10

Regression

Regression models provide a flexible tool to understand the drivers behind an outcome, response, or phenomenon of interest. These models can be used to complete a variety of tasks, including estimating the collective relationship between variables, hypothesis testing, and prediction of future outcome values.

Perhaps the simplest form of a regression is the **simple linear regression model**, which quantifies the **linear** relationship between a **covariate** (otherwise known as a predictor), X, and an **outcome** (otherwise known as response) of interest, Y. This model assumes a linear – or straight line – relationship between X and Y, of the form

$$Y = \beta_0 + \beta_1 X. \tag{10.1}$$

Box 10.1: Terminology

In other texts, the covariate may be referred to as an 'independent variable' and the outcome as a 'dependent variable', but this is misleading especially in terms of our covariate: the term 'independent' has a special meaning in statistics (see Section 8.1.3),and the covariate here is not necessarily 'independent' of the outcome. Later in this chapter, we'll also consider having multiple covariates, and these do not need to be statistically independent of each other. It's best, therefore, to avoid the term 'independent' variable and use a neutral term like 'covariate' or 'predictor' instead.

Of course, the model in (10.1) is a simplification of the real-world relationship between X and Y, and in particular, we don't expect every observation to lie exactly on this line. Our aim therefore, at least in part, is to find the 'best' estimates of the intercept, β_0, and slope, β_1, of this line, to create what is often referred to as a 'line of best fit'. These models and their extensions are the basis of this chapter.

In this chapter, we consider:

- simple linear regression models;
- multiple linear regression models;

The R Book, Third Edition. Elinor Jones, Simon Harden and Michael J. Crawley.

Companion website: www.wiley.com/go/jones/therbook3e

- hypothesis testing in simple and multiple linear regression models;
- checking the assumptions made on simple and multiple linear regression models;
- improving simple and multiple linear regression models (improving model fit).

The models we consider here can be substantially extended, for example in allowing non-linear relationships between covariates and outcome. Later in the book, we cover such extensions (see Table 10.1 in Section 10.7 for a list of *R* functions relating to these types of regression models).

10.1 The simple linear regression model

Let us start with an example which shows the growth of caterpillars fed on experimental diets differing in their tannin content:

```
caterpillardata <- read.table ("caterpillar.txt" , header = T)
attach (caterpillardata)
head (caterpillardata)
```

```
  growth tannin
1     12      0
2     10      1
3      8      2
4     11      3
5      6      4
6      7      5
```

```
plot (tannin, growth, pch = 19, col = hue_pal ()(3)[1],
      xlab = "% tannin in diet", ylab = "growth rate")
```

```
detach (caterpillardata)
```

Though there is no straight line that will pass through all points on the scatterplot in Figure 10.1, (it doesn't need to), there is a general linear trend: a well-chosen straight line through the scatter of points would describe the relationship between tannin and growth well. We would expect this line to have a negative slope, describing the nature of the relationship: the higher the percentage of tannin in the diet, the more slowly the caterpillars grew.

This section looks at the underlying model, the assumptions we make about our data in fitting this model, and how we do so in *R*.

10.1.1 Model format and assumptions

Let us suppose that we have n pairs of observations of X and Y, $(x_1, y_1), \ldots, (x_n, y_n)$. We'll assume that the ith observed value of the outcome, y_i, is related to the ith observed value of the covariate (or predictor), x_i, by way of a straight line plus an error term.

We can write out the general form as follows:

$$Y_i = \beta_0 + \beta_1 x_i + \epsilon_i \, . \tag{10.2}$$

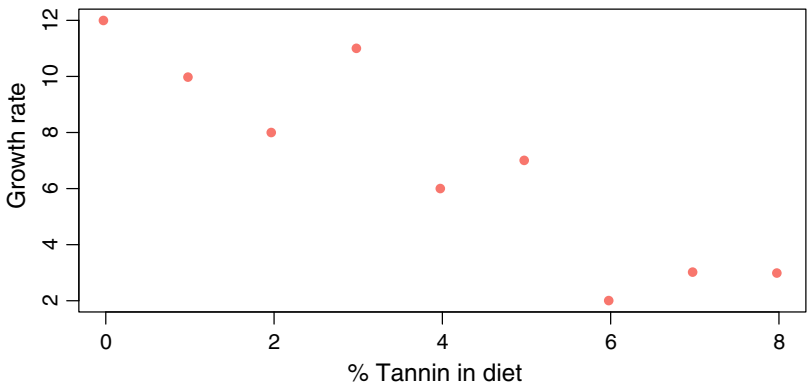


Figure 10.1 Caterpillar growth rate and percentage tannin in diet.

The error term, ϵ_i, is in recognition of the fact that the observations do not need to lie exactly on a straight line. Here, error does not mean 'mistake', but refers to unexplained variation about the line.

Box 10.2: Two important points

First, the distinction between $\{X, Y\}$ and $\{x, y\}$ is important. Lower-case letters are used to denote *observed* data, while capital letters are used to denote random variables (think of this as *before* we observe the data). In (10.2), the covariate is thought of as 'fixed' or observed, but the error term is a random variable. The latter implies that the outcome is also a random variable.

Second, though not explicitly mentioned here, the outcome is a numeric variable and generally a continuous variable. We look at what we can do when this is not the case in later chapters.

There is one obvious assumption that we make about this model: its linear structure. More often than not, however, we also need to make assumptions about the error term. Though the latter is not necessary to find the 'best' estimate of the intercept and slope, it is essential if, for example we want to test hypotheses with our regression model, or provide confidence intervals for predictions. The assumptions we make about the structure of the model, and about the error terms, can be simply stated as follows:

- the relationship between the outcome and covariate is linear;
- the error terms are independent: ϵ_i and ϵ_j are independent, when $i \neq j$;
- the error terms are normally distributed: $\epsilon_i \sim N(0, \sigma^2)$;
- the error terms have the same variance, say σ^2.

The first assumption implies that the relationship between X and Y can be described by a straight line – even if the observations themselves do not lie exactly on any line. For simple linear regression models this can be checked using a scatterplot. Figure 10.2 shows four examples of fitting a simple linear regression model to different data, with commentary below.

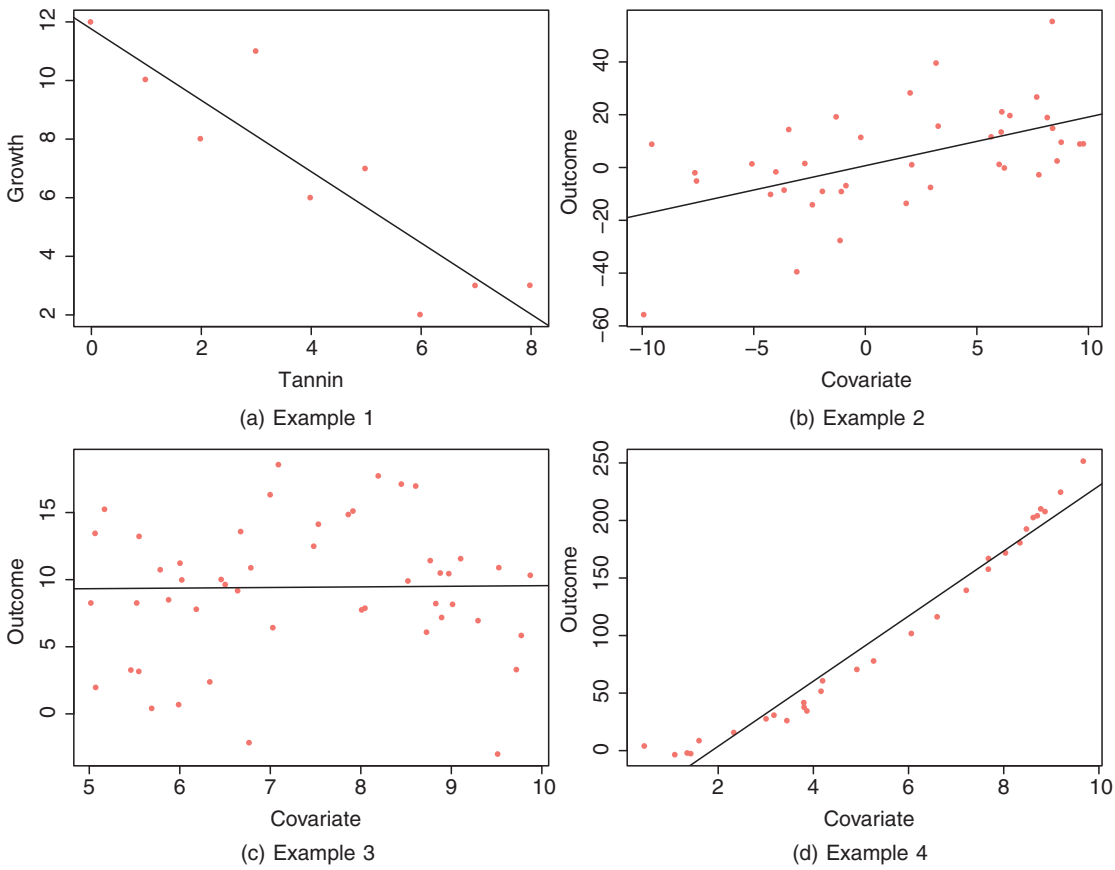


Figure 10.2 Simple linear regression models.

(a) This is the tannin-growth example, where a straight-line model seems to be a good fit for the data.

(b) The relationship here could still be described by a straight line despite there being a lot of variability around the line.

(c) There is little to no relationship between the covariate and outcome here. We can still fit a simple linear regression model, but this time the slope will be close to zero (why does that make sense?).

(d) While a strong relationship between the covariate and outcome is evident here, it isn't linear. Attempting to use simple linear regression here would be foolish. See Section 10.5.8 for suggestions of what could be done in this case.

There are many ways in which we could choose to find the 'best' slope and intercept for any given scatterplot, but they are generally computed using the so-called 'least squares' technique. This procedure minimises ('least') the squared vertical distance ('squares') between each observation and the straight line. These distances (before squaring) are known as **residuals** and are shown by solid blue lines in Figure 10.3. Notice what is happening here: for a specific value x, we are comparing the observed outcome y to the outcome that is predicted by the model (vertical distance between the green dotted horizontal lines for one observation).

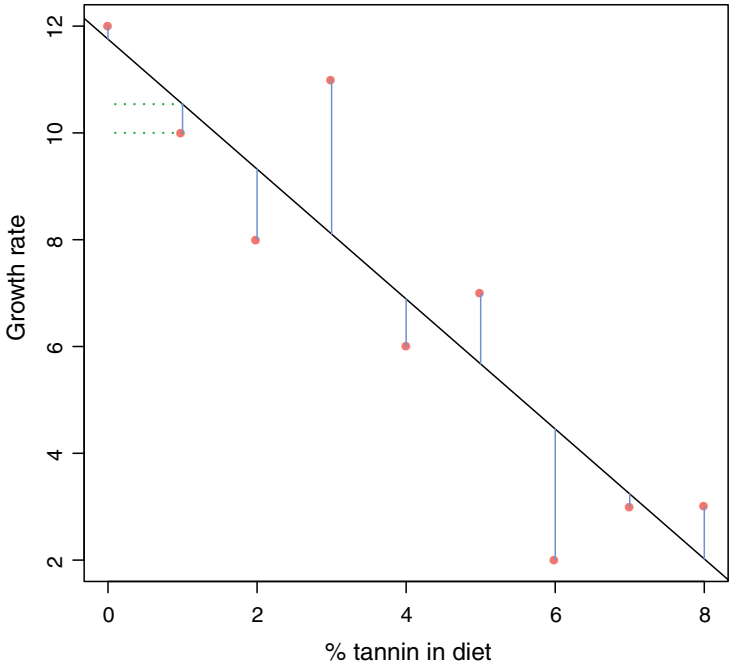


Figure 10.3 Distance between observations and line.

Why do we minimise the *squared* distance and not some other measure of distance between these points? There are convincing mathematical reasons why this is the case which are beyond the scope of this book, but rest assured that this is the method that *R* uses.

As we will see later in this chapter however, merely estimating the slope and intercept is rarely enough. Usually, we want to do a lot more with our regression model, for which we need to impose assumptions on the error term.

Our error terms, ϵ, are assumed to be independent and follow a Normal distribution with the same variance. Note that it makes sense that the mean of the error terms is zero (this is not really an assumption, per se): suppose that the mean error is, say, 3. Then this implies that we are systematically underestimating our outcome by 3. This can be easily remedied by adding 3 to the intercept, which means that the error term mean is then zero. Incidentally, a common misconception is that the outcome needs to be normally distributed, but this is not the case: we are assuming that the error terms are normally distributed, which happens to have implications for the format of the outcome, but we won't discuss this further here.

Though these assumptions on the error term are often 'hidden', these are exactly what *R* assumes in order to provide much of the model output. We'll consider this output in detail in Section 10.2, but for now we'll look at how to use *R* to run a simple linear regression model.

10.1.2 Building a simple linear regression model

The basic steps in building a simple linear regression model are shown in Figure 10.4. The first step is good practice regardless of the analysis that you plan to undertake, and indeed often helps to clarify which analyses might be appropriate. Cleaning will ensure that the data are in an appropriate format and can help in the detection of problems at an early stage.

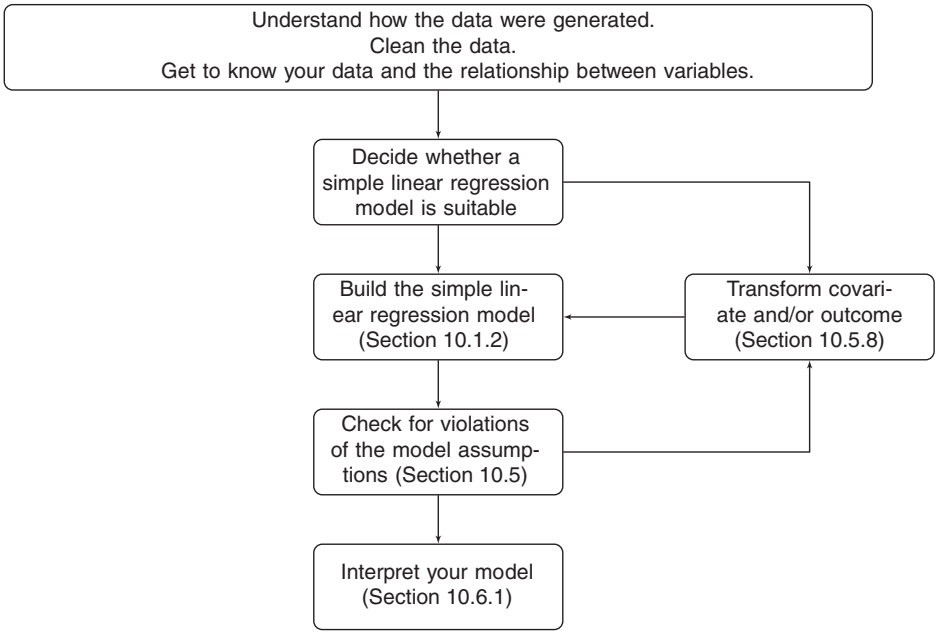


Figure 10.4 Building a simple linear regression model.

The second step – in this case deciding whether the simple linear regression model is an appropriate approach – will generally involve checking that the relationship between the covariate and outcome is linear. This can be simply done using a scatterplot. Steps can be taken at this point to improve any issues around linearity; see Section 10.5.8 on how transformation of variables may help here.

The third step is what we'll consider in this section: building the simple linear regression model in *R*. The remaining suggested steps are discussed in Sections 10.5 and 10.6.1, respectively.

Running a simple linear regression model in *R* is best done using the `lm ()` command (it stands for Linear Model). For the caterpillar growth example, the following code loads the required data and then runs the `lm ()` command on the variables of interest:

```
attach (caterpillardata)
lm (growth ~ tannin)


Call:
lm(formula = growth ~ tannin)

Coefficients:
(Intercept)       tannin
     11.756       -1.217

detach (caterpillardata)
```

Notice the way the variables growth and tannin are fed to the command `lm ()`: we insert the outcome first, followed by a twiddle symbol ~, and then the covariate. We don't have to `attach ()`

our data in order to run `lm ()`: the following code provides identical output where we specify the dataset to be used *within* the `lm ()` command.

```
lm (growth ~ tannin, data = caterpillardata)
```

The output from `lm ()` gives an estimate of the intercept and slope, but we can ask for more information (which will be vital later in the chapter). The most convenient way of doing this is to save the output of the `lm ()` command in an object – called `caterpillar_model` below – then request a summary of this object. This has the advantage that the output is accessible without having to run the model again.

```
caterpillar_model <- lm (growth ~ tannin, data = caterpillardata)
summary (caterpillar_model)


Call:
lm(formula = growth ~ tannin, data = caterpillardata)

Residuals:
    Min      1Q  Median      3Q     Max
-2.4556 -0.8889 -0.2389  0.9778  2.8944

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  11.7556     1.0408  11.295 9.54e-06 ***
tannin       -1.2167     0.2186  -5.565 0.000846 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.693 on 7 degrees of freedom
Multiple R-squared:  0.8157,Adjusted R-squared:  0.7893
F-statistic: 30.97 on 1 and 7 DF,  p-value: 0.0008461
```

An alternative, that will provide identical output but without saving the information in an object, is

```
summary (lm (growth ~ tannin, data = caterpillardata))
```

Box 10.3: Extracting information from the model output

We'll discuss the majority of the output later in the chapter, but for now

- The estimates of the slope and intercept are available in the `Estimate` column.
- The estimated standard deviation of the error term is known as `Residual standard error` in the output. Squaring this number gives an estimate of the variance of the error term.
- Information about the residuals – their five number summary – is given under the heading `Residuals`.

10.2 The multiple linear regression model

While the simple linear regression model provides a convenient way of describing the (linear) relationship between a covariate and an outcome, what should we do if we have more than one possible covariate?

Consider the air pollution dataset for example: our research question of interest may be to investigate how ozone concentration is related to wind speed, air temperature, and the intensity of solar radiation. Specified like this, it is clear that ozone is the outcome of interest, but there are three possible covariates to choose from: wind speed, air temperature, and the intensity of solar radiation. A good place to start might be to view the relationship between each pair of variables, which the `pairs ()` function automates. The resulting scatterplots are in Figure 10.5. Notice that the six plots under the diagonal are mirror images of those above.

```
ozone_pollution <- read.table ("ozone_pollution.txt", header = T)
attach (ozone_pollution)
pairs (ozone_pollution, col = hue_pal ()(3)[1])
```

```
detach (ozone_pollution)
```

The outcome variable, ozone concentration, is shown on the *y*-axis of the bottom row of panels: there is a strong negative relationship with wind speed, a positive correlation with temperature, and a rather unclear, humped relationship with radiation. There are also clear relationships between some of the proposed covariates, for example between wind speed and air temperature.

