Regression models in R

Antonio Gasparrini

01 November 2023

Contents

Regression models are nowadays essential statistical tools for data analysis. The availability of powerful computing systems allows complex regression routines, unimaginable only a few decades ago, to be implemented and performed cheaply and conveniently. R provides a thorough implementation of different regression models. If the object-oriented and functional style of R makes simple data management tasks less straightforward than in other statistical packages, these features represent an advantage in regression analysis. Different regression models can be fitted by specific functions following a common procedure, and fitted models are saved in objects, from which statistics or other components can be extracted for further computations. Regression models in R are covered in several books. Here we limit the description to the general structure of regression functions and the main steps in regression analysis. The first section introduces the model formula as a general definition of the regression model, while the second one reports some examples of performing linear and generalized linear models. The internal fitting structure of regression functions in R is covered in the third section, while the following sections address the issue of deriving residuals and predictions, as well as making inference from regression models. Functions for performing other regression models are summarized in the last section.

The model formula

A regression model defines a relationship between a set of *predictors* (or explanatory or independent) variables and a *response* (or dependent) variable. Predictors can be quantitative or qualitative categorical variables. In this section and the next ones, we will use the variables stored in the dataset BIRTHS of the package EPI to define regression models. We first load the package (which might also be installed, if not) and the dataset:

```
library(Epi)
data(births)
```

In R, such dependency is expressed through a *formula*. Such an object is a symbolic expression, also used in other contexts within R, which in this case offers a syntax specifically proposed for linear models. An example of a formula is `y ~ x1 + x2`, defining the regression model $E(y) = \beta_0 + \beta_1 x_1 + \beta_2 x_2$. This formula is a simple expression with a left-hand side specifying the response variable, a *tilde* operator '~', and a right-hand side specifying linear combinations of predictors. The coefficients β are not directly specified in the formula, while an intercept is included by default, so the formula above provides an identical result of `y ~ 1 + x1 + x2`. As common in R, a formula can be saved as an object of class `formula`, for example:

```
f1 <- bweight ~ gestwks
f2 <- bweight ~ gestwks + matage
```

The objects `f1` and `f2` can be used to specify a regression model with birth weight as the response variable and gestational week and/or maternal age as predictors (see `(help(births)` for details). The arithmetic operator '+' is commonly used to separate terms of the linear predictor on the right-hand side. Functions can be used on both sides of the model formula to transform the predictors. For instance, the expression `log(bweight) ~ log(gestwks) + matage` is a valid formula.

Categorical variables must be coded accordingly in regression models. In R, this process is automatic for variables corresponding to factors. In the case of the categorical variable `sex` in the data frame `births`, coded as a numeric variable with values 1-2, the transformation can be obtained by a call to the function `factor()` inside the formula:

```
bweight ~ gestwks + factor(sex)
```

Alternatively, the variable should be transformed in a factor externally. Interactions are commonly expressed with the operators '`:`' or '`*`':

```
bweight ~ gestwks:matage
bweight ~ gestwks*matage
```

The first expression is used to specify all the interactions between the terms, corresponding in this case to the single interaction between numeric variables, or alternatively to the main effects of a numeric variables for each category of a factor. The last expression is instead a *crossed interaction* and includes also the main terms: it corresponds to `bweight~ gestwks + matage + gestweeks:matage`. Alternative operators for specifying interactions are '`^`', indicating all the crossing interactions to a specified degree, and '`/`' (or identically '`%in%`'), indicating a *nested interaction*. See `help(formula)` for details.

The minus operator '`-`' is used to exclude main or interaction terms from the formula. It is often applied in the form `bweight ~ -1 + gestwks+matage` to express a model without intercept. Identically, `bweight ~ 0 + gestwks + matage`' suppresses the intercept.

