5.7: Correlations

Up to this point we have focused entirely on how to construct descriptive statistics for a single variable. What we haven't done is talked about how to describe the relationships *between* variables in the data. To do that, we want to talk mostly about the ***correlation*** between variables. But first, we need some data.

5.7.1 data

After spending so much time looking at the AFL data, I'm starting to get bored with sports. Instead, let's turn to a topic close to every parent's heart: sleep. The following data set is fictitious, but based on real events. Suppose I'm curious to find out how much my infant son's sleeping habits affect my mood. Let's say that I can rate my grumpiness very precisely, on a scale from 0 (not at all grumpy) to 100 (grumpy as a very, very grumpy old man). And, lets also assume that I've been measuring my grumpiness, my sleeping patterns and my son's sleeping patterns for quite some time now. Let's say, for 100 days. And, being a nerd, I've saved the data as a file called `parenthood.Rdata`. If we load the data…

```
load( "./data/parenthood.Rdata" )
who(TRUE)
```

```
##    -- Name --      -- Class --   -- Size --
##    parenthood     data.frame    100 x 4
##     $dan.sleep    numeric       100
##     $baby.sleep   numeric       100
##     $dan.grump    numeric       100
##     $day          integer       100
```

… we see that the file contains a single data frame called `parenthood`, which contains four variables `dan.sleep`, `baby.sleep`, `dan.grump` and `day`. If we peek at the data using `head()` out the data, here's what we get:

```
head(parenthood,10)
```

```
##    dan.sleep baby.sleep dan.grump day
## 1       7.59      10.18        56   1
## 2       7.91      11.66        60   2
## 3       5.14       7.92        82   3
## 4       7.71       9.61        55   4
## 5       6.68       9.75        67   5
## 6       5.99       5.04        72   6
## 7       8.19      10.45        53   7
## 8       7.19       8.27        60   8
## 9       7.40       6.06        60   9
## 10      6.58       7.09        71  10
```

Next, I'll calculate some basic descriptive statistics:

```
describe( parenthood )
```

```
##             vars   n  mean    sd median trimmed   mad   min    max range
## dan.sleep      1 100  6.97  1.02   7.03    7.00  1.09  4.84   9.00  4.16
## baby.sleep     2 100  8.05  2.07   7.95    8.05  2.33  3.25  12.07  8.82
## dan.grump      3 100 63.71 10.05  62.00   63.16  9.64 41.00  91.00 50.00
## day            4 100 50.50 29.01  50.50   50.50 37.06  1.00 100.00 99.00
##              skew kurtosis   se
## dan.sleep  -0.29    -0.72 0.10
## baby.sleep -0.02    -0.69 0.21
## dan.grump   0.43    -0.16 1.00
## day         0.00    -1.24 2.90
```

Finally, to give a graphical depiction of what each of the three interesting variables looks like, Figure 5.6 plots histograms.

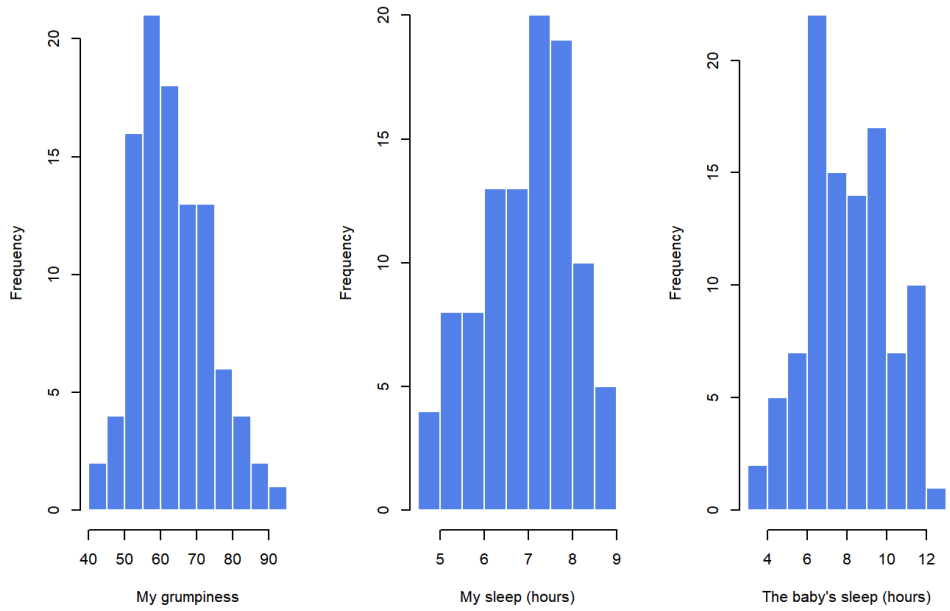

Figure 5.6: Histograms for the three interesting variables in the `parenthood` data set

One thing to note: just because R can calculate dozens of different statistics doesn't mean you should report all of them. If I were writing this up for a report, I'd probably pick out those statistics that are of most interest to me (and to my readership), and then put them into a nice, simple table like the one in Table **??**.[79] Notice that when I put it into a table, I gave everything "human readable" names. This is always good practice. Notice also that I'm not getting enough sleep. This isn't good practice, but other parents tell me that it's standard practice.