How do we make the most of all this information about the response? One option is to build a simple linear regression models for each possible covariate. This, however, is an unsatisfactory solution. We wouldn't be making the most of the *combined* information about the outcome contained in our collection of covariates.

A **multiple linear regression model** does just that: it extends the idea of a simple linear regression to allow more than one covariate. This is an exceptionally powerful idea and is a technique that is used extensively in practice.

10.2.1 Model format and assumptions

The multiple linear regression model handles data that come in the form $(y_i, x_{i1}, \ldots, x_{ip})$ for $(i = 1, \ldots, n)$. That is, for the ith observation, we have a single observed outcome y_i and in this case, p covariates, $x_{i1}, \ldots, x_{ip}$. The model is written as

$$Y_i = \beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip} + \epsilon_i \,. \tag{10.3}$$

Compare this with the general form of the simple linear regression model, and we'll see that this is a 'natural' extension of it. Indeed, the simple linear regression model is just a special case of the multiple linear regression model.

For the simple linear regression model, there was a nice geometric interpretation: we could visualise the model as a straight line passing through a scatterplot, or in other words, a line in two-dimensional space. What does the model in (10.3) 'look' like? This model with p covariates as in (10.3) will produce a **hyperplane**: with p covariates this is a hyperplane in $(p + 1)$-dimensional space. Anything above two covariates therefore means we can't visualise our scatter of points and the resulting hyperplane!

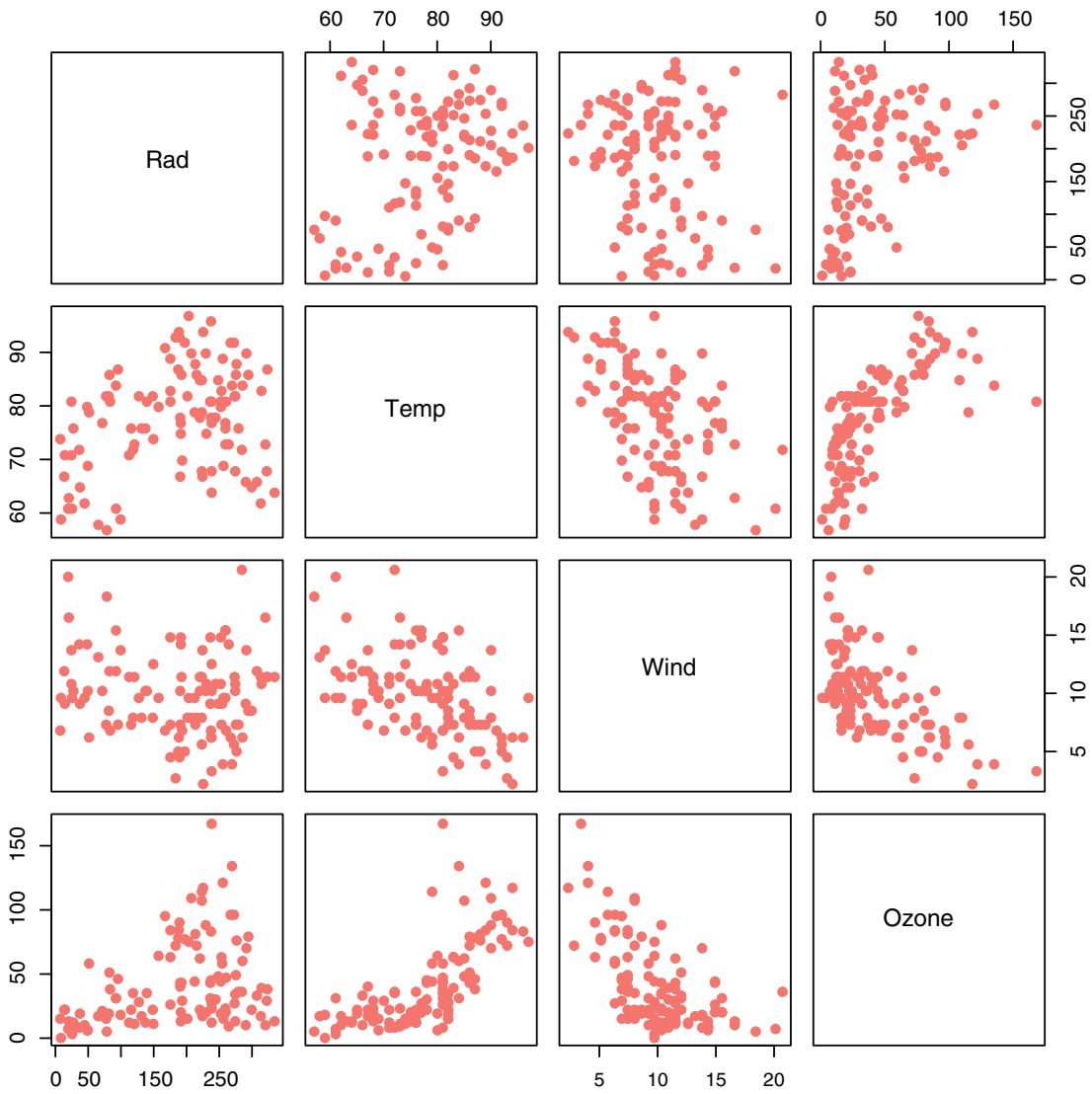


Figure 10.5 Scatterplots of all possible pairs of variables in the ozone dataset.

There are, of course, assumptions to be made in building this model. Once again, we assume linearity, but unlike in the simple linear regression case, we can't easily check this assumption by creating a single scatterplot. We will need alternative methods, which we'll revisit in Section 10.5, to check this assumption. The same assumptions apply to the error terms as did for the simple linear regression model; we'll also discuss how we can check these assumptions in Section 10.5.

10.2.2 Building a multiple linear regression model

Building a simple linear regression model was straightforward as we had only one covariate to consider. With more covariates comes added complexity and the process becomes more of an

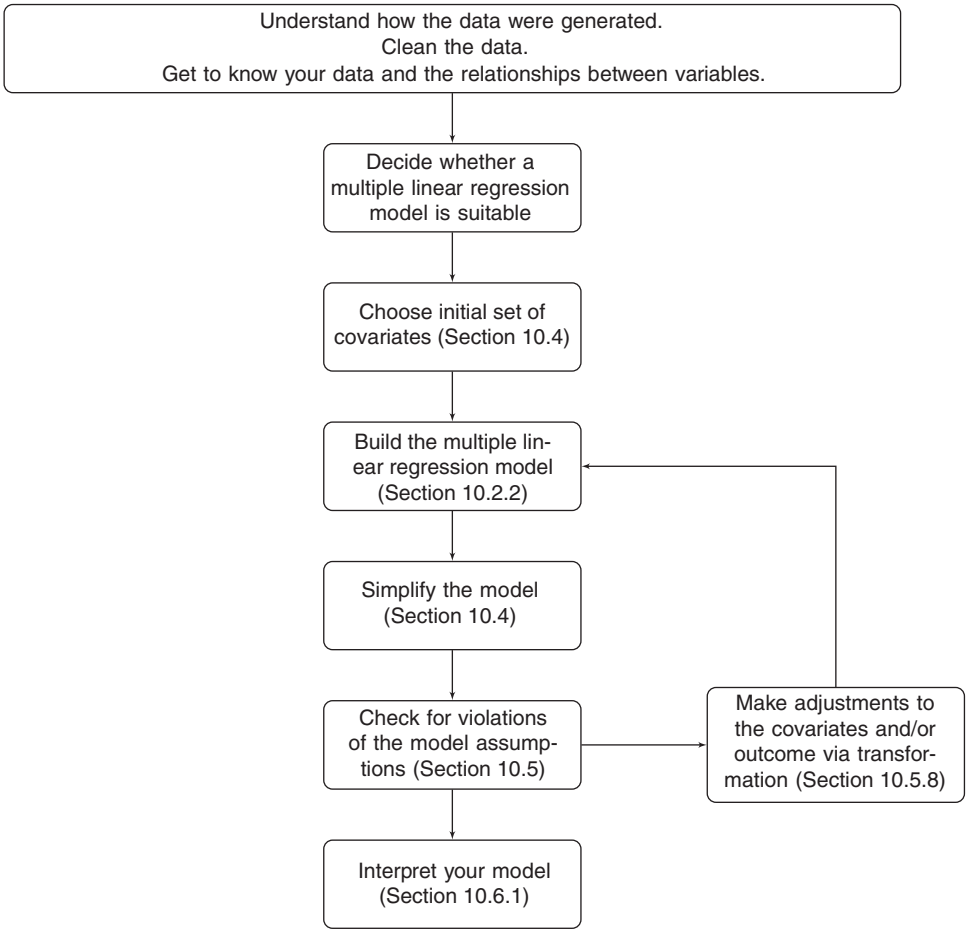


Figure 10.6 Building a multiple linear regression model.

art than a science. Figure 10.6 is a modified version of the suggested process for building simple linear regression models, as seen in Figure 10.4, and suggests a procedure for handling the task of modeling and which sections of this book deals with which parts.

Deciding whether a multiple linear model is suitable in the first place will come from knowledge of the context of the data and how it was generated, for example in ensuring that observations are not obviously dependent on one another which would immediately violate the assumption of independent errors. In particular, no longer can we identify problems with linearity before building the model, as we could for the simple linear regression model, and we have the additional hurdle of deciding how we deal with multiple covariates. These issues are discussed at various points in this chapter.

For now, we'll focus on how to run a multiple linear regression model in *R*, and do so using the air pollution example. We'll build a multiple linear regression model with ozone concentration as the outcome and wind speed, air temperature, and the intensity of solar radiation as covariates.

Running this model in *R* follows the same pattern as building a simple linear regression model: we use the `lm ()` command, and feed the covariates to this function, separating them using `+`.

```
attach (ozone_pollution)
ozone_mod1 <- lm (ozone ~ rad + temp + wind)
summary (ozone_mod1)


Call:
lm(formula = ozone ~ rad + temp + wind)

Residuals:
    Min      1Q  Median      3Q     Max
-40.485 -14.210  -3.556  10.124  95.600

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -64.23208   23.04204  -2.788  0.00628 **
rad           0.05980    0.02318   2.580  0.01124 *
temp          1.65121    0.25341   6.516 2.43e-09 ***
wind         -3.33760    0.65384  -5.105 1.45e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 21.17 on 107 degrees of freedom
Multiple R-squared:  0.6062,Adjusted R-squared:  0.5952
F-statistic: 54.91 on 3 and 107 DF,  p-value: < 2.2e-16

detach (ozone_pollution)
```

If, for some reason, we do not want an intercept in your model, we can suppress it in one of two ways (the `-1` or `+0` can go anywhere to the right of ~):

```
lm (ozone ~ rad + temp + wind - 1)
lm (ozone ~ rad + temp + wind + 0)
```

This forces the regression line through the origin, (0,0, 0,0), implying that if `rad=0`, `temp=0`, and `wind=0`, then we must have `ozone=0`. There is no reason to enforce this here, but this might be a sensible strategy in some cases. Consider, for example predicting the number of calories in 100 g of a product, with fat, carbohydrate, and protein content as covariates. It *might* be desirable to build a model that guarantees a zero calorie prediction if a food product has no fat, carbohydrate or protein; in this case, we could consider a model without intercept. Such cases are rare, and it is advisable to include an intercept unless there are exceptional reasons not to.

10.2.3 Categorical covariates

What if one or more of your covariates is categorical? Can you still use linear regression models? The good news is that yes, you can.

Categorical variables cannot be directly included as covariates in a linear model because their values are names (or labels) and not numbers: we need to convert them into a format that *R* can work with. Essentially, this means turning a categorical variable into a sequence of binary variables,

also known as dummy variables. These variables, as the name suggests, can only take one of two values: 0 or 1.

When you pass a categorical variable to *R* as a covariate, or specify that an apparently numeric variable should be treated as categorical (sometimes categorical variables are coded using numbers, but these numbers are just labels), *R* will *automatically create these binary variables so we don't have to*.

Let us take a simple example: we have an experiment in which crop yields per unit area were measured from 10 randomly selected fields on each of three soil types. All fields were sown with the same variety of seed and provided with the same fertilizer and pest control inputs. The question is whether soil type significantly affects crop yield, and if so, to what extent.

```
yields <- read.table ("yields.txt", header = T)
yields

   sand clay loam
1     6   17   13
2    10   15   16
3     8    3    9
4     6   11   12
5    14   14   15
6    17   12   16
7     9   12   17
8    11    8   13
9     7   10   18
10   11   13   14
```

At the moment, these data are not in the correct format to run a linear regression: these data are currently in **wide** format, whereas we want them in **long** format. The long format of these data will involve generating one column with the soil types (this will be our covariate) and another for the yield (our outcome variable). This can easily be achieved by using the `stack ()` function. This would give a long list in the output here, so we limit the displayed results to the first few observations by using the function `head ()`.

```
yields_long <- stack (yields)
head (yields_long)

  values  ind
1      6 sand
2     10 sand
3      8 sand
4      6 sand
5     14 sand
6     17 sand
```

We see that the `stack ()` function has invented names for the outcome variable `values` and the covariate `ind`. We will most likely want to change these:

```
names (yields_long) <- c ("yield", "soil")
head (yields_long)
```

```
  yield soil
1     6 sand
2    10 sand
3     8 sand
4     6 sand
5    14 sand
6    17 sand
```

Now that our data are in the correct format, we need to consider how to convert the categorical variable `soil` into something that *R* can handle. Notice that there are three categories in total, so we can do this using just two binary variables. Let us call these x_{i1} and x_{i2} for now and define them as follows:

$$x_{i1} = \begin{cases} 1 & \text{if clay} \\ 0 & \text{otherwise} \end{cases} \qquad x_{i2} = \begin{cases} 1 & \text{if loam} \\ 0 & \text{otherwise} \end{cases}$$

Note that x_{i1} and x_{i2} are each dummy variables, and:

- Soil is clay if $x_{i1} = 1$ (in which case $x_{i2} = 0$);
- Soil is loam if $x_{i2} = 1$ (in which case $x_{i1} = 0$);
- Soil is sand if $x_{i1} = 0$ and $x_{i2} = 0$.

The important thing to notice is that *two* dummy variables are enough to code a categorical variable with *three* categories, and in general $(q - 1)$ dummy variables are enough to code a categorical variable with q categories. The subsequent model can be written in the form

$$Y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \epsilon_i. \tag{10.4}$$

Notice what happens in (10.4):

- If the soil is clay: $Y_i = \beta_0 + \beta_1 + \epsilon_i$ (because $x_{i1} = 1; x_{i2} = 0$);
- If the soil is loam: $Y_i = \beta_0 + \beta_2 + \epsilon_i$ (because $x_{i1} = 0; x_{i2} = 1$);
- If the soil is sand: $Y_i = \beta_0 + \epsilon_i$ (because $x_{i1} = 0; x_{i2} = 0$).

The mean yield for sand type soil is described by the intercept β_0. If we consider clay or loam, then the additional β_1 or β_2 describes the mean yield difference between sand and each other soil type. Notice, however, that what we have here are three intercept-only models: the intercept is permitted to change by category. More generally, when we have at least one categorical variable as a covariate, the intercept can change for observations from different categories (if there are differences in outcome between the categories).

In our example here, `sand` is considered the **reference category**, but is an arbitrary choice: any of the categories will do. It is best, however, to avoid using a category with very few observations as your reference as this can cause some instability in your model.

When we run a multiple regression model for the yield data, we see that *R* automatically creates the dummy variables required. It has chosen `sand` as the reference category in this case, but of course the choice is rather arbitrary.

```
yields_model <- lm (yield ~ soil, data = yields_long)
summary (yields_model)


Call:
lm(formula = yield ~ soil, data = yields_long)

Residuals:
   Min     1Q Median     3Q    Max
  -8.5   -1.8    0.3    1.7    7.1

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)    9.900      1.081   9.158 9.04e-10 ***
soilclay       1.600      1.529   1.047  0.30456
soilloam       4.400      1.529   2.878  0.00773 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.418 on 27 degrees of freedom
Multiple R-squared:  0.2392,Adjusted R-squared:  0.1829
F-statistic: 4.245 on 2 and 27 DF,  p-value: 0.02495
```

We can see that the mean yield for sand soil is 9.9, while the difference between yields for clay and sand soils is 1.6, for example.

In some cases, it doesn't matter which category is chosen as the reference as is probably the case here (unless you specifically want to compare the performance of, say, loam to each of clay and sand). An obvious example of where it *does* matter is in comparing treatments: supposing you want to compare two new treatments against the current gold standard. It would make sense for the current gold standard to be treated as the reference category, so that comparisons can be made between it and each of the new treatments.

Changing the reference category can be done by reordering the categories using `factor ()`. To change the reference category from `sand` to `loam`, for example

```
yields_long$soil <- factor (yields_long$soil, levels = c ("loam", "clay", "sand"))
yields_model2 <- lm (yield ~ soil, data = yields_long)
summary (yields_model2)


Call:
lm(formula = yield ~ soil, data = yields_long)

Residuals:
   Min     1Q Median     3Q    Max
  -8.5   -1.8    0.3    1.7    7.1

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
```

```
(Intercept)   14.300      1.081  13.229 2.58e-13 ***
soilclay      -2.800      1.529  -1.832  0.07807.
soilsand      -4.400      1.529  -2.878  0.00773 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.418 on 27 degrees of freedom
Multiple R-squared:  0.2392,    Adjusted R-squared:  0.1829
F-statistic: 4.245 on 2 and 27 DF,  p-value: 0.02495
```

You can also use other methods to achieve the same, see for example Section 16.3 for an example.

Finally, two questions that you might be asking at this point are 'what would happen if we used *three* dummy variables to describe soil type?', and 'what would happen if we decided not to have an intercept in our model?'.

In answer to the first question, you would run into trouble: you would have four **regression coefficients** to estimate (the intercept and three 'slopes'), but only three pieces of information with which to do this (the yield information from each soil type). This is tantamount to trying to solve a set of three simultaneous equations involving four unknowns, which won't have a unique solution.

The second question is more interesting. If you don't have an intercept, then two dummy variables in this case won't distinguish between the three categories (otherwise, loam in the above example will be assumed to have $y_i = \epsilon_i$, implying a mean yield of zero). In this special case, we therefore need to use three dummy variables to code for our three categories. Remember that going ahead without an intercept is unusual and needs robust justification.

While *R* is perfectly happy to create dummy variables for categorical variables, it is sometimes helpful to create these explicitly. Creating tables of dummy variables for use in statistical modelling is extremely easy with the `model.matrix ()` function. You will see what the function does with a simple example.

In our modelling, we want to create a two-level dummy variable (present or absent) for each soil type (in three extra columns), so that we can ask questions such as whether the mean value of `yield` is significantly different in cases where each soil type was present, and when it was absent. So for the first row of the dataframe, we want `sand = TRUE`, and the rest as `FALSE`.

