4 Introduction to the Linear Model

Simple Linear Regression

4.1. Word Frequency Effects

The last chapter focused on modeling distributions with means. This chapter teaches you how to condition a mean on another variable. That is, you will create models that predict *conditional means*—means that shift around depending on what value some other piece of data assumes. In modeling such conditional means, you move from the topic of univariate statistics (describing single variables) to bivariate statistics (describing the relationship between two variables). The approach you learn in this chapter is the foundation for everything else in the book.

Hundreds of studies have found that frequent words are comprehended faster than infrequent words (e.g., Postman & Conger, 1954; Solomon & Postman, 1952; Jescheniak & Levelt, 1994). Figure 4.1 is an example of this, displaying response durations from a psycholinguistic study conducted as part of the English Lexicon Project (Balota et al., 2007). The y-axis extends from 400*ms* (two fifths of a second) to 1000*ms* (one second).[1] Longer response durations (up on the graph) mean that participants responded more slowly; shorter response durations (down on the graph) mean they responded faster. The word frequencies on the x-axis are taken from the SUBTLEX corpus (Brysbaert & New, 2009). They are represented on a logarithmic scale ($\log_{10}$). Logarithms will be explained in more detail in Chapter 5. From now on I will simply talk of 'word frequency' instead of 'log frequency', as for understanding the basics of regression models the logarithmic nature of the scale is not relevant.

The relationship between response duration and frequency is neatly summarized by a line. This line is the regression line, which represents the average response duration for different frequency values. Simple linear regression is an approach that models a single continuous response variable as a function of a predictor variable. Table 4.1 shows some common terminological differences that come up in regression modeling. Some researchers and textbooks use the language of 'regressing y on x'. I prefer to speak of 'modeling y as a function of x'. Researchers often use the terms 'independent variable' and 'dependent variable' (the dependent variable y *depends* on the

1 In what is called a 'lexical decision task', participants were asked whether a word was English or not. For example, *horgous* is not an English word, but *kitten* is. In Figure 4.1, each point is averaged over multiple participants' responses.

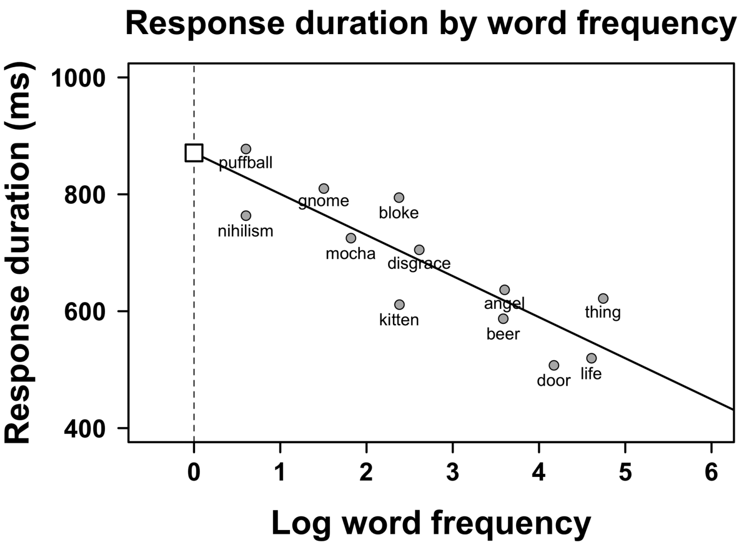

Figure 4.1. Response duration as a function of word frequency; each point represents the average response duration for a particular word; the x-axis is scaled logarithmically (see Chapter 5); the line represents the linear regression fit; the white square represents the intercept (the point where $x = 0$)

Table 4.1. Different names for response and predictors

y	x
response/outcome dependent variable	predictor independent variable explanatory variable regressor

independent variable x). I prefer to speak of x as a 'predictor', as it allows forming predictions for y, which I often call the 'response' or 'outcome' variable.

Generally, you specify regression in the direction of assumed causality (e.g., word frequencies affect response durations rather than the other way around). However, it is important to remember the slogan 'correlation is not causation', as the regression model cannot tell you whether there actually is a causal relationship between x and y.[2]

2 For some delightful examples of this principle in linguistics, see Roberts and Winters (2013). For example, cultures taking siestas speak languages with reduced morphological complexity.