Table 5.2: Descriptive statistics for the parenthood data.

variable	min	max	mean	median	std. dev	IQR
Dan's grumpiness	41	91	63.71	62	10.05	14
Dan's hours slept	4.84	9	6.97	7.03	1.02	1.45
Dan's son's hours slept	3.25	12.07	8.05	7.95	2.07	3.21

5.7.2 strength and direction of a relationship

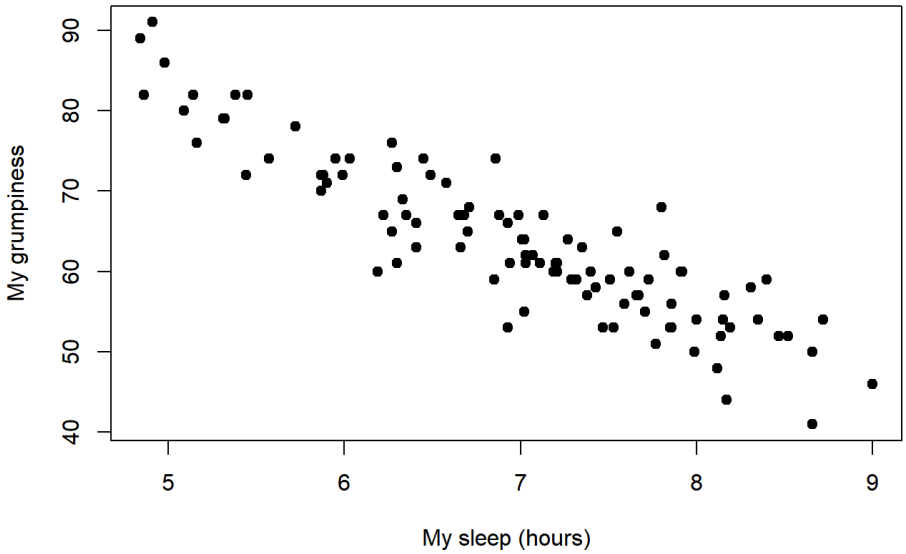

Figure 5.7: Scatterplot showing the relationship between `dan.sleep` and `dan.grump`

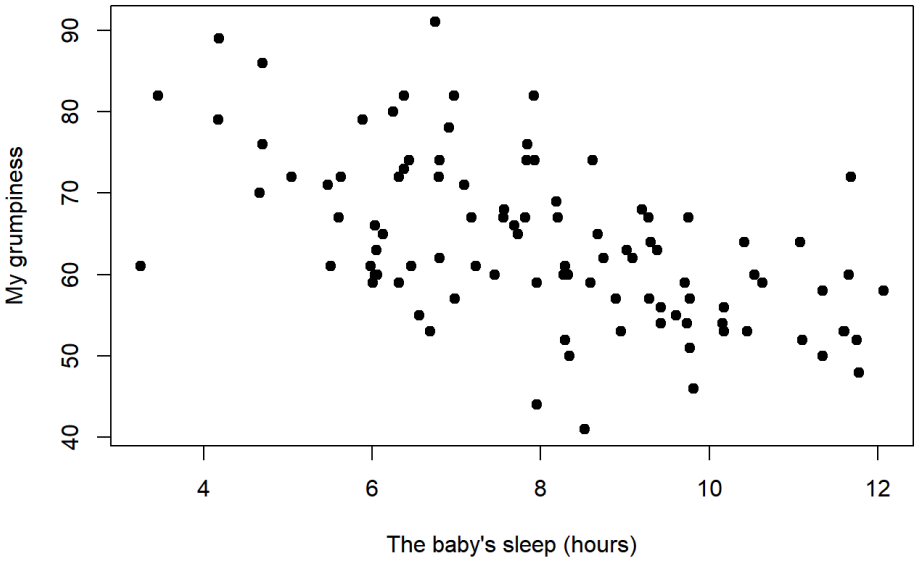

Figure 5.8: Scatterplot showing the relationship between `baby.sleep` and `dan.grump`