The long-winded way of doing this is to create a new factor for each soil type separately, but it is easy to do with `model.matrix ()`. The `-1` in the model formula ensures that we create a dummy variable for each of the three soil types (technically, it suppresses the creation of an intercept).

```
dummy_matrix <- model.matrix (~ yields_long$soil - 1)
head (dummy_matrix)

  yields_long$soilloam yields_long$soilclay yields_long$soilsand
1                    0                    0                    1
2                    0                    0                    1
3                    0                    0                    1
4                    0                    0                    1
5                    0                    0                    1
6                    0                    0                    1
```

This matrix has 30 rows, one for each observation in the original dataset. Now, we can join these three columns of dummy variables to the original dataframe, `yields_long`. We just join the new columns to it, after which we can use variable names like `yields_long.soilloam` in the statistical modelling (we might want to update the names to something snappier!):

```
new_frame <- data.frame (yields_long, model.matrix (~ yields_long$soil - 1))
head (new_frame)

  yield soil yields_long.soilloam yields_long.soilclay yields_long.soilsand
1     6 sand                    0                    0                    1
2    10 sand                    0                    0                    1
3     8 sand                    0                    0                    1
4     6 sand                    0                    0                    1
5    14 sand                    0                    0                    1
6    17 sand                    0                    0                    1
```

10.2.4 Interactions between covariates

In many circumstances, we would like to be able to describe the effect of an **interaction** between two (or more) covariates on the response. Interactions are interesting when dependence of the outcome on a specific covariate changes with the value of another covariate. This may sound a little abstract right now, but the next example should put that right.

The following experiment, with weight as the response variable, involved genotype, sex, and age as covariates. There are six levels of genotype and two levels of sex. We initially build a model for weight based on genotype, sex, and age as covariates.

```
gain <- read.table ("Gain.txt", header = T)
attach (gain)
gain_mod1 <- lm (Weight ~ Sex + Age + Genotype)
summary (gain_mod1)


Call:
lm(formula = Weight ~ Sex + Age + Genotype)

Residuals:
     Min       1Q   Median       3Q      Max
-0.40005 -0.15120 -0.01668  0.16953  0.49227

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)    7.93701    0.10066  78.851  < 2e-16 ***
Sexmale       -0.83161    0.05937 -14.008  < 2e-16 ***
Age            0.29958    0.02099  14.273  < 2e-16 ***
GenotypeCloneB 0.96778    0.10282   9.412 8.07e-13 ***
GenotypeCloneC -1.04361    0.10282 -10.149 6.21e-14 ***
GenotypeCloneD 0.82396    0.10282   8.013 1.21e-10 ***
GenotypeCloneE -0.87540    0.10282  -8.514 1.98e-11 ***
```

```
GenotypeCloneF  1.53460    0.10282  14.925  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.2299 on 52 degrees of freedom
Multiple R-squared:  0.9651,Adjusted R-squared:  0.9604
F-statistic: 205.7 on 7 and 52 DF,  p-value: < 2.2e-16
```

We could ask questions such as:

1. Does the effect of `Age` on `Weight` depend on `Sex`?
   - That is, should the coefficient of `Age` change depending on the given `Sex`?
2. Does the effect of `Age` on `Weight` depend on `Genotype`?
   - That is, should the coefficient of `Age` change depending on the given `Genotype`?
3. Does the effect of `Genotype` on `Weight` depend on `Sex`?
   - That is, should the coefficient of `Genotype` change depending on the given `Sex`?

And so on.

These are questions about interactions: how two (or more) covariates interact with each other in their effect on the outcome. Taking into account interactions in a regression model is generally a simple task: just multiply the two (or more) covariates which you think may interact with each other to form a set of new covariates. Of course, we don't need to literally create these covariates: we can simply tell *R* to include an interaction between specified covariates in the `lm ()` function by using `*` instead of `+`.

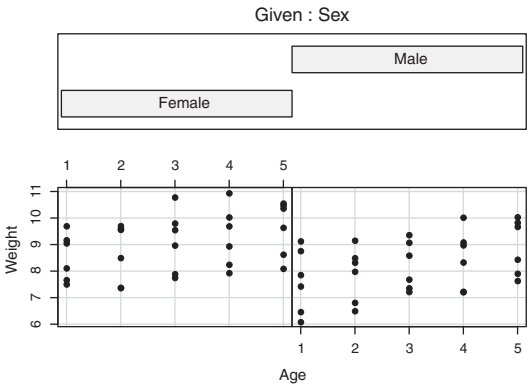
An important point to note is that interactions can be between any type of variable, but the number of interaction terms it produces as extra 'covariates' in our model will depend on the nature of the variables used. Interactions can be between:

- two categorical covariates, which will produce $(p-1)\times(q-1)$ additional covariates (where p and q are the number of levels for each covariate);
- one categorical and one numeric covariate, which will produce $(p-1)$ additional covariates (where p is the number of levels for the categorical covariate);
- two numeric covariates, which will produce one additional covariate.

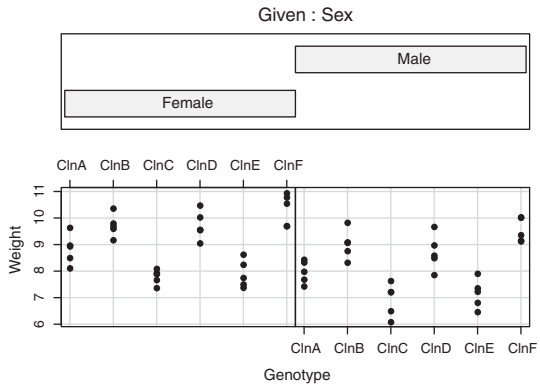
As adding interactions is straightforward, it might be tempting to include interactions between all variables. This is rarely wise. As a general rule of thumb, if context-specific information leads you to suspect an interaction may be at play, then by all means try adding an interaction to the model.

This being said, it might be helpful to assess potential interactions *before* including them in a model. Interaction plots provide a visual representation of how two covariates interact (if at all). The relevant *R* functions are `interaction.plot ()` for interactions between two categorical covariates, and `coplot ()` which is a more general function that can deal with any type of covariates.

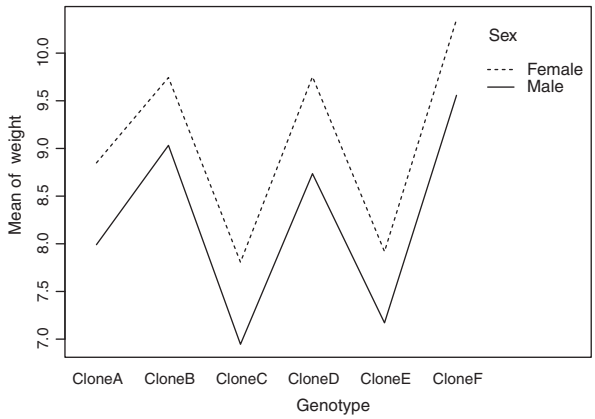
Let us go back to our experimental data, and suppose we have a reason to believe that an interaction between age and sex, and another between genotype and sex, may help us in explaining the outcome. The first is an interaction between a numeric and categorical variable, while the second



(a) Coplot between age and sex



(b) Coplot between genotype and sex



(c) Interaction plot between genotype and sex

Figure 10.7 Plots to check for interaction.

involves two categorical variables. Figures 10.7a and 10.7b show the `coplot ()` output for both situations, while Figure 10.7c shows the equivalent `interaction.plot ()` function in action for the latter: Figures 10.7b and 10.7c show the same information in slightly different formats.

```
coplot (Weight ~ Age | Sex, data = gain)
coplot (Weight ~ Genotype | Sex, data = gain)
interaction.plot (Genotype, Sex, Weight)
```

What we are looking for here is evidence that males and females differ in terms of gradients/slopes or patterns. There is no obvious difference between males and females by genotype (Figures 10.7b and 10.7c): though females are generally heavier than males, the pattern of change over genotype is very similar. Notice that the difference in weight between females and males will be addressed by the covariate `Sexmale` on its own; this doesn't suggest that we need an interaction.

Meanwhile, a similar story is evident for age and sex: the gradient of a regression line fitted to each window in Figure 10.7a wouldn't differ particularly between men and women so again there is no evidence of an interaction between these two covariates.

No interaction effect could be seen using plots, and therefore unless these effects were of particular scientific interest we wouldn't include them in our model. Supposing, however, the interaction between age and sex *was* of interest, then we can incorporate this using:

```
gain_mod2 <- lm (Weight ~ Sex * Age + Genotype)
summary (gain_mod2)

Call:
lm(formula = Weight ~ Sex * Age + Genotype)

Residuals:
     Min       1Q   Median       3Q      Max
-0.37202 -0.15893 -0.00302  0.15263  0.45188

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)    7.99759    0.11882  67.310  < 2e-16 ***
Sexmale       -0.95277    0.13933  -6.838 9.80e-09 ***
Age            0.27939    0.02970   9.406 9.98e-13 ***
GenotypeCloneB 0.96778    0.10290   9.405 1.00e-12 ***
GenotypeCloneC -1.04361   0.10290 -10.142 7.96e-14 ***
GenotypeCloneD 0.82396    0.10290   8.008 1.41e-10 ***
GenotypeCloneE -0.87540   0.10290  -8.507 2.36e-11 ***
GenotypeCloneF 1.53460    0.10290  14.914  < 2e-16 ***
Sexmale:Age    0.04039    0.04201   0.961    0.341
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.2301 on 51 degrees of freedom
Multiple R-squared:  0.9658,    Adjusted R-squared:  0.9604
F-statistic: 179.8 on 8 and 51 DF,  p-value: < 2.2e-16

detach (gain)
```

Box 10.4: Main effects and interaction effects

The term 'interaction effect' is exactly as it sounds: here our only interaction effect was the variable `Sexmale:Age` in the output. The main effects are the covariates listed in our model that are not interaction effects. This includes `Sexmale` and `Age` along with `Genotype`.

Here we have an interaction effect between a categorical and numeric covariate, so given that the categorical variable has two levels we would expect just one additional row to appear in the output relating to the interaction between age and sex. This is indeed the case: the (single) interaction effect is in the last row of the main table in the output. But how does this allocate a different coefficient for `Age` depending on `Sex`?

For a male (so that `Sexmale` is 1), supposing that their genotype is clone D for simplicity (so that `GenotypeCloneD` is 1, but all others are 0), we have

$$\hat{y} = 8 - 0.95 + 0.28 \times \text{age} + 0.82 + 0.04 \times \text{age} = 7.87 + 0.32 \times \text{age}$$

For a female, with the same Genetic type as above, we have:

$$\hat{y} = 8 + 0.28 \times \text{age} + 0.82 = 8.82 + 0.28 \times \text{age}$$

You should see that the regression coefficient for age has changed depending on sex. The different intercept is due to the `Sexmale` **main effect** and *not* due to the interaction.

Interactions of three or more variables are possible by multiplying the relevant covariates. For example, to fit an interaction between all three covariates here:

```
lm (Weight ~ Age * Genotype * Sex, data = gain)
```

The interpretation of a fitted model with such interactions is cumbersome. More on this in Section 10.6.1.

10.3 Understanding the output

We'll use the model that we built in the last section – a model for ozone concentration using wind speed, air temperature, and the intensity of solar radiation as covariates – to understand the output from the function `lm ()`. The output is displayed here again for convenience.

```
summary (ozone_mod1)

Call:
lm(formula = ozone ~ rad + temp + wind)

Residuals:
    Min      1Q  Median      3Q     Max
-40.485 -14.210  -3.556  10.124  95.600

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -64.23208   23.04204  -2.788  0.00628 **
rad           0.05980    0.02318   2.580  0.01124 *
temp          1.65121    0.25341   6.516 2.43e-09 ***
wind         -3.33760    0.65384  -5.105 1.45e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 21.17 on 107 degrees of freedom
Multiple R-squared:  0.6062,    Adjusted R-squared:  0.5952
F-statistic: 54.91 on 3 and 107 DF,  p-value: < 2.2e-16
```

10.3.1 Residuals

A residual is the difference between an observed outcome for a specific value of the covariate(s), and the outcome predicted by the model using the same value(s) of the covariate(s). The five number summary for the residuals is given. On its own, this is probably not the most helpful of information, but we will be using the residuals later in Section 10.4: these will turn out to be the key to assessing how well a model fits the data.

10.3.2 Estimates of coefficients

The `Estimate` column in the `Coefficients:` table of the output gives us the (least squares) estimates of the regression coefficients. The intercept is listed as `(Intercept)` – in this case it is estimated to be around -64.23. Estimates of the regression coefficients of the covariates are also given. In our case, the predicted model is

$$\hat{y} = -64.23 + 0.06 \times \text{rad} + 1.65 \times \text{temp} + -3.34 \times \text{wind}.$$

10.3.3 Testing individual coefficients

The remaining columns in the `Coefficients:` table of the output are

- `Std. Error`: The standard error of the estimated regression coefficient;
- `t value`: The t-statistic for the null hypothesis that the regression coefficient is zero. This is computed as the ratio of the estimated regression coefficient and its standard error;
- `Pr(>|t|)`: The *p*-value resulting from the t-test above.

The hypothesis being tested here is particularly interesting. Suppose we look at the case of the *j*'th regression coefficient, β_j:

$$H_0 : \beta_j = 0 \text{ given the other covariates in the model.}$$
$$H_1 : \beta_j \neq 0 \text{ given the other covariates in the model.}$$

This is an important test: if indeed the null hypothesis is true, then this is equivalent to leaving the *j*th covariate out of the model. This implies that the covariate doesn't help in explaining the variability in the outcome. The alternative hypothesis, of course, is that it remains in the model: that this particular covariate *does* help in explaining the variability in the outcome. The underlying test is a t-test: the test statistic as given in the `t value` column is compared to a t-distribution with $(n - p)$ degrees of freedom, where n is the number of observations and p is the number of regression coefficients to estimate (including the intercept). See, for example Section 9.1 for an example of a t-test.

Box 10.5: Testing the intercept

Even if we don't reject the hypothesis that the intercept is zero, it is rarely wise to remove it.

These hypothesis tests can be used to check if a *single* regression coefficient is nonzero. We shouldn't be eliminating multiple covariates simultaneously on the basis that their *p*-values are all 'too large': this is not what's being tested here. It can happen that two or more of the *p*-values for the regression coefficients are large so may lead us to think that we could remove these covariates simultaneously. However, as we'll see later, an appropriate statistical test to look at whether we can remove two or more covariates simultaneously may find that it is best not to. Remember that the hypothesis is conditional on all other variables being present, so while there may be no evidence to retain covariate *A* in the model when *B* is in there, and no evidence to retain covariate *B* in the model when *A* is in there, this is not evidence that we can remove *both A* and *B*.

The stars (if any) next to the *p*-values give a helpful at-a-glance view of how small the *p*-value is for each of these t-tests. As noted in the `Signif. codes` legend beneath the table of coefficients, three stars denotes a *p*-value of less than 0.001, two stars a *p*-value of between 0.001 and 0.01, and so on. These cut-offs for levels of *p*-values are arbitrary and should not be taken too seriously.

10.3.4 Residual standard error

The residual standard error is listed beneath the main table in the output. It is the sample estimate of the standard deviation, σ, of the error term, ϵ, in (10.3). For the ozone dataset in hand, the estimate is 21.17, and therefore the estimated variance of the error term is its square, 448.26.

10.3.5 R^2 and its variants

One useful measure we can extract from a model is the coefficient of determination, commonly known as R^2 ('R squared'). This is a sliding scale between 0 and 1 measuring the *proportion of variability in the outcome that is explained by the covariates in the model*. An R^2 of zero indicates that there is no linear relationship between the outcome and covariates (for example in Figure 10.8a), while an R^2 of 1 indicates that (a linear combination of) the covariate(s) can perfectly predict the outcome (for example in Figure 10.8b).

The R^2 can be used as a very rough guide to assessing the strength of the linear relationship between the response variable and the covariates in the model. It should not be used to decide on which model to use, or to compare between models, and should be viewed as just one tool in a modeller's toolkit.

A very important point to note is that a low R^2 isn't necessarily an indication of a poor-fitting model. See, for example Figure 10.8c where a simple linear regression model isn't a bad choice, but due to the amount of variability in the outcome, the R^2 is low. The converse is also true: a high R^2 doesn't necessarily mean you have a 'good' model. In Figure 10.8d, the R^2 is high but a simple linear regression model is clearly the wrong model.

In the *R* output, there are two versions of R^2: the `Multiple R-squared` and the `Adjusted R-squared`. The R^2 as described here is the `Multiple R-squared`. The `Adjusted R-squared` attempts to get around an undesirable property of R^2: adding covariates into a model will never decrease the R^2, and indeed will very likely increase it. This is undesirable since we could artificially inflate our R^2 by simply increasing the number of covariates we use in the model. Indeed, if we have n observations in our dataset, and we use $(n-1)$ covariates in our model, then we will ensure that $R^2 = 1$: perfect prediction of the outcome from the covariates! To see why this is the case, consider the case that we have two observations and only one covariate. We can guarantee that a simple linear regression model can be found that fits exactly through these two points, yielding an R^2 of 1. This argument extends to however many observations you happen to have. The adjusted R^2 includes a penalty to prevent this happening: the more covariates you add to your model, the bigger the penalty. This penalty is arbitrary, and there's nothing wrong with using `Multiple R-squared` without the penalty, subject to you bearing in mind this property.