4.2. Intercepts and Slopes

Mathematically, lines are represented in terms of intercepts and slopes. Let's talk about slopes first, then intercepts. In the case of the frequency effect shown in Figure 4.1, the slope of the line is negative. As the frequency values increase, response durations decrease. On the other hand, a positive slope goes 'up' (as *x* increases, *y* increases as well). Figure 4.2a shows two slopes that differ in sign. One line has a slope of +0.5, the other one has a slope of –0.5.

The slope is defined as the change in y (Δy, 'delta y') over the change in x (Δx, 'delta *x*').

$$slope = \frac{\Delta y}{\Delta x} \tag{E4.1}$$

Sometimes, the slogan 'rise over run' is used as a mnemonic for this calculation. How much do you have to 'rise' in *y* for a specified 'run' along the *x*-axis? In the case of the response duration data discussed in this chapter, the slope turns out to be $-70\frac{ms}{freq}$. Thus, for each increase in frequency by 1 unit, the predicted response duration decreases by 70ms. For a two-unit increase in word frequency, the predicted response duration decreases by 140ms ($= 70ms*2$), and so on.

However, a slope is not enough to specify a line. For any given slope, there is an infinity of possible lines. Figure 4.2b shows two lines with the same slope but differing 'intercepts'. You can think of the intercept informally as the point 'where the line starts' on the *y*-axis. As the *y*-axis is located at *x* = 0, this means that the intercept is the predicted *y*-value for *x* = 0. For the data shown in Figure 4.1, this happens to be the

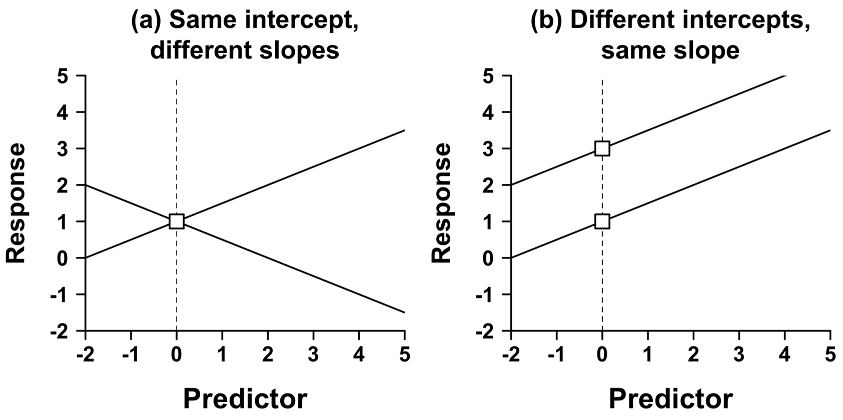

Figure 4.2. (a) Two lines with positive and negative slopes that go through the same intercept; (b) two lines with the same positive slope that have different intercepts

number 880ms (represented by the white square). Thus, for a (log) word frequency of 0, the model predicts the intercept 880ms.

Once the intercept and slope are fixed, there can be only one line. In math-speak, the line is said to be 'uniquely specified' by these two numbers. The intercept and slope are both 'coefficients' of the regression model. The letters b_0 and b_1 are commonly used to represent the intercept and the slope. Thus, the regression line has the following form:

$$y = b_0 + b_1 * x \tag{E4.2}$$

This equation yields different y means for different x-values. Let's plug in the actual coefficients from the regression shown in Figure 4.1.

$$response\ duration = 880ms + \left(-70\frac{ms}{freq}\right) * word\ frequency \tag{E4.3}$$

Because the slope has the unit $\frac{ms}{freq}$, multiplying it by a frequency value returns milliseconds. That is, $\frac{ms}{freq} * freq = ms$ (the two frequency units cancel each other out), highlighting how this regression model predicts response durations.

The coefficient estimates are the principal outcome of any regression analysis. A lot of your time should be spent on interpreting the coefficients—for example, by plugging in various values into the equation of your model to see what it predicts.

4.3. Fitted Values and Residuals

Let's see what response duration the regression model predicts for the word *script*. This word wasn't part of the original data, but if we know the word's frequency, the predictive equation E4.3 will churn out a response duration. It turns out that *script* has a (log) frequency of 3 in the SUBTLEX corpus, and thus:

$$response\ duration = 880ms + \left(-70\frac{ms}{freq}\right) * 3\ freq = 670ms \tag{E4.4}$$

The expected response duration for *script* is 670ms. Such a prediction is called a 'fitted value', as it results from 'fitting' a regression model to a dataset. In fact, all points along the regression line are fitted values. However, not all of these values may be equally sensible. Forming predictions based on a regression model generally only makes sense within the range of the attested data (this is called 'interpolation'). Regression models may produce odd results when predicting beyond the range of the attested data, what is called 'extrapolating'.[3] Word frequencies below zero make no

