7.3: A few More Mathematical Functions and Operations

In Section 7.2 I discussed the ideas behind variable transformations, and showed that a lot of the transformations that you might want to apply to your data are based on fairly simple mathematical functions and operations, of the kind that we discussed in Chapter 3. In this section I want to return to that discussion, and mention several other mathematical functions and arithmetic operations that I didn't bother to mention when introducing you to R, but are actually quite useful for a lot of real world data analysis. Table 7.1 gives a brief overview of the various mathematical functions I want to talk about (and some that I already have talked about). Obviously this doesn't even come close to cataloging the range of possibilities available in R, but it does cover a very wide range of functions that are used in day to day data analysis.

Table 7.1: Some of the mathematical functions available in R.

mathematical.function	R.function	example.input	answer
square root	sqrt()	sqrt(25)	5
absolute value	abs()	abs(-23)	23
logarithm (base 10)	log10()	log10(1000)	3
logarithm (base e)	log()	log(1000)	6.908
exponentiation	exp()	exp(6.908)	1000.245
rounding to nearest	round()	round(1.32)	1
rounding down	floor()	floor(1.32)	1
rounding up	ceiling()	ceiling(1.32)	2

7.3.1 Rounding a number

One very simple transformation that crops up surprisingly often is the need to round a number to the nearest whole number, or to a certain number of significant digits. To start with, let's assume that we want to round to a whole number. To that end, there are three useful functions in R you want to know about: `round()`, `floor()` and `ceiling()`. The `round()` function just rounds to the *nearest* whole number. So if you round the number 4.3, it "rounds down" to `4`, like so:

```
round( x = 4.3 )
```

```
## [1] 4
```

In contrast, if we want to round the number 4.7, we would round upwards to 5. In everyday life, when someone talks about "rounding", they usually mean "round to nearest", so this is the function we use most of the time. However sometimes you have reasons to want to always round up or always round down. If you want to always round down, use the `floor()` function instead; and if you want to force R to round up, then use `ceiling()`. That's the only difference between the three functions. What if you want to round to a certain number of digits? Let's suppose you want to round to a fixed number of decimal places, say 2 decimal places. If so, what you need to do is specify the `digits` argument to the `round()` function. That's pretty straightforward:

```
round( x = 0.0123, digits = 2 )
```

```
## [1] 0.01
```

The only subtlety that you need to keep in mind is that sometimes what you want to do is round to 2 ***significant digits*** and not to two decimal places. The difference is that, when determining the number of significant digits, zeros don't count. To see this, let's apply the `signif()` function instead of the `round()` function:

```
signif( x = 0.0123, digits = 2 )
```

```
## [1] 0.012
```

This time around, we get an answer of 0.012 because the zeros don't count as significant digits. Quite often scientific journals will ask you to report numbers to two or three significant digits, so it's useful to remember the distinction.

7.3.2 Modulus and integer division

Table 7.2: Two more arithmetic operations that sometimes come in handy

operation	operator	example.input	answer
integer division	%/%	42 %/% 10	4
modulus	%%	42 %% 10	2

Since we're on the topic of simple calculations, there are two other arithmetic operations that I should mention, since they can come in handy when working with real data. These operations are calculating a modulus and doing integer division. They don't come up anywhere else in this book, but they are worth knowing about. First, let's consider ***integer division***. Suppose I have $42 in my wallet, and want to buy some sandwiches, which are selling for $10 each. How many sandwiches can I afford[107] to buy? The answer is of course 4. Note that it's not 4.2, since no shop will sell me one-fifth of a sandwich. That's integer division. In R we perform integer division by using the `%/%` operator:

```
42 %/% 10
```

```
## [1] 4
```

Okay, that's easy enough. What about the ***modulus***? Basically, a modulus is the remainder after integer division, and it's calculated using the `%%` operator. For the sake of argument, let's suppose I buy four overpriced $10 sandwiches. If I started out with $42, how much money do I have left? The answer, as both R and common sense tells us, is $2:

```
42 %% 10
```

```
## [1] 2
```

So that's also pretty easy. There is, however, one subtlety that I need to mention, and this relates to how negative numbers are handled. Firstly, what would happen if I tried to do integer division with a negative number? Let's have a look:

```
-42 %/% 10
```

```
## [1] -5
```

This might strike you as counterintuitive: why does `42 %/% 10` produce an answer of `4`, but `-42 %/% 10` gives us an answer of `-5`? Intuitively you might think that the answer to the second one should be `-4`. The way to think about it is like this. Suppose I *owe* the sandwich shop $42, but I don't have any money. How many sandwiches would *I* have to give *them* in order to stop them from calling security? The answer[108] here is 5, not 4. If I handed them 4 sandwiches, I'd still owe them $2, right? So I actually have to give them 5 sandwiches. And since it's *me* giving them the sandwiches, the answer to `-42 %/% 10` is `-5`. As you might expect, the behaviour of the modulus operator has a similar pattern. If I've handed 5 sandwiches over to the shop in order to pay off my debt of $42, then *they* now owe me $8. So the modulus is now:

```
-42 %% 10
```

```
## [1] 8
```