10.3.6 The regression *F*-test

The final line of the `lm ()` output is an F-test, which is sometimes known as an Analysis of Variance for regression (or ANOVA for regression). This is a rather severe statistical test, but serves as a blunt tool to assess your model. Suppose that the model you are building has an intercept, β_0, and

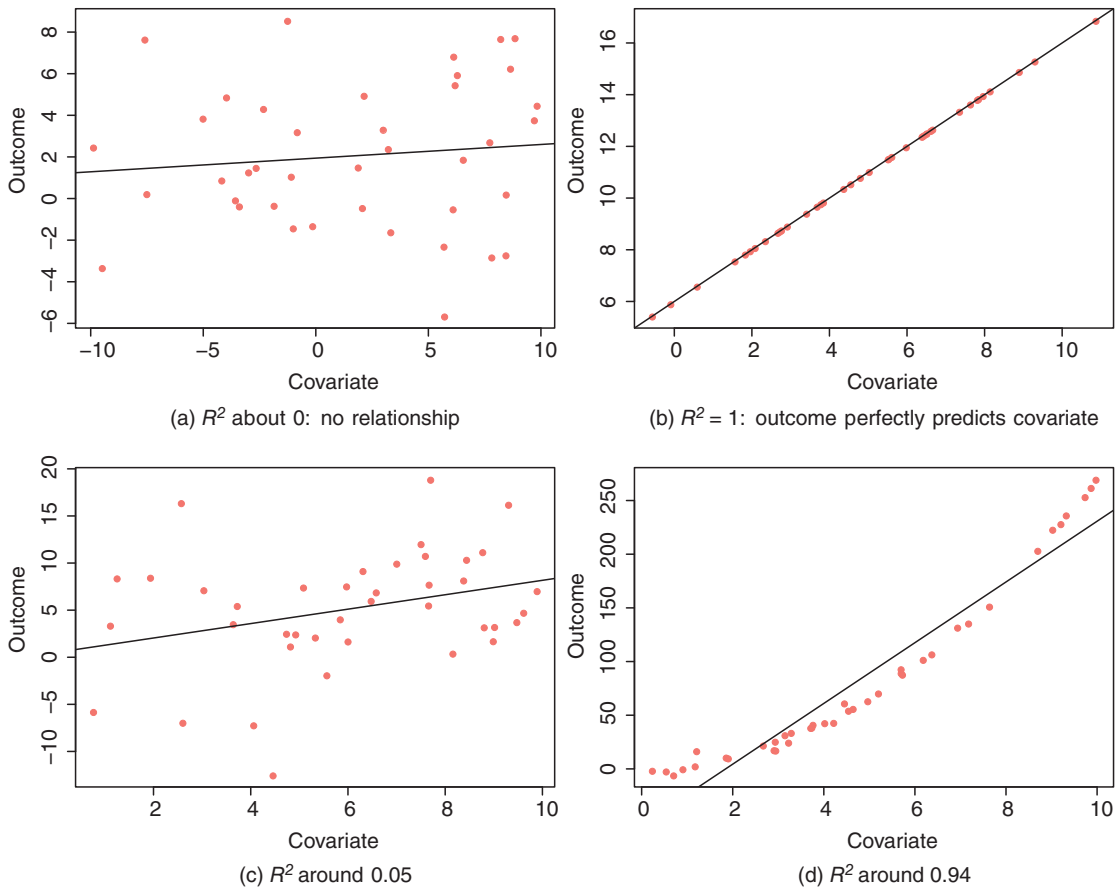


Figure 10.8 R^2 of various relationships.

further regression coefficients $\beta_1, \ldots, \beta_p$ for p covariates. The hypothesis in question here is

$$H_0 : \beta_1 = \ldots = \beta_p = 0 ;$$

$$H_1 : \text{At least one of } \beta_1, \ldots, \beta_p \text{ is non-zero.}$$

The null hypothesis can be rephrased as 'remove ALL covariates from the model'. This seems rather extreme, but it tells us something about whether the covariates have any power in describing the outcome. If we fail to reject this hypothesis, then we probably don't have a very powerful model on our hands.

10.3.7 ANOVA: Same model, different output

It is worth spending some time discussing the ANOVA for regression a little more. In Section 10.3.6, we discussed the ANOVA for regression which boils down to an F-test comparing the model in question with the intercept-only model. We'll also be using this function later in the chapter to compare models. There is a lot more going on behind the scenes here, which we'll now consider.

Let us start by applying the `anova ()` function to our model. As you can see, the output looks very different to the `lm ()` output, which is also given for reference. Can you see some similarity, however?

```
summary (ozone_mod1)


Call:
lm(formula = ozone ~ rad + temp + wind)

Residuals:
    Min      1Q  Median      3Q     Max
-40.485 -14.210  -3.556  10.124  95.600

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -64.23208   23.04204  -2.788  0.00628 **
rad           0.05980    0.02318   2.580  0.01124 *
temp          1.65121    0.25341   6.516 2.43e-09 ***
wind         -3.33760    0.65384  -5.105 1.45e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 21.17 on 107 degrees of freedom
Multiple R-squared:  0.6062,Adjusted R-squared:  0.5952
F-statistic: 54.91 on 3 and 107 DF,  p-value: < 2.2e-16

anova (ozone_mod1)

Analysis of Variance Table

Response: ozone
          Df Sum Sq Mean Sq F value    Pr(>F)
rad        1  14780   14780  32.971 8.853e-08 ***
temp       1  47378   47378 105.692 < 2.2e-16 ***
wind       1  11680   11680  26.057 1.450e-06 ***
Residuals 107  47964     448
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The `anova ()` output contains the building blocks of the F-test we saw in Section 10.3.6, but we won't worry about that here. What you should look at are the *p*-values. Notice that the last *p*-value – that for `wind` – is the same for both `lm ()` and `anova ()`. Let us do something seemingly innocent and switch around the *order* of the covariates in the model:

```
attach (ozone_pollution)
ozone_mod1a <-lm (ozone ~ wind + rad + temp)
summary (ozone_mod1a)
```

```
Call:
lm(formula = ozone ~ wind + rad + temp)

Residuals:
    Min      1Q  Median      3Q     Max
-40.485 -14.210  -3.556  10.124  95.600

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -64.23208   23.04204  -2.788  0.00628 **
wind         -3.33760    0.65384  -5.105 1.45e-06 ***
rad           0.05980    0.02318   2.580  0.01124 *
temp          1.65121    0.25341   6.516 2.43e-09 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 21.17 on 107 degrees of freedom
Multiple R-squared:  0.6062,Adjusted R-squared:  0.5952
F-statistic: 54.91 on 3 and 107 DF,  p-value: < 2.2e-16

anova (ozone_mod1a)

Analysis of Variance Table

Response: ozone
           Df Sum Sq Mean Sq F value    Pr(>F)
wind        1  45762   45762 102.088 < 2.2e-16 ***
rad         1   9044    9044  20.176 1.792e-05 ***
temp        1  19032   19032  42.457 2.429e-09 ***
Residuals 107  47964     448
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

detach (ozone_pollution)
```

Other than switching rows around in the `lm ()` output, it's identical to the previous version. However, the `anova ()` output is different. Notice again that the last *p*-value is the same for both `lm ()` and `anova ()`, but this time this relates to `temp`.

So what is the ANOVA output telling us? Going back to our `ozone_mod1` model, and applying the `anova ()` function:

- the first row of the output compares the model with `rad` to the intercept-only model (no covariates) via an F-test;
- the second row of the output compares the model with `rad, temp` to the model with only `rad` via an F-test;
- the third row of the output compares the model with `rad, temp, wind` to the model with `rad, temp` via an F-test.

By the time we reach the row corresponding to the last-entered covariate in the ANOVA table, we are comparing the 'full' model with removing just one covariate, in this case `wind`. This is identical to the t-test for `wind` that appears in the `lm ()` output, because the F-test in this special case is mathematically identical to a t-test which we saw in Section 10.3.3 (see Section 10.4.3 for full details why the F- and t-test are identical here). The other *p*-values in the ANOVA table are *not* equivalent to the t-tests in the `lm ()` output, so the *p*-values are different in this case.

10.3.8 Extracting model information

We often want to extract material from fitted models (e.g. slopes, residuals, or *p* values), and there are three different ways of doing this:

- extracting from the model object;
- extracting from the summary of the model;
- directly by name, e.g. `coef (model)`.

Let us take a look at what we can extract with each of the methods above from `ozone_mod1`. First, we'll look what was saved inside the model object itself.

```
names (ozone_mod1)

 [1] "coefficients"  "residuals"     "effects"       "rank"
 [5] "fitted.values" "assign"        "qr"            "df.residual"
 [9] "xlevels"       "call"          "terms"         "model"
```

There's a whole host of information here, which we can extract using the name or number attached to each element. For example, if we want to extract just the coefficients, we could do this via one of two ways:

```
ozone_mod1$coefficients

 (Intercept)          rad         temp         wind
-64.23208116   0.05979717   1.65120780  -3.33759763

ozone_mod1[1]

$coefficients
 (Intercept)          rad         temp         wind
-64.23208116   0.05979717   1.65120780  -3.33759763
```

Remembering which numbers go with which bits of the output is probably harder than using the names directly so we'll stick to names.

We may want to extract a *subset* of the coefficients:

```
ozone_mod1$coefficients[[2]]

[1] 0.05979717
```

or save all coefficients as a vector (useful if you're running many analyses and want to keep track of the estimated coefficients, for example):

```
coef_ozone_mod1 <- as.vector (ozone_mod1$coefficients)
coef_ozone_mod1

[1] -64.23208116   0.05979717   1.65120780  -3.33759763
```

A slightly different set of information is contained in the summary of the model:

```
names (summary (ozone_mod1))

 [1] "call"          "terms"         "residuals"     "coefficients"
 [5] "aliased"       "sigma"         "df"            "r.squared"
 [9] "adj.r.squared" "fstatistic"    "cov.unscaled"
```

Again, we can access these bits of information by name or by number. For example, if I wanted to extract the residual standard error:

```
summary (ozone_mod1)$sigma

[1] 21.17222
```

or save the residuals as a newly defined object:

```
resid_ozone_mod1 <- summary (ozone_mod1)$residuals
head (resid_ozone_mod1)

        1           2           3           4           5           6
7.9379186   0.9898344 -12.8133444  -0.4769439  -9.2724401  25.9497483
```

Finally, some information can be extracted using a particular function applied to the model object. These include the coefficients, the residual standard error, the residuals, and the fitted values. You get the idea by now, so we won't run these functions:

```
coef (ozone_mod1)
sigma (ozone_mod1)
residuals (ozone_mod1)
fitted (ozone_mod1)
```

10.4 Fitting models

10.4.1 The principle of parsimony

One of the most important themes running through this book concerns model simplification. The principle of parsimony is attributed to the early fourteenth-century English nominalist philosopher,

William of Occam, who insisted that, given a set of equally good explanations for a given phenomenon, *the correct explanation is the simplest explanation*. It is called Occam's razor because he 'shaved' his explanations down to the bare minimum: his point was that in explaining something, assumptions must not be needlessly multiplied. For multiple linear regression models, the principle of parsimony means that:

- models should have as few parameters as possible;
- a model without interactions is preferred to a model containing interactions between factors, all else being equal;
- the levels of a categorical covariate may need to be grouped if they do not differ significantly from each other in terms of the outcome.

In our zeal for model simplification, however, we must be careful not to throw the baby out with the bathwater. Einstein made a characteristically subtle modification to Occam's razor. He said: 'A model should be as simple as possible. But no simpler.' Thus, we should only include an explanatory variable in a model if it significantly improves the fit of the model; the fact that we went to the trouble of measuring something does not mean we have to include it in our model. So, models should have as few covariates as possible, while still explaining as much of the variability in the outcome as is realistically possible.

In practice, the principle of parsimony means that simplifying our models becomes an important task for any modeller. So, given all this, *how* should we go about simplifying a model? There are no fixed rules and no absolutes: how we achieve this is essentially a process of exploration. The thing to remember about multiple regression is that, in principle, there is no end to it. The number of combinations of interaction terms (and curvature terms, which are discussed later in Section 10.5.8) is endless, but a model resulting from any simplifying process:

- must not lead to significant reductions in the model's power to explain the outcome;
- must make good scientific sense;
- should not be more difficult to interpret than is necessary.

The second and third items on this list mainly are mainly concerned with understanding the context of the data and thoughtfully choosing which covariates to retain in the model. For a model to be appropriate in this sense, we may need to incorporate covariates that do not help us particularly in explaining the outcome. This includes, but is by no means limited to, dealing with interactions: if a model includes an interaction term, it is highly recommended that the terms for the individual covariates are kept in the model even if these main effects don't have much of an impact in helping to predict the outcome. Without these however, interpretation gets very tricky indeed (and what use is a model that is either very hard, or not possible, to interpret?). See Section 10.6.1 for further details.

To make matters more complicated, an important point to realise is that it is likely that at least some of the covariates are correlated with each other. Some correlation is expected and won't cause a problem (see Section 10.5.7 for potential problems when correlation is too strong). However, correlation between covariates means that deleting some of these from the model will *change how the remaining covariates influence the outcome*.

For example in our ozone pollution data, the covariates `wind` and `temp` are correlated with one another as we saw in Figure 10.5. If we remove, say, `temp` from the model, the regression coefficient

of `wind` changes considerably, as does the intercept and to a lesser extent the regression coefficient of `rad`. The standard errors and *p*-values of the remaining covariates also change, though these are not requested in the output here. This is nothing to worry about per se, but it is something that you should bear in mind when modelling.

```
attach (ozone_pollution)
lm (ozone ~ rad + temp + wind)


Call:
lm(formula = ozone ~ rad + temp + wind)

Coefficients:
(Intercept)          rad         temp         wind
   -64.2321       0.0598       1.6512      -3.3376

lm (ozone ~ rad + wind)


Call:
lm(formula = ozone ~ rad + wind)

Coefficients:
(Intercept)          rad         wind
    77.2687       0.1003      -5.4035

detach (ozone_pollution)
```

10.4.2 First plot the data

A sensible starting point is to visualise the data. Useful ways of doing this include the following:

- Plotting the response against each covariate separately;
  - This will highlight which covariates are highly correlated with the outcome, and therefore, potentially very important in our model.
- Plotting the explanatory variables against one another (e.g. `pairs ()`; see Section 10.2);
  - This will highlight which covariates are highly correlated with each other.
  - Care needs to be taken if they are highly correlated (see Section 10.5.7); one possible option is to remove one or more of these covariates.
  - Don't forget that correlation between covariates means that omitting covariates from the model will change how the remaining covariates influence the outcome.
- Plotting the response against covariates for different combinations of other covariates (checking for interactions) (e.g. conditioning plots via the function `coplot ()` or interaction plots via `interaction.plot ()`, see Section 10.2.4).

- This will highlight interaction terms that may be worth including in the model.
- However, building these plots to look for interactions between covariates generally means that you should have context-specific information to suspect an interaction may be at play.
- When there are lots of covariates, trying to do this for all possible interactions is not a sensible strategy!

Plotting the data will give a good indication of which covariates may play an important role when modelling the outcome. The more of a feel you get for the data the more likely you are to produce a meaningful model at the end of the process. It's also a good opportunity to check for potential problems with your data, for example high correlation between covariates. Only when we start building the model, however, can we assess just how important (or not) covariates really are in explaining the variability in the outcome.

10.4.3 Comparing nested models

There are no specific rules on where to start when building a model, but if you have a reasonably small number of covariates you could do worse than start by building a model with all covariates initially (which may include interaction terms if your initial data exploration suggested that these would be sensible). From this initial model, we must adopt the principle of parsimony and be prepared to simplify the model. This is often done on the basis of **deletion tests**: testing hypotheses about omitting covariates from a model.

You have already met the simplest type of a deletion test in Section 10.3.3: hypotheses relating to omitting a *single* covariate (or equivalently, testing the hypothesis that its regression coefficient is zero), given that the remaining covariates remain in the model. This was conducted using a t-test, and the results of which could be viewed in the output of `lm ()`.

You also saw a more severe form of deletion test in 10.3.6, which tested whether we could omit *all* covariates from the model. This was an F-test, and again, the results of this test could be viewed in the output of `lm ()`.

It would be reasonable to want to test a 'middle ground' hypothesis: whether we can omit a subset of the covariates in the model. For example, supposing that there are several covariates present in the model with associated regression coefficients $\beta_s, \beta_t, \ldots, \beta_u$, interest may be in testing a hypothesis such as

$$H_0 : \beta_s = \beta_t = \ldots = \beta_u = 0 , \tag{10.5}$$

for distinct indices $s, t, \ldots, u \in \{1, \ldots, p\}$. We cannot simply look at the result of the t-test for each of $\beta_s, \beta_t, \ldots, \beta_u$. We need a more sophisticated method that checks whether we can remove *all* covariates listed *simultaneously* (i.e. $\beta_s = \beta_t = \ldots = \beta_u = 0$). In the context of multiple linear regression, such hypotheses can be tested using F-tests (or possibly likelihood-ratio tests). Notice that the F-test given within the `lm ()` output – which tested whether we could omit *all* covariates – is the same F-test as here but applied to all covariates rather than just a subset.

A very important point to remember when we use these F-tests is that this is a comparison of **nested** models: we compare a model which uses a particular set of covariates with another model in which a *subset* of these covariates have been removed. This test can't be used when we are comparing non-nested models.

Box 10.6: Nested models

A model, let us call it Model A, is nested within another model, let us say Model B, if it contains a subset of the covariates that appear in Model B.

Running these F-tests is simply done in *R*. Let us consider our ozone data once again. At the moment, we have three covariates: `rad`, `temp`, and `wind`. We can test whether we can omit (simultaneously) the variables `rad` and `temp` from the model using:

```
ozone_mod2 <- lm (ozone ~ wind)
anova (ozone_mod2, ozone_mod1)

Analysis of Variance Table

Model 1: ozone ~ wind
Model 2: ozone ~ rad + temp + wind
  Res.Df   RSS Df Sum of Sq      F    Pr(>F)
1    109 76040
2    107 47964  2     28076 31.316 1.965e-11 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Here, the resulting p-value is $1.9647811 \times 10^{-11}$ – which is very small – so we reject the null hypothesis (which said that we could omit these variables from the model with minimal consequences), and we on balance prefer to retain these covariates in the model. This does not mean to say that both are important: only that at least one of them is important enough to prefer to keep both covariates rather than omit both.

Let us check that when we apply this F-test in testing whether we can omit all covariates, we get the same p-value as the `lm ()` gave in its output.

```
ozone_mod3 <- lm (ozone ~ 1)
anova (ozone_mod3, ozone_mod1)

Analysis of Variance Table

Model 1: ozone ~ 1
Model 2: ozone ~ rad + temp + wind
  Res.Df    RSS Df Sum of Sq      F    Pr(>F)
1    110 121802
2    107  47964  3     73838 54.907 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Remember that the nested model this time is the intercept-only model, and in conducting this test, we get the same p-value as given by the F-test in the output of `lm ()`, as expected (see Section 10.3.6).

What would happen if we used our F-test to test whether we can omit exactly one covariate? Of course, the `lm ()` output already performs a t-test for this purpose, but if we were to perform the

F-test in this case, then the *p*-value would be identical to the *p*-value from the t-test in the output. In the example below, we conduct an F-test to assess whether we can omit `rad` from the model (i.e. a single covariate). Both give the same *p*-value.

```
ozone_mod4 <- lm (ozone ~ wind + temp)
summary (ozone_mod4)


Call:
lm(formula = ozone ~ wind + temp)

Residuals:
    Min      1Q  Median      3Q     Max
-42.160 -13.209  -3.089  10.588  98.470

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -67.2008    23.6083  -2.846  0.00529 **
wind         -3.2993     0.6706  -4.920 3.12e-06 ***
temp          1.8265     0.2504   7.293 5.32e-11 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 21.72 on 108 degrees of freedom
Multiple R-squared:  0.5817,Adjusted R-squared:  0.574
F-statistic:  75.1 on 2 and 108 DF, p-value: < 2.2e-16

anova (ozone_mod4, ozone_mod1)

Analysis of Variance Table

Model 1: ozone ~ wind + temp
Model 2: ozone ~ rad + temp + wind
  Res.Df   RSS Df Sum of Sq      F  Pr(>F)
1    108 50948
2    107 47964  1    2983.9 6.6565 0.01124 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

So when applied to a single covariate, the F-test gives an identical *p*-value to the t-test. It would be rather inconvenient if this were not the case.