We can draw scatterplots to give us a general sense of how closely related two variables are. Ideally though, we might want to say a bit more about it than that. For instance, let's compare the relationship between `dan.sleep` and `dan.grump` (Figure 5.7 with that between `baby.sleep` and `dan.grump` (Figure 5.8. When looking at these two plots side by side, it's clear that the relationship is *qualitatively* the same in both cases: more sleep equals less grump! However, it's also pretty obvious that the relationship between `dan.sleep` and `dan.grump` is *stronger* than the relationship between `baby.sleep` and `dan.grump` . The plot on the left is "neater" than the one on the right. What it feels like is that if you want to predict what my mood is, it'd help you a little bit to know how many hours my son slept, but it'd be *more* helpful to know how many hours I slept.

In contrast, let's consider Figure 5.8 vs. Figure 5.9. If we compare the scatterplot of " `baby.sleep` v `dan.grump` " to the scatterplot of "` `baby.sleep` v `dan.sleep` ", the overall strength of the relationship is the same, but the direction is different. That is, if my son sleeps more, I get *more* sleep (positive relationship, but if he sleeps more then I get *less* grumpy (negative relationship).

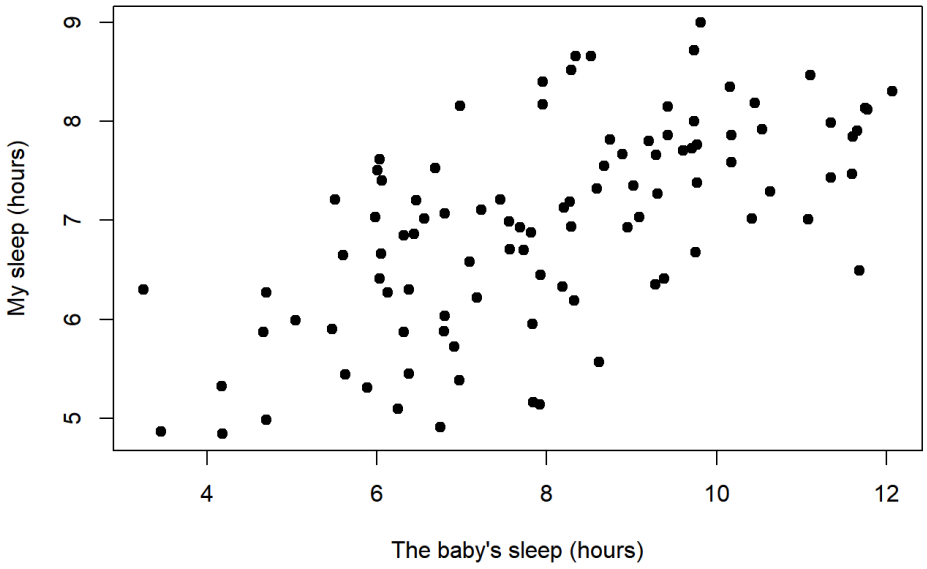

Figure 5.9: Scatterplot showing the relationship between `baby.sleep` and `dan.sleep`

5.7.3 correlation coefficient

We can make these ideas a bit more explicit by introducing the idea of a ***correlation coefficient*** (or, more specifically, Pearson's correlation coefficient), which is traditionally denoted by r. The correlation coefficient between two variables X and Y (sometimes denoted rXY), which we'll define more precisely in the next section, is a measure that varies from −1 to 1. When r=−1 it means that we have a perfect negative relationship, and when r=1 it means we have a perfect positive relationship. When r=0, there's no relationship at all. If you look at Figure 5.10, you can see several plots showing what different correlations look like.