3 See Tatem, Guerra, Atkinson, and Hay (2004) for a hilarious extrapolation error. Based on the fact that female sprinters have increased their running speed more quickly than male sprinters over

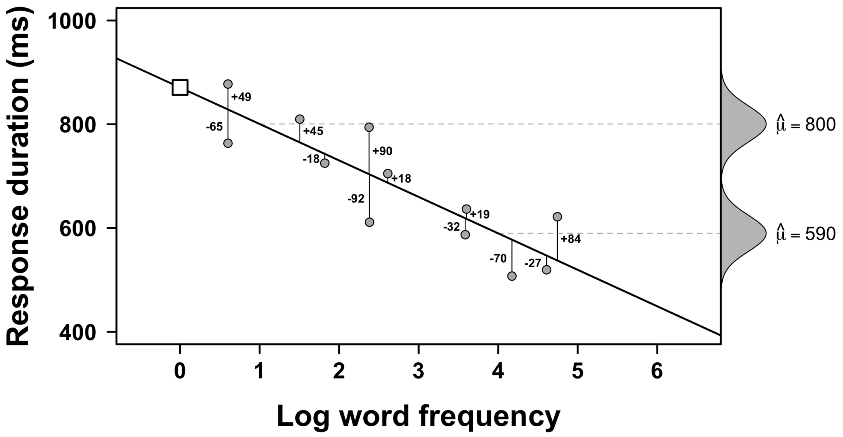

Figure 4.3. Regression line with vertical line segments indicating the residuals, which are positive (above the line) or negative (below the line); conditional means for $x = 1$ and $x = 4$ are projected onto the right y-axis; the density curves represent the constant variance and normality assumptions

sense, nevertheless, the regression model happily allows us to form predictions for negative values. For example, for the impossible x-value of –100, the model predicts a response duration of 7880ms. The model doesn't 'know' that negative frequencies make no sense.

The frequency model doesn't fit any of the data points perfectly. The extent to which the model is wrong for any specific data point is quantified by the residuals, which are the vertical distances of the observed data from the regression line, as shown in Figure 4.3. In this scatterplot, observed values above the line have positive residuals (+); observed values below the line have negative residuals (–). The actual numerical values represent how much the predictions would have to be adjusted upwards or downwards to reach each observed value.

The relationship between fitted values, observed values, and residuals can be summarized as follows:

$$\textit{observed values} = \textit{fitted values} + \textit{residuals} \tag{E4.5}$$

This equation can be rewritten the following way:

$$\textit{residuals} = \textit{observed values} - \textit{fitted values} \tag{E4.6}$$

the past few decades, these researchers claimed that, in 2156, women will run faster than men. As pointed out by Rice (2004) and Reinboud (2004), the same regression model would also predict instantaneous sprints in the year 2636.

Equation E4.6 shows that the residuals are what's 'left over' after subtracting your model's predictions from the data. Zuur, Ieno, Walker, Saveliev, and Smith (2009: 20) say that "the residuals represent the information that is left over after removing the effect of the explanatory variables".

Now that you know about residuals, the general form of a regression equation can be completed. All that needs to be done is to extend the predictive equation E4.2 with an 'error term', which I represent with the letter '*e*' in the equation below. In your sample, this term corresponds to the residuals.

$$y = b_0 + b_1 * x + e \tag{E4.7}$$

Essentially, you can think of the regression equation as being composed of two parts. One part is 'deterministic' and allows you to make predictions for conditional means (a mean dependent on *x*). This is the '$b_0 + b_1 * x$' part of the above equation. This part is deterministic in the sense that for any value of *x* that you plug in, the equation will always yield the same result. Then, there is a 'stochastic' part of the model that messes those predictions up, represented by *e*.

4.4. Assumptions: Normality and Constant Variance

Statistical models, including regression, generally rely on assumptions. All claims made on the basis of a model are contingent on satisfying its assumptions to a reasonable degree. Assumptions will be discussed in more detail later. Here, the topic will be introduced as it helps us to connect the topics of this chapter, regression, with the discussion of distributions from the last chapter.

For regression, the assumptions discussed here are actually about the error term *e*, that is, they relate to the *residuals* of the model. If the model satisfies the normality assumption, its residuals are approximately normally distributed. If the model satisfies the constant variance assumption (also known as 'homoscedasticity'), the spread of the residuals should be about equal while moving along the regression line.