10.4.4 Comparing non-nested models

What if we want to compare two models that are not nested? That is, the outcome is the same in both models, we're using the same data to fit the models, but the covariates in one of the models isn't a subset of the covariates in the other.

If you are looking to perform a statistical test here, you could do worse than consider the `lmtest` package (Zeileis and Hothorn, 2002) which includes tests for comparing non-nested normal linear models. These include the Cox test, `coxtest ()`, the Davidson–MacKinnon J test, `jtest ()`,

and the encompassing test, `encomptest ()` (though there are many 'flavours' of encompassing test out there).

Another popular option is to look at Akaike's information criterion (AIC). This is a statistic rather than a test, and comparing (non-nested) models boils down to choosing a model with the smallest AIC. It is known in the statistics trade as a **penalized log-likelihood**. See Section 2.5.2 for details on the likelihood function.

The idea is that the more parameters there are in the model, the better it fits the data, and so the larger the log likelihood. We could obtain a perfect fit if we had a separate parameter for every data point, but this model would have absolutely no explanatory power. The AIC takes the log-likelihood for the model under consideration then penalises it according to how many parameters we need to estimate. Mathematically, we have

$$\text{AIC} = -2 \times \text{log-likelihood} + 2(p + 1),$$

where p is the number of parameters in the model, and 1 is added for the estimated variance of the error terms (you could call this another parameter if you wanted to).

So the smaller the AIC (smaller because we're looking at the negative of the log-likelihood), the better the fit of the model. We could take two (potentially non-nested) models, compute the AIC for each of them, then the better fitting model would be the model with the *smallest* AIC.

There is an *R* function, `AIC ()`, to compute the information criterion directly from the model object. Let us build three models (notice that they aren't nested so the F-test from Section 10.4.3 wouldn't do), and try it out:

```
ozone_mod5 <- lm (ozone ~ temp + wind)
ozone_mod6 <- lm (ozone ~ rad + wind)
ozone_mod7 <- lm (ozone ~ rad + temp)
AIC (ozone_mod5, ozone_mod6)

           df      AIC
ozone_mod5  4 1003.327
ozone_mod6  4 1033.721

AIC (ozone_mod5, ozone_mod6, ozone_mod7)

           df      AIC
ozone_mod5  4 1003.327
ozone_mod6  4 1033.721
ozone_mod7  4 1020.820

detach (ozone_pollution)
```

When comparing `ozone_mod5` and `ozone_mod6`, we would prefer `ozone_mod5` because it has the smallest AIC. This is still the case when we also add `ozone_mod7` into the mix.

10.4.5 Dealing with large numbers of covariates

In Section 10.4, we suggested that a reasonable place to start building a model is to use all covariates in your dataset. With a reasonable sample size and a fairly small number of covariates, this shouldn't pose any problems. Simplifying your model is relatively straightforward too: after all, there

are only so many combinations of covariates you could try if we ignore the possibility of transforming the data as we do in Section 10.5.8.

When you have a large number of covariates to deal with however (and hopefully a much larger number of observations), things get more tricky. Now, the process of simplifying your model becomes unwieldly: a natural question to ask is 'where do I start?'. How do I choose subsets of covariates to test whether I can omit them? Lots of covariates in your model may have a large *p*-value in the individual t-tests of the regression coefficients, but testing whether this subset can be removed entirely may well yield a small *p*-value: *some* of those covariates were important after all. But which ones? To compound matters, remember that *how* covariates are related to the outcome in a multiple regression model changes depending on which other covariates are present. How do we go about building and simplifying a model in this case?

There are no hard-and-fast rules to apply here: it is a matter of personal choice. There are procedures which automate the selection of covariates for your outcome, but these should be used with extreme care:

1. They are automated routines that cannot take into account the context of the data collection mechanism, and therefore whether certain covariates are 'important' regardless of their impact on the outcome;
2. There is no theoretical justification to these procedures, they simply attempt to make your life easier. These algorithms can fit complicated models to completely random data;
3. The model that results from using these automated procedures should be used as a first model: this is where your model building starts, and it is usually simpler and more manageable to improve this particular model by tinkering with it than try to create one from scratch.
4. This process of tinkering may involve:
   - Adding covariates back into the model (especially if they are known to be important but were omitted by the automated procedure);
   - Simplifying the model further by omitting some of the covariates suggested by the automated procedure;
   - Adding interactions where/if necessary;
   - Transforming variables (see Section 10.5.8 for details).

The main automatic model-building procedures are backward elimination, forward selection, and stepwise regression. Each employs a set of pre-programmed rules to determine which covariates should be kept in the model, and which should not. In summary, these algorithms work as follows:

- forward selection starts with just the intercept and adds covariates to the model one-by-one;
- backward elimination starts by fitting a model with all possible covariates (if possible), and chooses covariates to omit one-by-one;
- stepwise regression is a combination of the above two procedures: this approach allows both entering and removing covariates at each step, allowing to later enter or remove a covariate that has been either removed or entered, respectively, at an earlier step.

Generally, these processes add or remove one covariate at a time, and continue doing so until the programmed 'stop' condition is met. One possible strategy is to run all three types of algorithm and

compare their results. If all three methods suggest roughly the same model then this can provide an *initial* direction for the analysis.

Implementation of these will vary from software to software. In *R*, for example, `step ()` provides forward, backward, or stepwise ('both') procedures. The algorithm behind the scenes here – what determines whether a covariate is added or deleted – is based on the current and proposed model's AIC. Implementation is straightforward, for example:

```
model_all <- lm (outcome ~., data = mydata)
model_auto <- step (model_all, direction = "both")
summary (model_auto)
```

The `.` in the `lm ()` function indicates that all variables should be used: we are staring with the model with all possible covariates, and using stepwise regression by choosing `direction = "both"`. Alternatively, `direction = "backward"` or `direction = "forward"` give the obvious other choices.

10.5 Checking model assumptions

In the case of the Normal linear model, as we have been studying here, the assumptions that we have made are

1. *Linearity*: the response is a linear combination of the parameters $\beta_1, \ldots, \beta_p$, and p covariates
$$y_i = \beta_1 x_{i1} + \ldots + \beta_p x_{ip} + \epsilon_i;$$
2. *Homoscedasticity*: the variance of ϵ_i is σ^2 for all $i = 1, \ldots, n$;
3. *Normality*: the error components $\epsilon_1, \ldots, \epsilon_n$ are Normally distributed;
4. *Independence of error components*: the error components are mutually independent (ϵ_i is independent of ϵ_j for $i \neq j$).

If one of the these assumptions fails, then the normal linear model might not be adequate for the data in hand. Nevertheless, given merely the data, it is impossible to *prove* (and hence be certain) that the above assumptions hold. We can merely develop tools that would give *indications against* the above assumptions: the best we can hope for is that we find *no evidence* of departure from these assumptions.

10.5.1 Residuals and standardised residuals

Our main tool in checking for departures from the model assumptions will be the residuals (see Section 10.1.1). We will call these **raw residuals**, and you can think of these as estimators of the error components. However, you should not think that errors and residuals are the same thing. Errors are theoretical constructs, whereas residuals are observed from data using our model.

Our errors are assumed to be independent and identically distributed, and even if this really is the case, the residuals can be dependent and not identically distributed. This poses a problem: even if the underlying assumptions on our error term are correct, our residuals may not behave in the same way. If, in that case, we see that residuals are correlated and/or not identically distributed, then we cannot draw conclusions about the underlying error term. This is a major drawback.

One way of getting around this is to use **standardised residuals** instead of the raw residuals: standardising ensures that *if* the errors are independent and identically normally distributed, then this will approximately be true for the standardised residuals (but not necessarily the raw residuals). There are many ways to standardise residuals – all of which require computing a function of the raw residual – but we'll stick to Pearson residuals here.

Pearson residuals can be computed in *R* painlessly using the `rstandard ()` function. This takes the ratio between the raw residuals and another quantity that is specific to the observation under consideration.

For our ozone data, the residuals and the standardised residuals can be computed as follows. Notice the difference in values between the raw (`ozone_res`) and standardised (`ozone_stres`) residuals.

```
attach (ozone_pollution)
ozone_res <- predict (ozone_mod1) - ozone
ozone_stdres <- rstandard (ozone_mod1)
head (cbind (ozone_res, ozone_stdres))

    ozone_res ozone_stdres
1  -7.9379186   0.38307433
2  -0.9898344   0.04731941
3  12.8133444  -0.60976646
4   0.4769439  -0.02338242
5   9.2724401  -0.45326743
6 -25.9497483   1.25472833
```

Instead of defining the (standardised) residuals ourselves, and creating our own plots for checking assumptions, another alternative is to use the `plot ()` function with our model as the argument like this (output not shown):

```
plot (ozone_mod1)
```

This generates a set of four model checking plots, but we prefer to use our own plots so that we can adjust them as necessary and use standardised residuals in all plots (some of *R*'s model checking plots use raw residuals).

10.5.2 Checking for linearity

The simplest way to check linearity is to plot the values of the response against the values of each covariate, like in Figure 10.5. If these plots indicate roughly linear relationships, then there is no evidence against the linearity assumption.

Another useful way to check linearity is by plotting the standardised residuals against each covariate in turn. If there is no evidence against the linearity assumption, then the plots should show a more or less random scatter around zero. If, on the other hand, you see patterns for a specific covariate, then this provides some evidence that the dependence of the response to that particular covariate isn't linear.

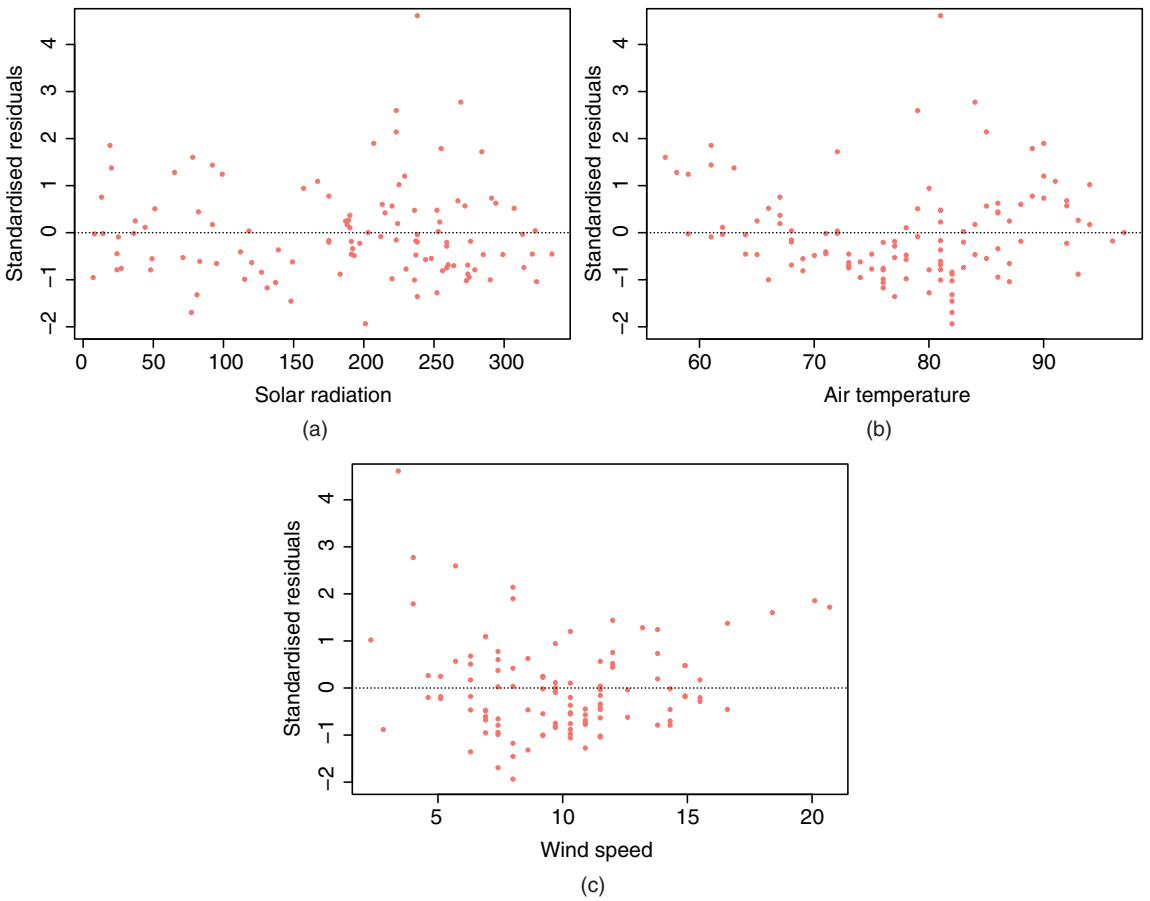


Figure 10.9 Checking for evidence against linearity.

The plots for our ozone data are in Figure 10.9 and are drawn like this (you could use the standard *R* plots by typing `plot (ozone_mod1)`, but this produces generic plots):

```
plot (rad, ozone_stdres, pch = 20, col = hue_pal ()(3)[1],
      ylab = "Standardised residuals", xlab = "Solar radiation", ylim = c (-2, 4.5))
abline (a = 0, b = 0, lty = 3)
plot (temp, ozone_stdres, pch = 20,  col = hue_pal ()(3)[1],
      ylab = "Standardised residuals", xlab = "Air temperature", ylim = c (-2, 4.5))
abline (a = 0, b = 0, lty = 3)
plot (wind, ozone_stdres, pch=20, col = hue_pal ()(3)[1],
      ylab = "Standardised residuals", xlab = "Wind speed", ylim = c (-2, 4.5))
abline (a = 0, b = 0, lty = 3)
```

There is some evidence here that the relationship between the outcome and air temperature is not linear, and perhaps to a lesser extent the same can be said for wind speed. The former in particular seems to have a slight 'U' shape. Problems with linearity could also be seen in the scatterplots of ozone versus each covariate in the last row of plots in Figure 10.5.

Be careful, however: the human eye is very good at detecting patterns – even in a genuinely random scatter of points – so we shouldn't make our lives difficult by hunting for patterns in these plots. Whether there is a pattern in these plots is often a judgement call, but the more you use these techniques, the more comfortable you will be in making such decisions.

What should be done if there is evidence against linearity? Transformation of either (some of) the covariates, or indeed, the outcome, can often result in a near-linear relationship between the transformed variables. The nature of these transformations can often be inferred from the above plots. For example, the relationship between the standardised residuals and air temperature has a slight 'U' shape indicating that some power transform of air temperature would be beneficial. See Section 10.5.8 for details.

10.5.3 Checking for homoscedasticity of errors

Box 10.7: Fitted values

The outcome predicted by the model, given a set of covariates.

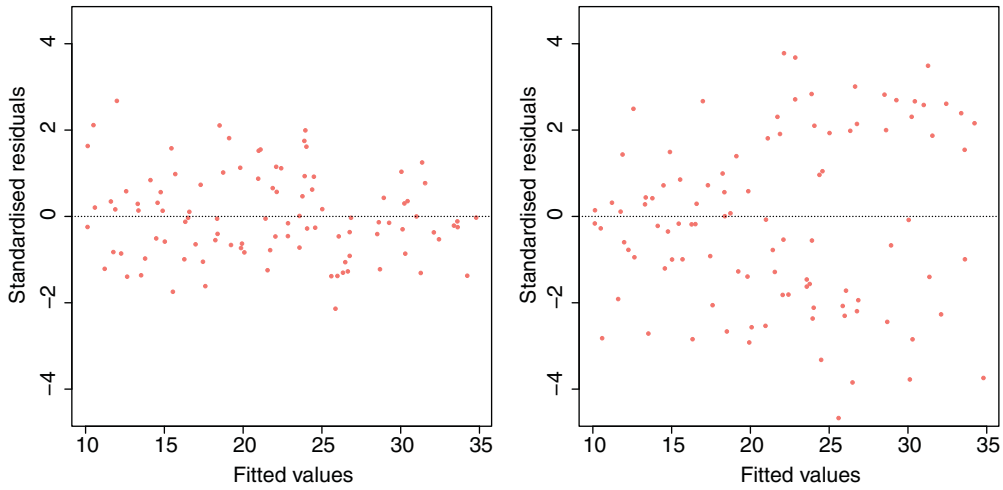
Homoscedasticity of the error terms simply means that we assume the variance of the error terms $\epsilon_1, \ldots, \epsilon_n$ is the same for all ϵ_i. If this assumption holds, then a plot of standardised residuals against the **fitted values** should look like the sky at night (points scattered at random over the whole plotting region), with no trend in the size or degree of scatter of the residuals. A common problem is that the variance increases with increasing value of the covariate (or vice versa) so that we obtain an expanding, fan-shaped pattern of residuals.

The plot in Figure 10.10a is what we want to see: no trend in the residuals with the fitted values. The plot in Figure 10.10b is a problem. There is a clear pattern of increasing residuals as the fitted values get larger, indicating that homoscedasticity does not hold in this case. Suggestions of what can be done in this case are given in Section 10.5.8.

10.5.4 Checking for normality of errors

The theory of multiple linear regression is based on the assumption of normal errors. If the errors are *not* normally distributed, then we shall not know how this affects our interpretation of the data or the inferences we make from it. In particular, consequences include but are not limited to inaccurate hypothesis testing and confidence intervals when using the model. This is, of course, a serious problem.

If the assumption of normally distributed errors holds, then the standardised residuals should be, approximately, from a standard Normal distribution. The most convenient way of checking this assumption is to look at normal probability plots. These plot the (ordered) standardised residuals against the quantiles of the standard Normal distribution. If the assumption holds, then we should see that these two quantities, when plotted against each other, roughly lie on the $y = x$ line.



(a) No evidence against homoscedasticity (b) Evidence against homoscedasticity

Figure 10.10 Checking for evidence against homoscedasticity.

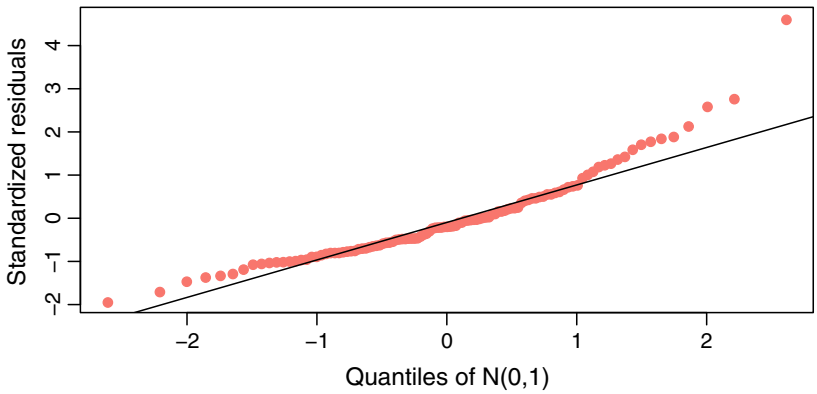


Figure 10.11 Normal probability plot.