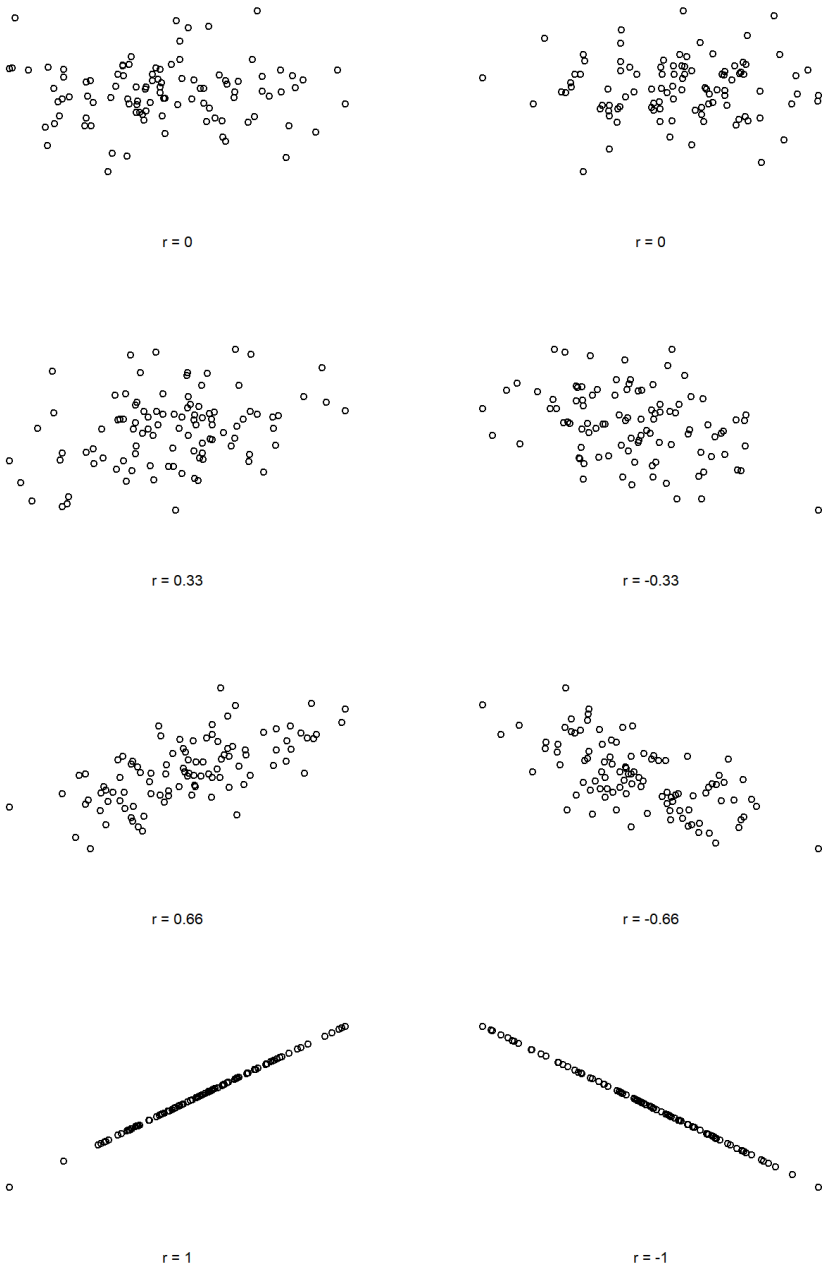

Figure 5.10: Illustration of the effect of varying the strength and direction of a correlation

The formula for the Pearson's correlation coefficient can be written in several different ways. I think the simplest way to write down the formula is to break it into two steps. Firstly, let's introduce the idea of a ***covariance***. The covariance between two variables X and Y is a generalisation of the notion of the variance; it's a mathematically simple way of describing the relationship between two variables that isn't terribly informative to humans:

$$\mathrm{Cov}(X,Y) = \frac{1}{N-1} \sum_{i=1}^{N} \left(X_i - \bar{X}\right) \left(Y_i - \bar{Y}\right)$$

Because we're multiplying (i.e., taking the "product" of) a quantity that depends on X by a quantity that depends on Y and then averaging[80], you can think of the formula for the covariance as an "average cross product" between X and Y. The covariance has the nice property that, if X and Y are entirely unrelated, then the covariance is exactly zero. If the relationship between them is positive (in the sense shown in Figure@reffig:corr) then the covariance is also positive; and if the relationship is negative then the covariance is also negative. In other words, the covariance captures the basic qualitative idea of correlation. Unfortunately, the raw magnitude of the covariance isn't easy to interpret: it depends on the units in which X and Y are expressed, and worse yet, the

actual units that the covariance itself is expressed in are really weird. For instance, if X refers to the `dan.sleep` variable (units: hours) and Y refers to the `dan.grump` variable (units: grumps), then the units for their covariance are "hours × grumps". And I have no freaking idea what that would even mean.

The Pearson correlation coefficient r fixes this interpretation problem by standardising the covariance, in pretty much the exact same way that the z-score standardises a raw score: by dividing by the standard deviation. However, because we have two variables that contribute to the covariance, the standardisation only works if we divide by both standard deviations.[81] In other words, the correlation between X and Y can be written as follows:

$$r_{XY} = \frac{\mathrm{Cov}(X,Y)}{\hat{\sigma}_X \hat{\sigma}_Y}$$

By doing this standardisation, not only do we keep all of the nice properties of the covariance discussed earlier, but the actual values of r are on a meaningful scale: r=1 implies a perfect positive relationship, and r=−1 implies a perfect negative relationship. I'll expand a little more on this point later, in Section@refsec:interpretingcorrelations. But before I do, let's look at how to calculate correlations in R.

