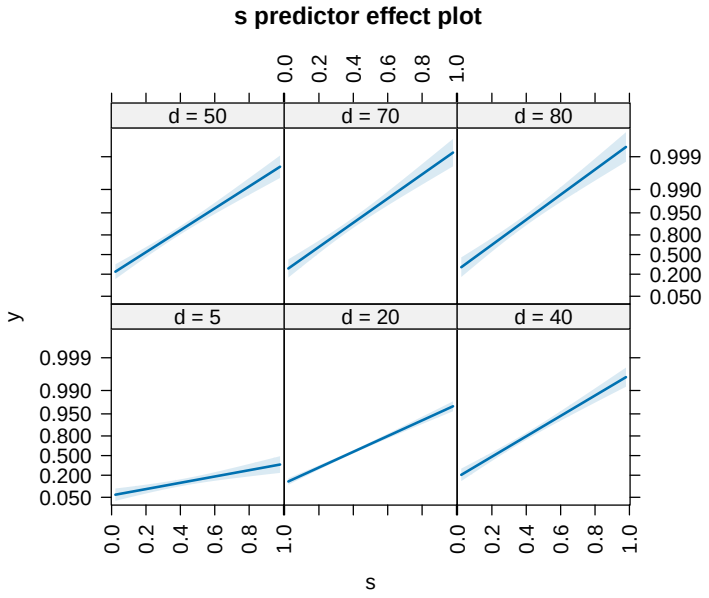
3.4.3 `array`: Multiple Predictor Effect Plots

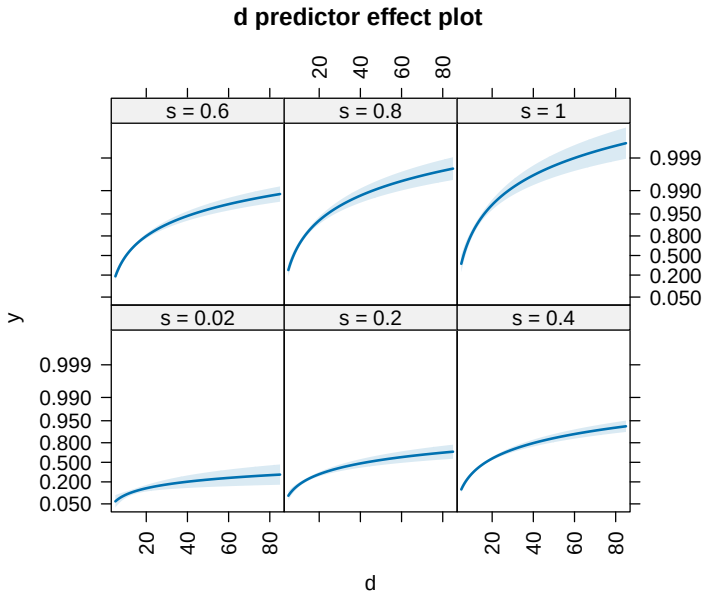
If you create several predictor effect objects with the `predictorEffects()` function, the `plot()` method for the resulting `"predictorefflist"` object divides the **lattice** graphics device into a rectangular array of sub-plots, so that the individual predictor effect plots, each potentially with several panels, are drawn without overlapping. An alternative is for the user to generate the predictor effect plots separately, subsequently supplying the `array` sub-argument to `plot()` directly to create a custom meta-array of predictor effect plots; this argument is ignored, however, for `"predictorefflist"` objects produced by `predictorEffects()`.

Suppose, for example, that we want to arrange the two predictor effect plots for the previous example vertically rather than horizontally. One way to do that is to save the object produced by `predictorEffects()` and to plot each of its two components individually, specifying the `position` or `split` and `more` arguments to the `print()` method for `"trellis"` objects: see `help("print.trellis")`.

Another approach is to generate the plots individually using `predictorEffect()` and to specify the `array` sub-argument to `plot()`, as follows:

```
R> plot(predictorEffect("s", gm3, xlevels=list(d=6)),
+         axes=list(x=list(rug=FALSE, rotate=90),
+                   y=list(ticks=list(at=c(.999, .99, .95, .8, .5, .2, .05)))),
+         lattice=list(layout=c(3, 2),
+                      array=list(row=1, col=1, nrow=2, ncol=1, more=TRUE)))
R> plot(predictorEffect("d", gm3, xlevels=list(s=6)),
+         axes=list(x=list(rug=FALSE, rotate=90),
+                   y=list(ticks=list(at=c(.999, .99, .95, .8, .5, .2, .05)))),
+         lattice=list(layout=c(3, 2),
+                      array=list(row=2, col=1, nrow=2, ncol=1, more=FALSE)))
```





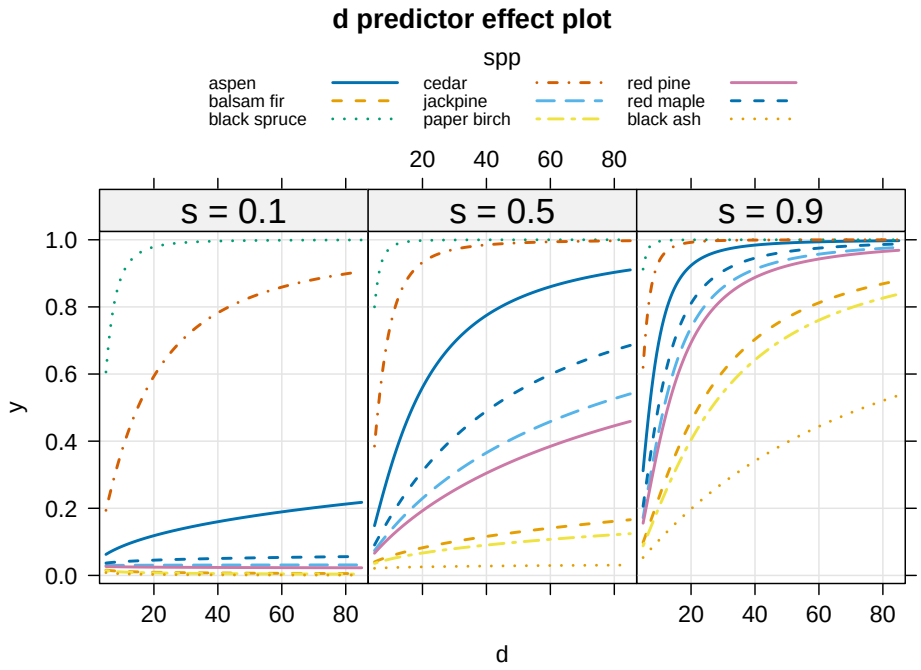
In each case, the `row` and `col` sub-arguments indicate the position of the current graph in the meta-array; `nrow` and `ncol` give the dimensions of the meta-array, here 2 rows and 1 column; and `more` indicates whether there are more elements of the meta-array after the current graph.