Creating these plots is easy: `qqnorm ()` creates the scatterplot of the quantiles and standardised residuals, and `qqline ()` inserts the $y = x$ line for comparison. For the ozone data, the so-called 'QQ-plot' (or quantile-quantile plot) is given in Figure 10.11.

```
qqnorm (ozone_stdres, col = hue_pal ()(3)[1], main = "",
        ylab = "Standardized Residuals", xlab = "Quantiles of N(0,1)")
qqline (ozone_stdres)
```

```
detach (ozone_pollution)
```

The QQ-plot alerts us to a problem with the assumption of normality of errors, as the scatter of points form a curve rather than a straight line. It is important to remember, however, that no QQ-plot will

ever look perfect, and in particular, the two ends of the plot where there are fewer observations often do no lie exactly on the line even if the assumption of normality holds. Here, however, it is clear it does not: the most worrying features are that the middle of the plot clearly lies beneath the $y = x$ line, and that the tails of the plot lie well above the $y = x$ line. The latter implies that the distribution has a lighter left tail (the smallest standardised residuals are larger than we would expect) and heavier right tail (the largest residuals are larger than we would expect) than a standard Normal distribution.

10.5.5 Checking for independence of errors

The last of our assumptions – that of independence of the error term – is the most difficult to check. Independence can be violated in a number of ways; therefore, no single test or procedure can check this assumption. The best piece of advice is to understand how the data were collected: this should alert you to any obvious problems with this assumption. For example, if data are collected on lifestyle and diet, multiple observations from the same household may indicate that the observations are not truly independent which would likely imply that the errors won't be independent either. Alternatively, data collected repeatedly from the same individual over time (e.g. weight or a child's height) clearly violates this assumption.

Other cursory checks can be made by looking at the residual plots discussed in Sections 10.5.2 and 10.5.3; any patterns therein could be due to a lack of independence between errors, especially if the observations appear clustered.

One specific way that independence can be violated is if we have **serial correlation**. This can only happen if it is possible to order the observations in some way, for example in terms of time or location. Serial correlation occurs if the (ordered) errors are correlated with the errors that come before them. The Durbin–Watson test is used for testing for serial correlation and is implemented as part of the `car` package (Fox and Weisberg, 2019). You may need to install this package first.

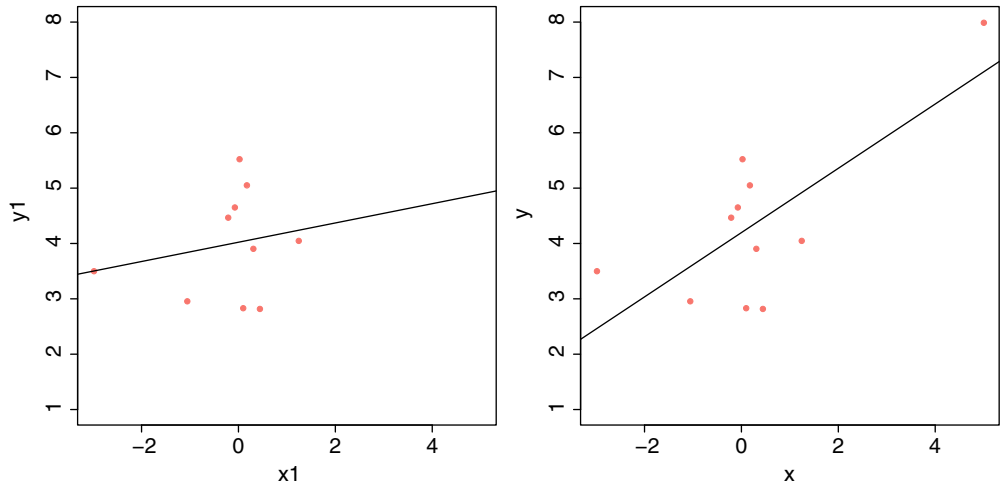
Notice that this test isn't suitable for our ozone data: the observations cannot be ordered by time or location. The dataset below, however, contains information on profit from the cultivation of a crop of carrots for a supermarket (`profit`) and associated costs of inputs including fertilizers, pesticides, energy, and labour (`profit`). These data are collected sequentially over time and therefore can be ordered; the order of observations in the dataset is already chronological. It makes sense, in this case, to check for serial correlation.

```
cost_profit <- read.table ("cost_profit.txt", header = T)
attach (cost_profit)
model_cost1 <- lm (profit ~ cost)
library ("car")
durbinWatsonTest (model_cost1)

 lag Autocorrelation D-W Statistic p-value
   1     -0.07946739      2.049899   0.854
Alternative hypothesis: rho != 0

detach (cost_profit)
```

There is no evidence of serial correlation in these residuals – the p-value is large – but note that this is just *one way* that the assumption of independent errors can be violated: a suggested absence of serial correlation in no way guarantees that this assumption holds in general.



(a) The standard data (b) With influential observation added

Figure 10.12 The influence of observations.

10.5.6 Checking for influential observations

One of the most common reasons for lack of fit is the existence of influential or unusual observations in the data. It is important to understand, however, that a point may *appear* to be unusual because of misspecification of the model, and not because there is anything wrong with the data. It is also worth noting that the analysis of residuals is a poor way of looking for influence: if a point is highly influential, it forces the regression line close to it, and hence, the influential point may have a very small residual.

Take a look at the data plotted in Figure 10.12a which includes a line of best fit. Now consider what happens when we add a single observation to these data, as in Figure 10.12b. The regression of y on x looks very different. The outlier is said to be highly *influential*. This makes our write-up much more complicated. We need to own up and show that the entire edifice depends upon the single point at (5, 8). This requires an explanation of two models rather than one. We cannot pretend that the point (5, 8) does not exist (that would be a scientific scandal), but we must describe just how influential it is.

Checking for the presence of influential points is an important part of statistical modelling. We cannot rely on analysis of the residuals, because by their very influence, these points force the regression line close to them. If we look at Figure 10.12b, we can see that the line of best fit is close to it, and closer than it is to some of the other observations. Looking at residual plots, therefore, might not be the best way to detect such observations.

One option is to use existing measures of influence. Some of these are usefully reported by the `influence.measures ()` function in *R*. When we run this function on our regression model, it helpfully highlights observations that may be influential using a `*` in the `inf` column:

```
reg <- lm (y ~ x)
influence.measures (reg)

Influence measures of
 lm(formula = y ~ x):
```

```
    dfb.1_    dfb.x   dffit cov.r  cook.d    hat inf
1  -0.1578  0.03305 -0.1581 1.319 0.01365 0.0951
2   1.3872 -1.49085  1.8979 0.277 0.82823 0.2374   *
3  -0.3649  0.18246 -0.3869 1.100 0.07378 0.1169
4  -0.4103  0.08364 -0.4108 0.969 0.07903 0.0948
5   0.0595  0.02480  0.0687 1.399 0.00264 0.1045
6   0.1635 -0.04472  0.1647 1.320 0.01480 0.0981
7   0.2847 -0.07210  0.2862 1.168 0.04207 0.0971
8  -0.1348  0.02784 -0.1350 1.340 0.01004 0.0949
9  -0.0740  0.02302 -0.0750 1.389 0.00314 0.1004
10 -0.2850  0.00369 -0.2877 1.143 0.04221 0.0909
11  0.8794  4.73210  5.0006 4.511 9.58238 0.8698   *
```

The statistics reported by `influence.measures ()` are

- DFBETAS (`dfb.1` and `dfb.x`): This looks at the difference in the regression parameters – here the intercept and slope – when we include the observation and when we don't.
- DFFITS (`dffit`): This is a function of the difference between the predicted value of an observation versus its predicted value if we dropped the observation from the model.
- Covariance ratio (`cov.r`): This looks at the effect of deleting each observation in turn on the variance–covariance matrix of the estimated regression parameters.
- Cook's distance (`cook.d`): This looks at the effect that omitting an observation has on *all* predicted values. Compare with DFFITS.
- Leverage (`hat`): This reports the *i*th diagonal of the so-called **hat matrix**, which is a measure of influence of the *i*th observation.

The observations with high influence are highlighted by an asterisk. To extract the subscripts of the influential points, use the `is.inf` attribute like this:

```
influence.measures (reg)$is.inf

   dfb.1_ dfb.x dffit cov.r cook.d   hat
1   FALSE FALSE FALSE FALSE  FALSE FALSE
2    TRUE  TRUE  TRUE  TRUE   TRUE FALSE
3   FALSE FALSE FALSE FALSE  FALSE FALSE
4   FALSE FALSE FALSE FALSE  FALSE FALSE
5   FALSE FALSE FALSE FALSE  FALSE FALSE
6   FALSE FALSE FALSE FALSE  FALSE FALSE
7   FALSE FALSE FALSE FALSE  FALSE FALSE
8   FALSE FALSE FALSE FALSE  FALSE FALSE
9   FALSE FALSE FALSE FALSE  FALSE FALSE
10  FALSE FALSE FALSE FALSE  FALSE FALSE
11  FALSE  TRUE  TRUE  TRUE   TRUE  TRUE
```

So it seems that our additional observation is detected as being unusual in this analysis. There are various rules of thumb for deciding whether an observation is influential according to a particular measure. A nice introduction can be found in the documentation of the `olsrr` package (Hebbali, 2020).

10.5.7 Checking for collinearity

Some dependence between covariates is perfectly normal and to be expected: the covariates are not required to be *independent* of one another. This is a common misconception, probably due to the unfortunate term 'independent variables' used by some when describing covariates.

While dependent covariates are not unusual, too much dependence between them can cause problems when fitting multiple linear regression models. In the worst-case scenario, where a linear combination of at least some of the covariates can be perfectly predicted from a subset of the other covariates, the algorithm that estimates the regression coefficients fails. The result is no output, which may be the first time this problem is encountered if pre-analysis checks, as recommended in Figure 10.6, are not conducted. Common examples where this might occur include accidentally using a two (or more) covariates that contain the same information (e.g. if we had a covariate measuring weight in grams, and another measuring the same weight in pounds, then we can predict precisely one from the other causing collinearity). Removing one of these covariates should resolve the problem. More subtle situations where several covariates act together to perfectly predict another covariate are more difficult to spot.

Box 10.8: Linear combinations

Combination of covariates $x_{ik}, \ldots, x_{ip}$ of the form $a_k x_{ik} + \ldots + a_p x_{ip}$, for constants $a_k, \ldots, a_p$.

Collinearity isn't just a problem when a covariate can be perfectly predicted from a linear combination of other covariates. Far more common, and probably more difficult to spot, is when a covariate is highly correlated with such a combination of other covariates. While in this case it is very probable that the regression coefficients can be estimated, other problems may appear:

- estimated standard errors are unexpectedly large;
- estimated regression coefficients may have signs that don't make sense (e.g. we expect covariate *A* to be negatively correlated with the outcome, but its regression coefficient is large and positive), but this may be the case due to reasons other than collinearity;
- many of the covariates are insignificant, despite strong observed relationships with the outcome observed during preliminary analysis.

Spotting potential collinearity is best done at the preliminary analysis stage thereby avoiding this problem entirely. Good strategies to detect collinearity include the following:

1. Plot covariates against one another, for example using the `pairs ()` function. Are any covariates highly correlated? Of course, this only looks for dependence between *pairs* of variables, but is a good initial strategy.
2. Compute measures such as the **variance inflation factor**, or **VIF** for short, which looks at linear dependence between each covariates and *all the other covariates*. This produces a single number for each covariate: a measure of collinearity.
3. If you have already started building your model, omitting some covariates and looking at the estimates from this simpler model may also be helpful: though they are expected to change as we add or remove covariates, a substantial change may indicate a problem with collinearity.

The first strategy may seem like a poor relation to the second, but this is probably something that you would be doing anyway during your preliminary analysis: you may as well spend a few additional seconds in looking for any signs of collinearity while you're getting to know your data.

Let us return to our ozone pollution data. We have already seen (Figure 10.5) that there is some fairly strong correlation between the covariates that could potentially mean that collinearity could become an issue. But is the correlation sufficiently pronounced to be a cause for concern? At this point, calculating the VIF score for each covariate may be helpful. The function `vif ()` in the `car` package (Fox and Weisberg, 2019) does just that.

```
attach (ozone_pollution)
library (car)
summary (ozone_mod1)


Call:
lm(formula = ozone ~ rad + temp + wind)

Residuals:
    Min      1Q  Median      3Q     Max
-40.485 -14.210  -3.556  10.124  95.600

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -64.23208   23.04204  -2.788  0.00628 **
rad           0.05980    0.02318   2.580  0.01124 *
temp          1.65121    0.25341   6.516 2.43e-09 ***
wind         -3.33760    0.65384  -5.105 1.45e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 21.17 on 107 degrees of freedom
Multiple R-squared:  0.6062,Adjusted R-squared:  0.5952
F-statistic: 54.91 on 3 and 107 DF,  p-value: < 2.2e-16

vif (ozone_mod1)

     rad     temp     wind
1.095241 1.431201 1.328979

detach (ozone_pollution)
```

The VIF for each covariate will be at least 1, with a VIF of 1 indicating that the covariate is approximately linearly independent of the other covariates. The higher the VIF, the more problematic the collinearity. As a rule of thumb, a VIF larger than 5 deserves further investigation. Note that the VIF isn't a cure-all: while it may tell you that there is some linear dependence between the covariates it doesn't tell you between exactly which ones, or what you should do next. The latter requires investigation that only you, the researcher, can carry out.

In this case, the VIFs are all fairly close to 1: there is no evidence here that collinearity is an issue.

10.5.8 Improving fit

What should we do if we detect a problem with one of the model assumptions? In this case, the model might not be appropriate as it is (or doesn't *fit* or *describe* the data particularly well). Then what? There are a number of options to try, including (in order or increasing complexity):

- transform the covariate(s) and/or the outcome;
- use weighted least squares;
- abandon the model and try a more sophisticated method such as generalised linear models.

We'll discuss the first of these options, with a brief look at weighted least squares. The models used in the third option are introduced in later chapters of this book. See, for example, Chapters 11, 12 and 14.

Transforming covariates

The idea of transforming covariates and/or the outcome through some function may sound strange: if we, for example, take the square root of (one of) the covariates, is the model still linear? Actually, it is. We assume the outcome is *linear in the parameters (regression coefficients)*. That is, we are free to transform the covariates (and indeed, the outcome). This is easiest to see when we consider simple linear regression models, though the same applies when you have multiple covariates.

In Figure 10.13, we see two examples:

- it is not clear whether the relationship between the untransformed variables in Figure 10.13a is linear. Transforming both the covariate and the outcome produces a clearer pattern which we might be happy to call linear (Figure 10.13b). In this case, both variables are transformed using the (natural) log function;
- the relationship between the untransformed variables in Figure 10.13c is not linear. Transforming just the covariate produces a linear relationship in Figure 10.13d. In this case, the covariate is squared to produce this linear relationship while the outcome remains untouched.

The earlier statement that the outcome is *linear in the parameters (regression coefficients)* might make more sense now. In the first example above, think of $\log(y)$ as the outcome and $\log(x)$ as the covariate, so that simple linear regression is a suitable strategy to model the relationship between $\log(y)$ and $\log(x)$. In the second example, think of the covariate as x^2 – and not x – so that a simple linear regression would be a suitable way of describing the relationship between y and x^2. Note the implication: the idea of building *linear* models isn't as restrictive as it may initially seem.

How should we decide whether to apply a transformation to the covariate(s), outcome, or both? If a transformation is necessary, which transformation should we apply?

This is easiest with the simple linear regression model. A scatterplot will help us in giving an educated guess to which transformation might be suitable. Usually, however, this is a matter of trial-and-error until we find the best transformation(s) to achieve linearity (if indeed this is possible). Common transformations include the following:

- logarithmic functions;
- power functions (including the square root);
- reciprocals.

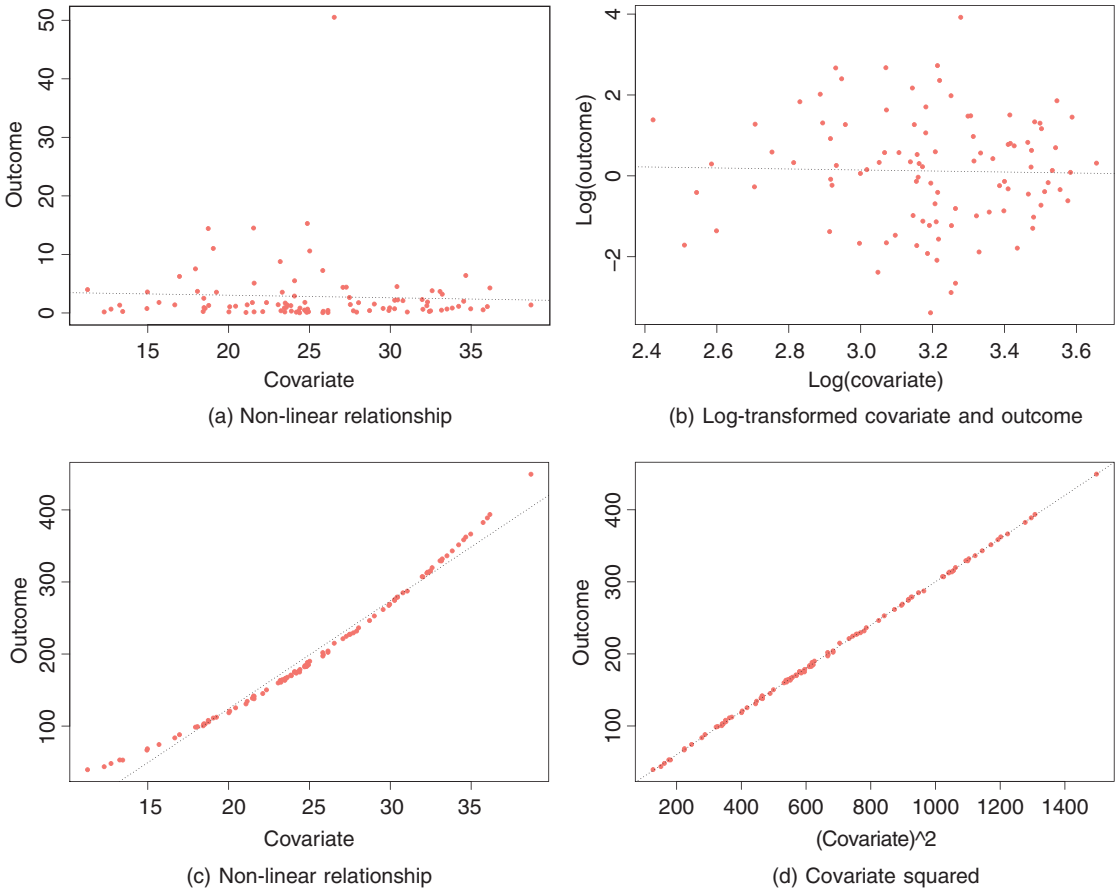


Figure 10.13 Transformations.