A clear violation of the constant variance assumption is shown in Figure 4.4a. In this case, the residuals are larger for larger *x*-values. Figure 4.4b demonstrates a clear violation of the normality assumption. In this case, a histogram of the residuals reveals 'positive skew', i.e., there are a few infrequent extreme values (see Chapter 5 for a discussion of skew). It is important to emphasize that the normality assumption is not about the response but about the residuals. It is possible that a model of a skewed response measure has normally distributed residuals (see Chapter 12). A more in-depth assessment of assumptions is found in the multiple regression chapter (Chapter 6).

4.5. Measuring Model Fit with R^2

The residuals are useful for creating a measure of the 'goodness of fit' of a model—how well a model 'fits' the observations overall. A well-fitting model will have small residuals.

Consider an alternative model of response durations that ignores word frequency. The word frequency slope is 0 in this case; the same response duration is predicted regardless of word frequency (see Figure 4.5a). Notice that the residuals shown in

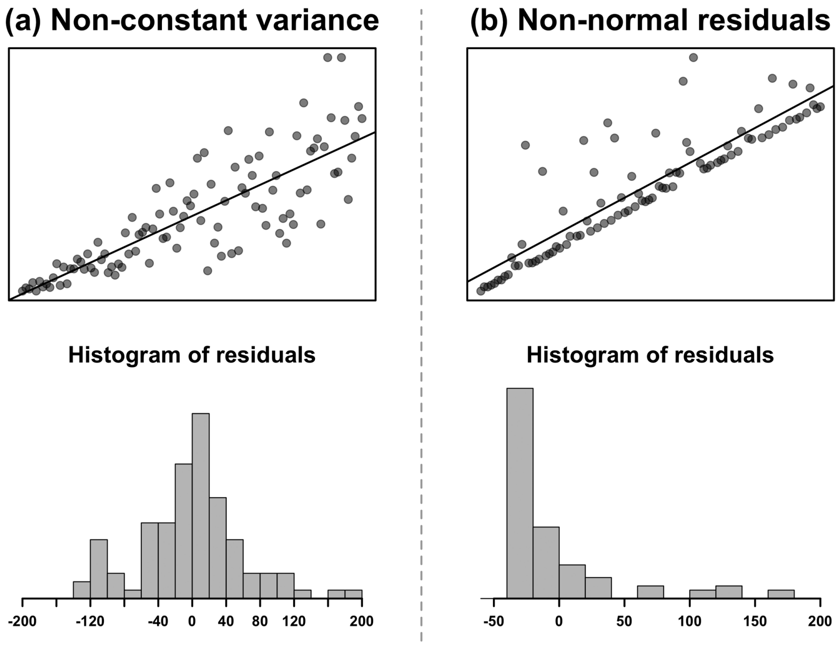

Figure 4.4. Clear violations of (a) constant variance and (b) normality

Figure 4.5a are overall larger than the residuals of the model shown in Figure 4.3a. This suggests that the fit for this zero-slope model is worse.

To get an overall measure of 'misfit', the residuals can be squared and summed.[4] The corresponding measure is called 'sum of squared errors' (*SSE*). The regression model with the slope of –70.24 has an *SSE* of 42,609. The zero-slope model has an *SSE* of 152,767, a much larger number. Imagine a whole series of possible slope values, as if you are slowly tilting the line in Figure 4.1. Figure 4.6 shows the *SSE* as a function of different slope values. This graph demonstrates that for the regression model of the response duration data, the estimated slope ($b_1 = -70.28$) results in the smallest residuals. Regression is sometimes called 'least squares regression' because it yields the coefficients that minimize the squared residuals. As the researcher, you only have to supply the equation $response\,duration = b_0 + b_1 * frequency$. In that case, you can think of b_0 and b_1 as 'placeholders' that are to be filled with actual numerical values based on the data.

Let's talk a bit more about the zero-slope model (Figure 4.5a). You can fit such a model by dropping the frequency predictor from your equation, which is the same as assuming that the slope is 0.

$$response\ duration = b_0 \tag{E4.8}$$

4 Why squaring? One reason is that this way you get rid of the negative signs. Otherwise the positive and negative residuals would cancel each other out.