7.3.3 Logarithms and exponentials

As I've mentioned earlier, R has an incredible range of mathematical functions built into it, and there really wouldn't be much point in trying to describe or even list all of them. For the most part, I've focused only on those functions that are strictly necessary for this book. However I do want to make an exception for logarithms and exponentials. Although they aren't needed anywhere else in this book, they are *everywhere* in statistics more broadly, and not only that, there are a *lot* of situations in which it is convenient to analyse the logarithm of a variable (i.e., to take a "log-transform" of the variable). I suspect that many (maybe most) readers of this book will have encountered logarithms and exponentials before, but from past experience I know that there's a substantial proportion of students who take a social science statistics class who haven't touched logarithms since high school, and would appreciate a bit of a refresher.

In order to understand logarithms and exponentials, the easiest thing to do is to actually calculate them and see how they relate to other simple calculations. There are three R functions in particular that I want to talk about, namely `log()`, `log10()` and `exp()`. To start with, let's consider `log10()`, which is known as the "logarithm in base 10". The trick to understanding a ***logarithm*** is to understand that it's basically the "opposite" of taking a power. Specifically, the logarithm in base 10 is closely related to the powers of 10. So let's start by noting that 10-cubed is 1000. Mathematically, we would write this:

$$10^3=1000$$

and in R we'd calculate it by using the command `10^3`. The trick to understanding a logarithm is to recognise that the statement that "10 to the power of 3 is equal to 1000" is equivalent to the statement that "the logarithm (in base 10) of 1000 is equal to 3". Mathematically, we write this as follows,

$$\log_{10}(1000)=3$$

and if we wanted to do the calculation in R we would type this:

```
log10( 1000 )
```

```
## [1] 3
```

Obviously, since you already know that $10^3=1000$ there's really no point in getting R to tell you that the base-10 logarithm of 1000 is 3. However, most of the time you probably don't know what right answer is. For instance, I can honestly say that I didn't know that $10^{2.69897}=500$, so it's rather convenient for me that I can use R to calculate the base-10 logarithm of 500:

```
log10( 500 )
```

```
## [1] 2.69897
```

Or at least it would be convenient if I had a pressing need to know the base-10 logarithm of 500.

Okay, since the `log10()` function is related to the powers of 10, you might expect that there are other logarithms (in bases other than 10) that are related to other powers too. And of course that's true: there's not really anything mathematically special about the number 10. You and I happen to find it useful because decimal numbers are built around the number 10, but the big bad world of mathematics scoffs at our decimal numbers. Sadly, the universe doesn't actually care how we write down numbers. Anyway, the consequence of this cosmic indifference is that there's nothing particularly special about calculating logarithms in base 10. You could, for instance, calculate your logarithms in base 2, and in fact R does provide a function for doing that, which is (not surprisingly) called `log2()`. Since we know that $23=2\times2\times2=8$, it's not surprise to see that

```
log2( 8 )
```

```
## [1] 3
```

Alternatively, a third type of logarithm – and one we see a lot more of in statistics than either base 10 or base 2 – is called the ***natural logarithm***, and corresponds to the logarithm in base e. Since you might one day run into it, I'd better explain what e is. The number e, known as ***Euler's number***, is one of those annoying "irrational" numbers whose decimal expansion is infinitely long, and is considered one of the most important numbers in mathematics. The first few digits of e are:

$$e=2.718282$$

There are quite a few situation in statistics that require us to calculate powers of e, though none of them appear in this book. Raising e to the power x is called the ***exponential*** of x, and so it's very common to see e^x written as exp(x). And so it's no surprise that R has a function that calculate exponentials, called `exp()`. For instance, suppose I wanted to calculate e^3. I could try typing in the value of e manually, like this:

```
2.718282 ^ 3
```

```
## [1] 20.08554
```

but it's much easier to do the same thing using the `exp()` function:

```
exp( 3 )
```

```
## [1] 20.08554
```

Anyway, because the number e crops up so often in statistics, the natural logarithm (i.e., logarithm in base e) also tends to turn up. Mathematicians often write it as loge(x) or ln(x), or sometimes even just log(x). In fact, R works the same way: the `log()` function corresponds to the natural logarithm[109] Anyway, as a quick check, let's calculate the natural logarithm of 20.08554 using R:

```
log( 20.08554 )
```

```
## [1] 3
```

And with that, I think we've had quite enough exponentials and logarithms for this book!