The arithmetic operators above have a symbolic use in the right-hand side of model formulas. In case they are needed to transform a variable, they must be `protected` by the special function `I()`. For instance:

```
bweight ~ gestwks + I(gestwks^2) + matage
```

This expression includes a quadratic term directly specified in the model, protecting the symbol `'^'` that otherwise will be interpreted as an interaction (see above). Similarly, `y ~ I(x1+x2)` defines a model with a single predictor equal to the sum of `x1` and `x2`.

Linear and generalized linear models

The statement provided by the model formula defines a linear dependency between (potentially transformed) variables. This expression can now be used to fit the model with real data. As a simple example, we fit a *linear model* through the regression function `lm()` included in the standard package STATS, using the same formula saved in `f1`:

```
lm1 <- lm(bweight ~ gestwks, data=births)
```

The first argument of `lm()` is the formula, directly expressed or saved in an object, while the second argument `data`, usually a data frame, provides the environment where the predictors can be found. Alternatively, some or all of these can be stored in some environment present in the search path, usually the global environment. As usual in R, the model is saved in a regression object `lm1`, in this case with class `lm`.

The printing of regression objects usually returns limited information, with the output of objects of class `lm` including only the function call and the estimated coefficients:

```
lm1
```

```
##
## Call:
## lm(formula = bweight ~ gestwks, data = births)
##
## Coefficients:
## (Intercept)      gestwks
##       -4489          197
```

The method function `summary()` offers more details. Let's see an example with a more complex model:

```
lm2 <- lm(bweight ~ gestwks + matage + factor(sex), data=births)
summary(lm2)
```

```
##
## Call:
## lm(formula = bweight ~ gestwks + matage + factor(sex), data = births)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1787.4  -277.2   -17.0   266.9  1293.1
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -4345.290    373.979 -11.619  < 2e-16 ***
## gestwks        196.380      8.611  22.807  < 2e-16 ***
## matage          -0.883      5.113  -0.173    0.863
## factor(sex)2  -190.171     39.869  -4.770 2.44e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 440.5 on 486 degrees of freedom
##   (10 observations deleted due to missingness)
## Multiple R-squared:  0.5293, Adjusted R-squared:  0.5264
....
```

The lengthy output includes estimates, standard errors and p-values, residual distribution, and other statistics such as the R^2. Also, the number of observations deleted due to missingness is added in this case. This model indicates that gestational week and sex are significantly associated with birth weight, while maternal age is not.

A *generalized linear model* (GLM) can be fitted with the function `glm()` from the package STATS. As an example, we can fit a logistic regression using the indicator of low birth weight `lowbw` as the response variable:

```
glm1 <- glm(lowbw ~ gestwks, births, family=binomial(link="logit"))
summary(glm1)
```

```
##
## Call:
## glm(formula = lowbw ~ gestwks, family = binomial(link = "logit"),
##     data = births)
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  31.8477     4.0574   7.849 4.18e-15 ***
## gestwks      -0.8965     0.1084  -8.272  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 360.38  on 489  degrees of freedom
## Residual deviance: 205.75  on 488  degrees of freedom
##   (10 observations deleted due to missingness)
## AIC: 209.75
##
## Number of Fisher Scoring iterations: 6
....
```

The result of the call is saved in an object with classes `glm` and `lm`. The call is similar to that of `lm()`, but with the additional argument `family` which calls other functions to specify the error distribution and link of the GLM. In this case, this represents a logistic regression based on a *binomial* family with a canonical link logit, where the estimated coefficient of the predictor is interpreted as a log-odds ratio. Other families, such as *Poisson*, *gamma*, or *Gaussian* can be called, with different links. See `help(glm)` and `help(family)` for additional details.