3.4.4 `strip`: Modifying the Text at the Tops of Panels

Lattice graphics with more than one panel typically provide a text label at the top of each panel in an area called the *strip*. The default strip text contains the name of the conditioning predictor and the value to which it is set in the panel; if there are more than one conditioning predictor, then all of their names and corresponding values are shown. For example:

```
R> plot(predictorEffects(gm4, ~ d, xlevels=list(s=c(0.1, 0.5, 0.9))),
+         axes=list(grid=TRUE,
+                   x=list(rug=FALSE),
+                   y=list(type="response")),
```

```
+         lines=list(multiline=TRUE, z.var="spp", lty=1:9),
+         lattice=list(layout=c(3, 1),
+                      strip=list(factor.names=TRUE,
+                                 values=TRUE,
+                                 cex=1.5)))
```

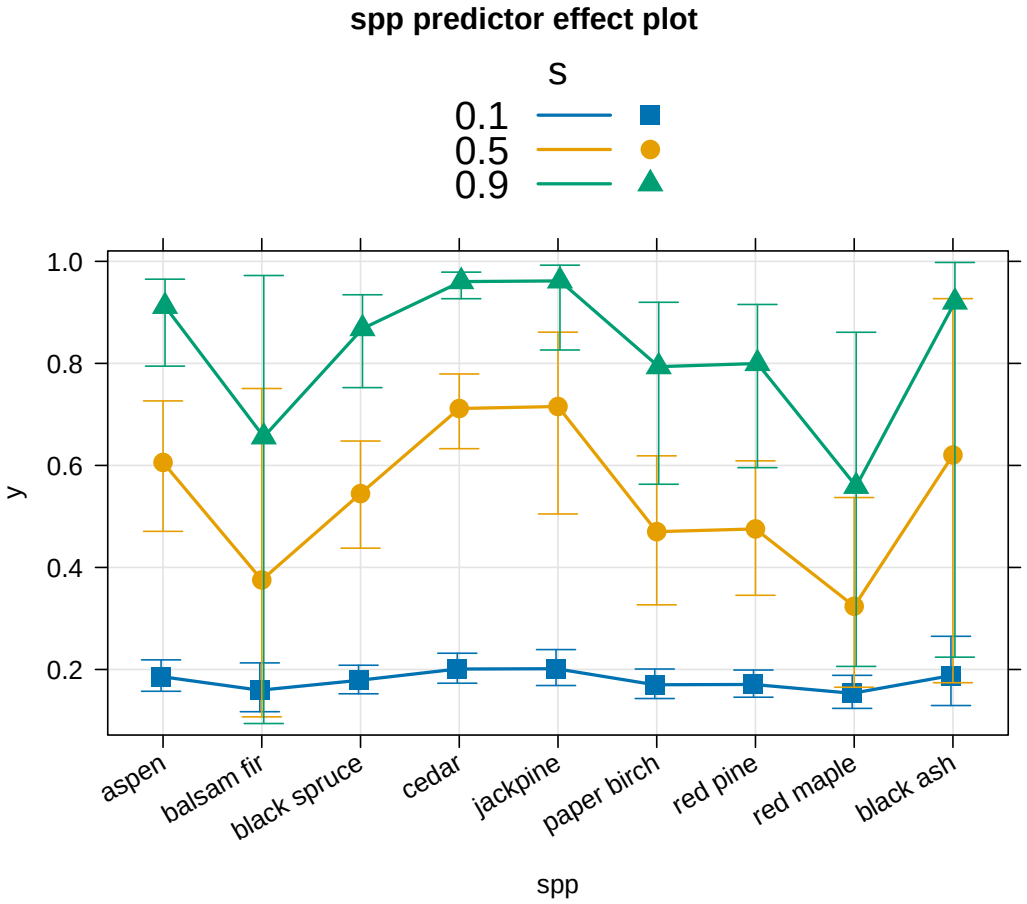


Setting `factor.names=FALSE` (the default is `TRUE`) displays only the value, and not the name, of the conditioning predictor in each strip; usually, this is desirable only if the name is too long to fit, in which case you may prefer to rename the predictor. Setting `values=FALSE` replaces the conditioning value with a line in the strip that represents the value: The line is at the left of the strip for the smallest conditioning value, at the right for the largest value, and in a proportional intermediate position in between the two extremes. The most generally useful sub-argument is `cex`, which allows you to reduce or expand the relative size of the text in the strip, in this case increasing the size to 150% of standard size.

3.5 `symbols`: Plotting symbols

Symbols are used to represent adjusted means when the focal predictor is a factor. You can control the symbols used and their relative size:

```
R> gm5 <- update(gm2, ~ . + spp:s)
R> plot(predictorEffects(gm5, ~ spp, xlevels=list(s=c(0.1, 0.5, 0.9))),
+         symbols=list(pch=15:17, cex=1.5),
+         axes=list(grid=TRUE,
+                   y=list(type="response"),
+                   x=list(rotate=30)),
+         lines=list(multiline=TRUE),
+         confint=list(style="auto"),
+         lattice=list(key.args=list(cex=1.5, cex.title=1.5)))
```



We use the `pch` sub-argument to set the symbol number for plotted symbols; you can enter the commands `plot(1:25, pch=1:25)` and `lines(1:25, lty=2, type="h")` to see the 25 plotting symbols in R. The sub-argument `pch` can also be a character vector, such as `letters[1:10]`. In this example, we set `cex=1.5` to increase the symbol size by the factor 1.5. Because only one value is given, it is recycled and used for all of the symbols. We need to change the size of the symbols in the key separately, as we do here via the `key.args` sub-argument to the `lattice` argument (see Section 3.4.1).

