but it is much more straightforward to use the `subset` argument, especially when there are lots of explanatory variables:

```r
model <- lm(Fruit ~ Root, subset = (Grazing == "Grazed"))
```

The answer, of course, is the same in both cases, but the `summary.lm` and `summary.aov` tables are neater with `subset`. Note the round brackets used with the `subset` option (not the square brackets used with subscripts in the first example).

### Weights

The default is for all the values of the response to have equal weights (all equal one)

```r
weights = rep(1, nobs)
```

There are two sorts of weights in statistical modelling, and you need to be able to distinguish between them:

- case weights give the relative importance of case, so a weight of 2 means there are two such cases;
- inverse of variances, in which the weights downplay highly variable data.

Instead of using initial root size as a covariate (as above) you could use `Root` as a weight in fitting a model with `Grazing` as the sole categorical explanatory variable:

```r
model <- lm(Fruit ~ Grazing, weights = Root)
summary(model)
```

```r
Call:
  lm(formula = Fruit ~ Grazing, weights = Root)

Coefficients:  
Estimate   Std. Error   t value   Pr(>|t|)
(Intercept)  70.725       4.849    14.59 <2e-16 ***
GrazingUngrazed  -16.953      7.469    -2.27 0.029 *

Residual standard error: 62.51 on 38 degrees of freedom
Multiple R-Squared: 0.1194,  Adjusted R-squared: 0.0962
F-statistic: 5.151 on 1 and 38 DF,  p-value: 0.02899
```

Needless to say, the use of weights alters the parameter estimates and their standard errors:

```r
model <- lm(Fruit ~ Grazing)
summary(model)
```

```r
Coefficients:  
Estimate   Std. Error   t value   Pr(>|t|)
(Intercept)  67.941       5.236    12.976 <2e-16 ***
GrazingUngrazed  -17.060      7.404    -2.304 0.0268 *

Residual standard error: 23.41 on 38 degrees of freedom
Multiple R-Squared: 0.1226,  Adjusted R-squared: 0.09949
F-statistic: 5.309 on 1 and 38 DF,  p-value: 0.02899
```

When weights (`w`) are specified the model is fitted using weighted least squares, in which the quantity to be minimized is \( \sum w \times d^2 \) (rather than \( \sum d^2 \)), where `d` is the difference between the response variable and the fitted values predicted by the model.
Missing values

What to do about missing values in the dataframe is an important issue (p. 120). Ideally, of course, there are no missing values, so you don’t need to worry about what action to take (na.action). If there are missing values, you have two choices:

- leave out any row of the dataframe in which one or more variables are missing, then na.action = na.omit
- fail the fitting process, so na.action = na.fail

If in doubt, you should specify na.action = na.fail because you will not get nasty surprises if unsuspected NAs in the dataframe cause strange (but unwarned) behaviour in the model. Let’s introduce a missing value into the initial root weights:

```r
Root[37]<-NA
model<-lm(Fruit ~ Grazing*Root)
```

The model is fitted without comment, and the only thing you might notice is that the residual degrees of freedom is reduced from 36 to 35. If you want to be warned about missing values, then use the na.action option.

```r
model<-lm(Fruit ~ Grazing*Root,na.action=na.fail)
Error in na.fail.default(list(Fruit = c(59.77, 60.98, 14.73, 19.28,
34.25, : missing values in object
```

If you are carrying out regression with time series data that include missing values then you should use na.action = NULL so that residuals and fitted values are time series as well (if the missing values were omitted, then the resulting vector would not be a time series of the correct length).

Offsets

You would not use offsets with a linear model (you could simply subtract the offset from the value of the response variable, and work with the transformed values). But with generalized linear models you may want to specify part of the variation in the response using an offset (see p. 518 for details and examples).

Dataframes containing the same variable names

If you have several different dataframes containing the same variable names (say x and y) then the simplest way to ensure that the correct variables are used in the modelling is to name the dataframe in the function call:

```r
model<-lm(y ~ x,data=correct.frame)
```

The alternative is much more cumbersome to type:

```r
model<-lm(correct.frame$y ~ correct.frame$x)
```