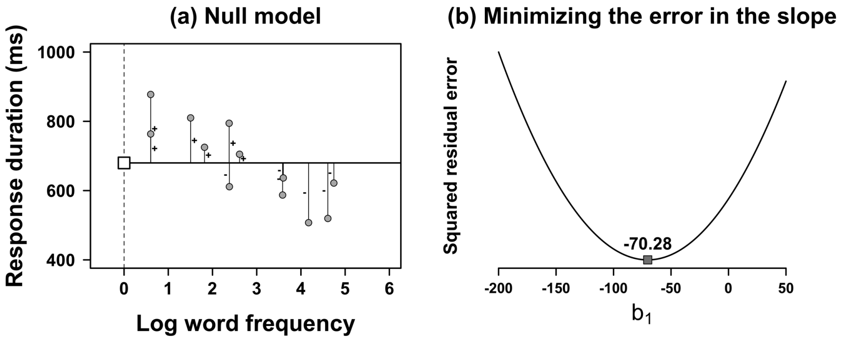

Figure 4.5. (a) A regression model predicting the same response duration regardless of frequency—vertical distances (residuals) are larger than in Figure 4.3a; (b) squared residual error for a whole range of slope estimates—the residuals are minimized for a slope of –70.28

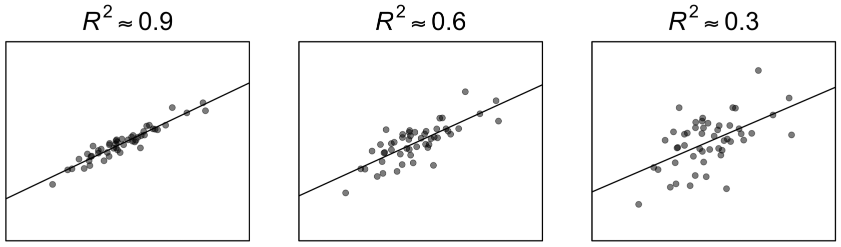

Figure 4.6. Larger residuals yield smaller R^2 values

This ‘intercept-only model’ model has not been instructed to look for an effect of frequency. What is the best estimate for the intercept in this case? In the absence of any other information, the best estimate is the mean, as it is the value that is closest to all data points (see Chapter 3).

The intercept-only model can be a useful a reference model or ‘null model’ for comparing *SSE* values. Remember that the main regression model (with frequency predictor) had an *SSE* of 42,609. Without context, this number is pretty meaningless; it is an unstandardized statistic that changes depending on the metric of the response. For example, if you measured response durations in seconds, rather than milliseconds, the *SSE* of our main model would shrink from 42,609 to 0.042609. The ‘null model’ without a slope can be used to put the *SSE* of the main model into perspective. It can be used to compute a ‘standardized’ measure of model fit, namely, R^2 (‘R squared’). The formula for R^2 is as follows:

$$R^2 = 1 - \frac{SSE_{model}}{SSE_{null}} \tag{E4.9}$$

The *SSE* achieves this standardization by dividing the main model's *SSE* by the corresponding null model's *SSE*. Division gets rid of the metric. Let's test this with the *SSEs* for both of the models just considered. The *SSE* of the main model (42,609) divided by the *SSE* of the null model (152,767) yields 0.28. The R^2 value then is $1 - 0.28 = 0.72$.

This number can be conceptualized as how much variance is 'described' by a model.[5] In this case, 72% of the variation in response durations can be accounted for by incorporating word frequency into the model. Conversely, 32% of the variation in response durations is due to chance, or due to factors the model omits. In actual linguistic data, R^2 values as high as 0.72 are almost unheard of. Language is complex and humans are messy, so our models rarely account for that much variance.

R^2 is actually a measure of 'effect size'. Specifically, R^2 measures the strength of the relationship between two variables (see Nakagawa & Cuthill, 2007). R^2 values range from 0 to 1. Values closer to one indicate better model fits as well as stronger effects, as shown in Figure 4.6.

Standardized metrics of effect size such as R^2 should always be supplemented by a thorough interpretation of the unstandardized coefficients. You have already done this when computing response durations for different frequency values. When looking at what your model predicts, it is important to use your domain knowledge. As the expert of the phenomenon that you study, you are the ultimate judge about what is a strong or a weak effect.