The result of a regression function usually consists of a list with several components. In addition, usually the method function `summary()` returns another list with some additional components. For instance, let's explore the content of the object `lm1` and its summary:

```
names(lm1)
```

```
##  [1] "coefficients"  "residuals"     "effects"       "rank"          "fitted.values" "assign"
##  [9] "na.action"     "xlevels"       "call"          "terms"         "model"
```

```
names(summary(lm1))
```

```
##  [1] "call"          "terms"         "residuals"     "coefficients"  "aliased"       "sigma"
##  [9] "adj.r.squared" "fstatistic"    "cov.unscaled"  "na.action"
```

These components store the estimated coefficients, residuals and fitted values, and model statistics, in addition to information on the model fitting. Some of these components are common to various functions, while others depend on the specific regression model. All of them can be directly accessed to produce further computations such as tests or diagnostics. Also, several functions can be applied to regression objects to derive summaries or extend the results of the regression model, as shown later in this chapter.

Fitting procedures

Regression functions in R, such as `lm()` and `glm()` seen earlier, usually share similar arguments, other than `formula` and `data`. As an example, we can revise the model saved in `lm1` by adding some extra arguments:

```
lm1b <- lm(bweight~gestwks, data=births, subset=sex==1&hyp==0, na.action=na.exclude)
```

The argument `subset` is applied to select observations. This argument, similarly to its use in the function `subset()`, is usually an expression producing a logical vector to identify the records to be kept. In the example here, it restricts the fitting of the model to male babies with non-hypertensive mothers. Alternatively, the argument accepts a sequence of positive or negative indices to select the observations which must be included or excluded, respectively.

The other common argument `na.action` specifies how to deal with missing values. It calls specialized functions, by default the function `na.omit()` which causes the exclusion of the records with missing values in either the response variable or any of the predictors. A similar result is obtained through `na.exclude()` in the call above, with the difference that in this case residuals or predictions are padded to the correct length by re-inserting missing values for omitted observations, as shown later. These functions can be called with their name included in quotes, so `na.action="na.exclude"` is valid. See `help(na.action)` for further details.

Other important arguments common to various regression functions are `weights`, `contrasts` and `offset`. The first of these arguments accepts a numeric vector with weights to be applied in the fitting process. The argument `contrasts` is used to define the parametrisation of the contrasts for the categories of factors included in the model formula, as shown later in this section. The default is to contrast each level with the reference category. The argument `offset` adds terms in the linear predictor with known coefficient 1, and is often used in Poisson models and other GLMs.

More generally, regression functions follow a common procedure of model fitting, which makes it easier to implement extensions or new regression techniques. The procedure involves the internal call to specific functions, of which the most important are `model.frame()` and `model.matrix()`. Although these functions are not expected to be called directly when fitting regression models, they are useful to understand the computing approach to regression used in R. The function `model.frame()` is called to produce a *model frame*, namely a special data frame object containing the variables used in the model fitting. The function can be called on an existing regression object or a model formula and data objects, for example:

```
mframe <- model.frame(lm2)
head(mframe, 3)
```

```
##   bweight gestwks matage factor(sex)
## 1    2974   38.52     34          2
## 3    2620   38.15     35          2
## 4    3751   39.80     31          1
```

The code above shows the content of the model frame of the regression model saved in `lm2`, including only the (potentially transformed) variables in `births` used for the fitting. Among other arguments, the function `model frame()` accepts `subset` and `na.action` from the original call to the regression functions to select the observations and exclude missing values, respectively. In the example above, you may notice from the row names that the second record of `births` was excluded due to a missing value in the variable `matage`.

The second important step in the internal procedure of regression functions is the creation of the *design matrix*, where each term is converted into one or more regressors. This step is produced by the function `model.matrix()`, which again accepts as arguments an existing regression object or alternatively an entire or right-hand model formula. Let's see an example using the variables in `births`, first creating a new one called `matagegr` by categorising `matage` :

```
births$matagegr <- cut(births$matage, breaks=c(0,30,35,100), right=F)
Xdes <- model.matrix(~ gestwks * matagegr + factor(hyp), births)
head(Xdes, 3)
```

```
##   (Intercept) gestwks matagegr[30,35) matagegr[35,100) factor(hyp)1 gestwks:matagegr[30,35) gestwks:
## 1           1   38.52               1                0           0                    38.52
## 3           1   38.15               0                1           0                     0.00
## 4           1   39.80               1                0           0                    39.80
```

The output shows how the predictors in the model are transformed in multiple regressors defined through dummy parametrisation as well as main/interaction terms for the model fitting.