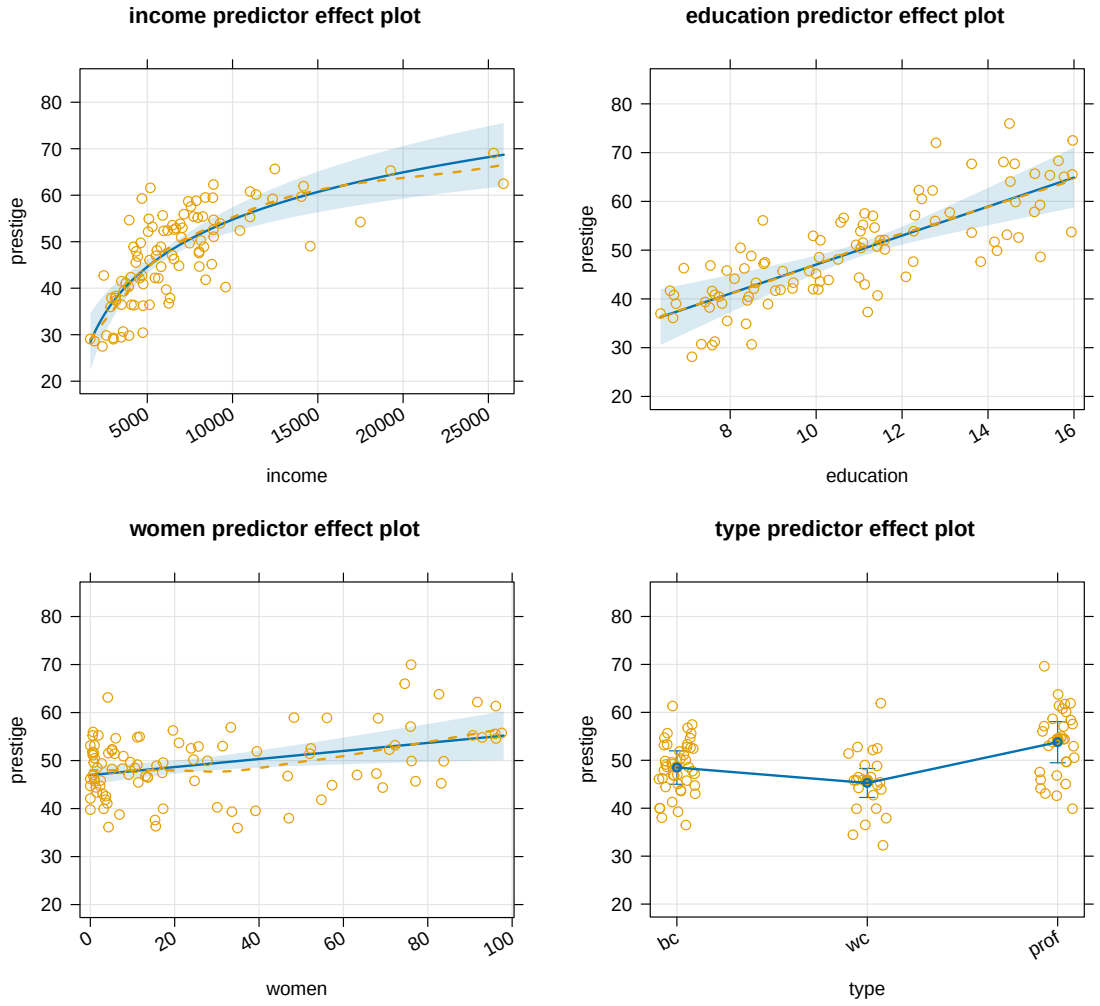
4 Displaying Residuals in Predictor Effect Plots

Fox and Weisberg (2018) introduce methodology for adding partial residuals to a predictor effect or effect plot. This can be desirable to display variation in data around a fitted partial regression surface or to diagnose possible lack of fit, as the resulting plots are similar to traditional component-plus-residual plots (Fox and Weisberg, 2019, Sec. 8.4).

The predictor effect plot for a numeric focal predictor that does not interact with other predictors is equivalent to a standard component-plus-residual plot; for example:

```
R> lm5 <- lm(prestige ~ log(income) + education + women + type,
+             Prestige)
R> plot(predictorEffects(lm5, residuals=TRUE),
+         axes=list(grid=TRUE,
+                   x=list(rotate=30)),
+         partial.residuals=list(smooth=TRUE,
+                                span=0.75,
```

```
+                                    lty="dashed"))
```



The partial residuals to be plotted are computed using the `residuals` argument to the `predictorEffect()`, `predictorEffects()`, or `Effect()` function. For the numeric predictors `income`, `education`, and `women`, the plotted points are each equal to a point on the fitted blue line, representing the partial fit, plus the corresponding residual. For `income`, the fitted partial-regression line in curved because of the log transformation of the predictor, but the partial-regression function is a straight line for the other two numeric predictors.

The dashed line produced by `lty="dashed"` in the same magenta color as the plotted points on the graph, is a loess nonparametric-regression smooth of the points. The sub-argument `smooth=TRUE` is the default if residuals are present in the effect object to be plotted. The sub-argument `span=0.75` adjusts the span of the loess smoother from the default of `2/3`—an unnecessary adjustment here specified simply to illustrate how to set the span. If the model adequately represents the data, then the dashed magenta line should approximately match the solid blue partial-regression line, which represents the fitted model.