4.6. A Simple Linear Model in R

As always, you need to load the tidyverse package if you haven't done so already:

```
library(tidyverse)
```

Let's start by generating some random data for a regression analysis, specifically, 50 random normally distributed numbers (see Chapter 3).

```
# Generate 50 random numbers:

x <- rnorm(50)
```

Check the resulting vector (remember that you will have different numbers).

```
# Check first 6 values:

head(x)
```

```
[1] 1.3709584 -0.5646982 0.3631284 0.6328626
[5] 0.4042683 -0.1061245
```

5 People often say that R^2 measures the 'variance explained'. An informative blog post by Jan Vanhove (accessed October 16, 2018) recommends against this terminology, among other reasons because it sounds too causal: https://janhove.github.io/analysis/2016/04/22/r-squared

To be able to do anything bivariate, you need *y*-values to go with the *x*-values. Let's say that the intercept is 10 and the slope is 3:

```
# Create y's with intercept = 10 and slope = 3:

y <- 10 + 3 * x
```

Plotting *y* against *x* in a scatterplot (using the optional argument `pch = 19` to change the point characters to filled circles) reveals a straight line with no scatter (Figure 4.7, left plot). In other words, *y* is a perfect function of *x* —something that would never happen in linguistic data.

```
plot(x, y, pch = 19)
```

To add noise, the `rnorm()` function is used a second time to generate residuals.

```
error <- rnorm(50)
y <- 10 + 3 * x + error
```

I want you to notice the similarity between this command and the regression equation ($y = b_0 + b_1 * x + e$) . The error term is what's going to create residuals in the following regression analysis. Next, rerun the plotting command, which yields the plot to the right of Figure 4.7.

```
plot(x., y, pch = 19)
```

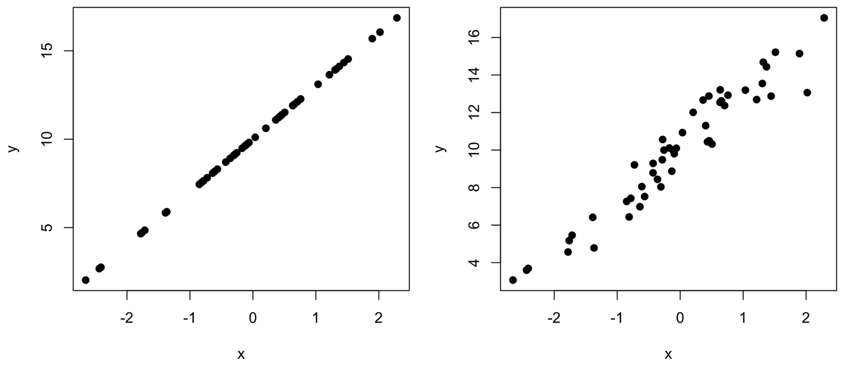

Figure 4.7. Randomly generated data where *y* is a perfect function of *x* (left plot); adding noise to *y* yields the plot on the right

The data is in place. Specifically, you generated random data where *y* depends on *x*. Since you generated the data yourself, you know that the intercept is 10 and the slope is 3. Let's see whether regression is able to retrieve these coefficients from the data.

For this, use the `lm()` function, which stands for linear model. The syntax used in '`y ~ x`' is called 'formula notation', and it can be paraphrased as '`y` as a function of `x`'. Usually, you want to save linear models in R objects so that you can access them for later use.

```
xmdl <- lm(y ~ x)

xmdl
```

```
Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)      x
     10.094  2.808
```

R spits out the coefficients, the main outcome of any regression analysis. Notice how these coefficients are fairly close to what was specified when you generated the data (you will see slightly different values). The deviations from the exact values 10 and 3 are not due to the model being wrong (regression always provides the coefficients that minimize the residuals), but due to the random noise that was added to the data.

The `fitted()` and `residuals()` functions can be used to retrieve the model's fitted values and residuals. The following command uses the `head()` function merely to reduce the output to the first six values.

```
head(fitted(xmdl))
```

```
        1         2         3         4         5
13.943480  8.508189 11.113510 11.870920 11.229031
        6
 9.795856
```

```
head(residuals(xmdl))
```

```
          1           2           3           4
 0.49132013 -0.98612217  1.55160234  0.67056756
          5           6
 0.07353501  0.16232124
```

The first element of the output of `fitted(xmdl)` is the prediction for the first data point, and so on. Similarly, the first element of the output of `residuals(xmdl)` is the deviation of the model's prediction for the first data point, and so on.