5.7.4 Calculating correlations in R

Calculating correlations in R can be done using the `cor()` command. The simplest way to use the command is to specify two input arguments `x` and `y`, each one corresponding to one of the variables. The following extract illustrates the basic usage of the function:[82]

```
cor( x = parenthood$dan.sleep, y = parenthood$dan.grump )
```

```
## [1] -0.903384
```

However, the `cor()` function is a bit more powerful than this simple example suggests. For example, you can also calculate a complete "correlation matrix", between all pairs of variables in the data frame:[83]

```
# correlate all pairs of variables in "parenthood":
cor( x = parenthood )
```

```
##              dan.sleep  baby.sleep   dan.grump         day
## dan.sleep   1.00000000  0.62794934 -0.90338404 -0.09840768
## baby.sleep  0.62794934  1.00000000 -0.56596373 -0.01043394
## dan.grump  -0.90338404 -0.56596373  1.00000000  0.07647926
## day        -0.09840768 -0.01043394  0.07647926  1.00000000
```

5.7.5 Interpreting a correlation

Naturally, in real life you don't see many correlations of 1. So how should you interpret a correlation of, say r=.4? The honest answer is that it really depends on what you want to use the data for, and on how strong the correlations in your field tend to be. A friend of mine in engineering once argued that any correlation less than .95 is completely useless (I think he was exaggerating, even for engineering). On the other hand there are real cases – even in psychology – where you should really expect correlations that strong. For instance, one of the benchmark data sets used to test theories of how people judge similarities is so clean that any theory that can't achieve a correlation of at least .9 really isn't deemed to be successful. However, when looking for (say) elementary correlates of intelligence (e.g., inspection time, response time), if you get a correlation above .3 you're doing very very well. In short, the interpretation of a correlation depends a lot on the context. That said, the rough guide in Table **??** is pretty typical.

```
knitr::kable(
rbind(
c("-1.0 to -0.9" ,"Very strong", "Negative"),
c("-0.9 to -0.7", "Strong", "Negative") ,
c("-0.7 to -0.4", "Moderate", "Negative") ,
c("-0.4 to -0.2", "Weak", "Negative"),
c("-0.2 to 0","Negligible", "Negative") ,
c("0 to 0.2","Negligible", "Positive"),
c("0.2 to 0.4", "Weak", "Positive"),
c("0.4 to 0.7", "Moderate", "Positive"),
c("0.7 to 0.9", "Strong", "Positive"),
c("0.9 to 1.0", "Very strong", "Positive")), col.names=c("Correlation", "Strength", '
  booktabs = TRUE)
```

Correlation	Strength	Direction
-1.0 to -0.9	Very strong	Negative
-0.9 to -0.7	Strong	Negative
-0.7 to -0.4	Moderate	Negative
-0.4 to -0.2	Weak	Negative
-0.2 to 0	Negligible	Negative
0 to 0.2	Negligible	Positive
0.2 to 0.4	Weak	Positive
0.4 to 0.7	Moderate	Positive
0.7 to 0.9	Strong	Positive
0.9 to 1.0	Very strong	Positive

However, something that can never be stressed enough is that you should *always* look at the scatterplot before attaching any interpretation to the data. A correlation might not mean what you think it means. The classic illustration of this is "Anscombe's Quartet" (**???** Anscombe1973), which is a collection of four data sets. Each data set has two variables, an X and a Y. For all four data sets the mean value for X is 9 and the mean for Y is 7.5. The, standard deviations for all X variables are almost identical, as are those for the the Y variables. And in each case the correlation between X and Y is r=0.816. You can verify this yourself, since the dataset comes distributed with R. The commands would be:

```
cor( anscombe$x1, anscombe$y1 )
```

```
## [1] 0.8164205
```

```
cor( anscombe$x2, anscombe$y2 )
```

```
## [1] 0.8162365
```

and so on.

You'd think that these four data setswould look pretty similar to one another. They do not. If we draw scatterplots of X against Y for all four variables, as shown in Figure 5.11 we see that all four of these are *spectacularly* different to each other.