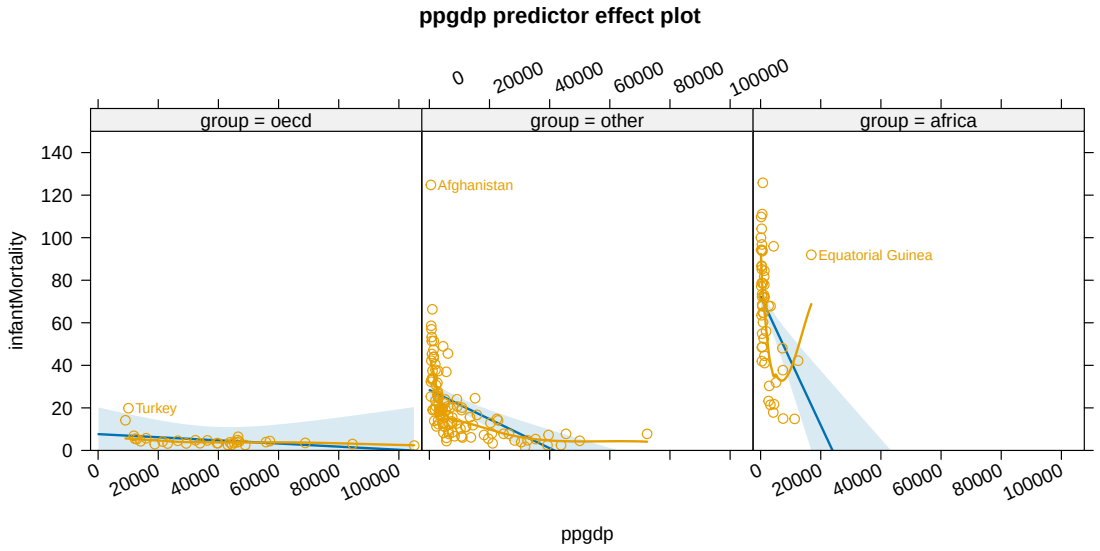
For the factor `type`, the points are jittered horizontally to separate them visually, because the only possible horizontal coordinates are at the three distinct factor levels. Smooths are not fit to factors and instead the conditional means of the partial residuals are plotted as solid magenta dots; in the current model, the magenta dots and the blue dots representing the fitted adjusted means of the response at the levels of `name` necessarily match.

The `plot()` method for effect objects has a `partial.residuals` argument, with several sub-arguments that control how partial residuals are displayed. In the command above, we used the

sub-argument `smooth=TRUE` to add the smoother, which is the default when residuals are included in the effect object, and `lty="dashed"` to change the line type for the smooth from the default solid line to a dashed line. All the `smooth` sub-arguments are described in `help("plot.eff")`.

For a second example, we fit a linear model with an interaction to the `UN` data set in the **carData** package, modelling national `infantMortality` rate (infant deaths per 1000 live births) as a function of `ppgdp`, per person GDP (in U.S. dollars), and country `group` (OECD nations, African nations, and other nations). The data are for roughly 200 nations of the world and are from approximately 2009 to 2011:

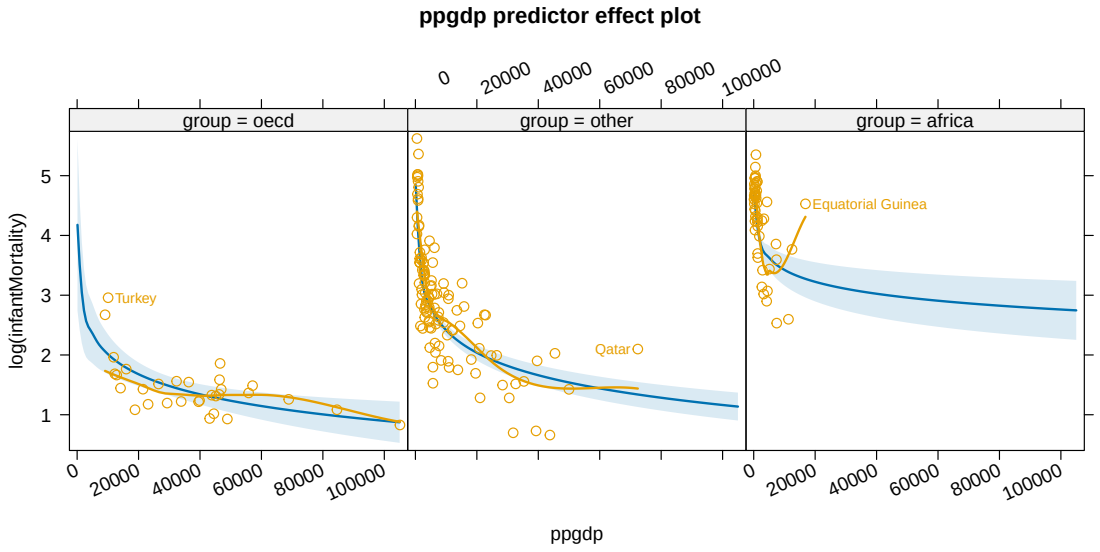
```
R> options(scipen=10) # suppress scientific notation
R> lm6 <- lm(infantMortality ~ group*ppgdp, data=UN)
R> plot(predictorEffects(lm6, ~ ppgdp, partial.residuals=TRUE),
+         axes=list(x=list(rotate=25),
+                   y=list(lim=c(0, 150))),
+        id=list(n=1),
+        lattice=list(layout=c(3, 1)))
```



The predictor effect plot for `ppgdp` conditions on the factor `group` because of the interaction between these two predictors. Several problems are apparent in this plot: The `id` argument is used to identify the most unusual point in each panel, as described in detail in `help("plot.eff")`. Turkey has higher than predicted infant mortality for the `"oecd"` group; Afghanistan, in the `"other"` group, has infant mortality much higher than predicted; and Equatorial Guinea is clearly unusual for the `"africa"` group. In addition, the smooths through the points do not match the fitted lines in the `"other"` and `"africa"` groups. We use the command `options(scipen=10)` to suppress annoying scientific notation in the tick-mark labels on the horizontal axis, and instead rotate these labels so that they fit without over-plotting.