It might help to first of all consider common transformation of the covariate, before moving on to transform the outcome if it is necessary (as it was in Figure 10.13a).

For multiple linear regression models where we can't simply draw a scatterplot, we rely on information from the plots used to check linearity in Section 10.5.2: plotting the standardised residuals against each of the covariates. These plots can be used to detect problems in linearity, and therefore, by default we may be able to take an educated guess as to a transformation to 'fix' issues. The plots in Figure 10.14 show various common problems with linearity and suggested fixes.

These plots take experience in deciphering, and there's nothing wrong in trying a variety of transformations to find a combination that improves linearity.

For our ozone pollution data, for example, the plots used to look at linearity are given in Figure 10.9, the plot concerning air temperature (Figure 10.9b) caused the most concern. Here we see a slightly 'U'-shaped pattern to the residuals over the values of air temperature. We could try a square transformation of temperature, or even the reciprocal, but in this case these transformations don't help very much. Figure 10.15 shows the resulting plots when we transform `temp` by squaring it.

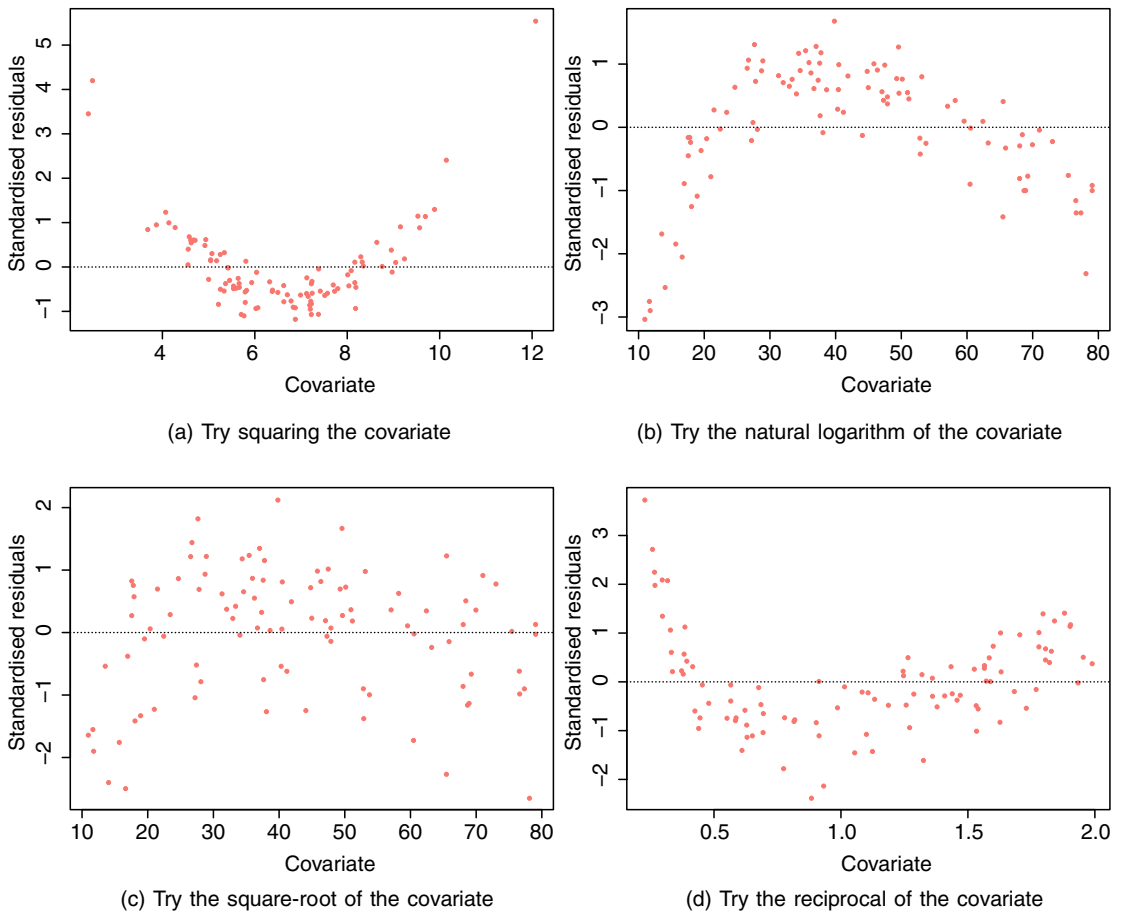


Figure 10.14 Transformations.

```
attach (ozone_pollution)
sq_temp <- temp^2
ozone_mod8 <- lm (ozone ~ wind + rad + sq_temp)
summary (ozone_mod8)

Call:
lm(formula = ozone ~ wind + rad + sq_temp)

Residuals:
    Min      1Q  Median      3Q     Max
-39.831 -13.790  -3.226  10.103  96.975

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -5.64005   14.17817  -0.398   0.6916
wind        -3.22264    0.64429  -5.002 2.24e-06 ***
```

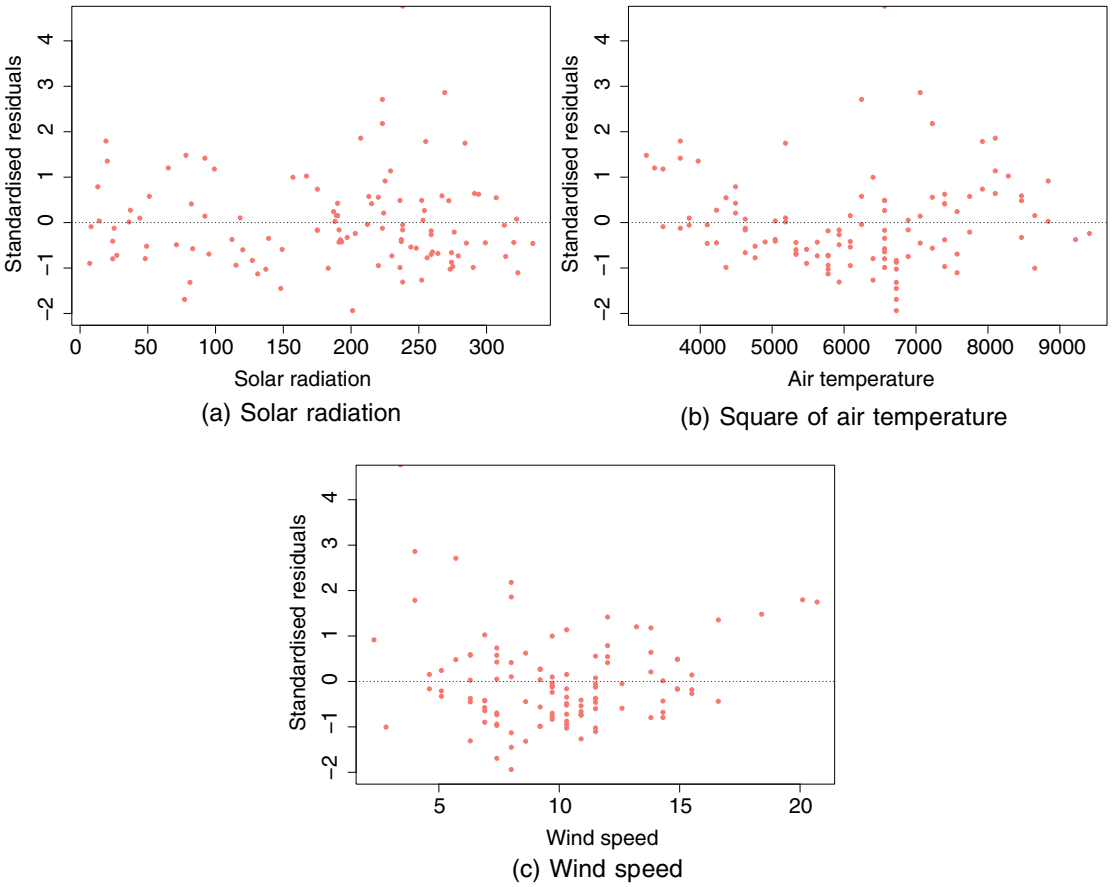


Figure 10.15 Transformations.

```
rad          0.05933    0.02272   2.611   0.0103 *
sq_temp      0.01120    0.00162   6.915 3.54e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 20.8 on 107 degrees of freedom
Multiple R-squared:  0.6199,    Adjusted R-squared:  0.6092
F-statistic: 58.16 on 3 and 107 DF,  p-value: < 2.2e-16

ozone8_stdres <- rstandard (ozone_mod8)
```

Given that a range of transformations (not shown) didn't improve matters here, it's natural to turn our attention to transforming the outcome. From the plots, we have used so far it is hard to know whether we should try this, let alone what the optimal transformation would look like.

In these circumstances, the Box–Cox transformation offers a simple empirical solution. The idea is to find a power, λ (lambda), so that transforming the outcome via y^{λ} provides a more linear

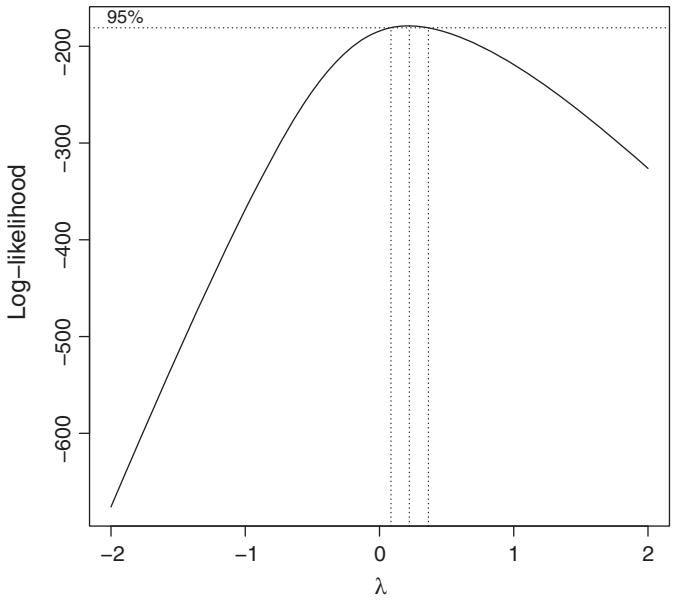


Figure 10.16 (Log)-likelihood as a function of lambda.

relationship with the covariates. The value of lambda can be positive or negative, when a λ of zero is suggested, then this implies a log transformation, while a λ of 1 implies that a transformation isn't necessary. The function `boxcox ()` in the package `MASS` (Venables and Ripley, 2002) automates this procedure. This function yields a plot from which we can deduce the 'best' value of λ to use. This is a **likelihood** function, and we would like to find the value of λ that *maximises* the likelihood.

Box 10.9: Box-Cox and maximising the likelihood

The graph produced by `boxcox ()` is the likelihood when the covariates are fitted to a model with $(y^\lambda - 1)/\lambda$ as the outcome. Whatever our preferred value for λ, this implies that fitting a model with y^λ is suggested: once λ is fixed, then we can rearrange the implied equation so that the outcome is y^λ instead of $(y^\lambda - 1)/\lambda$.

Applying `boxcox ()` to our ozone pollution data produces the plot in Figure 10.16.

```
library (MASS)
boxcox (ozone ~ rad + temp + wind)
```

The plot suggests that a power transformation may be suitable. It's difficult to see exactly what value of λ that the maximum of the (log)-likelihood suggests, but this does not matter: a value in the right vicinity is all we need (and it's probably better to choose something that is easy to describe). Here, the maximum seems to happen around the 0.25 mark, suggesting that $y^{1/4}$ might be the way to go, but equally we could consider $\lambda = 0$ which suggests $\log(y)$ as this is also in the vicinity and might be easier to interpret and/or to explain to users of your model.

For ease of interpretation, let us go for a log transformation of the outcome. Figure 10.17 shows the resulting scatterplots, and we immediately see that things have improved, especially for `temp`. Though still not quite perfect, but taking into consideration the principle of parsimony, it is probably sensible to stop here: there isn't strong evidence against linearity now. Be sure to check the other model checking plots, however, as you could find that your transformation has had unintended consequences on the other assumptions.

```
ozone_mod9 <- lm (log (ozone) ~ wind + rad + temp)
summary (ozone_mod9)

Call:
lm(formula = log(ozone) ~ wind + rad + temp)

Residuals:
     Min       1Q   Median       3Q      Max
-2.06212 -0.29968 -0.00223  0.30767  1.23572

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.2611739  0.5534102  -0.472 0.637934
wind        -0.0615925  0.0157037  -3.922 0.000155 ***
rad          0.0025147  0.0005567   4.518 1.62e-05 ***
temp         0.0491630  0.0060863   8.078 1.07e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.5085 on 107 degrees of freedom
Multiple R-squared:  0.6645,    Adjusted R-squared:  0.6551
F-statistic: 70.65 on 3 and 107 DF,  p-value: < 2.2e-16

ozone9_stdres <- rstandard (ozone_mod9)
plot (rad, ozone9_stdres, pch = 20, col = hue_pal ()(3)[1],
      ylab = "Standardised residuals", xlab = "Solar radiation", ylim = c (-2, 4.5))
abline (a = 0, b = 0, lty = 3)
plot (temp, ozone9_stdres, pch = 20,  col = hue_pal ()(3)[1],
      ylab = "Standardised residuals", xlab = "Air temperature", ylim = c (-2, 4.5))
abline (a = 0, b = 0, lty = 3)
plot (wind, ozone9_stdres, pch=20, col = hue_pal ()(3)[1],
      ylab = "Standardised residuals", xlab = "Wind speed", ylim = c (-2, 4.5))
abline (a = 0, b = 0, lty = 3)
detach (ozone_pollution)
```

Finally, *not all ailments with linearity can be fixed by transforming the covariates and/or outcome*. Other approaches such as non-linear models may be necessary. See Chapters 12 and 14 for details.

Weighted least squares

The default is for all the values of the response to have equal weights (all equal to 1). This might not be what we want: we could, for example have more faith in some of the observations than others and may want to reflect this in the *weighting* that the observations are given in an analysis.

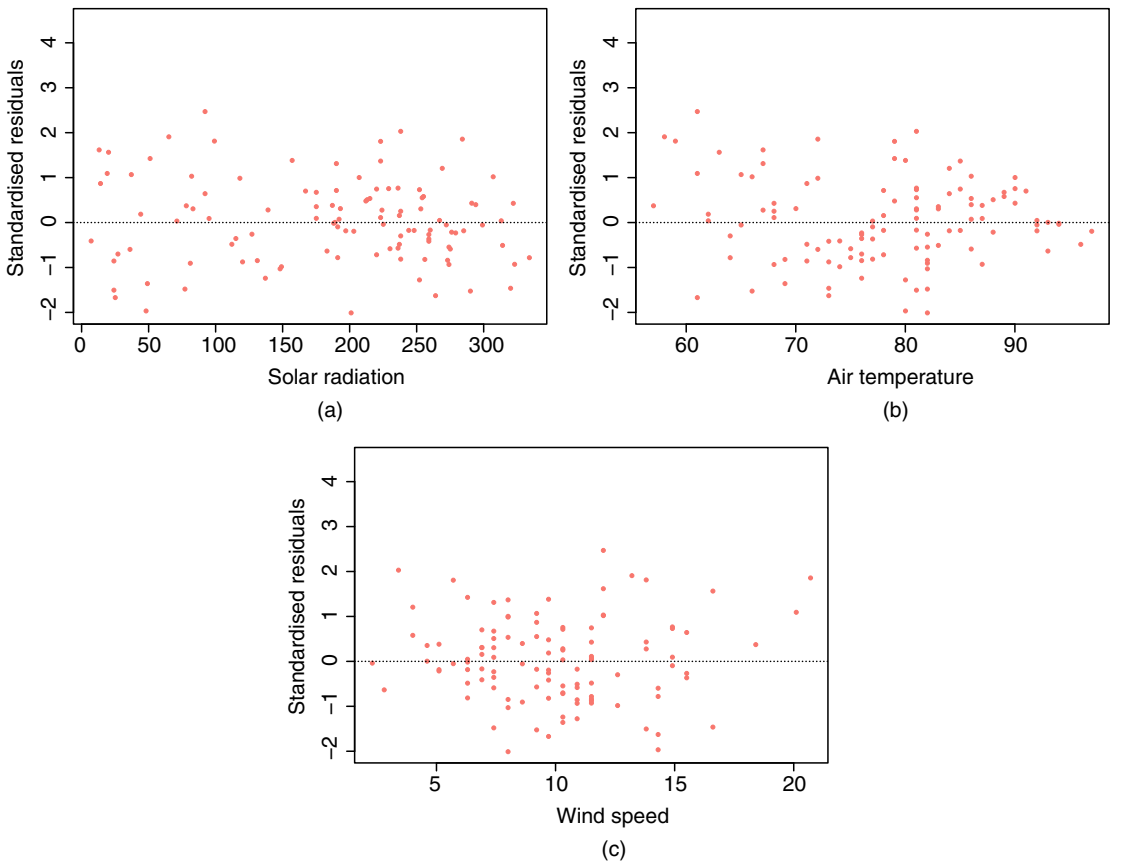


Figure 10.17 Transformations.

Where data points are to be weighted unequally, the classical approach is to weight each value by the inverse of the variance of the distribution from which that point is drawn. This downplays the influence of highly variable data and can result in a better fitting model. This strategy relies on us having a suitable variable which we can use to weight the observations, which is very unusual in practice.