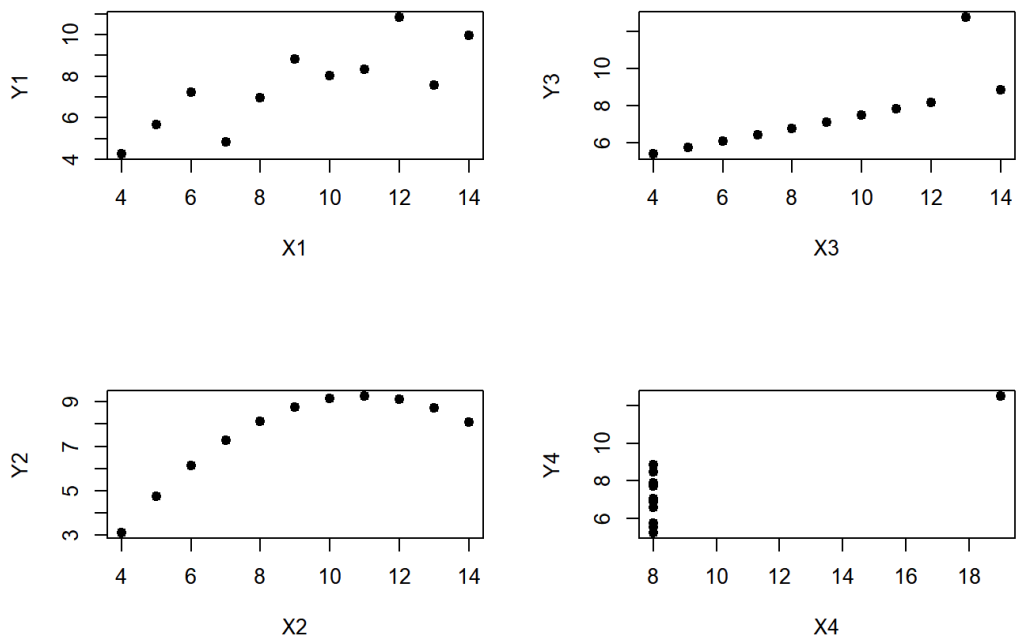

Figure 5.11: Anscombe's quartet. All four of these data sets have a Pearson correlation of r=.816, but they are qualitatively different from one another.

The lesson here, which so very many people seem to forget in real life is "*always graph your raw data*". This will be the focus of Chapter 6.

5.7.6 Spearman's rank correlations

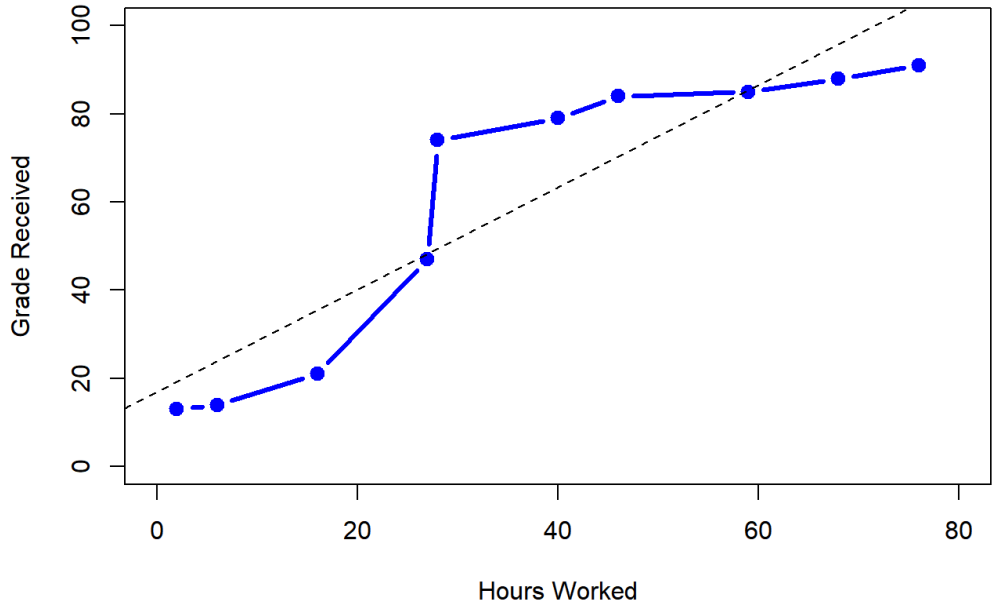

Figure 5.12: The relationship between hours worked and grade received, for a toy data set consisting of only 10 students (each circle corresponds to one student). The dashed line through the middle shows the linear relationship between the two variables. This produces a strong Pearson correlation of r=.91. However, the interesting thing to note here is that there's actually a perfect monotonic relationship between the two variables: in this toy example at least, increasing the hours worked always increases the grade received, as illustrated by the solid line. This is reflected in a Spearman correlation of rho=1. With such a small data set, however, it's an open question as to which version better describes the actual relationship involved.