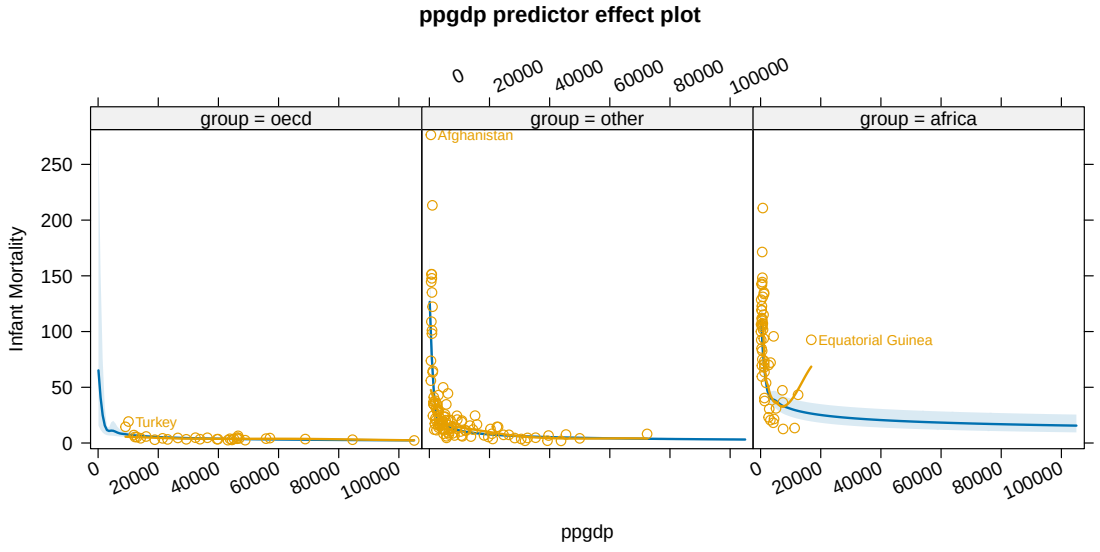
Log-transforming both the predictor `ppgdp` and the response `infantMortality` produces a better fit to the data:

```
R> lm7 <- lm(log(infantMortality) ~ group*log(ppgdp), data=UN)
R> plot(predictorEffects(lm7, ~ ppgdp, partial.residuals=TRUE),
+         axes=list(x=list(rotate=25)),
+         id=list(n=1),
+         lattice=list(layout=c(3, 1)))
```



Equatorial Guinea is still anomalous, however. Rescaling the vertical axis to arithmetic scale produces a slightly different, but possibly useful, picture:

```
R> plot(predictorEffects(lm7, ~ ppgdp, partial.residuals=TRUE),
+         axes=list(x=list(rotate=25),
+                   y=list(transform=list(trans=log, inverse=exp),
+                          type="response",
+                          lab="Infant Mortality")),
+         id=list(n=1),
+         lattice=list(layout=c(3, 1)))
```



Partial residuals can be added to effect plots for linear or generalized linear models in the default link scale, and to effect plots for linear or generalized linear mixed models.

4.1 Using the `Effect()` Function With Partial Residuals

In most instances, predictor effect plots produced by `predictorEffect()` or `predictorEffects()` visualize a fitted model in the most natural manner, but sometimes in looking for lack of fit, we

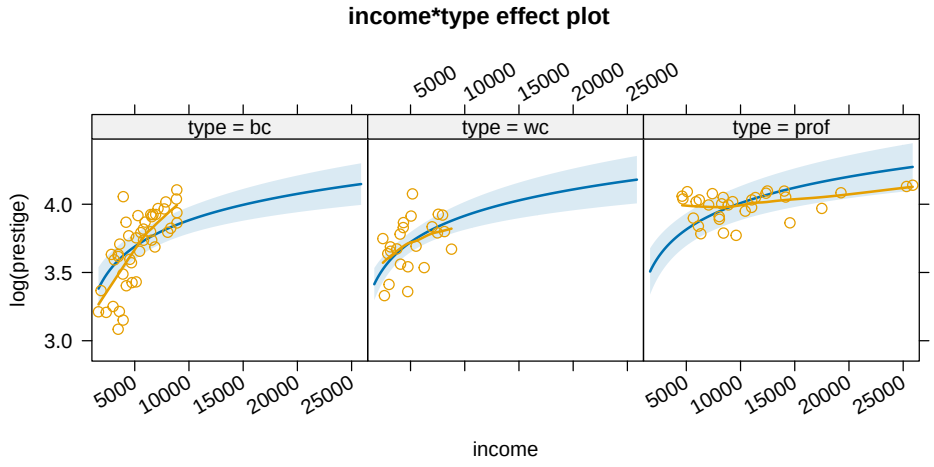
want to plot against arbitrary combinations of predictors. The more general `Effect()` function is capable of doing that.

Recall, for example, the additive model `lm2` fit to the `Prestige` data:

```
R> S(lm2)

Call: lm(formula = log(prestige) ~ log(income) + education + type, data =
        Prestige)

Coefficients:
            Estimate Std. Error t value    Pr(>|t|)
(Intercept)   0.6073     0.3666    1.66     0.10100
log(income)   0.2787     0.0458    6.09 0.000000026
education     0.0656     0.0162    4.05     0.00011
typewc        0.0325     0.0634    0.51     0.61014
typeprof      0.1255     0.0965    1.30     0.19662

Residual standard deviation: 0.177 on 93 degrees of freedom
  (4 observations deleted due to missingness)
Multiple R-squared: 0.792
F-statistic: 88.5 on 4 and 93 DF,  p-value: <2e-16
   AIC    BIC
-54.36 -38.85
```

Plotting partial residuals for the predictors `income` and `type` simultaneously reveals an unmodeled `income` × `type` interaction:

```
R> plot(Effect(c("income", "type"), lm2, residuals=TRUE),
+         axes=list(x=list(rotate=30)),
+         partial.residuals=list(span=0.9),
+         layout=c(3, 1))
```



5 Polytomous Categorical Responses

The **effects** package produces special graphs for ordered and unordered polytomous categorical response variables. In an ordinal regression, the response is an ordered categorical variable with three or more levels. For example, in a study of women's labor force participation that we introduce below, the response is not working outside the home, working part time, or working full time. The

proportional-odds model (Fox and Weisberg, 2019, Sec. 6.9) estimates the probability of a response in each of these three categories given a linear combination of regressors defined by a set of predictors, assuming a logit link function.