Functions such as `model.response()` and `model.offset()` can be used to extract specific components from a model frame (see `help(model.extract)`). The preparatory steps described above, common to most regression techniques, largely simplify the application of routines expressly dedicated to model fitting procedures, for example the least square estimation used in linear models.

Residuals, model diagnostics, and predictions

The fit of a specific model is usually assessed through regression diagnostics, applied for checking potential departures from the model assumptions. The functions `residuals()` and `fitted()` can be used to extract residuals and fitted values, important quantities in model checking, from a regression object. A simple check involves the plotting of residuals from the model saved in the object `lm2` in a previous section:

```
res <- residuals(lm2)
plot(res, col=4, ylim=c(-2000,2000), ylab="Residuals")
abline(h=0)
```

The kind of graphs, shown in the left panel of the figure below, is commonly used to identify a lack of fit for some observation. Another common check is to plot the residuals versus a continuous predictor to test the linearity assumption. We can produce this plot for the predictor `gestwks`. However, the vector of residuals `res` created above from model `lm2` has a different length, due to the exclusion of missing values from model fitting. As anticipated above, this issue can be solved by calling the function `na.exclude()` as argument `na.action` in `lm()`. Rather than refitting the model entirely, we can use the function `update()` with the new argument (see `help(update)`) and save a new model object, then finally extract the residuals:

```
lm2b <- update(lm2, na.action=na.exclude)
resb <- residuals(lm2b)
head(res, 3)
```

```
##         1         3         4
##  -25.06116 -305.51787  307.75260
```

```
head(resb, 4)
```

```
##         1         2         3         4
##  -25.06116        NA -305.51787  307.75260
```

The function `residuals()` called on the updated model now correctly pads missing values in the residuals to the correct length. The graph, shown in the mid panel below, is produced with:

```
plot(resb~gestwks, births, col=3, ylim=c(-2000,2000), xlab="Gestational week",
  ylab="Residuals")
abline(h=0)
lines(lowess(births$gestwks, resb, delta=0.1, f=1/3), col=2, lwd=1.5)
```

The function `lowess()` is called here to add a smoothed fit of the residuals using a locally-weighted polynomial regression. See `help(lowess)` for details.

Another common graph is the normal Q-Q plot to check the assumption of normality of the error distribution, provided by the high-level plotting function `qqnorm()` (see the related help page). The graph requires the computation of *standardized* residuals, obtained through the function `rstandard()`:

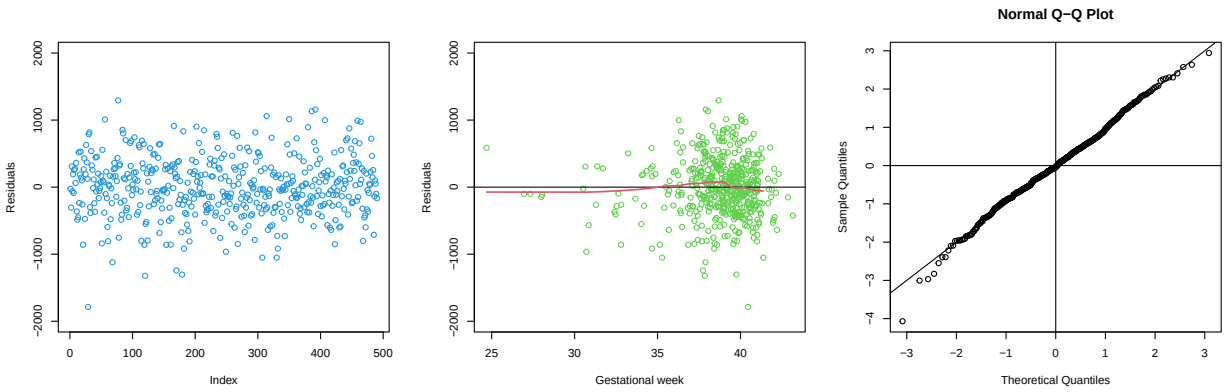

Figure 1: Residual distribution (left), testing the linearity assumption (mid), and a normal Q-Q plot (right)

```
standres <- rstandard(lm2b)
qqnorm(standres)
abline(a=0, b=1, h=0, v=0)
```

The plot is shown in the right panel (see `help(abline)` for details on the last expression to add straight lines to a plot). The same types of residuals used above can be computed for generalized linear models. In this case, the basic quantities are the *working* residuals computed in the scale of the linear predictor. Other important measures are the *response*, *Pearson*, and *deviance* residuals, obtained through the additional argument `type` in the method function `residuals()` for GLM objects (see `help(residuals.glm)`).

Many other functions are available for computing quantities used in model checking in linear and generalized linear models, such as *studentized* (or *jackknife*) residuals, *DFBETAs*, *Cook's distance*, and *hat values*. See `help(influence.measures)` for an overview.

Another important part of the model assessment is to obtain a visual representation of the model fit. In the simple case of a single predictor as in model `lm1`, the regression line can be plotted with the observed values using the function `abline()`:

```
plot(bweight ~ gestwks, data=births, col=grey(0.7))
abline(lm1, col=2, lwd=1.5)
```

The resulting graph (left panel in the figure below) adds the regression line to the scatter plot between birth weight and gestational week. More generally, the fit can be represented by predictions, obtained in R through the method function `predict()`:

```
pred <- predict(lm2b)
head(pred)
```

```
##        1        2        3        4        5        6
## 2999.061       NA 2925.518 3443.247 3262.776 3481.075
```

When simply called to the regression object as in the code above, the function returns results identical to `fitted()`, meaning the fitted values. However, one of the most important advantages of regression models in statistical analysis is the possibility to use the estimates to predict the outcome for specific combinations of the predictors which have not been observed. For instance, using the model `lm2` we can predict the birth weight at different gestational weeks for constant values of maternal age and sex:

```
newdata <- data.frame(gestwks=25:43, matage=mean(births$matage, na.rm=T), sex=1)
newpred <- predict(lm2, newdata=newdata, se.fit=T)
names(newpred)
```

```
## [1] "fit"            "se.fit"         "df"             "residual.scale"
```

```
head(newpred$fit)
```

```
##         1         2         3         4         5         6
##  534.1546  730.5343  926.9140 1123.2936 1319.6733 1516.0530
```

The graph is shown in the mid panel. The function `predict()` here accepts two additional arguments: `newdata`, with a data frame containing the combinations of the values of the predictors, and `se.fit=TRUE` (default to `FALSE`), stating that standard errors of the predictions must be added. In this case, the function returns a list of components including `fit` and `se` with predictions and standard error, respectively. These quantities are used to plot the predicted association between gestational week and birth weight including 95% confidence intervals (using the 1.96 multiplier):

```
plot(25:43, newpred$fit, type="l", xlim=c(23,45), ylim=c(0,5000),
  xlab="Gestational week", ylab="Birth weight", col=4, lwd=1.5)
lines(25:43, newpred$fit+1.96*newpred$se, col=4, lty=2)
lines(25:43, newpred$fit-1.96*newpred$se, col=4, lty=2)
```