Let us consider another dataset, where the response is seed production (`Fruit`) with a continuous explanatory variable (`Root`, Root diameter), and a two-level factor (`Grazing`, with levels `Grazed` and `Ungrazed`). Let us take a look at the data initially:

```
ipomopsis <- read.table ("ipomopsis.txt", header = T)
names (ipomopsis)

[1] "Root"    "Fruit"   "Grazing"

ipomopsis_mod1 <- lm (Fruit ~ Grazing + Root, data = ipomopsis)
summary (ipomopsis_mod1)
```

```
Call:
lm(formula = Fruit ~ Grazing + Root, data = ipomopsis)

Residuals:
     Min       1Q   Median       3Q      Max
-17.1920  -2.8224   0.3223   3.9144  17.3290

Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)    -127.829      9.664  -13.23 1.35e-15 ***
GrazingUngrazed  36.103      3.357   10.75 6.11e-13 ***
Root             23.560      1.149   20.51  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 6.747 on 37 degrees of freedom
Multiple R-squared:  0.9291,Adjusted R-squared:  0.9252
F-statistic: 242.3 on 2 and 37 DF,  p-value: < 2.2e-16
```

Instead of using initial root size as a covariate (as above) we could use `Root` as a weight in fitting a model with `Grazing` as the sole categorical explanatory variable. 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. Needless to say, the use of weights alters the parameter estimates and their standard errors:

```
ipomopsis_mod2 <- lm (Fruit ~ Grazing, data = ipomopsis, weights = Root)
summary (ipomopsis_mod2)


Call:
lm(formula = Fruit ~ Grazing, data = ipomopsis, weights = Root)

Weighted Residuals:
     Min       1Q   Median       3Q      Max
-137.822  -53.551    0.381   30.259  145.132

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 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

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
```

Fitting root size as a statistical weight is scientifically wrong in this case: why should values from larger plants be given greater influence? Also, this analysis gives entirely the wrong interpretation

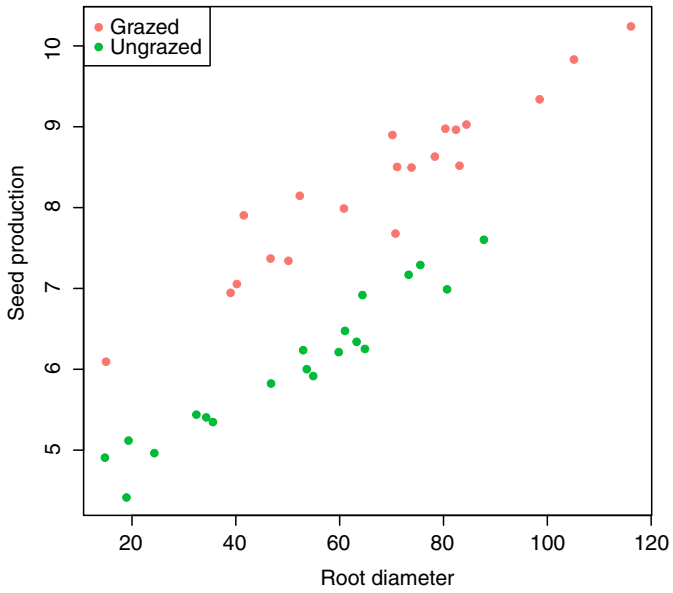


Figure 10.18 Plot of seed production by root diameter, coloured by grazing type.

of the data (ungrazed plants come out as being *less* fecund than the grazed plants). The original analysis in `ipomopsis_mod1` reverses this interpretation, showing that for a given root size, the grazed plants produced 36.013 *fewer* fruits than the ungrazed plants; the problem was that the big plants were almost all in the grazed treatment. We can see this clearly in Figure 10.18.

10.6 Using the model

10.6.1 Interpretation of model

There is little point in building a model unless we can interpret the results. This means not only understanding the output as described in Section 10.3 but also interpreting the regression coefficients: after all, if you have gone to the trouble of building a model to describe *how* a set of covariates impact the outcome, then it is reasonable to want to explain the role of each covariate in turn. The trick to doing this is to isolate the regression coefficient of interest. Understanding how this works in the case of the simple linear regression model is instructive, so we start by revisiting the caterpillar data from Section 10.1 for which the output is given again here for convenience.

```
summary (caterpillar_model)


Call:
lm(formula = growth ~ tannin, data = caterpillardata)

Residuals:
    Min      1Q  Median      3Q     Max
-2.4556 -0.8889 -0.2389  0.9778  2.8944
```

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  11.7556     1.0408  11.295 9.54e-06 ***
tannin       -1.2167     0.2186  -5.565 0.000846 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.693 on 7 degrees of freedom
Multiple R-squared:  0.8157,Adjusted R-squared:  0.7893
F-statistic: 30.97 on 1 and 7 DF,  p-value: 0.0008461
```

To *isolate* the intercept, we simply set `tannin = 0`. Isolating the (estimated) regression coefficient of tannin is also easy enough: take any value of tannin - let us say $a\%$ – and compare it to having $(a + 1)\%$ tannin in the diet. Then the expected change in growth rate between these two conditions of tannin is

$$(11.76 - 1.22 \times (a + 1)) - (11.76 - 1.22 \times a) = -1.22 \tag{10.6}$$

This leaves us with just the regression coefficient of tannin, as we wanted. Now let us think about how this converts into interpretation:

- the intercept tells us that when there is *no tannin* in the diet, the *expected* growth rate will be 11.76.
- the regression coefficient of tannin tells us that with a *one unit (one percent) increase* in tannin in the diet, the *expected* growth rate *decreases* by 1.22.

The italicised text above may seem pedantic, but without these caveats the interpretation wouldn't make sense. Notice also that the intercept here has a real-world interpretation, but this isn't always the case. If it doesn't make sense to set the covariate equal to zero, then by all means interpret the intercept but don't expect it to be meaningful. This *doesn't* mean that you don't need an intercept.

When we move to models with more than one covariate, the same ideas prevail. Take our experiment from Section 10.2.4 with weight as the response variable, and genotype, sex, and age as covariates. The first model we built was a multiple linear regression model without interactions (interactions make interpretation a little more tricky as we'll see shortly).

```
summary (gain_mod1)

Call:
lm(formula = Weight ~ Sex + Age + Genotype)

Residuals:
     Min       1Q   Median       3Q      Max
-0.40005 -0.15120 -0.01668  0.16953  0.49227

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)   7.93701    0.10066  78.851  < 2e-16 ***
Sexmale      -0.83161    0.05937 -14.008  < 2e-16 ***
Age           0.29958    0.02099  14.273  < 2e-16 ***
```

```
GenotypeCloneB  0.96778    0.10282   9.412 8.07e-13 ***
GenotypeCloneC -1.04361    0.10282 -10.149 6.21e-14 ***
GenotypeCloneD  0.82396    0.10282   8.013 1.21e-10 ***
GenotypeCloneE -0.87540    0.10282  -8.514 1.98e-11 ***
GenotypeCloneF  1.53460    0.10282  14.925  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.2299 on 52 degrees of freedom
Multiple R-squared:  0.9651, Adjusted R-squared:  0.9604
F-statistic: 205.7 on 7 and 52 DF,  p-value: < 2.2e-16
```

The intercept tells us that when all covariates are set to zero – so `Age = 0`, `Sex = 0` which implies female, and all genotype dummy variables are zero implying Genotype Clone A (the reference category)- the *expected* weight will be 7.94.

The slope of Age tells us that with a *one unit (one year) increase* in age, the *expected* weight *increases* by 0.3, *holding all other covariates constant*. The last part tells us that we have to keep all other covariates the same if we want to isolate the slope of Age in order to interpret it (try something similar to equation (10.6) to see this is the case).

The remaining covariates are categorical. We could continue the same method of interpretation, but as we'll see, this has particular meaning in this case. Starting with sex, remember that a male has `Sexmale = 1` while a female – the reference category – has `Sexmale = 0`. In this case, it doesn't always make sense to 'increase `Sexmale` by one unit' (what if `Sexmale = 1` already?) so our comparison is always to the reference category (in other words, the 'increase by one unit' is represented by going from female to male here). The regression coefficient of `Sexmale` tells us therefore that the weight of males is 0.83 *lower* than that for females on average, if Genotype and Age remain unchanged.

Finally, the same interpretation applies for all the `Genotype` dummy variables, but this time we need to compare each one to the reference category. For example, the regression coefficient of `GenotypeCloneD` tells us therefore that the weight of those with Genotype Clone D is 0.82 *higher* than for those with Genotype Clone A on average, if Sex and Age don't change.

Box 10.10: Interpretation of regression coefficients

Putting all this together, the regression coefficients in multiple linear regression models *without* interaction terms can be interpreted as follows:

- the intercept tells us that for a female with Genotype Clone A and age zero, their *expected* weight is 7.94;
- the regression coefficient of `age` tells us that with a *one unit (one year) increase* in age, the *expected* weight *increases* by 0.3, if Genotype and Sex don't change;
- the regression coefficient of `Sexmale` tells us that the weight of males is *expected* to be 0.83 *lower* than that for females, if Genotype and Age remain don't change.
- the regression coefficient of `GenotypeCloneD` tells us that the weight of those with Genotype Clone D is 0.82 *higher* than that for those with Genotype Clone A on average, if Sex and Age don't change.

When we add in interactions, things get a little trickier. Take the model `gain_mod2`, for example in which we had an interaction between sex and age. According to this model, how do sex, age, and genotype now impact weight?

```
summary (gain_mod2)


Call:
lm(formula = Weight ~ Sex * Age + Genotype)

Residuals:
     Min       1Q   Median       3Q      Max
-0.37202 -0.15893 -0.00302  0.15263  0.45188

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)    7.99759    0.11882  67.310  < 2e-16 ***
Sexmale       -0.95277    0.13933  -6.838 9.80e-09 ***
Age            0.27939    0.02970   9.406 9.98e-13 ***
GenotypeCloneB 0.96778    0.10290   9.405 1.00e-12 ***
GenotypeCloneC -1.04361   0.10290 -10.142 7.96e-14 ***
GenotypeCloneD 0.82396    0.10290   8.008 1.41e-10 ***
GenotypeCloneE -0.87540   0.10290  -8.507 2.36e-11 ***
GenotypeCloneF 1.53460    0.10290  14.914  < 2e-16 ***
Sexmale:Age    0.04039    0.04201   0.961    0.341
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.2301 on 51 degrees of freedom
Multiple R-squared:  0.9658,    Adjusted R-squared:  0.9604
F-statistic: 179.8 on 8 and 51 DF,  p-value: < 2.2e-16
```

Let us start with the easy one: genotype isn't involved with any interaction term, so our interpretation of the estimated regression coefficients for each dummy variable for genotype is interpreted as before.

For sex, however, notice that it appears twice in the model: once as a main effect, and again as part of an interaction effect. If we want to understand what the main effect of sex is telling us, the only way to do this is to set `age = 0`, so that the interaction term effectively disappears. Then we proceed as per normal: using model `gain_mod2`, the regression coefficient of sex tells us that the expected weight of males is 0.95 *lower* than it is for females, when `age = 0` and genotype remains the same.

For age, the same thought process is necessary: using model `gain_mod2`, the regression coefficient of age tells us that for a female (i.e. setting `Sexmale = 0`), the *expected* weight *increases* by 0.28 with a *one unit (one year) increase* in age, if Genotype doesn't change.

Finally, what about the interaction term itself? This one is even more cumbersome. Again, it might be easier to think about it as attempting to isolate this term from the model. The only way we can do this to compare:

- the expected change in weight for a one-year change in age for males, if genotype doesn't change;
- the expected change in weight for a one-year change in age for females, if genotype doesn't change.

The first comparison is estimated to be 0.28+0.04, while the second is estimated to be 0.28. Looking at the difference between the two gives the interaction effect.

Box 10.11: Interpretation of regression coefficients (including interactions)

Putting all this together, the regression coefficients in multiple linear regression models *with* interaction terms can be interpreted as follows:

- the intercept tells us that for a female with Genotype Clone A and age zero, their *expected* weight is 8;
- the regression coefficient of age tells us that for a female (i.e. setting `Sexmale = 0`), the *expected* weight *increases* by 0.28 with a *one unit (one year) increase* in age, if Genotype doesn't change;
- the regression coefficient of sex tells us that the expected weight of males is 0.95 lower than it is for females, when `age = 0` and genotype remains the same;
- the regression coefficient of `GenotypeCloneD` tells us that the weight of those with Genotype Clone D is 0.82 *higher* than that for those with Genotype Clone A on average, if Sex and Age remain unchanged;
- the regression coefficient of `Sexmale:Age` tells us that 0.04 is the expected difference between:
  - the expected change in weight for a one-year change in age for males, if genotype doesn't change;
  - the expected change in weight for a one-year change in age for females, if genotype doesn't change.

10.6.2 Making predictions

Using our model to make predictions about future observations is of interest in many situations. Using your model to make a point estimate of a future outcome, based on a set of covariates, is easy enough: you can think of it as plugging in the values of your covariates into the linear form of the model, along with the estimated regression coefficients. This is the *predicted mean* (or expected, or average) outcome based on your covariate values.

Before we make any predictions, you should be confident that the model fits the data well (otherwise, any prediction is unreliable at best) and that the values of the 'new' covariate(s) are in-line with what was used to build the model (otherwise, you are extrapolating outside the range of the data, for which you have no information).

R can, of course, automate the prediction using the `predict()` function. Let us use `ozone_mod1` to predict the outcome when:

- `rad = 110, temp = 60` and `wind = 15.3`;
- `rad = 110, temp = 80` and `wind = 9.5`.

Both sets of values are reasonable given the range of the values within the dataset, but the first set is unusual in its *combination* of the three covariate values.

```
attach (ozone_pollution)
predict (ozone_mod1, data.frame (rad = c (110, 110), temp = c (60, 80),
wind = c (15.3, 9.5)))

        1         2
-9.647168 42.735054
```

While the second set of 'new' covariates gives a broadly sensible estimate of pollution level, the first is negative which is clearly nonsense. We need to be careful, therefore, that not only are the covariate values sensible but that their combination is in-line with the data used to build the model. In this case, it was obvious something had gone wrong, but it may not be the case in other examples.

As with any prediction, it is not enough simply to give a point estimate: we need to qualify it with some estimate of variability. This could be in the form of a standard error or a confidence interval. It's likely that a confidence interval is more appealing, and this can be requested when using the `predict ()` function.

There's one complication before we can go further: *what type of prediction are you making*? Which of the following does your case fall in to?

1. Predicting the outcome for a *population*, given a specified set of covariate values;
2. Predicting the outcome for a *single observation*, given a specified set of covariate values.

The point estimate of these quantities will be the same, but their confidence intervals will differ: there is less variability in the (unknown) population (mean) outcome than there will be in the individual's outcome and so the confidence interval for the first case will be narrower than that of the second.

Box 10.12: Population vs individual prediction

This distinction is probably easier to visualise if we take a simple example. Imagine that you've built a model for the height of children, using age as a covariate. Suppose that you want to predict the height of a particular seven-year-old child, and also estimate the (mean) height of a population of seven-year-old children. There's a lot of variability in the heights of individual seven-year-olds, but when we think about the *population mean* height of seven-year-olds, there's a lot less uncertainty. The confidence interval we choose for our prediction must therefore reflect what we're trying to estimate.

Once you've decided what your prediction represents, you can specify this in the `predict` function as either:

1. `interval = confidence` when we want the confidence interval for the population (mean) outcome;
2. `interval = prediction` when we want the confidence interval for a specific observation.

```
predict (ozone_mod1, data.frame (rad = 110, temp = 80, wind = 9.5),
         interval = "confidence", level = 0.95)

       fit      lwr      upr
1 42.73505 37.20529 48.26482

predict (ozone_mod1, data.frame (rad = 110, temp = 80, wind = 9.5),
         interval = "prediction", level = 0.95)

       fit       lwr      upr
1 42.73505 0.4008949 85.06921

detach (ozone_pollution)
```

In the first, we request a 95% confidence interval for the population mean ozone concentration for the given covariate values, while in the second we assume that we want a confidence interval for a one-off observation. Notice how the latter is wider than the former, but the point estimate is the same for both.

10.7 Further types of regression modelling

We'll be extending the normal linear regression model over the next few chapters, and you will notice similarities in how we approach the modelling in *R*. Table 10.1 gives an overview of what's to come, including the relevant *R* function to fit the models.

For most of these models, a range of generic functions can be used to obtain information about the model, as we have in this chapter. The most important and most frequently used are listed in Table 10.2.

Table 10.1 Functions for various regression models.

`lm ()`	Fits a linear model with normal errors and constant variance; generally this is used for regression analysis using continuous explanatory variables.
`glm ()`	See Chapter 11. Fits generalised linear models to data using categorical or continuous explanatory variables, by specifying one of a family of **error structures** (e.g. Poisson for count data or binomial for proportion data) and a particular **link function**.
`gam ()`	See Chapter 12. Fits generalized additive models to data with one of a family of error structures (e.g. Poisson for count data or binomial for proportion data) in which the continuous explanatory variables can (optionally) be fitted as arbitrary smoothed functions using non-parametric smoothers rather than specific parametric functions.
`lme ()`; `lmer ()`	See Chapter 13. Fit linear mixed-effects models with specified mixtures of fixed effects and random effects and allow for the specification of correlation structure among the explanatory variables and autocorrelation of the response variable (e.g. time series effects with repeated measures). `lmer ()` allows for non-normal errors and non-constant variance with the same error families as a GLM.
`nls ()`	See Chapter 14. Fits a non-linear regression model via least squares, estimating the parameters of a specified non-linear function.

(*continued*)

Table 10.1 (Continued)

`nlme ()`	See Chapter 14. Fits a specified non-linear function in a mixed-effects model where the parameters of the non-linear function are assumed to be random effects; it allows for the specification of correlation structure among the explanatory variables and autocorrelation of the response variable (e.g. time series effects with repeated measures).
`loess ()`	See Chapter 12. Fits a local regression model with one or more continuous explanatory variables using non-parametric techniques to produce a smoothed model surface.
`tree ()`; `rpart()`	See Chapter 20. Fits a regression tree model using binary recursive partitioning whereby the data are successively split along coordinate axes of the explanatory variables so that at any node the split is chosen that maximally distinguishes the response variable in the left and right branches. With a categorical response variable, the tree is called a classification tree, and the model used for classification assumes that the response variable follows a Multinomial distribution.

Table 10.2 Frequently used functions to extract information about regression models.

`summary ()`	produces parameter estimates and standard errors from `lm ()`.
`plot ()`	produces diagnostic plots for model checking, including residuals against fitted values, normality checks, influence tests, etc.
`anova ()`	is a wonderfully useful function for comparing different models and producing ANOVA tables.
`update ()`	is used to modify the last model fit; it saves both typing effort and computing time.
`coef ()`	gives the coefficients (estimated parameters) from the model.
`fitted ()`	gives the fitted values, predicted by the model for the values of the explanatory variables included.
`resid ()`	gives the residuals (the differences between measured and predicted values of *y*).
`predict ()`	uses information from the fitted model to produce smooth functions for plotting a line through the scatterplot of your data. Make sure you provide a list or a dataframe containing all of the necessary information on each of the explanatory variables in your model to enable the prediction to be made.

References

Fox, J., & Weisberg, S. (2019). *An R companion to applied regression* (Third). Sage. https://socialsciences.mcmaster.ca/jfox/Books/Companion/.

Hebbali, A. (2020). *Olsrr: Tools for building OLS regression models* [R package version 0.5.3]. https://CRAN.R-project.org/package=olsrr.

Venables, W. N., & Ripley, B. D. (2002). *Modern applied statistics with S* (Fourth) [ISBN 0-387-95457-0]. Springer. https://www.stats.ox.ac.uk/pub/MASS4/.

Zeileis, A., & Hothorn, T. (2002). Diagnostic checking in regression relationships. *R News*, *2*(3), 7–10. https://CRAN.R-project.org/doc/Rnews/.