The Pearson correlation coefficient is useful for a lot of things, but it does have shortcomings. One issue in particular stands out: what it actually measures is the strength of the *linear* relationship between two variables. In other words, what it gives you is a measure of the extent to which the data all tend to fall on a single, perfectly straight line. Often, this is a pretty good approximation to what we mean when we say "relationship", and so the Pearson correlation is a good thing to calculation. Sometimes, it isn't.

One very common situation where the Pearson correlation isn't quite the right thing to use arises when an increase in one variable X really is reflected in an increase in another variable Y, but the nature of the relationship isn't necessarily linear. An example of this might be the relationship between effort and reward when studying for an exam. If you put in zero effort (X) into learning a subject, then you should expect a grade of 0% (Y). However, a little bit of effort will cause a *massive* improvement: just turning up to lectures means that you learn a fair bit, and if you just turn up to classes, and scribble a few things down so your grade might rise to 35%, all without a lot of effort. However, you just don't get the same effect at the other end of the scale. As everyone knows, it takes *a lot* more effort to get a grade of 90% than it takes to get a grade of 55%. What this means is that, if I've got data looking at study effort and grades, there's a pretty good chance that Pearson correlations will be misleading.

To illustrate, consider the data plotted in Figure 5.12, showing the relationship between hours worked and grade received for 10 students taking some class. The curious thing about this – highly fictitious – data set is that increasing your effort *always* increases your grade. It might be by a lot or it might be by a little, but increasing effort will never decrease your grade. The data are stored in `effort.Rdata` :

```
> load( "effort.Rdata" )
> who(TRUE)
   -- Name --    -- Class --    -- Size --
   effort        data.frame     10 x 2
    $hours       numeric        10
    $grade       numeric        10
```

The raw data look like this:

```
> effort
   hours grade
1      2    13
2     76    91
3     40    79
4      6    14
5     16    21
6     28    74
7     27    47
8     59    85
9     46    84
10    68    88
```

If we run a standard Pearson correlation, it shows a strong relationship between hours worked and grade received,

```
> cor( effort$hours, effort$grade )
[1] 0.909402
```

but this doesn't actually capture the observation that increasing hours worked *always* increases the grade. There's a sense here in which we want to be able to say that the correlation is *perfect* but for a somewhat different notion of what a "relationship" is. What we're looking for is something that captures the fact that there is a perfect ***ordinal relationship*** here. That is, if student 1 works more hours than student 2, then we can guarantee that student 1 will get the better grade. That's not what a correlation of r=.91 says at all.

How should we address this? Actually, it's really easy: if we're looking for ordinal relationships, all we have to do is treat the data as if it were ordinal scale! So, instead of measuring effort in terms of "hours worked", lets rank all 10 of our students in order of hours worked. That is, student 1 did the least work out of anyone (2 hours) so they get the lowest rank (rank = 1). Student 4 was the next laziest, putting in only 6 hours of work in over the whole semester, so they get the next lowest rank (rank = 2). Notice that I'm using "rank =1" to mean "low rank". Sometimes in everyday language we talk about "rank = 1" to mean "top rank" rather than "bottom rank". So be careful: you can rank "from smallest value to largest value" (i.e., small equals rank 1) or you can rank "from

largest value to smallest value" (i.e., large equals rank 1). In this case, I'm ranking from smallest to largest, because that's the default way that R does it. But in real life, it's really easy to forget which way you set things up, so you have to put a bit of effort into remembering!

Okay, so let's have a look at our students when we rank them from worst to best in terms of effort and reward:

	rank (hours worked)	rank (grade received)
student	1	1
student	2	10
student	3	6
student	4	2
student	5	3
student	6	5
student	7	4
student	8	8
student	9	7
student	10	9

Hm. These are *identical*. The student who put in the most effort got the best grade, the student with the least effort got the worst grade, etc. We can get R to construct these rankings using the `rank()` function, like this:

```
> hours.rank <- rank( effort$hours )   # rank students by hours worked
> grade.rank <- rank( effort$grade )   # rank students by grade received
```

As the table above shows, these two rankings are identical, so if we now correlate them we get a perfect relationship:

```
> cor( hours.rank, grade.rank )
[1] 1
```

What we've just re-invented is ***Spearman's rank order correlation***, usually denoted ρ to distinguish it from the Pearson correlation r. We can calculate Spearman's ρ using R in two different ways. Firstly we could do it the way I just showed, using the `rank()` function to construct the rankings, and then calculate the Pearson correlation on these ranks. However, that's way too much effort to do every time. It's much easier to just specify the `method` argument of the `cor()` function.

```
> cor( effort$hours, effort$grade, method = "spearman")
[1] 1
```

The default value of the `method` argument is `"pearson"`, which is why we didn't have to specify it earlier on when we were doing Pearson correlations.