The method function `predict()` can also be applied to GLMs. In this case, the additional argument `type` can be added to specify the scale over which the prediction should be computed, namely the linear predictor (`type="link"`, the default) or the response obtained by inverting the link function (`type="response"`). For example, we can compute the proportion of newborns of low birth weight at different gestational weeks at constant values of maternal age and sex, using the same `newdata` object:

```
newpred2 <- predict(glm1, newdata=newdata, type="response")
plot(25:43, newpred2, type="l", xlim=c(23,45), xlab="Gestational week",
  ylab="Proportion og low birth weight", col=2, lwd=1.5)
abline(v=37, lty=2)
```

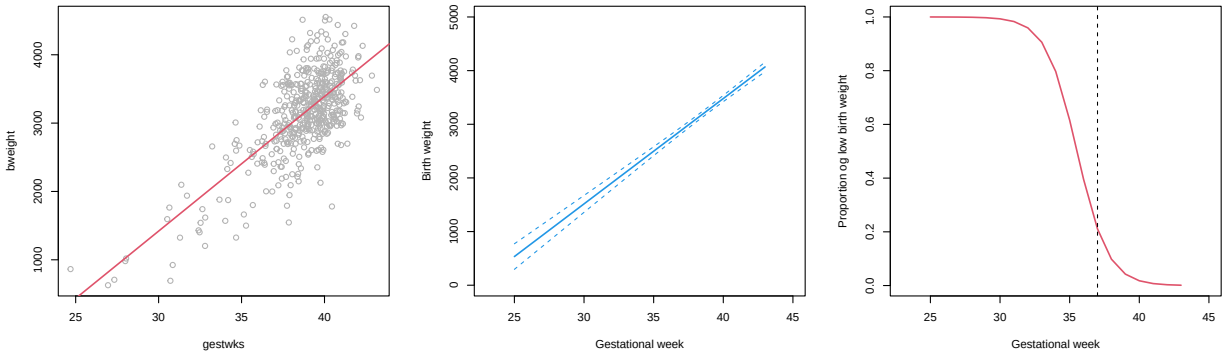

Figure 2: Prediction from the univariate (left) and multivariate (mid) linear models, and for the generalized liner model

The graph, displayed in the right panel, also illustrates the probability at 37 weeks, corresponding to:

```
predict(glm1, data.frame(gestwks=37), type="response")
```

```
##         1
## 0.2105899
```

Confidence intervals can be added to the predicted probability by setting `se.fit=T`, as in the previous example.

Inference and model selection

The output of the method function `summary()`, illustrated earlier, provides some statistics for making basic inference on the model parameters, namely the estimates, standard error, and results for the t and z tests (for linear and generalized linear models, respectively) with related 95% confidence intervals. Such results can be extended through the use of several other method functions applied to regression objects. Key tools are `anova()` and the related functions `drop1()` and `add()`, which offer tests from tables of the *analysis of variance* and *deviance* in linear and generalized linear models, respectively. When we call `drop1()` on a the model object `lm2`, it returns the following:

```
drop1(lm2, test="F")
```

```
## Single term deletions
##
## Model:
## bweight ~ gestwks + matage + factor(sex)
##             Df Sum of Sq       RSS    AIC  F value    Pr(>F)
## <none>                    94284733 5970.0
## gestwks      1 100909746 195194479 6324.6 520.1493 < 2.2e-16 ***
## matage       1      5787  94290519 5968.1   0.0298     0.863
## factor(sex)  1   4413954  98698687 5990.5  22.7522  2.44e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The function runs a series of *hypothesis* (or *significance*) tests by removing each term from the model. In this case, the argument `test="F"` selects an F test based on a $\mathcal{F}_{p,N-p}$ distribution, with p as the number of parameters and N as the number of observations. Alternatively, a Wald test based on a χ^2_p distribution is performed with `test="Chisq"`. Other quantities such as the *residual sum of squares* (RSS) and the *Akaike information criterion* (AIC) are returned as well.

In this case, the results are identical to the output of `summary()` seen above, as F tests for single coefficients correspond exactly to t tests. Hypothesis tests involving multiple terms can be instead performed using the function `anova()`, which can be used to compare the fit of two (nested) models. For example:

An advantage of the `anova()` function is that the tests are performed on the predictors, not the regressors, so that multiple parameters such as those of factors with more than two categories are tested all together. Also, the function can extend the hypothesis testing procedure to multiple factors by comparing two or more models:

```
anova(lm1, lm2, test="F")
```

```
## Analysis of Variance Table
##
## Model 1: bweight ~ gestwks
## Model 2: bweight ~ gestwks + matage + factor(sex)
##   Res.Df      RSS Df Sum of Sq      F    Pr(>F)
## 1    488 98698698
## 2    486 94284733  2   4413965 11.376 1.484e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The code above tests the combined effect of maternal age and sex, by comparing the fit of `lm1` (which does not include them) with `lm2` (which does). Be aware that model nesting is not internally checked and it is expected that the user defines a valid comparison.

The same functions can be applied to GLMs. As an example, we first fit another model by extending `glm1` through the function `update()` seen above:

```r
glm2 <- update(glm1, . ~ . + factor(hyp))
```

The second unnamed argument `formula.` of `updates()` changes the model formula by using a symbolic expression, where the dot '.' represents all the existing terms currently present on the left or right-hand sides. The arithmetic operators '+' and '−' can be used to add or eliminate terms to or from the formula, respectively. In this specific case, maternal hypertension is added to the model as a categorical variable. Then the test:

```r
anova(glm1, glm2, test="LR")
```

```
## Analysis of Deviance Table
##
## Model 1: lowbw ~ gestwks
## Model 2: lowbw ~ gestwks + factor(hyp)
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1       488     205.75
## 2       487     203.07  1   2.6744    0.102
```

The method function `anova()` offers additional types of tests for object of class `glm`. In this case, we select a *likelihood ratio* (LR) test with `test="LR"`. The results indicate that there is weak evidence that maternal hypertension is associated with a change in the odds of low birth weight.

Other functions are available for inference and model selection. Among others, `coef()` and `vcov()` extract the coefficients and associated (co)variance matrix, respectively; the function `logLik()` returns the log-likelihood of the model; `df.residual()` provides the residual degrees of freedom; finally `AIC()` provides the Akaike information criterion. Also, `step()` performs stepwise selection using the functions `add1()` and `drop1()` to test single parameters entering or exiting the model, respectively.

Other regression models

R provides functions for fitting several other types of regression models, many of them available in recommended contributed packages. A thorough introduction to each regression method is beyond the scope of these notes, and a list of the main functions and packages is provided in the table below.

Model	Function	Package
Linear mixed-effect models	`lme()`, `lmer()`	NLME, LME4
Generalized linear mixed-effect models (GLMM)	`lmer()` , `glmer()`, `glmmPQL()`	LME4, MASS
Generalized estimating equations (GEE)	`gee()`, `geeglm()`	GEE, GEEPACK
Cox proportional hazard	`coxph()`	SURVIVAL
Conditional logistic regression	`clogit()`	SURVIVAL
Generalized additive models (GAM)	`gam()`	MGCV
GLM with negative binomial family	`glm.nb()`	MASS
Proportional odds logistic regression	`polr()`	MASS
Multinomial log-linear models	`multinom()`	NNET
Meta-analysis and meta-regression	`rma()`	METAFOR
Non-linear least squares	`nls()`	STATS
Non-linear mixed-effect models	`nlme()`, `nlmer()`	NLME, LME4
Neural networks	`nnet()`	NNET
Random forests	`ranger()`	RANGER

Briefly, regression functions in R share some common features described in previous sections. Usually, the

model is defined in the first argument by a regression formula, using a tilde operator '$\sim$'. Variables can be found in the additional argument `data`, and specific `model.frame()` and `model.matrix()` method functions are applied to deal with missing values and prepare the design matrix. Although the fitting procedure changes from model to model, several other functions introduced earlier, such as `residuals()` and `anova()`, are generally available.