A very useful function to apply to linear model objects is `summary()`.

```
summary(xmdl)
```

```
Call:
lm(formula = y ~ x)

Residuals:
  Min    1Q      Median     3Q    Max
-2.6994 -0.6110 0.1832 0.6013 1.5516

Coefficients:
           Estimate Std. Error t value Pr(>|t|)
(Intercept) 10.0939     0.1284   78.61   <2e-16 ***
x            2.8080     0.1126   24.94   <2e-16 ***
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.9075 on 48 degrees of freedom
Multiple R-squared: 0.9284,    Adjusted R-squared: 0.9269
F-statistic:  622 on 1 and 48 DF, p-value: < 2.2e-16
```

This function spits out what's called a coefficient table (under the header '`Coefficients`'. It also includes useful summary statistics for the overall model fit. In this case, the R^2 value is 0.9284, indicating that the model describes about 93% of the variance in *y*. Chapter 6 will discuss '`Adjusted R-squared`'. The other values (*p*-value, standard error, etc.) will be discussed in the chapters on inferential statistics (Chapters 9–11).

The `coef()` function retrieves the coefficients of a linear model. The output of this is a vector of coefficients.

```
coef(xmdl)
```

```
(Intercept)         x
 10.093852  2.807983
```

This vector can be indexed as usual. The following commands retrieve the intercept and slope, respectively.

```
coef(xmdl)[1]
```

```
(Intercept)
   10.09385
```

```
coef(xmdl)[2]
```

```
       x
2.807983
```

```
# Alternative way of indexing (by name):

coef(xmdl)['(Intercept)']
```

```
(Intercept)
   10.09385
```

```
coef(xmdl)['x']
```

```
       x
2.807983
```

The following command computes the fitted value for an *x*-value of 10:

```
coef(xmdl)['(Intercept)'] + coef(xmdl)['x'] * 10
```

```
(Intercept)
   38.17369
```

Don't be confused by the fact that it says '`(Intercept)`' at the top of the predicted value. This is just because when R performs a mathematical operation on two named vectors (one being named '`(Intercept)`', the other one being named '`x`'), it maintains the name of the first vector.

The `predict()` function is useful for generating predictions for fitted models, thus saving you some arithmetic. The function takes two inputs: first, a model which forms the basis for generating predictions; second, a set of new values to generate predictions for.

Let's generate predictions for a sequence of numbers from –3 to +3 using the `seq()` function. This particular sequence is specified to increase in 0.1 intervals (`by = 0.1`).

```
xvals <- seq(from = -3, to = 3, by = 0.1)
```

The `predict()` function needs a data frame or tibble as input. Importantly, the column has to be named '`x`', because this is the name of the predictor in `xmdl`.

```
mypreds <- tibble(x = xvals)
```

Now that you have a tibble to generate predictions for, you can use `predict()`. The following code stores the output of the `mypreds` tibble in a new column, called `fit`.

```
mypreds$fit <- predict(xmdl, newdata = mypreds)

mypreds
```

```
# A tibble: 50 x 2
        x    fit
```

```
     <dbl> <dbl>
  1  -2.66  2.63
  2  -2.56  2.92
  3  -2.46  3.20
  4  -2.36  3.48
  5  -2.26  3.76
  6  -2.16  4.04
  7  -2.06  4.32
  8  -1.96  4.60
  9  -1.86  4.88
 10  -1.76  5.16
 # ... with 40 more rows
```

So, for an *x*-value of –2.66, the model predicts a *y*-value of 2.63, and so on.

4.7. Linear Models with Tidyverse Functions

Let's learn how to do a linear model analysis with tidyverse functions. First, let's put the data into a tibble, then, refit the model.

```
mydf <- tibble(x, y)

xmdl <- lm(y ~ x, data = mydf)
```

The `broom` package (Robinson, 2017) provides tidy model outputs, implemented via the `tidy()` function.

```
library(broom)

# Print tidy coefficient table to console:

tidy(xmdl)
```

```
         term  estimate std.error statistic      p.value
 1 (Intercept) 10.093852 0.1283994  78.61294 2.218721e-52
 2           x  2.807983 0.1125854  24.94091 3.960627e-29
```

The advantage of using `tidy()` is that the output has the structure of a data frame, which means that you can easily index the relevant columns, such as the coefficient estimates.

```
# Extract estimate column from coefficient table:

tidy(xmdl)$estimate
```

```
 [1] 10.093852 2.807983
```

The corresponding glance() function from the broom package gives you a 'glance' of the overall model performance (so far you only know R^2—some of the other quantities will be explained in later chapters).

```
# Check overall model performance:

glance(xmdl)
```

```
r.squared adj.r.squared      sigma  statistic   p.value
1 0.9283634     0.926871  0.9074764 622.0489 3.960627e-29
df  logLik       AIC      BIC deviance df.residual
1 2 -65.07199 136.144 141.88 39.52864      48
```