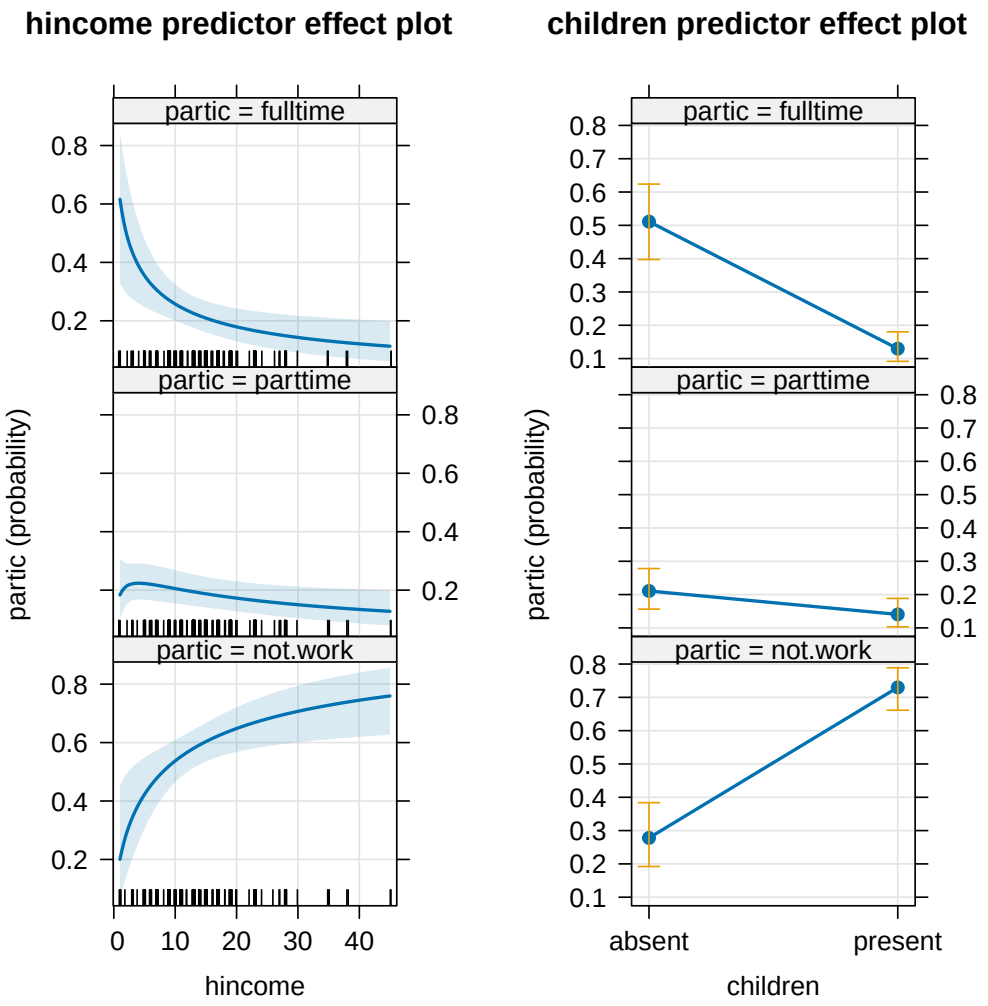
We illustrate the proportional-odds model with the `Womenlf` data set in the **carData** package, for young married Canadian women's labor-force participation, using the `polr()` function in the **MASS** package to fit the model:

```
R> library("MASS") # for polr()
R> Womenlf$partic <- factor(Womenlf$partic,
+        levels=c("not.work", "parttime", "fulltime")) # order response levels
R> or1 <- polr(partic ~ log(hincome) + children, data=Womenlf)
R> S(or1)


Re-fitting to get Hessian

Call: polr(formula = partic ~ log(hincome) + children, data = Womenlf)

 Coefficients:
               Estimate Std. Error z value      Pr(>|z|)
log(hincome)     -0.666      0.233   -2.86        0.0042
childrenpresent  -1.948      0.287   -6.80 0.00000000001

 Intercepts (Thresholds):
                  Estimate Std. Error z value Pr(>|z|)
not.work|parttime   -2.747      0.654   -4.20 0.000027
parttime|fulltime   -1.837      0.640   -2.87   0.0041

Residual Deviance: 441.12

 logLik      df     AIC     BIC
-220.56       4  449.12  463.40
```

The response variable `partic` initially has its levels in alphabetical order, which does not correspond to their natural ordering. We therefore start by reordering the levels to increase from `"not.work"`, to `"parttime"` work, to `"fulltime"` work. The predictors are the numeric variable `hincome` (husband's income), which enters the model in log-scale, and the dichotomous factor `children`, presence of children in the household.

The model summary is relatively complex, and is explained in Fox and Weisberg (2019, Sec. 6.9). Predictor effect plots greatly simplify interpretation of the fitted model:

```
R> plot(predictorEffects(or1),
+         axes=list(grid=TRUE),
+         lattice=list(key.args=list(columns=1)))


Re-fitting to get Hessian


Re-fitting to get Hessian
```



Unlike predictor effect plots for generalized linear models, the default scaling for the vertical axis is the probability scale, equivalent to `axes=list(y=list(type="response"))` for a GLM, and the alternative is `axes=list(y=list(type="logit"))`, which is analogous to `type="link"` for a GLM.[3] Confidence bands are present by default, unless turned off with the argument `confint=list(style="none")`. Numeric focal predictors are by default evaluated at 50 points. The plot for `hincome` suggests high probability of full-time work if husband's income is low, with the probability of full-time work sharply decreasing to about $15,000 and then nearly leveling off at about .1 to .2. The probability of not working rapidly increases with husband's income, while the probability of working part time is fairly flat. A similar pattern is apparent for children present in the home, with full-time work much less prevalent and not working much more prevalent when children are present than when they are absent.

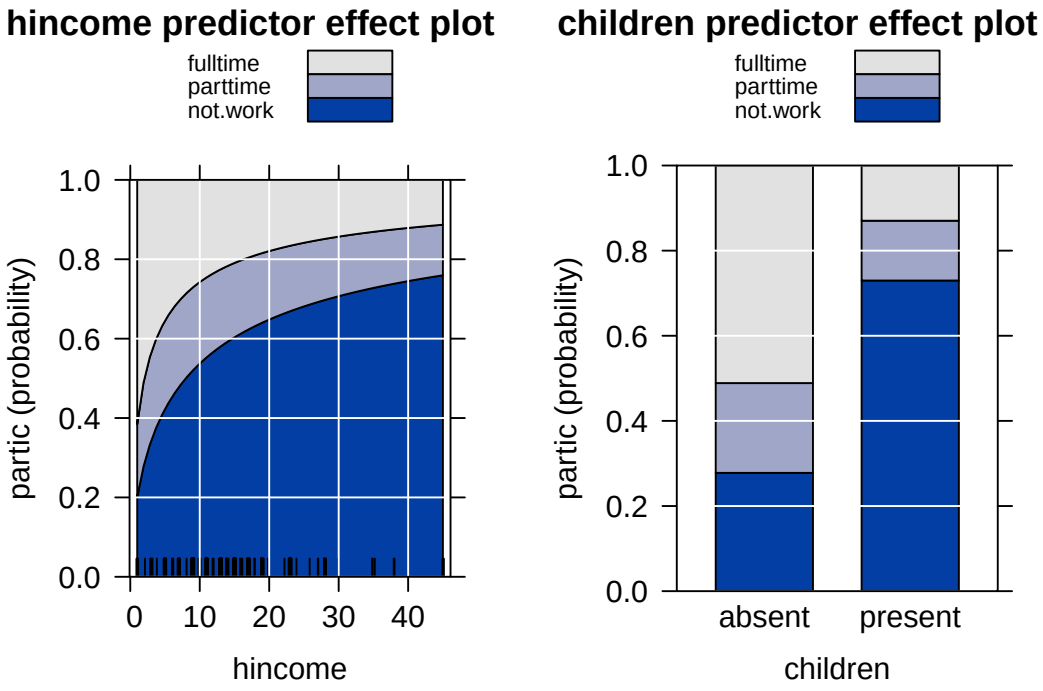
Stacked area plots are sometimes more useful for examining polytomous response models; for example:

```
R> plot(predictorEffects(or1),
+         axes=list(grid=TRUE, y=list(style="stacked")),
+         lattice=list(key.args=list(columns=1)))
```

```
Re-fitting to get Hessian
```