5.7.7 `correlate()` function

As we've seen, the `cor()` function works pretty well, and handles many of the situations that you might be interested in. One thing that many beginners find frustrating, however, is the fact that it's not built to handle non-numeric variables. From a statistical perspective, this is perfectly sensible: Pearson and Spearman correlations are only designed to work for numeric variables, so the `cor()` function spits out an error.

Here's what I mean. Suppose you were keeping track of how many `hours` you worked in any given day, and counted how many `tasks` you completed. If you were doing the tasks for money, you might also want to keep track of how much `pay` you got

for each job. It would also be sensible to keep track of the `weekday` on which you actually did the work: most of us don't work as much on Saturdays or Sundays. If you did this for 7 weeks, you might end up with a data set that looks like this one:

```
> load("work.Rdata")

> who(TRUE)
   -- Name --    -- Class --    -- Size --
   work          data.frame     49 x 7
    $hours       numeric        49
    $tasks       numeric        49
    $pay         numeric        49
    $day         integer        49
    $weekday     factor         49
    $week        numeric        49
    $day.type    factor         49

> head(work)
  hours tasks pay day   weekday week day.type
1   7.2    14  41   1   Tuesday    1  weekday
2   7.4    11  39   2 Wednesday    1  weekday
3   6.6    14  13   3  Thursday    1  weekday
4   6.5    22  47   4    Friday    1  weekday
5   3.1     5   4   5  Saturday    1  weekend
6   3.0     7  12   6    Sunday    1  weekend
```

Obviously, I'd like to know something about how all these variables correlate with one another. I could correlate `hours` with `pay` quite using `cor()`, like so:

```
> cor(work$hours,work$pay)
[1] 0.7604283
```

But what if I wanted a quick and easy way to calculate all pairwise correlations between the numeric variables? I can't just input the `work` data frame, because it contains two factor variables, `weekday` and `day.type`. If I try this, I get an error:

```
> cor(work)
Error in cor(work) : 'x' must be numeric
```

It order to get the correlations that I want using the `cor()` function, is create a new data frame that doesn't contain the factor variables, and then feed that new data frame into the `cor()` function. It's not actually very hard to do that, and I'll talk about how to do it properly in Section@refsec:subsetdataframe. But it would be nice to have some function that is smart enough to just ignore the factor variables. That's where the `correlate()` function in the `lsr` package can be handy. If you feed it a data frame that contains factors, it knows to ignore them, and returns the pairwise correlations only between the numeric variables:

```
> correlate(work)

CORRELATIONS
============
- correlation type:  pearson
- correlations shown only when both variables are numeric

          hours  tasks   pay    day weekday   week day.type
hours         .  0.800 0.760 -0.049       .  0.018        .
tasks     0.800      . 0.720 -0.072       . -0.013        .
pay       0.760  0.720     .  0.137       .  0.196        .
day      -0.049 -0.072 0.137      .       .  0.990        .
weekday       .      .     .      .       .      .        .
week      0.018 -0.013 0.196  0.990       .      .        .
day.type      .      .     .      .       .      .        .
```

The output here shows a `.` whenever one of the variables is non-numeric. It also shows a `.` whenever a variable is correlated with itself (it's not a meaningful thing to do). The `correlate()` function can also do Spearman correlations, by specifying the `corr.method` to use:

```
> correlate( work, corr.method="spearman" )

CORRELATIONS
============
- correlation type:  spearman
- correlations shown only when both variables are numeric

          hours  tasks   pay    day weekday   week day.type
hours         .  0.805 0.745 -0.047       .  0.010        .
tasks     0.805      . 0.730 -0.068       . -0.008        .
pay       0.745  0.730     .  0.094       .  0.154        .
day      -0.047 -0.068 0.094      .       .  0.990        .
weekday       .      .     .      .       .      .        .
week      0.010 -0.008 0.154  0.990       .      .        .
day.type      .      .     .      .       .      .        .
```

Obviously, there's no new functionality in the `correlate()` function, and any advanced R user would be perfectly capable of using the `cor()` function to get these numbers out. But if you're not yet comfortable with extracting a subset of a data frame, the `correlate()` function is for you.