To plot the model with ggplot2, you can use geom_smooth(). This 'smoother' plots a linear model when specifying the argument method = 'lm'. The geom_point() function and the geom_smooth() function of the following code snippet know what columns to draw the *x*- and *y*-values from because the aesthetic mappings have already been specified in the ggplot() command. These two geoms thus share the same set of aesthetic mappings.

```
mydf %>% ggplot(aes(x = x, y = y)) +
  geom_point() + geom_smooth(method = 'lm') +
  theme_minimal()
```

The resultant plot will look similar to Figure 4.9 (plot to the right) with a superimposed regression line. In addition, a gray ribbon is displayed around the regression line, which is the '95% confidence region', which will be explained later (Chapters 9–11).

4.8. Model Formula Notation: Intercept Placeholders

It's important to learn early on that the following two function calls yield equivalent results.

```
xmdl <- lm(y ~ x, data = mydf)

# Same as:

xmdl <- lm(y ~ 1 + x, data = mydf)
```

The intercept is represented by the placeholder '1'.[6] In R's model formula syntax, the intercept is always fitted by default, even if this is not explicitly specified. In other words,

6 It is no coincidence that the placeholder is the number one. However, since we don't focus on the mathematical details here, it is beyond the scope of this book to explain why. (Just to pique your interest: this has to do with the matrix algebra machinery that is underlying linear models.)

the shorthand notation ‘`y ~ x`’ actually fits the model corresponding to the formula ‘`y ~ 1 + x`’. The second function call is more explicit: it can be paraphrased as ‘estimate not only a slope for `x`, but also an intercept’.

This knowledge can be used to specify an intercept-only model (as was discussed above, see equation E4.8). Let’s do this for the data at hand:

```
# Fitting an intercept-only model:

xmdl_null <- lm(y ~ 1, data = mydf)
```

What does this model predict?

```
coef(xmdl_null)
```

```
(Intercept)
   9.993686
```

The model predicts only one number, the intercept. No matter what value `x` assumes, the model predicts the same number, the mean. Let’s verify that the coefficient of this null model is actually the mean of the response:

```
mean(y)
```

```
[1] 9.993686
```

Thus, we have used the intercept notation to fit a linear model that estimates the mean. For now, this will seem rather dull. (Why would you want to do this?) For the later parts of this book, it’s important to remember the intercept is implemented by a ‘`1`’ placeholder.

4.9. Chapter Conclusions

In this chapter, you performed your first regression analysis, regressing response duration on word frequency. The regression line represents a model of the data, specifically a conditional mean. This line is fully specified in terms of an intercept and a slope. The coefficients are the principal outcome of any regression analysis, and they allow making predictions, which are called fitted values. The extent by which the observed data points deviate from the fitted values are called residuals. The residuals are assumed to be normally distributed and homoscedastic (constant variance). For a given model specification, regression minimizes the size of the residuals. A comparison of a model’s residuals against a null model’s residuals yields a standardized measure of model fit called R^2. Throughout all of this, I emphasized that most of your time should be spend on interpreting the regression coefficients. When doing this, you have to use your field-specific scientific judgment. What does this slope *mean* with respect to your hypothesis?

4.10. Exercises

4.10.1. Exercise 1: Fit the Frequency Model

In this exercise, you will perform the analysis corresponding to Figure 4.1 above. Load in the dataset 'ELP_frequency.csv'. Use `mutate()` to apply the `log10()` function to the frequency column (logarithms will be explained in Chapter 5). Fit a model in which response durations are modeled as a function of log frequencies. Create a plot for the relationship between these two variables.

Additional exercise: can you add a horizontal line showing the mean response duration using `geom_hline()` and the `yintercept` aesthetic?

4.10.2. Exercise 2: Calculating R^2 by Hand

Run the following lines in R (this requires that you still have the random *x*-values and random *y*-values generated in the chapter). Try to make sense of each command. Compare the resulting number to the R^2 value reported by `summary(xmdl)` or `glance(xmdl)`.

```
xmdl <- lm(y ~ x)
xmdl_null <- lm(y ~ 1)

res <- residuals(xmdl)
res_null <- residuals(xmdl_null)
sum(res ^ 2)
sum(res_null ^ 2)
1 - (sum(res ^ 2) / sum(res_null ^ 2))
```