[3]The logits plotted, however, correspond to the individual-level probabilities and are not the ordered logits in the definition of the proportional-odds model.

```
Re-fitting to get Hessian
```



For each fixed value on the horizontal axis, the vertical axis "stacks" the probabilities in the three response categories. For example, with children absent from the household and `hincome` set to its mean, nearly 30% of women did not work outside the home, about 20% worked part time, and the remaining approximate 50% worked full time.

Some ordinal-response models produced by the functions `clm()`, `clm2()`, and `clmm()` in the **ordinal** package can be used with the **effects** package. To work with model objects produced by these functions, you must also load the **MASS** package.

The **effects** package can also draw similar graphs for the more general multinomial logit model, in which the polytomous categorical response has unordered levels (see Fox and Weisberg, 2019, Sec. 6.7). The details of the model, its parameters, and its assumptions are different from those of the proportional-odds model and other ordered-response models, but predictor effect plots for these models are similar.

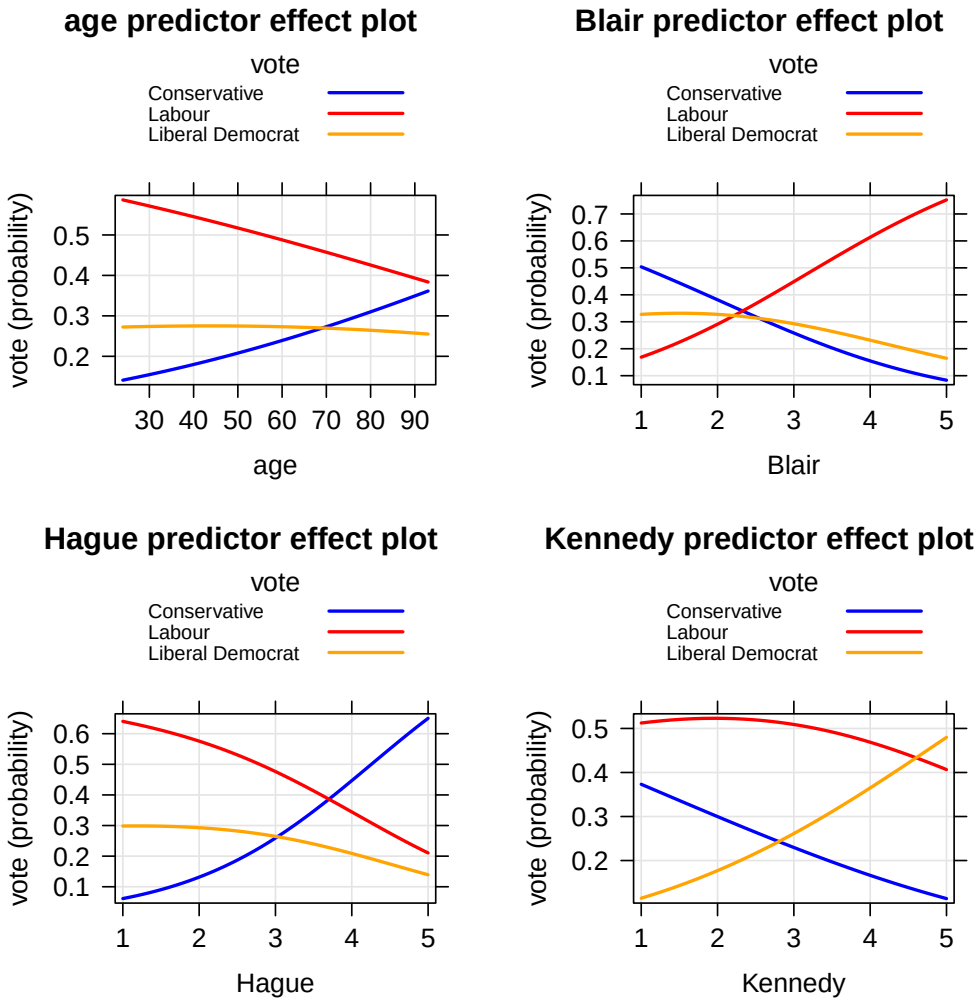
As an example, we use the `BEPS` data set in the **carData** package, consisting of about 1,500 observations from the 1997-2001 British Election Panel Study. The response variable, `vote`, is party choice, one of `"Liberal Democrat"`, `"Labour"`, or `"Conservative"`. There are numerous predictors of `vote` in the data set, and we fit the model

```
R> library("nnet") # for multinom()
R> mr1 <- multinom(vote ~ age + gender + economic.cond.national +
+                            economic.cond.household + Blair + Hague + Kennedy +
+                            Europe*political.knowledge, data=BEPS)

# weights:  36 (22 variable)
initial  value 1675.383740
iter  10 value 1240.047788
iter  20 value 1163.199642
iter  30 value 1116.519687
final  value 1116.519666
converged
```

There are nine predictors, seven of which are scales with values between 0 and 5 concerning respondents' attitudes; these predictors enter the model as main effects. The remaining two predictors are scales between 0 and 3 for `political.knowledge` and between 1 and 11 for `Europe` (attitude toward European integration of the UK in the European Union, with high values representing "Euroscepticism", a *negative* attitude toward Europe); these predictors enter the model with a two-factor interaction.

Drawing all nine predictor effect plots simultaneously is not a good idea because the plots won't fit reasonably in a single display. We therefore draw only a few of the plots at a time:

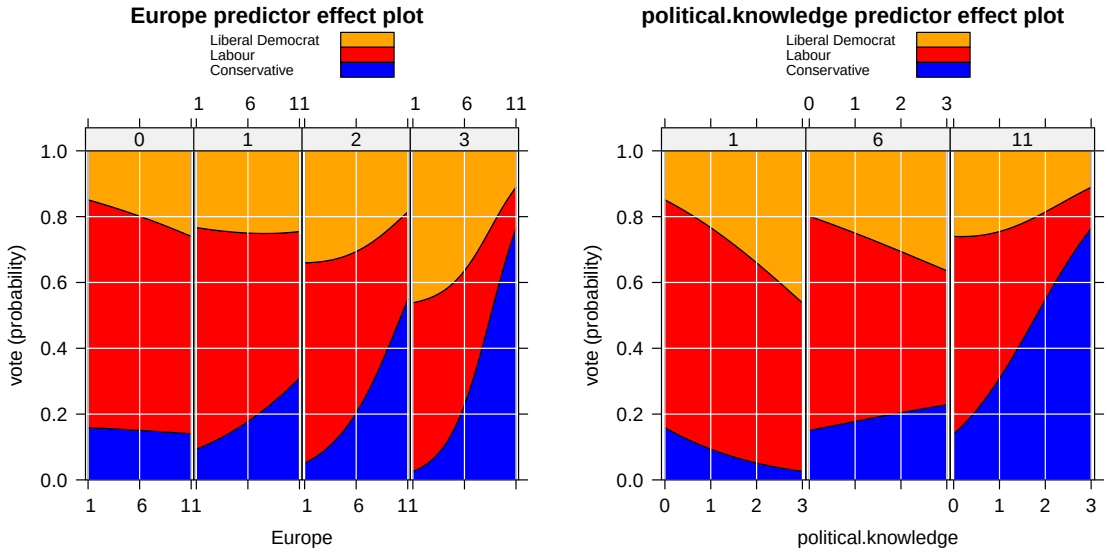
```
R> plot(predictorEffects(mr1, ~ age + Blair + Hague + Kennedy),
+         axes=list(grid=TRUE, x=list(rug=FALSE)),
+         lattice=list(key.args=list(columns=1)),
+         lines=list(multiline=TRUE, col=c("blue", "red", "orange")))
```



We use optional arguments to get a multiline plot, with a grid and no rug plot, and to modify the key. The color specification for the lines represents the traditional colors of the three parties. Interpreting these plots is challenging: For example, the probability of voting Labour decreases with age, increases with attitude toward the Labour leader Blair, strongly decreases with attitude toward the Conservative leader Hague, and is relatively unaffected by attitude toward the Liberal Democrat leader Kennedy. In general, a positive attitude toward a party leader increases the probability of voting for that leader's party, as one would expect. Of course, the causal direction of these relationships is unclear.

We next turn to the interaction between `Europe` and `political.knowledge`, this time drawing stacked area displays:

```
R> plot(predictorEffects(mr1, ~ Europe + political.knowledge,
+                            xlevels=list(political.knowledge=0:3,
+                                         Europe=c(1, 6, 11))),
+         axes=list(grid=TRUE,
+                   x=list(rug=FALSE,
+                          Europe=list(ticks=list(at=c(1, 6, 11))),
+                          political.knowledge=list(ticks=list(at=0:3))),
+                   y=list(style="stacked")),
+         lines=list(col=c("blue", "red", "orange")),
+         lattice=list(key.args=list(columns=1),
+                      strip=list(factor.names=FALSE)))
```



The `lines` argument is used to specify the colors for the stacked areas representing the parties. Both effect plots are of nearly the same fitted values,[4] in the first graph with `Europe` varying and conditioning on `political.knowledge`, and in the second with `political.knowledge` varying and conditioning on `Europe`. Setting `strip= list(factor.names=FALSE)` suppresses the names of the conditioning predictor in each effect plot; these names are too long for the strips at the tops of the panels. From the first graph, preference for the Conservative Party increases with `Europe` for respondents with high political knowledge, but not for those with low political knowledge. More generally, voters with high political knowledge are more likely to align their votes with the positions of the parties, Eurosceptic for the Convervatives, pro-Europe for Labour and the Liberal Democrats, than are voters with low political knowledge.

6 The Lattice Theme for the effects Package

The **effects** package uses the `xyplot()` and `barchart()` functions in the standard **lattice** package (Sarkar, 2008) to draw effect plots. The **lattice** package has many options for customizing the appearance of graphs that are collected into a *lattice theme*. We created a custom theme for use with the **effects** package that automatically supersedes the default Lattice theme when the **effects** package is loaded, *unless the* **lattice** *package has been previously loaded*. You can invoke the **effects** package theme directly by the command

[4] Not exactly the same because in each plot the focal predictor takes on 50 values and the conditioning predictor 3 or 4 values.

```
R> effectsTheme()
```

You can also customize the **effects** package Lattice theme; see `help("effectsTheme")`. Finally, because `plot()` methods in the **effects** package return lattice objects, these objects can be edited and manipulated in the normal manner, for example by functions in the **latticeExtra** package (Sarkar and Andrews, 2016).

References

Fox, J. and S. Weisberg (2018). Visualizing fit and lack of fit in complex regression models with predictor effect plots and partial residuals. *Journal of Statistical Software 87*(9), 1–27.

Fox, J. and S. Weisberg (2019). *An R Companion to Applied Regression* (Third ed.). Sage.

Hawkins, D. M. and S. Weisberg (2017). Combining the Box-Cox power and generalised log transformations to accommodate negative responses in linear and mixed-effects linear models. *South African Statistics Journal 51*, pp. 317–328.

Sarkar, D. (2008). *Lattice: Multivariate Data Visualization With R*. Springer Science & Business Media.

Sarkar, D. and F. Andrews (2016). *latticeExtra: Extra Graphical Utilities Based on Lattice*. R package version 0.6-28.